Základy programování (IZP)

Sedmé počítačové cvičení

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

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

8. týden

Organizační okénko - připomenutí

- Projekt č. 2
  - Odevzdání: 5.12.
  - týmy 3 – 5 členů
  - Vytvořena skupina na **MS Teams** pro domlouvání týmů, kdo nemá ☺
    - Kdo se neumí přihlásit? Je potřeba použít vás vut login: **xlogin00@vutbr.cz**
    - **Přihlášení přes prohlížeč nebo aplikaci**
- Příští týden není cvičení, bude **půlsemestrálka**

Projekt č. 1 - problémy

- Globální proměnné
- If (argv[4]){…}
  - Segmentation fault
- Nepřidávat žádné printf navíc
- Magické konstanty – pro co se hodí a pro co ne?
- KONSTANTA_MA_PODTRZITKA
- Dekompozice na funkce
- Komentáře
- Názvy funkcí a proměnných
  - Nezačínat velkým písmenem Funkce1()
- Zanoření funkcí by nemělo přesáhnout 4. úroveň
- Zakázané knihovny (math, unistd)
- Incializace řetězce
  - s[10] = {'-', '-', 's', 't', 'a', 't', 's', '\0'} vs s[] = “--stats”

Náplň cvičení

- ovládat práci s **ukazateli**
- porozumět souvislosti mezi **polem a ukazatelem**
- porozumět způsobu předávání polí a řetězců do funkcí
- ovládat vytváření funkcí modifikujících pole a řetězce
- porozumět dynamické alokaci paměti (volání **malloc** a **free**)

PRÁCE S UKAZATELI

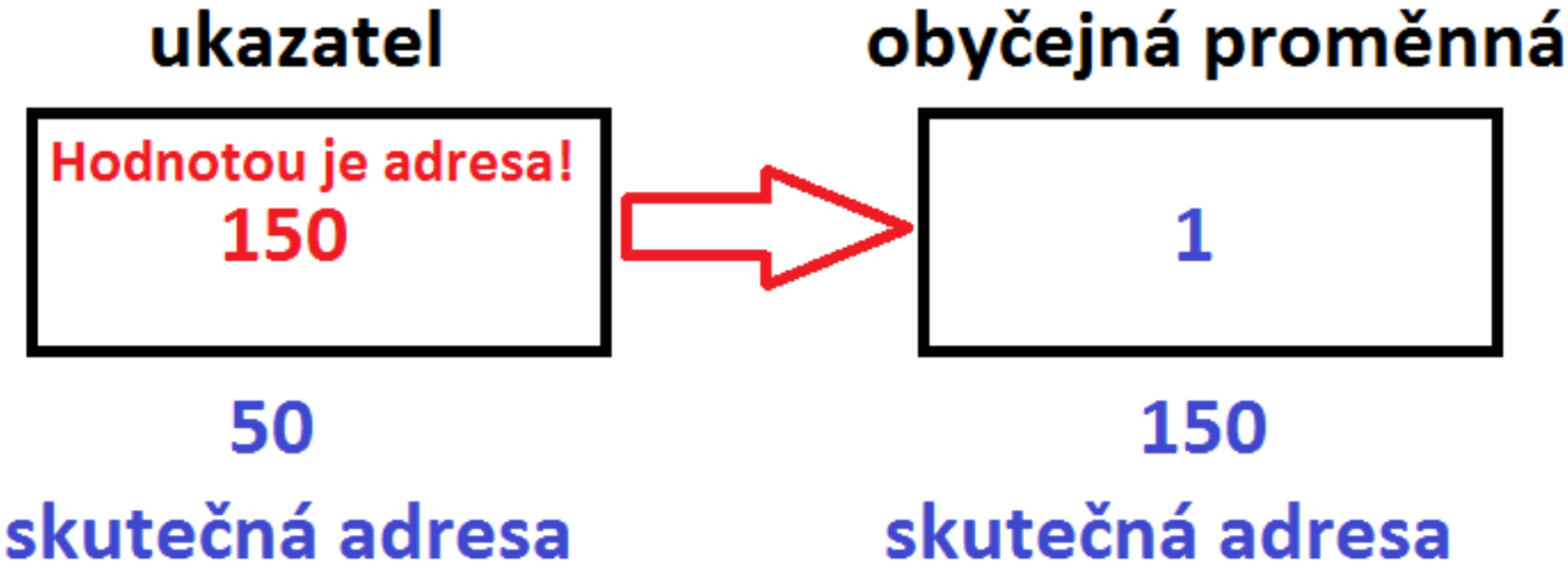
Ukazatel, reference, dereference

- **Ukazatel (pointer)**
- **Velikost ukazatele**
- **Jak získám adresu proměnné?**
- **Jak získám hodnotu z adresy?**

Ukazatel, reference, dereference

- **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
- **Velikost ukazatele** – závisí na tom, kolikabitový máme procesor/překladač (16, 32, 64 bitů)
- **Adresa proměnné** – referenční operátor **&**
- **Hodnota z adresy** – dereferenční operátor *****
- **NULL** – používá se pro inicializaci ukazatelů – říká, že ukazatel nikam neukazuje

Ukazatel, reference, dereference

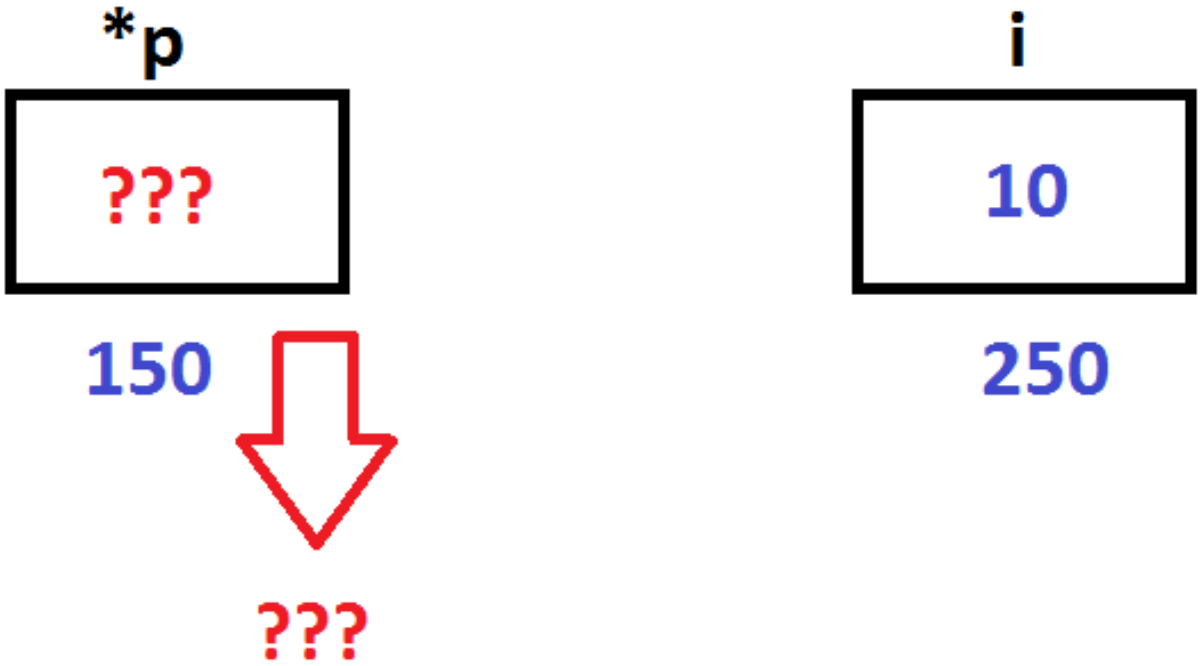


Ukazatel, reference, dereference

```
int i = 10;
int *p;  //?
p = &i;  //?
*p = 20; //?
printf("Hodnota promenne i: %d\n", i); //?
```

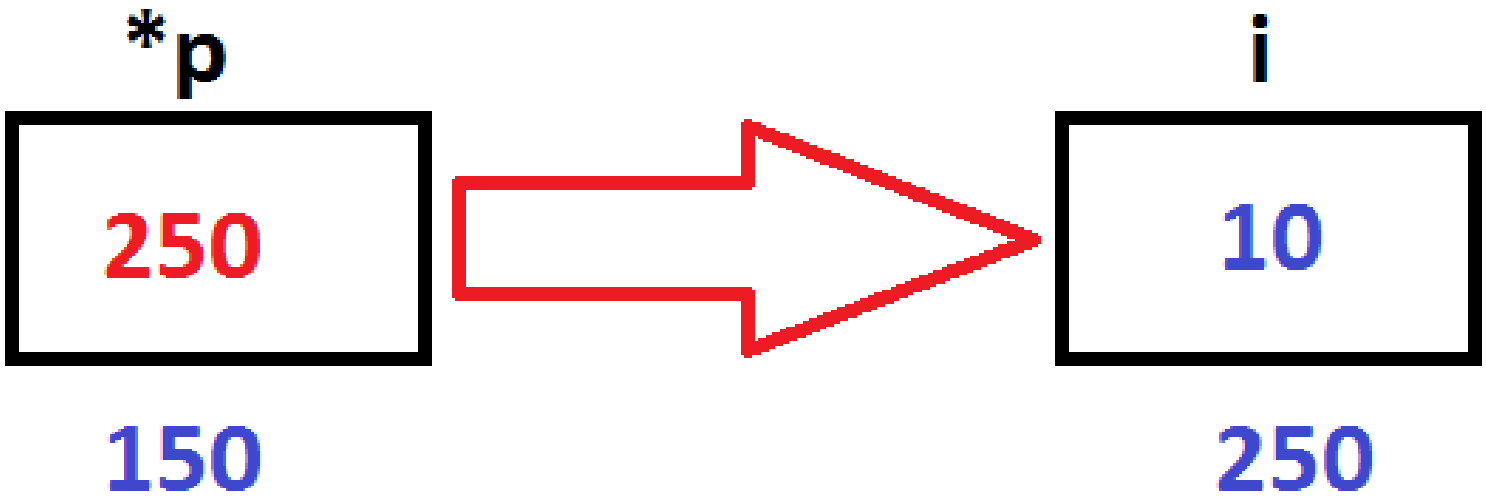
Ukazatel, reference, dereference

```
int i = 10;
int *p; // ukazatel p není inicializován!
p = &i;
*p = 20;
printf("Hodnota promenne i: %d\n", i);
```



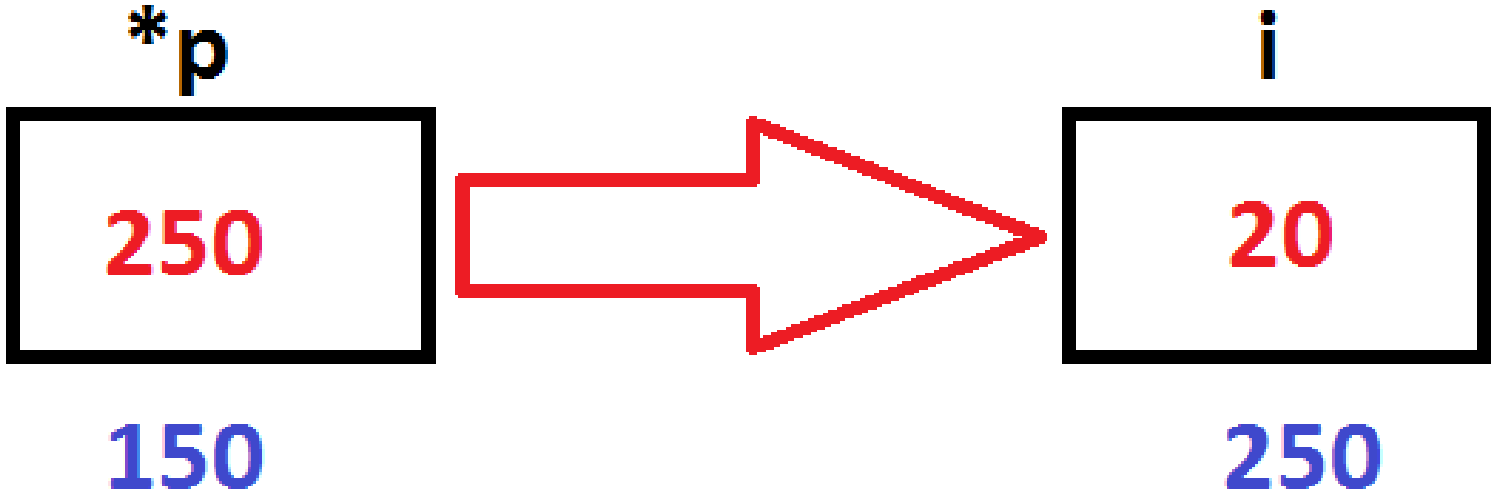
Ukazatel, reference, dereference

```
int i = 10;
int *p; // ukazatel p není inicializován!
p = &i; // ukazatel p ukazuje na i
*p = 20;
printf("Hodnota promenne i: %d\n", i);
```



Ukazatel, reference, dereference

```
int i = 10;
int *p; // ukazatel p není inicializován!
p = &i; // ukazatel p ukazuje na i
*p = 20; // pomocí p jsme změnili hodnotu i
printf("Hodnota promenne i: %d, hodnota
  proměnne, kam ukazuje pa: %d\n", i, *p);
```



Pole jako ukazatel

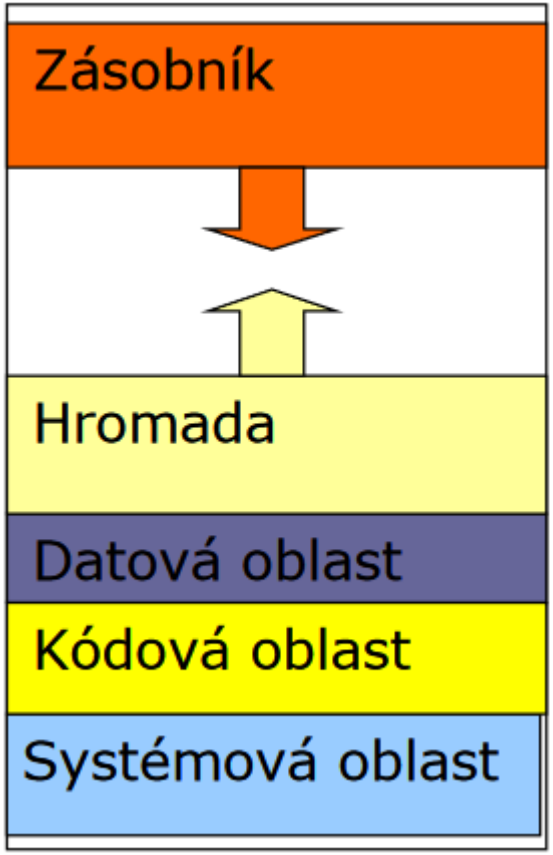
- Jaký je rozdíl mezi dynamickou a statickou alokací paměti?
- Jak se provede kopie jednoho pole do druhého?
- Jaký je vzájemný vztah pole a ukazatelů?

```
int x[10];
int *p_x = (int *) malloc(10 * sizeof(int))
```

- Lze předat pole do funkce hodnotou?
- Hlídá C meze polí?
- Naalokujte pomocí malloc pole o 5 prvcích (celá čísla)

Pole jako ukazatel

- Jaký je rozdíl mezi dynamickou a statickou alokací paměti?
  - **Statická** – nutnost znát velikost dat při překladu, data alokována v **datové** oblasti
    - Globální proměnné pouze staticky
  - **Dynamická** – velikost dat není nutné znát již při překladu, používá se zásobník a hromada (heap)
    - **Zásobník** – použit pro většinu lokálních proměnných (pozor, lokální proměnná je definována staticky, ale v paměti vzniká dynamicky), děje se automaticky
    - **Hromada** – pomocí malloc a free



Pole jako ukazatel

- Jak se provede kopie jednoho pole do druhého? Nelze najednou!
- Jaký je vzájemný vztah pole a ukazatelů?
  - **x** a **p_x** jsou poinery na pole o 10 prvků typu int
  - sizeof(x) != sizeof(p_x)
- Hlídá C meze polí? NE
- Naalokujte pomocí malloc pole o 5 prvcích (celá čísla)
  - int *a = (int *) malloc(sizeof(int) * 5)

- Kdo si nebyl jistý s odpověďmi, ať mrkne na oporu IZP
  - /soubory/Studijní opory/

Pole jako ukazatel

```
int main()
{
  int arr[] = { 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);
}
```

Předání parametrů funkcím

- Parametry můžeme funkcím předávat
  - **Hodnotou** (vytvoříme lokální kopii proměnné)
    - Lokální proměnná se vytvoří na **zásobníku**
  - **Odkazem** (do funkce předáváme pouze adresu proměnné v paměti)

- Jak předávat pole do funkce?
  - při předávání pole do funkce se předává *jenom ukazatel*, samotné pole zůstává tam, kde je (např. na zásobníku)

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

Příklady na předávání polí do funkcí

- Implementujte funkci, která najde pozici začátku (=první pismeno) podřetězce. Má dva parametry.

```
int find_substring_start(…){
  // funkce vraci -1, když nenalezne zacatek
  podretezce, jinak pozici (=index) zacatku
  subretezce
}
```

- **Pro rychlejší:** funkce vloží prvek do pole na zadaném indexu (zbytek prvků v poli posune, poslední prvek bude z pole odstraněn)

Argumenty programu

- Jednotlivé argumenty budeme oddělovat mezerou
- Argumenty se dají získat pomocí následující konstrukce:

```
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"

Úkol

- Vytvořte **datový typ** pro konfiguraci programu, tj. pro uchování informací o argumentech příkazové řádky, který musí být spouštěn následujícím způsobem:

```
./student NAME [-isHappy] [-y BIRTH]
```

- NAME - jméno studenta
- -isHappy - příznak, jestli je student šťastný
- y BIRTH – rok narození

Parse args

- Vytvořte funkci, která na základě argumentů programu nastaví jeho konfiguraci.

```
./student NAME [-isHappy] [-y BIRTH]
```

```
int parse_args(int argc, char *argv[])
```

1. Inicializace struktury (defaultní nastavení hodnot)
2. Kontrola počtu argumentů
   - Jaký je minimální a maximální počet argumentů?
3. Projít všechny argumenty (for, **strcmp**) a uložit do struktury

Operace nad vektory

- Uvažujme reprezentaci matematického vektoru pomocí jednorozměrného pole. Implementujte datový typ, který zapouzdří vektor a jeho velikost.

```
typedef struct {
   int *items;
   int size;
} Vector
```

- Implementujme následující funkce
  a) konstrukce, tj. alokace vektoru
  b) inicializace vektoru předem definovanými hodnotami
  c) destrukce, tj. dealokace vektoru z paměti

Úkol

- Vytvořte vektor v2, inicializujte ho hodnotami stejně jako v1
- Sečtěte vektory v1 a v2 a výsledek bude uložen ve vektoru v3
- Nezpomeňte vše uvolnit, co je potřeba

```
int vector_add(Vector *v1, Vector *v2, Vector *v3){
 // pozor, vektory musí mít stejnou velikost, pokud
   nemají, vrátíme 1
}
```

Děkuji Vám za pozornost!