

Programování v jazyce C pro chemiky

(C2160)

3. Příkaz switch, příkaz cyklu for, operátory ++ a --, pole

Příkaz switch

- Příkaz **switch** provede příslušnou skupinu příkazů na základě hodnoty proměnné (celočíselné nebo **char**) uvedené v záhlaví
- Za každým návěstím **case N** uvedeme příkazy které se mají vykonat, za nimi zpravidla následuje **break**, jinak by se pokračovalo vykonáváním příkazů za dalším case.

- Zapisuje se:

```
switch (proměnná)
{
    case číslo1 :
        příkazy1;
        break;
    case číslo2 :
        příkazy2;
        break;
    ...
    case čísloN :
        příkazyN;
        break;
    default:
        příkazy;
        break;
}
```

Příkaz switch - příklad

```
int a = 0;

scanf("%i", &a);

switch(a) {
    case 1:
        printf("Jednicka\n");
        break; // Zde vyskocime, jinak by se pokračovalo dalsim navestim
    case 2:
        printf("Dvojka\n");
        break;
    case 10:
        printf("Desitka\n");
        break;
    default:
        printf("Jina hodnota\n");
        break; // Tento break je vlastne zbytecny
}
```

Příkaz cyklu for

- Cyklus **for** se používá podobně jako **while**, umožňuje však některé další operace přímo v záhlaví cyklu
- Zapisuje se:

```
for (inicializace; podmínka; příkazy na konci cyklu)  
{  
    příkazy;  
}
```
- **Inicializace**: příkazy které se provedou před začátkem cyklu; nejčastěji přiřazení výchozích hodnot proměnným
- **Podmínka** se formuluje stejně jako u cyklu **while**
- **Příkazy na konci cyklu**: tyto příkazy se provedou po každém průchodu cyklem; zpravidla jde o změnu hodnot proměnných, např. zvýšení o jedničku a pod.
- Uvedené tři sekce v záhlaví cyklu oddělujeme **středníkem**

```
int i = 0, sum = 0;  
  
for (i = 1; i <= 100; i = i + 1)  
{  
    sum = sum + i;  
}
```

Příkaz cyklu for

- Každý cyklus **for** lze vždy nahradit ekvivalentním cyklem **while** (a obráceně)

```
for (inicializace; podmínka; příkazy na konci cyklu)  
{  
    příkazy;  
}
```
- Výše uvedenému cyklu **for** odpovídá následující cyklus **while**

```
inicializace;  
while (podmínka)  
{  
    příkazy;  
    příkazy na konci cyklu;  
}
```
- Kterákoli sekce záhlaví může být vynechána (středník je třeba zachovat). Je-li vynechána podmínka, považuje se za vždy splněnou.
- V cyklu **for** lze používat **break** úplně stejně, jako v cyklech **while** a **do while**

Příkaz cyklu for

- Pokud cyklus obsahuje jen jeden příkaz, je možné složené závorky vynechat (stejně jako u **if**), může to ale být na úkor přehlednosti

```
int i = 0;  
int sum = 0;  
  
for (i = 1; i <= 100; i = i + 1)  
    sum = sum + i;
```

Příkaz cyklu for

- Počínaje standardem C99 můžeme přímo v inicializační části cyklu **for** definovat proměnné
- Takto definovaná proměnná přestává existovat okamžikem opuštění cyklu

```
int sum = 0;
```

```
for (int i = 1; i <= 100; i = i + 1) {  
    sum = sum + i;  
}
```

```
// Tady muzeme dale pracovat s vyslednou hodnotou promenne sum  
// Promenna i uz tady ale neexistuje
```

Vnořené cykly

- Všechny typy cyklů (**while**, **do**, **for**) můžeme vzájemně vnořovat
- Při použití cyklu **for** používáme v záhlaví často celočíselnou proměnnou jejíž hodnota se změní po každém průchodu cyklem – tuto proměnnou pak nazýváme **řídící proměnná cyklu**
- Řídící proměnná se zpravidla jmenuje **i**, pro vnořené cykly pak používáme jména **j**, **k** a pod.

```
for (int i = 1; i <= 100; i = i + 1) {  
    // Nejake prikazy  
    for (int j = 100; j > 0; j = j - 1) {  
        // Nejake prikazy  
        for (int k = 1; k <= 100; k = k + 1) {  
            // Nejake prikazy  
        }  
        // Nejake prikazy  
    }  
    // Nejake prikazy  
}
```


Operátory ++ a --

- Operátor **++** se používá pro zvětšení hodnoty proměnné o 1
- Operátor **--** se používá pro zmenšení hodnoty proměnné o 1
- Tyto operátory se často používají v záhlaví cyklu **for**

```
int a = 0;
```

```
// Nasledujici dva prikazy provedou totez
```

```
a = a + 1;
```

```
a++;
```

```
// Typicke pouziti operatoru ++ v cyklu for
```

```
for (int i = 1; i < 10; i++) {
```

```
    // nejake prikazy
```

```
}
```

Příkaz `continue`

- Příkaz **`continue`** se používá v cyklech **`while`**, **`do`** a **`for`**
- Příkaz způsobí skok na konec cyklu a tím vynutí další iteraci, cyklus tedy není ukončen, ale pokračuje vyhodnocením podmínky cyklu, která rozhodne o dalším pokračování

```
for (int i = 0; i < 10; i++) {  
    // Je-li i rovno 2, provede se continue,  
    // preskoci se prikazy za nim  
    if (i == 2) {  
        continue;  
    }  
    printf("%i\n", i);  
}
```

Jednorozměrná pole

- Pole (angl. *array*) je speciální proměnná, která obsahuje více prvků téhož typu
- Pole definujeme podobně jako proměnnou, za ní v hranatých závorkách uvedeme počet prvků pole
- Výchozí hodnoty prvků pole lze specifikovat ve složených závorkách (tzv. inicializace pole)
- Hodnoty jednotlivých prvků pole získáme pomocí **[index]**
- POZOR: Prvky pole jsou v jazyce C **číslovány od 0**, tj. indexy nabývají hodnot **0 až (počet prvků - 1)**

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
------	------	------	------	------	------	------	------	------	------

```
int a[10];    // Definujeme pole celých čísel o deseti prvcích
int b[4] = {2, 4, -5, 7}; // Hodnoty prvků pole lze inicializovat
int c = 0;
```

```
// Ukazka prace s prvky pole
c = b[0] * b[1] + b[2] * b[3];
a[2] = c + b[0];
```

Inicializace polí

- Inicializujeme-li jen část prvků pole, všechny ostatní prvky budou automaticky inicializovány nulami
- Složené závorky musíme použít i v případě, že inicializujeme jen jednou hodnotou
- Standard C99 přidává možnost inicializovat jen konkrétní vybrané prvky syntaxí [*index*] = *hodnota*. Indexy nemusí být seřazené (ale většinou to tak je přehlednější)
- Pokud ne zadáme explicitně velikost pole, bude určena podle počtu hodnot ve složených závorkách nebo nejvyššího indexu

```
int a[5] = {1, 2};           // Ekvivalent {1, 2, 0, 0, 0}
int b[5] = { [2] = 1, [4] = 1}; // Ekvivalent {0, 0, 1, 0, 1}

int c[] = {1, 2, 3}; // Odpovídá int c[3] = {1, 2, 3};

int d[] = {[5] = 1}; // Odpovídá int d[6] = {0, 0, 0, 0, 0, 1};
```

Pole - příklad 1

Program pro výpočet součtu dvou 3-dimenzionálních vektorů:

```
int main()
{
    float va[3] = {0.0, 0.0, 0.0};
    float vb[3] = {0.0, 0.0, 0.0};
    float vc[3] = {0.0, 0.0, 0.0};

    printf("Zadejte souradnice prvnio vektoru: ");
    scanf("%f %f %f", &va[0], &va[1], &va[2]);
    printf("Zadejte souradnice druhoho vektoru: ");
    scanf("%f %f %f", &vb[0], &vb[1], &vb[2]);

    for (int i = 0; i < 3; i++) {
        vc[i] = va[i] + vb[i];
    }

    printf("Soucet vektoru: %f, %f, %f\n", vc[0], vc[1], vc[2]);

    return 0;
}
```

Symbolické konstanty

- Při programování často používáme tzv. symbolické konstanty
- Symbolickou konstantu definujeme následovně:
`#define JMENO HODNOTA`
- Překladač před zahájením překladu nahradí výskyt všech jmen symbolické konstanty JMENO její hodnotou HODNOTA
- Symbolické konstanty specifikujeme na začátku programu, ještě před definicí funkce `main()`
- Jména symbolických konstant píšeme zpravidla velkými písmeny
- Symbolické konstanty se často používají pro určení velikosti polí

```
// definujeme symbolickou konstantu SIZE s hodnotou 4
#define SIZE 4

int main()
{
    int a[SIZE]; // Definujeme pole velikosti SIZE

    // Zde bychom nastavili nějaké hodnoty pole a[]

    for (int i = 0; i < SIZE; i++)
        printf("Prvek pole %i ma hodnotu %i\n", i, a[i]);
}
```

Pole - příklad 2

Program pro výpočet součtu dvou n -dimenzionálních vektorů:

```
#define SIZE 4

int main()
{
    float va[SIZE] = {0.0}, vb[SIZE] = {0.0}, vc[SIZE] = {0.0};

    // Nacteni hodnot od uzivatele
    printf("Zadejte %i souradnic prvnioho vektoru", SIZE);
    for (int i = 0; i < SIZE; i++) {
        scanf("%f", &va[i]);
    }
    //... Podobne se nacte druhy vektor

    for (int i = 0; i < SIZE; i++) {
        vc[i] = va[i] + vb[i];
    }

    printf("Vysledny vektor je: ");
    for (int i = 0; i < SIZE; i++) {
        printf("%f ", vc[i]);
    }
    printf("\n");
    return 0;
}
```

Vícerozměrná pole

- Vícerozměrná pole jsou pole s více než jednou dimenzí
- Dvoudimenzionální pole jsou vlastně pole polí
- První index vybírá prvek vnějšího pole, další indexy potom prvky vnořených polí. Při iteraci většinou nejvnitřnější smyčka běží přes index nejvíc vpravo.

Příklad: Program na sčítání matic o 2 řádcích a 3 sloupcích:

```
float ma[2][3] = {{5.0, 7.0, 9.0}, {-2.0, 4.0, -6.0}};  
float mb[2][3] = {{1.0, 4.0, -4.0}, {2.0, 8.0, 5.0}};  
float mc[2][3] = {{0.0, 0.0, 0.0}, {0.0, 0.0, 0.0}};  
  
for (int i = 0; i < 2; i++) {  
    for (int j = 0; j < 3; j++) {  
        mc[i][j] = ma[i][j] + mb[i][j];  
    }  
}  
  
// Zde by se vypsala výsledná matice
```


Dodržujte následující pravidla

- Na začátku každého programu uveďte stručný komentář, vysvětlující, co program dělá
- Všechny proměnné vždy inicializujte vhodnou implicitní hodnotou při její definici
- Pole také inicializujte při jejich definici. K inicializaci všech prvků pole nulami stačí inicializovat jeden prvek
- Při používání cyklů upřednostňujte cyklus **for**. Cykly **while** a **do** používejte pouze tam, kde nepoužíváte řídicí proměnnou cyklu
- Dbejte na přehledné a konzistentní odsazování textu

Úlohy - část 1

1. Upravte program ze cvičení 1 / úloha 3 tak, aby využíval příkazu **switch** místo série podmínek. **1 bod**
2. Vytvořte program, který od uživatele vyžádá souřadnice dvou 3-dimenzionálních vektorů, spočítá jejich vektorový součin a výsledný vektor vypíše na obrazovku. **1 bod**
3. Vytvořte program, který od uživatele načte N celých čísel do pole (N bude specifikováno symbolickou konstantou s výchozí hodnotou 20). Dále definujte jiné pole, do kterého program zkopíruje všechna sudá čísla z prvního pole (do po sobě jdoucích pozic od prvního prvku). Na konci program vypíše počet nalezených sudých čísel a jejich hodnoty. **1 bod**

Úlohy - část 2

4. Vytvořte program, který vynásobí dvě uživatelem zadané čtvercové matice libovolné dimenze (dané symbolickou konstantou s výchozí hodnotou 3). **volitelná 1 bod**
5. Vytvořte program, který spočítá prvních 40 Fibonacciho čísel (za první a druhé Fibonacciho číslo považujte jedničky). Tato čísla uloží do pole. Dále program určí, kolik z čísel v tomto poli je dělitelných 2, 3, 4, 5, 6, 7, 8, 9 a 10. K uložení zjištěných počtů využijte druhé pomocné pole. Výsledky se nakonec vypíší na obrazovku. Také se vypíše, kolik čísel nebylo dělitelných žádným z čísel 2 až 10. **volitelná 2 body**

Fibonacciho čísla: https://en.wikipedia.org/wiki/Fibonacci_number