Podprogramy

1. Motivácia
   - čo je to podprogram
   - účel podprogramov
2. Druhy podprogramov
   - makrá
   - procedúry a funkcie
3. Vlastnosti podprogramov
   - vykonávanie podprogramov
   - vedľajšie efekty podprogramov
4. Komunikácia medzi podprogramami
   - odovzdávanie parametrov
   - použitie globálnych premenných
   - návratová hodnota funkcie

Čo je to podprogram?

- Pri analýze programu často zistíme, že istú časť výpočtu potrebujeme vykonať viackrát
- Viacnásobný zápis tej istej časti programu môže spôsobiť problém
  - neprehľadnosť a veľký rozsah programu
  - väčšia pravdepodobnosť vzniku chýb pri programovaní
- Samostatná časť programu, na ktorú sa môžeme v programe viacnásobne odvolávať – **podprogram**
- Spôsob realizácie odvolávania sa na podprogram
- Podprogram môže byť definovaný aj mimo programu a využívaný viacerými programami

Účel podprogramu

- Umožňuje naprogramovať viackrát sa **opakujúcu činnosť len raz**
  - sprehľadnenie a zefektívnenie programu
- Poskytuje **štandardné činnosti**
  - operácie počítačového systému využívané programami sú realizované ako podprogramy
- Podporuje **modulárne programovanie**
  - jeden zo spôsobov členenia programu na menšie časti, ktoré môžu byť vytvárané samostatne

Druhy podprogramov

- Z hľadiska konštrukcie podprogramy delíme
  - **makrá**
  - **procedúry** a **funkcie**
- Líšia sa spôsobom realizácie odvolávania sa na podprogram

Makrá

- Časť zdrojového textu programu, ktorá je v celom zdrojovom texte programu **prepisovaná podľa definovaných pravidiel**
  - prekladač pri preklade prepíše časť textu programu na všetkých miestach, kde má byť táto použitá
  - po prepísaní sa už ďalej kompiluje program, ktorý neobsahuje žiadne makrá
- Makro sa používa v počítačových systémoch, kde **nie je možné realizovať operáciu odvolávania** sa na podprogram
- Skompilovaný program obsahuje viacnásobne danú časť programu a je preto väčší (možná nevýhoda)
- Pri behu programu nie je potrebná réžia pre realizáciu odvolávania sa na podprogram a je preto rýchlejší (možná výhoda)

Makrá v jazyku C

- V jazyku C sa makro definuje príkazom **predprocesora**

```
#define názov(parametre) text
```

- Všade v zdrojovom texte programu, kde sa objaví odvolávanie sa na makro, prepíše sa definovaným textom
- Prepis prebieha pri preklade vo fáze spracovania predprocesorom
- Príklad makra bez parametrov (možnosť implementácie konštanty)

```
#define PI 3.14

o = 2 * PI * r;   // prepíše sa na výraz: o = 2 * 3.14 * r;
```

- Príklad makra s parametrom

```
#define max(x, y) ((x) > (y) ? (x) : (y))

x = max(5, 2);  // prepíše sa na výraz: x = ((5) > (2) ? (5) : (2));
```

Procedúry a funkcie

- Podprogram je v programe **definovaný iba raz** a aj počas behu programu sa **odvoláva na jednu definíciu**
- Procedúry a funkcie sa používajú v počítačových systémoch, kde **je možné realizovať operáciu odvolávania** sa na podprogram
- **Procedúra** – podprogram, ktorý ako výsledok vykonávania nevracia žiadnu hodnotu
- **Funkcia** – podprogram, ktorý ako výsledok vykonávania vracia hodnotu a tú je možné okamžite použiť vo výraze
- Niektoré programovacie jazyky nerozlišujú medzi procedúrou a funkciou
  - napr. v jazyku C sú tieto podprogramy vždy funkcie, procedúra je funkcia vracajúca hodnotu typu `void`

Vykonávanie podprogramov

- Podprogram pracuje ako samostatný program, ale v menšom meradle
- Program vykoná operáciu volania podprogramu (call) a skočí na adresu prvej inštrukcie podprogramu
- Adresu nasledujúcej inštrukcie, kde má program pokračovať po ukončení podprogramu je potrebné uchovať
  - zo začiatku sa používali registre procesora, čo je nepostačujúce pre hlbšie úrovne volania podprogramov a pre prenos údajov
  - zásobník je lepší prístup, navyše prirodzene umožňuje aj realizáciu rekurzívnych volaní funkcií
- Program čaká na ukončenie podprogramu, ktorý vykoná operáciu návratu z podprogramu (return)
- Program po návrate z podprogramu pokračuje od uchovanej adresy
- Podprogramu môžu byť odovzdávané parametre (údajové objekty), ktoré podprogram využíva
- Podprogram môže po ukončení preniesť návratovú hodnotu alebo aj ovplyvniť hodnoty iných údajových objektov

Vedľajšie efekty podprogramov

- V **imperatívnych jazykoch** (napr. C) môže podprogram spôsobiť **vedľajšie efekty** – zmeny, ktoré zostávajú aj po návrate z podprogramu
  - podprogram môže **ovplyvniť údajové objekty mimo podprogramu**
  - najčastejšie to spôsobujú procedúry, ktoré nemajú inú možnosť určenia návratovej hodnoty
- Prekladač nemôže odhaliť či podprogram spôsobí vedľajší efekt
- Vedľajší efekt môže spôsobiť, že podprogram (funkcia) **bude mať rôznu návratovú hodnotu aj pri použití rovnakých parametrov** (napr. funkcia vracajúca pseudonáhodné čísla)
  - toto správanie je **v rozpore so striktným matematickým pohľadom** na funkcie a je problém overiť **korektnosť daného programu**
- Výnimkou tohoto správania sú **funkcionálne jazyky** (napr. Haskell), kde podprogramy **nemôžu spôsobiť vedľajšie efekty**
  - vo funkcionálnych jazykoch sú všetky podprogramy funkcie, nemá zmysel uvažovať o procedúrach bez návratovej hodnoty, keď nemôžu mať iný vedľajší efekt

Komunikácia medzi podprogramami

- Spôsob prenášania informácií medzi programom a podprogramami v obidvoch smeroch
- Odovzdávanie informácií
  - pomocou **parametrov podprogramu**
  - pomocou **globálnych údajových objektov**
  - pomocou **návratovej hodnoty**

Odovzdávanie parametrov

- V podprograme sú definované **formálne parametre** – údajové objekty s ktorými sa v podprograme manipuluje
- Pri volaní podporgramu sa špecifikujú **skutočné parametre**
- Skutočné parametre sa odovzdávajú cez formálne parametre do podprogramu dvoma spôsobmi
  - **odkazom** – formálnym parametrom sa sprístupňuje celý skutočný údajový objekt, ktorému následne môžeme zmeniť hodnotu
  - **hodnotou** – formálnym parametrom sa sprístupňuje iba hodnota skutočného údajového objektu, ktorému podprogram nemôže zmeniť hodnotu
- V jazyku C sú parametre odovzdávané iba hodnotou
  - odovzdávanie odkazom sa môže emulovať odovzdaním smerníka na údajový objekt

Použitie globálnych údajových objektov

- Globálne údajové objekty sú viditeľné (a použiteľné) **v rámci celého programu** (pre všetky podprogramy) resp. v rámci modulu
- **Každý podprogram, môže meniť hodnoty globálnych údajových objektov** a použiť ich ako vstup a výstup podprogramu
  - žiadne parametre sa podprogramu nemusia odovzdávať a preto je tento spôsob rýchlejší
- Používanie globálnych údajových objektov pre odovzdávanie parametrov podprogramom **spôsobuje vedľajšie efekty** a preto sa používa len v špecifických prípadoch

Odovzdanie návratovej hodnoty

- **Výstup z podprogramu** je definovaný návratovou hodnotou (vo funkciách)
- Návratová hodnota je **odovzdaná po návrate** z podprogramu na miesto odkiaľ bol podprogram vyvolaný
- Vrátením hodnoty z podprogramu sa činnosť podprogramu končí
- Zvyčajne je návratová hodnota z podprogramu prístupná priamo a je potrebné ju niekde uložiť alebo hneď použiť vo výraze

Funkcie v jazyku C

- Hlavný program je tiež definovaný ako funkcia s názvom **`main`**
- Viditeľnosť funkcie
  - implicitne je funkcia viditeľná (a použiteľná) v celom programe
  - statické funkcie sú viditeľné iba v jednom module (súbore zdrojového textu)
  - funkcie nie je možné definovať v rámci iných funkcií
- Definícia funkcie
  - popisuje funkciu z hľadiska jej **vstupov** (parametre), **výstupov** (návratová hodnota), **algoritmov** a použitých **údajových objektov**

```
[typ_návratovej_hodnoty] názov_funkcie(deklarácia_parametrov)
{ telo_funkcie }
```

Typ (návratovej hodnoty) funkcie

- Implicitne je typ funkcie `int`

```
sucet(int x, int y) {
  return x + y;
}
```

- Funkcia bez návratovej hodnoty (procedúra) je typu `void`

```
void vypis() {
  printf("Text výpisu");
}
```

- Funkcia bez parametrov by mala (odporúčanie normy ANSI C) použiť kľúčové slovo `void`

```
void vypis(void) {
  printf("Text výpisu");
}
```


Návrat z funkcie

- Pre návrat z funkcie sa používa kľúčové slovo `return` v príkaze

```
return hodnota;
```

- Návrat z procedúry používa slovo `return` samostatne bez návratovej hodnoty
- Návrat z funkcie sa vykoná automaticky aj po dosiahnutí posledného príkazu funkcie (po dosiahnutí uzatvárajúcej zátvorky bloku) – je možné použiť pri procedúrach
- Funkcia vracajúca hodnotu inú ako typu `void` musí použiť kľúčové slovo `return` pre návrat konkrétnej hodnoty

Deklarácia a prototyp funkcie

- Deklarácia funkcie
  - popisuje funkciu z hľadiska jej **rozhrania** (meno a typ návratovej hodnoty)

```
typ_funkcie názov_funkcie();
```

- Prototyp funkcie
  - rozšírená forma deklarácie s určením **(typov) vstupných parametrov** (odporúčanie normy ANSI C)

```
typ_funkcie názov_funkcie(deklarácie_parametrov);
```


Volanie funkcie

- Volanie funkcie sa vykoná použitím jej názvu a vymenovaním parametrov

```
názov_funkcie(zoznam_parametrov)
```

- Volanie funkcie s návratovou hodnotou je potrebné použiť vo výrazoch ako operand, aby sa návratová hodnota nestratila, ale využila

```
void main() {
  int a, b, c;
  ...
  c = 5 + sucet(a, b);
  ...
}
```

- Volanie funkcie bez návratovej hodnoty (procedúry) musí byť použité samostatne a nesmie byť použité ako operand vo výrazoch

```
void main() {
  ...
  vypis();
  ...
}
```