Řídicí struktury v programování

- **Funkce**
- **Příkazy**

- **Funkce**
  - **Obecný formát programu.**
  - **Deklarace funkce (prototyp).**
  - **Definice funkce (implementace).**
  - **Volání funkcí.**
  - **Funkce s parametry.**

  - **Obecný formát programu**
    - $/*\ zde\ vložit\ hlavičkové\ soubory$ (#include) $*/$
    - $/*\ zde\ umístit\ případné\ prototypy\ funkcí\ \ (Umožní\ to\ používat\ všechny\ funkce$
    - $ve\ všech\ jiných\ funkcích) */$
    - $/*\ globální\ definice\ */$
    - $návratovýtyp\ funkce1(seznam - parametrů)$
    - {
    - $/*\ tělo\ funkce1\ */$
    - }
    - .
    - .
    - $/*\ globální\ definice\ */$
    - $návratovýtyp\ funkceN(seznam - parametrů)$
    - {
    - $/*\ tělo\ funkceN\ \ \ldots\ldots\ */$
    - }
    - $int\ main(seznam - parametrů)$
    - {
    - $/*\ tělo\ funkce\ main\ */$
    - }
  - **Deklarace funkce (prototyp).**
    - návratový-typ jménoFunkce (seznam-parametrů);
    - deklarace funkce před jejím použitím
    - využití v hlavičkových souborech
    - Příklad:
      - double sqrt (double x);
  - **Funkce – příkaz return**
    - Příkaz return má obecný formát:
    - return expression_{opt} ;
    - ukončuje provádění funkce
    - může se vyskytnout kdekoli v těle funkce

- Parametr vs Argument

```
#include <stdio.h>
void tisk_sum(int x, int y);
int main(void)
{
  int num1, num2;
  printf("Zadejte dve cela cisla: ");
  scanf("%d%d", &num1, &num2);
  tisk_sum(num1,num2);
  return 0;
}
void tisk_sum(int x, int y)
{
  printf("Soucet cisel je: %d \n", x + y);
}
```

argumenty

(formální) parametry

- **Příkazy**
  - **Typy:**
    - Složený příkaz.
    - Výraz, příkaz.
    - Oblast platnosti identifikátorů.
    - Podmíněné příkazy.
    - Cykly.
    - Příkazy skoku.

  - **Syntaxe složeného příkazu:**
    - *compound-statement:*
      { block-item-list$_{opt}$ }
      block-item-list:
      block-item
      block-item-list block-item
      block-item:
      declaration
      statement
    - V dřívějších verzích jazyka nejdřív deklarace, pak výpočty Např. v Herout: Učebnice jazyka C, kap.: 3.2
    - **Příklad:**

```
void vypisPrevod(int volba)
{
  if(volba == 1) // stopy na metry
  {
    float num;
    printf("Zadejte pocet stop: ");
    scanf("%f", &num); // !!!
    printf("%f  stop  je %f metru\n", num, num/KOEF_PREVODU);
  }
  else if(volba == 2)  // metry na stopy
  {
    printf("Zadejte pocet metru: ");
    float num;
    scanf("%f", &num); // !!!
    printf("%f  metru  je %f stop\n", num, num*KOEF_PREVODU);
  }
  else
  {
    printf("Chybna volba.");
  }
}
```

složené příkazy

lokální definice

- **Oblast platnosti identifikátorů**
  - **Na úrovni souboru**
    - od místa deklarace
    - do konce překládaného modulu (zdrojového souboru)
  - **Deklarace parametru funkce (při definici)**
    - od místa deklarace parametru
    - do ukončení bloku definice funkce
  - **V rámci bloku**
    - od deklarace
    - do konce aktuálního bloku
  - **Příklad:**

```
char znak = 'A';
int main(void)
{
  int cislo = 1;
  printf("%c%d", znak, cislo);     // A1
  char znak = 'B';                 // zastínění
  printf("%c%d", znak, cislo);     // B1
  {
    char znak = 'C';               // zastínění
    printf("%c%d", znak,cislo);    // C1
    float cislo = 2.0;             // zastínění
    printf("%c%.2f", znak, cislo); // C2.00
  }
  printf("%c%d", znak, cislo);     // B1
}
```

  - **Makra a symbolické konstanty:**
    - od místa definice (#define)
    - do místa potlačení definice (#undef)
    - nebo do konce modulu
    - Nelze zastínit!
    - **Příklad:**

```
#define N 10
int pole[N] = {0};
#undef N
int lany[N*N] = {0}; // zde už hodnotu N
                     // nelze použít
```

- **Podmíněné příkazy**
  - **IF**
    - Syntax:

selection-statement:
if (*expression* ***)*** *statement*
if (*expression* ***)*** *statement* ***else*** *statement*
switch (*expression* ***)*** *statement*

- Poznámka k formátování

```
// pozor na formátování, aby odpovídalo
// skutečnosti
if (num < 0)
  printf(…);
  printf(…);  // !

// lépe takto
if (num < 0)
{
  printf(…);
}
printf(…);
```

- **Switch**
  - **Syntax:**

selection-statement:
switch (*expression* **)** *statement*
labeled-statement:
case *constant-expression* **:** *statement*
default : *statement*

  - **Příklad:**

```
switch(vyraz)
{
  case 1:
    printf("volba 1");
    break;
  case 2: // volby lze spojovat takto
  case 3:
    printf("volba 2 nebo 3");
    break;
//case 2: - nelze mít dvě stejná návěští
  default:
    printf("vsechny ostatni volby");
    break;
}
```

- **Příkazy cyklu**
  - **Syntax:**

iteration-statement:
while (*expression* **)** *statement*
do *statement* **while (** *expression* **) ;**
for ($expression_{opt}$ **;** $expression_{opt}$ **;** $expression_{opt}$ **)** *statement*
for (*declaration* $expression_{opt}$ **;** $expression_{opt}$ **)** *statement*

  - **Příkaz while**
    - Dopředu neznáme počet iterací
    - Cyklus nemusí proběhnout ani jednou

- **Příkaz do-while**
  - Opakuje příkaz (statement) dokud je výraz (expression) pravdivý
  - Příkazová část se provede **vždy alespoň jednou**
- **Příkaz for**
  - provede inkrementační část cyklu
  - pak test podmínky cyklu, cyklus (ne)pokračuje
- **Příkazy skoku**
  - *jump-statement:*
    goto *identifier* ***;***
    continue ;
    break ;
    return *expression*$_{opt}$ ***;***
  - **Příkaz break**
    - umožňuje ukončit cyklus v libovolném místě těla cyklu
    - zpracování programu pokračuje příkazem následujícím za cyklem
  - **Příkaz skoku**
    - Jedním z mála akceptovatelných použitím goto je vyskočení z hluboko zanořené části programu, pokud nastane fatální chyba.
    - **Nepoužívat! (téměř) nikdy!**
    - ```
      goto navesti;
      printf("Toto se nikdy nevytiskne.");
      navesti: printf("Toto se vytiskne.");
      ```
  - **Příkaz continue**
    - Příkaz continue si vynutí nové vyhodnocení podmínky cyklu, přičemž se přeskočí všechny příkazy mezi ním a koncem těla cyklu
    - **V cyklech while, do-while**
      - skok na test podmínky, (ne)pokračování cyklu.
    - **V cyklu for**
      - provede inkrementační část cyklu
      - pak test podmínky cyklu, cyklus (ne)pokračuje.
      - Jeden z vhodných případů použití continue je nové spuštění posloupnosti příkazů, když nastane chyba.
  - **Příkaz return**
    - Příkaz return ukončí provádění funkce, která tento příkaz obsahuje
    - Ve funkci main ukončí příkaz return celý program
      ```