Základy programování (IZP)

Šesté počítačové cvičení

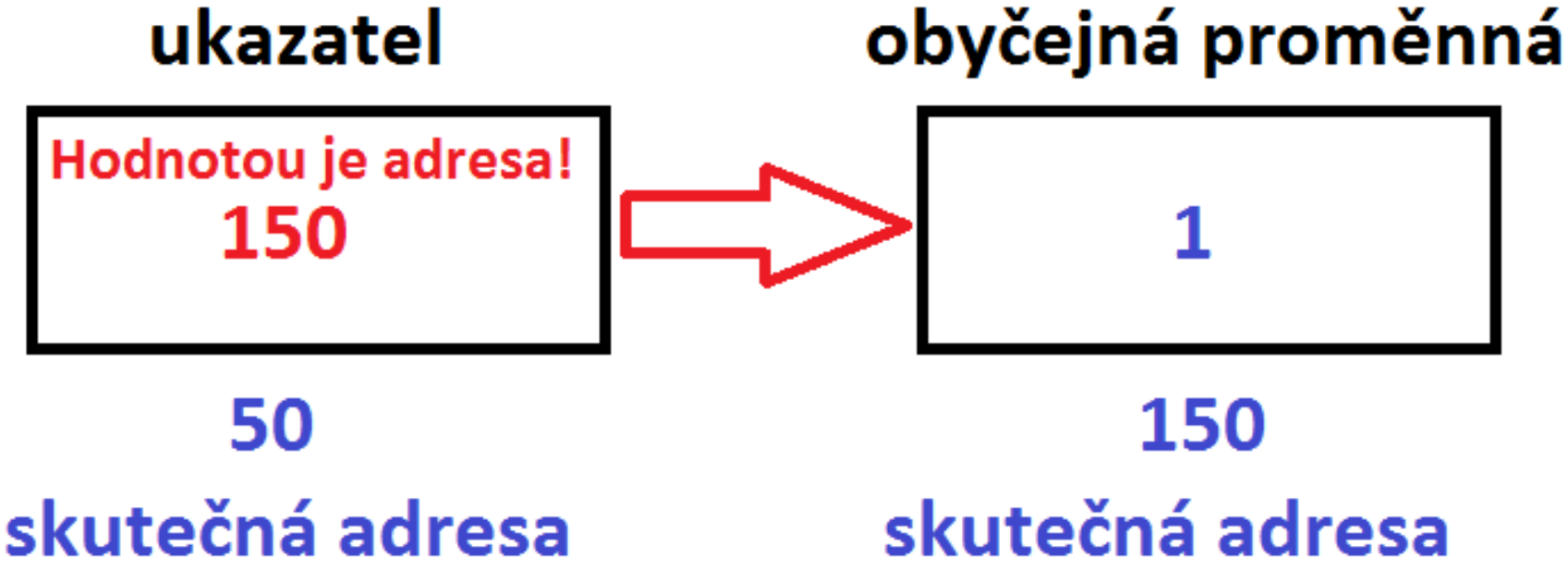
Brno University of Technology, Faculty of Information Technology
Božetěchova 1/2, 612 66 Brno - Královo Pole
Petr Veigend, veigend@fit.vut.cz

UKAZATELE

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

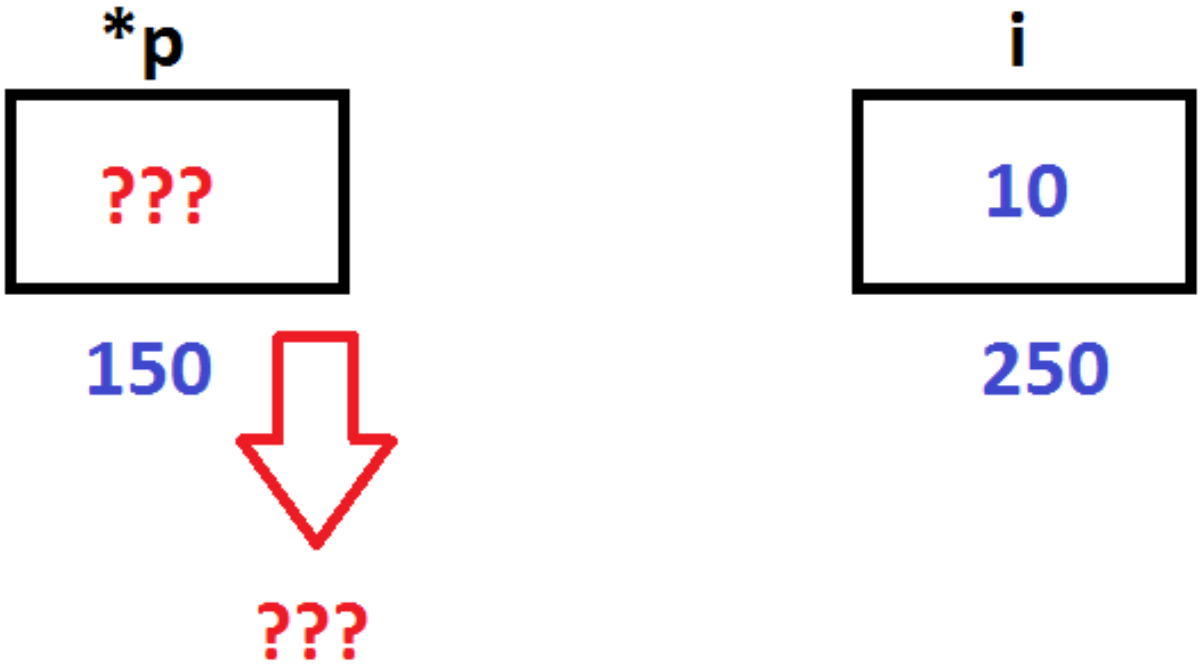


Ukazatel, reference, dereference

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

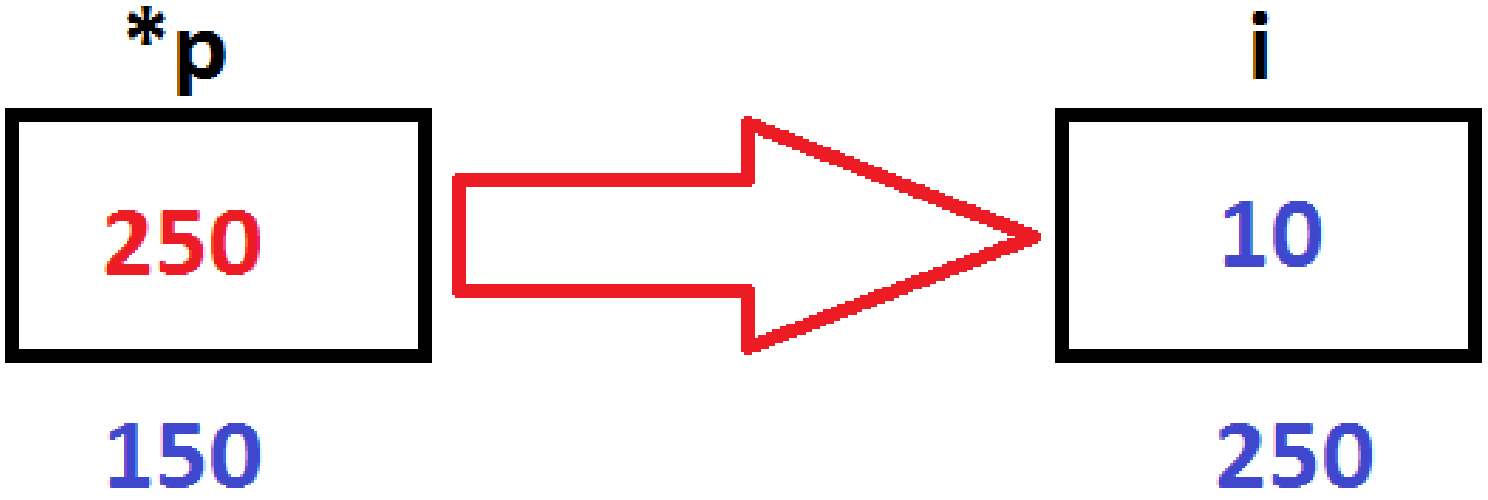
Ukazatel, reference, dereference

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

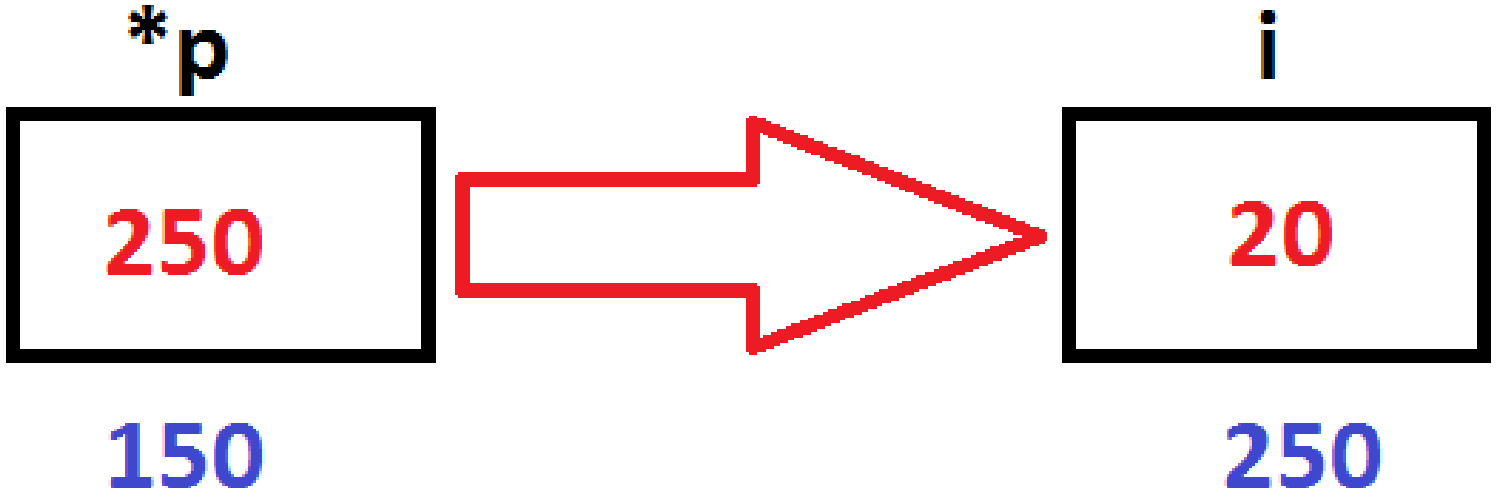


Ukazatel, reference, dereference

```
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);
```



```
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\n", i);
```



Ukazatel, reference, dereference

```
int a = 0, b = 42;
int* p;    // p -->
p = &b;    // p -->
p = &a;    // p -->
(*p) ++;   // p -->
*p ++;     // p -->
```

Ukazatel, reference, dereference

```
int a = 0, b = 42;
int* p;    // p --> nedefinováno
p = &b;    // p -->
p = &a;    // p -->
(*p) ++;   // p -->
*p ++;     // p -->
```

Ukazatel, reference, dereference

```
int a = 0, b = 42;
int* p;    // p --> nedefinováno
p = &b;    // p --> 42 = b
p = &a;    // p -->
(*p) ++;   // p -->
*p ++;     // p -->
```

Ukazatel, reference, dereference

```
int a = 0, b = 42;
int* p;    // p --> nedefinováno
p = &b;    // p --> 42 = b
p = &a;    // p --> 0 = a
(*p) ++;   // p -->
*p ++;     // p -->
```

Ukazatel, reference, dereference

```
int a = 0, b = 42;
int* p;    // p --> nedefinováno
p = &b;    // p --> 42 = b
p = &a;    // p --> 0 = a
(*p) ++;   // p --> 1 (operace přičtení 1)
*p ++;     // p -->
```

Ukazatel, reference, dereference

```
int a = 0, b = 42;
int* p;    // p --> nedefinováno
p = &b;    // p --> 42 = b
p = &a;    // p --> 0 = a
(*p) ++;   // p --> 1 (operace přičtení 1)
*p ++;     // p --> neznámý výsledek
           // k adrese a se přičte
           // 1 ne sizeof(int)
```

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)

- Předání hodnotou by mělo být bez problémů
- Jak funguje předání odkazem? Následuje příklad … ☺

- Abychom si ukázali, jak funguje **předávání odkazem**, definujme následující funkci

```
void inc(int* n) {
  *n = *n + 1; // proč?
}
```

- Vidíme, že funkce nic nevrací.
  - Hodnotu ale můžeme z funkce získat přes ukazatel
- Jak ji tedy zavoláme a použijeme v programu?

Předání parametrů funkcím – předávání odkazem

```
void inc(int* n) {
  *n = *n + 1;
}
```

```
int main()
{ …
  int a = 5;
  inc(&a); // proč?
  printf("hodnota a: %d", a); // ??
}
```

Funkce plus

- Napište dvě varianty funkce **`plus`**

```
// v = a + b
void plus_p(int a, int b, int* v);

int plus(int a, int b);
```

- Napište funkci deklarovanou jako

```
int del(double citatel, double jmenovatel,
   double *vysledek)
```

- **Návratová hodnota**: kód chyby (1 při chybě, 0 jinak)
- **citatel:** čitatel
- **jmenovatel:** jmenovatel
- **kvocient:** výsledek operace dělení

$$\boldsymbol{kvocient} = \frac{\boldsymbol{citatel}}{\boldsymbol{jmenovatel}}$$

Funkce `swap`

- Napište funkci, která prohodí hodnoty dvou proměnných typu **`int`**
- Proměnné do funkce předejte **odkazem**

```
void swap(int* a, int* b);
```

Záměna dvou prvků v poli

- Napište funkci, která zamění (prohodí) dva prvky v poli typu **`int`**
- Použijte funkci **`swap`**

RELACE

```
struct coordinate_t {
    int x;
    int y;
};
//typedef struct coordinate_t TCoordinate;
```

- Operátor . nebo ->
  - **-> (šipka)** pokud předáváme strukturu (složený datový typ) jako ukazatel

```
void setCoord(struct coordinate_t* c)
{
   c->x = 1;
   c->y = 2; }
```

  - **. (tečka)** jindy

```
struct coordinate_t setCoord(struct
   coordinate_t c)
{
   c.x = 2;
   return p; }
```

Inicializace prvků

```
int main() {
  struct coordinate_t c[2] = { {1,3},
  {2,4}};
  // nebo
  c[0].x = 1; c[0].y = 3;
  c[1].x = 2; c[2].y = 4; }
```

Relace na množině

- Na profilu najdete kostru, stáhněte si ji, budeme ji postupně doplňovat přibližně v tomto pořadí:
  - Datový typ pro dvojici prvků
  - Ověření, zda je zadaná relace funkce
    - K jednomu x může být max. jedno y
  - Prohození prvního a druhého prvku v relaci
    - [1,2] –> [2,1]

Děkuji Vám za pozornost!