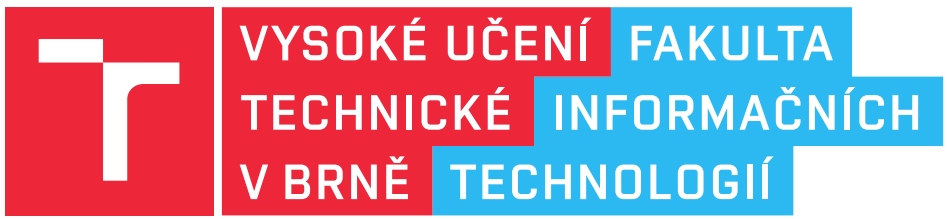
Základy programovania
Cvičenie 7

Práca s ukazovateľmi

Cvičiaci: Ing. Magdaléna Ondrušková, (iondruskova)



5. listopadu 2024

Ukazatele a polia

- V jazyku C sa názov pola (premenná) správa ako ukazovateľ na prvý prvok poľa

```c
#include <stdio.h>
int main() {
  int x[5] = {1, 2, 3, 4, 5};
  int* ptr;
  int* ptr2;

  // ptr is assigned the address of the third element
  ptr = &x[2];
  ptr2 = x;

  printf("*ptr = %d \n", *ptr);   // 3
  printf("*(ptr+1) = %d \n", *(ptr+1)); // 4
  printf("*(ptr-1) = %d", *(ptr-1));  // 2

  printf("*ptr2 = %d \n", *ptr2); // 1

  // Priority of operators:
  printf("*ptr2 = %d \n", *ptr2+6); // 7
  printf("*ptr2 = %d \n", *(ptr2+6)); // 1
  return 0;
}
```

Predávanie poľa do funkcie

```c
void spracujPole(int arr[]) {
    int velkost = sizeof(arr) / sizeof(arr[0]);
    printf("Velkost pola je: %d\n", velkost);

    for (int i = 0; i <= velkost; i++) {  3
        arr[i] = i * 2;
    }

    for (int i = 0; i < velkost; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int mojePole[] = {1, 2, 3, 4, 5};
    spracujPole(mojePole);

    printf("Pole po uprave:\n");
    for (int i = 0; i < 5; i++) {
        printf("%d ", mojePole[i]);
    }
}
```

Predávanie poľa do funkcie

- Predávame vlastne ukazovateľ na prvý prvok poľa

```
1 void vypisPole(int *arr, int size) {
2     for (int i = 0; i < size; i++) {
3         printf("%d ", arr[i]);
4     }
5     printf("\n");
6 }
```

String ako pole znakov

- Dátový typ string je definovaný ako pole znakov
- Pole je vlastne ukazovateľ na prvý znak

Dve formy zápisu:

- `char *str` - deklaruje ukazovateľ na konštantný reťazec
- `char str[]` - pole reťazcov, ktoré je možné modifikovať

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main() {
5      char *str1 = "Hello world.";
6      char str2[] = "Hello world.";
7      if (strcmp(str1, str2) == 0) {
8          printf("Retazce su rovnake\n");
9      } else {
10         printf("Retazce sa lisia\n");
11     }
12
13     printf("Velkost *str1: %d\n",sizeof(str1));  // 8
14     printf("Velkost str2[]: %d\n",sizeof(str2)); // 13
15     return 0;
16 }
```

Príklad 1

- Načítajte reťazec od užívateľa, max 20 znakov.
- Implementujte funkciu, ktorá zmení všetky písmená v reťazci na veľké

```
void transform_to_upper(char* str, int size)
```

- Implementujte funkciu, ktorá zašifruje reťazec Ceasorovou šifrou

```
void ceaser_cipher(char *str, int size,  int shift)
```

Príklad 2

- Vytvorte pole o 6 prvkov.
- Ak užívateľ zadá argument programu *multiply*, vynásobte každý prvok pola číslo 10.

```
void multiply_array(int *arr, int size)
```

- Ak užívateľ zadá argument programu *reverse*, otočíte pole na mieste (bez pomocného pola).

```
void reverse_arr(int *arr, int size)
```

Dynamická alokácia pamäte

- Funkcia rezervuje blok v pamäti pre špecifikovaný počet bytov.
- Umožňuje pridelovať pamäť počas behu programu, nie len pri kompilácii

Funkcia malloc

- Vyžaduje počet bajtov, ktoré má prideliť
- Vráti **ukazovateľ** na začiatok novoprideleného pamäťového bloku
- Ak alokácia zlyhá (napr. pre nedostatok pamäte) vráti `NULL`

```
1 void* malloc(size_t size);
```

`void*` znamená ukazovateľ na ľubovoľný typ. Je treba pretypovať na želaný dátový typ (napr. `int*`).

Obecný zápis **malloc:**

```
ptr = (castType*) malloc(size);
```

Príklad použitia **malloc:**

- Zarezervuje v pamäti 400 byte
- Veľkosť `float` je 4 byte.
- `ptr` obsahuje adresu prvého byte v alokovanej pamäti
- Výraz by vrátil `NULL` ak by sa alokácia nepodarila.

```
ptr = (float*) malloc(100 * sizeof(float));
```

Každú alokovanú pamäť je potreba uvoľniť
- pomocou funkcie **free()**

```
free(ptr);
```

Miesto v pamati kam ukazuje `ptr` sa uvoľní.

Príklad: Vypočítajte súčet N čísiel. N zadá užívateľ.

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
  int n, i, *ptr, sum = 0;

  printf("Enter number of elements: ");
  scanf("%d", &n);

  // doplnte alokaciu pamate pre zadany pocet prvkov
  // overte si uspesnu alokaciu pamate

  printf("Enter elements: ");
  // postupne nacitajte a scitajte jednotlive prvky

  printf("Sum = %d", sum);

  // uvolnite alokovanu pamat
  return 0;
}
```

Príklad: Vypočítajte súčet N čísiel. N zadá užívateľ.

```c
int main() {
  int n, i, *ptr, sum = 0;
  printf("Enter number of elements: ");
  scanf("%d", &n);
  ptr = (int*) malloc(n * sizeof(int));
  // if memory cannot be allocated
  if(ptr == NULL) {
    printf("Error! memory not allocated.");
    exit(0);
  }
  printf("Enter elements: ");
  for(i = 0; i < n; ++i) {
    scanf("%d", ptr + i);
    sum += *(ptr + i);
  }
  printf("Sum = %d", sum);
  // deallocating the memory
  free(ptr);
  return 0;
}
```

Vektor - uvažujte matematický vektor reprezentovaný pomocou jednorozmerného pola. Implementujte dátový typ pre vektor a jeho veľkosť. Implementujte nasledujúce funkcie:

- Konštrukcia vektora - alokovanie miesta pre vektor danej veľkosti: `int vector_const(vector_t *v, unsigned int size)`
- Inicializácia vektora predom definovanými hodnotami (0,1,2,...): `void vector_init(vector_t *v)`
- Dealokácia vektora z pamäti:

  `void vector_desctruct(vector_t *v)`

Následne implementujte funkcie nad vektorom:

- Veľkosť vektora $\|\vec{v}\| = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}$

  `double vector_size(const vector_t *v)`
- Skalárny súčin dvoch vektorov. Oba musia mať rovnakú veľkosť.

  `void scalar_vector(const vector_t *v1,const vector_t *v2, int *result)`