EDÍCIA SKRÍPT

PRISPÔSOBOVANIE UŽÍVATEĽSKÉHO ROZHRANIA A AUTOMATIZÁCIA PROCESOV V SOFTVÉRI AutoCAD

Martin Ambroz

STU SvF
SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
STAVEBNÁ FAKULTA

SPEKTRUM STU

PRISPÔSOBOVANIE UŽÍVATEĽSKÉHO ROZHRANIA A AUTOMATIZÁCIA PROCESOV V SOFTVÉRI AutoCAD

Martin Ambroz

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
2022

Recenzenti: Mgr. art. Mgr. Ladislav Šipeky, PhD.
Ing. Mária Bossert Tješšová, PhD.

Schválila Edičná rada Stavebnej fakulty STU v Bratislave.

ISBN 978-80-227-5258-9

Predslov

Skriptá oboznámia čitateľa s niektorými pokročilejšími možnosťami práce v prostredí softvéru AutoCAD 2023, pričom sú zamerané najmä na prispôsobenie užívateľského prostredia a automatizíciu procesov viacerými spôsobmi. Sú určené predovšetkým študentom predmetov Softvér (AutoCAD) a Softvér (AutoCAD) – pre pokročilých vyučovaných na Stavebnej fakulte STU v Bratislave, ale rovnako tak sú vhodné ako študijný materiál na iných univerzitách technického zameriania na Slovensku i v susednej Českej republike, a v neposlednom rade pre všetkých skúsenejších používateľov softvéru AutoCAD.

Obsah skrípt voľne nadväzuje na učebnicu [12], ktorá sa venuje najmä základom práce v softvéri AutoCAD. Je inšpirovaný pomocníkom softvéru AutoCAD [5], v ktorom užívatelia nájdu veľmi dobre zdokumentované príkazy, často aj s uvedením príkladov.

Skriptá sa členia na desať kapitol. V úvodnej kapitole oboznamujeme čitateľa s motiváciou prispôsobovania a automatizácie, rovnako tak aj so značením, s ktorým sa v nich môže stretnúť. Druhá kapitola popisuje možnosti prispôsobovania užívateľského prostredia. V tretej kapitole sa začíname venovať automatizácii procesov pomocou troch základných možností. Ďalšie kapitoly (štvrtá až desiata) sú venované programovaniu v jazyku AutoLISP. Najskôr v štvrtej kapitole predstavíme základné princípy programovania v jazyku AutoLISP, rovnako aj rôzne editory pre programovanie v tomto jazyku. Následne popíšeme základné dátové typy jazyka AutoLISP a funkcie na ich obsluhu (piata kapitola). V šiestej kapitole uvedieme spôsoby tvorby vlastných funkcií a ovládania príkazového riadku AutoCADu pomocou funkcií jazyka AutoLISP. V siedmeje a ôsmej kapitole sa venujeme pomienkam a cyklom, ktoré sú nevyhnutné pre vytvorenie väčšiny programov. Deviata kapitola popisuje možnosti vstupu a výstupu zo súborov a záverečná desiata kapitola sa venuje pohľadu do vnútornej organizácie entít AutoCAD a výberovým množinám.

Radi by sme vyjadrili vďačnosť recenzentom Mgr. art., Mgr. Ladislavovi Šipekymu, PhD., a Ing. Márii Bossaert Tješšovej, PhD., za ich podnetné pripomienky, ktoré prispeli k skvalitneniu skrípt.

Skriptá boli podporené grantovým projektom KEGA 008STU-4/2020.

Obsah

1 Úvod **7**

2 Prispôsobovanie užívateľského rozhrania **9**

2.1 Priame úpravy užívateľského rozhrania 10

2.1.1 Panel rýchleho prístupu 10

2.1.2 Pás kariet 10

2.1.3 Palety 11

2.2 Úpravy pomocou príkazu **CUI** 12

2.2.1 Úprava prvkov UI 14

2.2.2 Prenos prvkov medzi súbormi `*.cuix` 29

2.3 Palety nástrojov 30

2.3.1 Tvorba a úprava paliet na palete nástrojov 31

2.3.2 Spravovanie a prenos paliet a skupín paliet na palete nástrojov 35

2.4 Aliasy príkazov 36

2.5 Profily 38

3 Základné možnosti automatizácie **39**

3.1 Action Recorder 39

3.2 Command macro 41

3.3 Skript 45

3.3.1 Tvorba skriptov 45

3.3.2 Spúšťanie skriptov 47

3.3.3 Ďalšie príkazy AutoCADu pre prácu so skriptami 48

3.3.4 Skripty s výrazmi jazyka AutoLISP 50

3.3.5 AutoCAD Core Console 50

4 Úvod do jazyka AutoLISP a vývojových prostredí **53**

4.1 Príkazový riadok AutoCADu 57

4.2 Notepad a Notepad++ 58

4.3 Visual LISP Editor 58

4.4 Visual Studio Code 60

5 Základné dátové typy a práca s nimi **62**

5.1 Symboly a premenné 63

5.2 Celé a reálne čísla 66

5.3 Reťazce znakov 68

5.4 Zoznamy 70

5.4.1 Bodka-dvojica . . . 74
5.5 Konverzia dátových typov . . . 75
5.5.1 Celé čísla . . . 75
5.5.2 Reálne čísla . . . 76
5.6 Vstup od užívatela . . . 78
5.6.1 Číselný vstup . . . 79
5.6.2 Body . . . 80
5.6.3 Reťazce znakov . . . 80
5.6.4 Kľúčové slová . . . 80

6 Funkcie a príkazy v AutoLISPe 82
6.1 Definovanie funkcií . . . 82
6.2 Definovanie príkazov . . . 85
6.3 Ovládanie príkazového riadku AutoCADu cez AutoLISP . . . 86
6.3.1 Porovnanie funkcií `command` a `command-s` . . . 86
6.3.2 Výstup do príkazového riadku . . . 88
6.3.3 Získanie vstupu od užívateľa . . . 89
6.3.4 Transparentné príkazy . . . 93
6.3.5 Úpravy štandardných príkazov AutoCADu . . . 94

7 Podmienky a logické operácie 96
7.1 Funkcia `if` . . . 97
7.2 Funkcia `cond` . . . 100
7.3 Relačné operátory . . . 102
7.3.1 Porovnávanie textových reťazcov . . . 103
7.3.2 Overenie typu premennej . . . 105
7.4 Logické operátory . . . 106

8 Cykly a pokročilejšia práca so zoznamami 108
8.1 Funkcia `repeat` . . . 108
8.2 Funkcia `while` . . . 109
8.3 Cykly a zoznamy . . . 111
8.3.1 Funkcia `foreach` . . . 111
8.3.2 Funkcia `apply` . . . 112
8.3.3 Funkcia `mapcar` . . . 112
8.3.4 Rekurzia . . . 113

9 Práca so súbormi 115
9.1 Zápis a pripájanie do súboru . . . 115
9.2 Čítanie zo súboru . . . 117
9.3 Výber súborov . . . 117
9.4 Špeciálne znaky . . . 118
9.5 Otváranie súborov v externých programoch . . . 118

10 Entity a výberové množiny **120**
10.1 Entity . . . 120
10.1.1 Grafické entity . . . 120
10.1.2 Negrafické entity . . . 122
10.1.3 Úprava, vytváranie a odstraňovanie entít . . . 125
10.2 Výberové množiny . . . 127
10.2.1 Metódy výberu . . . 128
10.2.2 Filtre výberových množín . . . 130

1 Úvod

Softvér AutoCAD sa využíva v mnohých oblastiach už po desaťročia. Za toto obdobie, prirodzene, prešiel mnohými výraznými zmenami a s príchodom nových verzií sa stretávame s vylepšeniami celkovej funkcionality. Aj preto nie je ojedinelé, že si užívateľ môže vybrať z množstva spôsobov, ako chce svoju prácu vykonať. Napríklad, niekto radšej používa na ovládanie iba myš, iní užívatelia si pomáhajú aj klávesnicou.

Každý užívateľ AutoCADu si počas jeho používania vytvára svoj osobný postup prác, ktorý je do značnej miery jedinečný. Málokedy totiž kreslíme to isté a tým istým spôsobom ako iní užívatelia AutoCADu. Pri vyberaní toho správneho postupu je dôležitý čas, ktorý daný postup bude vyžadovať. Každý užívateľ by chcel optimalizovať čas strávený kreslením a pomocnú ruku v tomto smere podáva samotný AutoCAD. V prostredí AutoCADu totiž nájdeme veľmi veľa možností na prispôsobenie pracovného prostredia, úpravu tlačidiel a príkazov a v neposlednom rade aj automatizáciu postupov.

V skriptách budeme využívať softvér AutoCAD vo verzii 2023, pričom niektoré uvádzané možnosti, ako napr. programovanie v jazyku AutoLISP, nie je možné využiť vo verzii LT. V texte sa stretneme s rôznymi symbolmi, ikonami a fontami, ktoré by mali napomôť čitateľnosti a orientácii:

- Presné umiestnenie tlačidiel na páse kariet uvádzame v tvare Karta › Panel › Tlačidlo.
- **PRÍKAZY** a **SYSTÉMOVÉ PREMENNÉ** AutoCADu budeme uvádzať vždy veľkými tučnými písmenami. Rovnakým štýlom budeme uvádzať aj voľby v príkazoch.
- Funkcie a výrazy v jazyku AutoLISP uvádzame zvyčajne malými písmenami **takýmtofontom**. Farebne rozlišujeme **funkcie**, **"znakové ret'azce"**, **celé čísla**, **reálne čísla**, **zátvorky** a **komentáre**. Dlhšie časti kódov, najmä definície funcií uvádzame v samostatnej forme, pozri kód 1.1.

AutoLISP kód 1.1: Ukážka prostredia pre kódy v jazyku AutoLISP.

```
1 (defun pozdrav ()
2   (princ "Dobry den!")
3 )
```

- Makrá, resp. ich časti odlišujeme v texte použitím tohto fontu, pričom celé makrá uvádzame v samostatnej forme, pozri makro 1.1.

Makro 1.1: Ukážka prostredia pre makro príkazu

Retazec samotneho makra

- Skripty uvádzame v samostatnej forme, pozri skript 1.1.

Skript 1.1: Ukážka prostredia pre skript

```
Obsah skriptu, zvycajne rozdeleny
na viac riadkov
```

- Cestu k priečinkom uvádzme v nasledovnej forme `C:\AutoCAD` a podobne aj názvy súborov `vykres.dwg`.
- Interakcie užívateľa s AutoCADom prezentujeme prostredím podobnému príkazovému riadku AutoCADu. Najčastejšie využitie nájdeme pri spúšťaní krátkych výrazov v jazyku AutoLISP, napríklad:

2 Prispôsobovanie užívateľského rozhrania

AutoCAD je softvér, ktorý má veľmi široké uplatnenie. Aby vyhovel veľkému množstvu užívateľov z najrôznejších odvetví, obsahuje veľké množstvo príkazov a nastavení. V konečnom dôsledku sa často stáva, že AutoCAD je pre užívateľa príliš všeobecný a/alebo nedostatočne špecializovaný. Nedostatočnú špecializáciu AutoCADu vieme vyriešiť rozšíreniami a doplnkami, alebo automatizáciou, resp. programovaním, pozri kapitoly 3 až 10.

Prispôsobením užívateľského prostredia vieme veľmi rýchlo a jednoducho zvýšiť produktivitu práce v AutoCADe. Môžeme napr. odstrániť nevyužívané tlačidlá alebo naopak najpoužívanejšie tlačidlá môžeme preskupiť tak, aby sme ich mali všetky na jednom mieste a vždy na dosah.

Možnosť prispôsobovania užívateľského prostredia na užívateľa číha v AutoCADe takpovediac na každom kroku. Pridávať, odoberať alebo presúvať vybrané prvky užívateľského rozhrania (ďalej UI z angl. User Interface) totiž môžeme aj bez použitia príkazov. V nasledujúcich častiach tejto kapitoly budeme rozlišovať tieto spôsoby úpravy:

- priama úprava užívateľského rozhrania (UI),
- úprava prvkov UI pomocou dialógového okna Customize User Interface (**CUI**),
- úprava palety Tool Palettes (**TOOLPALETTES**),
- úpravu aliasov príkazov,
- nastavenia v dialógovom okne Options (**OPTIONS**) a práca s profilmi.

Okrem horeuvedených možností ako si prispôsobiť AutoCAD môžeme využiť ešte ďalšie možnosti úprav, napr. viditeľnosti prvkov UI (napr. navigačná kocka), špecifické nastavenia paliet (napr. paleta príkazový riadok). V niektorých prípadoch majú tieto nastavenia aj užívateľské rozhranie, väčšina nastavení sa však dá nastaviť iba priamo príkazom alebo zmenou hodnoty systémovej premennej v príkazovom riadku.

Priamo vieme upravovať tlačidlá na paneli rýchleho prístupu, karty alebo panely na páse kariet, alebo palety, ktoré môžu byť plávajúce, alebo ich možno pripnúť na vybranom mieste. Priame úpravy sú síce rýchle, ale ponúkajú značné obmedzenia (napr. nedokážeme vytvárať nové tlačidlá). Podstatne viac možností prispôsobenia získame pomocou príkazu **CUI**, kde okrem úprav viditeľných prvkov UI môžeme upravovať aj akcie dvojkliku alebo klávesové skratky. Tieto nastavenia užívateľského rozhrania sa ukladajú do súborov vo formáte `*.cuix` a môžeme ich jednoducho prenášať medzi rôznymi inštaláciami AutoCADu. Úpravy palety (alebo skupiny paliet) z palety nástrojov môžeme realizovať mimo príkazu **CUI** a uložiť palety do samostatných súborov vo formáte `*.xtp` (resp. skupiny paliet vo formáte `*.xpg`). Samostatne sa upravujú aj aliasy príkazov, ktoré sú uložené v textovom súbore `acadiso.pgp`. Pomocou profilov dokážeme ukladať stavy niektorých z doteraz uvedených nastavení, rovnako ako aj niektoré ďalšie nastavenia z dialógového okna Options, navyše uložené profily je možné prenášať medzi rôznymi inštaláciami AutoCADu.

Tip

Ak z akéhokoľvek dôvodu potrebujete obnoviť AutoCAD do stavu po inštalácii, môžeme využiť samostatný program **Reset Settings To Default**, ktorý môžeme nájsť v priečinku **AutoCAD 2023** po kliknutí na v prostredí Windows.

2.1 Priame úpravy užívateľského rozhrania

V AutoCADe dokážeme priamo (bez spúšťania príkazov) upravovať UI (užívateľské rozhranie) len v obmedzenom rozsahu. Na druhej strane sú priame úpravy intuitívne, jednoduché a rýchlejšie v porovnaní s úpravami cez príkaz **CUI**, pozri časť 2.2.

Pri priamych úpravách (s výnimkou úprav panelu rýchleho prístupu) treba mať na pamäti, že sú len dočasné, kým nezmeníme aktívny pracovný priestor (Workspace). Pokiaľ chceme, aby aj priame úpravy boli trvalé, môžeme zapnúť ukladanie zmien pri prepnutí AutoCADu do iného pracovného prostredia zmenou systémovej premennej **WSAUTOSAVE** na hodnotu **1**. Túto zmenu môžeme urobiť aj v dialógovom okne Workspace Settings, ktoré zobrazíme príkazom **WSSETTINGS**. V tomto dialógovom okne treba v časti When Switching Workspaces zvoliť možnosť Automatically save workspace changes.

Tip

Priame úpravy nemusíme ukladať len do aktuálneho pracovného prostredia, ale môžeme ich uložiť aj ako úplne nový pracovný priestor.

Priamo môžeme upravovať :

- panel rýchleho prístupu,
- poradie a viditeľnosť panelov a kariet na páse kariet
- pozíciu, rozmery a viditeľnosť paliet (vrátane pásu kariet).

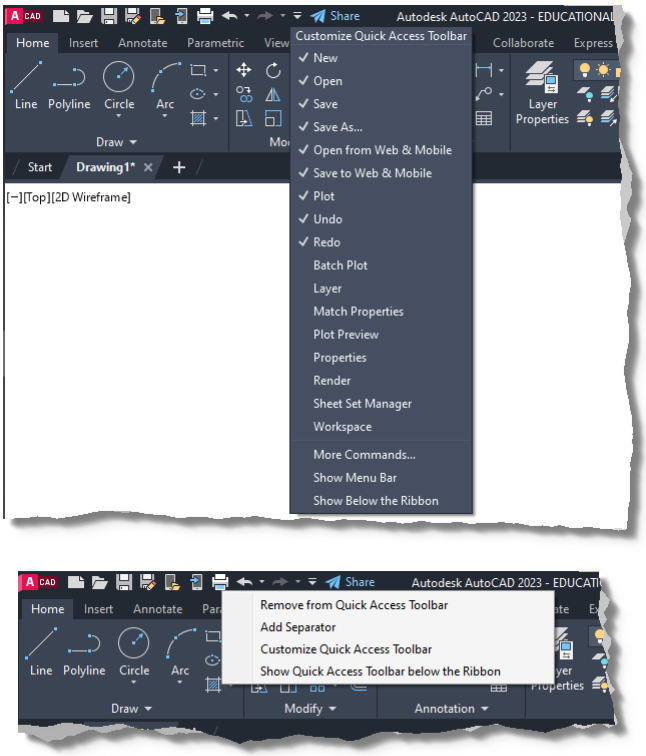
2.1.1 Panel rýchleho prístupu

Obsah panelu rýchleho prístupu môžeme upravovať, pozri obr. 2.1:

- (ne)začiarknutím prvkov v ponuke, ktorú zobrazíme po kliknutí na tlačidlo ,
- kliknutím na vybrané tlačidlo na páse kariet zobrazíme kontextovú ponuku v ktorej voľbou Add to Quick Access Toolbar pridáme toto tlačidlo aj na panel rýchleho prístupu. Podobne kliknutím na vybrané tlačidlo na paneli rýchleho prístupu a výberom voľby Remove from Quick Access Toolbar v kontextovej ponuke toto tlačidlo odstránime,
- pomocou dialógového okna Customize User Interface, pozri časť 2.2.1.2. Toto dialógové okno môžeme otvoriť napr. voľbou More Commands... v ponuke, ktorú zobrazíme po kliknutí na tlačidlo .

2.1.2 Pás kariet

Poradie kariet na páse kariet môžeme meniť uchopením vybranej karty za jej názov a pustením na požadovanom mieste, pozri obr. 2.2. Podobne, uchopením panelu za jeho názov, ho môžeme presúvať

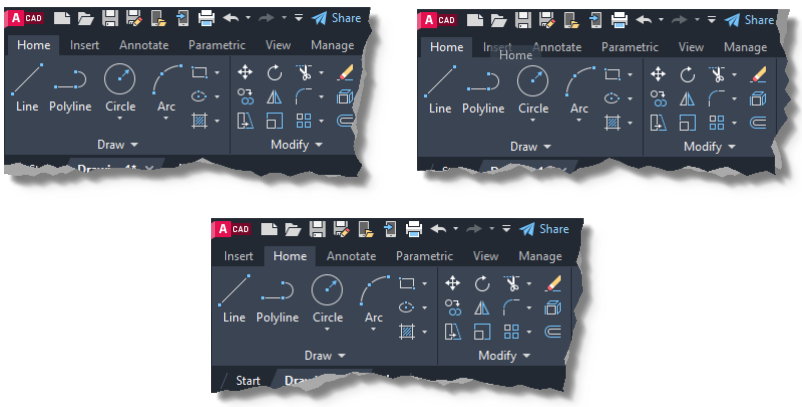


Obr. 2.1: Priame úpravy na panely rýchleho prístupu

na karte. V prípade, že uchopený panel uvoľníme mimo pásu kariet, stane sa z neho plávajúci panel. Takýto panel môžeme kedykoľvek uchopiť a presunúť kamkoľvek, napr.aj naspäť na pás kariet, pozri obr. 2.3. Viditeľnosť kariet a panelov môžeme nastaviť pomocou kontextového menu, ktoré zobrazíme po kliknutí kdekoľvek na páse kariet. V tomto kontextovom menu (ne)začiarknutím jednotlivých kariet v časti Show Tabs alebo panelov v časti Show Panels môžeme určiť viditeľnosť jednotlivých kariet a panelov.

2.1.3 Palety

Pod viditeľnosťou paliet rozumieme to, či je paleta zobrazená, alebo nie. Niektoré palety sú predvolene zobrazené (pás kariet alebo príkazový riadok), iné treba v prípade potreby zobraziť pomocou príkazov. Pre zobrazenie a zatvorenie niektorých paliet môžeme použiť aj klávesové skratky, pozri časť 2.2.1.7.



Obr. 2.2: Presúvanie kariet na páse kariet

Obr. 2.3: Presúvanie panelu v rámci karty a ukážka plávajúceho panela

Napr. ctrl + 1 zobrazí paletu Properties ak nie je zobrazená, alebo ju zavrie ak je zobrazená. Okrem toho, paletu môžeme zavrieť aj krížikom v jednom z jej horných rohov.

Pozíciu palety môžeme meniť uchopením palety (za časť s jej názvom) a presunutím na požadované miesto. Ak toto miesto bude príliš blízko pri ľavom alebo pravom okraji grafického okna AutoCADu, paleta sa bude snažiť pripnúť k okraju. Takéto automatické ukotvenie je v AutoCADe predvolene zapnuté, ale v prípade potreby sa dá vypnúť pre každú paletu zvlášť. Z pripnutej palety môžeme urobiť opäť plávajúcu uchopením palety a presunutím mimo okrajov pracovného okna.

Paleta sa môže aj automaticky skrývať (Auto-hide). To znamená že z palety zostane viditeľný len pás s jej názvom a pre zobrazenie celej palety musíme umiestniť kurzor na tento pás. Túto vlastnosť palety ovládame tlačidlom v hornej časti palety:

- tlačidlo sa zobrazuje ak sa paleta automaticky skrýva. Po stlačení tohto tlačidla sa automatické skrývanie vypne.
- tlačidlo sa zobrazuje ak sa paleta zobrazuje celá (automaticky sa neskrýva). Po stlačení tohto tlačidla sa automatické skrývanie zapne.

2.2 Úpravy pomocou príkazu CUI

Prvky užívateľského prostredia, ktoré upravujeme v dialógovom okne Customize User Interface sú uložené v súboroch užívateľských úprav vo formáte `*.cuix`. Po inštalácii AutoCADu 2023 sú to súbory:

- `acad.cuix` – hlavný súbor, ktorý sa vždy načíta ako prvý a na rozdiel od ostatných (čiastkových) súborov obsahuje aj definície pre pracovný priestor (Workspace),
- `custom.cuix` – súbor je prázdny a pripravený na ukladanie prvkov vytvorených užívateľom,
- `modeldoc.cuix` – obsahuje prvky užívateľského prostredia na tvorbu pohľadov z 3D objektov AutoCADu, prípadne súborov programu AutoDesk Inventor, ktoré nájdeme napr. na paneli Home › View,
- `acetmain.cuix` – obsahuje prvky užívateľského prostredia pre tzv. Express Tools na karte Express Tools a na paneloch nástrojov,
- `appmanager.cuix` – obsahuje prvky užívateľského prostredia na karte Add-ins,
- `featuredapps.cuix` – obsahuje prvky užívateľského prostredia na karte Featured Apps.

Zoznam týchto súborov `*.cuix` po inštalácii sa môže líšiť v závislosti od verzie AutoCADu a výberu z možností počas inštalácie. Inštalovaním rôznych doplnkov a rozšírení sa počet súborov `*.cuix` môže zväčšovať, zväčša totiž okrem nových funkcionalít obsahujú aj užívateľské rozhranie definované práve v súboroch `*.cuix`. Takýto súbor môže vytvoriť, prípadne upravovať aj samotný užívateľ, pozri časť 2.2.2.

Tip

Súbory `*.cuix` sú premenované ZIP archívy, v ktorých nájdeme množstvo ďalších súborov. Väčšinou sú to `*.cui` súbory, ktoré definujú jednotlivé prvky užívateľského prostredia. Ak `*.cuix` súbor obsahuje vlastné ikonky k tlačidlám, nájdeme tu aj obrázky vo formáte `*.bmp`. Zaujímavosťou je, že `*.cui` súbory sú textové súbory napísane v jazyku xml. Dajú sa teda upravovať aj v textovom editore, čo je vhodné najmä pri hromadných úpravách.

Pre prácu so súbormi `*.cuix` je najvhodnejším spôsobom dialógové okno Customize User Interface, pozri obr. 2.4. Otvoriť ho môžeme pomocou príkazu **CUI**, resp. tlačidlom Manage › Customization › User Interface. Toto dialógové okno má dve karty:

- Customize – na úpravu prvkov UI (pre všetky alebo vybraný `*.cuix` súbor),
- Transfer – na prenos prvkov UI medzi dvomi `*.cuix` súbormi.

V dialógovom okne Customize User Interface môžeme pracovať s nasledovnými prvkami UI:

- Workspaces – pracovné prostredia, v ktorých sú uložené aktuálne rozloženia a viditeľnosť prvkov UI. V AutoCADe sú po inštalácii štandardne 3 pracovné prostredia: Drafting & Annotation, 3D Basics a 3D Modeling, ktoré môžeme ďalej upravovať alebo dopĺňať o ďalšie pracovné prostredia. Ako už naznačujú uvedené názvy, výber pracovného prostredia sa odvíja od toho, na čo AutoCAD používame. Ak napr. pracujeme iba v 2D, nepotrebujeme tlačidlá pre prácu s 3D objektami, preto je vhodné pracovať v prostredí Drafting & Annotation, v ktorom nájdeme najviac tlačidiel pre prácu v 2D. Samozrejme, používať príkazy môžeme aj bez tlačidiel, teda bez ohľadu na zvolené pracovné prostredie. Hlavným poslaním pracovného prostredia je teda uľahčenie prístupu k tlačidlám, ktoré často používame.
- Quick Access Toolbars – panel rýchleho prístupu sa nachádza v záhlaví programu, vedľa tlačidla  . Tento panel zvyčajne obsahuje často používané tlačidlá, je stále viditeľný a ľahko upravovateľný.
- Ribbon – táto paleta je jedným z najdôležitejších prvkov užívateľského prostredia, na ktorom nájdeme prehľadne a funkčne usporiadané tlačidlá. V posledných verziách AutoCADu pás kariet nahradil klasické menu. V porovnaní s klasickým menu alebo panelmi nástrojov (Toolbars) je práca s paletou Ribbon jednoduchšia.
- Toolbars – panely nástrojov, ktoré obsahujú tlačidlá, podobne ako panely na palete Ribbon. Panely nástrojov boli používané najmä v starších verziách AutoCADu, napr. v pracovnom prostredí AutoCAD Classic, ktoré dnes už nie je súčasťou nových inštalácií (ale dá sa importovať).
- Menus – predchodca pásu kariet (paleta Ribbon). Ide o klasické menu, ktoré v novších verziách AutoCADu býva predvolene vypnuté. Jeho zobrazenie je kontrolované systémovou premennou **MENUBAR**.
- Quick Properties – paleta zobrazujúca vybrané vlastnosti objektov vo výbere. Zobrazené vlastnosti je možné upravovať priamo v palete.

- Rollover Tooltips – náhľad zobrazujúci vybrané vlastnosti objektu na ktorom zastane kurzor myši. Zobrazené vlastnosti nie je možné upravovať v náhľade.
- Shortcut Menus – kontextové menu, ktoré sa zobrazí po kliknutí napr. na objekt, objekty alebo počas aktívnych príkazov.
- Keyboard Shortcuts — klávesové skratky na spustenie príkazov, alebo dočasnú zmenu hodnôt systémových premenných.
- Double Click Actions – dvojklikom na grafický objekt sa vykoná akcia priradená konkrétnemu typu objektu.
- Mouse Buttons – tlačidlám myši (s výnimkou) môžu byť priradené príkazy (makro príkazu, pozri časť 3.2), ktoré sa spustia po kliknutí daným tlačidlom. Rovnako je možné priradiť príkaz (makro) aj kombináciám s tlačidlom ctrl, shift alebo dvojicou ctrl + shift.
- LISP Files – zoznam súborov `*.lsp`, ktoré sa automaticky načítajú pri spustení AutoCADu.
- Legacy – menej používané (zastarané) prvky užívateľského prostredia.
- Partial Customization Files – toto vlákno nájdeme iba v hlavnom súbore `acad.cuix` a sú v ňom uvedené všetky používané čiastkové súbory užívateľských úprav `*.cuix`.

V nasledujúcich častiach 2.2.1 a 2.2.2 sa budeme venovať najdôležitejším prvkom UI, resp. možnostiam ich prenosu medzi súbormi užívateľským úprav.

2.2.1 Úprava prvkov UI

Na úpravu prvkov UI budeme používať kartu Customize na dialógovom okne Customize User Interface, pozri obr. 2.4.

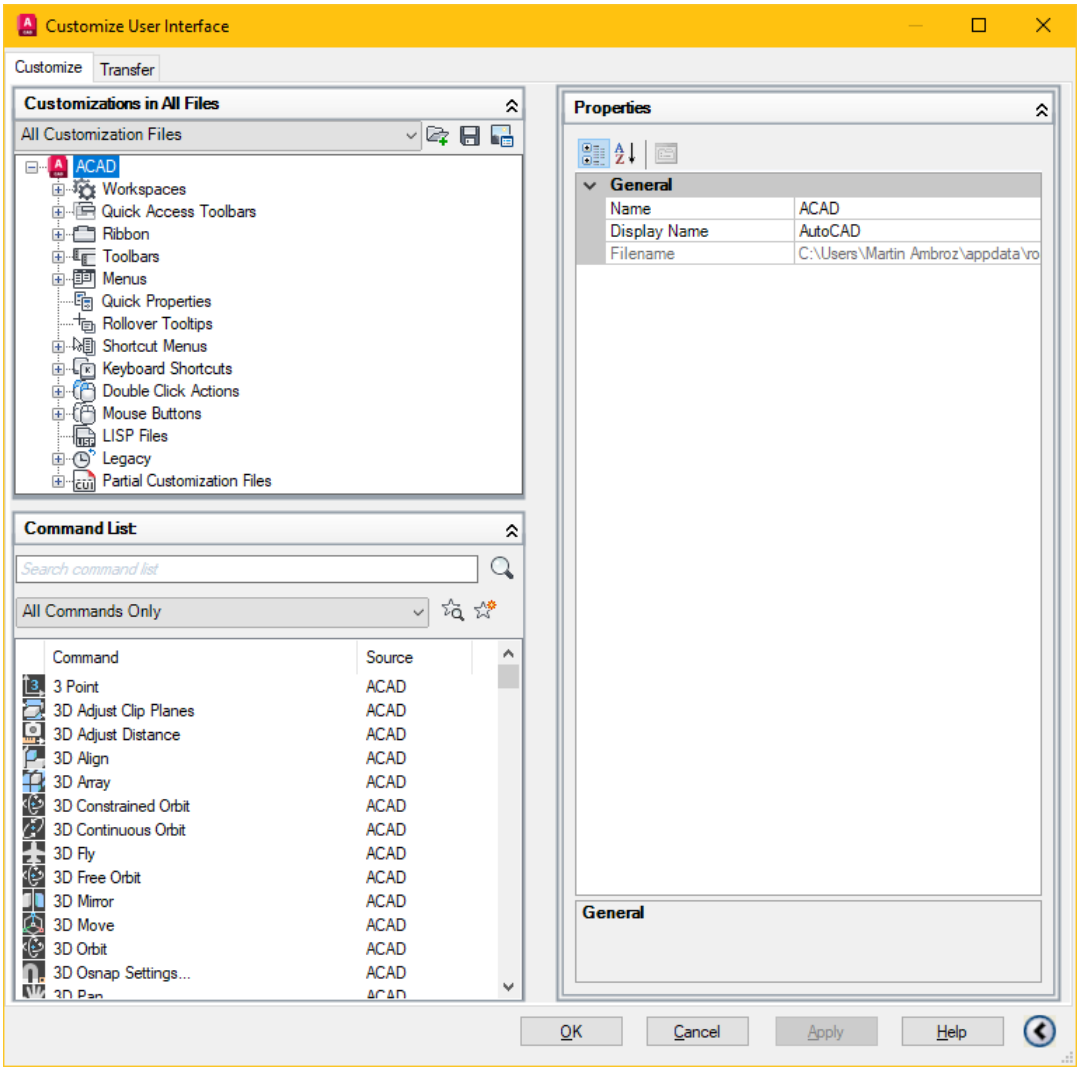
Táto karta má po otvorení dialógového okna tri podokná:

- Customizations in All Files: Úprava všetkých alebo vybraných `*.cuix` súborov, možnosť otvorenia ďalších `*.cuix` súborov a ich uloženia. Prvky UI sú v tomto podokne usporiadané formou stromu, ktorý môžeme upravovať. Úpravy vykonávame:
  - presúvaním prvkov v tomto podokne,
  - presúvaním prvkov z podokna Command List,
  - pomocou kontextovej ponuky po kliknutí ,
  - úpravou vlastností vybraného prvku v podokne Properties.
- Command List: Zoznam všetkých príkazov (presnejšie ide o príkazové makro, pozri časť 3.2), pre ktoré sú v UI vytvorené tlačidlá.
- Properties: Vlastnosti vybraného prvku UI. Po otvorení dialógového okna sa zobrazujú vlastnosti súboru `acad.cuix`.

Pravá časť dialógového okna Customize User Interface je dynamická. Počas práce v dialógovom okne, s ohľadom na vybraný prvok v podokne Customizations in All Files alebo Command List, sa podokno Properties nahradí iným podoknom, alebo sa rozdelí na dve podokná. Postupne pri jednotlivých úpravách opíšeme všetky nasledujúce podokná:

- Information,

- Workspace Contents,
- Panel Preview,
- Toolbar Preview,
- podokno pre úpravu Quick Properties a Rollover Tooltips,
- Shortcuts,
- Button Image.



Obr. 2.4: Dialógové okno Customize User Interface

Tip

Pravú časť dialógového okna Customize User Interface môžeme kliknutím na skryť (zbaliť), resp. kliknutím na zobraziť (rozbaliť). V rozbalenom stave sa dialógové okno otvorí napr. po zavolaní príkazu **CUI**, naopak v zbalenom stave sa otvorí po zavolaní príkazu **QUICKCUI** alebo **TOOLBAR**.

V podokne Customization in All Files je pomocou rozbaľovacieho zoznamu určené, ktorý súbor alebo súbory aktuálne upravujeme. Predvolenou voľbou sú všetky súbory (All Customization Files). V zozname

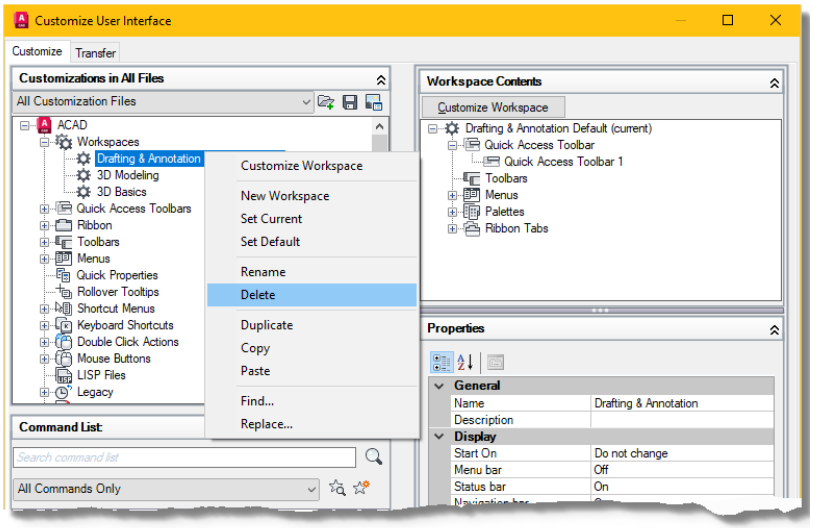
však máme na výber aj samostatné `*.cuix` súbory. Výberom jedného z nich sa zmení názov podokna na Customization in Main File a zobrazí sa obsah iba vybraného súboru. Na výber máme z hlavného `*.cuix` súboru a všetkých čiastkových `*.cuix` súborov, pozri časť 2.2.1.11. Okrem týchto súborov tu môžeme otvoriť aj ďalšie `*.cuix` súbory voľbou Open... v rozbaľovacom zozname, prípadne príslušným tlačidlom vedľa tohto zoznamu.

2.2.1.1 Workspaces

Po inštalácii AutoCADu má užívateľ na výber z troch pracovných prostredí, ktoré vieme upravovať, odstraňovať prípadne pridávať ďalšie pracovné prostredia.

Pridávať a odoberať samotné pracovné prostredia môžeme po kliknutí na jedno z pracovných prostredí výberom z kontextovej ponuky, pozri obr. 2.5:

- New Workspace vytvorí nové (prázdne) pracovné prostredie,
- Duplicate vytvorí kópiu vybraného pracovného prostredia,
- Delete odstráni vybrané pracovné prostredie.



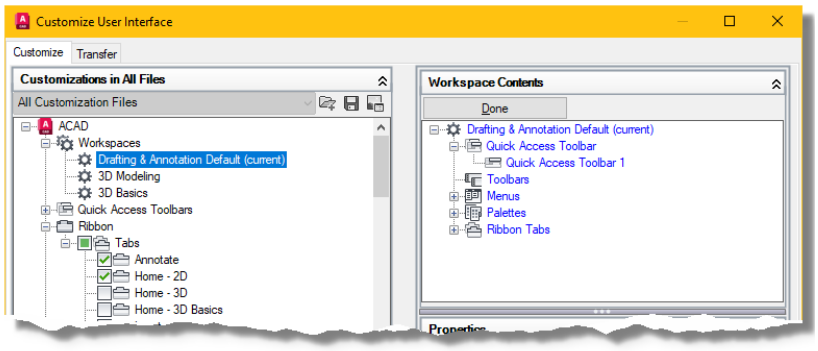
Obr. 2.5: Obsah pracovného prostredia otvorený v dialógovom okne Customize User interface. Po kliknutí na vybrané pracovné prostredia sa zobrazí kontextová ponuka.

Upravovať obsah pracovných prostredí môžeme pomocou podokna Workspace Contents, v ktorom sa zobrazuje obsah vybraného pracovného prostredia, pozri obr. 2.5. Po kliknutí na tlačidlo Customize Workspace v hornej časti podokna Workspace Contents sa zmení farba obsahu tohto podokna na modrú a v podokne Customization in All Files sa zobrazia pri jednotlivých prvkoch (panely rýchleho prístupu, karty, panely, panely nástrojov, menu a čiastkové súbory `*.cuix`) začiarkavacie políčka, pozri obr. 2.6. Začiarknutím, resp. odčiarknutím jednotlivých prvkov sa priradia, resp. odoberú z aktuálne upravovaného pracovného prostredia.

Tu sa môžeme stretnúť s tromi stavmi začiarkavacích políčok:

- ☐ žiadny z prvkov vo vnorenom vlákne nie je začiarknutý,
- ▣ niektoré z prvkov vo vnorenom vlákne sú začiarknuté,
- ☑ všetky prvky vo vnorenom vlákne sú začiaknuté.

Úpravy ukončíme tlačidlom Done v hornej časti podokna Workspace Contents, pozri obr. 2.6.



Obr. 2.6: Obsah pracovného prostredia otvorený na editáciu (v pravej časti modrým písmom) v dialógovom okne Customize User interface

2.2.1.2 Quick Acces Toolbars

Hoci môže byť definovaných viacero panelov rýchleho prístupu (PRP), použitý (viditeľný) môže byť vždy iba jeden. Môžeme si teda vyberať, ktorý panel rýchleho prístupu chceme zobraziť, resp. priradiť do Workspace u. Nový PRP vytvoríme:

- kliknutím priamo na uzol Quick Acces Toolbars alebo ľubovoľný jeho poduzol a výberom možnosti New Quick Acces Toolbar, alebo
- kliknutím priamo na uzol ľubovoľného PRP a výberom možnosti Duplicate, čím vytvoríme kópiu vybraného PRP.

V dialógovom okne Customize User Interface sa do PRP dajú prvky pridávať:

- „naťahaním" vybraných tlačidiel z podokna Command List, alebo
- „kopírovaním" prvku z podokna Command List pomocou volieb v kontextovom menu po stlačení alebo pomocou ctrl + C a ctrl + V. Týmto spôsobom však nevytvárame kópie tlačidiel, len ich umiestňujeme v rámci užívateľského prostredia.

Okrem pridávania jednotlivých tlačidiel máme možnosť pridať na PRP aj rozbaľovacie tlačidlo, do ktorého môžeme umiestniť viacero tlačidiel. Kliknutím priamo na uzol alebo poduzol ľubovoľného PRP a výberom možnosti New Drop-down.

Okrem pridávania tlačidiel, môžeme do PRP pridať aj iné prvky, ako napr. rozbaľovací zoznam (hladín, štýlov), alebo prepínač (zobrazenia paliet).

Vymazanie prvku z PRP je možné pomocou klávesu del alebo kliknutím na prvok a výberom Remove v kontextovej ponuke.

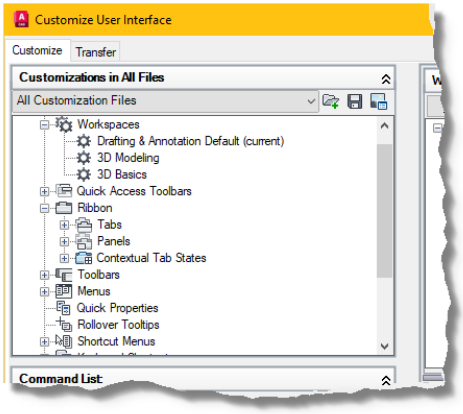
2.2.1.3 Ribbon

Paleta Ribbon pozostáva z viacerých kariet, na ktorých sú usporiadané panely. Na jednom paneli je spravidla niekoľko tlačidiel v rôznych formách. Každé tlačidlo má priradený príkazové makro, pozri časť 3.2, ktoré sa spustí po jeho stlačení. Pri upravovaní užívateľského prostredia je dobré mať túto štruktúru na pamäti.

Tip

Paleta Ribbon je vždy iba jedna. V každom Workspace však môže mať iný vzhľad (iné karty, panely, iné umiestnenie palety...).

Na obr. 2.7 vidíme, že pod vlákno Ribbon patria okrem kariet (Tabs) a panelov (Panels) aj kontextové stavy kariet (Contextual Tab States). Kontextové stavy kariet sa viažu na splnenie určitých podmienok, napr. spustenie príkazu (príkaz **HATCH** zobrazíme kontextovú kartu Hatch Creation) alebo výber objektu (po výbere objektu šrafy zobrazíme kontextovú kartu Hatch Editor). Po porušení podmienok (ukončenie príkazu, alebo zrušenie výberu) sa tieto karty skryjú.



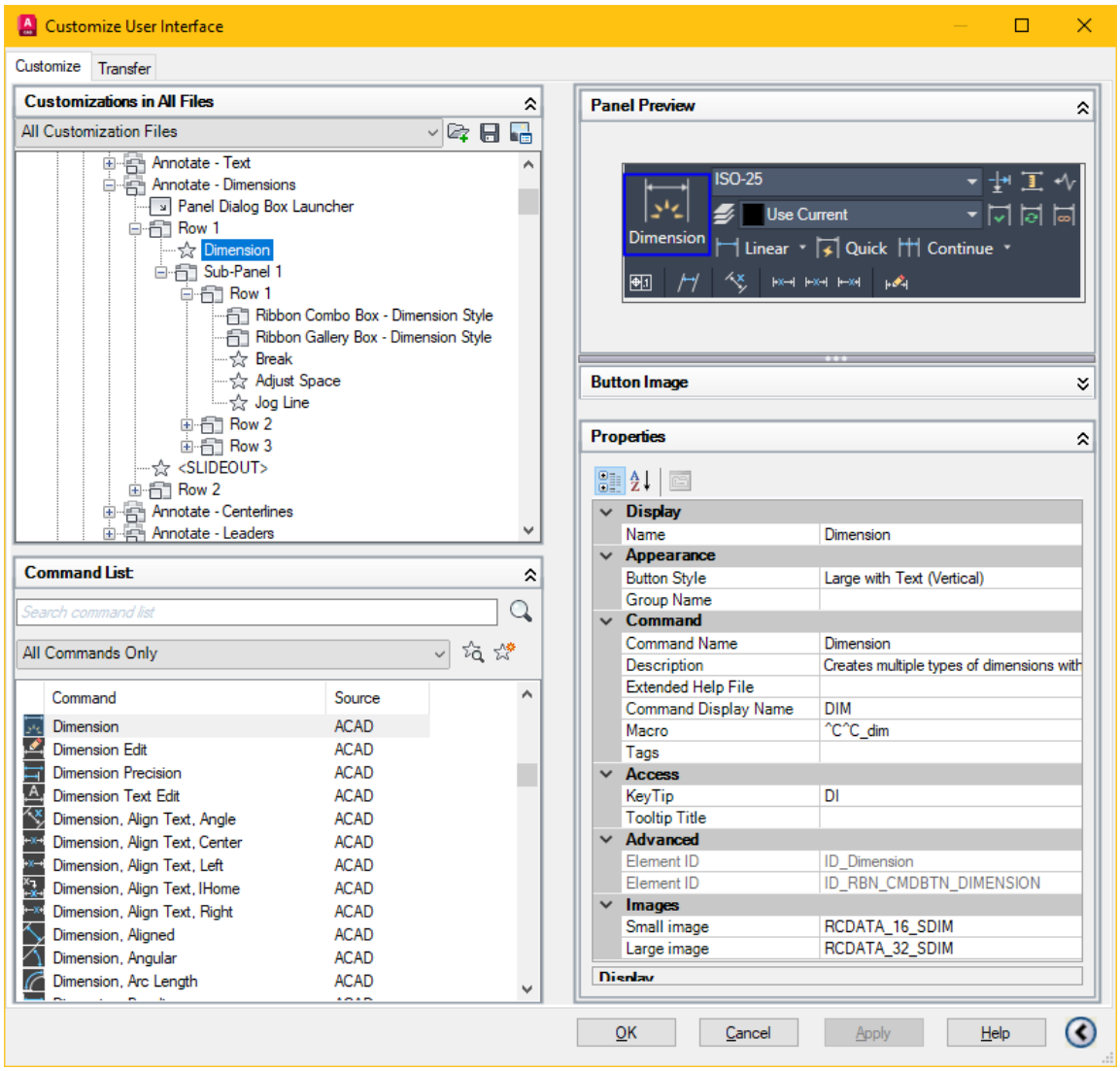
Obr. 2.7: Obsah pásu kariet (Karty, Panely a kontextové stavy kariet) otvorený v dialógovom okne Customize User interface

Tlačidlá Pod tlačidlom budeme rozumieť príkaz z podokna Command List. Ak vyberieme ľubovoľné tlačidlo, zobrazia sa vlastnosti tohto tlačidla, pozri obr. 2.8. Vybrať tlačidlo môžeme:

- v podokne Command List,
- v podokne Customization in All Files, čím sa automaticky vyberie tlačidlo aj v podokne Command List,
- v podokne Panel Preview kliknutím na konkrétne tlačidlo, čím sa automaticky vyberie tlačidlo aj v podokne Customization in All Files a rovnako aj v podokne Command List.

Týmto nám AutoCAD naznačuje, že akákoľvek zmena v tlačidle sa ukladá do definície tlačidla, pričom tlačidlo môže byť (a často aj býva) použité na viacerých miestach. Výnimkou sú vlastnosti, ktoré sa zobrazia iba pri výbere konkrétneho tlačidla (nie zo zoznamu v podokne Command List), ako napr. názov alebo štýl tlačidla.

Z vlastností tlačidiel budeme najviac pozornosti venovať vlastnosti Macro. V tejto vlastnosti definujeme príkazové makro, ktoré sa vykoná po stlačení tlačidla. Makrám sa bližšie venujeme v časti 3.2.



Obr. 2.8: Obsah a vlastnosti panelu Dimension na karte Annotate otvorený v dialógovom okne Customize User interface

Panely Panely môžme chápať ako „kontajnery" pre konkrétne prvky UI, ako napr. tlačidlá, rozbaľovacie tlačidlá alebo rozbaľovacie zoznamy. V paneloch môžu byť tlačidlá usporiadané do riadkov (Row) a pod-panelov (Sub-panels), pozri obr. 2.9. Obsah panelu vidíme formou klikateľného náhľadu v podokne Panel Preview. Vlastnosti v podokne Properties sa vzťahujú vždy k aktuálne vybranému prvku v podokne Customization in All Files (alebo v podokne Command List). Ak teda máme vo výbere napr. panel Annotate - Text, vidíme jeho vlastnosti a vieme napr. zmeniť jeho názov.

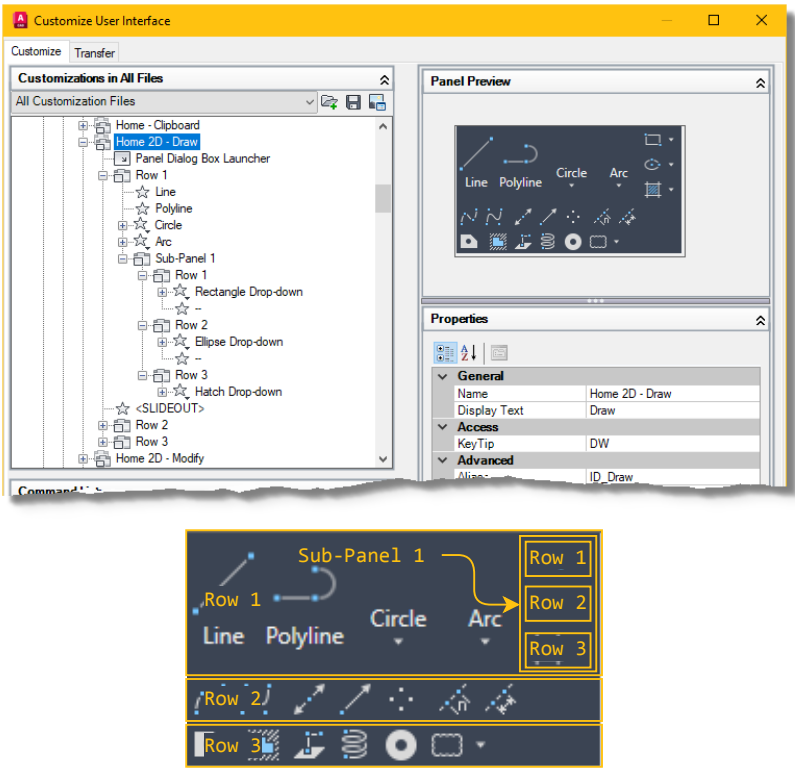
Vytvárať nové panely a odstraňovať nepotrebné panely môžeme po kliknutí na jedno z pracovných prostredí výberom z kontextovej ponuky, pozri obr. 2.5:

- New Panel vytvorí nový (prázdny) panel,
- Duplicate vytvorí kópiu vybraného panelu,
- Delete odstráni vybraný panel.

Obsah panelov môžeme upravovať:

- „ťahaním" vybraných tlačidiel z podokna Command List alebo v rámci podokna Customization in All Files, alebo

- „kopírovaním" vybraného prvku pomocou volieb v kontextovom menu po stlačení alebo pomocou ctrl + C a ctrl + V. Týmto spôsobom však nevytvárame kópie tlačidiel, len ich umiestňujeme v rámci užívateľského prostredia.



Obr. 2.9: Štruktúra panelu *Home 2D - Draw*, ktorý nájdeme na páse kariet Home › Draw. Panel má tri hlavné riadky, pričom prvý riadok obsahuje aj pod-panel s tromi vnorenými riadkami pre tri rozbaľovacie tlačidlá (pre kreslenie obdĺžnika, elipsy a šrafovanie).

Na paneloch sa môžeme stretnúť aj s rozbaľovacími tlačidlami, ktoré v sebe spájajú hneď niekoľko tlačidiel zo zoznamu v podokne Command List. Takéto rozbaľovacie tlačidlo môžeme pridať pomocou voľby New Drop-down v kontextovom menu po kliknutí na vybraný riadok panelu (Row) vo vlákne v podokne Customization in All Files. Vytvoríme tým nové vlákno pre novo vytvorené rozbaľovacie tlačidlo. Do tohto vlákna môžeme pridať príkazy, z ktorých si bude užívateľ vyberať po rozbalení tlačidla.

Tip

Na zobrazenie panelu v UI AutoCADu nestačí samotné vytvorenie panelu v dialógovom okne Customize User Interface. Panel je nutné priradiť ku karte, na ktorej sa má zobraziť.

Celkové rozmery panelu obmedzené nie sú (resp. len veľkosťou monitora). Obmedzenia sa aplikujú pri zobrazovaní panelu na páse kariet. Viditeľná výška panelov na páse kariet je obmedzená na jeden riadok. Pokiaľ má panel viac riadkov, druhý a ďalšie riadky sa presunú do rozbaľovacej časti panelu (Slideout). V dialógovom okne Customize User Interface je toto rozdelenie naznačené položkou < *SLIDEOUT* >, pozri obr. 2.9. Na vzhľad panelu má zásadný vplyv šírka ďalších panelov na karte. Šírka jednotlivých panelov je automaticky upravovaná tak, aby boli viditeľné všetky panely a čo najväčšia časť ich obsahu.

Tip

Niektoré panely sú rozbaľovacie. Rozbaľovací panel vieme rozoznať podľa znaku ▼ za názvom na spodnom okraji panelu. Pri plávajúcich paneloch sa môžeme stretnúť aj so znakom ▶ na pravom okraji panelu. Po kliknutí na spodný, resp. pravý okraj panelu na páse kariet sa tento panel rozbalí. Rozbalenú časť panelu môžeme pripnúť pomocou tlačidla umiestneného v ľavom dolnom rohu rozbaleného panelu. Pripnutý panel môžeme odopnúť pomocou tlačidla. Na niektorých paneloch môžeme v pravom dolnom rohu panelu vidieť aj tlačidlo, tzv. *Panel Dialog Box Launcher*, ktorým otvoríme priradené dialógové okno alebo paletu.

Karty Karty obsahujú jeden, alebo viac panelov. V dialógovom okne Customize User Interface ich nájdeme v podokne Customization in All Files vo vlákne Ribbon > Tabs. Obsah kariet môžeme upravovať podobným spôsobom, ako sme upravovali obsah panelov:

- „ťahaním" vybraných panelov v rámci podokna Customization in All Files z vlákna Ribbon > Panels do vlákna konkrétnej karty v Ribbon > Tabs, alebo
- „kopírovaním" vybraného panelu pomocou volieb v kontextovom menu po stlačení alebo pomocou ctrl + C a ctrl + V. Týmto spôsobom však nevytvárame kópie panelov, len ich umiestňujeme v rámci užívateľského prostredia.

Tip

Na zobrazenie karty v UI AutoCADu nestačí len kartu vytvoriť v dialógovom okne Customize User Interface. Kartu musíme priradiť do pracovného prostredia, v ktorom sa má karta zobrazovať, pozri 2.2.1.1.

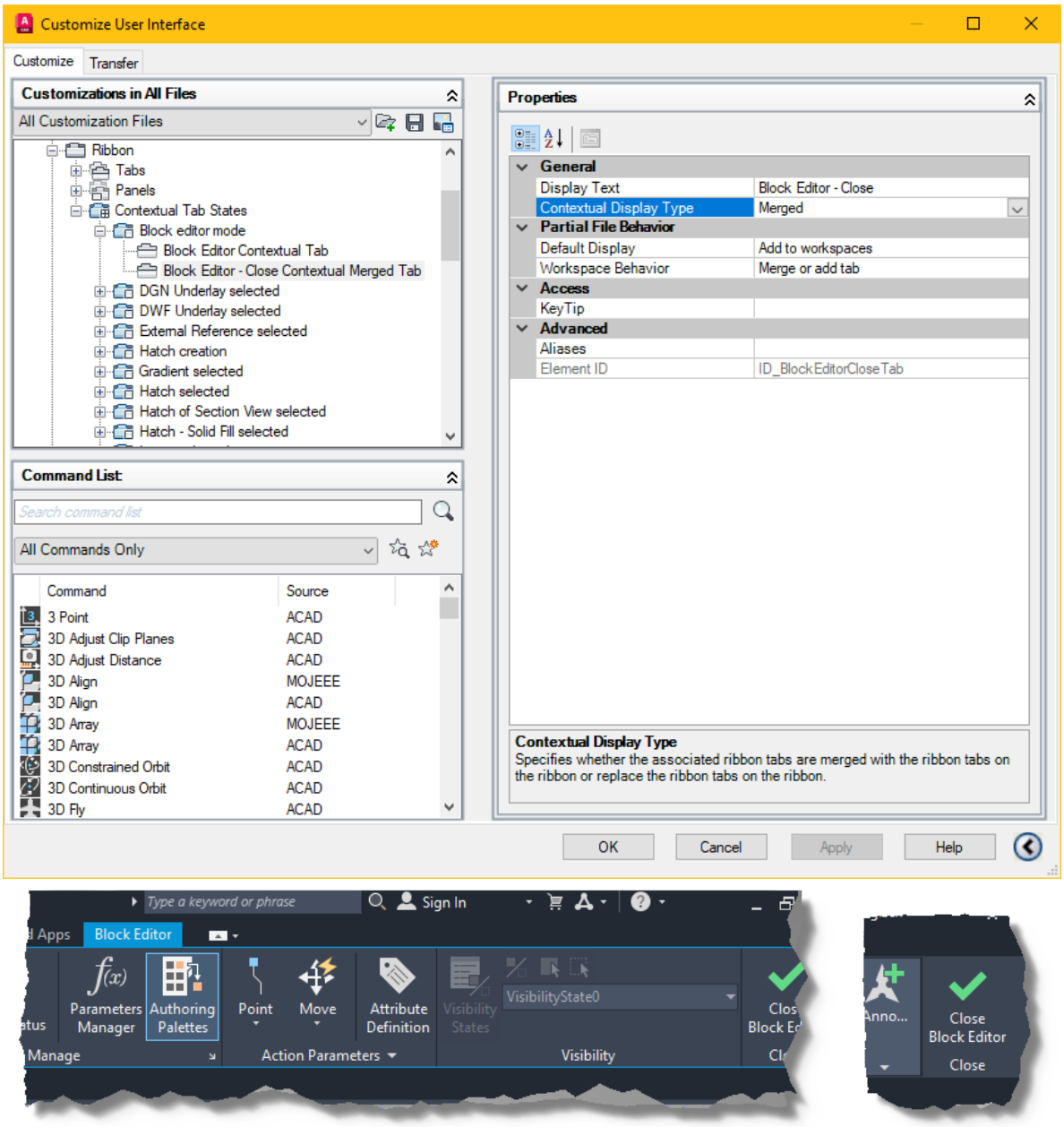
Rozmer karty je obmedzený len šírkou celkového okna AutoCADu. V prípade, že nie je možné zobraziť všetky panely priradené ku karte v celej šírke, AutoCAD nevyhnutné množstvo panelov, alebo tlačidiel na paneloch zmenší. To, ktoré panely sa zmenšujú ako prvé určuje vlastnosť Resize Style, ktorú má každý panel priradený karte. Napr. panel Annotate má túto vlastnosť v Ribbon > Tabs > Annotate. V závislosti na dôležitosti panelu môžeme zvoliť jednu z možností:

- Collapse as needed – panel sa môže zmenšovať v nevyhnutnej miere,
- Never collapse – panel sa nikdy nemôže zmenšovať,
- Collapse last – panel sa môže zmenšiť až po zmenšení ostatných panelov na karte.

Kontextové stavy kariet Kontextové stavy kariet môžeme len upravovať, nemôžeme vytvárať nové stavy. Jednotlivým kontextovým stavom kariet môžeme priradiť jednu alebo viac kariet. Každá z týchto priradených kariet sa zobrazuje jedným z troch nasledujúcich spôsobov, podľa nastavenia vlastnosti Contextual Display Type:

- Full with focus – samostatná karta, ktorá sa stane aktuálnou kartou,
- Full without focus – samostatná karta, pričom sa aktuálna karta nemení,
- Merged – obsah karty sa pridá na každú kartu na páse kariet.

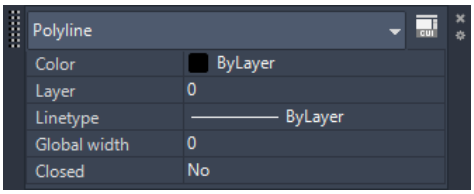
Napríklad, po spustení príkazu **BEDIT** sa zobrazí samostatné nová karta Block Editor. Okrem toho sa však pripojí k všetkým kartám pásu kariet aj panel Close, ktorý je priradený do ďalšej karty, pozri obr. 2.10.



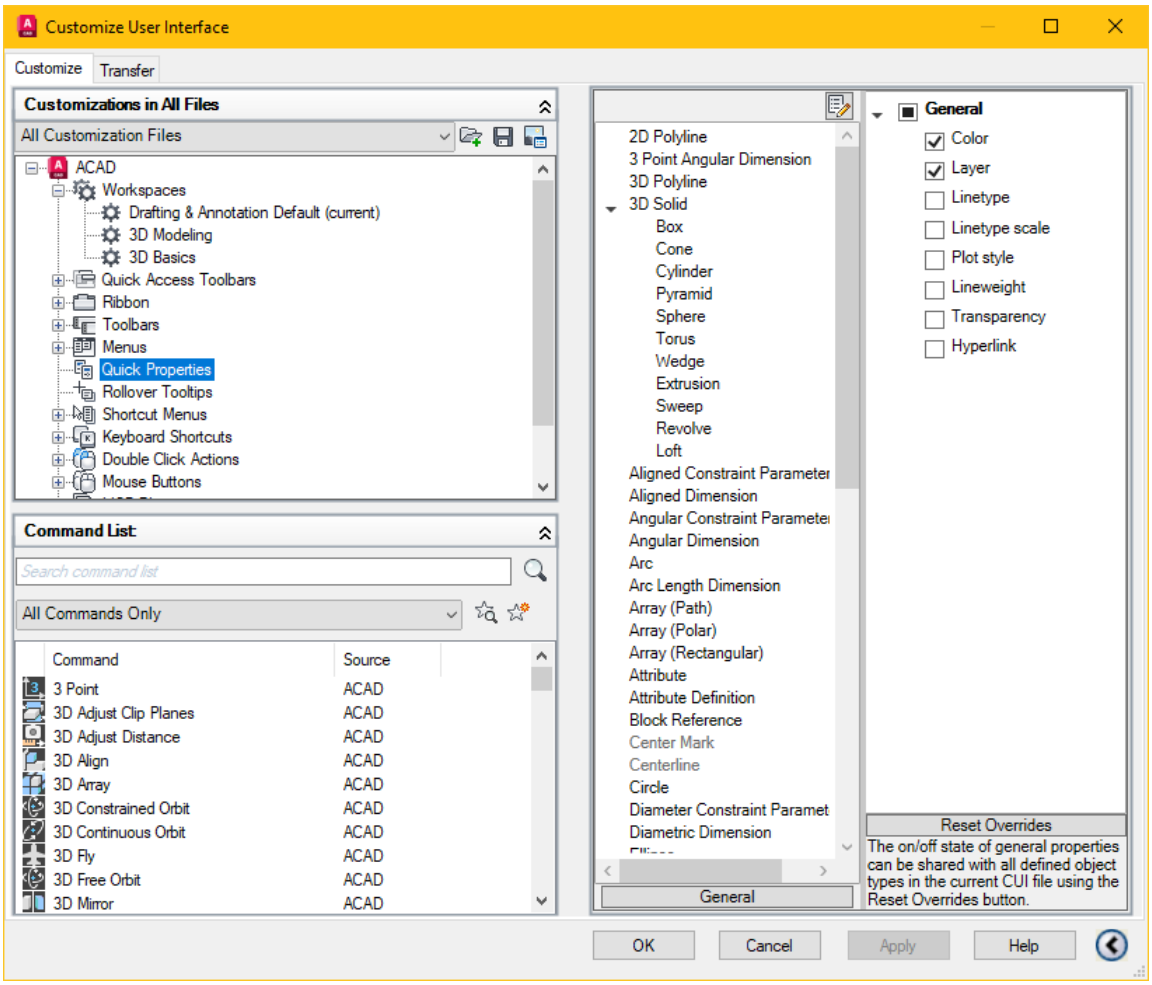
Obr. 2.10: Vlastnosti karty *Blok Editor - Close Contextual Merge Tab*, ktorá sa po spustení príkazu **BEDIT** zobrazí ako pripojená ku každej zobrazovanej karte. Táto karta obsahuje iba jeden panel Close s jedným tlačidlom Close Block Editor.

2.2.1.4 Quick Properties

Paleta Quick Properties, pozri obr. 2.11, je zjednodušená a dočasne zobrazená verzia palety Properties. Jej obsah môžeme prispôsobovať v dialógovom okne Customize User Interface vo vlákne Quick Properties. Po kliknutí na toto vlákno sa zobrazí v pravom podokne zoznam typov objektov a ich vybrané vlastnosti, ktoré sa majú zobrazovať po výbere týchto objektov. Konkrétne vlastnosti upravujeme začiarknutím alebo odčiarknutím v zozname vlastností. V spodnej časti pravého podokna môžeme pomocou tlačidla General určiť vlastnosti, ktoré chceme zobraziť pre ostatné objekty, ktoré nie sú v tomto zozname, pozri obr. 2.12.



Obr. 2.11: Palet Quick Properties po výbere objektu polyline



Obr. 2.12: Nastavenie obsahu palety Quick Properties v dialógovom okne Customize User Interface

Zoznam typov objektov môžeme upravovať v dialógovom okne Edit Object Type List, ktoré otvoríme

- tlačidlom  na vrchu zoznamu alebo

- výberom Edit Object Type List ... v kontextovom menu po kliknutí  v zozname typov objektov.

Zoznam upravuje jednoduchým začiarknutím alebo odčiarknutím jednotlivých typov objektov. Odstraňovať typy objektov zo zoznamu môžeme aj jednoduchšie, a to výberom Remove from Object Type List v kontextovom menu po kliknutí  na konkrétny typ objektu.

Tip

Aj keď je zobrazovanie palety Quick Properties vypnuté, môžeme jednorazovo zobraziť túto paletu pomocou príkazu **QUICKPROPERTIES**.

Zobrazovanie palety Quick Properties závisí od prepínača Quick Properties s ikonou  . Týmto prepínačom môžeme meniť hodnotu systémovej premennej **QPMODE**, ktorá môže mať jednu z nasledovných hodnôt:

- **2**, paleta sa zobrazuje len pri výbere tých typov objektov, ktoré sú vybrané v zozname typov objektov,
- **1**, paleta sa zobrazuje pri výbere všetkých typov objektov,
- **0**, paleta sa nezobrazuje pri žiadnom type objektov,
- **-1**, paleta sa nezobrazí pri žiadnom type objektov, ale po zapnutí prepínačom  sa zmení hodnota systémovej premennej **QPMODE** na **1**.
- **-2**, paleta sa nezobrazí pri žiadnom type objektov, ale po zapnutí prepínačom  sa zmení hodnota systémovej premennej **QPMODE** na **2**.

Tip

Paleta Quick Properties sa predvolene zobrazuje pri kurzore. Jej pozícia sa síce dá zmeniť po uchopení palety za jej pravý alebo ľavý okraj, ale pri ďalšom výbere objektov si paleta nezachová túto pozíciu. Ak by sme chceli paletu „pripnúť" na nejakom mieste, musíme zmeniť systémovú premennú **QPLOCATION** z hodnoty **0** na hodnotu **1**.

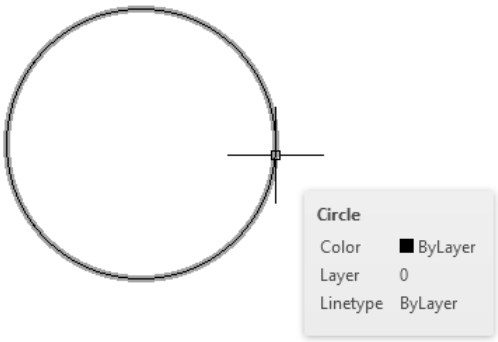
2.2.1.5 Rollover Tooltips

Prispôsobovať môžeme aj náhľad vybraných vlastností objektu, ktoré sa zobrazia pri zastavení kurzorom na objekte. Táto pomôcka (a aj jej prispôsobovanie cez dialógové okno Customize User Interface je veľmi podobná palete Quick Properties. Rozdiely však môžeme nájsť v tom, že:

- vlastnosti objektu sa v náhľade iba zobrazia, nemôžeme ich pomocou neho meniť,
- zobrazujú sa vlastnosti iba jedného objektu, na ktorom zastavil kurzor myši,
- objekt, ktorého vlastnosti chceme zobraziť nemusí byť vo výbere, pozri obr. 2.13.

Tip

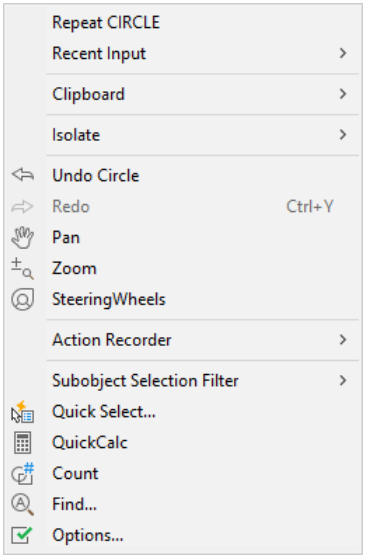
Zobrazovanie náhľadov sa je ovládané systémovou premennou **ROLLOVERTOOLTIPS**. Ak je nastavená na hodnotu **1**, náhľady sa zobrazujú, ak je nastavené na **0**, zobrazovanie je vypnuté. Nastavovať túto systémovú premennú môžeme aj v dialógovom okne Options, na karte Display začiarkavacím políčkom Show Rollover ToolTips.



Obr. 2.13: Príklad náhľadu vlastností kružnice po výbere objektu (vľavo) a bez toho, aby sme objekt vybrali (vpravo)

2.2.1.6 Shortcut Menus

Shortcut Menus môžu podstatne zvýšiť efektivitu práce užívateľov, ktorí preferujú prácu s myšou. Vďaka prispôsobeniu kontextového menu totiž môžeme mať najčastejšie používané príkazy vždy po ruke.



Obr. 2.14: Príklad kontextového menu (Shortcut Menu) po stlačení

Vo vlákne Shortcut Menus sú definované položky kontextového menu, ktoré sa zobrazí po kliknutí , pozri obr. 2.14. Ponuka v kontextovom menu je variabilná a je určená samostatne napríklad pre:

- stav bez spusteného príkazu a bez objektov vo výbere, vo vlákne Shortcut Menus › Default Menu,
- stav so spusteným príkazom, vo vlákne Shortcut Menus › Command Menu,
- určité objekty vo výbere, napr. pre objekty kót vo vlákne Shortcut Menus › Dimension Objects Menu.

Vytvoriť nové menu alebo odstrániť nepotrebné môžeme po kliknutí kdekoľvek vo vlákne Shortcut Menus a výberom z kontextovej ponuky:

- New Menu vytvorí nové menu,
- Delete odstráni vybrané menu.

Pri kliknutí na ktorékoľvek podvlákno vlákna Shortcut Menus máme navyše možnosť:

- Duplicate vytvoriť kópiu vybraného menu,
- New Sub-menu vytvoriť nové vnorené menu.

Tip

Pri tvorbe aliasov naviazaných na konkrétny typ objektu, často nepoznáme presné označenie typov objektov. Získať ho môžeme napríklad pomocou výberu od užívateľa pomocou výrazu v jazyku AutoLISP: `(cdr (assoc 0 (entget (car (entsel)))))`

Každé menu má vlastnosť Aliases, v ktorej je pomocou kódov definované, kedy je toto menu zobrazené. Prehľad vybraných kódov uvádzame v tabuľke 2.1 [1].

Tabuľka 2.1: Prehľad vybraných kódov pre vlastnosť Aliases

Kód aliasu	Popis
CMCOMMAND	kontextové menu počas akéhokoľvek spusteného príkazu
COMMAND_menoprikazu	kontextové menu počas spusteného konkrétneho príkazu menoprikazu, napr. COMMAND_LINE pre príkaz **LINE**,
CMDEFAULT	predvolené kontextové menu, t.j. bez spusteného príkazu a objektov vo výbere,
CMEDIT	kontextové menu v prípade, že je jeden alebo viac objektov vo výbere
OBJECT_typobjektu	kontextové menu v prípade, že je vo výbere jeden objekt typu typobjektu, napr. OBJECT_HATCH pre objekt šrafy,
OBJECTS_typobjektu	kontextové menu v prípade, že je vo výbere jeden alebo viac objektov typu typobjektu, napr. OBJECTS_POLYLINE pre objekty typu polyline,

2.2.1.7 Keyboard Shortcuts

Pomocou klávesových skratiek (nie skratiek príkazov, pozri časť 2.4) môžeme pohotovo meniť hodnoty spúšťať príkazy alebo meniť hodnoty systémových premenných. V dialógovom okne Customize User Interface sa vo vlákne Keyboard Shortcuts rozlišujú dva typy skratiek:

- Shortcut Keys - klávesové skratky, ktorými spustíme príkaz (presnejšie makro príkazu, pozri časť 3.2), napr. príkaz **PLOT** spustíme skratkou ctrl + P alebo paletu Properties vieme zobraziť/skryť pomocou skratky ctrl + 1,
- Temporary Override Keys - dočasné klávesové skratky. ktorými počas ich držania dočasne zmeníme hodnoty systémových premenných, napr. uchopovanie iba na koncové body pomocou klávesov shift + E.

V dialógovom okne Customize User Interface môžeme vytvárať nové klávesové skratky príkazov vo vlákne Keyboard Shortcuts > Shortcut Keys

- „ťahaním" vybraných príkazov z podokna Command List alebo v rámci podokna Customization in All Files, alebo
- „kopírovaním" vybraného prvku pomocou volieb v kontextovom menu po stlačení alebo pomocou ctrl+C a ctrl+V. Týmto spôsobom však nevytvárame kópie tlačidiel, len ich umiestňujeme v rámci užívateľského prostredia.

Po pridaní nového príkazu do vlákna Keyboard Shortcuts > Shortcut Keys je ešte potrebné priradiť mu klávesovú skratku. V podokne Properties sa zobrazujú vlastnosti príkazu po jej výbere v podokne Shortcuts alebo Customization in All Files. Klávesová skratka je určená vlastnosťou Key(s).

Tip

Pri priraďovaní klávesových skratiek sa môže stať, že priradíme príkazu klávesovú skratku, ktorá sa už používa. Ak skratku priraďujeme priamo prepisovaním hodnoty Key(s), AutoCAD neupozorní na kolíziu a skratku priradí novému príkazu. V ostatných prípadoch (ak použijeme na editáciu tlačidlo ... vo vlastnosti Key(s), alebo priamo v podokne Shortcuts) sa zobrazí upozornenie a užívateľ rozhodne, ktorý príkaz bude uprednostnený a bude skratku používať.

Dočasné klávesové skratky vieme vytvoriť po kliknutí na New Temporary Overide v kontextovom menu po kliknutí na vlákno Keyboard Shortcuts > Temporary Override Keys v podokne Properties. Okrem samotnej kombinácie klávesov cez vlastnosť Key(s) je ešte potrebné určiť makro príkazu, ktoré sa vykoná pri stlačení klávesov. Po uvoľnení klávesov sa obnoví pôvodný stav, prípadne je možné určiť aj makro, ktoré sa vykoná po uvoľnení klávesov.

2.2.1.8 Double Click Actions

Dvojklikom na grafické objekty môžeme spustiť príkaz. Táto funkcionalita AutoCADu je pomerne často využívaná a je prednastavená. Napr. pri dvojkliku na objekt jednoriadkového textu sa spustí príkaz **TEXTEDIT**, alebo dvojklikom na referenciu bloku (bez atribútu) sa spustí príkaz **BEDIT**. Pri niektorých objektoch, ako napr. šrafa alebo kružnica, sa spustí príkaz **QPROPERTIES**, ktorý zobrazí paletu Quick Properties, pozri časť 2.12.

Tip

Akcie dvojklikom sú ovládané systémovou premennou **DBLCLKEDIT**. Pokiaľ je premenná nastavená na **0**, všetky akcie dvojklikom sú neprístupné. Pri hodnote **1** sú akcie dvojklikom aktívne.

Upravovať priradené akcie, rovnako ako aj pridávať nové akcie môžeme v dialógovom okne Customize User Interface vo vlákne Double Click Actions. Úprava existujúcich akcií je veľmi jednoduchá, pretože stačí priradiť vybraný príkaz:

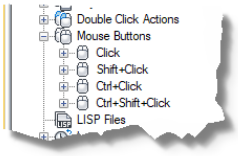
- „ťahaním" z podokna Command List alebo v rámci podokna Customization in All Files, alebo
- „kopírovaním" pomocou volieb v kontextovom menu po stlačení alebo pomocou ctrl + C a ctrl + V. Týmto spôsobom však nevytvárame kópie príkazov, len ich umiestňujeme v rámci užívateľského prostredia.

Novú akciu vytvoríme pomocou voľby New Double Click Action v kontextovom menu po stlačení kdekoľvek na vlákne Double Click Actions. Okrem priradenia samotného príkazu, kedy postupujeme podľa horeuvedených možností, musíme určiť aj názov akcie a najmä určiť typ objektu, ktorým budeme túto akciu spúšťať. Typ objektu je určený vlastnosťou Object Name, ktorú môžeme určiť v podokne Properties.

2.2.1.9 Mouse Buttons

Vo vlákne Mouse Buttons je možné priradiť príkazy tlačidlám myši. Počet tlačidiel myši, ktoré môžeme upravovať nie je v AutoCADe limitovaný. Jediné obmedzenie sa týka , ktoré nemôžeme nijako upravovať. Ostatné tlačidlá vieme upravovať pre štyri rôzne prípady, pozri obr. 2.15:

- vo vlákne Mouse Buttons > Click upravujeme kliknutie tlačidlom,
- vo vlákne Mouse Buttons > Shift+Click upravujeme kliknutie tlačidlom za súčasného držania shift,
- vo vlákne Mouse Buttons > Ctrl+Click upravujeme kliknutie tlačidlom za súčasného držania ctrl ,
- vo vlákne Mouse Buttons > Ctrl+Shift+Click upravujeme kliknutie tlačidlom za súčasného držania shift a ctrl.



Obr. 2.15: Rôzne kombinácie klávesov a myši, ktorým v dialógovom okne Customize User Interface možno priradiť príkazy

Priradiť alebo aj nahradiť priradené príkazy pre jednotlivé tlačidlá môžeme:

- „ťahaním" z podokna Command List alebo v rámci podokna Customization in All Files, alebo
- „kopírovaním" pomocou volieb v kontextovom menu po stlačení alebo pomocou ctrl+C a ctrl+V. Týmto spôsobom však nevytvárame kópie príkazov, len ich umiestňujeme v rámci užívateľského prostredia.

Poradie jednotlivých tlačidiel môžeme meniť jednoducho presunutím vybraného prvku vo vlákne pomocou myši. Po kliknutí na konkrétny prvok vo vlákne môžeme:

- zrušiť priradený príkaz pre konkrétne tlačidlo myši voľbou Clear Assignment,
- vytvoriť nové tlačidlo myši vo vlákne voľbou New Button, prípadne duplikovať voľbou Duplicate,
- tlačidlo myši úplne odstrániť z vlákna voľbou Delete.

Tip

Na obr. 2.15 ste si mohli všimnúť, že predvolene je pre stlačenie kolieska myši (Button 3) priradené makro pre uchopovanie. To je však v rozpore s tým, ako sa AutoCAD správa. Stlačením kolieska myši totiž transparentne spustíme príkaz **PAN**. Nejde o chybu v programe, AutoCAD sa prioritne riadi hodnotou systémovej premennej **MBUTTONPAN**. Pokiaľ je hodnota tejto premennej **1**, stlačeným kolieskom myši sa vieme presúvať vo výkrese, pretože AutoCAD ignoruje nastavenie tlačidla v dialógovom okne Customize User Interface. Ak má premenná hodnotu **0**, po stlačení kolieska myši sa vykoná akcia určená v dialógovom okne Customize User Interface.

2.2.1.10 LISP Files

V tomto vlákne môžeme pridať cesty k `*.lsp` súborom, ktoré sa načítajú vždy pri spustení AutoCADu (nie vo verzii LT) a budú prístupné pre všetky otvorené dokumenty `*.dwg`. V prípade potreby je možné vybrané cesty k súborom `*.lsp` aj odstrániť z tohoto zoznamu. Samotné pridávanie a odoberanie ciest k súborom `*.lsp` je možné po kliknutí na vlákno LISP Files a výbere možnosti Load LISP, resp. Unload.

Viac o načítavaní `*.lsp` súboroch uvádzame v úvode kapitoly 4.

2.2.1.11 Partial Customization Files

Súbory `*.cuix` v tomto vlákne môžu dopĺňať hlavný súbor `acad.cuix` v každom vlákne (okrem vlákna Legacy a Partial Customization Files). To znamená, že všetky prvky z čiastkových `*.cuix` súborov sa spoja s hlavným `*.cuix` súborom a vznikne výsledné užívateľské prostredie, ktoré sa načítajú pri spustení AutoCADu.

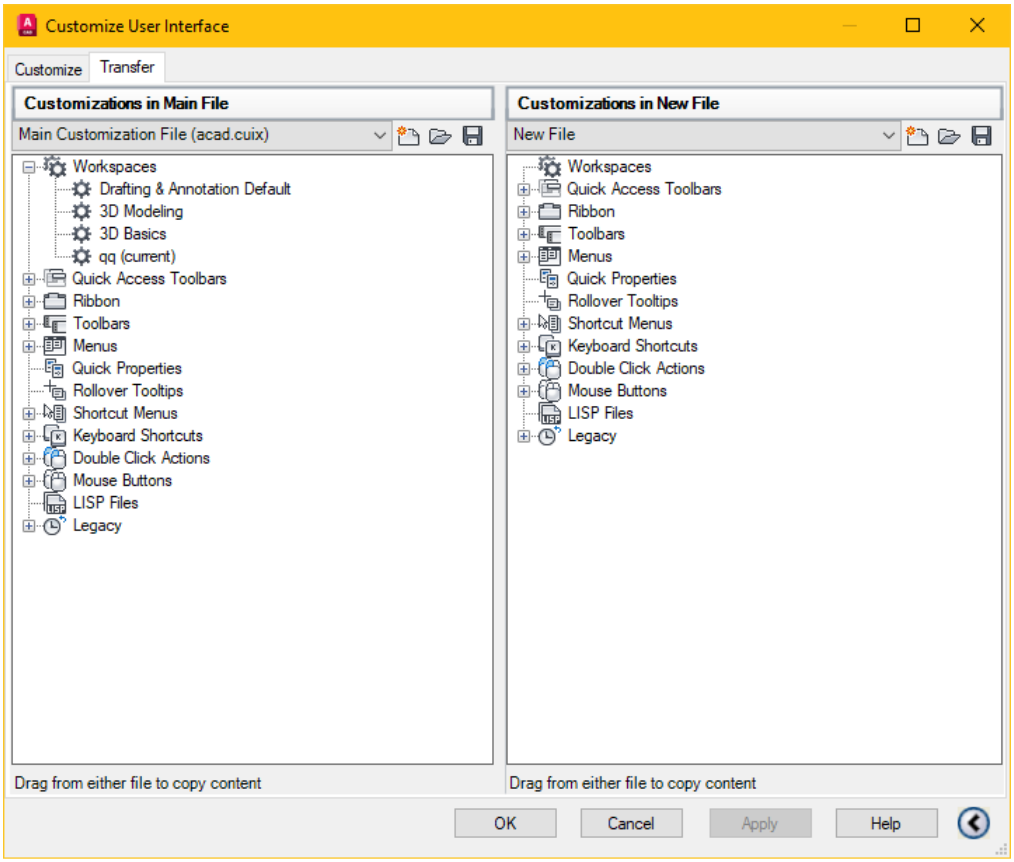
Cesty k čiastkovým `*.cuix` súborom môžeme do vlákna Partial Customization Files pridať kliknutím na toto vlákno a voľbou Load partial customization file. Naopak, odstrániť cesty k týmto súborom môžeme kliknutím na vybranú cestu a voľbou Unload [*.CUIX]

Malou nevýhodou čiastkových `*.cuix` súborov je to, že pracovný priestor definovaný v čiastkovom súbore nie je automaticky zlúčený s ostatnými prvkami. Teda ak chceme pracovný priestor z čiastkového `*.cuix` súboru používať, musíme ho presunúť do hlavného `*.cuix` súboru pomocou karty Transfer v dialógovom okne Customize User Interface, pozri časť 2.2.2.

Pozor, ak chceme upravovať prvky z čiastkových `*.cuix` súborov, treba v rozbaľovacom zozname v hornej časti podokna Customizations in Main File vybrať konkrétny súbor.

2.2.2 Prenos prvkov medzi súbormi `*.cuix`

Na export alebo import jednotlivých prvkov z alebo do `*.cuix` súborov môžeme použiť kartu Transfer v dialógovom okne Customize User Interface, pozri obr. 2.16. Dialógové okno je na tejto karte rozdelené na dve podokná, v každom podokne je zobrazený obsah jedného `*.cuix` súboru. Prvky prenášame „ťahaním" vždy zo súboru v ľavom podokne do súboru v pravom podokne.



Obr. 2.16: Obsah karty Transfer v dialógovom okne Customize User Interface

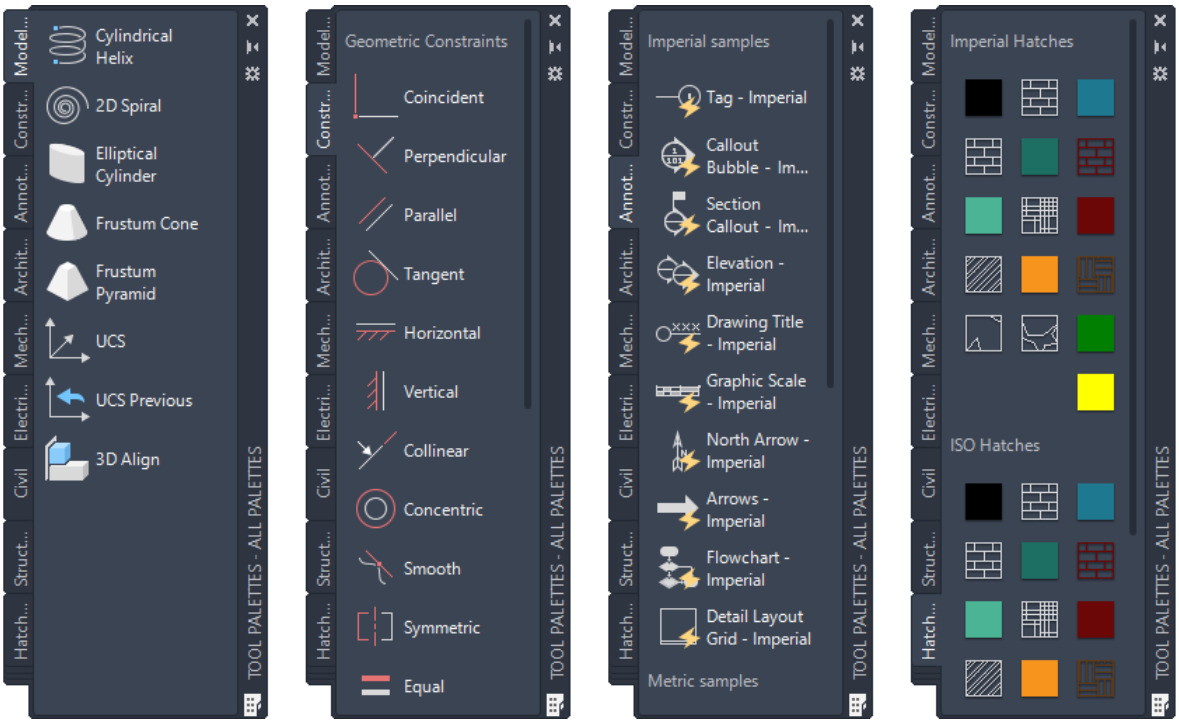
Na správu súborov môžeme použiť rozbaľovací zoznam v hornej časti alebo tlačidlá na vytvorenie nového súboru (zvyčajne na export), otvorenie existujúceho súboru a na uloženie zmien v súbore.

Tip

Na export `*.cuix` súborov môžeme využiť aj príkaz **CUIEXPORT**, ktorým otvoríme dialógové okno Customize User Interface v rozložení pripravenom na export, teda v ľavom podokne je otvorený existujúci súbor a v pravom nový. Podobne na import môžeme použiť príkaz **CUIIMPORT**.

2.3 Palety nástrojov

Palety nástrojov sú veľmi šikovnou alternatívou pre pás kariet (Ribbon). Palety môžu obsahovať nielen tlačidlá príkazov, podobne ako je to na páse kariet, ale napríklad aj konkrétne bloky alebo šrafy. Na rozdiel od pásu kariet je však každé tlačidlo definované samostatne (nie je naviazané na definíciu tlačidla z dialógového okna Customize User Interface). Naopak, každé tlačidlo je ľahko prispôsobiteľné, a preto tlačidlá na palete nástrojov označujeme ako nástroje. Používanie palety nástrojov je vhodnou voľbou najmä pre svoju praktickosť pri používaní a jednoduchosť či už pri tvorbe paliet, úprave ale aj pri ich prenose.



Obr. 2.17: Ukážky rôznych paliet nástrojov

Paletu nástrojov Tool Palettes zobrazíme pomocou príkazu **TOOLPALETTES**, prípadne pomocou tlačidla Manage › Customization › Tool Palettes. Na každej palete (karte palety nástrojov) môžeme mať niekoľko rôznych nástrojov v rôznej forme, pozri obr. 2.17. Nástroj palety môžeme použiť:

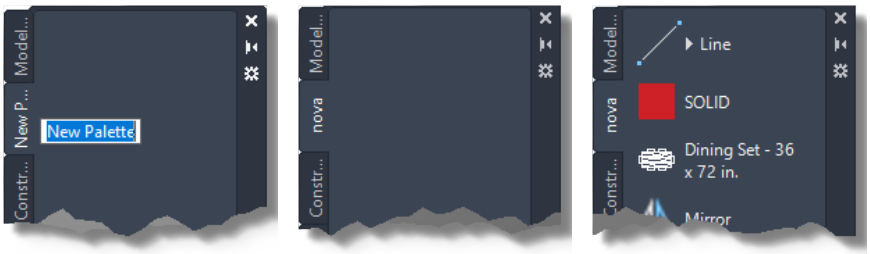
- po kliknutí na nástroj a následne postupom podľa príkazového riadku, alebo
- pri niektorých nástrojoch, napr. šrafa alebo vkladanie referencie bloku, je možné nástroj uchopiť pomocou a následne nástroj uvoľniť na mieste, kde ho chceme použiť.

Okrem množstva preddefinovaných paliet je možné vytvoriť vlastné palety s vlastným obsahom (nástrojmi). Tvorbe a úprave vlastnej palety sa venujeme v časti 2.3.1. Okrem toho sa palety na palete nástrojov dajú organizovať do skupín, zvyčajne podľa použitia jednotlivých paliet. Samostatné palety ako aj skupiny paliet je možné aj prenášať medzi rôznymi inštaláciami AutoCADu. Palety nástrojov sú totiž uložené do samostatných súborov formátu `*.xtp`, resp. skupiny paliet vo formáte `*.xpg`. Spravovaniu a prenosu paliet a skupín paliet sa viac venujeme v časti 2.3.2.

2.3.1 Tvorba a úprava paliet na palete nástrojov

Novú paletu môžeme vytvoriť po kliknutí kdekoľvek na palete nástrojov a výbere voľby New Palette v kontextovom menu. Následne sa na palete nástrojov vytvorí nová paleta s preddefinovaným názvom, ktorý môžeme hneď zmeniť, pozri obr. 2.18 vľavo a v strede. Do novovytvorenej palety môžeme postupne pridávať nástroje, pozri obr. 2.18 vpravo, niektorým z nasledujúcich spôsobov:

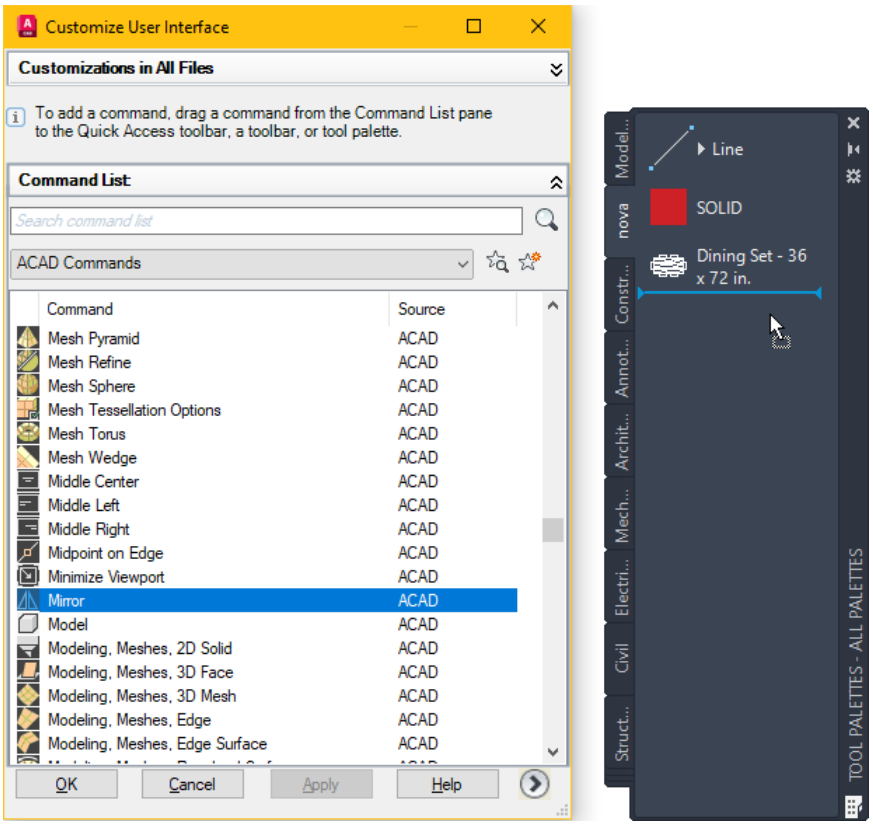
- kliknutím na vybraný objekt a jeho „ťahaním" z grafického okna AutoCADu na paletu, alebo
- kopírovaním (ctrl + C) vybraného objektu v grafickom okne a jeho prilepením (ctrl + V) na palete, alebo
- „ťahaním" vybraného príkazu z podokna Command List dialógového okna Customize User Interface na paletu, pozri obr. 2.19.



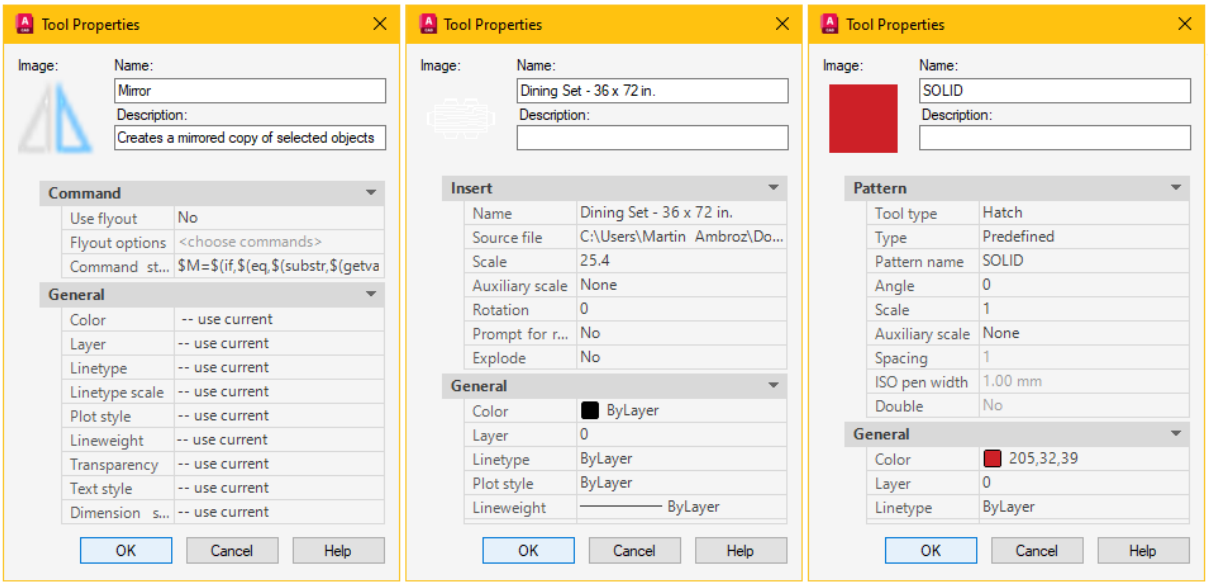
Obr. 2.18: Tvorba novej palety

Dialógové okno Customize User Interface je nevyhnutné použiť len pri niektorých príkazoch, napr. pri editačných príkazoch. Veľká výhoda pri vytváraní obsahu palety je, že grafické objekty do nej stačí jednoducho „naťahať". Dôležitou skutočnosťou je to, že nástroj vytvorený „ťahaním" si zapamätá okrem samotného príkazu, potrebného na vytvorenie objektu aj jeho vlastnosti, ako napr. farba, hladina, typ čiary, textový štýl a podobne. A rovnaké vlastnosti bude mať objekt vytváraný týmto nástrojom. Ak máme napr. úsečku vytvorenú s farbou zmenenou na červenú, nástroj vytvorený „ťahaním" z takejto úsečky si túto vlastnosť zapamätá. To znamená, že pri použití tohto nástroja budeme vytvárať vždy červené úsečky. Ak ale na paletu umiestnime príkaz **LINE** z dialógového okna Customize User Interface, všetky vlastnosti budú nastavené na - - use current, teda pri použití nástroja sa použijú aktuálne nastavenia AutoCADu.

Na úpravu vlastností nástrojov využívame dialógové okno Tool Properties, (obr. 2.20), ktoré spustíme po kliknutí na konkrétny nástroj na palete a voľbe Properties... v kontextovom menu. Zaujímavosťou je, že toto dialógové okno zobrazuje pri väčšine nástrojov rovnaké vlastnosti (obr. 2.20 vľavo), ale pri niektorých nástrojoch vytvorených „ťahaním" sa zobrazia vlastnosti v závislosti na tom, z akého typu objektu bol nástroj vytvorený. Ako príklad uvádzame nástroj na vkladanie konkrétneho bloku (obr. 2.20 v strede) a nástroj na šrafovanie konkrétnym šrafovacím vzorom (obr. 2.20 vpravo).



Obr. 2.19: Tvorba nástroja „ťahaním" príkazu z dialógového okna Customize User Interface na paletu



Obr. 2.20: Dialógové okno Tool Properties

V nasledujúcich riadkoch popíšeme niekoľko vybraných nástrojov a ich vlastnosti. Postupne popíšeme príkazy, kresliace a kótovacie príkazy, šrafovacie vzory a nakoniec aj definície blokov.

2.3.1.1 Príkazové nástroje

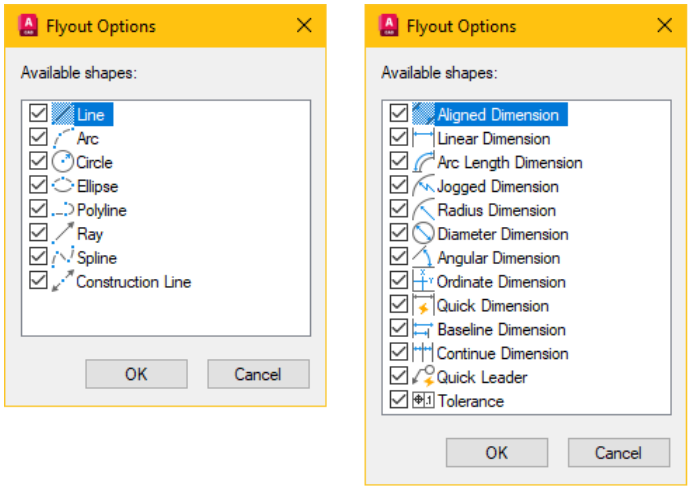
Ľubovoľný príkaz vieme na paletu pridať z podokna Command List dialógového okna Customize User Interface. Toto dialógové okno môžeme otvoriť akýmkoľvek spôsobom, ale ak máme otvorenú paletu nástrojov, máme navyše možnosť kliknúť pod nástroje na palete a vybrať voľbu Customize Commands... Pridaním na paletu sa z príkazu stáva nástroj, a teda, pokiaľ zmeníme nejakú vlastnosť tohto nástroja, na pôvodný príkaz z dialógového okna Customize User Interface to nemá už žiadny vplyv. Nástroj si teda z príkazu prevzal názov a makro príkazu, ktoré môžeme v prípade upravovať cez vlastnosti nástroja, pozri obr. 2.20. Pre viac informácií o makre príkazu, pozri časť 3.2.

2.3.1.2 Kresliace a kótovacie nástroje

Pod označením kresliace príkazy máme na mysli napríklad príkazy **LINE**, **ARC**, **PLINE** alebo **SPLINE**, a pod označením kótovacie príkazy napríklad **DIMLINEAR**, **DIMALIGNED**, **DIMANGULAR** alebo **QDIM**. Týmto dvom skupinám príkazom venujeme samostatnú pozornosť, pretože môžu byť zlúčené do jedného rozbaľovacieho nástroja. To, či sú zlúčené určuje vlastnosť Use flyout v dialógovom okne Tool properties, pozri obr. 2.20. Zoznam všetkých zlúčených nástrojov je určený prostredníctvom vlastnosti Flyout options. Tento zoznam upravujeme ne/začiarknutím jednotlivých príkazov v dialógovom okne Flyout Options, pozri obr. 2.21, ktoré otvoríme po kliknutí na tlačidlo pri vlastnosti nástroja Flyout options v dialógovom okne Tool Properties.

Tip

Fixné nastavenie vlastností ako typ čiary, resp. hladina s nastaveným typom čiary môže veľmi urýchliť prácu. Namiesto manuálnej zmeny hladiny a spusteniu jedného z kresliacich príkazov stačí kliknúť na správny kresliaci nástroj, ktorým automaticky vytvoríme nové objekty v ňom určenej hladine.



Obr. 2.21: Rôzne možnosti tzv. Flyout nástrojov palety, ktoré združujú v jednom nástroji viacero príkazov

2.3.1.3 Šrafovacie nástroje

Šrafovacie nástroje majú špecifické vlastnosti, pozri obr. 2.20 vpravo. Okrem iných vlastností, ktoré poznáme z klasického šrafovania pomocou príkazu **HATCH** dávame do pozornosti vlastnosti:

- Type, v ktorej je určený typ šrafovacieho vzoru (preddefinovaný alebo vlastný) a
- Pattern Name, ktorá určuje konkrétny vzor.

Všetky vlastnosti nástroja sa síce prevezmú z objektu, ktorým ho vytvárame. Avšak je potrebné si uvedomiť, najmä v prípade vlastných šrafovacích vzorov, že AutoCAD stále potrebuje mať prístup k súborom, v ktorých sú definované. Tieto súbory teda musia byť v jednom z priečinkov patriacich do Support File Search Path na karte Files v dialógovom okne Options (ktoré spustíme príkazom **OPTIONS**). Pri prenose paliet (pozri časť 2.3.2 s nástrojmi na šrafovanie, netreba zabudnúť aj na prenos súboru so šrafovacím vzorom (obzvlášť ak ide o vzor, ktorý nie je štandardnou súčasťou AutoCADu.

Tip

Medzi vlastnosti šrafovacieho nástroja nenájdeme vlastnosť Annotative, ktorou by sme mohli previazať objekt s vybranými mierkami. Pri vytváraní objektu o tom rozhoduje hodnota systémovej premennej **HPANNOTATIVE**. Ak je jej hodnota **1**, objekt s aktuálnou mierkou previaže. Pri hodnote **0** sa objekt vytvorí s vypnutou vlastnosťou Annotative.

Tip

Ak je vo vlastnostiach šrafovacieho nástroja určená konkrétna hladina, t. j. nie - - use current, užívateľ očakáva, že šrafa sa pri použití tohto nástroja vytvorí v tejto hladine. Toto nanešťastie neplatí vždy. Dominantné slovo má v tomto prípade systémová premenná **HPLAYER**. Ak je nastavená na **"USE CURRENT"**, nástroj vytvorí šrafu v ňom určenej hladine. Ak je ale v systémovej premennej **HPLAYER** určené akákoľvek konkrétna hladina, všetky šrafy sa vytvoria v nej. Túto systémovú premennú vieme nastaviť aj na kontextovej karte po spustení príkazu **HATCH** v rozbaľovacom zozname po rozbalení panelu Hatch Creation › Properties alebo pri editácii objektu šrafy po rozbalení panelu Hatch Editor › Properties.

2.3.1.4 Nástroje pre vkladanie blokov

Podobne ako pri šrafovacích nástrojoch, aj nástroje pre vkladanie blokov majú špecifické vlastnosti, pozri obr. 2.20 v strede. Vkladanie referencií blokov z palety nástrojov je podobné vkladaniu pomocou karty Favorites na palete Blocks. Podstatný rozdiel je však v prístupe k zdrojovým súborom, v ktorých sú bloky definované. Kým na palete Blocks sa o všetko postará AutoCAD, na palete nástrojov si musíme dať pozor na dostupnosť súborov sami. Vo vlastnostiach nástrojov je cesta súboru určená vlastnosťou Source file a názov bloku v zdrojovom súbore je uložený vo vlastnosti Name. Pri prenose paliet (pozri časť 2.3.2 s nástrojmi na vkladanie blokov, netreba zabudnúť aj na prenos súboru s definíciou bloku.

Tip

Pri tvorení nástrojov pre vkladanie blokov môžeme využiť aj paletu DesignCenter, z ktorej potrebné bloky na paletu nástrojov jednoducho „naťaháme".

2.3.2 Spravovanie a prenos paliet a skupín paliet na palete nástrojov

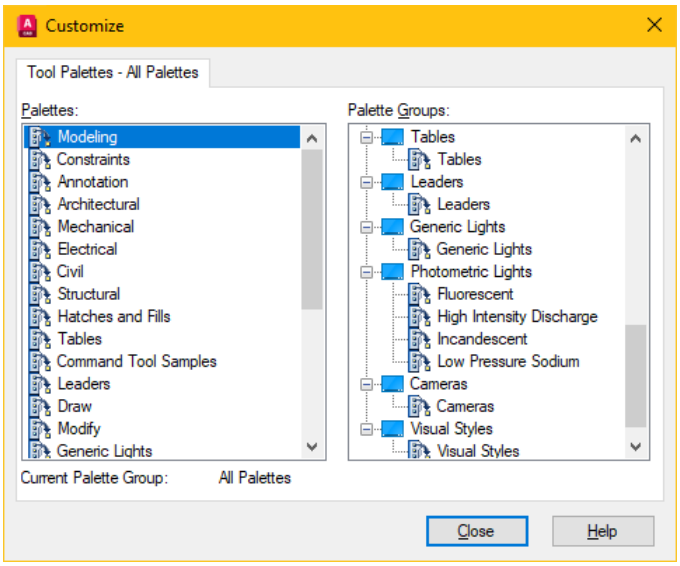
Palety z palety nástrojov môžeme organizovať do skupín. Na palete nástrojov sa predvolene zobrazujú všetky palety (vidíme to aj podľa doplnku v názve palety Tool Palettes - All Palettes). Ak je všetkých paliet príliš veľa, môže byť vhodné zobraziť iba niektoré palety, ktoré sú spoločne zoskupené, a to výberom konkrétnej skupiny v spodnej časti kontextového menu, ktoré vyvoláme:

- tlačidlom v hornej časti palety, alebo
- kliknutím na okraji palety nástrojov.

Zaraďovanie paliet do skupín, ako aj správu skupín zabezpečuje dialógové okno Customize, ktoré sa zobrazí po spustení príkazu **CUSTOMIZE**. Toto dialógové okno, pozri obr. 2.22, je rozdelené na dve časti:

- Palettes, kde je zoznam všetkých paliet z palety nástrojov a
- Palette Groups, kde je formou stromu zobrazený zoznam všetkých skupín paliet aj s priradenými paletami.

Palety môžeme do skupiny priradiť jednoducho potiahnutím z časti Palettes. Takýmto spôsobom môžeme jednu paletu priradiť aj do viacerých skupín. Okrem tohto spôsobu môžeme palety presúvať medzi skupinami v časti Palette Groups.



Obr. 2.22: Dialógové okno Customize na správu paliet v rámci palety nástrojov a ich import a export

Odstániť paletu z palety nástrojov môžeme po kliknutí na konkrétnu paletu v časti Palettes a výberom voľby Delete, resp. tlačidlom del. Toto odstránenie je trvalé. Odstrániť priradenie palety do skupiny môžeme po kliknutí na konkrétnu paletu v časti Palette Groups a výberom voľby Remove, resp. tlačidlom del. Týmto odstránením dôjde iba k odstráneniu zo skupiny, ale paleta samotná ostane medzi všetkými paletami v časti Palettes.

Vytvorené palety a skupiny paliet môžeme prenášať, či už medzi rôznymi verziami AutoCADu alebo medzi rôznymi zariadeniami. Palety prenášame vo forme súborov `*.xtp`, skupiny paliet vo formáte `*.xpg`. Tieto súbory vieme importovať alebo exportovať v dialógovom okne Customize, ktoré sa zobrazí po spustení príkazu **CUSTOMIZE**, pozri obr. 2.22. Exportovať paletu, resp. skupinu paliet môžeme po kliknutí na konkrétnu paletu v časti Palettes, resp. skupinu paliet v časti Palette Groups a výberom voľby Export...

V nasledujúcom dialógovom okne vyberieme umiestnenie a názov exportovaného súboru. Veľmi podobne prebieha aj import paliet, resp. skupín paliet. Pri importe však môžeme kliknúť na ľubovoľnú paletu v časti Palettes, resp. skupinu paliet v časti Palette Groups a výberom voľby Import... . Následne vyberieme súbor, ktorý chceme importovať.

Tip

Pri prenose skupiny paliet treba dať pozor na to, aby sme preniesli aj samotné palety. Pokiaľ totiž prenesieme súbor skupiny paliet vo formáte `*.xpg`, obsahujúci palety, ktoré nie sú definované, skupina bude prázdna. Ak následne prenesieme aj samotné palety, automaticky sa zobrazia aj v skupine.

2.4 Aliasy príkazov

Príkazy v AutoCADe môžeme spustiť mnohými spôsobmi. Medzi najrýchlejšie spôsoby patrí zadanie a potvrdenie príkazu do príkazového riadku. Okrem samotného príkazu však môžme do príkazového riadku zadať aj jeho alias. Alias môže byť skratka príkazu, napríklad:

- **L** je aliasom príkazu **LINE**,
- **C** je aliasom príkazu **CIRCLE**,
- **CO** je aliasom príkazu **COPY**,

ale nie je to pravidlom. Napríklad aliasom príkazu **LWEIGHT** je **LINEWEIGHT**.

Aliasy príkazov sú uložené v súbore parametrov programu vo formáte `*.pgp`. Ide o textový súbor, v ktorom sú uložené dvojice alias a príkaz v predpísanej forme, pozri obr. 2.23. Navyše platí, že príkaz môže mať viac aliasov, napr. **LINETYPE** má alias **LT** aj **LTYPE**, ale každý alias môže byť použitý iba raz.

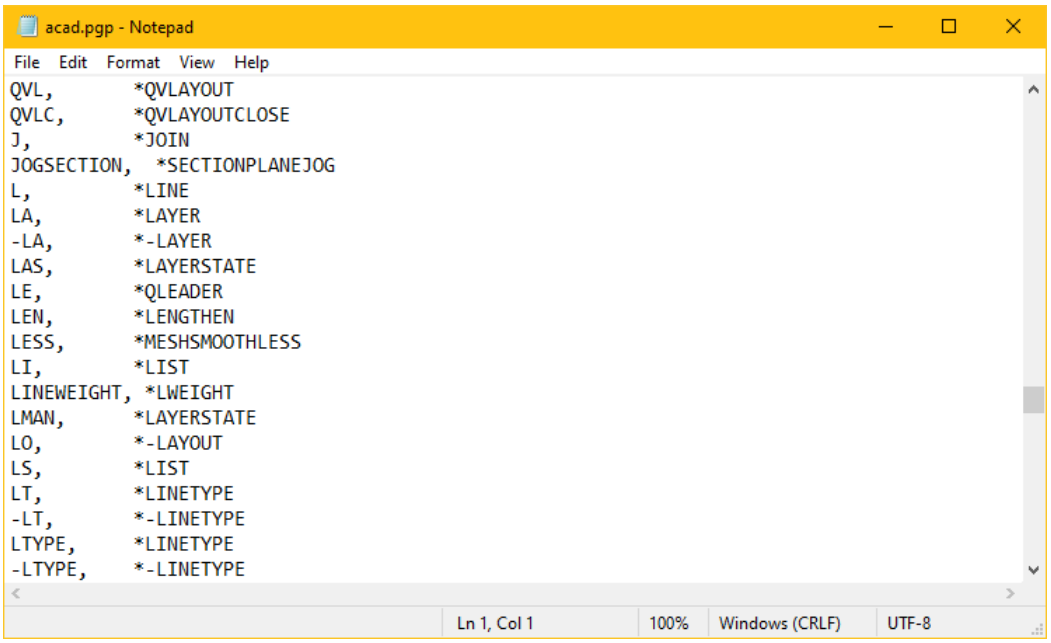
Tip

Ak by bol alias priradený viacerým príkazom, platiť bude iba posledné priradenie. Teda ak by sme na koniec súboru `acad.pgp` pridali alias **L** pre príkaz **CIRCLE**, zrušili by sme tým skôr uvedený alias pre príkaz **LINE**.

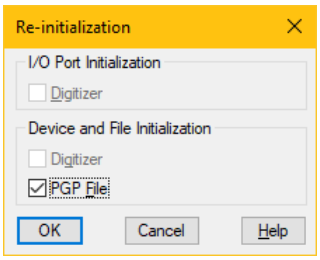
K dispozícii máme hneď tri súbory v tomto formáte:

- `acad.pgp` – databáza aliasov príkazov,
- `AutoCorrectUserDB.pgp` – databáza častých preklepov pri zadávaní príkazov, generuje sa automaticky. Táto databáza sa vytvára v závislosti od voľby v dialógovom okne Input Search Options (zobrazíme príkazom **INPUTSEARCHOPTIONS**). Rovnako sa tu dá nastaviť počet preklepov (predvolená hodnota je 3), po ktorých sa má vytvoriť záznam do tejto databázy,
- `acadSynonymsGlobalDB.pgp` – databáza synonym príkazov.

Všetky tri uvedené súbory sú editovateľné priamo cez textový editor. Hoci sa priamo v týchto súboroch môžeme dočítať, že ich všetky môžeme aj editovať, priamo v pomocníkovi AutoCADu [5] je odporúčané editovať iba súbor `acad.pgp`.



Obr. 2.23: Obsah súboru `acad.pgp`



Obr. 2.24: Dialógové okno pre nastavenie znovunačítania súboru `acad.pgp` po zmene jeho obsahu

Pristupovať k týmto súborom môžeme nasledovne:

- priamo v `C:/Users/[meno_pouzivatela]/AppData/Roaming/Autodesk/Autocad 2023/r24.2/enu/Support`,
- príkazom **AI_EDITCUSTFILE** a následne zadaním názvu súboru, ktorý chceme editovať, prípadne priamym výberom cez rozbaľovacie tlačidlo Manage › Customization › Edit Aliases,
- príkazom **ALIASEDIT** z Expres Tools › Tools › Command Aliases, čím otvoríme jednoduchý program na editáciu súboru `acad.pgp`.

Tip

Priečinok `AppData` je skrytý priečinok. Preto sa doň nevieme v prieskumníkovi „preklikať". Môžeme sa doň však dostať zadaním cesty do prieskumníka, alebo rýchlejšie pomocou skratky. Zadaním `%appdata%` do prieskumníka sa toti5 dostaneme ihneď do priečinka `AppData/Roaming`.

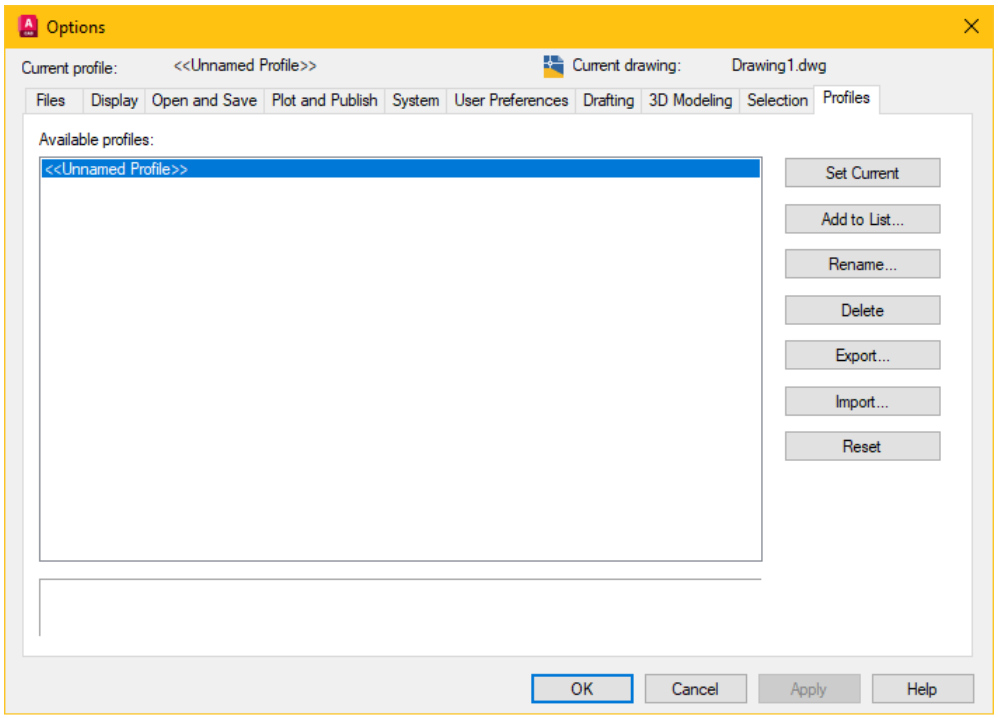
Aby sa zmeny v súbore `acad.pgp` prejavili v spustenom AutoCADe, teda aby sme mohli využívať nové aliasy, je nutné znovu načítať tento súbor pomocou príkazu **REINIT**. Po spustení príkazu sa zobrazí dialógové okno, v ktorom treba začiarknuť voľbu PGP File a potvrdiť tlačidlom OK, pozri obr. 2.24.

2.5 Profily

Na rozdiel od vyššie uvedených možností úprav užívateľského prostredia sa pri profiloch stretneme najmä s prispôsobeniami, ktoré nie sú na prvý pohľad viditeľné. V profile môžeme definovať napr. [5]:

- cesty pre vyhľadávanie,
- cesty k podporným súborom,
- cesty k súborom šablón,
- cesty k súborom typov čiar `*.lin` a šrafovcích vzorov `*.pat`,
- predvolené nastavenie tlačiarne,
- nastavenie zobrazenia palety nástrojov.

Profily vieme spravovať v dialógovom okne Options, ktoré zobrazíme pomocou príkazu **`OPTIONS`**. Toto dialógové okno umožňuje pomerne rozsiahlu úpravu správania sa AutoCADu, ktorej sa detailnejšie venovať nebudeme. V súvislosti s profilmi je ale dôležité poznamenať, že vybrané nastavenia z tohto dialógového okna sa ukladajú do profilov. Profily môžeme spravovať v tomto dialógovom okne na karte Profiles, pozri obr. 2.25. Na tejto karte môžeme aj meniť profily, a prenášať pomocou súborov vo formáte `*.arg`.



Obr. 2.25: Dialógové okno Options na karte Profiles

Tip

Do profilu sa neukladajú hodnoty všetkých systémových premenných. Hodnoty niektorých systémových premenných sa ukladajú nezávisle od profilu, iné sa neukladajú vôbec (ich hodnota sa uchováva iba počas spusteného AutoCADu.

3 Základné možnosti automatizácie

Na automatizácia procesov ponúka AutoCAD množstvo nástrojov. Pre neskúseného používateľa sú vhodnou voľbou možnosti, ktoré sú jednoduchšie na tvorbu. Na druhej strane však majú obmedzené schopnosti (v porovnaní s programovacími jazykmi, ako napr. AutoLISP, pozri kapitoly 4 až 10). Automatizovať a teda aj urýchliť procesy pozostávajúce spravidla len z malého počtu volaní príkazov (zvyčajne rádovo v jednotkách) môžeme pomocou:

- Action Recorder, pozri časť 3.1,
- Command Macro, pozri časť 3.2 a
- Skript, pozri časť 3.3.

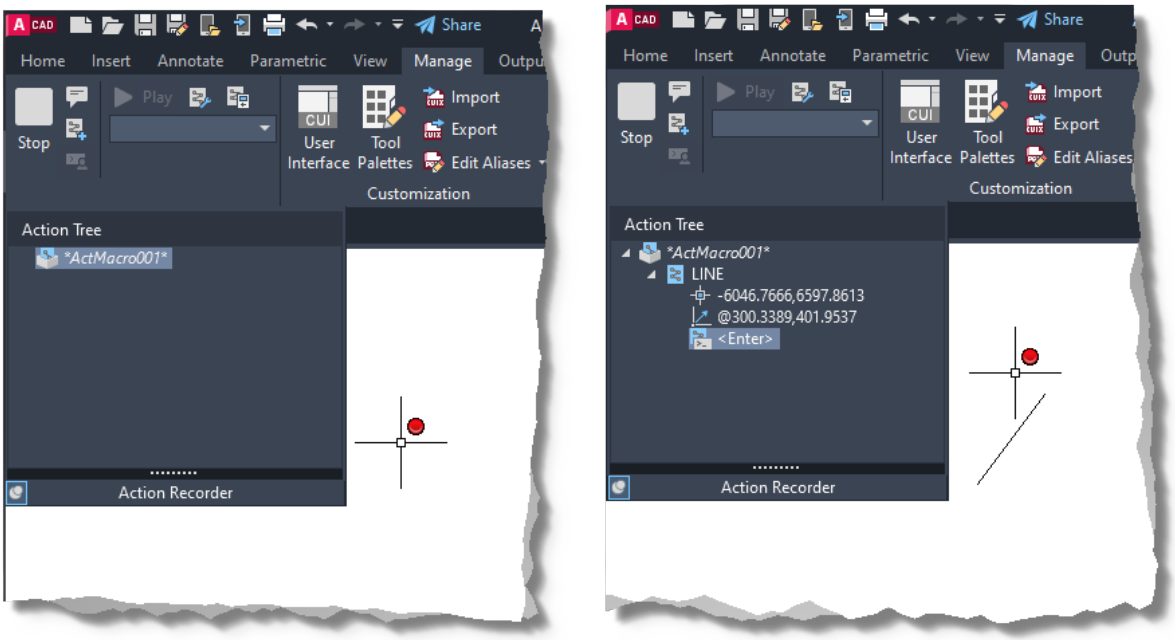
3.1 Action Recorder

Action Recorder je tou najprístupnejšou možnosťou automatizácie práce v AutoCADe a nájdeme ho na Manage › Action Recorder. Pomocou tejto funkcionality totiž dokážete zaznamenávať akcie užívateľa, ako napr. spustenie príkazu, voľba v príkaze, zadané hodnoty a pod. Tento záznam akcií vieme uložiť ako tzv. Action Macro a v budúcnosti opakovane používať a ušetriť tak množstvo času pri opakujúcich sa činnostiach. Každé takéto makro je uložené v súbore s koncovkou `*.actm`, pričom názov tohto súboru je zároveň aj názov príkazu, ktorým toto makro v AutoCADe spustíme.

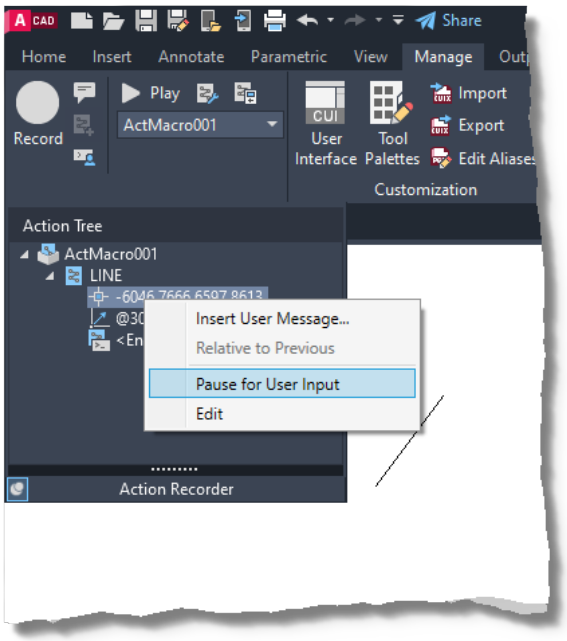
Vytvorenie makra je veľmi jednoduché. Stačí len vykonávať to, čo chcete, aby makro robilo za Vás. Nahrávanie makra spustíme príkazom **ACTRECORD**. Po spustení nahrávania sa štandardne rozbalí panel Manage › Action Recorder, v ktorom budeme vidieť postupnosť zaznamenaných akcií, pozri obr. 3.1. Po vykonaní všetkých akcií, ktoré chceme mať v makre nahrávanie zastavíme príkazom **ACTSTOP** a nakoniec je ešte potrebné zadať názov makra (prípadne prijať dočasný názov v tvare ActMacro###, kde ### je poradové číslo makra.

Vytvorené makro však môžeme ešte ďalej upravovať. Majme napríklad makro **CIARA**, ktoré spustí príkaz **LINE**, zadáme prvý a následne aj druhý bod úsečky a na záver stlačíme enter, pozri obr. 3.1 . Takýmto makrom by sme dokázali tvoriť úsečku iba na tých istých súradniciach. Ak ale klikneme na akciu zadávania prvého bodu (pozri obr. 3.2) a zaškrtneme možnosť Pause for User Input, po spustení makra nás AutoCAD vyzve na zadanie prvého bodu, z ktorého čiaru vykreslí. Štandardne sa pozícia druhého bodu určuje relatívne, teda voči predchádzajúcemu bodu. To znamená, že takýmto makrom budeme kresliť rovnako dlhé čiary z bodov, na ktoré klikneme. Zaisťuje to voľba Relative to Previous v kontextovom okne po kliknutí na vybranú položku v makre. Ak by sme túto voľbu pri akcii zadávania druhého bodu v makre nemali zaškrtnutú, jeho pozícia by sa na predchádzajúci bod nevzťahovala a bola by fixná.

Pri vytváraní makra sa môže stať, že sa pomýlite, či už v zadanej hodnote, alebo spustíte omylom iný príkaz. Fixne dané hodnoty v makrách vieme meniť voľbou Edit, a to či už súradnice, ale aj číselné



Obr. 3.1: Action Recorder na začiatku (vľavo) a na konci (vpravo) nahrávania makra



Obr. 3.2: Úprava makra v strome akcií, konkrétne vloženie pauzy pre užívateľský vstup

hodnoty. Nepotrebné vstupy do príkazov, alebo nepotrebné príkazy vieme z makra dodatočne odstrániť voľbou Delete. Okrem toho je možné kdekoľvek vložiť správu užívateľovi.

Nevýhodou takýchto makier je obmedzené použitie, v podstate len na príkazy v príkazovom riadku. Action Recorder totiž nezachytí akcie mimo AutoCADu a dokonca ani v dialógových oknách AutoCADu. Preto ak chceme v makre zaznamenať prácu s hladinami, textovým štýlom, blokmi a podobne, musíme použiť príkazy vo verzii pre príkazový riadok. Takýto príkaz spustíme pomocou pomlčky pred názvom štandardného príkazu, teda napr. **-LAYER**, **-STYLE** alebo **-BLOCK**. Pri nahrávaní makier sa vo všeobecnosti odporúča spúšťať príkazy z príkazového riadku a nie napr. z pásu kariet (paleta Ribbon). Ak chceme v makre upravovať systémovú premennú, odporúča sa použiť na to príkaz **SETVAR**.

Tip

Pri zadávaní hodnôt, napr. výšky textu, AutoCAD ponúka možnosť stlačením ↲ enter potvrdiť prednastavenú hodnotu. Pokiaľ však v makre akceptujeme prednastavenú hodnotu, môže to viesť k neželanému výsledku. Prednastavená hodnota totiž nie je konštantná. Preto ak chceme zaistiť v makre nemenné hodnoty, treba vždy hodnotu do makra zadať.

Výhodou Action Recorderu je jednoduchosť, pre tvorbu makra nie je potrebné ovládať ani len základné princípy programovania. Takto vytvorené makrá sú prístupné vo všetkých výkresoch a v každom ďalšom spustení AutoCADu. Navyše je makro ľahko prenosné na iný počítač, keďže je uložené v `*.actm` súbore, štandardne v `C:/Users/[meno_pouzivatela]/AppData/Roaming/Autodesk/AutoCAD 2023/R24.2/enu/support/Actions`.

3.2 Command macro

Command macro (ďalej len makro) ponúka podstatne širšie možnosti pri automatizácii v porovnaní s využitím nástroja Action Recorder, pozri 3.1. Na druhú stranu, makro musí užívateľ napísať, teda musí poznať názvy príkazov, postupnosť krokov v príkazoch a ovládať špeciálne znaky makra, pozri tabuľku 3.1. Vytvorené makro môžeme priradiť tlačidlu na palete Ribbon (pozri časť 2.2.1.3), alebo nástroju na palete nástrojov (pozri časť 2.3).

Makro väčšiny tlačidiel vyzerá veľmi jednoducho, napr. tlačidlo pre spustenie príkazu **LINE** má priradené makro 3.1, kde ^C je kód pre kláves esc. Tým ukončíme príkaz (ak je nejaký aktívny). Kód ^C voláme ešte raz, a to pre prípad, že v aktívnom príkaze sa nachádzame v nejakej podvoľbe, z ktorej rovnako potrebujeme odísť pred spustením nového príkazu. Po ukončení príkazu môžeme spustiť nový príkaz. Príkazy spúšťame s predponou _, ktorá zaistí funkčnosť v každej jazykovej mutácii AutoCADu. Na záver makra je ešte znak medzery, ktorý plní funkciu klávesu ↲ enter.

Makro 3.1: Makro tlačidla Home › Draw › Line

```
^C^C_LINE
```

Tip

Pred názvom príkazu je dobrým zvykom používať okrem znaku _ aj znak ., ktorým zabezpečíme, že spustíme originálny príkaz AutoCADu. Príkazy sa totiž dajú aj upravovať, resp. nahradiť pomocou funkcie **defun** jazyka AutoLISP, pozri časť 6.3.5.

Tabuľka 3.1: Prehľad vybraných špeciálnych znakov makra

Znak	Popis
;	stlačí kláves enter
^I	stlačí kláves tab
␣	znak pre medzeru (v príkazovom riadku plní funkciu klávesu enter)
\	vytvorí pauzu pre vstup od užívateľa
.	umožní spustiť štandardné príkazy aj keď bola odstránená ich definícia pomocou UNDEFINE
_	použitím pred anglickým názvom príkazu zabezpečí kompatibilitu makra naprieč jazykovými mutáciami AutoCADu
*	opakuje makro kým sa neukončí (napr. klávesom esc)
$	predchádza DIESEL výrazom (napr. podmienkam)
^B	zapne/vypne uchopovanie (Object snap)
^C	ukončuje aktívny príkaz alebo možnosť príkazu (kláves esc)
^R	zabezpečuje kompatibilitu starších makier aj v najnovších verziách)

Tip

Pri spúšťaní príkazov, ktoré otvárajú dialógové okno alebo paletu (napr. **LAYER** alebo **PURGE**) je potrebné v makre používať znak -, ktorým spustíme príkaz vo verzii pre príkazový riadok, teda bez dialógového okna. Takýto príkaz vieme na rozdiel od jeho verzie s dialógovým oknom ovládať pomocou makra.

Makro ale možno využiť aj na podstatne zložitejšie úlohy ako len spúšťanie príkazov. Makro môže obsahovať:

- špeciálne znaky makra, pozri tabuľku 3.1,
- príkazy a vstupy do príkazov, teda voľby príkazov, čísla, texty a podobne,
- výrazy v jazyku DIESEL (Direct Interpretively Evaluated String Expression Language),
- výrazy v jazyku AutoLISP (nie vo verzii AutoCAD LT), pozri kapitoly 4 až 10.

Postup tvorby makra je celkom jednoduchý, avšak mimoriadne náchylný na chyby. Najspoľahlivejším spôsobom ako napísať makro je postupne si prejsť všetky príkazy a ich vstupy, ktoré chceme do makra zahrnúť. Všetky tieto vstupy si môžeme niekam zapisovať, alebo ich nakoniec môžeme získať z príkazového riadku AutoCADu. Ak by sme chceli vytvoriť napríklad makro na vytvorenie jednotkovej kružnice v bode danom užívateľom, v príkazovom riadku uvidíme nasledovné:

```
Command: CIRCLE
Specify center point for circle or [3P/2P/Ttr (tan tan radius)]:
Specify radius of circle or [Diameter]: 1
Type a command
```

Teda spustíme príkaz **CIRCLE**, stlačíme enter, užívateľ vyberie bod kliknutím myši, zadáme hodnotu polomeru a potvrdíme pomocou enter. Treba ešte myslieť na to, že pred spustením príkazu **CIRCLE** je dobrým zvykom pridanie dvojnásobného stlačenie esc pre prípad, že treba ukončiť iný (dovtedy aktívny) príkaz, pozri makro 3.2.

Makro 3.2: Makro pre vytvorenie jednotkovej kružnice v mieste určenom užívateľom

```
^C^C_.CIRCLE;\1;
```

Príkazy AutoCADu sa postupne vyvíjajú, menia a menia aj niektoré svoje vstupy, alebo postupy. Kompatibilita makier v novších verziách je zabezpečená pomocou špeciálneho znaku ^R. Ako príklad môžeme uviesť príkaz **FILLET**, v ktorom pomocou makra 3.3 nastavíme polomer zaoblenia a následne zaoblíme hrany. Makrom 3.3 spustíme príkaz **FILLET** v správnej verzii (reťazcom ^R), nastavíme polomer zaoblenia (voľbou _Radius, resp. jej skratkou _R) na **0**. Tým sa makro ukončí a spustený a nastavený príkaz **FILLET** vyžaduje, aby užívateľ vybral prvý objekt na zaoblenie. Ak by sme ale znak ^R vynechali, príkaz by síce mohol nastaviť polomer zaoblenia správne, ale príkaz by sa ukončil pred výberom objektov na zaoblenie.

Makro 3.3: Makro pre spustenie príkazu **FILLET** a nastavenie polomeru zaoblenia

```
^C^C^R_.FILLET;_R;0;
```

DIESEL Použitím jazyka DIESEL sa z makier stáva pomerne silný nástroj, keďže ide o jednoduchý programovací jazyk s neveľa funkciami [7], medzi ktorými nechýba funkcia na získavanie hodnôt systémových premenných, alebo funkcia pre podmienečný výraz if.

Tip

Použitie jazyka DIESEL možno na prvý pohľad nedáva zmysel, keď môžeme použiť aj omnoho schopnejší AutoLISP, no vo verzii AutoCAD LT je DIESEL jediným podporovaným programovacím jazykom.

Ak v makre používame výraz v jazyku DIESEL, vždy mu musí predchádzať reťazec $M= a podobne pred každým volaním funkcie jazyka DIESEL musíme vložiť znak $. Jazyk DIESEL nemá k dispozícii mnoho príkazov, ich zoznam nájdeme napr. na [7].

V nasledujúcich príkladoch vytvoríme makrá, ktoré budú pracovať ako prepínače medzi dvomi stavmi. Na to použijeme funkcie if, = a eq. Vo funkcii if určíme podmienku a akcie pre prípad keď je podmienka splnená a pre prípad, keď splnená nebude. Funkcie = a eq využijeme pri tvorbe podmienky, pričom funkcia = porovnáva rovnosť čísel, funkcia eq porovnáva zhodnosť reťazcov znakov.

V makre 3.4 meníme systémovú premennú **CURSORSIZE**, ktorá určuje veľkosť kurzora (nitkového kríža) v percentách vo vzťahu k veľkosti . Makrom budeme prepínať medzi štandardnou hodnotou 5 a maximálnou hodnotou 100.

Makro 3.4: Makro na prepínanie veľkosti kurzora

```
^C^C$M=$(if,$(=,$(getvar,cursorsize),100),cursorsize;5,cursorsize;100)
```

Podobnú funkcionalitu prezentujeme v ďalších dvoch makrách, ktorými prepíname premenné medzi dvomi stavmi. V makre 3.5 meníme systémovú premennú **CANNOSCALE**, zodpovednú za aktuálnu mierku vlastnosti Annotative, a v makre 3.6 premennú **TEXTSTYLE**, v ktorej je uložený aktuálny textový štýl.

Makro 3.5: Makro na prepínanie mierky

```
^C^C$M=$(if,$(eq,$(getvar,CANNOSCALE),1:100),CANNOSCALE;1:50,CANNOSCALE;1:100)
```

Makro 3.6: Makro na prepínanie textového štýlu

```
^C^C$M=$(if,$(eq,$(getvar,TEXTSTYLE),Standard),TEXTSTYLE;Annotative,TEXTSTYLE;
    Standard)
```

Tip

DIESEL môžeme využiť aj pri výpise v stavovom riadku AutoCADu. Systémová premenná **MODEMACRO**, do ktorej môžeme uložiť textový reťazec, ktorý chceme zobraziť v stavovom riadku totiž dokáže vyhodnotiť aj výrazy jazyka DIESEL. V stavovom riadku AutoCADu teda môžeme vypysovať napr. aktuálny textový štýl Current Text Style is : $(getvar,textstyle), cestu k súboru Path to current file: $(getvar,dwgprefix) alebo jednotky výkresu Current drawing units: $(if,$(=,4,$(getvar,insunits)),"metric","non-metric").

Pri tvorbe makra je užitočným pomocníkom aj výpis v príkazovom riadku AutoCADu. Pre aktivovanie výpisu vyhodnocovania DIESEL funkcií je potrebné zmeniť systémovú premennú **MACROTRACE** na hodnotu **1**. Ako príklad uvádzame výpis po volaní makra 3.6:

```
Command: Eval: $(IF, $(eq,$(getvar,TEXTSTYLE),Standard), TEXTSTYLE
Annotative, TEXTSTYLE
Standard)
Eval: $(EQ, $(getvar,TEXTSTYLE), Standard)
Eval: $(GETVAR, TEXTSTYLE)
===> Standard
===> 1
===> TEXTSTYLE
Annotative
TEXTSTYLE
Enter new value for TEXTSTYLE <"Standard">: Annotative
Type a command
```

V prvých troch riadkoch je uvedený celý DIESEL výraz z makra 3.6. Tento výraz je v príkazovom riadku rozdelený na tri riadky, pretože sa znak ";" pri vyhodnocovaní makra vyhodnotil ako nový riadok. Štvrtý a piaty riadok sú vnorené DIESEL výrazy. Následne sa tieto výrazy vyhodnocujú (riadky začínajúce na ===>), pričom sa začína od toho najviac vnoreného výrazu. Teda výraz $(GETVAR, TEXTSTYLE) sa vyhodnotil ako Standard, výraz $(EQ, $(getvar,TEXTSTYLE), Standard) sa vyhodnotil ako 1, teda pravdivý a nakoniec sa celý výraz vyhodnotil na dva riadky ako TEXTSTYLE Annotative. V posledných dvoch riadkoch vidíme. že sa vyhodnotený výraz vykonal. Ak by sme spravili napríklad preklep v názve systémovej premennej vo funkcii getvar, vo výpise uvidíme chybové hlásenie:

```
Command: Eval: $(IF, $(eq,$(getvar,TEXTSTYLO),Standard), TEXTSTYLE
Annotative, TEXTSTYLE
Standard)
Eval: $(EQ, $(getvar,TEXTSTYLO), Standard)
Eval: $(GETVAR, TEXTSTYLO)
Err: $(GETVAR,??)
===> 0
===> TEXTSTYLE
Standard
TEXTSTYLE
Enter new value for TEXTSTYLE <"Standard">: Standard
Type a command
```

Chybové hlásenie $(GETVAR,??) je jedným zo štyroch možných hlásení. Ich prehľad a stručný popis uvádzame v tabuľke 3.2.

Tabuľka 3.2: Prehľad chybových hlásení jazyka DIESEL v príkazovom riadku AutoCADu.

Chybové hlásenie	Popis
$?	syntaktická chyba, napr. chýbajúca zátvorka
$(XY,??)	nesprávne argumenty funkcie XY
$(XY)??	neznáma funkcia XY
$(++)	príliš dlhý reťazec - výraz bol skrátený

AutoLISP Použitím výrazov jazyka AutoLISP v makrách získame ešte silnejší nástroj než pri použití jazyka DIESEL. Môžeme tu totiž pracovať s ľubovoľným množstvom premenných, získavať hodnoty od užívateľa a rovnako môžeme využívať aj cykly. Podrobne sa jazyku AutoLISP venujeme v kapitolách 4 až 10.

Ako príklad makra, v ktorom využívame jazyk AutoLISP, uvádzame makro 3.7. V tomto makre pomocou výrazu v jazyku AutoLISP získame od užívateľa dve reálne čísla, ktoré použijeme pri kreslení severky. Toto makro postupne od užívateľa získa polomer severky, uhol jej natočenia a nakoniec aj stred severky.

Makro 3.7: Makro na kreslenie severky s využitím výrazov jazyka AutoLISP

```
^C^C(setq polomer (getreal "Zadajte polomer severky: "));\(setq uhol (getreal "Zadajte uhol
    natocenia: "));_CIRCLE;\!polomer;_LINE;@;(strcat "@" (rtos polomer) "<" (rtos uhol));;
```

3.3 Skript

Skripty sú, podobne ako AutoLISP alebo makrá, prostriedkom na zrýchlenie a zjednodušenie práce v AutoCADe. Po syntaktickej stránke sú podobné makrám, respektíve vstupom do príkazového riadku. V skripte totiž zaznamenávame (spravidla) len postupnosť príkazov AutoCADu a vstupov do nich. Na rozdiel od makra sa ale skript ukladá vo forme textového súboru s koncovkou `*.scr`. Veľkou výhodou skriptov je, že na rozdiel od makra ich spustíme aj v AutoCAD for Mac a na rozdiel od AutoLISPu ich spustíme aj v AutoCAD LT. Nevýhody skriptov:

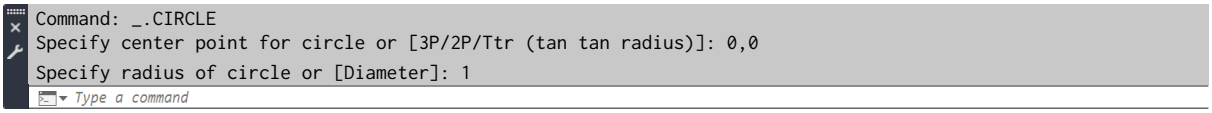
- nepodporujú užívateľský vstup (bod, číslo...) (mimo LT verzie je možné to vyriešiť pomocou funkcií AutoLISPu),
- nepodporujú dialógové okná. Dokáže ich otvoriť, ale nedokáže s nimi pracovať a je nutný zásah užívateľa. Vstupy treba riešiť cez príkazový riadok,
- počas behu skriptu nie je možné v AutoCADe robiť nič iné, pretože je stále aktívny príkaz **SCRIPT**.

Tip

Spustenie skriptu je vlastne spustenie príkazu **SCRIPT** s adresou skriptu ako vstupom do príkazu. To znamená, že skript sa môže spúšťať aj pomocou palety nástrojov alebo tlačidla, pre ktoré vytvoríme vhodné makro príkazu.

3.3.1 Tvorba skriptov

Tvorba skriptov je veľmi jednoduchá. Ako prvý si vytvoríme skript na vytvorenie jednotkovej kružnice s pevne daným stredom. Začnime tým, že si takúto kružnicu vytvoríme v AutoCADe:



Ak z príkazového riadku ponecháme iba užívateľský vstup (teda všetko za dvojbodkami), dostaneme skript 3.1. Vidíme, že skript je vlastne len postupnosť vstupov, ktoré postupne odosielame do AutoCADu. V tomto prípade vstupy odosielame pomocou nového riadku. Teda nový riadok musí byť za každým vstupom, čo znamená, že každý skript bude mať na svojom konci len prázdny riadok. V tejto ukážke na prázdny riadok upozorňuje aj komentár, ktorý začína znakom ";".

Skript 3.1: Vytvorenie jednotkovej kružnice

```
_.CIRCLE
0,0
1
; prazdny riadok = Enter
```

Alternatívou k odosielaniu vstupov pomocou nového riadku je odosielanie pomocou medzery. Posledný vstup v skripte však vždy musíme odoslať novým riadkom, pozri skript 3.2.

Skript 3.2: Vytvorenie jednotkovej kružnice v jednom riadku

```
_.CIRCLE 0,0 1
; koniec
```

Pomocou skriptov vieme vo výkrese upravovať a nastavovať štýly (napr. textový alebo kótovací), vytvoriť hladiny podľa daných štandardov, alebo nastaviť rôzne systémové premenné. Musíme pri tom použiť verzie príkazov pre príkazový riadok.

Tip

Všetky tieto nastavenia výkresov vieme zabezpečiť samozrejme aj inak. Napríklad pomocou šablóny, v ktorej máme už všetky nastavenia uložené, alebo pomocou palety Design Center, prípadne pomocou makra (pozri časť 3.2). Použitie skriptov má výhodu vo veľmi jednoduchom použití a možnosti prenosu medzi počítačmi, najmä ak chceme použiť iba určitú časť nastavení výkresu.

Ako ďalšie, komplexnejšie, príklady využitia skriptov uvádzame skript na vytvorenie nového kótovacieho štýlu (skript 3.3) a nového textového štýlu (skript 3.4). Kompletné nastavenie textového štýlu sme zabezpečili pomocou príkazu **-STYLE**. Pri tvorbe kótovacieho štýlu nám príkaz **-DIMSTYLE** stačiť nebude. Pomocou neho síce vieme vytvoriť nový štýl, ale veľa možností nastavenia neponúka. Tie budeme nastavovať iba priamo cez systémové premenné, ktoré ukladajú tieto nastavenia a až nakoniec ich uložíme cez príkaz **-DIMSTYLE** a voľbu **SAVE**.

Zoznam všetkých systémových premenných, ktoré sa ukladajú pri kótovacom štýle vieme získať napríklad aj priamo v príkaze **-DIMSTYLE**, s následnou voľbou **STATUS**. V skripte 3.3 nastavujeme len tri systémové premenné z celkového počtu sedemdesiatdeväť. To môže mať za následok neočakávaný výstup. Nevieme nič o nastavení výšky textu, textovom štýle a množstve ďalších nastavení, ktoré nový kótovací štýl prebral z dovtedy aktívneho kótovacieho štýlu. Preto, ak si chceme byť istý všetkými nastaveniami, treba všetky do skriptu zahrnúť.

Skript 3.3: Vytvorenie a nastavenie nového kótovacieho štýlu

```
;Vytvorenie noveho kotovacieho stylu
;Vyber sipky
DIMBLK
ArchTick
;Zapnutie fixnej dlzky vynasacich ciar
DIMFXLON
On
;Nastavenie dlzky vynasacich ciar
DIMFXL
5.0
;Ulozenie noveho kotovacieho stylu
_-DIMSTYLE
S
novy
;prazdny riadok
```

Skript 3.4: Vytvorenie a nastavenie nového textového štýlu

```
;Vytvorenie noveho textoveho stylu
\_-STYLE
;Enter name of text style or [?] <Standard>:
novy
;Specify font name or font filename (for SHX) <txt>:
arial
;Specify height of text or [Annotative] <0.0000>:
\_A
;Create annotative text style [Yes/No] <Yes>:
\_Y
;Match text orientation to layout? [Yes/No] <No>:
\_N
;Specify height of text or [Annotative] <0.0000>:
0.0
;Specify width factor <1.0000>:
1.0
;Specify obliquing angle <0>:
0
;Display text backwards? [Yes/No] <No>:
\_N
;Display text upside-down? [Yes/No] <No>:
\_N
;prazdny riadok
```

3.3.2 Spúšťanie skriptov

Aby sme skript mohli spustiť v AutoCADe, musíme ho uložiť ako textový súbor. Na názve súboru nezáleží, no dôležité je, aby mal koncovku `*.scr`. To môžeme dosiahnuť tak, že vytvoríme textový súbor, napr. s koncovkou `*.txt`, a túto koncovku jednoducho prepíšeme na `*.scr`. Takýto súbor vieme otvoriť

v textovom editore (Notepad, Notepad++,...) a upravovať jeho obsah.

Spúšťať skripty v AutoCADe je možné viacerými spôsobmi. Napríklad pri tvorbe skriptu a jeho ladení sa najviac hodí spôsob „potiahni a pusti" ("drag and drop"), kde súbor so skriptom (napr. skript 3.1) jednoducho potiahneme a pustíme v grafickom okne AutoCADu. Skript sa okamžite vykoná, teda vykreslí sa jednotková kružnica so stredom v bode `0,0`. Okrem toho, v príkazovom riadku budeme vidieť:

```
Command: _SCRIPT
Enter script file name <D:\cad\vykres.scr>: "D:\cad\kruznica.scr"
Command: _.CIRCLE
Specify center point for circle or [3P/2P/Ttr (tan tan radius)]: 0,0
Specify radius of circle or [Diameter] <1.0000>: 1
Type a command
```

Z príkazového riadku vieme vyčítať, že sme spustili príkaz **SCRIPT**. Z toho je jasné, že skript vieme spustiť príkazom aj priamo v AutoCADe, pričom ho nájdeme aj na páse kariet Manage › Applications › Run Script.

Keďže skript vieme spustiť pomocou príkazu, dokážeme ho spustiť aj v AutoLISPe. To ale nie je úplne priamočiare, pretože príkaz **SCRIPT** vyvolá dialógové okno, a to nedokážeme cez AutoLISP ovládať. Príkaz **-SCRIPT** nanešťastie neexistuje a tak to musíme vyriešiť systémovou premennou **FILEDIA**. Pokiaľ ju nastavíme na hodnotu **0**, dialógové okno sa po spustení príkazu **SCRIPT** neotvorí, ale výzva sa objaví len v príkazovom riadku. Finálnu funkciu na spúšťanie skriptov uvádzame v kóde 3.1.

AutoLISP kód 3.1: Definícia funkcie **SpustiSkript**

```
1 (defun SpustiSkript (cesta / povodneFileDia)
2     (setq povodneFileDia (getvar 'FILEDIA))
3     (setvar 'FILEDIA 0)
4     (command "_.SCRIPT" cesta)
5     (setvar 'FILEDIA povodneFileDia)
6 )
```

Argumentom našej funkcie je cesta k súboru skriptu, takže ju môžeme zavolať na konkrétny súbor:

```
Command: (SpustiSkript "D:\cad\kruz.scr")
Type a command
```

Prípadne môžeme pomocou funkie **getfiled**, pozri časť 9.3, vyvolať dialógové okno pre výber súboru užívateľom:

```
Command: (SpustiSkript (getfiled "Vyberte skript na spustenie" "D:\\cad\\" "scr" 16))
Type a command
```

3.3.3 Ďalšie príkazy AutoCADu pre prácu so skriptami

So skriptami sa v AutoCADe viažu aj ďalšie príkazy, konkrétne **SCRIPTCALL**, **DELAY** a **RSCRIPT**, ktoré popíšeme v tejto časti.

Príkaz **SCRIPTCALL** sa využíva, keď potrebujeme zavolať skript vo vnútri nejakého iného skriptu. Tento príkaz zaistí, že po vykonaní vnoreného skriptu sa dokončí aj pôvodný skript. Ak by sme totiž použili na zavolanie vnoreného skriptu príkaz **SCRIPT**, hlavný skript by sa prerušil po dokončení vnoreného skriptu by sa už hlavný skript nedokončil.

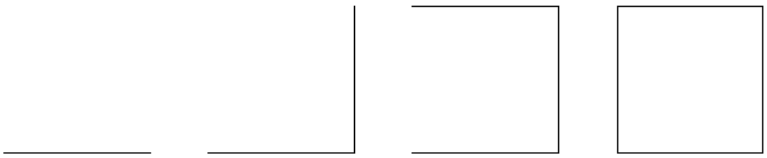
Majme napríklad trojicu skriptov, `skript1.scr`, `skript2.scr`, a `skript3.scr`. Namiesto spúšťania skriptov po jednom ich môžeme zavolať všetky naraz pomocou jedného hlavného skriptu 3.5.

Skript 3.5: Skript na postupné spúšťanie troch skriptov

```
FILEDIA 0
_.SCRIPTCALL "D:\cad\skript1.scr"
_.SCRIPTCALL "D:\cad\skript2.scr"
_.SCRIPTCALL "D:\cad\skript3.scr"
FILEDIA 1
; prazdny riadok = Enter
```

Ak by sme ale použili príkaz **SCRIPT**, `skript1.scr` by sa vykonal, ale do skriptu `skriptHlavny.scr` by sme sa už nevrátili. To by zároveň znamenalo, že systémová premenná **FILEDIA** by ostala na hodnote **0**.

Príkaz **DELAY** sa dá využiť pri tvorbe animácií v AutoCADe na vytvorenie akejsi pauzy pred nasledujúcou inštrukciou. Dĺžka pauzy sa udáva v milisekundách. Ak by sme napríklad chceli spraviť animovaný návod na nakreslenie štvorca, pozri obr. 3.3, mohli by sme použiť skript 3.6. Príkaz **DELAY** vieme využiť tiež pri prezentácii, kedy chceme meniť uložené pohľady na výkres. Pri takýchto prezentáciách je dobré tieto pohľady meniť dookola. Na nekonečné opakovanie skriptu využijeme príkaz **RSCRIPT** na konci skriptu 3.7. Takýto nekonečný skript ukončíme až klávesom esc alebo ⟵ backspace. Po jeho ukončení ho môžeme opätovne spustiť príkazom **RESUME**.



Obr. 3.3: 4 kroky animácie vytvorenej pomocou skriptu 3.6

.

Skript 3.6: Skript na animovanie postupu kreslenia štvorca príkazom **LINE**

```
LINE
10,10
20,10

DELAY 5000
LINE
20,20
20,10

DELAY 5000
LINE
20,20
10,20

DELAY 5000
LINE
10,20
10,10
 ;prazdny riadok
```

Skript 3.7: Skript na prepínanie medzi 3 uloženými pohľadmi po 5 sekundách

```
−view r PRVYHOHLAD
DELAY 5000
−view r DRUHYHOHLAD
DELAY 5000
−view r TRETIHOHLAD
DELAY 5000
RSCRIPT
;
```

3.3.4 Skripty s výrazmi jazyka AutoLISP

Súčasťou skriptu môže byť aj výraz v jazyky AutoLISP (pripomíname, že AutoCAD LT nepodporuje AutoLISP, a to ani v skriptoch). Pomocou AutoLISPu teda vieme aj do skriptov zahrnúť podmienky, cykly a užívateľský vstup. Rovnako môžeme použiť aj jednoduché aritmetické funkcie AutoLISPu, pozri skript 3.8.

Skript 3.8: Ukážka jednoduchého skriptu s použitím výrazu jazyka AutoLISP

```
_.CIRCLE 0,0 (/ 1.0 3)
;
```

Na druhú stranu si treba uvedomiť, že do skriptov síce môžeme vložiť výrazy v AutoLISPe, avšak naopak to neplatí. Skript 3.9 nie je po syntaktickej stránke v poriadku, pretože do funkcie **repeat** sme ako argument posiela neplatný výraz (z pohľadu AutoLISPu). Takýto výraz patrí do príkazového riadku. Po správnosti by takýto skript mal vyzerať tak, ako uvádzame v skripte 3.10.

Skript 3.9: Nefunkčný skript. Voláme totiž funkciu **repeat** s neplatnými argumentami

```
(repeat 5
_.CIRCLE 0,0 1
)
;
```

Skript 3.10: Skript v ktorom do funkcie **repeat** posielame platné argumenty

```
(repeat 5
(command "_.CIRCLE" "0,0" "1")
)
;
```

Z uvedeného príkladu vidno, že vo vnútri výrazov AutoLISPu musíme používať výhradne výrazy v jazyku AutoLISP.

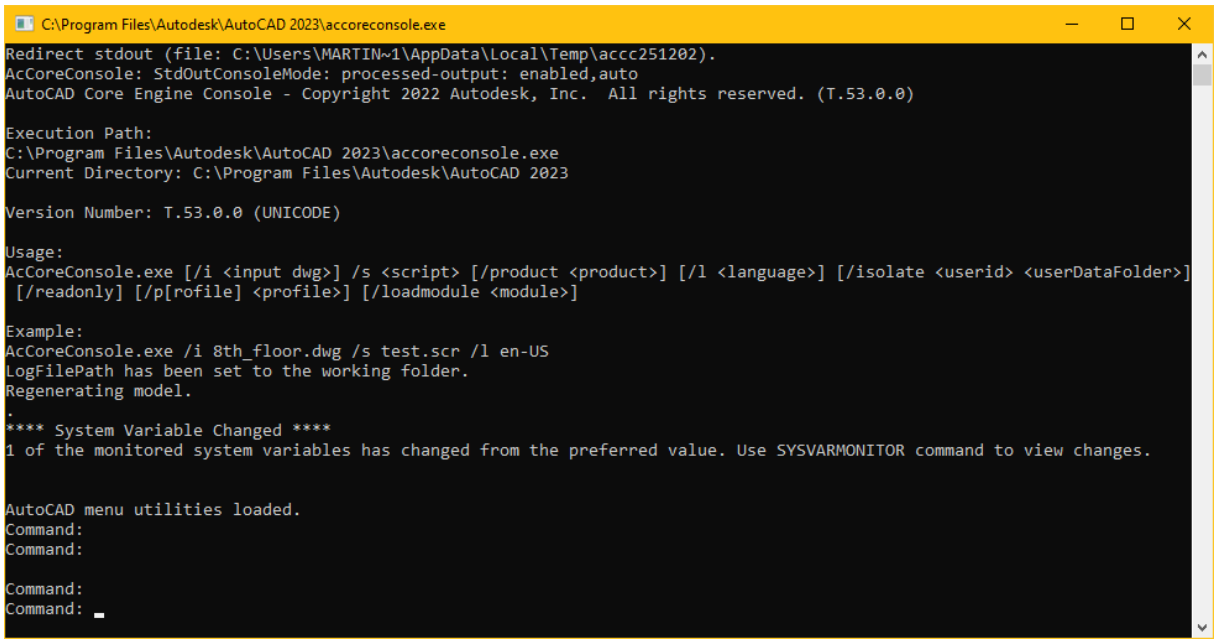
3.3.5 AutoCAD Core Console

AutoCAD Core Console si môžeme predstaviť ako zjednodušenú verziu AutoCADu. Konkrétne ide o verziu AutoCADu bez tlačidiel, kresliacej plochy, teda bez celého užívateľského prostredia, pozri obr. 3.4. Z „plného” AutoCADu tu zostala iba väčšina príkazov. AutoCAD Core Console sa spúšťa v programe

Príkazový riadok (Command Prompt), kde môžeme pracovať podobne ako v príkazovom riadku AutoCADu.

Tip

AutoCAD Core Console nájdeme podobne ako „plný" AutoCAD v 🗀 `C:\Program Files\Autodesk\AutoCAD 2023\accoreconsole.exe`.



Obr. 3.4: Konzola po spustení AutoCAD Core Console

AutoCAD Core Console nie je vhodný ako plnohodnotná alternatíva „plného" AutoCADu. Je však skvelou voľbou pri automatízácii práce s veľkým množstvom súborov. Oproti "plnému" AutoCADu je totiž výhodou AutoCAD Core Console rýchle spustenie a ukončenie samotného programu, ale aj rýchle otvorenie, uloženie a zatvorenie súborov. V kombinácii so skriptami a `*.bat` súbormi (pozri nižšie) môže užívateľ napr. [2]:

- vytvoriť a nastaviť hladiny,
- vytvoriť a nastaviť rôzne štýly (kótovací, textový ...),
- nastaviť systémové premenné,
- extrahovať dáta z atribútov,
- upraviť výkres,
- tlačiť výkres.

a to aj hromadne, pre veľké množstvo výkresov. Súbory s koncovkou `*.bat` [13] sú textové súbory, ktoré obsahujú skripty. Týmito skriptami môžeme automatizovať prácu v prostredí Microsoft Windows.

Ako príklad uvádzame `*.bat` skript 3.11, ktorým v AutoCAD Core Console otvoríme všetky `*.dwg` súbory v priečinku, v ktorom sa nachádza samotný `*.bat` súbor. Po otvorení každého `*.dwg` súboru sa spustí skript 3.12, uložený v súbore `plot.scr`, v 🗀 `D:\cadcore`. Samotný `*.bat` súbor spustíme dvojklikom 🖱 na súbor, čím otvoríme konzolu ako na obr. 3.4.

Skript 3.11: Obsah `*.bat` skriptu na otvorenie všetkých `*.dwg` súborov v aktuálnom priečinku v AutoCAD Core Console a spustením skriptu `plot.scr`

```
FOR %%f IN (*.dwg) DO "C:\Program Files\Autodesk\AutoCAD 2023\accoreconsole.exe" /i "%%f"
    /s "D:\cadcore\plot.scr" /l en-US
```

Skript 3.12: Tlač Layoutu s názvom Layout1 do formátu `*.pdf`

```
FILEDIA
0
_ZOOM
_E
_.-PLOT
_N
Layout1

DWG to PDF

N
Y
FILEDIA
1
_QUIT
Y
;
```

Tip

Na uvedenom príklade môžeme naznačiť aj spomínanú úsporu času. Samotná manuálna tlač jedného výkresu do formátu `*.pdf` trvá aspoň pár sekúnd, nehovoriac o otváraní a zatváraní výkresu. Pri tlači výkresov pomocou skriptu 3.11 za ten istý čas ich stihneme vytlačiť možno aj desať.

4 Úvod do jazyka AutoLISP a vývojových prostredí

AutoLISP je programovací jazyk založený na jazyku LISP. Programovací jazyk LISP vznikol v roku 1958, čo z neho robí druhý najstarší programovací jazyk spomedzi tzv. vyšších programovacích jazykov. To znamená, že by mal byť jeho zápis bližšie mysleniu ľudí, než procesom v počítači. Samotný názov LISP vznikol zo slovného spojenia **LIS**t **P**rocessing, čo by sme mohli voľne preložiť ako spracovanie zoznamov. Z toho by sa dalo správne predpokladať, že je primárne určený na prácu s dátami vo forme zoznamov (list = zoznam).

Tip

To, že LISP stále nepatrí do starého železa sa ukázalo aj v Google AI Contest v roku 2010. Víťaz totiž programoval v jazyku LISP, pričom súperil aj s podstatne mladšími jazykmi, ako C++, C#, Python alebo Go či Scala.

V AutoCADe je práve spracovanie zoznamov veľmi časté. Pod zoznamom si môžeme predstaviť napr. bod s tromi súradnicami, alebo ľubovoľný objekt s jeho vlastnosťami uloženými ako zoznam. Aj táto vlastnosť prispela k tomu, že LISP bol ideálnym programovacím jazykom, ktorý by pomohol vyvíjať užívateľom AutoCADu vlastné príkazy. V roku 1986, len štyri roky po prvej verzii AutoCADu, tak uzrel svetlo sveta programovací jazyk AutoLISP. AutoLISP je verziou jazyku LISP, upravenou pre lepšiu interakciu s AutoCADom.

Jazyk AutoLISP bol vyvíjaný až do roku 1995, kedy padlo rozhodnutie o jeho náhrade za modernejšie jazyky, ako napr. VBA (**V**isual **B**asic for **A**pplications). O to väčším prekvapením bolo následné predstavenie jazyka Vital LISP, ktorý priniesol širšie možnosti použitia a hlavne zabudované vývojové prostredie "Integrated Development Environment" (IDE). Takéto prostredie v AutoCADe chýbalo a kódy sa zvyčajne písali v textovom editore (napr. Notepad). Autodesk onedlho Vital LISP kúpil a premenoval ho na Visual LISP. Od verzie AutoCAD 2000 je viac menej v tej istej podobe súčasťou softvéru AutoCAD až dodnes. Veľkou zmenou bolo doplnenie podpory pre vývojové prostredie Visual Studio Code od verzie AutoCAD 2021. Viac sa vývojovým prostrediam pre AutoLISP venujeme v ďalších častiach tejto kapitoly.

Tip

V roku 2022 bola pridaná podpora jazyka AutoLISP dokonca aj do webovej verzie AutoCADu, ktorú nájdete na https://web.autocad.com.

Programy v jazyku AutoLISP a ich načítanie do AutoCADu AutoLISP si v AutoCADe našiel uplatnenie pri automatizácii procesov. Nejeden užívateľ AutoCADu si ušetril hodiny práce použitím príkazov napísaných v jazyku AutoLISP. A to už prostredníctvom vlastných kódov, alebo nahratím verejne dostupných kódov do AutoCADu. Program napísaný v jazyku AutoLISP môže byť uložený v troch rôznych

formátoch s príponou:

- `*.lsp` má textový súbor so zdrojovým kódom v jazyku AutoLISP,
- `*.fas` (**FAS**t Load AutoLISP) má skompilovaný súbor v jazyku AutoLISP,
- `*.vlx` (**V**isual **L**isp e**X**ecutable) má súbor, v ktorom sú skompilované viaceré súbory `*.lsp`, alebo napr. `*.dcl` (**D**ialog **C**ontrol **L**anguage), ktorý definuje dialógové okno.

Formáty získané skompilovaním (`*.fas` a `*.vlx`) sú optimalizované a teda rýchlejšie pracujú, ale hlavne sa nevieme dostať k ich zdrojovému kódu. Skompilovaním teda vieme ochrániť kódy pred neželanými zásahmi do zdrojového kódu.

Pred samotným spustením takýchto programov ich treba najprv nahrať do AutoCADu. To sa dá viacerými spôsobmi:

1. príkazom **APPLOAD** (tlačidlom Manage › Applications › Load Application),
2. príkazom **CUI** (tlačidlom Manage › Customization › User Interface, pozri časť 2.2.1.10,
3. „potiahnutím" do okna AutoCADu a „pustením" (drag and drop),
4. vývojovým prostredím Visual LISP Editor, pozri časť 4.3, alebo Visual Studio Code, pozri časť 4.4,
5. požitím funkcií AutoLISPu **autoload** alebo **load** v súboroch `acad.lsp` alebo `acaddoc.lsp`.

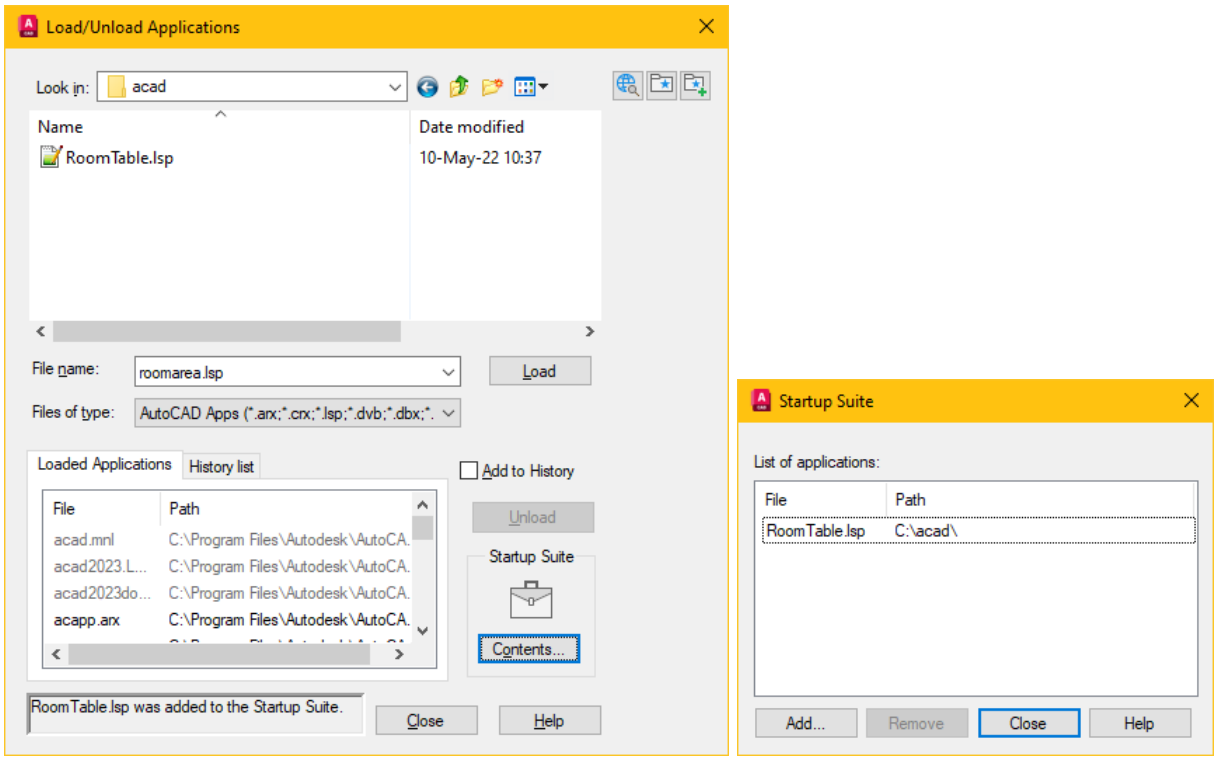
Výber spôsobu nahratia závisí na tom, či chceme mať súbor nahratý vždy a pre všetky otvorené `*.dwg` súbory, alebo len pre jeden dokument a len kým nevypneme AutoCAD. Pokiaľ chceme mať súbory nahraté vždy, treba na to použiť príkaz **CUI**, ktorý je popísaný v časti 2.2.1.10, alebo príkaz **APPLOAD**. Pri použití príkazu **APPLOAD** treba vybrané súbory nahrať v časti Startup Suite, pozri obr. 4.1. Po stlačení tlačidla Contents... sa zobrazí dialógové okno Startup Suite, kde môžeme jednoducho pridať alebo odstrániť súbory na nahrávanie.

Posledný krok je povolenie nahratia súboru v dialógovom okne Security - Unsigned Executable File, pozri obr. 4.2. Ak vyberieme možnosť Always Load, AutoCAD už pri svojom ďalšom spustení tieto súbory nahrá bez upozornenia. Pri voľbe Load Once sa síce súbory nahrajú tiež, avšak pri ich najbližšom nahrávaní sa znova objaví také isté dialógové okno. V prípade, že súbory nahrávame len dočasne, sú tieto voľby rovnocenné.

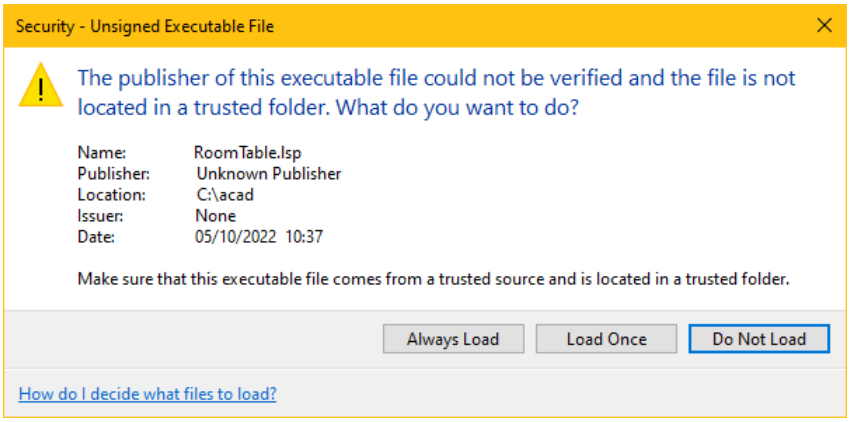
Tip

Express tools pozná snáď každý užívateľ AutoCADu. Málokto však vie, že väčšina týchto príkazov je napísaná v AutoLISPe. Navyše, množstvo z nich napísali samotní užívatelia AutoCADu. Kódy niektorých príkazov zvyčajne nájdete v `C:/Program Files/Autodesk/AutoCAD 2021/Express`.

Syntax jazyka AutoLISP Kód v jazyku AutoLISP má svoju charakteristickú syntax plnú zátvoriek. V AutoLISPe je totiž všetko zoznamom, a keďže zoznam sa zapisujeme do zátvoriek, napr. `(1 2 3)`, dá sa rýchlo pochopiť, prečo budeme v AutoLISPe obklopený zátvorkami. V tejto ukážke sú v zozname 3 prvky, v AutoLISPe označované ako atómy alebo prvky. Definícia atómu je celkom jednoduchá, a síce všetko, čo nie je zoznam, je atóm. Príkladom atómu je číslo `1` alebo reťazec znakov `"text"`.



Obr. 4.1: Dialógové okno Load/Unload Application (vľavo), v ktorom po stlačení tlačidla Contents... otvoríme ďalšie dialógové okno Startup Suite (vpravo). V tomto okne môžeme určiť `*.lsp` súbory, ktoré sa majú načítať vždy pri spustení AutoCADu.



Obr. 4.2: Dialógové okno Security - Unsigned Executable File, v ktorom musíme potvrdiť dôveryhodnosť nahrávaných `*.lsp` súborov

Tip

Pre lepšiu čitateľnosť sa najmä v dlhších kódoch pridávajú prázdne riadky na oddelenie rôznych častí kódu. Tieto prázdne riadky však nemajú vplyv na funkčnosť kódu, podobne ako rozdelenie výrazov na viac riadkov.

V praxi patrí prvé miesto za zátvorkou názvu funkcii. Počet zvyšných atómov závisí od konkrétnej funkcie. Tieto atómy sú vstupom do funkcie. Symbolicky by sme to mohli zapísať nasledovne (uvedená forma je použitá aj v pomocníkovi AutoCADu): **(nazov_funkcie argumenty [nepovinne_argumenty])**.

Z uvedeného zápisu je zrejmé, že za názvom funkcie nasledujú jej argumenty, pričom niektoré z nich

môžu byť aj nepovinné (voliteľné). Názov funkcie je vždy len jedno slovo. Počet argumentov závisí na konkrétnej funkcii, sú funkcie, ktoré vyžadujú 0, 1, 2 alebo viac argumentov. Ak má funkcia aj nepovinné argumenty, znamená to, že sa táto funkcia dá zavolať rovnako s alebo bez týchto nepovinných argumentov, prípadne len s niektorými z nich. Ako konkrétny príklad funkcie s ľubovoľným počtom argumentov uvádzame rôzne volania funkcie **+**:

```
Command:(+)
0
Command:(+ 5)
5
Command:(+ 5 3)
8
Command:(+ 5 3 20)
28
Type a command
```

Naopak, ako príklad funkcie s fixným počtom argumentom uvádzame funkciu **1+**, ktorá vyžaduje presne jeden argument. V prípade ak funkciu **1+** zavoláme s iným počtom argumentov, vráti nám chybové hlásenie:

```
Command: (1+)
; error: too few arguments
Command: (1+ 4)
5
Command: (1+ 4 5)
; error: too many arguments
Type a command
```

Ako ste si mohli všimnúť, príkazový riadok AutoCADu sa dá využiť na spúšťanie výrazov v jazyku AutoLISP, pozri časť 4.1. Do príkazového riadku môžeme zadávať aj rozsiahlejšie výrazy, výrazy vnorené do iných výrazov, alebo jednoducho len postupnosť výrazov. Avšak čím komplikovanejší výraz používame, tým menej vhodné je používanie príkazového riadku na jeho spustenie. Hlavné využitie AutoLISPu však nájdeme pri vytváraní vlastných a zvyčajne dlhších programov. Pod programom si môžeme predstaviť skupinu volaní funkcií s nejakým spoločným cieľom, napríklad opakovaním tých istých operácií na iných objektoch, spojenie viacerých funkcií do jedného celku alebo jednoducho len s cieľom úspory času.

Tip

Pri programovaní v jazyku AutoLISP často potrebujeme nájsť ten správny názov príkazu, funkcie alebo systémovej premennej. Tu sa môže hodiť príkaz **LSP**, ktorým vieme zobraziť zoznam všetkých príkazov AutoCADu, funkcií AutoLISPu alebo systémových premenných. Ak by sme potrebovali viac informácií o systémových premenných, môžeme použiť príkaz **SYSVDLG**.

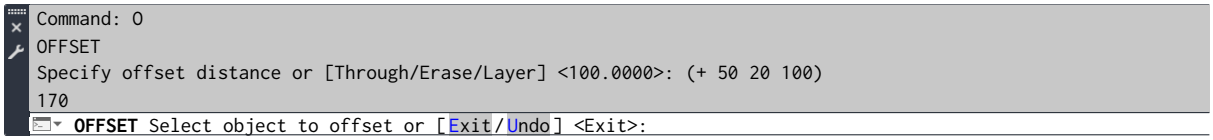
V ďalších častiach kapitoly postupne priblížime rôzne možnosti ako tvoriť vlastné programy v jazyku AutoLISP. Začneme s použitím príkazového riadku AutoCADu, pozri časť 4.1. Pokračovať budeme s klasickým spôsobom pomocou jednoduchého textového editora, pozri časť 4.2, popíšeme aj integrované vývojové prostredie AutoCADu nazývanom Visual LISP Editor, pozri časť 4.3. Napokon si predstavíme aj nedávno pridanú možnosť a využijeme bezplatný softvér Visual Studio Code od Microsoftu, pozri časť 4.4.

4.1 Príkazový riadok AutoCADu

AutoCAD je s AutoLISPom prepojený natoľko, že výrazy v jazyku LISP je možné zadávať aj priamo do príkazového riadku AutoCADu. Vyskúšať si to môžeme na jednoduchom príklade. Výraz `(+ 5 3)` jednoducho napíšeme do príkazového riadku rovnako, ako zvykneme písať názvy príkazov:

A po odoslaní tlačidlom enter dostaneme:

V prvom riadku vidíme zadaný výraz. V druhom riadku vidíme výsledok súčtu, inými slovami vidíme, čo vrátila funkcia `+`. Výrazy jazyka AutoLISP však môžeme využiť aj priamo v príkazoch. V takom prípade sa hodnota vrátená výrazom využije ako vstup do príkazu. Napríklad pri príkaze **OFFSET** môžeme zadať vzdialenosť odsadenia nasledovne:



Príkazový riadok je výborný prostriedok na otestovanie krátkych výrazov. Pri dlhších výrazoch, resp. programoch prináša v podstate iba nevýhody:

- neprehľadnosť – písanie iba v jednom riadku, teda bez možnosti štruktúrovania kódu
- nemožnosť opraviť výraz, ktorý sme už v príkazovom riadku potvrdili
- nemožnosť ukladať kódy pre použitie v budúcnosti
- žiadna kontrola syntaxe.

Snáď jedinou pomôckou príkazového riadku (na ktorú sa však môžeme spoľahnúť pri každom spustení výrazu v AutoLISPe) je pomoc s nevyváženými (neuzatvorenými) zátvorkami a úvodzovkami. Ak napr. odošleme výraz `(+ 5 3`, reťazcom `(_>` nám AutoCAD dáva do pozornosti, že máme jednu neuzavretú zátvorku:

Následne môžeme chýbajúcu zátvorku odoslať v príkazovom riadku a uzavrieť tak výraz:

Alebo môžeme výraz medzi zátvorkami ešte doplniť a až následne uzavrieť zátvorkou:

Okrem reťazca `(_>` sa môžeme stretnúť aj s reťazcom `((_>` pri dvoch chýbajúcich ukončovacích zátvorkách, s reťazcom `(((_>` pri troch a tak ďalej. Podobne nás AutoCAD upozorní na chýbajúce ukončenie textového reťazcu v úvodzovkách obdobnými reťazcami `("_>`, `(("_>`, `(((("_>` a tak ďalej.

4.2 Notepad a Notepad++

Notepad, resp. Poznámkový blok je jednoduchý textový editor, ktorý na písanie kódu úplne postačí. Otvoriť ho vieme aj priamo z AutoCADu príkazom **NOTEPAD**. Toto prostredie je ale veľmi nepohodlné, pretože pri písaní a kontrole kódu nám nijakým spôsobom nepomáha.

O máličko lepšie sa pracuje v editore Notepad++ [10], ktorý ponúka aj farebné odlíšenie komentárov, názvov funkcií, čísiel, textových reťazcov a zátvoriek (resp. dvojíc zátvoriek). Okrem toho nám aj našepkáva názvy funkcií a symbolov. Pri písaní dlhších kódov poteší aj možnosť skrytia (zrolovania) časti kódu medzi zátvorkami. Písanie spríjemnia aj rôzne klávesové skratky (napr. ctrl + Q pre (od)komentovanie), tvorba makier alebo možnosť inštalácie rôznych pluginov.

Tip

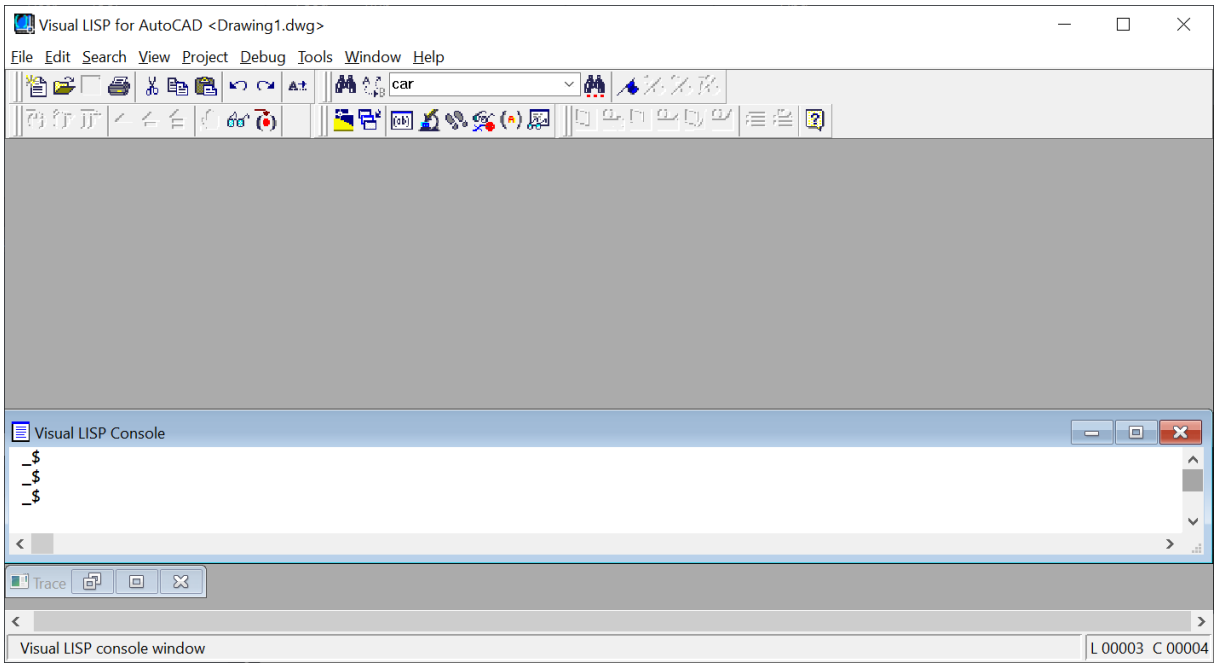
V AutoCADe môžeme využiť zabudovaný príkaz **NOTEPAD** na spustenie editora Poznámkový blok. Naopak, editor Notepad++ nemá v AutoCADe zabudovaný takýto príkaz. Bez problémov sa ale dá vytvoriť s použitím funkcie **startapp** jazyka AutoLISP.

V ani jednom z týchto editorov však nie je podporované debugovanie (ladenie) a nahrávanie programov do AutoCADu. Nahrávať súbory je tak potrebné robiť manuálne jedným zo spôsobov popísaných vyššie, v úvode tejto kapitoly.

4.3 Visual LISP Editor

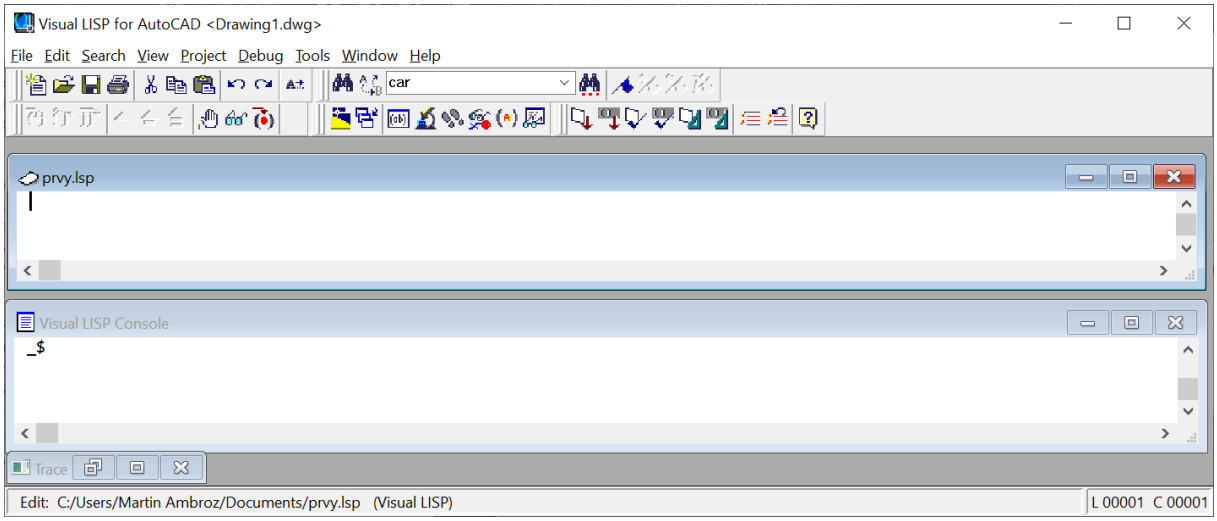
Ak pracujeme v AutoCADe, máme k dispozícii priamo zabudované vývojové prostredie (IDE) pre vývoj programov v jazyku LISP. Toto prostredie nájdeme v Manage › Applications › Visual LISP Editor , alebo ho spustíme príkazom **VLIDE** alebo **VLISP**. Visual Lisp Editor, pozri obr. 4.3, nám bude pomáhať pri písaní kódov nie len farebným odlíšením rôznych častí kódu, ale aj kontrolou syntaxe, alebo formátovaním kódu. Po otvorení súboru alebo vytvorení nového súboru sa v IDE sprístupnia tlačidlá v paneli nástrojov, ktoré budeme počas programovania používať, pozri obr. 4.4. Konkrétne ide o 9 tlačidiel, ktoré sú v pravej spodnej časti panela nástrojov:

- nahrá do AutoCADu kódy v aktuálnom `*.lsp` súbore,
- nahrá do AutoCADu kódy vo výbere v aktuálnom `*.lsp` súbore,
- skontroluje syntax kódu v aktuálnom `*.lsp` súbore,
- skontroluje syntax kódu vo výbere v aktuálnom `*.lsp` súbore,
- naformátuje (odsadí) riadky kódu v aktuálnom `*.lsp` súbore,
- naformátuje (odsadí) riadky kódu vo výbere v aktuálnom `*.lsp` súbore,
- zakomentuje aktuálny riadok alebo riadky vo výbere,
- odkomentuje aktuálny riadok alebo riadky vo výbere,



Obr. 4.3: Okno prostredia Visual LISP Editor po prvom spustení. Vidíme v ňom dve okná - Trace a Visual LISP Console, teda konzolu, v ktorej si môžeme overiť funkčnosť kódu.

- otvorí pomocníka, ak je vo výbere nejaká funkcia, help sa otvorí s informáciami o danej funkcii.



Obr. 4.4: Okno prostredia Visual LISP editor s otvoreným prázdnym `*.lsp` súborom. V ďalšom texte budeme uvádzať pod označením LISP kód práve obsah `*.lsp` súboru.

Tip

Vo Visual LISP editore sa môžeme stretnúť aj s takýmto zvláštnym kurzorom. Visual LISP editor nám takýmto kurzorom dáva najavo, že v ňom práve nie je možné pracovať, pretože je zaneprázdnený. Skoro vždy za to môže spustený užívateľom definovaný príkaz. Akonáhle príkaz ukončíme, kurzor sa vráti do pôvodnej podoby a môžeme pokračovať v práci v editore.

Prednastavená farebná schéma v editore Visual LISP je nasledovná:

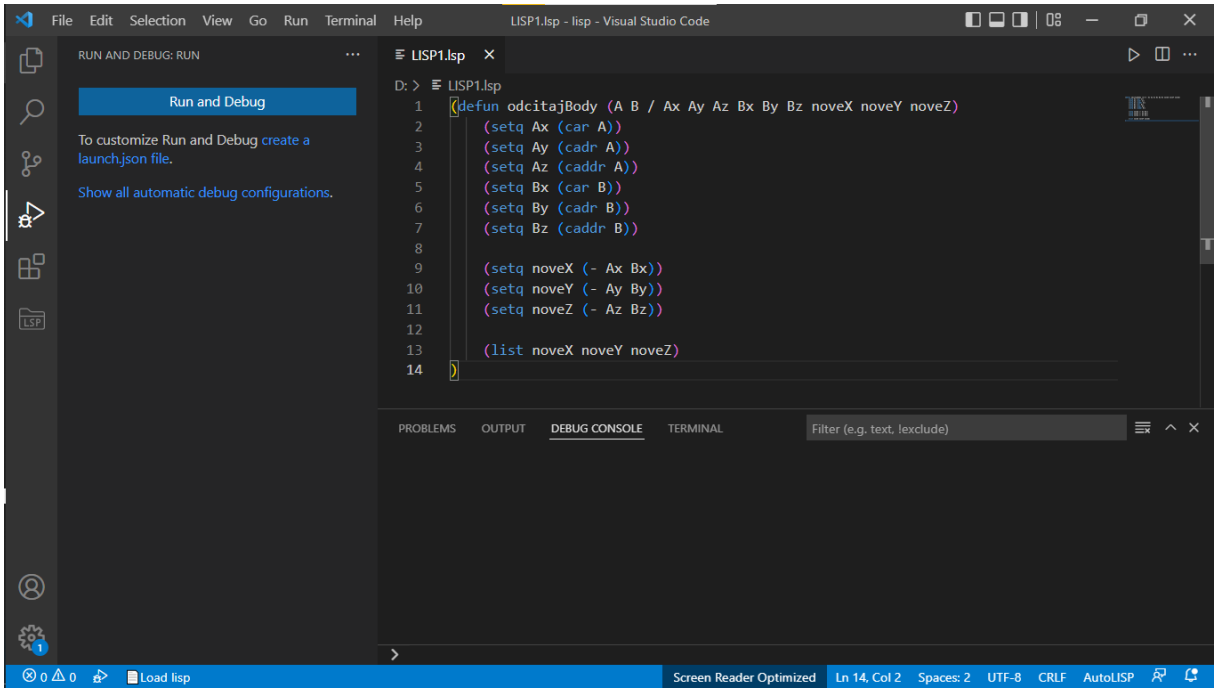
- zátvorky `( )`
- názvy funkcií, napr. `alert`
- celé čísla, napr. `1`
- desatinné čísla, napr. `3.14159265`
- znakové reťazce, napr. `"Toto je retazec."`
- komentáre, napr. `; Takto vyzerá komentár v kóde`
- ostatné (názvy premenných, názvy užívateľom definovaných funkcií atď.), napr. `mojaPremenna`

Tip

Dvojklikom na zátvorky vo Visual LISP Editore sa do výberu zahrnie všetko, čo patrí do danej dvojice zátvoriek. Výrazne to pomáha v čitateľnosti. Užitočné je to najmä pri kontrole syntaxe alebo aj pri čítaní kódu. Podobnú funkciu majú klávesové skratky ctrl + [a ctrl +], ktorými vieme prehadzovať kurzor na otváraciu, resp. uzatváraciu zátvorku.

4.4 Visual Studio Code

Od verzie AutoCADu 2021 je okrem Visual LISP Editora plne podporované aj externé vývojové prostredie Visual Studio Code, pozri obr. 4.5. Toto moderné prostredie sa využíva na programovanie v množstve programovacích jazykov, ktorých podporu si nainštalujeme formou rozšírení. Nás zaujíma konkrétne rozšírenie z dielne spoločnosti Autodesk s názvom AutoCAD AutoLISP Extension.



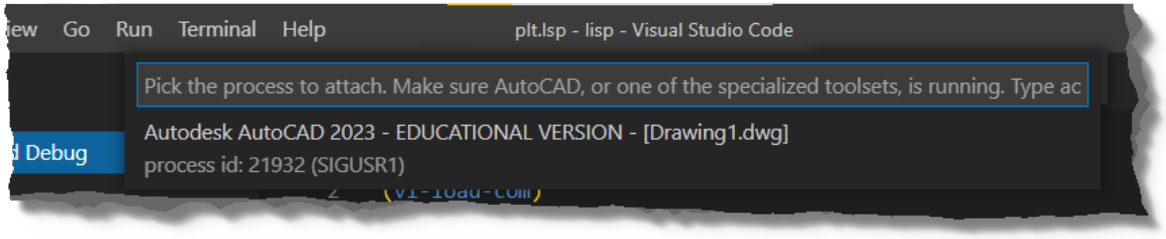
Obr. 4.5: Okno prostredia Visual Studio Code s otvoreným `*.lsp` súborom

Tip

Do editora Visual Studio Code je možné inštalovať rôzne doplnky. Ak Vám napríklad nevyhovuje farebná schéma jazyka AutoLISP, môžete siahnuť po doplnku, ktorý túto schému zmení.

Prostredie editora Visual Studio Code je podstatne modernejšie a vhodnejšie, najmä pre skúsenejších programátorov. Dá sa totiž povedať, že dokáže všetko to, čo Visual LISP Editor, avšak s tou nevýhodou, že ide o externý editor, ktorý napr. pre spúšťanie rozpracovaného kódu v AutoCADe musíme najskôr prepojiť s procesom, na ktorom softvér AutoCAD beží, pozri obr. 4.6.

Zámerom tejto časti je popísať len základné črty tohto editora a porovnať ho s ostatnými popisovanými editormi. Podrobný popis by bol príliš obsiahly a s ohľadom na to, že tieto skriptá sú určené pre programátorov začiatočníkov, nejde ani o najvhodnejšie prostredie.



Obr. 4.6: Výber procesu, s ktorým má editor Visual Studio Code prepojiť spúšťaný kód

5 Základné dátové typy a práca s nimi

V AutoLISPe rozlišujeme len dva základné dátové typy. Tým prvým sú zoznamy a druhým typom je prvok zoznamu, nazývaný atóm. Atóm je niečo nedeliteľné, teda v AutoLISPe napríklad číslo alebo reťazec znakov. Tým sa dostávame k podrobnejšiemu deleniu dátových typov, kde podľa [5] rozoznávame týchto deväť dátových typov, pozri tabuľku 5.1.

Tabuľka 5.1: Prehľad dátových typov

Dátový typ	Popis
INT (integer)	Celé číslo môže byť v intervale od -2 147 483 648 do 2 147 483 647. Ak potrebujeme pracovať s väčšími hodnotami, využívame s reálnymi číslami. Podrobnejšie sa celým číslam venujeme v časti 5.2.
REAL	Reálne čísla poskytujú vysokú presnosť (najmenej 14 signifikantných číslic). Reálne číslo môže byť napr. 0.123, -1.3, 2323.03, 3.6e-8 alebo 8.56e9. Podrobnejšie sa reálnym číslam venujeme v časti 5.2.
STR (string)	Reťazce znakov sú ľubovoľné znaky, ohraničené úvodzovkami. Dĺžka reťazca je prakticky obmedzená až kapacitou pamäti počítača. S obmedzením dĺžky reťazca sa stretneme pri načítavaní pomocou funkcie **getstring**. Podrobnejšie sa reťazcom znakov venujeme v časti 5.3.
LIST (zoznam)	Zoznam je množina prvkov ohraničená zátvorkami, pričom samotné prvky sú oddelené medzerou. Typickým príkladom zoznamu v AutoCADe sú súradnice bodu, napr. (2.3232 18.3902 0.0000). Podrobnejšie sa reťazcom znakov venujeme v časti 5.4.
PICKSET (selection sets)	Výberovou množinou je skupina 1 a viac objektov, s ktorými vieme ďalej jednoducho pracovať, napr filtrovať objekty podľa vybranej vlastnosti (druh objektu, hladina a podobne). Výberovým množinám je venovaná kapitola 10.2.
ENAME (entity name)	Meno entity (objektu) je číselné sústave) označenie objektu vo výkrese. Pomocou tohto označenia vieme pristupovať priamo k objektu v databáze výkresu a čítať alebo meniť jeho vlastnosti, podrobnejšie popísané v 10.
VLA-OBJECT (ActiveX object)	ActiveX funkcie nepracujú priamo s objektami (entitami), ale pracujú s VLA-objektami. Na konverziu entít na VLA-objekty používame funkciu jazyka Visual-LISP. Tejto oblasti sa bližšie nevenujeme, pre viac informácií odporúčame napr. [11] alebo [4].
FILE (file descriptor)	Pri čítaní alebo zapisovaní do súborov potrebujeme poznať adresu súboru s ktorým pracujeme. V AutoLISPe sa táto adresa nazýva deskriptor súboru. Práci so súbormi je venovaná časť 9.
SYM (symbol)	Symboly v AutoLISPe používame (aj) na označenie premenných, pričom premenné rozdeľujeme na tie, ktoré definoval užívateľ, systémové premenné a premenné operačného systému, pozri časť 5.1

V nasledujúcich častiach tejto kapitoly si bližšie predstavíme jednotlivé dátové typy najmä prostredníctvom funkcií, ktoré s daným typom pracujú.

5.1 Symboly a premenné

V AutoLISPe sa pojem symbol používa na (symbolické) označenie premenných alebo funkcií. Napríklad symbol **+** sa používa na označenie funkcie na sčítanie, symbol **pi** označuje premennú, v ktorej je uložená hodnota konštanty π alebo systémová premenná, ktorá riadi aktívne uchopovacie body pri kreslení v AutoCADe sa označuje symbolom **OSMODE**. To, že ide naozaj o symboly si môžeme overiť pomocou funkcie **type**, v ktorej budú tieto symboly jediným argumentom. Samotnému symbolu musí vo volaní funkcie **type** predchádzať znak **'**, ktorým zabránime AutoLISPu vo vyhodnotení symbolu, inými slovami dosadeniu jeho hodnoty:

```
Command: (type '+)
SYM
Comma nd: (type 'pi)
SYM
Command: (type 'OSMODE)
SYM
Type a command
```

Tip

Symbolom môžeme priradiť aj funkciu. Takéto priradenie sa nazýva definovanie funkcie a podrobnejšie je popísané v časti 6.1 .

Užívateľom definovaná premenná Symbol môžeme priradiť v podstate k čomukoľvek - číslu, reťazcu znakov (text) alebo bodu. Symboly sa môžu skladať z ľubovoľného znaku, okrem znakov "**(**", "**)**", "**.**", "**'**", "**"**" a "**;**" a z pochopiteľných dôvodov sa nemôžu skladať výhradne z číslic. Pri pomenovaniach sa nerozlišujú veľké a malé písmená, takže symboly **meno**, **MENO** a **mEnO** sú z pohľadu AutoLISPu totožné. Nepísaným, ale veľmi dôležitým pravidlom je, že názvy symbolov treba voliť rozumne, teda aby bolo stručné, ale výstižné.

Priradením hodnoty k symbolu funkciou **set** získame užívateľom definovanú premennú. Vytvorme teda premennú **cislo** priradením hodnoty **10** symbolu **cislo**:

```
Command: (set 'cislo 10)
10
Type a command
```

Teraz už môžeme používať premennú **cislo** ako vstup do ďalších funkcií, prípadne, ak pracujeme s výrazmi AutoLISPu v príkazovom riadku AutoCADu, môžeme si hodnotu priradenú premennej aj jednoducho vypísať v príkazovom riadku, pozri časť 4.1:

```
Command: (set 'inecislo (+ cislo 5))
15
Command: !inecislo
15
Command: !cislo
10
Type a command
```

Na horeuvedenom príklade si môžeme všimnúť, že pri priraďovaní hodnoty premennej používame znak **'** (apostrof) pred názvom symbolu. Naopak, ak chceme pracovať s premennou (napr. k nej pripočítať číslo **10**), apostrof nepoužijeme a AutoLISP premennú vyhodnotí, teda použije jej hodnotu.

Tip

Vo výraze `(set 'pocet 10)` je znak `'` len skráteným zápisom funkcie `quote`. Je teda totožný s výrazom `(set (quote pocet) 10)`.

Na priradenie hodnôt do premenných môžeme použiť aj funkciu `setq`, do ktorej už nevstupuje ako prvý argument symbol s apostrofom, ale samotný symbol. Teda premennej `cislo` priradíme hodnotu `100` nasledovným výrazom:

```
Command: (setq cislo 100)
100
Type a command
```

Funkciu `setq` môžeme zavolať aj s viac než dvomi vstupnými argumentami. Musíme ale dodržať párny počet argumentov. Prvý z dvojice argumentov je totiž vždy symbol a druhý argument je hodnota, ktorú symbolu chceme priradiť. Namiesto troch samostatných volaní funkcie `setq` na vytvorenie premenných pre číslo, zoznam, pozri časť 5.4, a reťazec znakov:

```
Command: (setq pocet 10)
10
Command: (setq bod (list 2.0 33.4 0))
(2.0 33.4 0)
Command: (setq meno "Filip")
"Filip"
Type a command
```

môžeme zavolať funkciu `setq` iba raz:

```
Command: (setq pocet 10
(_> bod (list 2.0 33.4 0)
(_> meno "Filip"
(_> )
(2.0 33.4 0)
Type a command
```

čo je len prehľadnejšou verziou zápisu do jedného riadku:

```
Command: (setq pocet 10 bod (list 2.0 33.4 0) meno "Filip")
(2.0 33.4 0)
Type a command
```

Medzi symbolmi má špeciálne miesto trojica preddefinovaných symbolov `pi`, `T` a `nil`. Týmto symbolom (okrem `nil`) je možné priradiť aj inú hodnotu, ale dôsledne sa odporúča nerobiť to. Symbol `pi` má hodnotu približne π s pomerne veľkou presnosťou (hoci pri výpise sa vypíše len šesť desatinných miest). Symbol `nil` má hodnotu nil a nedá sa zmeniť. Tento symbol má veľa významov a preto sa s ním aj často budeme stretávať, napríklad pri nepravdivých logických operáciách. Symbol `T` má hodnotu tzv. „non-nil", teda opak hodnoty `nil`.

Systémová premenná Pojmom systémová premenná sa označujú premenné, ktoré sú systémové, teda zabudované v AutoCADe. K hodnotám premenných vieme pristupovať pomocou funkcie `getvar`. Jediným argumentom tejto funkcie je názov premennej, ktorý musí byť v úvodzovkách ako reťazec znakov:

```
Command: (getvar "FILEDIA")
1
Type a command
```

alebo môžeme využiť funkciu `quote`, resp. jej skrátenú verziu:

```
Command: (getvar (quote FILEDIA))
1
Command: (getvar 'FILEDIA)
1
Type a command
```

Systémové premenné majú väčšinou celočíselné hodnoty, ale nájdu sa aj premenné ktoré majú ako hodnotu textový reťazec (napr. **HPLAYER**), reálne číslo (napr. **LTSCALE**) alebo súradnice bodu (napr. **HPORIGIN**).

Tip

Systémové premenné môžu byť buď globálne nastavené pre celý AutoCAD, alebo lokálne pre konkrétny výkres. Zistiť, ktorá systémová premenná je globálna a ktorá lokálna môžeme napríklad pomocou príkazu **SYSVDLG**.

Nastavovať hodnoty systémových premenných vieme pomocou funkcie **setvar**. Avšak premenným nemôžeme priradiť ľubovoľnú hodnotu, ale iba jednu z povolených, resp. z povoleného rozsahu hodnôt. Pri pokuse o priradenie nepovolenej hodnoty, alebo nesprávneho dátového typu sa nám vráti chybové hlásenie. Napríklad systémová premenná **FILEDIA** akceptuje len dve hodnoty, a to **0** a **1**. Ak sa jej pokúsime priradiť akúkoľvek inú hodnotu, nebudeme úspešní, pre hodnotu mimo rozsahu povolených hodnôt:

```
Command: (setvar 'FILEDIA 2)
; error: AutoCAD variable setting rejected: "FILEDIA" 2
Type a command
```

Volanie, v ktorom nastavíme hodnotu premennej **FILEDIA** na povolenú hodnotu je v poriadku a zmenia hodnotu systémovej premennej, čo má za následok zmenu správania sa AutoCAD. Pri zmene systémovej premennej **FILEDIA** na hodnotu **0** sa potlačia niektoré dialógové okná, napr. pre otvorenie alebo uloženie výkresu a bude nutné pracovať iba s príkazovým riadkom:

```
(setvar 'FILEDIA 0)
0
OPEN
Enter name of drawing to open <.>:
```

Tip

Pred zmenou hodnoty systémovej premennej vo vlastnej funkcii je dobré zálohovať jej pôvodnú hodnotu. Bežne potrebujeme kvôli ovládaniu AutoCADu cez AutoLISP zmeniť štandardné nastavenie, aby náš kód v jazyku AutoLISP prebehol hladko. Po vykonaní potrebných procesov sa ako posledný krok obnovujú zmenené systémové premenné na ich pôvodné (zálohované) hodnoty.

Premenná operačného systému Okrem systémových premenných využíva AutoCAD aj premenné operačného systému, do ktorých ukladá niektoré nastavenia. Na rozdiel od systémových premenných sa však všetky ukladajú do registra.

Tip

V názvoch premenných operačného systému sa, na rozdiel od systémových premenných alebo premenných definovaných užívateľom, rozlišujú malé a veľké písmená.

Pristupovať k hodnotám premenných môžeme prostredníctvom funkcie **getenv**, pričom jej jediným argumentom je názov premennej, ktorý musí byť výhradne vo forme reťazca znakov. Hodnotu premennej `Background`, ktorá určuje farbu pozadia modelového priestoru, získame nasledovne:

```
(getenv "Background")
0
(getenv 'Background)
; error: bad argument type: stringp BACKGROUND
Type a command
```

Hodnoty premenných operačného systému môžeme meniť pomocou funkcie **setenv**. Oba argumenty musia byť reťazce znakov, prvý reťazec je premenná, druhý reťazec je priraďovaná hodnota:

```
Command: (setenv "Background" "121479")
"121479"
Type a command
```

5.2 Celé a reálne čísla

Celé číslo v pamäti zaberá 32 bitov, a teda ponúka miesto pre 2^{32} čísel. Zvyčajne potrebujeme mať možnosť zapisovať kladné aj záporné celé čísla, jeden bit sa preto použije pre znamienko a ostáva nám 2^{31} kombinácií pre kladné i záporné čísla. Celé číslo tak môže byť iba v intervale od **-2 147 483 648** do **2 147 483 647**. Prekročenie týchto hraníc má za následok tzv. pretečenie, ktoré vyústi do nesprávneho výsledku, napr:

```
Command: (type 2147483647)
INT
Command: (type (+ 1 2147483647))
INT
Command: (+ 1 2147483647)
-2147483648
Type a command
```

Preto ak potrebujeme pracovať s väčšími hodnotami, je bezpečnejšie pracovať s reálnymi číslami:

```
Command: (type 2147483647.)
REAL
Command: (type (+ 1 2147483647.))
REAL
Command: (+ 1 2147483647.)
2.14748e+09
Type a command
```

Reálne čísla poskytujú neporovnateľne vyššiu presnosť (najmenej 14 signifikantných číslic), pričom môžu byť zapísané v rôznych formách napr. **-1.3**, **123.**, **3.6e-8** alebo **8.56e9** (resp. **8.56e+9**). Z čísel sa vždy vypisuje len šesť platných číslic, preto sme v horeuvedenom príklade nedostali výpis **2147483648.**, ale **2.14748e+09**, hoci výsledok je uložený správny (nezaokrúhlený). Overiť si to môžeme napríklad takto:

```
Command: (setq a (+ 1 2147483647.))
2.14748e+09
Command: (- a 2147483648)
0.0
Type a command
```

Na celé i reálne čísla môžeme aplikovať tie isté matematické funkcie. Celé i reálne čísla dokonca môžeme ako argumenty týchto funkcií kombinovať. Základné aritmetické funkcie v AutoLISPe sú: **+**, **-**, ***** a **/**. Tu treba upozorniť na rozdiel vo výsledku delenia celých a reálnych čísel. V prípade delenia celých čísel dostaneme výsledok delenia zaokrúhlený nadol. Ak je aspoň jeden argument funkcie reálne číslo, dostávame nezaokrúhlený výsledok:

```
Command: (/ 5 2)
2
Command: (/ 5 2.0)
2.5
Command: (/ 5.0 2)
2.5
Command: (/ 5.0 2.0)
2.5
Type a command
```

Všetky hore uvedené funkcie akceptujú dva a viac argumentov, čo znamená, že aj nasledujúce výrazy sú v poriadku:

```
Command: (- 10 5)
5
Command: (- 10 5 20 -10.5)
-4.5
Command: (/ 10 3)
3
Command: (/ 10. 3)
3.33333
Command: (/ 100 12 3)
2
Command: (/ 100 12. 3)
2.77778
Type a command
```

Ukázali sme, že pri celočíselnom delení prichádzame o zvyšok. Ak by nás zvyšok po takomto delení zaujímal, môžeme použiť funkciu **rem**. Do tejto funkcie môžeme poslať iba dva argumenty:

```
Command: (- 10 5)
5
Command: (rem 10 10)
0
Command: (rem 10 3)
1
Type a command
```

Medzi ďalšími aritmetickými funkciami nájdeme funkcie **1+** a **1-**, ktoré sa často využívajú pri cykloch, pozri časť 8. Tieto funkcie vrátia hodnotu argumentu zvýšenú, resp. zníženú o **1**. Argumentom môžu byť celé i reálne čísla:

```
Command: (- 10 5)
5
Command: (1+ 10)
11
Command: (1- 3.4)
2.4
Type a command
```

Pretypovanie reálnych čísel na celé čísla a naopak zabezpečujú funkcie **float** a **fix**. Pri pretypovaní reálneho čísla na celé treba pamätať na to, že funkcia nezaokrúhľuje, ale vždy „zahodí" časť čísla za desatinnou čiarkou a vráti celé číslo:

```
Command: (float 3)
3
Command: (fix 3.2)
3
Command: (fix 3.9)
3
Command: (fix -3.9)
-3
Type a command
```

Spomedzi ďalších funkcií vyberáme funkciu **abs**, ktorá vráti absolútnu hodnotu argumentu:

```
Command: (abs 1)
1
Command: (abs -4.03)
4.03
Command: (abs 0)
0
Type a command
```

AutoLISP má zabudované aj funkcie **min** a **max** na nájdenie najmenšieho, resp. najväčšieho čísla spomedzi ľubovoľného počtu argumentov:

```
Command: (min 2 32 -2.4 183)
-2.4
Command: (max 2 32 -2.4 183)
183
Type a command
```

Na výpočet mocnín, odmocnín a logaritmov sú v AutoLISPe zabudované funkcie **exp** a **expt**, **sqrt** a funkcia **log**. Do funkcie **expt** vstupujú dva argumenty, ostatné funkcie akceptujú iba jeden argument:

```
Command: (exp 0)
1
Command: (exp 1)
2.71828
Command: (exp 1)
2.71828
Command: (expt 2 3)
8
Command: (expt 10 4)
1000
Command: (sqrt 100)
10.0
Command: (log 1)
0.0
Type a command
```

Na záver si ešte ukážeme goniometrické funkcie **sin**, **cos** a **atan**, ktoré akceptujú jeden argument, a to uhol v radiánoch:

```
Command: Command: (sin (/ pi 2))
1.0
Command: (atan 0)
0.0
Command: (cos 3.14)
-0.999999
Type a command
```

5.3 Reťazce znakov

Reťazec znakov vieme vytvoriť veľmi jednoducho obklopením znakov dvojicou úvodzoviek. V reťazci môžu byť ľubovoľné znaky, niektorým špeciálnym znakom však musí predchádzať znak spätnej lomky "\". Napr. znaky "\", "_", "'" musia byť v reťazci zapísané nasledovne: "\\", "_", "\'".

V reťazcoch znakov môžeme využiť aj riadiace znaky, pričom najčastejšie sa využívajú znaky "\n" a "\t". Znak "\n" vloží nový riadok a znak "\t" vloží tabulátor. V nasledujúcej ukážke pomocou funkcie **princ**, pozri časť 6.3.2, vypíšeme niekoľko reťazcov znakov s riadiacimi znakmi:

```
Command: (princ "ID\tNazov\tCena\n001\tpero\t2\n002\tceruza\t1\n")
ID   Nazov    Cena
001  pero     2
002  ceruza   1
"ID\tNazov\tCena\n001\tpero\t2\n002\tceruza\t1\n"
Type a command
```

Pri textových reťazcoch AutoLISP rozlišuje malé a veľké písmená. Pre zmenu veľkosti písmen používame funkciu **strcase** s jedným voliteľným argumentom. Pokiaľ je tento argument vynechaný alebo je **nil**, funkcia vráti textový reťazec so všetkými písmenami veľkými. Naopak, ak je argument iný ako

nil, napr. **T**, funkcia vráti textový reťazec so všetkými písmenami malými:

```
Command: (strcase "Prvy pokus")
"PRVY POKUS"
Command: (strcase "Prvy pokus" nil)
"PRVY POKUS"
Command: (strcase "Prvy pokus" T)
"prvy pokus"
Type a command
```

Spájať viaceré reťazce znakov do jedného vieme pomocou funkcie **strcat**. Argumentami funkcie sú samotné reťazce a ich počet je ľubovoľný:

```
Command: (strcat "Prvý pokus" " o spojenie reťazca.")
"Prvý pokus o spojenie reťazca."
Command: (strcat "jeden" "dva" "tri")
"jedendvatri"
Command: (strcat)
""
Type a command
```

Ak by sme naopak potrebovali reťazec orezať na menší reťazec, použijeme funkcie **substr**. Dvomi povinnými argumentami funkcie sú samotný reťazec a poradové číslo znaku, ktorým chceme nový reťazec začať:

```
Command: (substr "Prvý pokus" 6)
"pokus"
Type a command
```

Funkcia **substr** má však aj voliteľný argument ktorým vieme zadať počet znakov nového reťazca:

```
Command: (substr "Prvý pokus" 1)
"Prvý pokus"
Command: (substr "Prvý pokus" 1 4)
"Prvý"
Type a command
```

Pre zistenie dĺžky textového reťazca (počtu znakov v reťazci) používame funkciu **strlen**. Počet argumentov je voliteľný, pričom funkcia zráta spolu znaky vo všetkých reťazcoch:

```
Command: (strlen "Prvý pokus")
10
Command: (strlen "jeden" "dva" "tri")
11
Type a command
```

Na triedenie zoznamu (pozri časť 5.4), ktorý obsahuje len textové reťazce môžeme použiť funkciu **acad_strlsort**. Jediný argument funkcie je samotný zoznam. Táto funkcia triedi zoznamy abecedne:

```
Command: (acad_strlsort '("ahoj" "pokus" "A" "4" "a" " " "text" ""))
("" " " "4" "a" "A" "ahoj" "pokus" "text")
Type a command
```

Rozdelenie reťazca na menšie reťazce pomocou nejakého deliaceho znaku, napr. medzery, si môžeme urýchliť použitím funkcie **ACET-STR-TO-LIST**. Táto funkcia je definovaná v tzv. Express tools AutoCADu a vyžaduje dva argumenty - deliaci znak a samotný reťazec:

```
Command: (ACET-STR-TO-LIST " " "Túto vetu rozdelíme do zoznamu pomocou medzier.")
("Túto" "vetu" "rozdelíme" "do" "zoznamu" "pomocou" "medzier.")
Command: (ACET-STR-TO-LIST " z" "Túto vetu rozdelíme do zoznamu pomocou reťazca zloženého z dvoch znakov.")
("Túto vetu rozdelíme do" "oznamu pomocou reťazca" "loženého" " dvoch" "nakov.")
Type a command
```

5.4 Zoznamy

Pod pojmom zoznam v jazyku AutoLISP rozumieme množinu prvkov (atómov) ohraničenú zátvorkami, pričom prvky zoznamu sú navzájom oddelené medzerou. Typickým príkladom zoznamu v AutoCADe sú súradnice bodu, napr. `(2.3232 18.3902 0.0000)`. Zoznam však môžu tvoriť nielen reálne čísla, ale akýkoľvek iný dátový typ. Navyše, v jednom zozname môžeme použiť viacero dátových typov. Zoznamom teda je aj napr. `(2.39 420 "AutoCAD" ("toto je vnorený zoznam" x))`.

Tip

Špeciálnym typom zoznamu je prázdny zoznam `()`, ktorý je súčasne aj atómom a označuje sa aj ako `nil`.

Vytváranie zoznamu Keby sme zoznam chceli vytvoriť napríklad z atómov `0` a `"text"` jeho priamym zápisom, dostali by sme chybové hlásenie:

```
Command: (0 "text")
; error: bad function: 0
Type a command
```

AutoLISP totiž očakáva na prvom mieste za zátvorkou názov funkcie. Náš výraz teda vyhodnotil tak, že voláme funkciu s názvom `0` a jedným argumentom `"text"`. Ak chceme, aby sa na výraz `(0 "text")` pozeral ako na zoznam, musíme zabrániť jeho vyhodnoteniu. To znamená, že musíme uvedený zoznam použiť ako argument funkcie `quote`, resp. skrátene pomocou apostrofu:

```
Command: (quote (0 "text"))
(0 "text")
Command: '(0 "text")
(0 "text")
Type a command
```

Ďalší spôsob, ako vytvoriť zoznam, je pomocou funkcie `list`, ktorá vytvorí zoznam zo všetkých jej argumentov:

```
Command: (list 0 "text")
(0 "text")
Type a command
```

Tieto dva spôsoby vytvárania zoznamov ale nepracujú celkom rovnako. Môžeme to ilustrovať na príklade, kedy zoznam vytvárame z premenných namiesto konštánt. Pri použití funkcie `list` sa premenné vyhodnotia pri vytváraní zoznamu, použitím funkcie `quote` získame zoznam nevyhodnotených premenných:

```
Command: (setq a 0 b "text")
"text"
Command: (setq zozList (list a b))
(0 "text")
Command: (setq zozQuote (quote (a b)))
(A B)
Type a command
```

Vidíme, že symboly v zozname **zozQuote** sú stále nevyhodnotené. Nedošlo totiž iba k skopírovaniu hodnôt v premenných **a** a **b**, ako pri vytváraní zoznamu **zozList**. Preto je **zozQuote** zoznamom vždy aktuálnych hodnôt premenných **a** a **b**. To znamená, že ak zmeníme hodnotu premennej **a**, zmena sa prejaví aj v zozname **zozQuote**, nie však v zozname **zozList**. Porovnajme teraz prvé prvky týchto zoznamov pred a po zmene hodnoty v premennej **a**. V nasledujúcej ukážke získame funkciou `car` prvý prvok zoznamu a funkciou `eval` vyhodnotíme symboly:

```
Command: (car zozList)
0
Command: (car zozQuote)
A
Command: (eval (car zozQuote))
0
Command: (setq a 1)
1
Command: (eval (car zozQuote))
1
Command: (car zozList)
0
Type a command
```

Tip

Zoznam je možné vytvoriť aj pomocou funkcie **cons**. Využíva sa ale najmä v cykloch, pretože nejde o priame vytváranie zoznamov, ale o postupné skladanie zoznamu z atómov. Napr. **(cons "text" nil)** vytvorí zoznam s jedným atómom **("text")**, alebo výrazom **(cons 0 (cons "text" nil))** získame zoznam dvoch atómov **(0 "text")**.

Prístup k prvkom zoznamu K prvkom zoznamu môžeme pristupovať viacerými funkciami, od funkcií **car**, **cdr** a ich kombinácií, cez funkciu **nth** až po funkciu **last**.

Pokiaľ pracujeme so zoznamami, ktoré obsahujú len niekoľko málo atómov, môžeme využiť funkcie **car** a **cdr**. Ilustrujme to na príklade trojprvkového zoznamu **(x y z)**. Ak by sme použili funkciu **car**, vrátila by nám prvý atóm v zozname, teda prvú súradnicu. Nanešťastie, prístup k ďalším súradniciam už taký priamočiary nie je. Funkcia **cdr** vráti pôvodný zoznam bez prvého atómu. Postupným použitím funkcií **cdr** a **car** teda dostaneme druhú súradnicu. Tretiu súradnicu zasa dvojnásobným použitím funkcie **cdr** a nakoniec získame z jedno-atómového zoznamu atóm funkciou **car**:

```
Command: (setq bod '(x y z))
(X Y Z)
Command: (car bod)
X
Command: (car (cdr bod))
Y
Command: (car (cdr (cdr bod)))
Z
Type a command
```

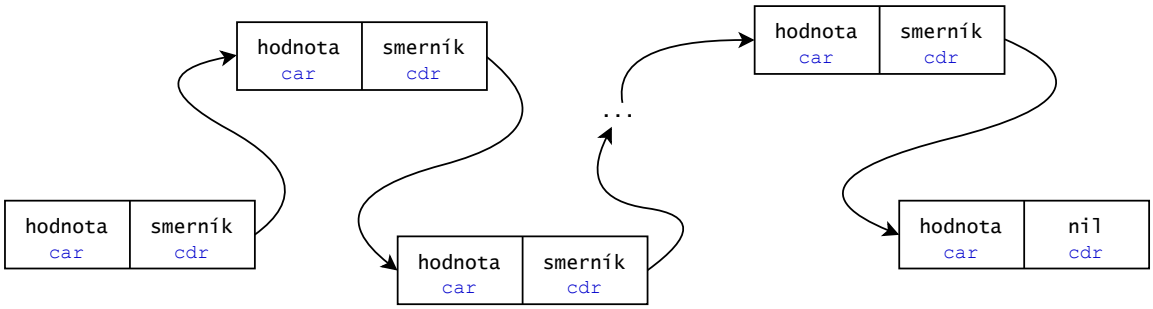
Takýto prístup k atómom je vhodný pre krátke zoznamy. Pre dlhšie zoznamy alebo viac rozmerné zoznamy sa tento postup stáva príliš komplikovaným a zdĺhavým.

Tip

Názov funkcie **car** je skratkou z anglického „Contents of Address Register", čo môžeme voľne preložiť ako obsah na adrese. Podobne aj funkcia **cdr** je skratkou z anglického „Contents of Decrement Register", teda obsah zvyšku [9]. Tieto funkcie ilustruje aj obrázok 5.1.

Mierne zjednodušenie prinášajú funkcie, poskladané z postupného aplikovania funkcií **car** alebo **cdr**, pozri tabuľku 5.2. Majme štvorcovú maticu so štyrmi riadkami a štyrmi stĺpcami (zoznam štyroch štvorprvkových zoznamov), na ktorú budeme postupne volať rôzne kombinácie týchto funkcií a uvedieme aj príslušnú výslednú funkciu, pozri tabuľku 5.2:

```
Command: (setq matica '((11 12 13 14) (21 22 23 24) (31 32 33 34) (41 42 43 44)))
((11 12 13 14) (21 22 23 24) (31 32 33 34) (41 42 43 44))
Command: (car (cdr (car matica)))
12
Command: (cadar matica)
12
Command: (car (cdr (cdr (car matica))))
13
Command: (caddar matica)
13
Command: (car (cdr (cdr (cdr (car (cdr (cdr (cdr matica))))))))
44
Command: (cadddr (cadddr matica))
44
Type a command
```



Obr. 5.1: Ilustrácia vnútornej štruktúry zoznamu, ktorý sa skladá z bodka-dvojíc, pozri časť 5.4.1. Bodka-dvojica v zozname pozostáva zo samotnej hodnoty a smerníka na ďalšiu bodka-dvojicu zoznamu [9].

Tabuľka 5.2: Prehľad kombinácií funkcií **car** a **cdr** a výsledná funkcia, ktorou možno dané kombinácie nahradiť

Funkcie **car** a **cdr**	Výsledná funkcia
(car (car x))	(caar x)
(car (cdr x))	(cadr x)
(cdr (car x))	(cdar x)
(cdr (cdr x))	(cddr x)
(car (car (car x)))	(caaar x)
(car (car (cdr x)))	(caadr x)
(car (cdr (car x)))	(cadar x)
(car (cdr (cdr x)))	(caddr x)
(cdr (car (car x)))	(cdaar x)
(cdr (car (cdr x)))	(cdadr x)
(cdr (cdr (car x)))	(cddar x)
(cdr (cdr (cdr x)))	(cdddr x)
(car (car (car (car x))))	(caaaar x)
(car (car (car (cdr x))))	(caaadr x)

Funkcie **car** a **cdr**	Výsledná funkcia
(car (car (cdr (car x))))	(caadar x)
(car (car (cdr (cdr x))))	(caaddr x)
(car (cdr (car (car x))))	(cadaar x)
(car (cdr (car (cdr x))))	(cadadr x)
(car (cdr (cdr (car x))))	(caddar x)
(car (cdr (cdr (cdr x))))	(cadddr x)
(cdr (car (car (car x))))	(cdaaar x)
(cdr (car (car (cdr x))))	(cdaadr x)
(cdr (car (cdr (car x))))	(cdadar x)
(cdr (car (cdr (cdr x))))	(cdaddr x)
(cdr (cdr (car (car x))))	(cddaar x)
(cdr (cdr (car (cdr x))))	(cddadr x)
(cdr (cdr (cdr (car x))))	(cdddar x)
(cdr (cdr (cdr (cdr x))))	(cddddr x)

Vidíme, že použitie funkcií podľa tabuľky 5.2 síce sprehľadňuje kód, na druhú stranu nie je ľahké nájsť správnu funkciu. Práve pri komplikovanom prístupe k atómom je vhodné použiť funkciu **nth**. Funkcia **nth** potrebuje dva argumenty. Prvým je poradie atómu, ktorý má funkcia vrátiť zo zoznamu, ktorý je druhým argumentom. Pomocou tejto funkcie sa dá k atómom zoznamu pristupovať oveľa intuitívnejšie. Treba však myslieť na to, že sa atómy indexujú od nuly. Atómy matice z predchádzajúceho príkladu by

sme získali volaním:

```
Command: (nth 1 (nth 0 matica))
12
Command: (nth 2 (nth 0 matica))
13
Command: (nth 3 (nth 3 matica))
44
Type a command
```

Na prístup k atómom môžeme využiť aj funkciu **last**, ktorá vracia posledný atóm zoznamu a je jej jediný argument:

```
Command: (last matica)
(41 42 43 44)
Command: (last (last matica))
44
Type a command
```

Pripájanie prvkov k zoznamu Pre pridanie nových atómov do už existujúceho zoznamu môžeme využiť funkcie **cons** a **append**. Funkciou **cons** vieme priradiť nový atóm na začiatok zoznamu:

```
Command: (cons 0 '(1 2 3 4))
(0 1 2 3 4)
Command: (cons '(8 7 6) '(1 2 3 4))
((8 7 6) 1 2 3 4)
Type a command
```

Funkcia **append** priradí nové atómy na koniec zoznamu. Argumentami funkcie ale nie sú atóm a zoznam, ale dva zoznamy. To znamená, že ak chceme pripojiť len jeden atóm, musíme z neho najskôr vytvoriť zoznam:

```
Command: (append '(8 7 6) '(1))
(8 7 6 1)
Command: (append '(0 8) '(1 2 3 4))
(0 8 1 2 3 4)
Type a command
```

Ďalšie funkcie na prácu so zoznamom Pri práci so zoznamom je dobré, obzvlášť pri cykloch, vedieť dĺžku zoznamu, alebo inak povedané počet prvkov zoznamu. Využiť na to môžeme funkciu **length**, ktorej argumentom je zoznam a vráti dĺžku tohto zoznamu:

```
Command: (length '(1 2 10))
3
Command: (length '(1))
1
Command: (length '())
0
Type a command
```

Ak by sme chceli zistiť dĺžku zoznamu, v ktorom je vnorený ďalší zoznam, funkcia **length** počíta s vnoreným zoznamom ako s 1 prvkom:

```
Command: (length '(1 2 (10 3 4 5)))
3
Type a command
```

Ak by sme hľadali konkrétny prvok v zozname, použijeme funkciu **member**, ktorej argumentom je

hľadaný prvok a zoznam, v ktorom prvok hľadáme. Ak sa prvok v zozname nájde, funkcia vráti zvyšok zoznamu začínajúci hľadaným prvkom:

```
Command: (member 3 '(1 2 10 3 4 5))
(3 4 5)
Command: (member 0 '(1 2 10 3 4 5))
nil
Command: (member 3 '(1 2 10 (3 4 5)))
nil
Type a command
```

Funkcia **reverse** jednoducho otočí poradie atómov v zozname. Jediným argumentom funkcie je teda samotný zoznam:

```
Command: (reverse '(1 2 3 4))
(4 3 2 1)
Type a command
```

Funkcia **subst** nahradí v zozname daný atóm za iný. Argumenty funkcie sú teda tri, prvý je atóm, ktorým chceme nahradiť atóm, ktorý je druhým argumentom a tretím argumentom je zoznam, v ktorom atómy nahrádzame:

```
Command: (subst 8 1 '(0 1 2 1 4 1))
(0 8 2 8 4 8)
Command: (subst 8 7 '(0 1 2 1 4 1))
(0 1 2 1 4 1)
Type a command
```

5.4.1 Bodka-dvojica

Samostatné miesto medzi zoznamami patrí tzv. bodka-dvojiciam (z angl. dotted pair). Bodka-dvojica sa od klasického zoznamu vizuálne líši tým, že má medzi atómami bodku. Ak chceme pochopiť, čo vlastne bodka-dvojica je, musíme detailnejšie popísať, čo je to zoznam.

Zoznam je skupina takých bodka-dvojíc, v ktorých prvé miesto patrí hodnote a druhé miesto zaberá smerník na ďalšiu bodka-dvojicu, pozri obr. 5.1. Ako príklad uveďme zoznam **(1 2)**, ktorý je zložený z dvoch bodka-dvojíc: **(1 . (2 . nil))**. Prvá bodka-dvojica je zložená z hodnoty **1** a smerníka (smerník je adresa v pamäti), na ďalšiu bodka-dvojicu. Hodnota v tejto ďalšej bodka-dvojici je **2** a na mieste pre smerník na ďalšiu bodka-dvojicu je **nil**, inými slovami smerník neukazuje nikam. Vďaka tomu AutoLISP vie, že **2** je posledný atóm zoznamu. Symbolicky by sme teda zoznam **(1 2)** mohli zapísať pomocou bodka-dvojíc nasledovne **(1 . (2 . nil))**. To, že sú tieto výrazy naozaj totožné, môžeme overiť nasledovne:

```
Command: (eval (quote '(1 . (2 . nil))))
(1 2)
Type a command
```

V bodka-dvojici ale môžeme obe miesta obsadiť hodnotami, napr. **(1 . 2)**. Takáto bodka-dvojica sa nemá kam rozrastať, keďže miesto pre smerník je obsadené hodnotou. Využitie bodka-dvojíc je veľké a stretneme sa s nimi napríklad pri práci s entitami, pozri časť 10, kde sú takýmto spôsobom ukladané vlastnosti entít.

Bodka-dvojicu vieme vytvoriť pomocou funkcie **cons**, ktorú sme už spomínali skôr v tejto kapitole pri vytváraní, resp. pripájaní prvkom k zoznamu. Tam bol prvým argumentom nový atóm v zozname a druhým bol zoznam. Ak sú ale oba argumenty atómy, výsledkom je práve bodka-dvojica:

```
Command: (cons 1 2)
(1 . 2)
Type a command
```

Ilustrujme teraz prácu s bodka-dvojicami, kde prvá z dvojice bude nejaká vlastnosť a druhá bude hodnota. Zoznam uložme do premennej **zoz**:

```
Command: (setq zoz (list (cons 'meno "Albert") (cons 'vyska 180) (cons 'vaha 90)))
((MENO . "Albert") (VYSKA . 180) (VAHA . 90))
Type a command
```

V takomto zozname vieme k jednotlivým bodka-dvojiciam pristupovať pomocou funkcie **assoc** s dvomi argumentami. Prvým argumentom je hodnota na prvom mieste bodka-dvojice (vlastnosť), ktorú hľadáme a druhým argumentom je zoznam bodka-dvojíc:

```
Command: (assoc 'meno zoz)
(MENO . "Albert")
Command: (assoc 'vyska zoz)
(VYSKA . 180)
Command: (assoc 'vaha zoz)
(VAHA . 90)
Type a command
```

Na prístup k vlastnosti (prvej pozícii v bodka-dvojici) pristupujeme funkciou **car**, k jej hodnote (druhej pozícii v bodka dvojici) pomocou funkcie **cdr**:

```
Command: (car (assoc 'meno zoz))
MENO
Command: (cdr (assoc 'meno zoz))
"Albert"
Type a command
```

5.5 Konverzia dátových typov

Konverziu dátových typov môžeme potrebovať z rôznych dôvodov. Napríklad ak potrebujeme použiť funkciu, ktorá vyžaduje argument typu reťazec znakov, nemôžeme ho nahradiť číslom. Toto číslo najskôr musíme konvertovať na reťazec znakov a až ten poslať ako argument do funkcie. V tejto časti predstavíme vybrané funkcie konvertujúce reťazec znakov na celé čísla a reálne čísla (kde rozlišujeme dĺžky a uhly) a naopak.

5.5.1 Celé čísla

Konverzia medzi celými číslami a reťazcami znakov je možná pomocou funkcií **atoi** a **itoa**. Obe funkcie majú jeden vstupný argument. Funkcia **atoi** požaduje ako argument reťazec, ktorý má konvertovať na celé číslo a naopak, funkcia **itoa** požaduje celé číslo, ktoré konvertuje na reťazec znakov. Uveďme príklady použitia:

```
Command: (atoi "845")
845
Command: (atoi "-143")
-143
Command: (itoa -368)
"-368"
Type a command
```

5.5.2 Reálne čísla

Pre konverziu znakových reťazcov na reálne čísla môžeme využiť funkciu **atof**. Jediným argumentom funkcie je práve reťazec znakov:

```
Command: (atof "-3.48")
-3.48
Command: (atof "10e-8")
1.0e-07
Command: (atof "1 5/16")
1.0
Type a command
```

Všimnite si, že posledná uvedená konverzia neprebehla správne. Funkcia **atof** nedokázala spracovať reťazec **"1 5/16"**. Pre konverziu čísla v takomto formáte, musíme využiť inú funkciu. K dispozícii máme ešte dve dvojice funkcií, ktoré sa líšia v pohľade na to, čo reálne číslo reprezentuje, pričom rozlišujeme dĺžku a uhol.

Dĺžka Dĺžku na reťazec znakov konvertujeme funkciou **rtos** a na konverziu opačným smerom používame funkciu **distof**.

V AutoCADe môže byť dĺžka v 5 nasledovných módoch:

- **1** = exponenciálny,
- **2** = desatinný,
- **3** = palcový desatinný,
- **4** = palcový zlomkový,
- **5** = zlomkový.

Funkciou **distof** získame z reťazca znakov dĺžku vo forme reálneho čísla. Reťazec znakov reprezentujúci dĺžku je prvým povinným argumentom. Druhým (nepovinným) argumentom je celé číslo, ktorým definujeme mód v ktorom je zadaná dĺžka:

```
Command: (distof "10e-8" 1)
1.0e-07
Command: (distof "-3.48" 2)
-3.48
Command: (distof "1 5/16" 5)
1.3125
Type a command
```

Ak funkcii **distof** druhý argument (mód) nepošleme, bude uvažovať mód podľa systémovej premennej **LUNITS**.

Funkcia **rtos** konvertuje reálne číslo na reťazec znakov. Jediným povinným argumentom je reálne číslo, ktoré vyjadruje dĺžku. Mód dĺžky môžeme rovnako ako pri funkcii **distof** špecifikovať nepovinným argumentom. V prípade, že tento argument vynecháme, funkcia bude uvažovať mód dĺžky podľa systémovej premennej **LUNITS**. Navyše, ak mód nešpecifikujeme, prichádzame o možnosť zadať ďalší nepovinný argument, ktorým je presnosť (počet desatinných miest), s ktorou sa má dĺžka konvertovať. Konvertujme teraz dĺžky v rôznych módoch:

```
Command: (rtos 1234 1)
"1.2340E+03"
Command: (rtos 10e4 2)
"100000"
Command: (rtos 10e3 3)
"833'-4\""
Command: (rtos 10e3 4)
"833'-4\""
Command: (rtos 1.3 5)
"1 5/16"
Type a command
```

Uhol Uhol na reťazec znakov konvertujeme funkciou **angtos** a na konverziu opačným smerom používame funkciu **angtof**.

V AutoCADe môže byť uhol v 5 nasledovných módoch:

- **0** = stupne,
- **1** = stupne,minúty a sekundy,
- **2** = grady,
- **3** = radiány,
- **4** = geodetické jednotky

Funkciou **angtof** získame z reťazca znakov uhol vo forme reálneho čísla. Reťazec znakov reprezentujúci uhol je prvým argumentom. Druhým (nepovinným) argumentom je celé číslo, ktorým definujeme mód v ktorom je zadaný uhol. Výsledný uhol v reálnom čísle vyjadrený vždy v radiánoch:

```
Command: (angtof "90" 0)
1.5708
Command: (angtof "9m32'45\"" 1)
0.166606
Command: (angtof "100" 2)
1.5708
Command: (angtof "1.312" 3)
1.312
Command: (angtof "N" 4)
1.5708
Type a command
```

Ak druhý arument funkcii **angtof** nepošleme, bude uvažovať mód podľa systémovej premennej **AUNITS**.

Funkcia **angtos** konvertuje reálne číslo na reťazec znakov. Jediným povinným argumentom je reálne číslo, ktoré vyjadruje veľkosť uhla. Mód uhla môžeme rovnako ako pri funkcii **angtof** špecifikovať nepovinným argumentom. V prípade, že tento argument vynecháme, funkcia bude uvažovať mód uhla podľa systémovej premennej **AUNITS**. Navyše, ak mód nešpecifikujeme, prichádzame o možnosť zadať ďalší nepovinný argument, ktorým je presnosť (počet desatinných miest), s ktorou sa má uhol konvertovať. Konvertujme teraz uhol **3.14** do všetkých možných formátov uhla:

```
Command: (angtos 3.14 0 7)
"179.9087477"
Command: (angtos 3.14 1 7)
"179d54'31.492\""
Command: (angtos 3.14 2 7)
"199.8986085g"
Command: (angtos 3.14 3 7)
"3.14r"
Command: (angtos 3.14 4 7)
"N 89d54'31.492\" W"
Type a command
```

5.6 Vstup od užívatela

Vstup od užívateľa je veľmi všeobecný pojem. Môžeme totiž očakávať vstup vo forme čísla, reťazca znakov, súradníc bodu a podobne. AutoLISP disponuje hneď niekoľkými funkciami na získanie vstupu od užívateľa, ktoré hromadne nazývame **getXXX** funkcie. Tieto funkcie sa líšia v dátovom type očakávaného vstupu. Napríklad celé číslo vráti funkcia **getint**, raálne číslo funkcia **getreal**, textový reťazec zas funkcia **getstring**. Prehľad všetkých **getXXX** funkcií nájdete v častiach 5.6.1 až 5.6.4 alebo v tabuľke 5.3.

Tip

Medzi **getXXX** funkcie sa môžu zaraďovať aj funkcie ako **entsel**, **entget** alebo **ssget**. Týmto funkciám sa venujeme v samostatnej kapitole 10.

S **getXXX** funkciami úzko súvisí funkcia **initget**, ktorá dokáže upraviť (obmedziť alebo uvoľniť) požiadavky na vstup od užívateľa. Tieto úpravy požiadavky sú vyjadrené číslami, pozri tabuľku 5.3, alebo ich kombináciami (sčítaním).

Tabuľka 5.3: Prehľad hodnôt argumentov vstupujúcich do funkcie **initget** a ich vplyv na vstup od užívateľa pri funkciách **getXXX**. Zároveň pri jednotlivých funkciách ✓ označuje možnosť použitia vybranej hodnoty. Naopak ✗ znamená, že hodnotu nemôžeme použiť. Pozn. hodnota 16 sa už nepoužíva.

Popis vplyvu na vstup do funkcie	vstup ≠ **nil**	Vstup ≠ 0	Vstup > 0	Aj mimo limitov výkresu (LIMITS)	Čiarkovaná čiara medzi bodmi	Vzdialenosť len v XY rovine	Bez obmedzení na dát. typy	Vynúti priamy vstup	Dočasné UCS v 3D rovine	Bez trasovania v smere osi Z
Hodnota	**1**	**2**	**4**	**8**	**32**	**64**	**128**	**256**	**512**	**1024**
getint	✓	✓	✓	✗	✗	✗	✓	✗	✗	✗
getreal	✓	✓	✓	✗	✗	✗	✓	✗	✗	✗
getdist	✓	✓	✓	✗	✓	✓	✓	✓	✗	✓
getangle	✓	✓	✗	✗	✓	✗	✓	✓	✗	✓
getangle	✓	✓	✗	✗	✓	✗	✓	✓	✗	✓
getpoint	✓	✗	✗	✓	✓	✗	✓	✓	✓	✓
getcorner	✓	✗	✗	✓	✓	✗	✓	✓	✓	✓
getkword	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗

Práve toto číslo je jediným argumentom funkcie **initget**, ktorá upraví požiadavky v nasledujúcom volaní vybranej **getXXX** funkcie. Ako príklad môžeme uviesť štandardné zavolanie funkcie **getint** a nasledujúci prázdny vstup od užívateľa:

```
Command: (setq cislo (getint "Zadaj cele cislo:")
Zadaj cele cislo:
nil
!cislo
nil
Type a command
```

Ak by sme ale najskôr zavolali funkciu **initget** s argumentom **1**, zabránime užívateľovi zadať prázdny vstup:

```
Command: (initget 1)
nil
Command: (setq cislo (getint "Zadaj cele cislo:"))
Zadaj cele cislo: *TU UŽÍVATEĽ ODOSLAL PRÁZDNY VSTUP*
Requires an integer value.
Zadaj celé číslo:10
10
!cislo
10
Type a command
```

Po odoslaní prázdneho vstupu AutoLISP vypíše v príkazovom riadku informáciu o tom, že užívateľ musí zadať celé číslo.

Hodnoty z tabuľky 5.3 sa dajú aj kombinovať. Ak by sme napríklad od užívateľa potrebovali kladné číslo a zároveň mu chceme zabrániť zadať prázdny vstup, sčítaním príslušných hodnôt z tabuľky 5.3 získame výslednú hodnotu **5**, ktorú odošleme ako argument funkcii **initget**:

```
Command: (initget 5)
nil
Command: (setq cislo (getint "Zadaj kladne cele cislo:"))
Zadaj kladne cele cislo: *TU UŽÍVATEĽ ODOSLAL PRÁZDNY VSTUP*
Requires a positive integer.
Zadaj kladne cele cislo:-5
Value must be positive.
Zadaj kladne cele cislo:5
5
Type a command
```

5.6.1 Číselný vstup

Ak od užívateľa potrebujeme číslo, máme na výber hneď päticu funkcií, ktorými ho môžeme získať: **getint**, **getreal**, **getdist**, **getangle**, **getorient**. To, ktorú funkciu použiť, závisí na tom, čo so získaným číslom chceme ďalej robiť, prípadne ako chceme, aby užívateľ číslo zadal.

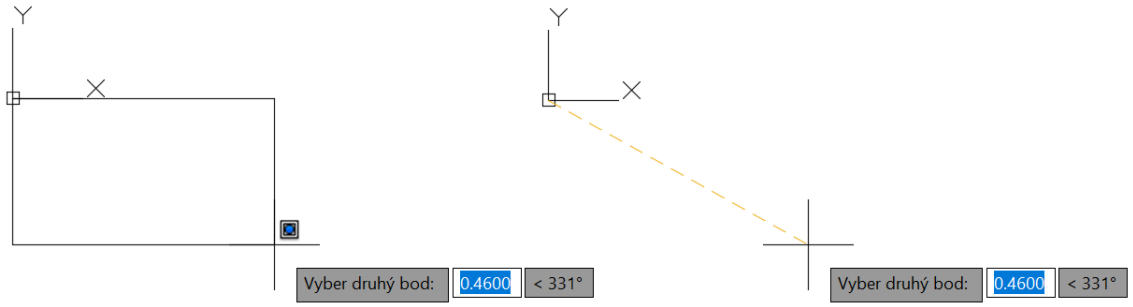
Všetky uvedené funkcie majú len nepovinné argumenty. Funkcie **getint** a **getreal** na získanie celého, resp. desatinného čísla majú len jeden nepovinný argument, ktorým je výzva na zadanie hodnoty. Pri týchto funkciách musí užívateľ zadať hodnotu pomocou klávesnice. Pri funkciách **getdist**, **getangle** a**getorient** je prvým nepovinným argumentom zoznam (súradnice bodu), druhým argumentom je opäť výzva. Tieto funkcie umožňujú hodnotu zadať klávesnicou, ale aj výberom bodu (bodov) vo výkrese. Funkcia **getdist** vracia vzdialenosť dvoch bodov ako reálne číslo. Funkcie **getangle** a**getorient** vracajú uhol v radiánoch ako reálne číslo, pričom rozdiel je v orientácii nulového uhla. Kým pri funkcii **getorient** je nula vždy napravo (resp. na tretej hodine), pri funkcii **getangle** je uhol natočenia daný systémovou premennou **ANGBASE** a smer parametrizácie je uložený v premennej **ANGDIR**. Na funkcii **getdist** ilustrujeme niekoľko príkladov použitia spomínaných funkcií:

```
Command: (getdist '(200 20)) *TU UŽÍVATEĽ KLIKNUTÍM URČIL DRUHÝ BOD*
345.668
Command: (getdist '(200 20) "Zadajte vzdialenost\n")
Zadajte vzdialenost *TU UŽÍVATEĽ KLIKNUTÍM URČIL DRUHÝ BOD*
431.93
Command: (getdist "Zadajte vzdialenost\n")
Zadajte vzdialenost *TU UŽÍVATEĽ KLIKNUTÍM URČIL PRVÝ BOD*
Specify second point: 616.837 *TU UŽÍVATEĽ KLIKNUTÍM URČIL DRUHÝ BOD*
Command: (getdist "Zadajte vzdialenost\n")
Zadajte vzdialenost
200 *TU UŽÍVATEĽ ZADAL HODNOTU KLÁVESNICOU*
200.0
Type a command
```

5.6.2 Body

Na získanie bodu od užívateľa môžeme použiť funkcie **getpoint** a **getcorner**. Výstup oboch funkcií je zoznam so súradnicami 3D bodu. Obe funkcie môžeme zavolať s dvoma argumentami - prvým bodom a výzvou. Kým pri funkcii **getpoint** sú oba argumenty voliteľné, funkcia **getcorner** má prvý bod ako povinný argument. Rozdiel medzi týmito funkciami je aj vo vizualizácii prepojenia prvého bodu (argumentu) a druhého bodu, ktorý zadáva užívateľ. Funkcia **getcorner** toto prepojenie znázorňuje obdĺžnikom, funkcia **getpoint** čiarkovanou spojnicou, pozri obr. 5.2 a nasledovné ukážky použitia funkcií:

```
Command: (getpoint "Vyber bod:")
Vyber bod:(303.395 68.1487 0.0) *TU UŽÍVATEĽ KLIKNUTÍM URČIL BOD*
Command: (getpoint '(0 0 0 ) "Vyber druhy bod:") *TU UŽÍVATEĽ KLIKNUTÍM URČIL BOD*
Vyber druhy bod: (0.402325 -0.223012 0.0)
Command: (getcorner '(0 0 0 ) "Vyber druhy bod:") *TU UŽÍVATEĽ KLIKNUTÍM URČIL BOD*
Vyber druhy bod: (0.402325 -0.223012 0.0)
Type a command
```



Obr. 5.2: Porovnanie grafiky pri výbere bodu pomocou funkcií **getcorner** (vľavo) a **getpoint** (vpravo).

5.6.3 Reťazce znakov

Reťazce znakov od užívateľa získame pomocou funkcie **getstring**. Táto funkcia má dva voliteľné argumenty. Prvým môžeme povoliť medzery v reťazci. Treba si totiž uvedomiť, že štandardne znak medzery v príkazovom riadku potvrdzuje vstup. Pokiaľ teda očakávame v reťazci aj medzery, prvým argumentom musí byť čokoľvek iné ako **nil**, napr. **T**. Druhým argumentom môže byť výzva na zadanie reťazca. Príklad volania funkcie **getstring**:

```
Command: (getstring "Zadajte Vaše meno a priezvisko:")
Zadajte Vaše meno a priezvisko:Martin "Martin" *UŽÍVATEĽ ZADAL MEDZERU MEDZI MENOM A PRIEZVISKOM*
Command: (getstring T "Zadajte Vaše meno a priezvisko:")
Zadajte Vaše meno a priezvisko:Martin Ambroz *UŽÍVATEĽ POTVRDIL VSTUP KLÁVESOU ENTER*
"Martin Ambroz"
Type a command
```

5.6.4 Kľúčové slová

Ak chceme dať užívateľovi na výber iba z vopred daných možností, použijeme funkciu **getkword**. Táto funkcia akceptuje iba tie možnosti, ktoré umožníme pomocou funkcie **initget**.

Majme príklad, kde chceme dať užívateľovi na výber z dvoch možností: `Ano` alebo `Nie`:

```
Command: (initget 1 "Ano Nie")
nil
Command: (getkword "Rozumiete ako funguje funkcia getkword? ")
Rozumiete ako funguje funkcia getkword? A
"Ano"
Command: (getkword "Rozumiete ako funguje funkcia getkword? ")
Rozumiete ako funguje funkcia getkword? mozno
Invalid option keyword.
Rozumiete ako funguje funkcia getkword? nie
"Nie"
Type a command
```

Ak by sme chceli užívateľovi ukázať aké má možnosti na odpoveď, môžeme to urobiť v podobnej forme, ako sme na to zvyknutí pri príkazoch AutoCADu. Stačí nám upraviť text výzvy, kde treba tieto možnosti uviesť v predpísanej forme, v tomto prípade `"[Ano/Nie]"`. Po odoslaní výrazu bude mať užívateľ umožnené dokonca na jedno z kľúčových slov aj kliknúť:

```
Command: (initget 1 "Ano Nie")
nil
Command: (getkword "Rozumiete ako funguje funkcia getkword? ")
Rozumiete ako funguje funkcia getkword? [Ano Nie]
```

Tip

Nastavenie pomocou funkcie **`initget`** nefunguje správne pri volaní z príkazového riadka. Správne fungovanie zaistíme napríklad použitím funkcie v rámci funkcie **`defun`**, ktorou definujeme vlastné funkcie, pozri časť 6.1.

6 Funkcie a príkazy v AutoLISPe

V kapitole 5 sme ukázali množstvo funkcií, ktoré vieme použiť pri práci so základnými dátovými typmi. Pojmu funkcia sme pritom venovali minimum priestoru. V tejto kapitole sa preto bližšie pozrieme na funkcie, na ich fungovanie a definovanie nových funkcií. Neskôr sa pozrieme na vzťah medzi funkciou AutoLISPu a príkazom AutoCADu a povieme si niečo o transparentných príkazoch.

Štandardné funkcie AutoLISPu sa na prvý pohľad môžu javiť ako čierne skrinky. Funkcie obsahujú inštrukcie, ktorými sa riadi ich priebeh. Väčšina funkcií má v inštrukciách informáciu o tom, že majú počítať s nejakými externými informáciami, ktoré dostanú na vstupe. Týmto vstupom hovoríme argumenty. Argumenty sa funkcii posielajú pri jej použití (zavolaní). Ak dodáme funkcii správny počet argumentov, funkcia použije tieto argumenty na vykonanie svojich inštrukcií, ktoré (väčšinou) vyústia do výstupu. Výstupom môže byť číslo, textový reťazec, zoznam alebo aj vytvorenie objektov. Tu je dôležité povedať, že počet argumentov musí zodpovedať požiadavkám funkcie. Ak dodáme funkcii príliš málo alebo príliš veľa argumentov, prípadne argumenty iného dátového typu, funkcia nám vráti chybové hlásenie, napr.:

```
Command: (1+)
; error: too few arguments
Type a command
```

6.1 Definovanie funkcií

Vlastné funkcie vieme definovať pomocou funkcie **defun**, ktorá má nasledovnú syntax:

```
(defun symbol ([argumenty] [/ lokalne_symboly]) vyrazy)
```

kde

- **symbol** je názov novovytváranej funkcie, pomocou tohto symbolu ju budeme volať,
- **[argumenty]** sú argumenty novovytváranej funkcie (nie sú povinné),
- **[/ lokalne_symboly]** sú symboly, ktoré budeme používať iba v novovytváranej funkcii (nie sú povinné),
- **vyrazy** sú inštrukcie, ktoré má novovytváraná funkcia vykonať po jej zavolaní.

Definovanie funkcie bez argumentov a lokálnych premenných To najnutnejšie v definícii funkcie je jej symbol (názov) a výraz alebo výrazy, ktoré má funkcia vykonávať. Definícia takejto funkcie je veľmi jednoduchá. Ako príklad sme vytvorili funkciu s názvom **DvaMinusJedna**, ktorá nemá ani argumenty ani lokálne premenné, pozri kód 6.1. Poďme ju zavolať a pozrime sa na jej výstup:

```
Command: (DvaMinusJedna)
1
Type a command
```

AutoLISP kód 6.1: Definícia funkcie **DvaMinusJedna**

```
1  (defun DvaMinusJedna ()
2      (- 2 1)
3  )
```

Definícia funkcie s argumentami Funkcia **DvaMinusJedna** nie je príliš užitočná, pretože vždy keď ju zavoláme, dostaneme rovnaký výstup (**1**). Definujme funkciu **XminusY**, pozri kód 6.2, s podobnými inštrukciami (výrazmi). Tejto funkcii však dodáme dve čísla (argumenty) **X** a **Y**, ktoré chceme od seba odčítať. Využitie takejto funkcie je o niečo vyššie, keďže môžeme počítať rozdiel dvoch ľubovoľných čísel.

AutoLISP kód 6.2: Definícia funkcie **XminusY**

```
1  (defun XminusY (X Y)
2      (- X Y)
3  )
```

Pri volaní funkcie **XminusY** netreba zabudnúť na to, že očakáva dva argumenty:

Vidíme, že funkcia **XminusY** dokáže odčítať dve čísla, podobne ako funkcia **-**. V ďalšej funkcii preto naprogramujme niečo, čo štandardná funkcia **-** nedokáže, a síce odpočítať dva body po súradniciach. Majme dva body **A** a **B**, ktoré budú argumentami našej funkcie **odcitajBody**, pozri kód 6.3.

AutoLISP kód 6.3: Definícia funkcie **odcitajBody**

```
1    ;definicia funkcie s 2 argumentami - bodmi A a B
2   (defun odcitajBody (A B)
3       ;ulozenie rozlozenych suradnic bodov, A = (Ax Ay Az)
4       (setq Ax (car A))
5       (setq Ay (cadr A))
6       (setq Az (caddr A))
7       (setq Bx (car B))
8       (setq By (cadr B))
9       (setq Bz (caddr B))
10      ;pocitanie rozdielov po suradniciach a ulozenie tychto rozdielov
11      (setq noveX (- Ax Bx))
12      (setq noveY (- Ay By))
13      (setq noveZ (- Az Bz))
14      ;vytvorenie noveho bodu z vypocitanych rozdielov
15      (list noveX noveY noveZ)
16  )   ;ukoncenie definicie
17
```

Vieme, že body sú zoznamy zložené z troch atómov. Takže jednotlivé súradnice bodov vieme získať pomocou funkcií **car** a **cdr**, resp. funkciou **nth**. Uložme si tieto hodnoty do pomocných premenných **Ax**, **Ay**, **Az**, **Bx**, **By**, a **Bz**. Následne od seba odčítame hodnoty v korešpondujúcich si zložkách (x, y a z). Na záver vytvoríme z vypočítaných hodnôt zoznam. Keďže posledný výraz v definícii funkcii je vytvorenie

zoznamu, aj definovaná funkcia vráti tento zoznam. Novovytvorenú funkciu **odcitajBody** nasledovným vstupom:

```
Command: (odcitajBody '(10 3) '(4 5))
(6 -2)
Type a command
```

Definícia funkcie s lokálnymi premennými Poďme sa teraz pozrieť na to, aký zmysel má používať lokálne symboly pre premenné. V predchádzajúcej definícii funkcii, pozri kód 6.3, sme používali iba globálne premenné, ktoré sú aj po ukončení funkcie stále k dispozícii a môžeme pristupovať k ich hodnotám. Lokálny symbol funkcie je symbol, ktorý využívame iba v rámci danej funkcie. Môžu to byť napr. pomocné symboly využívané pri čiastkových výpočtoch. To znamená, že vo funkcii sa vytvorí miesto pre prácu s lokálnou premennou a na konci funkcie sa toto miesto uvoľní a lokálna premenná prestane byť prístupná. Hlavná motivácia používať lokálne symboly je teda udržiavanie poriadku v pamäti počítača, v premenných a v neposlednom rade aj čitateľnosť kódu.

Vypíšme hodnotu premennej **Ax**, ktorú sme už použili vo funkcii **odcitajBody**:

```
Command: !Ax
10
Type a command
```

Vidíme, že máme prístup k premennej **Ax**, ktorá by po správnosti mala byť len lokálna. Vytvorme si upravenú verziu funkcie na odčítanie bodov s názvom **odcitajBodyLokal**, pozri kód 6.4, kde budeme pracovať len s lokálnymi symbolmi. Pripomíname, že lokálne symboly musia byť od argumentov oddelené znakom "**/**".

AutoLISP kód 6.4: Definícia funkcie **odcitajBodyLokal** s využitím lokálnych symbolov

```
1  (defun odcitajBodyLokal (A B / Ax Ay Az Bx By Bz noveX noveY noveZ)
2      (setq Ax (car A))
3      (setq Ay (cadr A))
4      (setq Az (caddr A))
5      (setq Bx (car B))
6      (setq By (cadr B))
7      (setq Bz (caddr B))
8
9      (setq noveX (- Ax Bx))
10     (setq noveY (- Ay By))
11     (setq noveZ (- Az Bz))
12
13     (list noveX noveY noveZ)
14 )
```

Otestujme našu funkciu **odcitajBodyLokal** nasledovným vstupom:

```
Command: (odcitajBodyLokal '(20 0) '(8 5))
(12 -5)
Type a command
```

Pozrime sa aj na to, či máme prístup k lokálnej premennej **Ax**:

```
Command: !Ax
10
Type a command
```

Na prvý pohľad to možno vyzerá, že máme prístup aj k lokálnej premennej, opak je však pravdou. Lokálna premenná **Ax** z funkcie **odcitajBodyLokal** mala hodnotu **20**. Nám sa teraz vypísala hodnota

10, ktorá patrí globálnej premennej **Ax**, ktorú sme použili vo funkcii **odcitajBody**. Ak by sme chceli odstrániť globálnu premennú **Ax** z pamäti, stačí ju nastaviť na hodnotu **nil**.

Globálne premenné sú využívané pre informácie, ktoré si chceme uchovať aj po skončení funkcie. Typický príklad, s ktorým sa v AutoCADe stretneme veľmi často je štandardná hodnota, napr. výška textu alebo polomer zaoblenia.

6.2 Definovanie príkazov

Vytvorenie nového príkazu pre AutoCAD je veľmi podobné tvorbe novej funkcie AutoLISPu. Pri tvorbe príkazu ide vlastne o vytváranie špeciálnej funkcie, ktorú dokážeme spustiť aj ako príkaz v AutoCADe. To znamená, že pre vytvorenie príkazu sa takisto používa **defun**. Rozdiel je len v počte argumentov a názve funkcie:

- príkaz musí mať nula argumentov,
- názov príkazu musí začínať znakmi "c:".

Kým funkcie môžeme vytvárať s ľubovoľným počtom argumentov, príkaz nesmie mať žiadny argument. Pri volaní príkazov v AutoCADe predsa stačí zadať iba názov príkazu a nie všetky jeho vstupy. Pri príkazoch sa totiž prípadné vstupy od užívateľa vždy získavajú pomocou **getXXX** funkcií, pozri časť 6.3.3.

Názov funkcie, ktorú chceme používať ako príkaz, sa musí začínať znakmi "c:". Teda napríklad definovaním funkcie **c:MOJPRIKAZ** zároveň definujeme príkaz, ktorý v AutoCADe spustíme pomocou príkazu **MOJPRIKAZ**.

Vytvorme teda prikaz **ODCITAJBODY** upravením funkcie **odcitajBodyLokal**, pozri kód 6.4. Nahradíme argumenty vstupom od užívateľa (funkciou **getpoint**) a upravíme aj názov funkcie tak, aby sa dala zavolať ako príkaz, pozri kód 6.5.

AutoLISP kód 6.5: Definícia funkcie **c:ODCITAJBODY**, ktorá sa dá spustiť aj ako príkaz **ODCITAJBODY**

```
1  (defun c:ODCITAJBODY ( / A B Ax Ay Az Bx By Bz noveX noveY noveZ)
2      (setq A (getpoint "Vyberte prvy bod:\n"))
3      (setq B (getpoint "Vyberte druhy bod:\n"))
4
5      (setq Ax (car A))
6      (setq Ay (cadr A))
7      (setq Az (caddr A))
8      (setq Bx (car B))
9      (setq By (cadr B))
10     (setq Bz (caddr B))
11
12     (setq noveX (- Ax Bx))
13     (setq noveY (- Ay By))
14     (setq noveZ (- Az Bz))
15
16     (list noveX noveY noveZ)
17 )
```

Po spustení príkazu **ODCITAJBODY** postupujeme podľa výziev v príkazovom riadku a vyberieme dva body. Body môžeme vyberať zadávaním súradníc do príkazového riadku, alebo klikaním v grafickom okne AutoCADu:

```
Command: ODCITAJBODY
Vyberte prvy bod: *TU UŽÍVATEĽ URČIL BOD KLIKNUTÍM*
Vyberte druhy bod: 234,42 *TU UŽÍVATEĽ URČIL BOD ZADANÍM DO PRÍKAZOVÉHO RIADKU*
(2339.37 1695.55 0.0)
Type a command
```

Tip

Nami vytvorené príkazy stále vieme zavolať aj ako funkcie AutoLISPu. Nemá to zmysel robiť priamo v príkazovom riadku AutoCADu, ale v AutoLISPe takéto volanie môže prísť vhod.

6.3 Ovládanie príkazového riadku AutoCADu cez AutoLISP

Príkazy AutoCADu dokážeme ovládať aj prostredníctvom AutoLISPu. Vieme teda zoskupiť viacero príkazov do jedného, podobne ako pri makrách príkazov, pozri časť 3.2. Na rozdiel od makra príkazu ponúka AutoLISP oveľa viac možností, napr. vytvorenie nového príkazu alebo použitie podmienok a cyklov. Na ovládanie príkazového riadku môžeme využiť dve funkcie: **command** a **command-s**.

6.3.1 Porovnanie funkcií command a command-s

Prvou možnosťou, ktorá nám umožní posielať vstupy do príkazového riadku AutoCADu je funkcia **command**. Funkcia **command** má voliteľný počet argumentov. Môžeme teda napríklad zavolať príkaz **LINE**:

```
(command "_LINE")
_LINE
Specify first point: nil
Specify first point:
```

Vidíme, že sa zavolal príkaz **_LINE**, v ďalšom riadku sa prí výzve na zadanie prvého bodu objavilo **nil**, ktoré je návratovou hodnotou funkcie **command** a v aktívnej výzve máme (opätovnú) výzvu na zadanie prvého bodu. Okrem spúšťania príkazov, vieme použiť funkciu **command** aj na odosielanie vstupov do príkazov. Pokračujme v predchádzajúcom príklade:

```
Command: (command "_LINE")
_LINE
Specify first point: nil
Specify first point: (command "0,0")
0,0
Specify next point or [Undo]: nil
Specify next point or [Undo]:
```

Takto by sme mohli pokračovať, zadávať ďalšie body, využiť možnosť **"U"** pre krok späť alebo ukončiť aktívny príkaz (odoslaním prázdneho znaku "" alebo zavolaním funkcie **command** bez argumentov).

Horeuvedený príklad samozrejme treba chápať len ako ukážku funkcionality funkcie **command** a nie ako jej bežné využitie (nemá zmysel volať funkciu **command** z príkazového riadku AutoCADu, pretože môžeme príkaz spustiť priamo). Bežné využitie nájdeme pri vytváraní vlastných príkazov, v ktorých môžeme volať hneď niekoľko štandardných príkazov AutoCADu alebo funkcií AutoLISPu.

Dobrým zvykom je v jednom volaní funkcie **command** poslať všetky argumenty až po ukončenie celého príkazu. Na rozdiel od funckie **command-s** to nie je nutné, ale pomáha to čitateľnosti kódu. Nasledujúce kódy vykonajú to isté, rozdiel v čitateľnosti je však jednoznačný, pozri kód 6.6.

AutoLISP kód 6.6: Tri rôzne volania funkcie **command** s rovnakým výsledkom. V prvej verzii v jednom volaní funkcie **command** obslúžime celý priebeh príkazu, v druhej je delenie náhodné a v poslednej verzii voláme funkciu **command** iba raz.

```
1  ; prva verzia
2  (command "_line" "10,10" "15,5" "20,10" "25,5" "30,10" "")
3  (command "_circle" "10,20" 5)
4  (command "_circle" "30,20" 5)
5  (command "_circle" "20,15" 20)
6
7  ; druha verzia
8  (command "_line" "10,10" "15,5" "20,10" "25,5")
9  (command "30,10" "" "_circle" "10,20" 5 "_circle")
10 (command "30,20" 5 "_circle" "20,15" 20)
11
12 ; tretia verzia
13 (command "_line" "10,10" "15,5" "20,10" "25,5" "30,10" "" "_circle" "10,20" 5
14 "_circle" "30,20" 5 "_circle" "20,15" 20)
```

Ak pri využívaní funkcie **command** spravíme chybu, napr. preklep v názve príkazu, môže nastať neočakávaná situácia. Majme výraz, pomocou ktorého chceme nakresliť obdĺžnik so zaoblenými rohmi. Pomocou voľby **FILLET** v príkaze **RECTANG** nastavíme polomer zaoblenia a nakoniec zvolíme dva body definujúce obdĺžnik:

```
Command: (command "_.RECTANG" "_FILLET" 10 "100,100" "200,150")
_RECTANG
Specify first corner point or [Chamfer/Elevation/Fillet/Thickness/Width]: _FILLET
Specify fillet radius for rectangles <0.0000>: 10
Specify first corner point or [Chamfer/Elevation/Fillet/Thickness/Width]: 100,100
Specify other corner point or [Area/Dimensions/Rotation]: 200,150
Type a command
```

Úmyselne teraz spravíme chybu v názve príkazu **RECTANG** a pokúsime sa spustiť príkaz **RECTENG**:

```
Command: (command "_.RECTENG" "_FILLET" 10 "100,100" "200,150")
_RECTENG Unknown command "RECTENG". Press F1 for help.
Command: _FILLET
Current settings: Mode = TRIM, Radius = 0.0000
Select first object or [uNdo/Polyline/Radius/Trim/mUltiple]: 10
Select first object or [uNdo/Polyline/Radius/Trim/mUltiple]: 100,100
Select first object or [uNdo/Polyline/Radius/Trim/mUltiple]: 200,150
Select first object or [uNdo/Polyline/Radius/Trim/mUltiple]: nil
Select first object or [uNdo Polyline Radius Trim mUltiple]:
```

AutoCAD nerozpoznal príkaz **RECTENG**, ale ďalší vstup, **FILLET**), už rozpoznal. Namiesto voľby **FILLET** v príkaze **RECTANG** sme spustili teda príkaz **FILLET** a AutoCAD očakáva, že užívateľ vyberie objekty. My mu avšak ďalšími vstupmi dodávame len číslo a súradnice bodov. Ak na daných bodoch nenájde žiadne objekty, vyústi to do neukončeného príkazu. To znamená, že príkaz **FILLET** sa nemusí ukončiť a v prípade volania ďalších príkazov môžu vzniknúť ďalšie problémy.

Použitím funkcie **command-s** sa podobným situáciám dá predísť. Funkcia **command-s** totiž kontroluje argumenty. Pokiaľ napríklad zavoláme príkaz **RECTENG**:, ktorý neexistuje, funkcia **command-s** už ďalšie argumenty nepošle do príkazového riadku:

```
Command: (command-s "_RECTENG" "_FILLET" 10 "100,100" "200,150")
; error: Unknown (command-s) failure.
Type a command
```

Tip

Aby sme predišli ukončeniu programu pri chybovom hlásení funkcie je potrebné tieto hlásenia ošetriť. Tejto téme sa ďalej venovať nebudeme. Možné riešenia sa dajú nájsť napríklad v [5].

Funkciou **command-s** nemôžeme ani posielať ďalšie vstupy do príkazového riadku, pokiaľ je nejaký príkaz aktívny:

```
Command: (command-s "_.LINE" "10,10" "15,5")
_line
Specify first point: 10,10
Specify next point or [Undo]: 15,5
Specify next point or [Undo]: (command-s "20,10")
Can't reenter LISP.
Point or option keyword required.
Specify next point or [Undo]:
```

6.3.2 Výstup do príkazového riadku

Pri vytváraní vlastných príkazov alebo funkcií je niekedy potrebné vypísať nejaké informácie do príkazového riadku. Môžeme tak prispieť k lepšej práci a použiteľnosti vytváraného príkazu alebo funkcie.

Pre výpis do príkazového riadku AutoCADu máme k dispozícii trojicu príkazov: **princ**, **prin1** a **print**, ktoré dokážu do príkazového riadku vypísať akýkoľvek dátový typ. Tieto funkcie akceptujú nula, jeden alebo dva argumenty. Pokiaľ nedodáme funkcii argument, nič nevypíše (v nasledujúcom odseku uvedieme, ako sa to dá využiť). Prvým argumentom funkcie je samotná hodnota, ktorú chceme vypísať. Ak zavoláme funkcie iba s jedným argumentom, výpis sa zrealizuje v príkazovom riadku. Pokiaľ by sme dodali funkcii aj druhý argument, ktorým je deskriptor súboru, výpis by sa zapísal do tohto súboru, pozri časť 9.

Funkie **princ**, **prin1** a **print** sa líšia formou výpisu, a to konkrétne v dvoch detailoch:

- Funkcia **print** vkladá pred každý výpis aj znak **"\n"**, čo zaistí, že každý výpis sa začne v novom riadku. Naopak, funkcie **prin1** a **princ** vypisujú v aktuálnom riadku.
- Kým funkcie **print** a **prin1** vypisujú reťazce znakov v úvodzovkách, funkcia **princ** ich vypisuje bez nich.

Výpis uvedenou trojicou funkcií ilustrujeme nasledovnými príkladmi:

```
Command: (princ "vypis funkcie princ")(princ 231)(princ (list "a" "b" 1 4))(princ "dalsi vypis")
vypis funkcie princ231(a b 1 4)dalsi vypis"dalsi vypis"
Command: (print "vypis funkcie print")(print 231)(print (list "a" "b" 1 4))(print "dalsi vypis")
"vypis funkcie print"
231
("a" "b" 1 4)
"dalsi vypis" "dalsi vypis"
Command: (prin1 "vypis funkcie prin1")(prin1 231)(prin1 (list "a" "b" 1 4))(prin1 "dalsi vypis")
"vypis funkcie prin1"231("a" "b" 1 4)"dalsi vypis""dalsi vypis"
Type a command
```

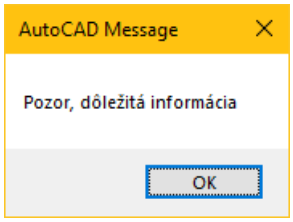
Z uvedených príkladov vidno rozdiely medzi výpismi týchto funkcií. Rovnako však vidno aj to, že posledné volanie každej z funkcií vypísalo reťazec **"dalsi vystup"** až dva krát, pričom to bolo vždy v úvodzovkách. Všetky tri funkcie totiž majú návratovú hodnotu, a tou nie je nič iné ako argument funkcie. Tento neželaný efekt nastáva vždy iba pri poslednom volanom výraze. Vyriešiť to môžeme tak, že posledný výraz bude vždy volanie jednej z uvedených funkcií bez argumentov:

```
Command: (princ "vypis funkcie princ")(princ 231)(princ (list "a" "b" 1 4))(princ "dalsi vypis")(princ)
vypis funkcie princ231(a b 1 4)dalsi vypis
Type a command
```

Tip

S neželaným výpisom sa môžeme stretnúť aj pri definovaní vlastných funkcií. Na predchádzanie týmto výpisom sa na úplnom konci definícií funkcií zvykne zavolať funkcia **princ** bez argumentov.

Pokiaľ by sme sa chceli uistiť, že užívateľ uvidí náš výpis, môžeme namiesto výpisu do príkazového riadku vytvoriť dialógové okno s výpisom. Týmto spôsobom zároveň znemožníme ďalšiu prácu v AutoCADe, až kým užívateľ okno s výpisom nezavrie. Takéto okno môžeme vytvoriť pomocou funkcie **alert**, ktorej jediným argumentom je reťazec znakov určený na výpis. Napríklad, výrazom `(alert "Pozor, dôležitá informácia")` vytvoríme dialógové okno s daným výpisom, pozri obr. 6.1.



Obr. 6.1: Dialógové okno s výpisom vytvorené pomocou funkcie **alert**

6.3.3 Získanie vstupu od užívateľa

Ak by sme chceli prenechať niektoré vstupy na užívateľa, máme dve možnosti: funkcie **getXXX**, pozri časť 6.3.3, alebo pomocou symbolu **PAUSE**. **PAUSE** je reťazec znakov s hodnotou "\\", ktorý zabezpečí, že daný vstup bude príkaz AutoCADu žiadať priamo od užívateľa.

Tip:

Namiesto **PAUSE** môžeme použiť aj priamo reťazec "\\". Vystavujeme sa tak ale potenciálnemu riziku, pretože AutoCAD v minulosti upozornil na to, že môže dôjsť k zmene reťazca **PAUSE**. Ak k tomu dôjde, priamo napísané reťazce "\\" môžu spôsobiť nefunkčnosť kódu.

Definujme príkaz **KRUZPAUSE**, ktorý využíva **PAUSE**, pozri kód 6.7 a príkaz **KRUZGETPOINT**, ktorý využíva funkciu **getpoint**, pozri kód 6.8.

AutoLISP kód 6.7: Definícia funkcie **c:KRUZPAUSE**

```
1 (defun c:KRUZPAUSE ()
2     (command "_CIRCLE" PAUSE 1)
3 )
```

AutoLISP kód 6.8: Definícia funkcie **c:KRUZGETPOINT**

```
1 (defun c:KRUZGETPOINT ()
2     (command "_CIRCLE" (getpoint "Vyber bod: ") 1)
3 )
```

Obidve funkcie vytvoria jednotkovú kružnicu v mieste, ktoré vyberie užívateľ. Rozdiel medzi nimi je vo výzve na výber bodu. Kým pri príkaze **KRUZGETPOINT** je užívateľ vyzvaný štandardnou výzvou príkazu **CIRCLE**:

```
Command: KRUZGETPOINT
_CIRCLE
Specify center point for circle or [3P 2P Ttr (tan tan radius)]: Vyber bod:
```

Pri príkaze **KRUZPAUSE** užívateľa vyzveme aj reťazcom z funkcie **getpoint**:

```
Command: KRUZPAUSE
_CIRCLE
Specify center point for circle or [3P 2P Ttr (tan tan radius)]:
```

Pri oboch príkazoch sa nám teda zobrazujú aj voľby príkazu **CIRCLE**. Ak si nejakú z nich vyberieme, berie sa to ako vstup od užívateľa. A to je ten najdôležitejší rozdiel medzi týmito príkazmi. V príkaz **KRUZPAUSE** AutoCAD urobí pauzu pre akýkoľvek užívateľský vstup. To znamená, že napriek tomu, že očakávame, že užívateľ vyberie bod, môže vybrať napríklad jednu z volieb príkazu **CIRCLE** (napr. **2P**). Následne zadaním polomeru ako ďalšieho vstupu z funkcie **command** kružnicu nevytvoríme. Príkaz **KRUZPAUSE** skončí neúspechom a príkaz **CIRCLE** sa neukončí:

```
Command: KRUZPAUSE
_CIRCLE
Specify center point for circle or [3P/2P/Ttr (tan tan radius)]: 3P
Specify first point on circle: 1
Specify second point on circle: nil
Specify second point on circle:
```

Ak by sme rovnako postupovali aj v príkaze **KRUZGETPOINT**, teda zvolili by sme voľbu príkazu **CIRCLE** (napr. **2P**) namiesto bodu, výzva na zadanie bodu sa bude opakovať. Je to preto, lebo funkcia **getpoint** kontroluje vstup a akceptuje iba body.

```
Command: KRUZGETPOINT
_CIRCLE
Specify center point for circle or [3P/2P/Ttr (tan tan radius)]: Vyber bod: 2P
Invalid point.
Vyber bod:
```

Teda ak chceme zaistiť, aby sme vytvorili kružnicu zadaním jej stredu a polomeru, pred použitím symbolu **PAUSE** uprednostníme použitie funkcie **getpoint**.

6.3.3.1 Systémové premenné a ich vplyv na priebeh funkcií

Hodnoty systémových premenných môžu mať zásadný vplyv na priebeh funkcií, ktoré využívajú príkazy AutoCADu. Pre správny priebeh užívateľom definovaných funkcií je často potrebné nastaviť hodnotu inak, ako je to bežné.

Zmenu hodnôt systémových premenných je vhodné robiť iba dočasne, a pred ukončením príkazu vrátiť premenným ich pôvodnú hodnotu. Inak povedané, najskôr sa pozrieme, ako je systémová premenná nastavená, potom ju nastavíme tak, ako potrebujeme, vykonáme, čo potrebujeme a nakoniec obnovíme pôvodnú hodnotu premennej. S týmto postupom je dobré sa stotožniť pri práci s azda všetkými systémovými premennými.

6.3.3.2 Systémová premenná CMDECHO

Pri volaní príkazov AutoCADu cez funkciu **command** sa všetky výzvy a informácie o príkazoch zobrazujú v príkazovom riadku. Ak však všetky vstupy do príkazov dodáme, či už priamo, alebo pomocou **getXXX** funkcií, je často zbytočné vypisovať všetky výstupy príkazov.

Výpis príkazov do príkazového riadku kontroluje systémová premenná **CMDECHO**. Štandardne je nastavená na hodnotu **1**, teda výpis je zapnutý. Nastavením premennej **CMDECHO** na **0** tento výpis vypneme:

Teraz zavolajme príkaz **KRUZPAUSE**:

V príkazovom riadku nevidíme žiadnu výzvu. Jedinú informáciu o tom, že príkaz je spustený a čaká na nejaký vstup od užívateľa môže užívateľ získať pri kurzore, pokiaľ má aktívne dynamický vstup (resp. systémová premenná **DYNMODE** nemá hodnotu 0). Pri príkaze **KRUZGETPOINT** takisto nevidíme výpisy príkazu **CIRCLE**, vidíme ale výzvu funkcie **getpoint**, čo užívateľovi postačuje:

Zmenou systémovej premennej **CMDECHO** sme ovplyvnili beh všetkých príkazov vytvorených v AutoLISPe. To môže viesť k nepríjemným situáciám, ako sme si ukázali pri príkaze **KRUZPAUSE**. Preto teraz vráťme hodnotu **CMDECHO** naspäť na **1** a ukážeme si, ako kontrolovať výpis príkazov v rámci samotného príkazu.

Dočasnú zmenu hodnoty systémovej premennej **CMDECHO** vieme docieliť tak, že zmeníme jej hodnotu na začiatku a na konci príkazu priamo v AutoLISPe funkciou **setvar**. Definujme funkciu **c:KRUZVYPIS**, pozri kód 6.9, len doplnením funkcie **c:KRUZGETPOINT**, pozri kód 6.8. Na začiatku funkcie si uložíme pôvodnú hodnotu premennej **CMDECHO**, potom ju nastavíme na **0**. Nasleduje volanie už skôr definovanej funkcie **c:KRUZGETPOINT** a nakoniec obnovíme pôvodnú hodnotu systémovej premennej.

AutoLISP kód 6.9: Definícia funkcie **c:KRUZVYPIS**

```
1 (defun c:KRUZVYPIS ( / CMDpovodne)
2     (setq CMDpovodne (getvar "CMDECHO"))
3     (setvar "CMDECHO" 0)
4     (c:KRUZGETPOINT)
5     (setvar "CMDECHO" CMDpovodne)
6 )
```

6.3.3.3 Systémová premenná OSMODE

Pri volaní príkazov AutoCADu cez AutoLISP môže vznikať problém s nesprávnym vykreslením objektov. Definujme jednoduchý príkaz **STVOREC**, pozri kód 6.10 na vykreslenie štvorca pomocou príkazu **LINE**. Okrem toho voláme dva krát aj príkaz **ZOOM**. Prvý krát preto, aby sme zaistili, že budeme od vykresľovaného štvorca dostatočne vzdialení a druhý krát sa priblížime na miesto kam vykresľujeme štvorec.

AutoLISP kód 6.10: Definícia funkcie **c:STVOREC**

```
1 (defun c:STVOREC ()
2     (command "_ZOOM" "_W" "0,0" "100000,100000")
3     (command "_LINE" "10,10" "10,20" "20,20" "20,10" "10,10" "")
4     (command "_ZOOM" "_W" "0,0" "30,30")
5 )
```

Pozrime sa na výpis po spustení príkazu **STVOREC** v príkazovom riadku:

```
Command: STVOREC
_ZOOM
Specify corner of window, enter a scale factor (nX or nXP), or
[All/Center/Dynamic/Extents/Previous/Scale/Window/Object] <real time>: _W
Specify first corner: 0,0 Specify opposite corner: 100000,100000
_LINE
Specify first point: 10,10
Specify next point or [Undo]: 10,20
Specify next point or [Undo]: 20,20 Zero length line created at (10.0000, 20.0000, 0.0000)
Specify next point or [Close/Undo]: 20,10
Specify next point or [Close/Undo]: 10,10 Zero length line created at (10.0000, 10.0000, 0.0000)
Specify next point or [Close/Undo]:
Command: _ZOOM
Specify corner of window, enter a scale factor (nX or nXP), or
[All/Center/Dynamic/Extents/Previous/Scale/Window/Object] <real time>: _W
Specify first corner: 0,0 Specify opposite corner: 30,30
Command:
Type a command
```

Z výpisu vidíme, že všetko neprebehlo úplne v poriadku. Pri vytváraní úsečky do bodu 20,20, resp. 10,10 AutoCAD vypísal upozornenie, že vytvoril úsečku s nulovou dĺžkou. Výsledkom príkazu **STVOREC** je nakreslenie dvoch prekrývajúcich sa úsečiek a dvoch úsečiek s nulovou dĺžkou.

Príčinu treba hľadať v uchopovaní. V závislosti od priblíženia v grafickom okne a nastavenia hodnoty systémovej premennej **OSMODE** môžeme totiž dostať dva rôzne výstupy. Ak sme dostatočne priblížení alebo je uchopovanie vypnuté, vykreslí sa štvorec. Opačný prípad je popísaný vyššie.

Priamo v AutoCADe ovládame uchopovanie prepínačom, kde si zaškrtávama rôzne uchopovacie body. Týmto zaškrtávaním meníme hodnotu systémovej premennej **OSMODE**. Jej hodnota je daná ako súčet všetkých zaškrtnutých uchopovacích bodov, ktoré majú hodnoty uvedené v tabuľke 6.1.

Tabuľka 6.1: Prehľad uchopovacích bodov a ich hodnôt v systémovej premennej **OSMODE**.

Ikona	Skratka	Popis	Hodnota
	NONe	žiadny	0
	ENDpoint	koniec	1
	MIDpoint	stred (LINE, PLINE)	2
	CENter	stred (CIRCLE, ARC)	4
	NODe	bod	8
	QUAdrant	kvadrant	16
	INTersection	priesečník	32
	INSertion	bod vloženia (BLOCK)	64
	PERpendicular	kolmica	128
	TANgent	dotyčnica	256
	NEArest	najbližšie	512
	Geometric CEnter	geometrický stred	1024
	APParent Intersection	zdanlivý priesečník	2048
	EXTension	predĺženie	4096
	PARallel	rovnobežka	8192
	Suppresses the current running object snaps	vypne uchopovanie	16384

Riešením vyššie popísaného problému je teda nastavenie hodnoty premennej **OSMODE** na **0** pred volaním príkazu **LINE**. Definujme novú funkciu **c:STVOREC2**, pozri kód 6.11, do ktorej tento krok zahrnieme. Po spustení príkazu **STVOREC2** vidíme v príkazovom riadku výpis (ktorý je v tomto príklade zbytočný a tiež ho môžeme skryť) bez upozornení na vytvorenie úsečiek nulovej dĺžky a aj výstup v podobe vytvoreného

štvorca je zaručený:

```
Command: STVOREC2
_ZOOM
Specify corner of window, enter a scale factor (nX or nXP), or
[All/Center/Dynamic/Extents/Previous/Scale/Window/Object] <real time>: _W
Specify first corner: 0,0 Specify opposite corner: 100000,100000
Command: _LINE
Specify first point: 10,10
Specify next point or [Undo]: 10,20
Specify next point or [Undo]: 20,20
Specify next point or [Close/Undo]: 20,10
Specify next point or [Close/Undo]: 10,10
Specify next point or [Close/Undo]:
Command: _ZOOM
Specify corner of window, enter a scale factor (nX or nXP), or
[All/Center/Dynamic/Extents/Previous/Scale/Window/Object] <real time>: _W
Specify first corner: 0,0 Specify opposite corner: 30,30
Command: 16385
Type a command
```

AutoLISP kód 6.11: Definícia funkcie **c:STVOREC2**

```
1 (defun c:STVOREC2 ( / OSMODEpovodne)
2     (setq OSMODEpovodne (getvar "OSMODE"))
3     (setvar "OSMODE" 0)
4     (command "_ZOOM" "_W" "0,0" "100000,100000")
5     (command "_LINE" "10,10" "10,20" "20,20" "20,10" "10,10" "")
6     (command "_ZOOM" "_W" "0,0" "30,30")
7     (setvar "OSMODE" OSMODEpovodne)
8 )
```

6.3.4 Transparentné príkazy

V AutoCAD poznáme aj tzv. transparentné príkazy. Ide o príkazy, ktoré môžeme zavolať aj počas behu iného príkazu. Typickým príkladom sú príkazy **ZOOM** a **PAN**, ktoré pomocou kolieska myši využívajú snáď všetci. Málokto ale vie, že sú za tým práve (upravené) transparentné príkazy.

Okrem týchto dvoch príkazov sa môžu ako transparentné príkazy použiť aj príkazy **CAL**, **COLOR**, **LAYER**, **LINETYPE** a mnohé iné. Okrem príkazov sa dajú transparentne meniť aj systémové premenné, či už príkazom **SETVAR**, funkciou **setvar** alebo priamo zadaním názvu premennej do príkazového riadku AutoCADu.

Príkaz vieme transparentne zavolať tak, že pred názov príkazu pridáme apostrof. Ak napríklad chceme počas spusteného príkazu **LINE** zmeniť hladinu príkazom **-LAYER**, postup bude nasledovný:

```
Command: LINE
Specify first point:
Specify next point or [Undo]:
Specify next point or [Undo]: '-_LAYER
Current layer: "0"
>>Enter an option [?/Make/Set/New/Rename/ON/OFF/Color/Ltype/LWeight/TRansparency/MATerial/Plot/Freeze/Thaw/
LOck/Unlock/stAte/Description/rEconcile/Xref]: s
>>Enter layer name to make current or <select object>: priecky
>>Enter an option
[?/Make/Set/New/Rename/ON/OFF/Color/Ltype/LWeight/TRansparency/MATerial/Plot/Freeze/Thaw/LOck/Unlock/stAte/
Description/rEconcile/Xref]:
Resuming LINE command.
LINE Specify next point or [Undo]:
```

Všimnite si, že riadky s výzvami transparentného príkazu začínajú znakmi ">>".

Transparentne dokážeme zavolať aj príkazy vytvorené v AutoLISPe. Sú tu ale značné obmedzenia, ktoré sa týkajú len funkcií obsluhujúcich príkazový riadok, teda napr. funkcie **command**. V takomto príkaze nemôžeme volať netransparentný príkaz, ktorých je drvivá väčšina. Navyše, ani všetky možnosti v ponuke transparentne zavolaných príkazov nie je vždy možné uskutočniť.

Uvádzame dva príklady definícií príkazov, ktoré je možné zavolať transparentne. Príkaz **ZC**, pozri kód 6.12, využíva transparentne zavolaný príkaz **'ZOOM**, ktorým vycentrujeme pohľad na posledný zadaný bod. Druhý príkaz, pozri kód 6.13, je príklad nastavenia systémovej premennej, ktorá definuje uhol natočenia nitkového kríža. Tento uhol je uložený v systémovej premennej **SNAPANG**. Každým zavolaním príkazu **SA** sa natočí nitkový kríž o 45°.

AutoLISP kód 6.12: Definícia funkcie **c:ZC**

```
1 (defun c:ZC ()
2     (command-s "'ZOOM" "C" "@" "")
3     (princ)
4 )
```

AutoLISP kód 6.13: Definícia funkcie **c:SA**

```
1 (defun c:SA ()
2     (setvar "SNAPANG" (+ (getvar "SNAPANG") (/ PI 4)))
3     (princ)
4 )
```

6.3.5 Úpravy štandardných príkazov AutoCADu

Ak Vám nevyhovujú štandardné príkazy AutoCADu, môžete si ich pomocou AutoLISPu zmeniť. Ku zdrojovým kódom štandardných príkazov síce neviete pristúpiť, dokážete ich ale nahradiť Vašim kódom. Použime príklad AutoCADu na úpravy príkazu **LINE**, pozri kód 6.14. V novej definícii príkazu **LINE** najskôr upozorníme užívateľa, že by mal použiť príkaz **PLINE**, ktorý mu následne spustíme.

AutoLISP kód 6.14: Redefinícia funkcie **c:LINE**

```
1 (defun C:LINE ( )
2     (princ "Pouzite prikaz PLINE!\n")
3     (command "_PLINE")
4     (princ)
5 )
```

Ak túto definíciu nahráte do AutoCADu a zavoláte príkaz **LINE**, prekvapujúco sa zavolá štandardný príkaz **LINE**. Na to, aby sme mohli používať našu definíciu príkazu **LINE**, musíme najskôr zavolať príkaz **UNDEFINE**, ktorým dokážeme odstrániť definície štandardných AutoCAD príkazov. V našom prípade teda zavoláme:

V tejto chvíli AutoCAD nepozná príkaz **LINE**. Ak doň teraz nahráme našu definíciu príkazu **LINE**, pozri kód 6.14, môžeme nový príkaz **LINE** používať:

Pomocou príkazu **UNDEFINE** sa odstránia definície funkcií len počas aktuálnej relácie AutoCADu vo všetkých jeho výkresoch. V prípade potreby sa dajú vymazané definície vrátiť späť pomocou príkazu **REDEFINE**. Postup je rovnaký ako pri príkaze **UNDEFINE**.

Štandardné príkazy AutoCADu však vieme zavolať aj keď sú ich definície vymazané. Stačí, ak pred názov príkazu dáme znak ".". Teda ak zavolám **.LINE** spustí sa štandardný príkaz **LINE**. Je preto veľmi dobrým zvykom, používať pred názvami príkazov v AutoLISPe okrem znaku "_" aj znak ".".

7 Podmienky a logické operácie

Ako sme už písali v úvodnej kapitole, pomocou programovania si môžeme zjednodušiť prácu. Ako jednu z motivácii sme uviedli možnosť zoskupiť sadu príkazov do jedného celku. Teda, namiesto niekoľkých príkazov AutoCADu by nám mohol stačiť jeden nami vytvorený príkaz, ktorý postupne vykoná jeden príkaz po druhom. Pomocou podmienok však vieme takýto program rozvetviť, čím môžeme zvýšiť napr. stabilitu alebo využiteľnosť. Uveďme niekoľko príkladov, kedy nájdu podmienky svoje uplatnenie.

Užívateľ môže zadať prázdny reťazec namiesto očakávaného celého čísla:

```
Command: (setq cislo (getint "Zadajte cele cislo: "))
Zadajte cele cislo: *TU UŽÍVATEĽ ODOSLAL PRÁZDNY REŤAZEC*
nil
Type a command
```

Problém sa prejaví, ak následne s premennou **cislo** ďalej pracujeme a neoveríme, či je v nej uložené celé číslo:

```
Command: (+ cislo 10)
; error: bad argument type: numberp: nil
Type a command
```

Tip

Mnoho prípadov nevhodného užívateľského vstupu je už ošetrených priamo vo funkciách. Napríklad funkcia **getint** môže kontrolovať, či je zadaná hodnota skutočne celé číslo. Pokiaľ užívateľ nezadá celé číslo, v príkazovom riadku sa vypíše `"Requires an integer value."` a funkcia očakáva nový vstup od užívateľa. Viac informácií o funkcii **getint** a podobných funkciách uvádzame v časti 6.3.3

Chybové hlásenie môžeme získať aj pri delení nulou, teda na príklad by sme mohli do premennej **cislo** priradiť hodnotu **0** a následne ňou deliť:

```
Command: (setq cislo (getint "Zadajte cele cislo: "))
Zadajte cele cislo: 0
0
Command: (/ 1 cislo)
; error: divide by zero
Type a command
```

Podmienku môžeme využiť aj na prepínanie medzi dvoma hodnotami alebo stavmi, pozri definície príkazov **CBPOZADIE** a **LAYOUTMODEL** na konci časti 7.1. Alebo jednoducho chceme rozvetviť náš program pridaním rôznych volieb.

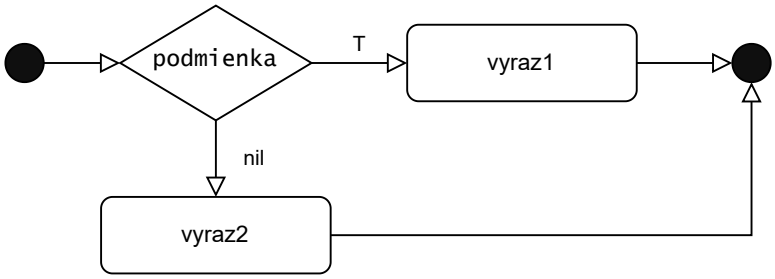
Aby program vedel správne rozhodnúť o svojej ďalšej činnosti, musíme mu nastaviť správnu podmienku. Ak sa podmienka vyhodnotí ako splnená, program vykoná dané inštrukcie. V prípade, že podmienka splnená nebude, program pôvodné inštrukcie nevykoná, ale môžme mu zadať iné inštrukcie. V AutoLISPe je podmienka splnená, ak je rôzna od **nil**. Najčastejšie je to konštanta **T**, ale rovnako sa ako

splnená podmienka vyhodnotí aj napr. `5`, `-5.3`, `"nie"`, `"nil"`, `""`.

Splnenie alebo nesplnenie podmienky vyhodnocuje podmienená funkcia. V AutoLISPe máme na výber dve takéto funkcie: **if** a **cond**.

7.1 Funkcia if

Funkcia **if** má dva povinné argumenty. Prvým je samotná podmienka, ktorá ak je rôzna od **nil**, vyhodnocuje sa výraz v druhom argumente funkcie **if**. Ak podmienka splnená nebola, nevykoná sa nič, resp. ak funkcii dodáme tretí (voliteľný) argument, vyhodnotí sa ten, pozri obr. 7.1. Syntax funkcie **if** teda vyzerá nasledovne: **(if podmienka vyraz1 [vyraz2])**, kde **podmienka** rozhoduje o tom, či sa vyhodnotí **vyraz1** (ak je podmienka splnená) alebo **vyraz2**, pričom **vyraz2** nemusíme uvádzať, ak to nepotrebujeme.



Obr. 7.1: Diagram funkcie **if**. Ak je **podmienka** splnená (**T**), vykoná a výraz **vyraz1**, ak splnená nie je (**nil**), vykoná sa výraz **vyraz2**.

Uveďme príklad použitia funkcie **if**:

```
Command: (if (getreal "Zadaj lubovolne cislo: ") (princ "Podarilo sa!"))
Zadaj lubovolne cislo: 3
Podarilo sa!"Podarilo sa!"
Command: (if (getreal "Zadaj lubovolne cislo: ") (princ "Podarilo sa!"))
Zadaj lubovolne cislo: *TU UŽÍVATEĽ ODOSLAL PRÁZDNY VSTUP*
nil
Type a command
```

Vidíme, že ak užívateľ zadal ľubovoľné číslo, vrátil sa mu zdvojený výstup `Podarilo sa!"Podarilo sa!"`. Prvý krát sa reťazec vypísal funkciou **princ**, a druhý krát (v úvodzovkách) sa vypísal ako návratová hodnota funkcie **if** (pre odstránenie zdvojeného výpisu pozri časť 6.3.2). Ak užívateľ nesplnil podmienku, funkcia **if** vrátila **nil**. Ak by sme doplnili aj výraz, na vykonanie v prípade nesplnenia podmienky, dostali by sme takýto výstup:

```
Command: (if (getreal "Zadaj lubovolne cislo: ") (princ "Podarilo sa!") (princ "Nevyslo to!"))
Zadaj lubovolne cislo: 3
Podarilo sa!"Podarilo sa!"
Command: (if (getreal "Zadaj lubovolne cislo: ") (princ "Podarilo sa!") (princ "Nevyslo to!"))
Zadaj lubovolne cislo: *TU UŽÍVATEĽ ODOSLAL PRÁZDNY VSTUP*
Nevyslo to!"Nevyslo to!"
Type a command
```

Použitie funkcie **if** je veľmi jednoduché. Problém ale nastane, ak chceme vykonať viac ako len jeden výraz (či už pri splnení alebo nesplnení podmienky). Funkcia **if** totiž akceptuje iba jeden výraz pre jednu vetvu a jeden výraz pre druhú vetvu. Vyriešiť to môžeme zoskupením viacerých výrazov do jedného funkciou **progn**. Funkcia **progn** akceptuje ľubovoľný počet argumentov. Ukážme si to na príkaze **KRUZNICAINFO**, pozri kód 7.1. Vo funkcii **c:kruznicaInfo** najskôr získame reálne číslo, ktoré následne kontrolujeme, očakávame totiž kladné reálne číslo.

AutoLISP kód 7.1: Definícia funkcie **c:kruznicaInfo**

```
(defun c:kruznicaInfo ( / polomer plocha obvod)
  (setq polomer (getreal "Zadaj polomer kruznice:" ))
  (if (> polomer 0)
    (progn
      (setq plocha (* PI polomer polomer)
            obvod  (* 2.0 PI polomer)
      )
      (princ "Plocha kruznice je ")
      (princ plocha)
      (princ " a jej obvod je ")
      (princ obvod)
      (princ ".")
    )
    (princ "Polomer musi byt vacsi ako 0!")
  )
  (princ)
)
```

Pri použití príkazu **KRUZNICAINFO** a zadaní vhodného vstupu dostaneme nasledovný výstup:

```
Command: KRUZNICAINFO
Zadaj polomer kruznice:10
Plocha kruznice je 314.159 a jej obvod je 62.8319.
Type a command
```

V prípade nevhodného vstupu, napríklad textového reťazca "sedem" zasiahne kontrolný mechanizmus funkcie **getreal**, resp. pri zápornom čísle sa príkaz ukončí s chybovým výpisom:

```
Command: KRUZNICAINFO
Zadaj polomer kruznice:sedem
Requires numeric value.
Zadaj polomer kruznice:-10
Polomer musi byt vacsi ako 0!
Type a command
```

Funkcia **if** je ideálna na rozvetvenie programu na dve vetvy. Ak potrebujeme program rozvetviť viac, môžeme funkciu **if** zavolať aj viac krát, či už postupne za sebou, alebo vnorene. Najskôr si ukážme volanie funkcie **if** za sebou, pozri kód 7.2.

AutoLISP kód 7.2: Definícia funkcie **c:IF1**

```
(defun c:IF1 ( / cislo)
  (if (setq cislo (getreal "Zadajte cislo: "))
    (princ cislo)
    (princ "Prazdny vstup")
  )
  (if (>= cislo 0)
    (princ " je kladne")
    (princ " je zaporne")
  )
  (princ)
)
```

Pri rôznych zadaných číslach dostávame rôzne výpisy príkazu **IF1**:

```
Command: IF1
Zadajte číslo: 3
3.0 je kladné
Command: IF1
Zadajte číslo: 0
0.0 je kladné
Command: IF1
Zadajte číslo: -32
-32.0 je záporné
Command: IF1
Zadajte číslo: *TU UŽÍVATEĽ ODOSLAL PRÁZDNY VSTUP*
Prázdny vstup je záporné
Type a command
```

Vidíme, že v prípade, keď užívateľ zadá záporné číslo, dostaneme zvláštny výpis `"Prazdny vstup je zaporne"`, ktorý vznikol spojením výpisu z prvého volania funkcie `if` (`"Prazdny vstup"`) a druhého volania funkcie `if` (`" je zaporne"`), keďže sa obe podmienky vyhodnotili ako nesplnené. Oveľa väčší zmysel by dávalo, keby sa vypísalo iba reťazec `"Prázdny vstup"`. Teda v prípade prázdneho vstupu nebudeme vyhodnocovať druhé volanie funkcie `if`. To dosiahneme tak, že funkcie `if` vnoríme jednu do druhej, pozri kód 7.3.

AutoLISP kód 7.3: Definícia funkcie **c:IF2**

```
1  (defun c:IF2 ( / cislo)
2    (if (setq cislo (getreal "Zadajte cislo: "))
3      (progn
4        (princ cislo)
5        (if (>= cislo 0)
6          (princ " je kladne")
7          (princ " je zaporne")
8    )
9      )
10     (princ "Prazdny vstup")
11   )
12   (princ)
13 )
```

Ďalším nemenej zaujímavým príkladom použitia funkcie `if` je pri vytváraní príkazu na prepínanie medzi dvoma stavmi, napríklad prepínanie medzi bielym a čiernym pozadím modelového priestoru príkazom **CBPOZADIE**, pozri kód 7.4, alebo prepínanie medzi kartou Model a naposledy zobrazenou kartou Layout príkazom **LAYOUTMODEL**, pozri kód 7.5.

AutoLISP kód 7.4: Definícia funkcie **c:LAYOUTMODEL**

```
1  (defun c:LAYOUTMODEL ()
2    (if (= (getvar "TILEMODE") 1)
3      (setvar "TILEMODE" 0)
4      (setvar "TILEMODE" 1)
5    )
6    (princ)
7  )
```

AutoLISP kód 7.5: Definícia funkcie **c:CBPOZADIE**

```
1 (defun c:CBPOZADIE ()
2   (if (= (getenv "Background") "16777215")
3     (setenv "Background" "0")
4     (setenv "Background" "16777215")
5   )
6   (c:LAYOUTMODEL)
7   (c:LAYOUTMODEL)
8   (princ)
9 )
```

V týchto príkazoch je len jednoduchá podmienka na kontrolu hodnoty systémovej premennej. Pokiaľ je rovná jednej z očakávaných hodnôt, zmeníme ju na druhú hodnotu a naopak. Pri funkcii **c:CBPOZADIE** sa ale môže stať, že pozadie nie je ani čierne (hodnota premennej **Background** je rovná **0**), ale ani biele (hodnota premennej **Background** je rovná **16777215**). Uvedená funkcia si ale poradí aj s takouto situáciou a po zistení, že sa podmienka nesplnila, nastaví pozadie na bielu farbu. Na záver tejto funkcie ešte voláme dva krát funkciu **c:LAYOUTMODEL** (teda prepneme sa na kartu Layout a hneď na späť na Model), aby sa zmena farby pozadia prejavila.

Tip

Na výpočet hodnoty bielej farby pre premennú **Background** sme použili vzťah:

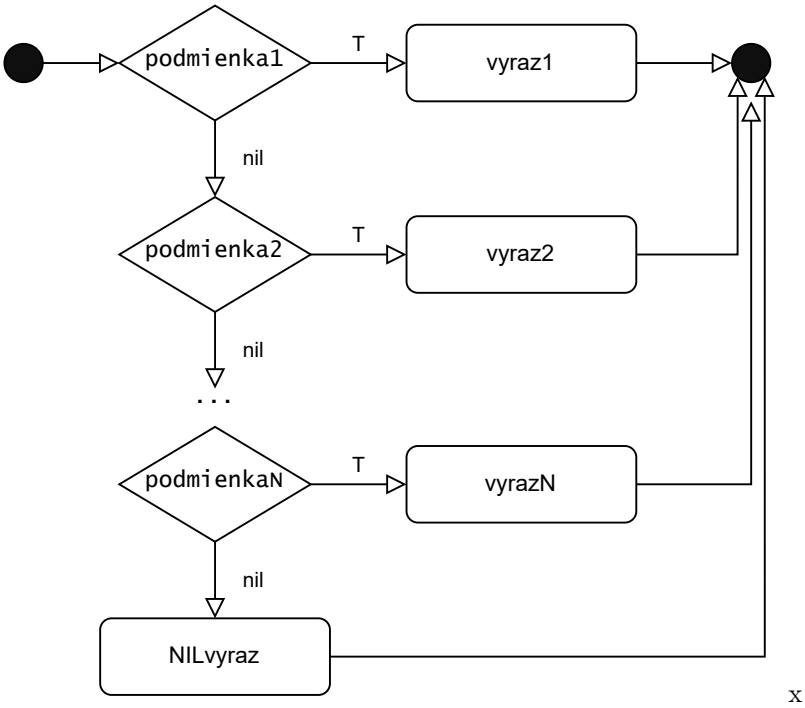
$$red + (green * 256) + (blue * 65536)$$

kde za *red*, *green* a *blue* dosadíme hodnoty požadovanej farby v intervale 0 až 255. Pre bielu farbu tak získame hodnotu 16777215, pre čiernu 0, alebo napríklad pre čisto zelené pozadie získame hodnotu 65280 [6].

7.2 Funkcia cond

Pri väčšom rozvetvovaní je vhodnejšie použiť funkciu **cond**, ktorá je pri viacerých vetvách prehľadnejšia a menej náchylná na syntaktickú chybu. Počet jej argumentov je ľubovoľný, vždy ide ale o dvojprvkové zoznamy zložené z podmienky a výrazu, ktorý sa vykoná ak je podmienka splnená. Treba mať na pamäti, že ak je nejaká podmienka vo funkcii **cond** splnená, ďalšie podmienky v tomto volaní funkcie sa už nevyhodnocujú. Počet argumentov funkcie **cond** je ľubovoľný, ale vždy to musí byť dvojprvkový zoznam, kde prvý prvok v zozname je podmienka a druhým prvkom je výraz, ktorý sa vykoná, ak je podmienka splnená, pozri obr. 7.2. Syntax funkcie **cond** je pomerne jednoduchá, po názve nasleduje ľubovoľný počet dvojprvkových zoznamov **podmienka argument**: **(cond [(podmienka vyraz) ...])**.

Uveďme najskôr jednoduchý príklad príkazu **PRACOVNYDEN**, pozri kód 7.6, ktorý má rozhodnúť, či je zadaný deň v týždni pracovný alebo nie a výsledok vypíše v príkazovom riadku. Vo funkcii **cond** rozlišujeme v tomto prípade sedem možností (vetiev) pre sedem dní v týždni. Ôsma vetva pokrýva nesprávny vstup, teda všetky prípady okrem reťazcov kontrolovaných v predchádzajúcich vetvách.



Obr. 7.2: Diagram funkcie **cond**. Pokiaľ je niektorá z podmienok splnená (**T**), vykoná sa výraz pri tejto podmienke a funkcia **cond** sa ukončí. Ak podmienka splnená nie je **nil**, nasleduje vyhodnocovanie ďalšej podmienky. Pokiaľ už nenasleduje žiadna ďalšia podmienka, vykoná sa výraz **NILvyraz** a funkcia **cond** sa ukončí.

AutoLISP kód 7.6: Definícia funkcie **c:pracovnyDen**

```
1  (defun c:pracovnyDen ( / den)
2    (setq den (getstring "Zadajte den v tyzdni (slovom, bez diakritiky): "))
3    (cond
4      ((= den "pondelok") (princ "Pondelok je pracovny den."))
5      ((= den "utorok") (princ "Utorok je pracovny den."))
6      ((= den "streda") (princ "Streda je pracovny den."))
7      ((= den "stvrtok") (princ "Stvrtok je pracovny den."))
8      ((= den "piatok") (princ "Piatok je pracovny den."))
9      ((= den "sobota") (princ "Sobota nie je pracovny den."))
10     ((= den "nedela") (princ "Nedela nie je pracovny den."))
11     (T (princ "Zadany den neexistuje!"))
12   )
13   (princ)
14 )
```

A takto potom vyzerá práca s príkazom **PRACOVNYDEN**, pričom pri rôznych vstupoch dostaneme rozličný výstup:

```
Command: PRACOVNYDEN
Zadajte den v tyzdni: sobota
Sobota nie je pracovny den.
Command: PRACOVNYDEN
Zadajte den v tyzdni: pondelok
Pondelok je pracovny den.
Command: PRACOVNYDEN
Zadajte den v tyzdni: styri
Zadany den neexistuje!
Type a command
```

Iným príkladom využitia funkcie **cond** môže byť nasledovná funkcia **c:MULTIPRIKAZ**. V tejto funkcii najskôr užívateľ najskôr vyberie nejaký objekt v AutoCADe, a ďalšie kroky závisia od toho, o aký typ objektu sa jedná. Napríklad pri výbere kružnice sa táto kružnica vymaže, pri úsečke sa táto úsečka začne kopírovať a podobne, pozri kód 7.7.

AutoLISP kód 7.7: Definícia funkcie **c:MULTIPRIKAZ**

```
(defun c:MULTIPRIKAZ ( / entita menoEntity typObjektu)
  (setq entita (entsel))
  (setq menoEntity (cdr (assoc -1 (entget (car entita)))))
  (setq typObjektu (cdr (assoc 0 (entget (car entita)))))
  (cond
    ((= typObjektu "CIRCLE") (entdel menoEntity))
    ((= typObjektu "LINE") (command "COPY" entita ""))
    ((= typObjektu "LWPOLYLINE") (command "_.EXPLODE" entita))
    ((= typObjektu "TEXT") (command "_.TEXTEDIT" entita))
    ((= typObjektu "MTEXT") (alert "multiline TEXT"))
    (T (princ "Nepodporovany objekt!"))
  )
  (princ)
)
```

7.3 Relačné operátory

Na vytvorenie podmienky často využívame relačné operátory, inak povedané funkcie na porovnávanie hodnôt. V AutoLISPe poznáme funkcie **=**, **/=** , **<**, **<=**, **>**, **>=**. Funkcia **=** má v AutoLISPe dve podobné funkcie **equal** a **eq** (rozdielom medzi týmito funkciami sa venujeme nižšie). Všetky tieto funkcie vracajú **T**, ak je výrok pravdivý a **nil** vracia ak je výrok nepravdivý. V týchto funkciách môžeme porovnávať aj celé a reálne čísla, pričom môžeme funkcie volať s dvomi a viac argumentami. Porovnávanie viacerých argumentov sa v prípade funkcií **=** a **/=** dá ľahko pochopiť. Na druhej strane, pri ostatných funkciách (**<**, **<=**, **>**, **>=**) treba dodať, že sa porovnávajú po sebe nasledujúce argumenty. Napríklad, volaním výrazu **(<** 1.0 1 3**)** zistíme, či platí nerovnosť 1.0 < 1 a zároveň 1 < 3 . Uvádzame niekoľko príkladov použitia týchto funkcií:

```
Command: (= 1.0 1)
T
Command: (= 1.0 1 3)
nil
Command: (= 1.0 1)
nil
Command: (/= 1.0 1 3)
T
Command: (/= 1.0 4 3)
T
Command: (< 1.0 1 3)
nil
Command: (< 1.0 2 3)
T
Command: (<= 1.0 1 3)
T
Type a command
```

Výnimočne ale môžeme dostať neočakávaný výsledok :

```
Command: (= 6.0 (- 8.2 2.2))
nil
Type a command
```

Táto chyba nastala kvôli binárnej reprezentácii čísel, výsledok výrazu **(- 8.2 2.2)** je v skutočnosti číslo **5.999999999999999**:

```
Command: (= 5.999999999999999 (- 8.2 2.2))
T
Type a command
```

Takáto chyba dokáže narobiť veľa problémov a je lepšie jej predchádzať. To sa dá nastavením hodnoty tolerancie na úrovni 15 desatinného miesta. Toleranciu však vo funkcii **=** nastaviť nedokážeme a preto použijeme funkciu **equal**:

```
Command: (equal 6.0 (- 8.2 2.2))
nil
Command: (equal 6.0 (- 8.2 2.2) 0.000000000000001)
T
Type a command
```

Funkcia **equal** má na rozdiel od funkcie **=** pevne daný počet porovnávaných argumentov. Argumenty môžu byť len dva, pričom tretí, voliteľný, argument je tolerancia. To však nie je jediný rozdiel medzi týmito funkciami. Funkciou **=** môžeme porovnávať čísla a znakové reťazce. Funkcia **equal** okrem toho zvládne porovnať aj zoznamy a mená entít. V úvode tejto sekcie sme spomínali aj funkciu **eq**, ktorá neporovnáva len hodnoty výrazov, ale zisťuje, či ide o identické výrazy, resp. či sú naviazané na ten istý objekt. Funkcia akceptuje iba dva povinné argumenty. Uvádzame nasledovný príklad:

```
Command: (setq z1 '(a b c))
(A B C)
Command: (setq z2 '(a b c))
(A B C)
Command: (setq z3 (list 'a 'b 'c))
(A B C)
Command: (setq z4 z2)
(A B C)
Command: (eq z1 z2)
nil
Command: (eq z1 z3)
nil
Command: (eq z1 z4)
nil
Command: (eq z2 z4)
T
Type a command
```

Vidíme, že identické sú len zoznamy **z2** a **z4**. Zoznamy **z1**, **z2** a **z3** síce obsahujú tie isté premenné, nejde o ten istý zoznam. Len pre porovnanie uvádzame, že porovnávaním týchto 4 zoznamov funkciou **equal** vráti vždy **T**.

7.3.1 Porovnávanie textových reťazcov

V neposlednom rade sa pri práci s textovými reťazcami môžeme stretnúť so zástupnými znakmi (angl. wildcard characters). Ide o špeciálne znaky, ktorými dokážeme pri viacerých reťazcoch nahradiť ich časť. Najčastejšie sa asi stretneme so znakom "*", ktorým nahrádzame ľubovoľný reťazec znakov. Takýchto znakov poznáme ale oveľa viac. V AutoLISPe dokážeme funkciou **wcmatch** nájsť podobnosť textových reťazcov (prehľad používaných zástupných znakov prinášame v tabuľke 7.1).

Funkcia **wcmatch** má dva povinné argumenty. Prvým je reťazec, ktorý chceme porovnať. Druhým je reťazec zložený zo zástupných znakov, ktoré môžu byť doplnené aj plnohodnotnými znakmi. Pokiaľ sa

Tabuľka 7.1: Prehľad zástupných znakov znakových reťazcov

Znak	Popis
#	zastúpi jednu ľubovoľnú číslicu
@	zastúpi jedno ľubovoľné písmeno abecedy
.	zastúpi jeden ľubovoľný znak okrem číslice alebo písmena abecedy
*	zastúpi ľubovoľný počet ľubovoľných znakov
?	zastúpi jeden ľubovoľný znak.
~	ak je prvý znak nezastúpi nasledujúce zástupné znaky (zastúpi ich negáciu)
[...]	zastúpi len jeden znak jedným zo znakov ohraničených zátvorkami
[~...]	zastúpi len jeden znak jedným zo znakov, ktorý nie je ohraničený zátvorkami
-	označuje rozsah znakov (len medzi zátvorkami)
,	oddeľuje dva vzory zástupných znakov
‘	únikový znak (nasledujúci znak sa neuvažuje ako zástupný znak)

nájde zhoda, funkcia vráti **T**, v opačnom prípade vráti **nil**. Uvedieme niekoľko príkladov, kde zástupnými znakmi nahradíme len jednotlivé znaky:

```
Command: (wcmatch "blok7.dwg" "blok#.dwg")
T
Command: (wcmatch "blok7.dwg" "bl@k#.dwg")
T
Command: (wcmatch"blok7.dwg" "bl@k#.dw?")
T
Type a command
```

Postupne sme využili zástupné znaky `"#"`, `"@"` a `"?"`. Medzi zástupné znaky však patrí aj bodka `"."`, ktorá zastúpi akýkoľvek znak okrem číslic a písmen, teda napríklad aj:

```
Command: (wcmatch "blok7;dwg" "bl@k#.dw?")
T
Type a command
```

Ak by sme chceli bodku vyhodnotiť ako znak a nie ako zástupný znak, použijeme únikový znak (`"'"`):

```
Command: (wcmatch "blok7;dwg" "bl@k#‘.dw?")
nil
Command: (wcmatch "blok7.dwg" "bl@k#‘.dw?")
T
Type a command
```

Zastúpiť ľubovoľný počet znakov vieme znakom `"*"`:

```
Command: (wcmatch "blok7.dwg" "blok*.dwg")
T
Command: (wcmatch "blok.dwg" "blok*.dwg")
T
Command: (wcmatch "blok7ver3.2.dwg" "blok*.dwg")
T
Type a command
```

Ak by sme chceli zistiť zhodu s akýmsi negovaným reťazcom, použijeme prázdny znak `" "`. Pod pojmom negovaný reťazec si môžme predstaviť všetko to, čo sa nedá daným reťazcom zastúpiť. Takéto porovnanie nám dá presne opačný výstup ako keby sme použili znak `"~"`:

```
Command: (wcmatch "blok7.dwg" "blok#.dwg")
T
Command: (wcmatch "blok7.dwg" "~ blok#.dwg")
nil
Command: (wcmatch "blok17.dwg" "~ blok#.dwg")
T
Type a command
```

Pomocou zátvoriek [a] vieme doteraz uvedené zástupné znaky aplikovať lokálne. Môžeme teda pre jeden znak špecifikovať viac možností:

```
Command: (wcmatch "blok7.dwg" "blok[1,6,7].dwg")
T
Command: (wcmatch "blok6.dwg" "blok[1,6,7].dwg")
T
Command: (wcmatch "blok5.dwg" "blok[1,6,7].dwg")
nil
Type a command
```

Podobne môžeme definovať možnosti rozsahom:

```
Command: (wcmatch "blok7.dwg" "blok[1-7].dwg")
T
Command: (wcmatch "blok7.dwg" "blok[1-27].dwg")
nil
Command: (wcmatch "blok17.dwg" "blok[1-27].dwg")
nil
Command: (wcmatch "blokA.dwg" "blok[a-z].dwg")
nil
Command: (wcmatch "blokA.dwg" "blok[A-Z].dwg")
T
Type a command
```

Prípadne môžeme vyčleniť nechcené možnosti:

```
Command: (wcmatch "blok7.dwg" "blok[~#].dwg")
nil
Command: (wcmatch "blok7.dwg" "blok[~7].dwg")
nil
Command: (wcmatch "blok5.dwg" "blok[~7].dwg")
T
Type a command
```

7.3.2 Overenie typu premennej

Na overenie hodnôt priradených premenným máme k dispozícii skupinu funkcií, ktoré akceptujú jeden argument a vrátia **T** alebo **nil**.

Funkciou **boundp** zisťujeme, či je premennej priradená nejaká hodnota. Argumentom funkcie je symbol premennej (symbol získame pridaním ' pred názov premennej):

```
Command: (boundp 'c)
nil
Command: (setq c 1)
1
Command: (boundp 'c)
T
Command: (boundp c) *NESPRÁVNY ARGUMENT FUNKCIE*
nil
Type a command
```

Funkcia **minusp** overí, či je zadaná hodnota záporná:

```
Command: (setq c -1)
-1
Command: (minusp c)
T
Command: (minusp -23.42)
T
Command: (minusp 23.42)
nil
Type a command
```

Funkcia **numberp** overí, či je zadaná hodnota číslom:

```
Command: (setq c 1)
10
Command: (numberp c)
T
Command: (numberp -23.42)
T
Command: (numberp "8")
nil
Type a command
```

Overiť nulovú hodnotu môžeme pomocou funkcie **zerop**:

```
Command: (setq c 0)
0
Command: (zerop c)
T
Command: (zerop 0.0)
T
Command: (zerop 3)
nil
Type a command
```

Na overenie toho, či ide o zoznam, slúži funkcia **listp**:

```
Command: (setq z (list 1 2 3))
(1 2 3)
Command: (listp z)
T
Command: (listp '(1 2 3))
T
Command: (listp 'z)
nil
Command: (listp 1.234)
nil
Type a command
```

7.4 Logické operátory

Často potrebujeme vytvoriť komplikovanejšiu podmienku, pri ktorej musíme vyhodnotiť viacero výrazov. Jednotlivé výrazy v podmienke musia byť spojené jedným z dvoch logických výrazov **and** alebo **or**. Ak použijeme **and**, na to aby bola splnená podmienka, všetky výrazy musia byť pravdivé:

```
Command: (and nil nil)
nil
Command: (and nil T)
nil
Command: (and nil T nil)
nil
Command: (and T T)
T
Type a command
```

Pri použití funkcie **or** stačí, ak bude pravdivý jeden z výrazov v podmienke:

```
Command: (or nil nil)
nil
Command: (or nil T)
T
Command: (or nil T nil)
T
Command: (or T T)
T
Type a command
```

Ako posledný logický operátor si ukážeme negáciu **not**, ktorá vráti **T** ak je pôvodný výraz nepravdivý a naopak:

8 Cykly a pokročilejšia práca so zoznamami

Vo veľkej časti programov dochádza k opakovaniu činností, čo sa väčšinou nezaobíde bez cyklov. Použitím cyklov na opakované vykonávanie výrazov získame prehľadný kód bez ohľadu na to, či sa výraz zopakuje päťkrát alebo stokrát.

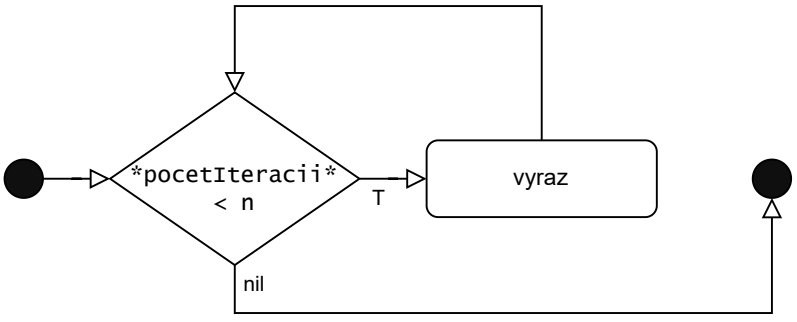
V AutoLISPe máme k dispozícii dve základné funkcie na opakovanie výrazov **repeat** a **while**. Základný rozdiel medzi týmito funkciami je, že funkcia **repeat** vykoná fixný počet opakovaní, pozri časť 8.1, zatiaľ čo funkcia **while** opakuje výrazy len pokiaľ je splnená podmienka, ktorú vo funkcii určíme, pozri časť 8.2.

```
Command: (entsel)
Select object: (<Entity name: 208afa1e4c0> (4214.71 1041.61 0.0))
Type a command
```

Pri opakovaní činností na prvkoch zoznamu môžeme využiť aj ďalšie funkcie, ako **foreach**, **apply** alebo **mapcar**, pozri časť 8.3. Výhodou týchto funkcií je kratší kód, napríklad aj preto, že nepotrebujeme vedieť počet prvkov zoznamu. Okrem týchto funkcií môžeme využiť aj rekurziu, pozri časť 8.3.4.

8.1 Funkcia repeat

Funkcia **repeat** má povinný argument celé číslo, ktorým zadáme počet opakovaní cyklu. Následne môžeme zadať ľubovoľný počet argumentov, teda výrazov, ktoré bude cyklus **repeat** opakovať daný počet krát, pozri obr. 8.1.



Obr. 8.1: Diagram funkcie **repeat**. V tejto funkcii sa kontroluje, či je počet iterácií menší, než vopred stanovený počet iterácií. Ak je podmienka splnená (**T**), vykoná sa **vyraz**. Naopak, ak už bol dosiahnutý stanovený počet iterácií, podmienka už nie je splnená (**nil**) a funkcia **repeat** sa ukončí.

Priamo v príkazovom riadku zavolajme funkciu **repeat**, ktorá trikrát vypíše daný reťazec znakov:

```
Command: (repeat 3 (princ "Dobry den!\n"))
Dobry den!
Dobry den!
Dobry den!
"Dobry den!\n"
Type a command
```

Vidíme, že sa výpis výrazom `(princ "Dobry den!\n")` vykonal 3-krát. Štvrtý výpis už nevznikol preto, že sa výpis vykonal ešte raz, ale ide o návratovú hodnotu funkcie **repeat**, resp. funkcie **princ**. Funkcia **repeat** totiž vracia vyhodnotený posledný výraz, ktorý opakuje, čo je v tomto prípade výraz `(princ "Dobry den!\n")`. Návratová hodnota funkcie **princ** je jej argument a preto sa na rozdiel od 3 opakovaných výpisov, ten štvrtý vypísal v úvodzovkách a so znakom `"\n"` na konci reťazca.

Definujme teraz funkciu **opakujtrikrat**, v ktorej upravíme cyklus tak, aby sa v každom opakovaní vypísal iný reťazec. Takisto potlačíme návratovú hodnotu funkcie **repeat**, ktorou je posledný vyhodnotený výraz, pozri kód 8.1. Zavolaním funkcie **opakujtrikrat** dostaneme:

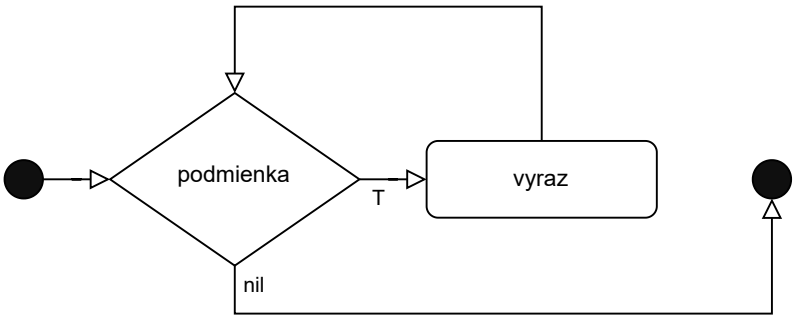
```
Command: (opakujtrikrat)
Opakovanie číslo 1!
Opakovanie číslo 2!
Opakovanie číslo 3!
Type a command
```

AutoLISP kód 8.1: Definícia funkcie **opakujtrikrat**

```
1  (defun opakujtrikrat ( / i)
2      (setq i 1)
3      (repeat 3
4          (princ (strcat "Opakovanie cislo " (itoa i) "!\n"))
5          (setq i (+ 1 i))
6      )
7      (princ)
8  )
```

8.2 Funkcia **while**

Funkciou **while** dokážeme vytvoriť sofistikovanejšie cykly, než pomocou funkcie **repeat**. Syntax funkcie je podobná ako pri funkcii **repeat**, rozdiel je iba v tom, že prvým argumentom nebude počet opakovaní, ale ľubovoľná podmienka. Následne môžeme zadať ľubovoľný počet argumentov, teda výrazov, ktoré bude cyklus **while** opakovať, kým bude podmienka splnená, pozri obr. 8.2. Cyklus **while** pred vykonaním výrazu vyhodnotí podmienku. Pokiaľ je podmienka pravdivá, nasledujúce výrazy sa vyhodnotia. Kontrola podmienky (a následné vyhodnotenie výrazov) sa opakuje až dovtedy, kým sa podmienka nevyhodnotí ako nepravdivá. Tým sa zároveň ukončí celý cyklus.



Obr. 8.2: Diagram funkcie **while**. V tejto funkcii rozhoduje **podmienka** o tom, či sa vykoná **vyraz**, alebo sa cyklus ukončí. Pokiaľ je podmienka splnená (**T**), vykoná sa **vyraz**. Ak podmienka splnená nie je (**nil**), funkcia **while** sa ukončí.

Upravme teraz funkciu **opakujtrikrat**, pozri kód 8.1, tak, že namiesto funkcie **repeat** použijeme funkciu **while**, pozri kód 8.2.

AutoLISP kód 8.2: Definícia funkcie **opakujtrikrat** s využitím funkcie **while**

```
1 (defun opakujtrikrat ( / i)
2     (setq i 1)
3     (while (<= i 3)
4         (princ (strcat "Opakovanie cislo " (itoa i) "!\n"))
5         (setq i (+ 1 i))
6     )
7     (princ)
8 )
```

Podstatný rozdiel oproti predchádzajúcej verzii príkazu **opakujtrikrat** je, že cyklus **repeat** sa zopakuje vždy fixný počet krát. Teda ak vieme, koľkokrát chceme niečo opakovať, stačí použiť jednoduchšiu a rýchlejšiu funkciu **repeat**. Niekedy ale neviem dopredu povedať, koľkokrát chceme výrazy opakovať, napríklad načítaní súboru alebo pri hľadaní nejakej hodnoty.

Vo funkcii **c:od0do100**, pozri kód 8.3, sa cyklus opakuje dovtedy, kým užívateľ nezadá číslo z intervalu **0** až **100**:

```
Command: OD0DO100
Zadajte vysledny pocet bodov (0-100): f
Requires an integer value.
Zadajte vysledny pocet bodov (0-100): 1000
Zadajte vysledny pocet bodov (0-100): 42
42
Command:
Type a command
```

AutoLISP kód 8.3: Definícia funkcie **c:od0do100**

```
1 (defun c:od0do100 ( / cislo)
2     (setq cislo nil)
3     (while (not (and (>= cislo 0)
4         (<= cislo 100))) (setq cislo (getint "Zadajte pocet bodov (0-100): "))
5     )
6 )
```

Ako ďalší príklad uvádzame funkciu **vypisDo10**, pozri kód 8.4, ktorá na vstupe dostane zoznam, z ktorého bude vypisovať prvky až kým nenarazí na prvok s hodnotou väčšou ako **10**. Otestujme funkciu nasledovnými volaniami:

```
Command: (vypisdo10 (list 8 3 4 12 4 15))
8
3
4
3
Command: (vypisdo10 (list 10 4 415))
nil
Type a command
```

AutoLISP kód 8.4: Definícia funkcie **vypisDo10**

```
1 (defun vypisDo10 (zoznam / i)
2   (setq i 0)
3   (while (< (nth i zoznam) 10)
4     (princ (nth i zoznam))
5     (princ "\n")
6     (setq i (+ i 1))
7   )
8 )
```

Použitím funkcie **while** vieme vytvoriť aj nekonečný cyklus, ktorý ukončí až užívateľ stlačením esc, pozri kód 8.5, ktorým definujeme funkciu **c:unitCircles**. Touto funkciou môžeme vytvárať jednotkové kružnice v bode, ktorý zadá užívateľ až kým vytváranie neukončíme klávesou esc.

AutoLISP kód 8.5: Definícia funkcie **unitCircles**

```
1 (defun c:unitCircles ( / bod)
2     (while T
3         (setq bod (getpoint "Specify center point for unit circle: "))
4         (command "._CIRCLE" bod 1)
5     )
6 )
```

8.3 Cykly a zoznamy

Cykly sa často používajú aj pri práci so zoznamami, pričom sa zvyčajne prechádza všetkými prvkami zoznamu.

Pri opakujúcej sa práci so zoznamami, resp. s prvkami zoznamov máme okrem základných funkcií **repeat** a **while**, pozri časti 8.1 a 8.2, k dispozícii aj ďalšie funkcie **foreach**, **apply** a **mapcar**, ktoré prinesú jednoduchšiu a teda aj prehľadnejšiu syntax.

8.3.1 Funkcia foreach

Majme funkciu **SucetPloch**, ktorou môžeme sčítať všetky prvky zoznamu. Tieto prvky môžu predstavovať plochy miestností.

Ak na definíciu našej funkcie použijeme funkciu **repeat**, pozri kód 8.6, musíme využívať pomocnú premennú **i**, ktorá bude uchovávať poradové číslo aktuálneho prvku v zozname.

Použitie funkcie **SucetPloch** v príkazovom riadku potom vyzerá nasledovne:

```
Command: (sucetploch '(10.2 2.8 33.5))
46.5
Type a command
```

Ak nahradíme funkciu **repeat** funkciou **foreach**, získame jednoduchší a prehľadnejší kód, pozri kód 8.7. Namiesto pomocnej premennej **i** totiž používame pomocnú premennú **plocha**, ktorá predstavuje vždy jednu plochu zo zoznamu plôch.

AutoLISP kód 8.6: Definícia funkcie **SucetPloch** s využitím funkcie **repeat**

```
(defun SucetPloch (zoznam / i suma)
    (setq i 0)
    (setq suma 0)
    (repeat (length zoznam)
        (setq suma (+ suma (nth i zoznam)))
        (setq i (+ i 1))
    )
  suma
)
```

AutoLISP kód 8.7: Definícia funkcie **SucetPloch** s využitím funkcie **foreach**

```
(defun SucetPloch (zoznam / suma)
    (setq suma 0)
    (foreach plocha zoznam
        (setq suma (+ suma plocha))
    )
  suma
)
```

8.3.2 Funkcia apply

Pomocou funkcie **apply** môžeme použiť vybranú funkciu na prvky zoznamu, pričom prvky zoznamu budú použité ako argumenty vybranej funkcie. Napr. môžeme použiť funkciu **+** na prvky zoznamu, v ktorom máme uložené plochy miestnosti, presne ako v kódoch 8.6 a 8.7. Použitím funkcie **apply**, aplikujeme funkciu **+** na všetky prvky zoznamu, čím získame kompaktnú verziu funkcie **SucetPloch**, pozri kód 8.8.

AutoLISP kód 8.8: Definícia funkcie **SucetPloch** s využitím funkcie **apply**

```
(defun SucetPloch (zoznam / )
    (apply '+ zoznam)
)
```

Ďalším zaujímavým príkladom využitia funkcie **apply** je spájanie zoznamu reťazcov znakov:

8.3.3 Funkcia mapcar

Použiť vybranú funkciu na jednotlivé prvky zoznamu môžeme pomocou funkcie **mapcar**. Na rozdiel od funkcie **apply**, pozri časť 8.3.2, sa prvky zoznamu nepoužijú ako argumenty vo vybranej funkcii súčasne, ale postupne a preto aj výsledok funkcie nebude jedna hodnota, ale zoznam modifikovaných prvkov.

Majme zoznam **("2.3232" "18.3902" "0.0000")**, ktorý je naplnený textovými reťazcami. Na prvý pohľad vidíme, že všetky reťazce znakov obsahujú iba čísla. Takýto zoznam môžeme dostať napr. pri načítaní zo súboru, pozri časť 9.2. Ak s týmito hodnotami chceme pracovať ako s číslami, potrebujeme ich konvertovať (pretypovať) na čísla, v tomto prípade reálne čísla. Vytvorme funkciu **konvertujZoznam**,

ktorá pomocou funkcie **foreach** prejde všetky prvky zoznamu a prekonvertované na čísla ich vloží do nového zoznamu, pozri kód 8.9.

AutoLISP kód 8.9: Definícia funkcie **konvertujZoznam** pomocou funkcie **foreach**

```
1 (defun konvertujZoznam (zoznam / Czoznam)
2     (setq Czoznam '())
3     (foreach n zoznam
4         (setq Czoznam(append Czoznam (list (atoi n))))
5     )
6     Czoznam
7 )
```

Použitím funkcie **mapcar**, môžeme aplikovať funkciu **atoi** na všetky prvky zoznamu omnoho jednoduchšie, pozri kód 8.10.

AutoLISP kód 8.10: Definícia funkcie **konvertujZoznam** pomocou funkcie **mapcar**

```
1 (defun konvertujZoznam (zoznam)
2     (mapcar 'atoi zoznam)
3 )
```

8.3.4 Rekurzia

V niektorých prípadoch môžeme namiesto vyššie spomínaných funkcií využiť rekurziu. T.j. vytvoriť funkciu, v ktorej vnútri budeme volať samotnú funkciu. Na vytvorenie rekurzívnej funkcie v AutoLISPe sa zvyčajne používa trojica funkcií: **if** na vytvorenie podmienky, **cdr** na získanie prvého prvku v zozname a funkcia **car** na získanie zvyšných prvkov v zozname.

Majme zoznam reťazcov znakov, napr. **("a" "b" "c")**. Všetky prvky v zoznam postupne vypíšeme pomocou rekurzívnej funkcie **vypisZoznamRetazcov**, pozri kód 8.11:

```
Command: (vypisZoznamRetazcov '("a" "b" "c"))
a
b
c
nil
Type a command
```

Funkciu **if** využívame na kontrolu, či zoznam nie je prázdny, Funkciou **car** získame na výpis prvý prvok zoznamu. Nakoniec pošleme do funkcie **vypisZoznamRetazcov** aktuálny zoznam bez prvého prvku, ktorý získame pomocou funkcie **cdr**.

AutoLISP kód 8.11: Definícia funkcie **vypisZoznamRetazcov**

```
1 (defun vypisZoznamRetazcov (z / )
2   (if z
3     (progn
4       (princ (car z))
5       (princ "\n")
6       (vypisZoznamRetazcov (cdr z))
7     )
8   )
9 )
```

Ďalšie príklady využitia rekurzie uvádzame vo funkciách **sucet**, pozri kód 8.12, pomocou ktorej môžeme získať súčet všetkých čísel v zozname a **naCisla**, pozri kód 8.13, ktorá prekonvertuje zoznam znakových reťazcov na zoznam reálnych čísel.

AutoLISP kód 8.12: Definícia funkcie **sucet**

```
1 (defun sucet (z / )
2   (if z
3     (+ (car z) (sucet (cdr z)))
4     0
5   )
6 )
```

AutoLISP kód 8.13: Definícia funkcie **naCisla**

```
1 (defun naCisla (z / )
2   (if z
3     (append (list (atof (car z))) (naCisla (cdr z)))
4   )
5 )
```

9 Práca so súbormi

Pomocou AutoLISPu môžeme zapisovať a čítať dáta z textového súboru, čo je možné využiť napr. pri prenose dát medzi ďalšími softvérmi.

Pri práci so súborom potrebujeme najskôr získať prístup k samotnému súboru. Prístup k súboru získame pomocou funkcie **open**, ktorá má dva povinné argumenty. Prvým argumentom je názov súboru (aj s príponou), ktorý chceme otvoriť. AutoLISP bude súbor s takýmto názvom hľadať v `C:\Users\VASE_UZIVATELSKE_MENO\Documents`. Ak sa ale súbor nachádza v inom priečinku, musíme zadefinovať aj cestu k tomuto súboru. Cestu môžeme definovať ako absolútnu (plnú) alebo relatívnu. Druhým parametrom je mód otvorenia súboru. Ide o reťazec znakov, ktorým upresníme či otvárame súbor na čítanie (**"r"**), zápis (**"w"**) alebo na pripájanie k existujúcemu súboru (**"a"**). V prípade, že daný súbor neexistuje, funkcia pri móde (**"r"**) vráti **nil**, ale pri módoch (**"w"**) a (**"a"**) vytvorí nový súbor. Uvádzame niekoľko príkladov. Majme `subor.txt` v `D:`. Postupne tento súbor otvoríme na čítanie bez zadania cesty, zadaním relatívnej cesty a nakoniec zadaním plnej cesty:

```
Command: (open "subor.txt" "r")
nil
Command: (open "../../../subor.txt" "r")
#<file "../../../subor.txt">
Command: (open "D:/subor.txt" "r")
#<file "D:/subor.txt">
Type a command
```

Tip

Pri relatívnej ceste sa vieme dostať o adresár vyššie nielen pomocou "../", ale aj pomocou "./", ".\\" alebo "..\\".

Vidíme, že funkcia **open** vracia deskriptor súboru otvoreného na čítanie. Deskriptor súboru by sme získali aj pri otváraní súborov na zápis a pripájanie. Pomocou deskriptoru budeme môcť so súborom ďalej pracovať, preto je nutné uložiť do premennej pomocou **setq**.

Tip

Pri otváraní súboru na zápis alebo pripájanie sa môže stať, že funkcia **open** vráti **nil**, aj keď súbor existuje. Dôvodom je nedostatočné povolenie na prístup k súborom (napr. na disku `C:\`).

9.1 Zápis a pripájanie do súboru

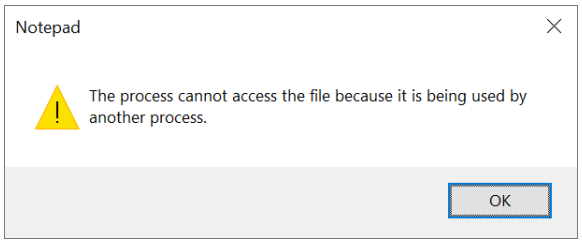
Zavolaním výrazu **(setq subor (open "D:/zapis.txt" "w"))** otvoríme (alebo ak neexistuje aj vytvoríme) na zápis súbor `zapis.txt` uložený v `D:\`. Deskriptor uložíme do premennej **subor**:

```
Command: (setq subor (open "D:/zapis.txt" "w"))
#<file "D:/zapis.txt">
Type a command
```

Na zápis do súboru môžme použiť funkciu **write-line**, ktorej argumentom bude textový reťazec a voliteľným parametrom je deskriptor súboru (pri volaní funkcie **write-line** bez deskriptoru sa reťazec vypíše do príkazového riadku). Napíšme teda do súboru niekoľko riadkov:

```
Command: (write-line "riadok 1" subor)
Command: (write-line "riadok 2" subor)
Type a command
```

Otvorme teraz súbor `zapis.txt` napríklad v Poznámkovom bloku. V súbore vidíme dva riadky s reťazcami, ktoré sme špecifikovali. Ak by sme teraz v tomto súbore urobili nejaké zmeny pomocou poznámkového bloku, nedali by sa nám uložiť, pozri obr. 9.1.



Obr. 9.1: Upozornenie pri snahe o opätovné otvorenie súboru, ktorý už máme otvorený na zápis pomocou AutoLISPu

Súbor máme totiž stále otvorený v AutoCADe a preto Poznámkový blok do tohto súboru nemôže zapisovať, kým ho v AutoCAD neuzavrieme. Z tohto dôvodu treba vždy po ukončení práce so súborom uzavrieť prístup k súboru pomocou funkcie **close**. Jediný argument funkcie **close** je deskriptor súboru. My ho máme uložený v premennej **subor**, preto zavoláme:

```
Command: (close subor)
nil
Type a command
```

Na zápis do súboru môžme použiť aj funkciu **write-char**. Na rozdiel od funkcie **write-line** je argumentom tejto funkcie len jeden znak. Tento znak však nezadávame v úvodzovkách, ale ho špecifikujeme pomocou čísla z ASCII tabuľky [3]. Tieto čísla si našťastie nemusíme pamätať, pretože funkcia **ascii** nám vráti ASCII kód znaku, ktorý do nej pošleme.

Otvorme si znovu náš súbor, tentokrát ale v móde **"a"**. Ak teda chceme do súboru pripisovať znaky po jednom, použijeme funkciu **write-char**:

```
Command: Command: (setq subor (open "D:/zapis.txt" "a"))
#<file "D:/zapis.txt">
Command: (write-char (ascii "B") subor)
66
Command: (close subor)
nil
Type a command
```

Tip

Zapisovať do súboru dokážu aj funkcie **princ**, **prin1** a **print**, pozri časť 6.3.2. Okrem samotného reťazca im treba ako argument dodať aj deskriptor súboru.

9.2 Čítanie zo súboru

Otvorme si na čítanie súbor, ktorý sme vytvorili v predchádzajúcej časti 9.1:

```
Command: (setq subor (open "D:/zapis.txt" "r"))
#<file "D:/zapis.txt">
Type a command
```

Na čítanie zo súboru sa využíva najmä funkcia **read-line**, ktorá má jediný argument – deskriptor súboru, z ktorého chceme čítať. Táto funkcia nám vráti celý riadok zo súboru ako textový reťazec:

```
Command: (read-line subor)
"riadok 1"
Command: (read-line subor)
"riadok 2"
Command: (read-line subor)
nil
Type a command
```

Vidíme, že opakovaným volaním tejto funkcie nám vracia textový reťazec z nasledujúceho riadku súboru. Ak sa dostaneme na koniec súboru, funkcia vráti **nil**. Teraz môžeme súbor uzavrieť.

Čítať zo súboru môžeme nielen po celých riadkoch, ale aj po znakoch. Funkcia **read-char** prečíta jeden znak a vráti nám jeho ASCII kód. Funkciou **chr** vieme zas premeniť ASCII kód na znak. Otvorme si opätovne náš súbor a prečítajme z neho niekoľko znakov:

```
Command: (setq subor (open "D:/zapis.txt" "r"))
#<file "D:/zapis.txt">
Command: (chr (read-char subor))
"r"
Command: (chr (read-char subor))
"i"
Command: (read-char subor)
97
Type a command
```

Prečítali sme prvé tri znaky v súbore, z toho sme na prvé dva použili funkciu **chr**, ktorá nám ASCII kódy znakov premenila na znaky, a pri treťom znaku sme vypísali jeho ASCII kód. Ak by sme takto pokračovali až na koniec riadku, vrátil by sa nám aj znak **"\n"**:

```
Command: (chr (read-char subor))
"1"
Command: (chr (read-char subor))
"\n"
Type a command
```

Funkcie **read-line** a **read-char** zdieľajú jednu polohu v načítavanom súbore. Pokračujme v čítaní druhého riadku nasledovne:

```
Command: (chr (read-char subor))
"r"
Command: (read-line subor)
"iadok 2"
Command: (chr (read-char subor))
nil
Type a command
```

Vidíme teda, že funkcia **read-line** pokračuje tam, kde **read-char** skončila a naopak.

9.3 Výber súborov

Okrem priameho zadania cesty k súboru textovým reťazcom, môžeme využiť aj funkcie **findfile** a **getfiled**. Súbor podľa jeho názvu hľadáme funkciou **findfile**, pričom názov súboru je jej jediný argument. Pri hľadaní sa prehľadávajú priečinky zaradené medzi tzv. Support File Search Path, Trusted Locations,

a Working Support File Search Path. Teda ak pridáme cestu k priečinku napr. medzi Support File Search Path, budú sa prehľadávať aj tie. Výstupom funkcie je plná cesta k súboru, ak sa súbor nenájde, vráti **nil**.

Na výber súboru klasickým dialógovým oknom, ako napr. pri vytváraní nového výkresu, používame funkciu **getfiled**. Funkcia má štyri povinné argumenty. Prvým argumentom je textový reťazec s názvom dialógového okna. Nasleduje argument, ktorým zadáme predvolený názov súboru. Oveľa častejšie sa však využíva na definovanie priečinku, ktorý sa zobrazí v dialógovom okne. Tretím argumentom je typ súboru, ktorý chceme otvoriť. Ak zadáme prázdny reťazec znakov **""** alebo **"*"** vyberáme zo všetkých typov. Zadaním jedného typu, napr. **"dwg"**, vyberáme iba tieto súbory. Ak chceme vyberať z viacerých vybraných typov, oddeľujeme tieto typy bodkočiarkou, napr. **"dwg;pdf"**. Posledným argumentom je celé číslo, ktoré definuje správanie dialógového okna. Toto číslo vyberáme kombináciou z hodnôt:

1 Vytvorenie nového súboru. Ak vyberieme existujúci súbor, zobrazí sa upozornenie.

4 Bez ohľadu na tretí argument funkcie dovolí zadať názov súboru s ľubovoľnou príponou.

8 Vyhľadá súbor s predvoleným názvom v priečinkoch. Ak ho nájde, po potvrdení vráti iba názov súboru s príponou. Ak sa súbor nenájde, výber súboru je prenechaný na užívateľa.

16 Na mieste druhého argumentu očakáva iba cestu k priečinku.

32 Pri vytváraní nového súboru neupozorní užívateľa v prípade, ak dôjde k prepísaniu existujúceho súboru.

64 Ak užívateľ zadá URL súboru, vráti sa táto cesta. V opačnom prípade by sa súbor z URL stiahol do počítača a funkcia by vrátila cestu k tomuto uloženému súboru.

128 Neakceptuje užívateľom zadanú URL súboru.

9.4 Špeciálne znaky

Až do verzie AutoCAD 2021 bolo používanie znakov s diakritikou pomerne náročné, keďže AutoLISP plne nepodporoval UNICODE znaky. Od tejto verzie však môžeme využiť plnú podporu týmto znakom pomocou systémovej premennej **LISPSYS** nastavenej na hodnotu **1**.

Tip

Správne kódovanie je dôležité, najmä ak používame znaky mimo ASCII tabuľky. Dobrým zvykom však je, používať pri programovaní iba znaky z ASCII tabuľky.

9.5 Otváranie súborov v externých programoch

Okrem otvorenia súborov na načítavanie alebo zapisovanie vo funkcii AutoLISPu môžeme otvárať aj súbory v externých programoch. Pomocou funkcie **startapp** vieme spustiť ľubovoľný externý program. Povinným argumentom tejto funkcie je plná cesta k `*.exe` súboru programu, ktorý chceme spustiť (uvádzať príponu nie je povinné):

```
Command: (startapp "C:/Windows/System32/notepad.exe")
33
Command: (startapp "C:/Windows/System32/notepad")
33
Type a command
```

Pokiaľ je program v jednom z priečinkov v premennej prostredia **"PATH"**, stačí zadať iba názov `*.exe` súboru programu bez cesty:

```
Command: (startapp "notepad.exe")
33
Command: (startapp "notepad")
33
Type a command
```

Vo všetkých horeuvedených prípadoch sa otvorí program Poznámkový blok s novým súborom. Okrem toho vrátila funkcia **startapp** hodnotu **33**. Pokiaľ otvorenie prebehne úspešne, vždy sa vráti kladné celé číslo, v opačnom prípade **nil**.

Ak by sme chceli v externom programe otvoriť existujúci súbor, zavoláme funkciu **startapp** aj s voliteľným argumentom, ktorým zadáme cestu k súboru. Pri takomto otváraní súboru máme v zásade dve možnosti:

- špecifikovať súbor a program, v ktorom otvoríme súbor,
- otvoriť súbor v predvolenom programe (predvolený v prostredí Windows).

Prvá možnosť je síce zdĺhavejšia, ale máme istotu, v akom programe otvoríme daný súbor. Nevýhodou je, že sa súbor otvorí v novej inštancii programu a tiež, že často musíme poznať zdĺhavú cestu k `*.exe` súboru programu. Otvorme napríklad nasledovné súbory (cesty k programom a súborom sa u Vás môžu líšiť):

```
Command: (startapp "C:/Program Files/Microsoft Office/Office16/WINWORD.EXE" "D:/test.docx")
33
Command: (startapp "acad.exe" "D:/test.dwg")
33
Type a command
```

Druhú možnosť využijeme najmä, ak nám stačí otvoriť súbor v akomkoľvek programe. To sa hodí napr. pri textových súboroch. Výhodou je aj to, že súbory sa otvoria v už existujúcich inštanciách programu (ak to program umožňuje). Na takéto otvorenie využijeme prieskumník, ktorý spustíme súborom `explorer.exe`:

```
Command: (startapp "explorer" "D:/test.docx")
33
Command: (startapp "explorer" "D:/test.dwg")
33
Type a command
```

Prieskumník (**"explorer.exe"**) môžeme využiť aj na otvorenie vybraného priečinka. Pri spustení bez zadania priečinka sa prieskumník otvorí v predvolenom priečinku (napr. Tento počítač). Priečinok ale môžeme určiť pomocou voliteľného argumentu, napr. priečinok `D:\` otvoríme takto:

```
Command: (startapp "explorer" "D:/mojeCAD")
33
Type a command
```

Ak zadaný priečinok neexistuje, prieskumník sa otvorí v priečinku `Dokumenty`.

10 Entity a výberové množiny

Pri práci v AutoCADe sa po celý čas stretávame s entitami a výberovými množinami. Pracujeme totiž s entitami ako napr. úsečka, kružnica, hladina alebo textový štýl, resp. pri manipulácii s objektami vytvárame výberové množiny, teda skupiny entít.

Z hľadiska programátorského je výhodné, ba až nevyhnutné pristúpiť priamo k entitám, resp. výberovým množinám. Podrobne sa prístupu k entitám (grafickým aj negrafickým) a ich popisu venujeme v časti 10.1 a práci s výberovými množinami venujeme časť 10.2.

10.1 Entity

S entitami sa v AutoCADe stretneme takpovediac na každom kroku. Entity delíme na grafické, ako napr. úsečka, kóta alebo text, pozri časť 10.1.1, a na negrafické, ako napr. textový štýl, layout alebo hladina, pozri časť 10.1.2.

V nasledujúcich častiach 10.1.1 a 10.1.2 popíšeme ako k entitám pristupovať a popíšeme aj základné operácie na entitách.

Tip

Ako jedna z hlavných motivácii, prečo pristupovať k objektom ako k entitám je rýchlosť. Napr. vytvoriť úsečku príkazom **LINE** je mnohonásobne pomalšie, než vytvorením entity úsečky.

10.1.1 Grafické entity

Pod pojmom grafická entita si môžme predstaviť akýkoľvek objekt nakreslený v AutoCADe, teda napríklad úsečku, kružnicu alebo kótu. Podobne ako si v AutoCADe môžeme aj pomocou AutoLISPu čítať a meniť jeho vlastnosti. V AutoLISPe však potrebujeme najskôr získať meno entity, s ktorou chceme pracovať, pozri časť 10.1.1.1. Až následne, s využitím mena entity vieme pristupovať (čítať ale aj meniť) dáta entity, teda vlastnosti objektu, pozri časť 10.1.1.2.

10.1.1.1 Získanie mena entity

Základný prístup k menu entity je pomocou funkcie **entsel**, ktorá vyzve užívateľa na výber entity priamo v AutoCADe. Táto funkcia má jeden voliteľný argument, ktorým je text výzvy na výber entity. Ak výzvu nezadáme, použije sa štandardná výzva `Select object:`. Ak teda zavoláme funkciu **entsel** bez argumentov, po užívateľovom výbere sa nám vráti zoznam s dvomi prvkami:

```
Command: (entsel)
Select object: (<Entity name: 208afa1e4c0> (4214.71 1041.61 0.0))
Type a command
```

Prvým prvkom zoznamu je meno vybranej entity a druhým je bod, ktorým užívateľ vybral túto entitu.

Predtým, ako si ukážeme ďalšie spôsoby, ako získať meno entity si skúsme predstaviť výkres v AutoCADe ako jednu veľkú databázu, kde sú uložene (okrem iného) aj informácie o všetkých nakreslených objektoch. Tieto objekty sú v tejto databáze zoradené postupne, v poradí v akom ich vytvárame. Ukázali sme si, že funkciou **entsel** vieme bez ohľadu na toto poradie pristúpiť k entite, ktorú si označíme. Ak by sme ale chceli pristúpiť k entite, ktorá bola vytvorená hneď po vybranej entite, potrebovali by sme prístup do databázy výkresu. Na prehľadávanie objektov v databáze výkresu môžeme použiť funkcie **entnext** a **entlast**. Funkcia **entlast** vráti meno poslednej vytvorenej entity v databáze výkresu:

```
Command: (entlast)
<Entity name: 208afa1e4e0>)
Type a command
```

Funkcia **entnext** vráti meno prvej entity v databáze výkresu:

```
Command: (entnext)
<Entity name: 208afa1e4c0>
Type a command
```

Funkcia **entnext** má jeden voliteľný parameter, ktorým je meno entity. Ak teda dodáme funkcii meno entity, vráti nám meno entity, ktorá v databáze výkresu nasleduje po tejto entite.

Pri komplexných objektoch ako je blok alebo 3D polyline môžeme pristúpiť aj k subentitám, a to pomocou funkcie **nentsel** alebo **nentselp**. Funkcia **nentsel** slúži na získanie subentity pomocou výberu užívateľa. Na druhej strane funkcia **nentselp** umožňuje zadať súradnice bodu, v ktorom sa daná subentita nachádza, čím sa zbavíme potreby užívateľského vstupu. Takéto volania môžu vyzerať napr. nasledovne:

```
Command: (nentsel)
Select object: (<Entity name: 208afa1e540> (3816.89 726.006 0.0))
Command: (nentselp)
Select object: (<Entity name: 208afa1e540> (3815.94 713.482 0.0))
Command: (nentselp '(3816.9 726.0 0))
(<Entity name: 208afa1e540> (3816.9 726.0 0.0))
Type a command
```

K menu entity vieme pristúpiť ešte pomocou funkcie **ssname**, ktorá vracia meno entity z výberovej množiny, pozri časť 10.2. Prvým argumentom je výberová množina, z ktorej chceme získať entitu a druhým argumentom je poradové číslo entity vo výberovej množine. Majme napríklad výberovú množinu uloženú v premennej **vm**, potom získame meno prvej entity v tejto množine nasledovne:

```
Command: (ssname vm 0)
<Entity name: 208afa1e510>
Type a command
```

Meno entity nám samo o sebe veľa informácií neposkytuje. Toto meno je vlastne číslo v šestnástkovej sústave, ktoré označuje miesto v pamäti, kde sú dáta entity uložené (čo znamená, že meno entity je len dočasné a platí iba kým nezavrieme výkres). Meno entity je však nevyhnutné pre prístup k dátam (vlastnostiam) entity, pozri časť 10.1.1.2.

10.1.1.2 Dáta entity

Na základe mena entity môžeme pomocou funkcie **entget** pristúpiť k samotným dátam entity. Funkcia má len jeden argument, a to práve meno entity, pričom vracia zoznam, v ktorom sú dáta danej entity. Takémuto zoznamu hovoríme asociačný zoznam. Ilustrujme získanie dáta entity na nasledujúcom príklade, v ktorom do premennej **ciara** priradíme meno entity ľubovoľnej úsečky:

```
Command: (setq ciara (car (entsel)))
Select object: <Entity name: 27b00ad7530>
Type a command
```

Následne môžeme pristúpiť k samotným dátam entity:

```
Command: (entget ciara)
((-1 . <Entity name: 27b00ad7530>) (0 . "LINE") (330 . <Entity name: 27b00ad51f0>) (5 . "2B3") (100 .
"AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbLine") (10 3167.01 1850.12 0.0) (11 4199.38
1132.35 0.0) (210 0.0 0.0 1.0))
Type a command
```

Ako sme písali vyššie, ide o asociačný zoznam dát úsečky. Dáta entít sú uložené vo forme bodka-dvojíc, pozri časť 5.4.1, s výnimkou dát so súradnicami bodov, ktoré sú uložené vo forme zoznamu. Prvé miesto patrí tzv. DXF kódom skupín, na druhom mieste je samotná hodnota. V horeuvedenej entite máme napríklad DXF kódy:

- **-1**, ktorý patrí menu entity a má hodnotu `<Entity name: 186aba8bd60>`,
- **0**, ktorý patrí druhu entity a má hodnotu **"LINE"**,
- **5**, ktorý patrí indexu entity (tento index je fixný vo výkrese) a má hodnotu **296**,
- **8**, ktorý patrí hladine, do ktorej entita patrí a má hodnotu **"MyLayer"**,
- **10**, ktorý patrí súradniciam prvého bodu úsečky a má hodnotu **(3495.67 776.62 0.0)**,
- **11**, ktorý patrí súradniciam druhého bodu úsečky a má hodnotu **(3999.59 902.442 0.0)**.

DXF kódov skupín je oveľa viac, ale podrobne sa všetkým venovať nebudeme. Prehľad DXF kódov je možné nájsť aj priamo v pomocníkovi AutoCADu pod názvom "DXF Group Codes in Numerical Order Reference". [8]

Pomocou týchto DXF kódov skupín a funkcie **assoc**, pozri časť 5.4.1, dokážeme pristupovať k samotným dátam. Napríklad k hladine entity **ciara** pristúpime pomocou DXF kódu **8**:

```
Command: Command: (assoc 8 (entget ciara))
(8 . "0")
Type a command
```

10.1.2 Negrafické entity

Pojmom negrafická entita označujeme napríklad textový štýl, hladiny, layouty alebo skupiny. V AutoCADe sú negrafické entity uložené buď v tzv. tabuľkách symbolov alebo v slovníkoch (nie slovníky na kontrolu pravopisu). Tabuliek symbolov je 9, pozri tabuľku 10.1. Slovníkov je o niečo viac a všetky sú uložene v tzv. hlavnom slovníku. V tab. 10.2 uvádzame niekoľko vybraných slovníkov.

Tabuľka 10.1: Prehľad tabuliek symbolov

Tabuľka symbolov	Popis
APPID	registrované aplikácie
BLOCK	bloky (pomenované aj anonymné)
DIMSTYLE	kótovacie štýly
LAYER	hladiny
LTYPE	typy čiar
STYLE	textové štýly
UCS	pomenované užívateľské súradnicové systémy
VIEW	pomenované pohľady
VPORT	pomenované viewporty

Tabuľka 10.2: Prehľad vybraných slovníkov

Slovník	Popis
ACAD_GROUP	skupiny (pomenované aj nepomenované)
ACAD_LAYOUT	layouty (vrátane modelu)
ACAD_PLOTSETTINGS	nastavenie tlače
ACAD_SCALELIST	zoznam mierok
ACAD_TABLESTYLE	tabuľkové štýly
ACAD_VISUALSTYLE	vizuálne štýly

Tip

Dôvod rozdelenia negrafických objektov do dvoch skupín je veľmi jednoduchý. Všetky negrafické entity v starších verziách AutoCADu patrili do tabuliek. Avšak s príchodom nových verzií prichádzali aj nové funkcie a s nimi aj ďalšie negrafické entity, ktoré sú už uložené v novšej podobe – v slovníkoch. Staršie entity sú z dôvodu spätnej kompatibility ponechané v tabuľkách symbolov.

10.1.2.1 Slovníky

K hlavnému slovníku výkresu, ktorý môžete v literatúre nájsť aj pod označením „named object dictionary", sa dostaneme pomocou funkcie **namedobjdict**, ktorá nám vráti meno entity. S pomocou funkcie **entget**, pozri časť 10.1.1.2, teda vieme pristúpiť aj k dátam v tomto slovníku:

```
Command: (entget (namedobjdict))
((-1 . <Entity name: 186a6aa68c0>) (0 . "DICTIONARY") (330 . <Entity name: 0>) (5 . "C") (100 .
"AcDbDictionary") (280 . 0) (281 . 1) (3 . "ACAD_CIP_PREVIOUS_PRODUCT_INFO") (350 . <Entity name:
186a6a858e0>) (3 . "ACAD_COLOR") (350 . <Entity name: 186a6aa6bb0>) (3 . "ACAD_DETAILVIEWSTYLE") (350 .
<Entity name: 186a6a85930>) (3 . "ACAD_GROUP") (350 . <Entity name: 186a6aa68d0>) (3 . "ACAD_LAYOUT") (350 .
<Entity name: 186a6aa69a0>) (3 . "ACAD_MATERIAL") (350 . <Entity name: 186a6aa6ba0>) (3 .
"ACAD_MLEADERSTYLE") (350 . <Entity name: 186a6aa58d0>) (3 . "ACAD_MLINESTYLE") (350 . <Entity name:
186a6aa6970>) (3 . "ACAD_PLOTSETTINGS") (350 . <Entity name: 186a6aa6990>) (3 . "ACAD_PLOTSTYLENAME") (350 .
<Entity name: 186a6aa68e0>) (3 . "ACAD_RENDER_ACTIVE_RAPIDRT_SETTINGS") (350 . <Entity name: 186abf53d40>) (3
. "ACAD_RENDER_ACTIVE_SETTINGS") (350 . <Entity name: 18695ef01f0>) (3 . "ACAD_RENDER_RAPIDRT_SETTINGS") (350
. <Entity name: 186abf53cd0>) (3 . "ACAD_SCALELIST") (350 . <Entity name: 186a6aa5840>) (3 .
"ACAD_SECTIONVIEWSTYLE") (350 . <Entity name: 186a6a85910>) (3 . "ACAD_TABLESTYLE") (350 . <Entity name:
186a6aa6c60>) (3 . "ACAD_VISUALSTYLE") (350 . <Entity name: 186a6aa6e70>) (3 . "ACAD_WIPEOUT_VARS") (350 .
<Entity name: 186ac17e240>) (3 . "AcDbVariableDictionary") (350 . <Entity name: 186a6aa6ae0>) (3 .
"AEC_PROPERTY_SET_DEFS") (350 . <Entity name: 186aba7a2b0>))
Type a command
```

Vráti sa nám na prvý pohľad neprehľadný a dlhý výstup. Pri druhom pohľade si však môžete všimnúť veľmi jednoduchú štruktúru. Po hlavičke, kde sa okrem mena entity uvádza aj druh objektu slovníka `"DICTIONARY"`, nasleduje časť, v ktorej sa opakujú bodka-dvojice s DXF kódmi skupín **3** a **350**. Teda najprv je vždy uvedené označenie entity, napr. `ACAD_LAYOUT` alebo `ACAD_MLEADERSTYLE`, ktoré nasleduje meno entity, teda v tomto prípade `<Entity name: 186a6aa69a0>` resp. `<Entity name: 186a6aa58d0>`.

Na postupné prechádzanie sa medzi záznamami v slovníku slúži funkcia **dictnext**. Meno entity, ktorú chceme prechádzať, je prvým povinným argumentom funkcie. Druhým, dobrovoľným, argumentom sa vieme vrátiť k prvému záznamu a pokračovať od neho. To docielime, ak na mieste druhého argumentu bude čokoľvek iné ako **nil**.

Druhou funkciou na prístup k záznamom v slovníkoch je funkcia **dictsearch**. Má dva povinné argumenty – meno entity (jedna z (`"DICTIONARY"`), v ktorej hľadáme záznam a samotný názov záznamu je druhým argumentom.

Na demonštráciu funkcií **dictnext** a **dictsearch** majme výkres s dvomi layoutami. Uložme do premennej **layoutSlovnik** entitu slovníka, ktorý nájdeme funkciou **dictsearch** v hlavnom slovníku výkresu podľa názvu slovníka "ACAD_LAYOUT".

```
(setq layoutSlovnik (dictsearch (namedobjdict) "ACAD_LAYOUT"))
((-1 . <Entity name: 27b00ad51a0>) (0 . "DICTIONARY") (5 . "1A") (102 . "{ACAD_REACTORS") (330 . <Entity name:
27b00ad50c0>) (102 . "}") (330 . <Entity name: 27b00ad50c0>) (100 . "AcDbDictionary") (280 . 0) (281 . 1) (3 .
"Layout1") (350 . <Entity name: 27b00ad55b0>) (3 . "Layout2") (350 . <Entity name: 27b00ad7570>) (3 . "Model")
(350 . <Entity name: 27b00ad5220>))
Type a command
```

Následne môžeme prehľadať všetky layouty (pričom layoutom je z tohto pohľadu aj model) vo výkrese funkciou **dictnext**. Argumentom tejto funkcie je meno entity, takže nemôžme použiť priamo premennú **layoutSlovnik**, ale nájdeme v nej jej meno a uložíme ho do premennej **menoLayoutSlovnika**:

```
Command: (setq menoLayoutSlovnika (cdr (assoc -1 layoutSlovnik)))
<Entity name: 27b00ad51a0>
Type a command
```

Nakoniec môžeme opakovane volať funkciu **dictnext** s argumentom **menoLayoutSlovnika**, čím prejdeme postupne všetky layouty vo výkrese:

```
Command: (dictnext menoLayoutSlovnika)
((-1 . <Entity name: 25a62494db0>) (0 . "LAYOUT") (5 . "D3") (102 . "{ACAD_REACTORS") (330 . <Entity name:
25a624949a0>) (102 . "}") (330 . <Entity name: 25a624949a0>) (100 . "AcDbPlotSettings") (1 . "") (2 .
"C:\\Documents and Settings\\basas\\Application Data\\Autodesk\\AutoCAD 2005\\R16.1\\enu\\plotters\\Default
Windows System Printer.pc3") (4 . "") (40 . 0.0) (41 . 0.0) (42 . 0.0) (43 . 0.0) (44 . 0.0) (45 . 0.0) (46 .
0.0) (47 . 0.0) (48 . 0.0) (49 . 0.0) (140 . 0.0) (141 . 0.0) (142 . 1.0) (143 . 1.0) (70 . 688) (72 . 0) (73
. 0) (74 . 5) (7 . "") (75 . 16) (147 . 1.0) (76 . 0) (77 . 2) (78 . 300) (148 . 0.0) (149 . 0.0) (100 .
"AcDbLayout") (1 . "Layout1") (70 . 1) (71 . 1) (10 0.0 0.0 0.0) (11 12.0 9.0 0.0) (12 0.0 0.0 0.0) (14 0.0
0.0 0.0) (15 0.0 0.0 0.0) (146 . 0.0) (13 0.0 0.0 0.0) (16 1.0 0.0 0.0) (17 0.0 1.0 0.0) (76 . 0) (330 .
<Entity name: 25a62494da0>))
Type a command
```

```
Command: (dictnext menoLayoutSlovnika)
((-1 . <Entity name: 25a62494df0>) (0 . "LAYOUT") (5 . "D7") (102 . "{ACAD_REACTORS") (330 . <Entity name:
25a624949a0>) (102 . "}") (330 . <Entity name: 25a624949a0>) (100 . "AcDbPlotSettings") (1 . "") (2 .
"C:\\Documents and Settings\\basas\\Application Data\\Autodesk\\AutoCAD 2005\\R16.1\\enu\\plotters\\Default
Windows System Printer.pc3") (4 . "") (40 . 0.0) (41 . 0.0) (42 . 0.0) (43 . 0.0) (44 . 0.0) (45 . 0.0) (46 .
0.0) (47 . 0.0) (48 . 0.0) (49 . 0.0) (140 . 0.0) (141 . 0.0) (142 . 1.0) (143 . 1.0) (70 . 688) (72 . 0) (73
. 0) (74 . 5) (7 . "") (75 . 16) (147 . 1.0) (76 . 0) (77 . 2) (78 . 300) (148 . 0.0) (149 . 0.0) (100 .
"AcDbLayout") (1 . "Layout2") (70 . 1) (71 . 2) (10 0.0 0.0 0.0) (11 12.0 9.0 0.0) (12 0.0 0.0 0.0) (14 0.0
0.0 0.0) (15 0.0 0.0 0.0) (146 . 0.0) (13 0.0 0.0 0.0) (16 1.0 0.0 0.0) (17 0.0 1.0 0.0) (76 . 0) (330 .
<Entity name: 25a62494de0>))
Type a command
```

```
Command: (dictnext menoLayoutSlovnika)
((-1 . <Entity name: 25a62494a20>) (0 . "LAYOUT") (5 . "22") (102 . "{ACAD_XDICTIONARY") (360 . <Entity name:
25a6249c950>) (102 ."}") (102 . "{ACAD_REACTORS") (330 . <Entity name:25a624949a0>) (102 . "}") (330 .
<Entity name: 25a624949a0>) (100 . "AcDbPlotSettings") (1 . "") (2 . "none device") (4 . "ISO_A4_
(210.00_x_297.00_MM)") (40 . 0.0) (41 . 7.5) (42 . 7.5) (43 . 20.0) (44 . 210.0) (45 . 297.0) (46 . 11.55) (47
. -13.65) (48 . 0.0) (49 . 0.0) (140 . 0.0) (141 . 0.0) (142 . 1.0) (143 . 8.70408) (70 . 11952) (72 . 1) (73
. 0) (74 . 0) (7 . "") (75 . 0) (147 . 0.114889) (76 . 0) (77 . 2) (78 . 300) (148 . 0.0) (149 . 0.0) (100 .
"AcDbLayout") (1 . "Model") (70 . 1) (71 . 1) (10 0.0 0.0 0.0) (11 12.0 9.0 0.0) (12 0.0 0.0 0.0) (14 0.0 0.0
0.0) (15 0.0 0.0 0.0) (146 . 0.0) (13 0.0 0.0 0.0) (16 1.0 0.0 0.0) (17 0.0 1.0 0.0) (76 . 0) (330 . <Entity
name: 25a62494da0>))
Type a command
```

```
Command: (dictnext menoLayoutSlovnika)
nil
Type a command
```

Vidíme, že prvý layout je layout s názvom `Layout1`, nasleduje `Layout2` a posledný layout je samotný modelovací priestor – `Model`.

Na prácu so slovníkmi máme k dispozícii ešte funkcie na pridanie (**dictadd**), premenovanie (**dictrename**) a odstránenie (**dictremove**) slovníka. O podrobnostiach týchto funkcií a o ich použití sa môžete viac dozvedieť napr. v pomocníkovi AutoCADu.

10.1.2.2 Tabuľky symbolov

Na rozdiel od slovníkov, pri tabuľkách symbolov neexistuje hlavná tabuľka, v ktorej by sme vedeli nájsť ostatné vnorené tabuľky. Názvy tabuliek, pozri tabuľku 10.1, si musíme jednoducho pamätať, príp. vyhľadať v pomocníkovi. Pri práci s entitami v tabuľkách symbolov máme k dispozícii obdobné funkcie, ako pri práci s entitami v slovníkoch, nájdeme však medzi nimi značné rozdiely.

Začnime s funkciu **tblsearch** ktorá nám vyhľadá vo vybranej tabuľke symbolov hľadaný záznam vo forme zoznamu. Napríklad, ak hľadáme hladinu `0` v tabuľke `"LAYER"`, dostaneme nasledovný výstup:

```
Command: (tblsearch "LAYER" "0")
((0 . "LAYER") (2 . "0") (70 . 0) (62 . 7) (6 . "Continuous"))
Type a command
```

Podobne pri funkcii **tblnext** je výstupom zoznam záznamov o nasledujúcej položke v tabuľke symbolov (názov tabuľky symbolov je argumentom). Ak by sme mali výkres s dvomi hladinami, výstup po volaniach funkcie by bol nasledovný:

```
Command: (tblnext "LAYER")
((0 . "LAYER") (2 . "0") (70 . 0) (62 . 7) (6 . "Continuous"))
Command: (tblnext "LAYER")
((0 . "LAYER") (2 . "novaHladina") (70 . 0) (62 . 1) (6 . "Continuous"))
Command: (tblnext "LAYER")
nil
Type a command
```

Na získanie mena entity musíme použiť funkciu **tblobjname**, ktorá má rovnaké argumenty ako funkcia **tblsearch**. Zostaňme pri hladinách a vypíšme dostupné dáta o hladine `0`:

```
Command: (entget (tblobjname "LAYER" "0"))
((-1 . <Entity name: 25a62494900>) (0 . "LAYER") (5 . "10") (102 . "{ACAD_XDICTIONARY") (360 . <Entity name:
25a62498a40>) (102 . "}") (330 . <Entity name: 25a62494820>) (100 . "AcDbSymbolTableRecord") (100 .
"AcDbLayerTableRecord") (2 . "0") (70 . 0) (62 . 7) (6 . "Continuous") (290 . 1) (370 . -3) (390 . <Entity
name: 25a624948f0>) (347 . <Entity name: 25a62494ee0>) (348 . <Entity name: 0>))
Type a command
```

Vidíme, že výstup z funkcie **tblobjname** je rozsiahlejší. Na druhej strane zjednodušený výpis z funkcií **tblsearch** a **tblnext** obsahuje najpodstatnejšie informácie.

Tip

Pre prácu s tabuľkou `"VIEW"` máme k dispozícii aj funkciu **setview**, ktorá zobrazí daný pomenovaný pohľad. Povinným argumentom je teda popis pohľadu (výstup s funkcií **tblsearch** a **tblnext**). Ak teda chceme zobraziť pohľad s názvom `detail`, zavoláme

```
(setview (tblsearch "VIEW" "detail"))
```

10.1.3 Úprava, vytváranie a odstraňovanie entít

Doposiaľ sme entity iba prehľadávali a zobrazovali sme ich záznamy. My však môžeme tieto záznamy aj upravovať, prípadne dokážeme vytvoriť aj celkom nový záznam, teda novú entitu.

10.1.3.1 Úprava existujúcich entít

Na to, aby sme mohli nejakú entitu upravovať, je potrebné aby nejaká entita vo výkrese bola vytvorená. Na nasledujúcom príklade ukážeme, ako môžeme entity upravovať pomocou funkcie **entmod**.

Majme výkres s jedným textovým objektom, ktorý vytvoríme nasledovne:

```
Command: TEXT
Current text style:  "Standard"  Text height:  2.5000  Annotative:  No  Justify:  Left
Specify start point of text or [Justify/Style]: 0,0
Specify height <2.5000>: 10
Specify rotation angle of text <0>: 45
TEXT ahoj
```

Na úpravu vlastností entít môžeme využiť funkciu **entmod**. Funkcia **entmod** má jediný argument a to upravený zoznam vlastností entity. To znamená, že ak chceme zmeniť nejaký záznam vybranej entity, potrebujeme najprv získať ten aktuálny, upraviť ho a až nakoniec ho pomocou spomínanej funkcie **entmod** priradiť, a tým aj upraviť existujúcu entitu. Poďme v troch krokoch upraviť text textového objektu, ktorý sme hore vytvorili. Najprv získame zoznam vlastností entity a uložíme ho do premennej **entita**:

```
Command: (setq entita (entget (entnext)))
((-1 . <Entity name: 206ae665ce0>) (0 . "TEXT") (330 . <Entity name: 206ae65f9f0>) (5 . "28E") (100 .
"AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbText") (10 0.0 0.0 0.0) (40 . 10.0) (1 . "ahoj")
(50 . 0.785398) (41 . 1.0) (51 . 0.0) (7 . "Standard") (71 . 0) (72 . 0) (11 0.0 0.0 0.0) (210 0.0 0.0 1.0)
(100 . "AcDbText") (73 . 0))
Type a command
```

Následne zoznam uložený v premennej **entita** upravíme, konkrétne nahradíme záznam s DXF kódom 1, čo zodpovedá textu textového objektu. Takto upravený zoznam uložíme do premennej **edEntita**:

```
Command: (setq edEntita (subst (cons 1 "Dobrý deň") (assoc 1 entita) entita ))
((-1 . <Entity name: 206ae665ce0>) (0 . "TEXT") (330 . <Entity name: 206ae65f9f0>) (5 . "28E") (100 .
"AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbText") (10 0.0 0.0 0.0) (40 . 10.0) (1 . "Dobrý
deň") (50 . 0.785398) (41 . 1.0) (51 . 0.0) (7 . "Standard") (71 . 0) (72 . 0) (11 0.0 0.0 0.0) (210 0.0 0.0
1.0) (100 . "AcDbText") (73 . 0))
Type a command
```

Posledným krokom je nahradenie pôvodného zoznamu vlastností entity modifikovaným zoznamom, ktorý máme uložený v premennej **edEntita**:

```
Command: (entmod edEntita)
((-1 . <Entity name: 206ae665ce0>) (0 . "TEXT") (330 . <Entity name: 206ae65f9f0>) (5 . "28E") (100 .
"AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbText") (10 0.0 0.0 0.0) (40 . 10.0) (1 . "Dobrý
deň") (50 . 0.785398) (41 . 1.0) (51 . 0.0) (7 . "Standard") (71 . 0) (72 . 0) (11 0.0 0.0 0.0) (210 0.0 0.0
1.0) (100 . "AcDbText") (73 . 0))
Type a command
```

V predchádzajúcej ukážke sme zmenili len jeden záznam s DXF kódom skupiny **1**, ale v prípade potreby je možné zmeniť aj viacero záznamov naraz. Vždy treba ale dávať pozor na to, aby nová hodnota bola vždy správneho typu. Ak by sme v tomto príklade priradili ku kódu **1** napríklad celé číslo **10**, pri volaní funkcie **entmod** by sme dostali chybové hlásenie:

```
Command: (setq edEntita (subst (cons 1 10) (assoc 1 entita) entita ))
((-1 . <Entity name: 206ae665ce0>) (0 . "TEXT") (330 . <Entity name: 206ae65f9f0>) (5 . "28E") (100 .
"AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbText") (10 0.0 0.0 0.0) (40 . 20) (1 . 10) (50 .
0) (41 . 1.0) (51 . 0.0) (7 . "Standard") (71 . 0) (72 . 0) (11 0.0 0.0 0.0) (210 0.0 0.0 1.0) (100 .
"AcDbText") (73 . 0))
Command: (entmod edEntita)
; error: bad DXF group: (1 . 4)
Type a command
```

10.1.3.2 Vytváranie nových entít

Na vytvorenie novej entity používame funkciu **entmake**, ktorej argumentom je zoznam vlastností vytváranej entity. Pri vytváraní novej entity je nevyhnutné dohliadnuť na to, aby v zozname vlastností

nechýbal žiadny povinný DXF kód skupiny. Ak by sme na nejaký predsa len zabudli a poslali by sme nekompletný zozname vlastností do funkcie **entmake**, namiesto novej entity by nám vrátila **nil**.

Pri každej entite by určite nemal chýbať DXF kód skupiny **0** (druh entity), ostatné kódy potom závisia práve od druhu entity. Napr. pre úsečku je dostatočný zoznam

```
'((0 . "LINE") (10 . bod1) (11 . bod2))
```

pre kružnicu

```
'((0 . "CIRCLE") (10 . stred) (40 . polomer))
```

a pre hladinu

```
'((0 . "LAYER") (100 . "AcDbSymbolTableRecord")
(100 . "AcDbLayerTableRecord") (2 . "nazovhladiny") (70 . flag))
```

Napríklad novú hladinu "nazov" s minimom definovaných vlastností teda vytvoríme nasledovne:

```
Command: (entmake '((0 . "LAYER") (100 . "AcDbSymbolTableRecord") (100 . "AcDbLayerTableRecord") (2 .
"nazov") (70 . 0)))
((0 . "LAYER") (100 . "AcDbSymbolTableRecord") (100 . "AcDbLayerTableRecord") (2 . "nazov") (70 . 0))
Type a command
```

Ak by sme chceli napr. určiť aj farbu novovytváranej hladiny "cervena", musíme uviesť hodnotu aj pre DXF kód skupiny **62**:

```
Command: (entmake '((0 . "LAYER") (100 . "AcDbSymbolTableRecord") (100 . "AcDbLayerTableRecord") (2 .
"nazov") (70 . 0) (62 . 1)))
((0 . "LAYER") (100 . "AcDbSymbolTableRecord") (100 . "AcDbLayerTableRecord") (2 . "nazov") (70 . 0) (62 . 1))
Type a command
```

10.1.3.3 Odstraňovanie entít

Na odstránenie entity sa používa funkcia **entdel**, ktorej jediný argument je meno entity, ktorú chceme odstrániť. Zaujímavosťou tejto funkcie je, že ak ju zavoláme na odstránenú entitu, entita sa obnoví. Toto obnovenie platí aj pri odstránení príkazom **DELETE**.

10.2 Výberové množiny

Výberová množina nie je nič iné ako množina entít. Táto množina býva určená výberom, či už zo strany užívateľa, alebo výberom priamo v kóde. V oboch prípadoch je navyše možné aplikovať rôzne filtre, napríklad na výber iba určeného druhu objektu.

Základnou funkciou pri práci s výberovými množinami je funkcia **ssget**. Volaním tejto funkcie dôjde v AutoCADe k výzve na výber objektov. Jednoduchým volaním funkcie bez argumentov sa nám vráti výberová množina, napr. `<Selection set: 0>`:

```
Command: (ssget)
Select objects: Specify opposite corner: 1 found
Select objects:
<Selection set: a>
Type a command
```

Označenie výberovej množiny `<Selection set: 0>` nehovorí nič o počte entít v množine ani o poradovom čísle množiny a v AutoLISPe ho nevieme využiť. Aby sme s výberovou množinou mohli ďalej pracovať, je nevyhnutné ju priradiť premennej, napr:

```
Command: (setq vyber (ssget))
Select objects: Specify opposite corner: 4 found
Select objects:
<Selection set: c>
Type a command
```

Teraz máme výberovú množinu uloženú v premennej **vyber** a môžeme napríklad zistiť koľko objektov (entít) bolo vybraných. Zistíme to pomocou funkcie **sslength** ktorej jediným argumentom je výberová množina:

10.2.1 Metódy výberu

Volaním funkcie **ssget** bez argumentov nechávame užívateľovi absolútnu voľnosť vo výbere. Metódu výberu objektu vieme obmedziť prvým voliteľným argumentom. Možností je veľa, väčšinu poznáme z výberu v AutoCADe, napr. pomocou príkazu **SELECT**. V tejto časti uvádzame rôzne spôsoby výberu aj s konkrétnymi príkladmi volaní funkcie **ssget**.

Výber bez zásahu užívateľa Objekty je možné vyberať aj bez toho, aby funkcia **ssget** vyžadovala vstup od užívateľa. V závislosti od vybranej metódy totiž netreba okrem samotnej výberovej metódy žiadny ďalší vstup, pri niektorých je navyše nutné dodať aj body, resp. zoznam bodov. Nasledujúci prehľad popisuje všetky metódy aj s príkladmi volaní funkcie **ssget**:

- `"_A"` (All objects)
  - výber všetkých objektov, vrátane objektov z vypnutých hladín (bez objektov v zmrazených hladinách)
  - `(ssget "_A")`
- `"_C"` (Crossing selection)
  - výber krížením, je nutné poslať body ako argumenty funkcie **ssget**
  - `(ssget "_C" '(0 0) '(100 100))`
- `"_CP"` (Crossing polygon selection)
  - výber krížením polygónom, je nutné poslať body ako argumenty funkcie
  - `(ssget "_CP" '((0 0) (100 100) (430 10)))`
- `"_F"` (Fence selection)
  - výber krížením hranicou, je nutné poslať body ako argumenty funkcie
  - `(ssget "_F" '((0 0) (100 100) (430 10)))`
- `"_I"` (Implied selection)
  - vytvorí výber iba z objektov, ktoré boli vo výbere už pred zavolaním funkcie **ssget**, pričom systémová premenná **PICKFIRST** musí byť zapnutá (nastavená na hodnotu **1**), v opačnom prípade vráti **nil**
  - `(ssget "_I")`
- `"_L"` (Last)
  - posledný pridaný viditeľný objekt

  - `(ssget "_L")`

- `"_P"` (Previous)
  - predchádzajúca výberová množina
  - `(ssget "_P")`

- `"_W"` (Window selection)
  - výber oknom, je nutné poslať body ako argumenty funkcie **ssget**
  - `(ssget "_W" '(0 0) '(100 100))`

- `"_WP"` (Window polygon selection)
  - výber oknom polygónu, je nutné poslať body ako argumenty funkcie
  - `(ssget "_WP" '((0 0) (100 100) (430 10)))`

- `"_X"` (Extended search)
  - výber všetkých objektov (vrátane objektov zo zmrazených a vypnutých hladín)
  - `(ssget "_X")`

Výber zo strany užívateľa Pri horeuvedených metódach sa od užívateľa neočakáva žiadny vstup. Vždy totiž ide o vopred dané definície toho, z čoho sa má vytvoriť výberová množina. Či už ide o všetky objekty, objekty v nejakej danej oblasti, prípadne ide o množiny už predtým vytvorené.

Pokiaľ by sme chceli do výberu objektov zapojiť aj užívateľa, mohli by sme to v prípade metód `C` a `W` vyriešiť použitím funkcií **getpoint** a **getcorner**, pozri časť 6.3.3. Takéto riešenie ale nie je úplne ideálne, pretože užívateľ nebude pri výbere vidieť štandardný zelený obdĺžnik pri výbere krížením (Crossing), resp. modrý obdĺžnik pri výbere oknom (Window). Môžeme však využiť aj celkom iný prístup. Pri funkcii **ssget** totiž môžeme, rovnako ako pri príkaze **SELECT**, využívať výberové módy. Výberový mód na rozdiel od metódy výberu môže pre uľahčenie výberu určiť samotný užívateľ po výzve na výber objektov. K dispozícii sú štandardne nasledovné módy:

- `_A` – všetky objekty (All),
- `_B` – výber oknom (Window) pri potiahnutí doľava, resp. výber krížením (Crossing) pri potiahnutí doprava (Box),
- `_C` a `_CP` – výber krížením pri potiahnutí do ľubovoľnej strany (Crossing, resp. Crossing polygon),
- `_F` – výber krížením naprieč hranicou (Fence),
- `_G` – výber skupiny (Group),
- `_L` – výber posledného pridaného viditeľného objektu (Last),
- `_M` – viacnásobný výber objektov po jednom (Multiple),
- `_P` – výber predchádzajúcej výberovej skupiny (Previous),
- `_W` a `_WP` – výber oknom pri potiahnutí do ľubovoľnej strany,
- `_.` – výber kliknutím na objekt, pri zadávaní do príkazového riadku voľbou `SIngle`.

Výber z uvedeného zoznamu môžeme pri volaní funkcie **ssget** vopred obmedziť pomocou znakov "+" a "-", ktoré pridajú, resp. odoberú užívateľovi možnosť vybrať si daný výberový mód. Výberové módy môžeme obmedziť, napríklad na voľby "_F" a "_P" nasledovne:

```
Command: (ssget "_+F+P")
Select objects: w
*Invalid selection*
Valid keywords: Fence/Add/Remove/Previous/Undo/SIngle
Select objects: f
Specify first fence point or pick/drag cursor:
```

Tip

Nie všetky kombinácie sú prípustné. Napríklad `(ssget "-W")` vráti `; error: bad point argument`, pretože očakáva aj argument s bodmi výberového okna.

Ak chceme aby užívateľ objekty určil ručne, ale nechceme mu nechať úplnú voľnosť, môžeme využiť jeden alebo rovno hneď niekoľko z nasledovných modifikátorov:

- "_:D" – povolený duplicitný výber objektov
- "_:E" – výber všetkého vo vnútri výberového terčíka (pickbox) kurzora
- "_:L" – výber iba z objektov neuzamknutých hladín
- "_:N" – umožní výber vnorených objektov, napríklad v bloku. Do výberovej množiny sa však zaradí entita hlavného objektu (bloku), pričom pri výbere rôznych vnorených objektov v bloku sa do výberovej množiny zaradí duplicitne hlavný objekt. Pri následnej práci s výberovou množinou získame meno entity hlavného objektu použitím funkcie **ssname** a funkciou **ssnamex** získame meno subentity, ktorou bola hlavná entita (blok) vybraná.
- "_:S" – umožní iba jeden výber jedného alebo viacerých objektov
- "_:U" – umožní výber subentít (napr. segment objektu Polyline) pri stlačenom ctrl. Nedá sa kombinovať s "_:D" ani "_:N"
- "_:V" – umožní iba výber subentít (napríklad segment objektu Polyline). Nedá sa kombinovať s "_:D" ani "_:N".

Môžeme teda napr. umožniť výber všetkého vo výberovom terčíku (pickbox) kurzora: `(ssget "_:E")` alebo výber iba z objektov neuzamknutých hladín: `(ssget "_:L")`.

Modifikátory môžeme aj vzájomne kombinovať, napr. pre výber iba z objektov neuzamknutých hladín vo výberovom terčíku (pickbox) kurzora: `(ssget "_:E:L")` alebo jeden výber iba z objektov neuzamknutých hladín vo výberovom terčíku (pickbox) kurzora: `(ssget "_:E:L:S")`. Popri modifikátoroch môžeme zároveň využiť aj výberové módy, kde sa najčastejšie využíva kombinácia pre výber jedného objektu: `(ssget "_:S+.")`.

10.2.2 Filtre výberových množín

Pri tvorbe výberových množín sa často používajú tzv. filtre výberových množín, ktoré sú voliteľným posledným argumentom funkcie **ssget**. Filtre sú založené na DXF kódoch skupín. Ak napríklad chceme filtrovať výber pomocou typu entity, použijeme podmienku pre DXF kód skupiny 0. Takto vieme výber

Tabuľka 10.3: Prehľad logických operátorov pre filtrovanie pri tvorení výberových množín

Otvárací operátor	Počet položiek	Ukončovací operátor
"<AND"	1 a viac	"AND>"
"<OR"	1 a viac	"OR>"
"<XOR"	2	"XOR>"
"<NOT"	1	"NOT>"

obmedziť napr. na objekty jednoriadkového textu: `(ssget '((0 . "TEXT")))` alebo výber objektov úsečiek: `(ssget '((0 . "LINE")))`

Filtre môžeme kombinovať aj s metódami, módmi a modifikátormi výberu, pozri časť 10.2.1, napr. na výber objektov jednoriadkového textu z predchádzajúcej výberovej množiny môžeme použiť výraz `(ssget "_P" '((0 . "TEXT")))` alebo na výber všetkých objektov v hladine `popis` použijeme výraz `(ssget "_X" '((8 . "popis")))` .

Často ale nestačí odfiltrovať len jednu vlastnosť. Ak chceme výber zúžiť, nič nám nebráni v použití viac DXF kódov skupín vo filtri. Môžeme teda napríklad vybrať všetky objekty jednoriadkového textu v hladine `popis`: `(ssget "_X" '((0 . "TEXT") (8 . "popis")))`.

O niečo náročnejšie je vytvoriť filter napr. pre dva rôzne objekty, povedzme úsečky a objekty jednoriadkového textu. Ak by sme to skúsili nasledovným spôsobom, neboli by sme úspešní:

Toto volanie by vrátilo `nil` a výberová množina by sa nevytvorila, pretože v celom výkrese nebola nájdená entita, ktorá by bola zároveň `"TEXT"` aj `"LINE"`. Na takéto rozšírenie filtra potrebujeme logické operátory (pozn. nie logické funkcie z časti 7.2). Pri práci s výberovými množinami disponujeme štyrmi dvojicami operátorov: `"AND"`, `"OR"`, `"XOR"` a `"NOT"`, pozri tabuľku 10.3. Ide o dvojice operátorov, z ktorých je vždy jeden otvárací a jeden ukončovací a samotná podmienka sa nachádza medzi nimi. Tieto operátory budú súčasťou zoznamu bodka-dvojíc, pričom ich priraďujeme ku DXF kódu `-4`. Takže výber všetkých objektov dvoch druhov entít, napr. `"TEXT"` a `"LINE"` by sme získali pomocou: `(ssget "_X" '((-4 . "<OR") (0 . "TEXT") (0 . "LINE") (-4 . "OR>")))`, resp. o niečo prehľadnejšie zapísané v kóde 10.1.

AutoLISP kód 10.1: Filter výberu všetkých objektov úsečky a jednoriadkového textu pomocou funkcie `ssget`

```
(ssget "X" '(
         (-4 . "<OR")
                (0 . "TEXT")
                (0 . "LINE")
         (-4 . "OR>"))
)
```

Bibliografia

[1] Jeanne Aarhus. *Making the CUI Work for You*. URL: https://www.autodesk.com/autodesk-university/class/Making-CUI-Work-You-2012 (cit. 19. 01. 2021).

[2] Dan Abbott. *AutoCAD: secrets every user should know*. John Wiley & Sons, 2007. ISBN: 978-0470109939.

[3] *ASCII Table - ASCII Character Codes, HTML, Octal, Hex, Decimal*. URL: https://www.asciitable.com/ (cit. 19. 11. 2021).

[4] *AutoCAD 2000 Visual LISP Tutorial*. Autodesk, 1999.

[5] *AutoCAD 2023 Help | Autodesk*. URL: https://help.autodesk.com/view/ACD/2023/ENU/ (cit. 01. 10. 2022).

[6] *CAD Forum - How to set the drawing window background color from menu, script or LISP?* URL: https://www.cadforum.cz/en/qaID.asp?tip=3088 (cit. 07. 03. 2022).

[7] *DIESEL Functions Reference | AutoCAD LT | Autodesk Knowledge Network*. URL: https://knowledge.autodesk.com/support/autocad-lt/learn-explore/caas/CloudHelp/cloudhelp/2015/ENU/AutoCAD-LT/files/GUID-F94A885A-4DA2-432B-AC1A-EB49CC6C1C72-htm.html (cit. 04. 06. 2022).

[8] *DXF Reference: Group Codes in Numerical Order*. URL: http://docs.autodesk.com/ACAD_E/2012/ENU/filesDXF/WS1a9193826455f5ff18cb41610ec0a2e719-7a62.htm (cit. 04. 02. 2022).

[9] *Lesson 11, List manipulation*. URL: http://ronleigh.com/autolisp/ales11.htm (cit. 07. 03. 2022).

[10] *Notepad++*. URL: https://notepad-plus-plus.org/ (cit. 19. 07. 2022).

[11] D Stein. *The Visual LISP Developers Bible, 2003 Edition (v2)*. 2003.

[12] Zuzana Tereňová, Juliana Beganová a Martin Ambroz. *Základy počítačovej podpory projektovania pomocou AutoCADu*. Spektrum STU, Bratislava, 2022.

[13] *Windows Batch Scripting - Wikibooks, open books for an open world*. URL: https://en.wikibooks.org/wiki/Windows_Batch_Scripting (cit. 05. 04. 2022).

Ing. Martin Ambroz, PhD.

PRISPÔSOBOVANIE UŽÍVATEĽSKÉHO ROZHRANIA A AUTOMATIZÁCIA PROCESOV V SOFTVÉRI AUTOCAD

Vydala Slovenská technická univerzita v Bratislave vo Vydavateľstve SPEKTRUM STU, Bratislava, Vazovova 5, v roku 2022.

Edícia skrípt

Rozsah 132 strán, 41 obrázkov, 6 tabuliek, 8,633 AH, 8,851 VH, 1. vydanie,
edičné číslo 6131, vydané v elektronickej forme.

85 – 236 – 2022

ISBN 978-80-227-5258-9