

Sedmé počítačové cvičení

Základy programování (IZP)

Brno University of Technology, Faculty of Information Technology
Božetěchova 1/2, 612 66 Brno - Královo Pole
Petr Veigend, veigend@fit.vut.cz



- Z <https://www.fit.vut.cz/person/veigend/teaching/#nav> si stáhněte zip s kostrami k tomuto cvičení

- **Proměnná**
- **Ukazatel (pointer)**
- **Adresa proměnné**
- **Hodnota z adresy**
- **NULL**

- **Proměnná** – pojmenované místo v paměti, ve kterém uchováváme data
- **Ukazatel (pointer)** – proměnná, která uchovává adresu nějakého místa v paměti
 - Říkáme, že ukazatel ukazuje na místo, které je určeno touto adresou
- **Adresa proměnné** – referenční operátor **&**
- **Hodnota z adresy** – dereferenční operátor *****
- **NULL** – používá se pro inicializaci ukazatelů – říká, že ukazatel nikam neukazuje

```
int main() {
    int arr[5] = { 1, 2, 3, 4, 5 };
    int *ptr = arr;
    printf("%p\n%p\n", arr, ptr);
    *arr = 42;
    printf("%d\n", arr[0]);
    *ptr = 0;
    printf("%d\n", arr[0]);
    printf("%d\n", ptr[0]);
    printf("%p\n%p\n%p\n", arr, &arr[0], ptr);
}
```

Soubor: 02-arrays.c

PŘEDÁVÁNÍ POLÍ DO FUNKCÍ

- Pole do funkce předáme takto

```
void foo(int arr[], int size);  
void bar(int *arr, int size);
```

- Zápisy jsou ekvivalentní (`arr[]` je jasněji pole)
- Co tedy vlastně předáváme?

- Napište funkci s definicí

```
void arrMultConst(int arr[], int size, int c);
```

která vynásobí všechny prvky pole `arr` konstantou `c`

- Ve funkci `main` pole vytvoříme tak, jak jsme zvyklí

```
int pole[5] = {1,2,3,4,5};
```

- Napište funkci s definicí

```
void changeCase(char str[]);
```

Která všechny malá písmena v řetězci `str` změní na velká.

- Ve funkci `main` řetězec vytvoříme tak, jak jsme zvyklí

```
char retezec[] = "Hello";
```

- Napište funkci s definicí

```
int findSubstr(char str[], char substr[]);
```

Která vrátí pozici začátku podřetězce **substr** v řetězci **str**. Pokud se v řetězci **str** podřetězec **substr** nenachází, funkce vrátí -1.

- Ve funkci **main** řetězec (a podřetězec) vytvoříme tak, jak jsme zvyklí

```
char retezec[] = "Hello";
```

- Napište funkci s definicí

```
void insert(int arr[], int size, int item, int pos);
```

Která vloží prvek **item** na pozici **pos** v poli **arr**. Zbytek prvků v poli se posune, poslední prvek bude z pole odstraněn.

- *Doporučení: pole procházejte od posledního prvku*

Soubor: 05-alloc.c

ALOKACE PAMĚTI

- Tento příklad budeme řešit v online prostředí <https://hub.bazar.nesad.fit.vutbr.cz/hub/home>
 - Empty Environment
- Kvůli nástroji valgrind (viz dále)

- Při práci s dynamickou alokací paměti doporučuji používat nástroj **valgrind**.

```
valgrind ./<nazev programu>
```

- Umožňuje testovat, zda správně pracujete s dynamickou pamětí.
- Druhý projekt **bude nutné** testovat na Merlinovi, případně lokálním Linuxu (WSL)
 - Prezentaci k programování vzdáleně najdete na profilu.

- Alokujte paměť pro uložení řetězce (zvolte si jeho maximální velikost)

```
char* first = malloc(<velikost alokované paměti>);  
if (first == NULL) // chyba
```

- Do této paměti uložte řetězec (**scanf**)
- Zkopírujte řetězec **first** do jiného řetězce
 - Opět pro něj alokujte paměť
- Vytiskněte oba řetězce a uvolněte je z paměti

```
free(first);
```

Soubor: 04-vectors.c

VEKTORY

- Implementujte funkce pro práci s vektory

- Konstruktor, destruktork

```
int vector_ctor(vector_t *v, unsigned int size);  
void vector_dtor(vector_t *v);
```

- Naplnění hodnotami

```
void vector_init(vector_t *v);
```

- Tisk vektoru

```
void vector_print(vector_t *v);
```

- A dále funkce pro operaci s vektory

- Násobení vektoru skalárem (konstantou)

```
void vector_mult(vector_t *v, int c);
```

- Součet dvou vektorů ($v1 = v1 + v2$)

```
void vector_add(vector_t *v1, vector_t *v2);
```

Soubor: 03-args.c

ZPRACOVÁNÍ ARGUMENTŮ PŘÍKAZOVÉ ŘÁDKY

- Jednotlivé argumenty budeme oddělovat mezerou
- Argumenty, se kterými byl program spuštěn, se dají získat pomocí doplnění dvou parametrů do hlavičky funkce **main**:

```
int main(int argc, char* argv[])
{
    // argc - počet argumentů
    // argv - jednotlivé argumenty, argv[0]
    // (název souboru s programem)
}
```

- Argumenty jsou opět **pole**

```
int main(int argc, char* argv[])
{
    // argc - počet argumentů
    // argv - jednotlivé argumenty, argv[0]
    // (název souboru s programem)
}
```

- Pro `./hello -sum 10 20` argc=4

argv[0]	argv[1]	argv[2]	argv[3]
"hello"	"-sum"	"10"	"20"

- Vytvořte datový typ, který bude uchovávat konfiguraci programu
- Vytvořte funkci, která bude na základě argumentů programu nastavovat jeho konfiguraci

```
const char *usage = "syntax: [-x|-y] [-n COUNT] -s STR\n"-x and -y are optional and mutually exclusive\n"-s STR - mandatory, STR is a string\n"-n COUNT - optional, COUNT is non-negative integer\n(default: 10)\n";
```

Děkuji Vám za pozornost!