

Procedurálne programovanie

slido

slido.com
#4095204
PrPr – P2

Ján Zelenka
Ústav Informatiky
Slovenská akadémia vied



Obsah prednášky

1. Opakovanie
 - operátory
2. Riadiace štruktúry
 - Vetvenie (?:, if-else, switch)
 - Cykly (while, do-while, for)
3. Statické jednorozmerné pole

Spätná väzba: <https://forms.gle/6q5D2G6UwrtimXEx9>

Opakovanie

Operátory

Obsah prednášky

1. Úvod do predmetu

Podmienky absolvovania

2. Algoritmy a paradigmy programovania

Procedurálna paradigma

3. Základy programovania v jazyku C

Kompilovanie programov v jazyku C

Premenné

Terminálový vstup a výstup

Operátory

Aritmetické výrazy

- **Unárne operátory**
- Bibárne operátory
- Relačné operátory
- Logické operátory
- Bitové operátory

- plus (+) (používanie v bežnom význame)
- mínus (-) (používanie v bežnom význame)
- špeciálne unárne operátory
inkrement (++), dekrement (--)

```
...  
x = +5;  
y = -7;  
...
```

Operátory

Obsah prednášky

1. Úvod do predmetu

Podmienky absolvovania

2. Algoritmy a paradigmy programovania

Procedurálna paradigma

3. Základy programovania v jazyku C

Kompilovanie programov v jazyku C

Premenné

Terminálový vstup a výstup

Operátory

Aritmetické výrazy

- Unárne operátory
- **Bibárne operátory**
- Relačné operátory
- Logické operátory
- Bitové operátory

či je delenie celočíselné alebo reálne závisí na type operandov:

int / int \Rightarrow celočíselné

int / float \Rightarrow reálne

float / int \Rightarrow reálne

float / float \Rightarrow reálne

- sčítanie (+)
- odčítanie (-)
- násobenie (*)
- reálne delenie (/)
- celočíselné delenie (/)
- zvyšok po delení celým číslom - modulo (%)

Operátory

Obsah prednášky

1. Úvod do predmetu

Podmienky absolvovania

2. Algoritmy a paradigmy programovania

Procedurálna paradigma

3. Základy programovania v jazyku C

Kompilovanie programov v jazyku C

Premenné

Terminálový vstup a výstup

Operátory

Aritmetické výrazy

- Unárne operátory
- **Bibárne operátory**
- Relačné operátory
- Logické operátory
- Bitové operátory

```
x += vyraz
x -= vyraz
x *= vyraz
x /= vyraz
x %= vyraz
```

či je delenie celočíselné alebo reálne závisí na type operandov:

int / int \Rightarrow celočíselné

int / float \Rightarrow reálne

float / int \Rightarrow reálne

float / float \Rightarrow reálne

- sčítanie (+)
- odčítanie (-)
- násobenie (*)
- reálne delenie (/)
- celočíselné delenie (/)
- zvyšok po delení celým číslom - modulo (%)

Operátory

Obsah prednášky

1. Úvod do predmetu

Podmienky absolvovania

2. Algoritmy a paradigmy programovania

Procedurálna paradigma

3. Základy programovania v jazyku C

Kompilovanie programov v jazyku C

Premenné

Terminálový vstup a výstup

Operátory

Aritmetické výrazy

- Unárne operátory
- Bibárne operátory
- **Relačné operátory**
- Logické operátory
- Bitové operátory

- **<, >, <=, >=, ==, !=**

Výsledkom je logická hodnota

- TRUE: nenulová hodnota
- FALSE: 0

Operátory

Obsah prednášky

1. Úvod do predmetu

Podmienky absolvovania

2. Algoritmy a paradigmy programovania

Procedurálna paradigma

3. Základy programovania v jazyku C

Kompilovanie programov v jazyku C

Premenné

Terminálový vstup a výstup

Operátory

Aritmetické výrazy

- Unárne operátory
- Bibárne operátory
- Relačné operátory
- **Logické operátory**
- Bitové operátory

- logické spojky z výrokovej logiky
 - `||` - disjunkcia (or, alebo)
 - `&&` - konjunkcia (and, a súčasne);
 - `!` - negácia (not, opak)
- **skrátené vyhodnocovanie**
(Short-circuit evaluation)

```
if ( (1==2) && (hocico() == 1) ) ...
```

Nikdy sa nezavolá!

Operátory

Obsah prednášky

1. Úvod do predmetu

Podmienky absolvovania

2. Algoritmy a paradigmy programovania

Procedurálna paradigma

3. Základy programovania v jazyku C

Kompilovanie programov v jazyku C

Premenné

Terminálový vstup a výstup

Operátory

Aritmetické výrazy

- Unárne operátory
- Bibárne operátory
- Relačné operátory
- Logické operátory
- **Bitové operátory**

- | | |
|----------------------------|---------------------------|
| & -> AND, | -> OR |
| ~ -> INVERT, | ^ -> XOR |
| << -> LSHIFT, | >> -> RSHIFT |

Operátory

Obsah prednášky

1. Úvod do predmetu

Podmienky absolvovania

2. Algoritmy a paradigmy programovania

Procedurálna paradigma

3. Základy programovania v jazyku C

Kompilovanie programov v jazyku C

Premenné

Terminálový vstup a výstup

Operátory

Aritmetické výrazy

- Unárne operátory
- Bibárne operátory
- Relačné operátory
- Logické operátory
- Bitové operátory

1. ++ (inkrement), -- (dekrement),
! (logická negácia)
2. *, /, %
3. +, -
4. <, >, <=, >=
5. ==, !=
6. && (logická konjunkcia)
7. || (logická disjunkcia)
8. = (priradenie)

*Väčšina operátorov sa vyhodnocuje
zľava doprava (sú zľava asociatívne)*

Výnimkou sú: !, ++, -- a =
(viacnásobné priradenie a = b = c,)

Poradie vyhodnotenia operátorov

Aritmetické operátory a operátory porovnania majú väčšiu prioritu ako logické operátory

```
(( c >= 'A' ) && ( c <= 'Z' ) )
```

Zátvorky tam nemusia byť, pretože `>=` a `<=` má väčšiu prioritu ako `&&`

- ak si nie ste istí, či zátvorky dať, radšej ich uveďte
- nezamieňajte `&&` za `&` a `||` za `|` - `&` a `|` sú bitové operácie

Riadiace štruktúry

Vnorené vetvenie

Na jednoduché vetvenie programu môžeme použiť príkaz **if**. Je definovaný v dvoch formách ako:

Neúplný:

```
if (podmienka)
    prikaz
```

ak platí podmienka,
vykoná sa prikaz

Úplný:

```
if (podmienka)
    prikaz_1;
else {
    prikaz_2;
    prikaz_3;
}
```

ak platí podmienka,
vykoná sa prikaz_1,
inak sa vykoná prikaz_2
a prikaz_3

Vnorené vetvenie

- ak je v sebe vnorených viac príkazov `if`, tak `else` patrí vždy k najbližšiemu `if`-u
- zjednodušenie cez logické výrazy

```
if (a == 0) {  
    if (b == 0) {  
        ...  
    }  
}
```

```
if (a == 0 && b == 0) {  
    ...  
}
```

```
if (prom = 12)  
    printf("True");  
else  
    printf("False");
```

Podmieňený výraz

Ternárny operátor

```
podmienka ? vyraz_1 : vyraz_2
```

Význam:

ak podmienka **tak** vyraz_1, **inak** vyraz_2

```
int i, k, j = 2;  
  
i = (j == 2) ? 1 : 3;  
k = (i > j) ? i : j;
```

i bude 1, pretože j == 2

k bude maximum z i a j

Mnohonásobné vetvenie

```
if (c == 'a')  
    ...  
else if (c == 'b')  
    ...  
else if (c == 'c')  
    ...  
else if (c == 'd')  
    ...  
else  
    ...
```

jednoduchšie: príkazom `switch`

Príkaz `switch`

- výraz, podľa ktorého sa rozhoduje, musí byť typu `int` alebo typu, ktorý sa dá naň previesť
- iné celočíselné typy (`char`, `short`, `long`)
- každá vetva sa ukončuje príkazom `break`
- v každej vetve môže byť viac príkazov, ktoré nie je nutné uzatvárať do zátvoriek
- vetva `default` - vykonáva sa, keď žiadna iná vetva nie je splnená

```
switch (vyraz) {  
    case hodnota_1 : prikaz_1; break;  
    ...  
    case hodnota_n : prikaz_n; break;  
        default : prikaz_def; break;  
}
```

Príkaz `switch`

- ak je viac hodnôt, pre ktoré chceme vykonať rovnaký príkaz (napr. hodnoty `h_1`, `h_2`, `h_3`):

```
switch (vyraz) {  
    case h_1 :  
    case h_2 :  
    case h_3 : prikaz_123; break;  
    case h_4 : prikaz_4; break;  
    default  : prikaz_def; break;  
}
```

- ak nie je vetva ukončená príkazom *break*, program neopustí *switch*, ale spracováva nasledujúcu vetvu v poradí - až po najbližšie *break*, alebo konca *switch*

Príkaz `switch`

- príkaz `break`
 - ruší najvnútornejšiu slučku cyklu, alebo
 - ukončuje príkaz `switch`
- treba dávať pozor na cyklus vo vnútri `switch` a naopak
- vetva `default` nemusí byť ako posledná, z konvencie sa tam dáva
- ak je vetva `default` na konci, nie je `break` nutný, dáva sa z konvencie

Zhrnutie

- operátor priradenia: `=`
- operátor pre porovnanie: `==`
- logické `&&` (AND) a `||` (OR) majú skrátene vyhodnocovanie
- pre ukončenie slučky cyklu - príkaz `break`
- za každou vetvou príkazu `switch` musí byť `break`
- ak nie je, vetvy musia súvisieť
- ak si nie ste istí s prioritami, zátvorkujte

Špeciálne unárne operátory +/- 1

- inkrement (++)
- dekrement (--)

```
i++;
```

```
j--;
```

- oba operátory sa dajú použiť ako:

1. prefix: ++**vyraz**

```
++i;
```

```
--j;
```

- inkrementovanie **pred** použitím
- **vyraz** je zvýšený o jednotku a potom je táto nová hodnota vrátená ako hodnota výrazu

2. postfix: **vyraz**--

```
i--;
```

```
j++;
```

- inkrementovanie **po** použití
- je vrátená pôvodná hodnota **vyraz-u** a potom je výraz zväčšený o jednotku

Špeciálne unárne operátory +/- 1

~~45++;~~ ++i; ~~--(i + j);~~

- použitie operátorov ++ a -- :

```
int i = 5, j = 1, k;
```

```
i++;
```

i bude 6

```
j = ++i;
```

j bude 7, i bude 7

```
j = i++;
```

j bude 7, i bude 8

```
k = --j + 2;
```

k bude 8, j bude 6, i bude 8

Iteračné príkazy

- umožňujú opakovať vykonávanie príkazu alebo bloku príkazov
- tri príkazy: *for*, *while*, *do-while*

Cyklus for

Používa sa, keď dopredu vieme počet prechodov cyklom

- nie nutne fixný počet, ale ukončovacia podmienka je rovnaká (prejdi pole od začiatku do konca)

```
for(inicializacny_vyraz; podmienka_vykonania; inkrementalny_vyraz){  
    //telo cyklu...príkazy;  
}
```

- `inicializacny_vyraz` - vykoná sa raz
 - nastaví sa riadiace premenné
- `podmienka_vykonania` - ak je TRUE vykonajú sa príkazy z tela cyklu
- `inkrementalny_vyraz` - vykoná sa na konci každej iterácie
 - typicky zmena riadiacej premennej
- jednotlivé výrazy nemusia spolu súvisieť, ani byť uvedené. Vždy treba napísať bodkočiarku.

Cyklus while

- cyklus iteruje pokým platí **podmienka_vykonania**:
 - test riadiacej premennej voči jej končovej hodnote
 - väčšinou sa riadiaca premenná modifikuje v tele cyklu

```
while (podmienka_vykonania){  
    //telo cyklu...prikazy  
    //zvycajne zmena riadiacej premennej  
}
```

- testuje podmienku **pred** prechodom cyklu
 - cyklus teda nemusí prebehnúť ani raz

Cyklus while

Telo cyklu môže byť aj prázdne:

```
while (podmienka_vykonania)  
    ;
```

- používame ho, keď ukončovacia podmienka závisí na nejakom príkaze v tele cyklu
 - opakuj cyklus kým nenastane chyba, si nedošiel na koniec súboru...
 - ak nie, podmienka by bola splnená stále a cyklus by bol nekonečný

Cyklus do-while

- testuje podmienku **po** prechode cyklu
 - cyklus sa vykoná aspoň raz!

```
do {  
    // telo cyklu...príkazy  
    // zvyčajne zmena riadiacej premennej  
}while (podmienka_vykonania);
```

Operátor sekvenčného vykonania ,

- Zreťazuje vykonávanie niekoľkých výrazov:
 - vyhodnotia sa všetky zľava doprava
- Je ho možné umiestniť tam, kde sa dá umiestniť jeden výraz
 - výstup zreťazenia je výstup posledného výrazu
- **Nepoužívajte ho mimo riadiacich príkazov cyklov**
 - vyhodnotenie predchádzajúcich výrazov je ignorované
 - sťažuje čitateľnosť kódu

```
int x = 7;  
if (x == 10) printf("x == 10");
```

```
if (x=10,x == 10) printf("x == 10");
```

Cyklus `for` vs Cyklus `while` (`do-while`)

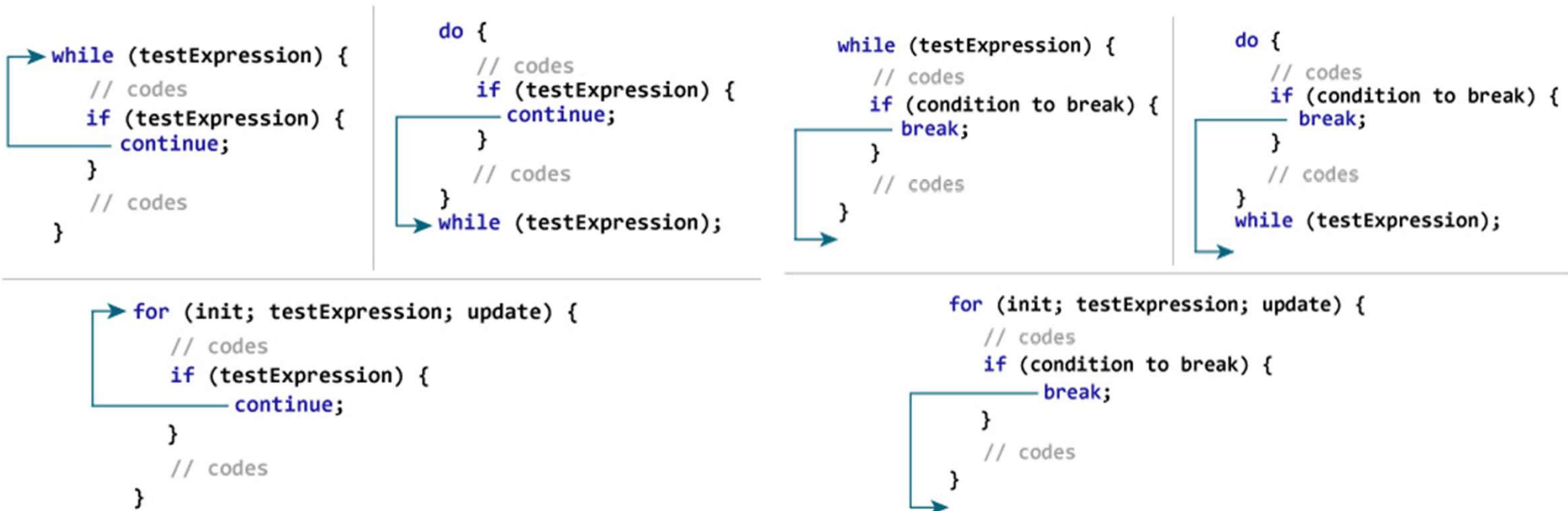
```
for(inicializacny_vyraz; podmienka_vykonania; inkrementalny_vyraz){  
    //telo cyklu...prikazy;  
}
```

```
inicializacny_vyraz;  
while(podmienka_vykonania){  
    //telo cyklu...prikazy;  
    inkrementalny_vyraz;  
}
```

```
inicializacny_vyraz;  
do{  
    //telo cyklu...prikazy;  
    inkrementalny_vyraz;  
}while(podmienka_vykonania);
```

Predčasné ukončenie cyklu

- *break* – ukončenie cyklu a pokračovanie za cyklom
- *continue* – ukončenie tela cyklu a pokračovanie ďalšou iteráciou
- *goto* – neodporúča sa používať (v štruktúrovanom programovacom jazyku sa mu dá vždy vyhnúť)



Zdroj:

<https://www.programiz.com/c-programming/c-break-continue-statement>

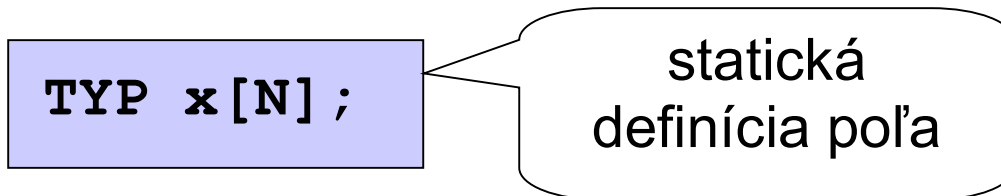
Odporúčania pri používaní cyklov

- mať len jednu riadiacu premennú
- riadiaca premenná má byť ovplyvňovaná len v riadiacej časti cyklu, nie v jeho tele
- inicializácia v inicializačnej časti
- ak má cyklus (nie len *for*) prázdne telo, bodkočiarku dať na nový riadok
- príkaz *continue* je vhodné nahradiť *if-else* konštrukciou
- príkaz *break* - len v najnutnejších prípadoch, najlepšie maximálne na jednom mieste
- cykly *while* a *for* sú prehľadnejšie ako *do-while*, preto ich uprednostňujte

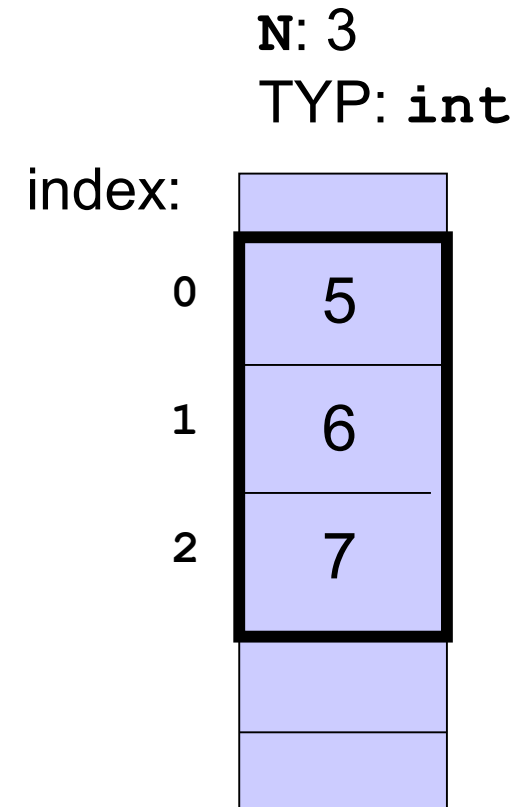
Statické jednorozmerné pole

Základy práce s poliami

- pole je štruktúra zložená z niekoľkých prvkov rovnakého typu (blok prvkov)



- pole obsahuje **N** prvkov
- dolná hranica je vždy 0
 - \Rightarrow horná hranica je **N-1**
- číslo **N** musí byť známe v čase prekladu
- hodnoty nie sú inicializované na 0
- hranica pola nie je kontrolovaná



Príklady definícií statického poľa

definícia konštanty
(inak sa nejedná o
statické pole!)

```
#define N 10  
  
int x[N], y[N+1], z[N*2];
```

x má	10	prvkov poľa, od indexu	0	po index	9
y má	11	prvkov poľa, od indexu	0	po index	10
z má	20	prvkov poľa, od indexu	0	po index	19

Prístup k prvkom poľa

```
#define N 10
```

```
...
```

```
int x[N], i;
```

```
x[0] = 1;
```

```
for (i = 0; i < N; i++)  
    x[i] = i+1;
```

```
for (i = 0; i < N; i++)  
    printf("x[%d]: %d\n", i, x[i]);
```

priradenie hodnoty do
prvého prvku poľa

v cykle priradenie
hodnoty postupne
všetkým prvkom poľa

výpis prvkov poľa

Prístup k prvkom poľa

Kompilátor nekontroluje rozsah hodnôt (range-checking) t.j. či index je mimo rozsahu poľa

```
x[10] = 22;
```

- program sa skompiluje, ale hodnota 22 sa zapíše na zlé miesto v pamäti
- prepísanie obsahu iných premenných
- prepísanie časti kódu

Inicializácia poľa

```
int A[3] = { 1, 2, 3 };
```

```
int B[4]; //spravne
```

```
B[4]={ 1, 2, 3, 4 }; //nespravne
```

```
B[0]=1; B[1]=2; B[2]=3; B[3]=4; //spravne
```

```
double C[5]={5.1, 6.9};
```

```
double C[5]={0};
```

pole tu
nedefinujeme

meníme prvky
existujúceho pola

jednoduchá
inicializácia
všetkých prvkov
na 0

inicializuje
C[0]=5.1,
C[1]=6.9 a
C[2]..C[4]=0

Porovnávanie poľa

- nie je možné vykonať pomocou operátora ==
- neporovná sa obsah poľa, ale adresa
- meno poľa bez [] vracia adresu na začiatok poľa (väčšinou prvý prvok – závisí od kompilátora)
- treba vykonať prvok po prvku

```
for (int i = 0; i < N; i++) {  
    if(A[i] != B[i]) {  
        return 0; // false  
    }  
}  
return 1; // true
```

Kopírovanie poľa

- nie je možné vykonať pomocou operátora =
- neskompiluje sa
- treba vykonať prvok po prvku

```
for (int i = 0; i < N; i++) {  
    B[i] = A[i];  
}
```

Ďakujem vám za pozornosť!

Spätná väzba: <https://forms.gle/6q5D2G6UwrtimXEx9>