Základy programování (IZP)

Šesté 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

7. týden

Organizační okénko

- Projekt č. 1
  - S čím jste měli největší problém?
  - Odevzdalo 37/40 studentů
  - Hodnocení do 14 dní
- Projekt č. 2
  - Týmy 3 – 5 členů
  - Zadání již na **wiki**

Projekt č. 1 – Takto NE!

1. Překladač počet prvků ve funkci ignoruje, takto se délka nepředává!

```
int pocet_znaku(char slovo[102]);
```

2. Hvězdné soustavy nejsou potřeba

```
/**
 * ****************************
 * *
         *
 * *        Alena Tesařová          *
 * *              xtesar36                          *
                                    *
 * ****************************
 * */
```

3. Počet parametrů musí odpovídat i v komentářích

```
/**
 * Funkce overi, ze se jedna o cislo
 * @param slovo je typu char a chceme zjistit, jestli se jedna o
 * @param cislo
 * @return Vraci 1 jestli se jedna o cislo, 0 jestli ne
 */
int je_to_cislo(char slovo[])
```

4. Pozor na magické konstanty!!!

```
 if ((heslo[pozice_znaku]>=32 && heslo[pozice_znaku] <= 47 )||
   (heslo[pozice_znaku]>=58 && heslo[pozice_znaku] <= 64) ||
   (heslo[pozice_znaku]>=123 && heslo[pozice_znaku] <= 126)){
                    sznak++;
}
```

5. Při použití doxygen komentářů jsou potřeba dvě **

```
/*
 * Funkce overi, ze se jedna o cislo
 * @param slovo pole znaků
 * @return Vraci 1 jestli se jedna o cislo, 0 jestli ne
 */
int je_to_cislo(char slovo[])
```

6. Prosím **NEpoužívat** diakritiku

```
char passwords[80];
char uniquesymbols[500]; //jedineÄnÄŠ symboly
int elements_counter = 0; //spoÄÄtat poÄet hesel
```

5. Takto to stačit nebude…

```
/**
 * Function to compare two strings.
 * @param x
 * @param y
 * @return
 */
bool stringsCompare(char *x, char *y);
```

6. Struktury nic nevrací, @nasleduje_klicove_slovo

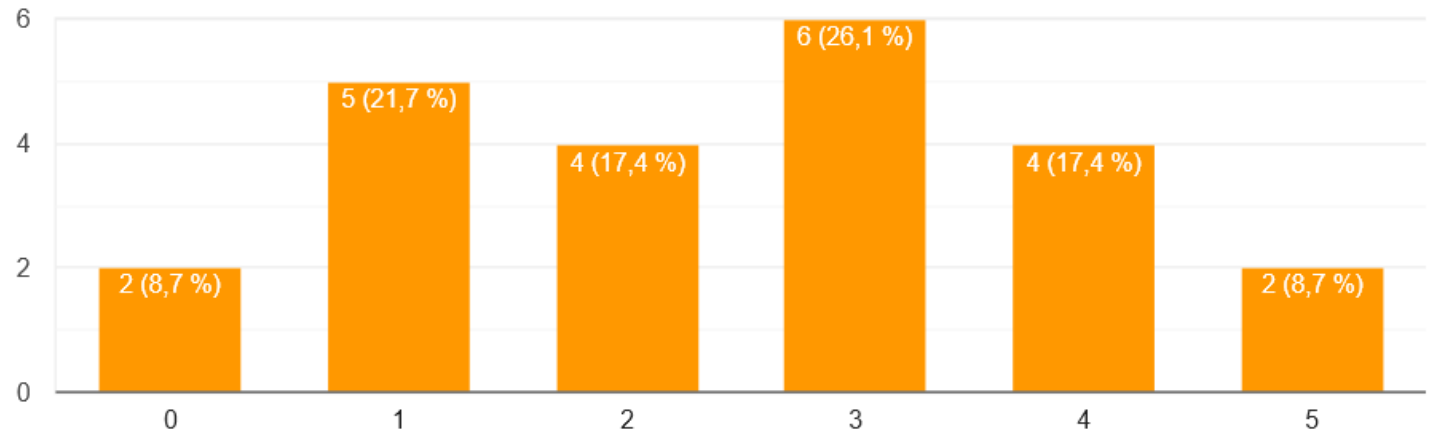
- zde stačí komentář přímo na řádek anebo se můžeme obejít i bez něco

```
/**
* structure of password
* @smallC if password has char of small alphabet
* @bigC if password has char of small alphabet
* @numberC if password has number
* @specC if password has special char
* @level password's level
* @return 1 if it is true 0 if it is false
**/
typedef struct
{
  int smallC;
  int bigC;
  int numberC;
  int specC;
  int level;
} Password;
```

Děkuji za vyplnění dotazníku !

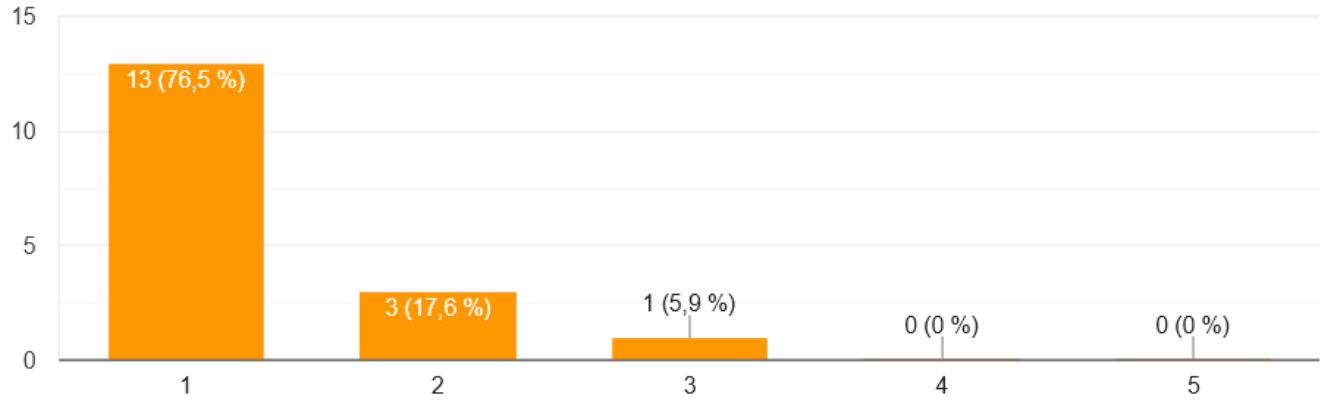
Jaké jsou Vaše zkušenosti s programováním (libovolný jazyk)? Kolik let už programujete?

23 odpovědí



Jaké byly vaše zkušnosti z jazyku C před kurzem IZP?

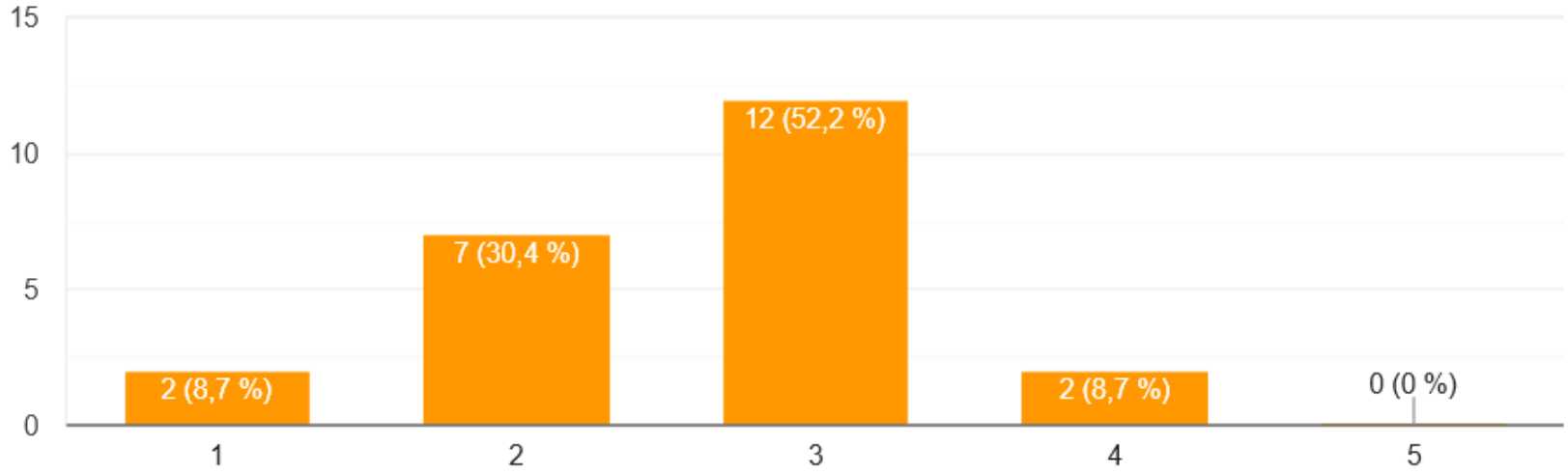
17 odpovědí



Děkuji za vyplnění dotazníku !

Tempo cvičení

23 odpovědí



Náplň cvičení

- ovládat datové struktury (typu záznam, struct),
- **porozumět přímým ukazatelům** na data, vysvětlit rozdíl mezi referenčním a dereferenčním operátorem (&, *),
- ovládat práci s předáním parametrů odkazem.

PRÁCE S UKAZATELI

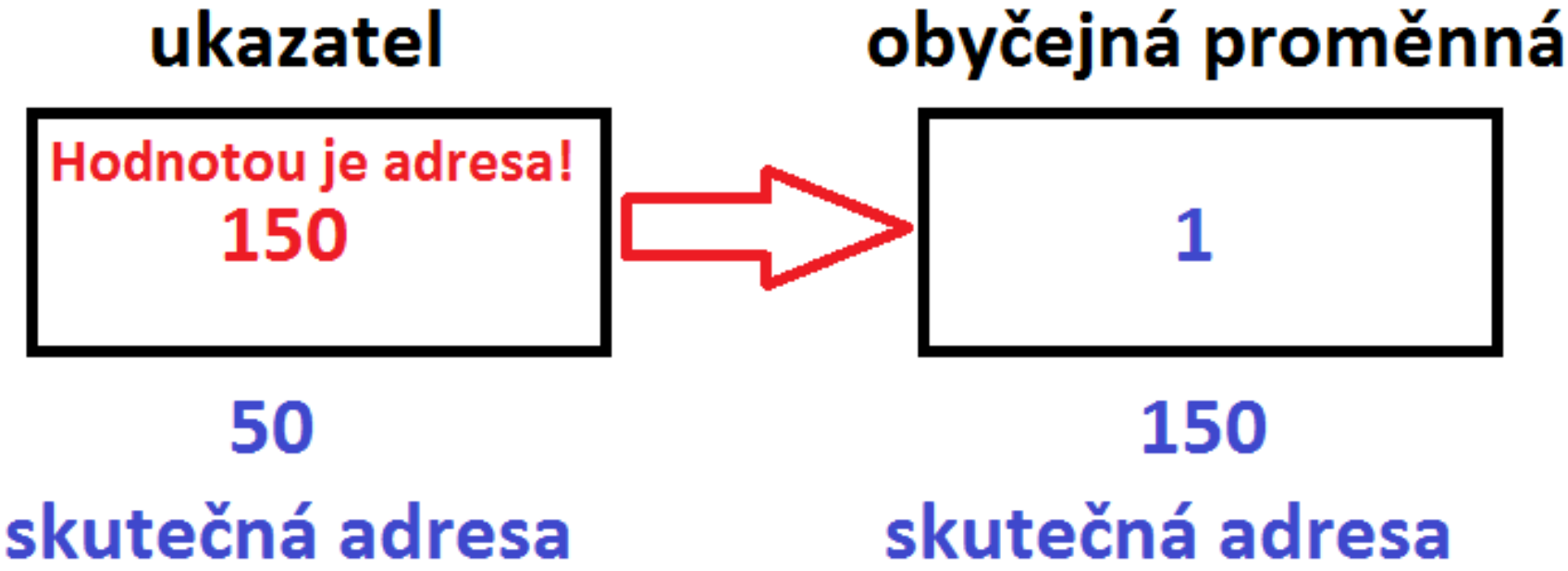
Ukazatel, reference, dereference

- **Proměnná**
- **Ukazatel (pointer)**
- **Velikost ukazatele**
- **Jak získám adresu proměnné?**
- **Jak získám hodnotu z adresy?**
- **Co se používá pro inicializaci ukazatelů?**

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

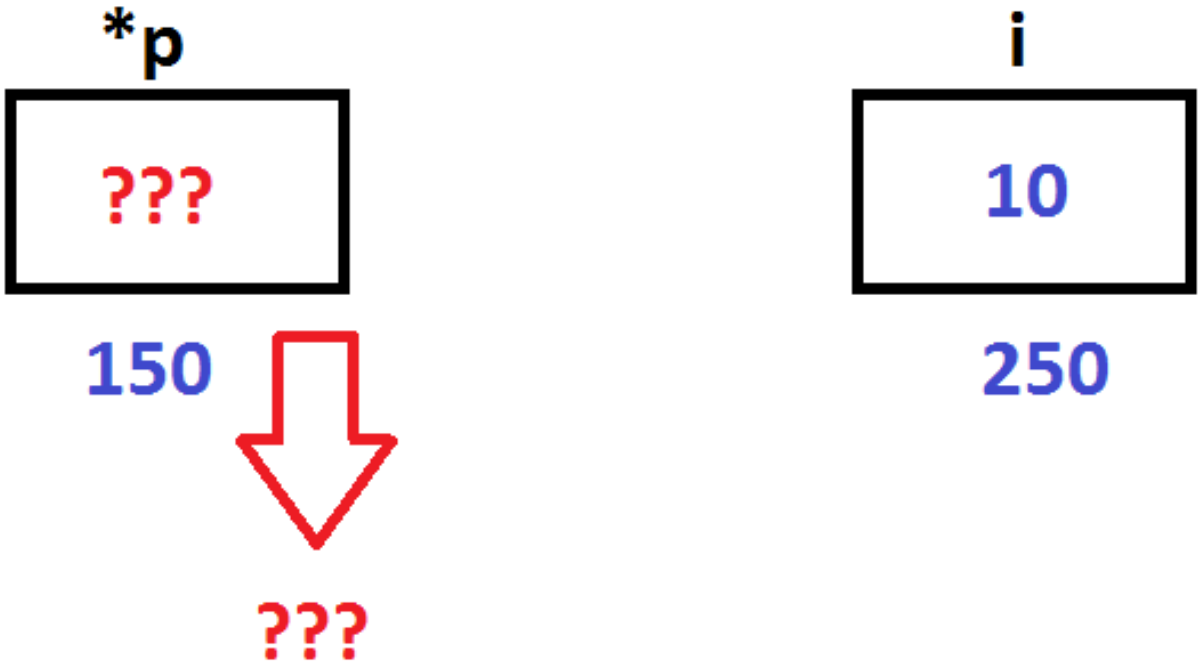


Ukazatel, reference, dereference

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

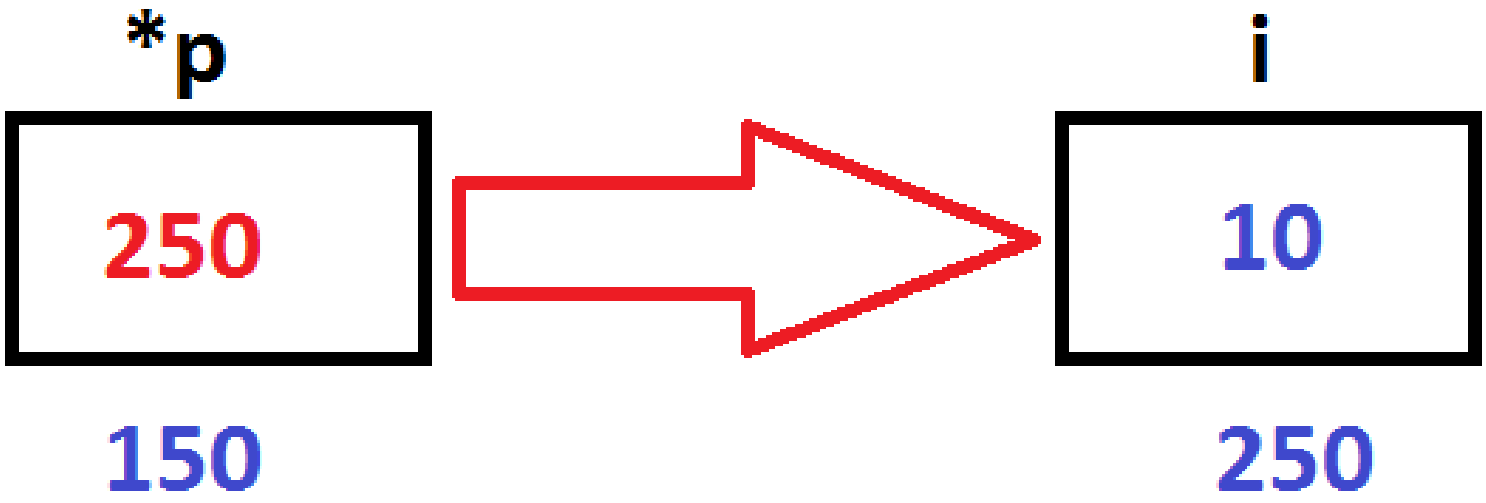
Ukazatel, reference, dereference

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



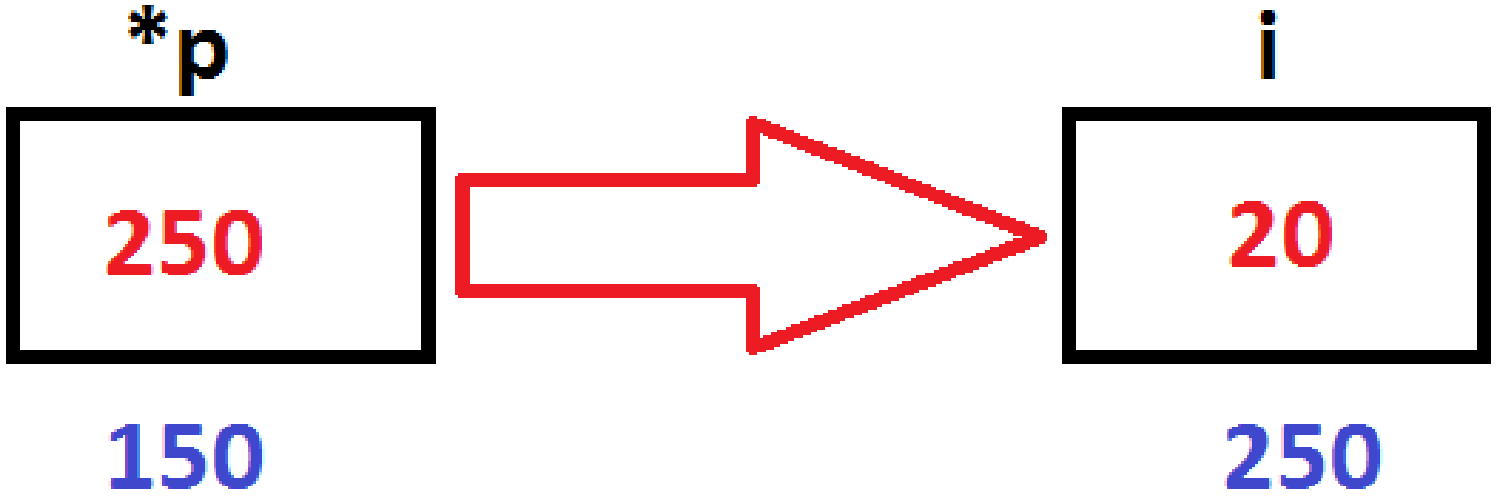
Ukazatel, reference, dereference

```
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);
```



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
           // sizeof(int)
```

Pozor (inicializace vs operátor):

```
int* p;            vs            b = *p;
```

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 … ☺

Předání parametrů funkcím – předávání odkazem

- 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

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


  …
}
```

Funkce plus

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

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

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

Funkce provádí v = a+b .

Funkce `swap`

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

STRUKTURY, VLASTNÍ DATOVÉ TYPY

Pole x Struktura

- **Pole**
  - **Homogenní**
  - **Musí** obsahovat položky **stejného** datového typu

- **Struktura**
  - **Heterogenní**
  - **Může** obsahovat položky **různého** datového typu

- Klíčové slovo **struct**

```
struct person {
  char* name;       // jmeno
  char* surname;    // prijmeni
  int pay;          // plat
};
int main() {
  struct person test;
  test.pay = 1000;
  test.name = "Pepa"; //podobne surname
  printf("pay:  %d\n", test.pay);
  printf("name: %s\n", test.name);   }
```

Struktury – vlastní datový typ

- Nový datový typ – klíčové slovo **typedef**

```
typedef struct person {
  char* name;       // jmeno
  char* surname;    // prijmeni
  int pay;          // plat
}TPerson;
int main() {
  Tperson test;
  test.pay = 1000;
  test.name = "Pepa"; //podobne surname
  printf("pay:  %d\n", test.pay);
  printf("name: %s\n", test.name);   }
```

Struktury – demo: alokace paměti

```
typedef struct person {
  char* name;       // jmeno
  char* surname;    // prijmeni
  int pay;          // plat
}TPerson;
int main() {
  Tperson test;
  test.name = // jak alokujeme paměť?

}
```

Struktury – demo: alokace paměti

```
typedef struct person {
  char* name;       // jmeno
  char* surname;    // prijmeni
  int pay;          // plat
}TPerson;
int main() {
 Tperson test;
 test.name =
 malloc((strlen("Pepa")+1)*sizeof(char);
  // jak ověříme, že je alokace OK?
}
```

Struktury – demo: alokace paměti

```
typedef struct person {
  char* name;       // jmeno
  char* surname;    // prijmeni
  int pay;           // plat
}TPerson;
int main() {
 Tperson test;
 test.name=
 malloc((strlen("Pepa")+1)*sizeof(char);
 if(test.name == NULL) {
    fprintf(stderr, "Chyba alokace!„);
    return -1;
  }
}
```

Struktury – přístup k prvkům

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

```
void setPay(TPerson* p)
{
   // ?
}
```

  - **. (tečka)** jindy

```
TPerson setPay(TPerson p)
{
   // ?
}
```

Struktury – přístup k prvkům

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

```
void setPay(TPerson* p)
{
  p->pay = 1000;
}
```

  - **. (tečka)**

```
TPerson setPay(TPerson p)
{
  p.pay = 10;
  return p;
}
```

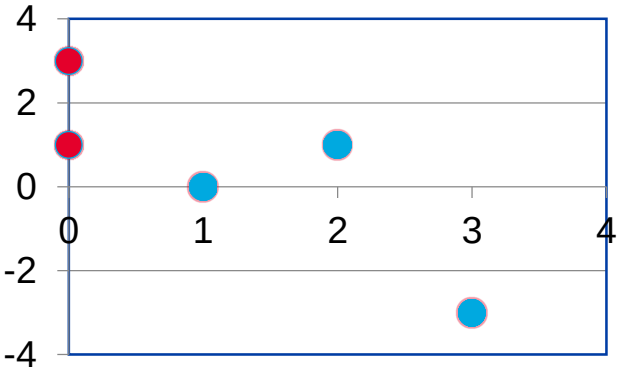
Příklad

- Definujte datový typ pro dvojici čísel.
- Implementujte funkci, která provede inverzi bodu (zamění hodnoty ve dvojici).

- **void tuple_invert(Tuple *point);**

- Dále: invertujte všechny body v relaci a zjistěte, jestli každému x odpovídá právě jedno y (viz obrázek)

```
typedef struct{
…
} Pair;
Pair rel1[] = {  {1, 2},  {2, 3}, {1, 3} };
```



DÚ

Pokud jste tuto hodinu vůbec nevěděli, o čem je řeč, je potřeba si všechny příklady projít znovu.

Dále si přečtěte kapitolu o ukazatelích v učebnici jazyka C od Herouta (kap. 10, str. 144).

Děkuji Vám za pozornost!