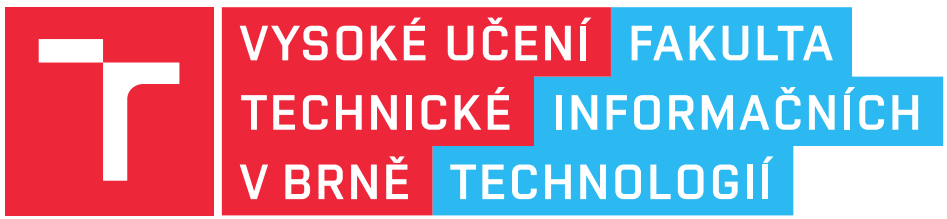
Základy programovania
Cvičenie 6

Úvod do ukazovateľov

Cvičiaci: Ing. Magdaléna Ondrušková, (iondruskova)



22. října 2024

Operátor referencie

- Všetky dáta programu sú uložené na nejakej adrese v operačnej pamäti počítača
- Adresu premennej získame pomocou **operátoru referencie** `&`

Operátor dereferencie

- Adresy ukladáme do premenných dátového typu **ukazovateľ (pointer)**.
- Hodnotu z adresy získame **operátorom dereferencie** `*`

```
1 int x = 10;
2 int *p; // Deklaracia ukazovatela na int
3 p = &x; // Inicializacia ukazovatela adresou premennej x
4
5 printf("Hodnota x je: %d\n", x);    // Vypis hodnoty x
6
7 // Vypis hodnoty, na ktoru ukazuje p
8 printf("Hodnota na ktoru ukazuje p je: %d\n", *p);
```

Deklarácia ukazovateľov

- Ukazovatele sú špeciálne premenné, ktoré slúžia na uloženie adresy namiesto hodnoty.

Základný spôsob deklarácie ukazovateľov:

```
1 int* p;
```

Iné spôsoby deklarácie ukazovateľov:

```
1 int *p1;
2 int * p2;
```

Príklad: Čo som deklarovala tu?

```
1 int* p1, p2;
```

Priradenie adresy ukazovateľu

```
1 int* pc, c;
2 c = 5;
3 pc = &c;
```

Získanie hodnoty kam ukazuje ukazovateľ

```
1 int* pc, c;
2 c = 5;
3 pc = &c;
4 printf("%d", *pc);    // Vypise: 5
```

Pozor! Ukazovateľ je iba `pc`, NIE `*pc`. Toto je nesprávne: `*pc = &c` !

Zmena hodnoty premennej, na ktorú ukazuje ukazovateľ

```
1 int* pc, c;
2 c = 5;
3 pc = &c;
4 c = 1;
5 printf("%d", c);     // Vystup: 1
6 printf("%d", *pc);   // Vystup: 1
```

Zmena hodnoty pomocou ukazovateľa

```
1 int* pc, c;
2 c = 5;
3 pc = &c;
4 *pc = 1;
5 printf("%d", *pc);   // Vystup: 1
6 printf("%d", c);     // Vystup: 1
```

Príklad - práca s ukazovateľmi

```c
#include <stdio.h>
int main()
{
    int* pc, c;

    c = 22;
    printf("Address of c: %p\n", &c);
    printf("Value of c: %d\n\n", c);  // 22

    pc = &c;
    printf("Address of pointer pc: %p\n", pc);
    printf("Content of pointer pc: %d\n\n", *pc); // 22

    c = 11;
    printf("Address of pointer pc: %p\n", pc);
    printf("Content of pointer pc: %d\n\n", *pc); // 11

    *pc = 2;
    printf("Address of c: %p\n", &c);
    printf("Value of c: %d\n\n", c); // 2
    return 0;
}
```

Priradenie hodnoty premennej c do ukazovateľa pc

- `c` je obyčajná celočíselná premenná. Ukladá hodnotu typu `int`
- `pc` je ukazovateľ, ktorý má ukladať **pamäťovú adresu** - číselnú hodnotu reprezentujúcu miesto v pamäti
- Chyba nastáva, pretože priraďujeme hodnotu typu **int** do premennej, ktorá očakáva **pamäťovú adresu**.
- V C nie je priamo možné priradiť obyčajnú celočíselnú hodnotu do ukazovateľa, pretože ukazovateľ potrebuje byť inicializovaný platnou adresou (napr. pomocou operátora `&`).

```
1 int c, *pc;
2 pc = c;  // Chyba
```

Nesprávny typ priradenia

- `*pc` - obsahuje hodnotu, na ktorú ukazovateľ ukazuje. Očakáva hodnotu typu `int`
- `&c` - pamäťová adresa premennej c, teda hodnota typu `ukazovateľ na int` - adresa v pamäti
- Pokúšame sa priradiť pamäťovú adresu do hodnoty typu int.
- `*pc` očakáva obyčajnú celočíselnú hodnotu, nie adresu.

```
1 int c, *pc;
2 *pc = &c;  // Chyba
```

Priradenie adresy

- `pc` - ukazovateľ, ktorý očakáva pamäťovú adresu premennej `c`
- `&c` - Operátor `&` vracia pamäťovú adresu premennej `c`
- Ukazovateľ `pc` teraz ukazuje na pamäťovú adresu premennej `c`. Pamäťová adresa je kompatibilná s typom ukazovateľa `pc`, preto je príkaz správny.
- Po vykonaní tohoto príkazu ukazovateľ `pc` obsahuje adresu premennej `c`, a teda ho môžeme dereferencovať, aby sme pracovali s hodnotou premennej `c`.

```
1 int c, *pc;
2 // obe premenne &c a pc su adresa
3 pc = &c;
```

Priradenie hodnoty

- `*pc` - dereferencovaný ukazovateľ, teda hodnota, na ktorú ukazovateľ ukazuje. Keďže `pc` ukazuje na premennú `c`, tak `*pc` je odkaz na hodnotu premennej `c`
- `c` - obyčajná číselná premenná typu int.
- Hodnota premennej `c` sa priradí na miesto, na ktoré ukazuje ukazovateľ `pc`.

```
1 int c, *pc;
2 // obe premenne &c a pc su hodnoty
3 *pc = c;
```

```c
#include <stdio.h>
int main() {
    int c = 5;
    int *p = &c;

    printf("%d", *p);  // 5
    return 0;
}
```

Otázka: Vie niekto, prečo sme nedostali chybu pri príkaze `int *p = &c;`?

NULL ukazovatele

- Pri vytvorení ukazovateľa, ukazovateľ môže ukazovať na náhodné miesto v pamäti
- Ošetríme to priradením špeciálnej hodnoty `NULL`
- Na signalizáciu, že ukazovateľ ešte nebol inicializovaný/neexistuje platný cieľ, na ktorý by ukazovateľ ukazoval

```
1 int *p = NULL;
2 if (p == NULL) {
3     printf("Ukazovatel je neplatny.\n");
4 }
```

Ukazovatele na ukazovatele

- Ukazovateľ môže ukazovať aj na iné ukazovatele.
- Využitie pri zložitejších dátových štruktúrach alebo dynamických poliach.
- V tomto príklade `pp` je ukazovateľ, ktorý ukazuje na ukazovateľ `p`, ktorý ukazuje na premennú `x`. Pomocou `**pp` sa dostaneme až k hodnote `x`.

```
1 int x = 100;
2 int *p = &x;  // Ukazovatel na x
3 int **pp = &p; // Ukazovatel na ukazovatel p
4
5 printf("Hodnota x cez pp: %d\n", **pp);
```

Predávanie parametrov hodnotou (Pass by value)

- Funkcia pracuje s kópiou premennej, nie s originálnou premennou
- Zmeny v rámci funkcie nemajú vplyv na pôvodnú premennú
- Bezpečnejšie pre jednoduché typy.
- Môže byť neefektívne pre veľké dátové štruktúry - kopírovali by sa všetky údaje

Príklad:

```
1  void zvysOJedno(int x) {
2      x++; // Zmen me  k piu premennej
3      printf("Hodnota x vo funkcii: %d\n", x); // Toto uk   e
       zmenen  hodnotu len vo funkcii
4  }
5  int main() {
6      int a = 5;
7      zvysOJedno(a); // Tu sa odovzda kopia hodnoty premennej a
8      printf("Hodnota a v main: %d\n", a); // Hodnota a ostane
       nezmenena
9      return 0;
10 }
```

Výstup:

```
1  Hodnota x vo funkcii: 6
2  Hodnota a v main: 5
```

Predávanie parametrov odkazom (Pass by reference)

- Funkcia pracuje s originálnou premennou cez jej adresu
- Zmeny v rámci funkcie sa prejavia aj mimo nej
- Efektívne pre veľké dátové štruktúry (napr. polia), pretože nepracujeme s kópiami, ale priamo s pamäťou.

Príklad:

```c
#include <stdio.h>

void zvysOJedno(int *x) {
    (*x)++; // Dereferencujeme ukazovate a zmen me hodnotu na adrese
}

int main() {
    int a = 5;
    zvysOJedno(&a); // Odovzd me adresu premennej a
    printf("Hodnota a po zavolan funkcie: %d\n", a); // Hodnota a sa zmen
    return 0;
}
```

Výstup:

```
Hodnota a po zavolan funkcie: 6
```

Príklad

- Vytvorte pole čísel typu `int` a načítajte do neho hodnoty zo vstupu
- Napíšte funkciu **najdiNajvacsie**, ktorá vráti ukazovateľ na najväčší prvok v poli
- Napíšte funkciu **najdiNajmensie**, ktorá vráti ukazovateľ na najmenší prvok v poli
- Napíšte funkciu **vymenCisla**, ktorá vymení najväčšie a najmenšie číslo
- Vypíšte pole po výmene čísiel.
- Neriešte špeciálne prípady (prázdne pole, pole rovnakých čisiel...)

Do premennej `velkost` ulozte pocet prvkov pola.

```c
#include <stdio.h>
int* najdiNajvacsie(int *arr, int velkost);
int* najdiNajmensie(int *arr, int velkost);
void vymenCisla(int *a, int *b);

int main() {
    int velkost = 5;
    int pole[velkost];
    // Nacitanie prvkov pola od uzivatela
    printf("Zadajte %d cisel:\n", velkost);
    for (int i = 0; i < velkost; i++) {
        scanf("%d", &pole[i]);
    }
    // Najdenie ukazovatelov na najvacsi a najmensi prvok
    int *najvacsie = najdiNajvacsie(pole, velkost);
    int *najmensie = najdiNajmensie(pole, velkost);
    // Vymenim najvacsie a najmensie cislo
    vymenCisla(najvacsie, najmensie);
    // Vypis pola po vymene
    return 0;
}
```

Predávanie polí

Je možné predávať aj hodnotou, ale toto je neefektívne a prakticky sa nepoužíva. Namiesto toho:

- Polia sú vždy predávané odkazom
- Ak zmeníme hodnotu jeho prvkov vo volanej funkcii, zmena sa vždy prejaví

```
1  void naMalePismena(char text[]) {
2      int i = 0; // Inicializujeme index
3      while (text[i] != '\0') { // Pokia  nie je koniec retazca
4          text[i] = tolower(text[i]);
5          i++;
6      }
7  }
8  int main() {
9      char text[13] = "Hello WORLD!"
10     printf("Povodny retezec: %s", text);
11     // Transformacia retazca na male pismena
12     naMalePismena(text);
13
14     // Vypis upraveneho retazca
15     printf("Upraveny retezec: %s", text);
16
17     return 0;
18 }
```

Viacrozmerné polia

- Polia môžu mať viac rozmerov (dimenzií)
- Sú to vlastne polia polí
- Vytvárame pomocou viacero hranatých zátvoriek

Príklad deklarovania dvojrozmerného pola:

```
1 int matica[3][3];
```

K prvkom pola pristupujeme pomocou viacero indexov a hranatými zátvorkami.
Pole môžeme inicializovať pomocou vnorených zložených zátvoriek:

```
1 int matica[3][3] = {{1,2,3}, {4,5,6}, {7,8,9}}; // Deklarujeme
      maticu 3x3
2 for (int i = 0; i < 3; i++) { // pre kazdy riadok
3     for (int j = 0; j < 3; j++) { // pre kazdy stlpec
4         printf("%d", &matica[i][j]); // vypise jeden prvok pola
5     }
6 }
```

Príklad

- Vytvorte program, ktorý načíta maticu 3x3 (dvojrozmerné pole)
- Napíšte funkciu, ktorá nájde najväčší prvok v matici a nahradí ním všetky prvky hlavnej diagonály
- Vypíšte výslednú maticu

```c
#include <stdio.h>
// Funkcia na najdenie najvacsieho prvku v matici
int najdiNajvacsie(int n, int matica[n][n]);
// Funkcia na vymenu hlavnej diagonaly za najvacsi prvok
void nahradDiagonalu(int n, int matica[n][n], int najvacsie);
// Funkcia na vypis matice
void vypisMaticu(int n, int matica[n][n]);

int main() {
    int matica[3][3]; // Deklarujeme maticu 3x3, aj s prvkami

    // Najdenie najvacsieho prvku v matici
    int najvacsie = najdiNajvacsie(3, matica);

    // Nahradenie hlavnej diagonaly najvacsim prvkom
    nahradDiagonalu(3, matica, najvacsie);

    // Vypis upravenej matice
    printf("Upravena matica:\n");
    vypisMaticu(3, matica);

    return 0;
}
```

Predávanie štruktúry hodnotou

- Funkcia dostáva kópiu celej štruktúry

```c
#include <stdio.h>
struct Bod {
    float x, y;
};
void posunBod(struct Bod b, float dx, float dy) {
    b.x += dx; // Zmenime hodnotu x v kopii
    b.y += dy; // Zmenime hodnotu y v kopii
    printf("Bod po posunuti (vo fci): (%.2f, %.2f)\n", b.x, b.y)
    ;
}
int main() {
    struct Bod b = {2.0, 3.0}; // Vytvorenie bodu
    printf("Povodny bod: (%.2f, %.2f)\n", b.x, b.y);
    // Posunieme bod
    posunBod(b, 1.0, -1.0); // Posuneme bod o (1, -1)
    // Vypiseme bod po pokuse o posun
    printf("Bod po pokuse o posunutie: (%.2f, %.2f)\n", b.x, b.y
    );
    return 0;
}
```

Predávanie štruktúry odkazom

- Funkcia dostane ukazovateľ na pôvodnú štruktúru.

```
1  #include <stdio.h>
2  struct Bod {
3      float x, y;
4  };
5  void posunBod(struct Bod *b, float dx, float dy) {
6      b->x += dx;  // Pr stup cez ukazovate  (->)
7      b->y += dy;
8      printf("Bod po posunuti (vo fci): (%.2f, %.2f)\n", b->x, b->
       y);
9  }
10 int main() {
11     struct Bod b = {2.0, 3.0}; // Vytvorenie bodu
12     printf("Povodny bod: (%.2f, %.2f)\n", b.x, b.y);
13
14     // Posunieme bod
15     posunBod(&b, 1.0, -1.0); // Pred vame adresu bodu pomocou &
16
17     // Vypiseme bod po posunuti
18     printf("Bod po posunut : (%.2f, %.2f)\n", b.x, b.y);
19     return 0;
20 }
```

Úlohy na precvičenie:

Úloha: Definujte štruktúru Bod (x, y). Vytvorte pole bodov, o aspoň 5 prvkov.

- Implementujte funkciu, ktorá vypočíta ich geometrický stred.

  ```
  void geometrickyStred(int n, bod array[], bod* S)
  ```

- Implementujte funkciu, ktorá vypíše Bod, ktorý je najďalej od počiatočného bodu X (0,0)

  ```
  void najdalejBod(int n, bod array[])
  ```

- Implementujte funkciu, ktorá nájde najbližší bod k vypočítanenému geometrickému stredu.

  ```
  bod* najblizsiBod(int n, bod array[], bod S)
  ```

- Implementujte funkciu, ktorá vypočíta priemernú vzdialenosť bodov. Teda, pre každú dvojicu bodov v poli spočíta ich vzdialenosť a vypočíta priemer.

  ```
  float priemernaVzdialenost(int n, bod array[])
  ```

Rada: Najskôr si implementujte funkciu, ktorá pre dva body spočíta ich vzdialenosť.