# Základy programování (IZP)

## Jedenácté počítačové cvičení

Brno University of Technology, Faculty of Information Technology
Božetěchova 1/2, 612 66 Brno - Královo Pole
Gabriela Nečasová, inecasova@fit.vutbr.cz


2016/2017

# Důležité informace

- **Můj profil**: http://www.fit.vutbr.cz/~inecasova/
  - Kancelář: A221
  - Konzultační hodiny: po domluvě emailem
  - Karta Výuka → odkaz na osobní stránky:
    IZP 2016/2017 Cvičení → Materiály
- **Komunikace**: email – prosím používejte předmět:
  IZP - <předmět emailu>

# Obsah cvičení

- **Seznámení se zadáním třetího projektu**
- **Práce s ukazateli**
  - Opakování, reference, dereference, datový typ pole
- **Struktury, vlastní datové typy**

- **Praktické příklady**
  - Struktury (datový typ **`Object`**)
  - Pole (datový typ **`Array`**)

# SEZNÁMENÍ SE TŘETÍM PROJEKTEM

**Shluková analýza** (max 10 bodů)

- Obhajoba: 5.12.2016
- Odevzdání: **11.12.2016, 23:59:59** do **WISu**

- Název: **proj3.c**
  **(stáhněte si kostru projektu, doplnit TODO sekce)**
- Vstupy/výstupy:
  - **Vstup**: textový soubor
  - **Výstup**: standardní výstup (**`stdout`**)
  - **Chyby**: standardní chybový výstup (**`stderr`**)
- **Překlad**: opět nutný **`-lm`** (výpočet vzdálenosti obj.)

```
gcc -std=c99 -Wall -Wextra -Werror -DNDEBUG
proj3.c -o proj3 -lm
```

# Syntaxe spuštění a formát souboru

- **Syntax spuštění**

```
./proj3 SOUBOR [N]
```

  - **SOUBOR** – vstupní soubor s daty
  - **N** – nepovinný argument, cílový počet shluků, default:1

- **Formát vstupního souboru**

| | |
|---|---|
| `count=N` | počet objektů, ostatní ignorovat |
| `OBJID X Y` | ID objektu a jeho souřadnice |

- Podmínky:
  - N > 0 (`int`)
  - 0 <= X <= 1000, 0 <= Y <= 1000 (`float`)

- **2. podúkol**: základní funkce

```
void init_cluster(struct cluster_t *c,
                  int cap);
void clear_cluster(struct cluster_t *c);
void append_cluster(struct cluster_t *c,
                    struct obj_t obj);
int load_clusters(char *filename, struct
                  cluster_t **arr);
```

- **3. podúkol**: další funkce

```
void merge_clusters(struct cluster_t *c1,
                    struct cluster_t *c2);
int remove_cluster(struct cluster_t *carr,
                   int narr, int idx);
float obj_distance(struct obj_t *o1,
                   struct obj_t *o2);
float cluster_distance(struct cluster_t *c1,
                       struct cluster_t *c2);
void find_neighbours(struct cluster_t *carr,
                     int narr, int *c1, int *c2);
```

- **Aplikace na vizualizaci shluků (online)**
  - Nakopírovat výstupy vašeho programu
  - Pozor: každý řádek musí být končen znakem konce řádku

- Wiki - stránka 3. projektu: http://www.fit.vutbr.cz/study/courses/IZP/public/cluster.php

# PRÁCE S UKAZATELI

# Ukazatel, reference, dereference

- **Ukazatele**
- **Velikost ukazatele**
- **Adresa proměnné**
- **Hodnota z adresy**
- **NULL**

# Ukazatel, reference, dereference

- **Ukazatele** – proměnné uchovávají adresu, která ukazuje do paměti počítače
- **Velikost ukazatele**
- **Adresa proměnné**
- **Hodnota z adresy**
- **NULL**

# Ukazatel, reference, dereference

- **Ukazatele** – proměnné uchovávají adresu, která ukazuje do paměti počítače
- **Velikost ukazatele** – závisí na tom, kolika bitový máme procesor/překladač  (16, 32, 64 bitů)
- **Adresa proměnné**
- **Hodnota z adresy**
- **NULL**

# Ukazatel, reference, dereference

- **Ukazatele** – proměnné uchovávají adresu, která ukazuje do paměti počítače
- **Velikost ukazatele** – závisí na tom, kolika bitový máme procesor/překladač (16, 32, 64 bitů)
- **Adresa proměnné** – referenční operátor **&**
- **Hodnota z adresy**
- **NULL**

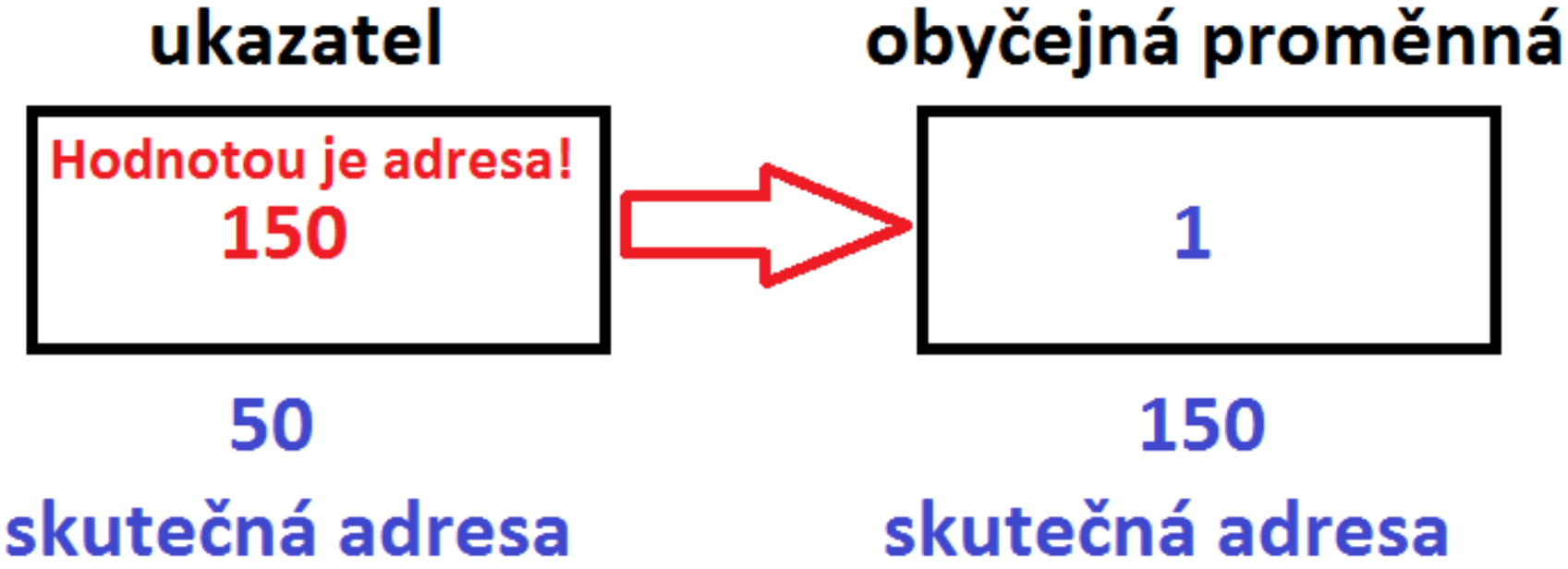
# Ukazatel, reference, dereference

- **Ukazatele** – proměnné uchovávají adresu, která ukazuje do paměti počítače
- **Velikost ukazatele** – závisí na tom, kolika bitový máme procesor/překladač (16, 32, 64 bitů)
- **Adresa proměnné** – referenční operátor **&**
- **Hodnota z adresy** – dereferenční operátor *****
- **NULL**

# Ukazatel, reference, dereference

- **Ukazatele** – proměnné uchovávají adresu, která ukazuje do paměti počítače
- **Velikost ukazatele** – závisí na tom, kolika bitový 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

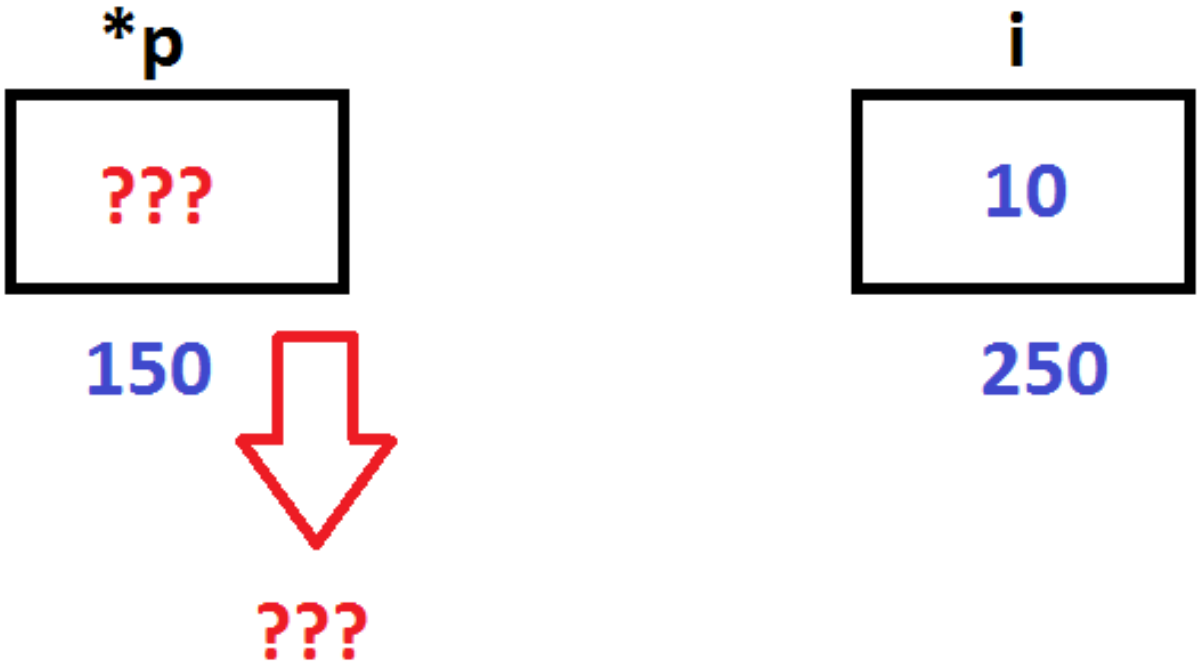


# Ukazatel, reference, dereference

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

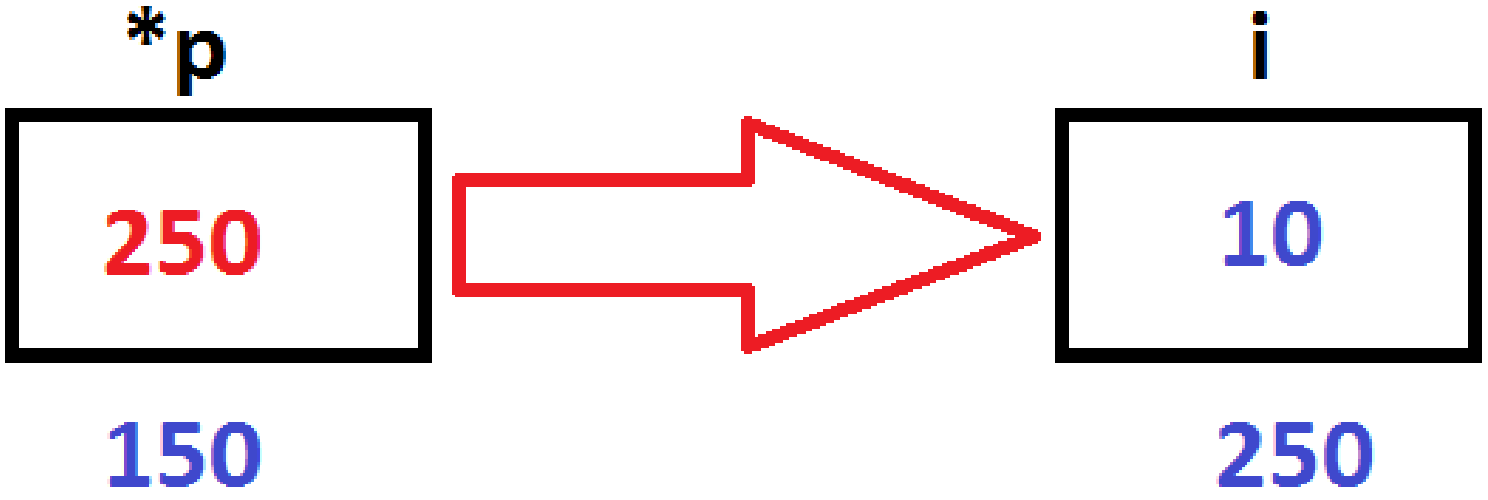
# Ukazatel, reference, dereference

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

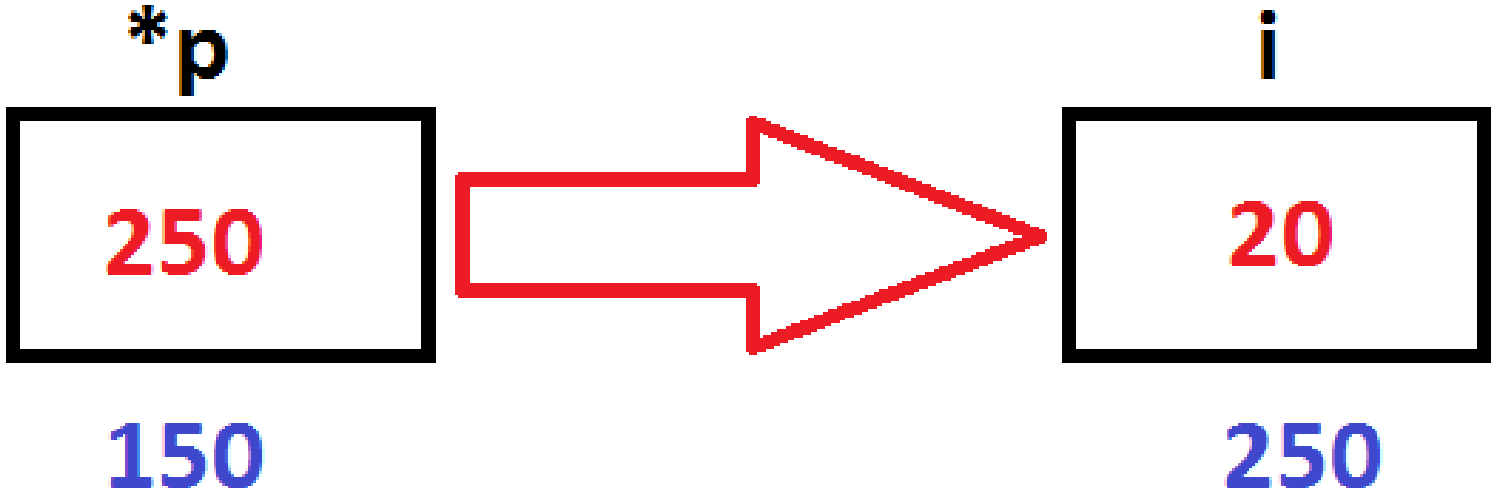


# Ukazatel, reference, dereference

```
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
           // sizeof(int)
```

# DATOVÝ TYP POLE

# Datový typ pole

| Skutečná adresa | 1634 | 1635 | 1636 | 1637 | 1638 | 1639 |
|---|---|---|---|---|---|---|
| Index | 0 | 1 | 2 | 3 | 4 | 5 |
| Hodnota | 10 | 20 | 30 | 40 | 50 | 60 |

# Datový typ pole

| Skutečná adresa | 1634 | 1635 | 1636 | 1637 | 1638 | 1639 |
|---|---|---|---|---|---|---|
| Index | 0 | 1 | 2 | 3 | 4 | 5 |
| Hodnota | 10 | 20 | 30 | 40 | 50 | 60 |

- **Pole:** prvky stejného typu, spojité místo v paměti

# Datový typ pole

| Skutečná adresa | 1634 | 1635 | 1636 | 1637 | 1638 | 1639 |
|---|---|---|---|---|---|---|
| Index | 0 | 1 | 2 | 3 | 4 | 5 |
| Hodnota | 10 | 20 | 30 | 40 | 50 | 60 |

- **Pole:** prvky stejného typu, spojité místo v paměti
- **Deklarace staticky**: `int moje_pole[6];`

# Datový typ pole

| Skutečná adresa | 1634 | 1635 | 1636 | 1637 | 1638 | 1639 |
|---|---|---|---|---|---|---|
| Index | 0 | 1 | 2 | 3 | 4 | 5 |
| Hodnota | 10 | 20 | 30 | 40 | 50 | 60 |

- **Pole:** prvky stejného typu, spojité místo v paměti
- **Deklarace staticky**: `int moje_pole[6];`
- **Velikost pole**: operátor `sizeof()` – velikost v bajtech
  - U polí vrací součet velikostí jeho položek

# Datový typ pole

| Skutečná adresa | 1634 | 1635 | 1636 | 1637 | 1638 | 1639 |
|---|---|---|---|---|---|---|
| Index | 0 | 1 | 2 | 3 | 4 | 5 |
| Hodnota | 10 | 20 | 30 | 40 | 50 | 60 |

- **Pole:** prvky stejného typu, spojité místo v paměti
- **Deklarace staticky**: `int moje_pole[6];`
- **Velikost pole**: operátor `sizeof()` – velikost v bajtech
  - U polí vrací součet velikostí jeho položek
- **Velikost moje_pole:**

```
int velikost = 6*sizeof(int);
```

# Datový typ pole

| Skutečná adresa | 1634 | 1635 | 1636 | 1637 | 1638 | 1639 |
|---|---|---|---|---|---|---|
| Index | 0 | 1 | 2 | 3 | 4 | 5 |
| Hodnota | 10 | 20 | 30 | 40 | 50 | 60 |

- **Pole:** prvky stejného typu, spojité místo v paměti
- **Deklarace staticky**: `int moje_pole[6];`
- **Velikost pole**: operátor `sizeof()` – velikost v bajtech
  - U polí vrací součet velikostí jeho položek
- **Velikost moje_pole:**
  `int velikost = 6*sizeof(int);`
- **Pozor**: velikost datového typu záleží na procesoru
  - `sizeof(int)` může být 2, 4 nebo 8

# STRUKTURY, VLASTNÍ DATOVÉ TYPY

- **Pole**

- **Struktura**

- **Pole**
  - **Homogenní**
  - **Musí** obsahovat položky **stejného** datového typu
- **Struktura**
  - **Heterogenní**
  - **Může** obsahovat položky **různého** datového typu

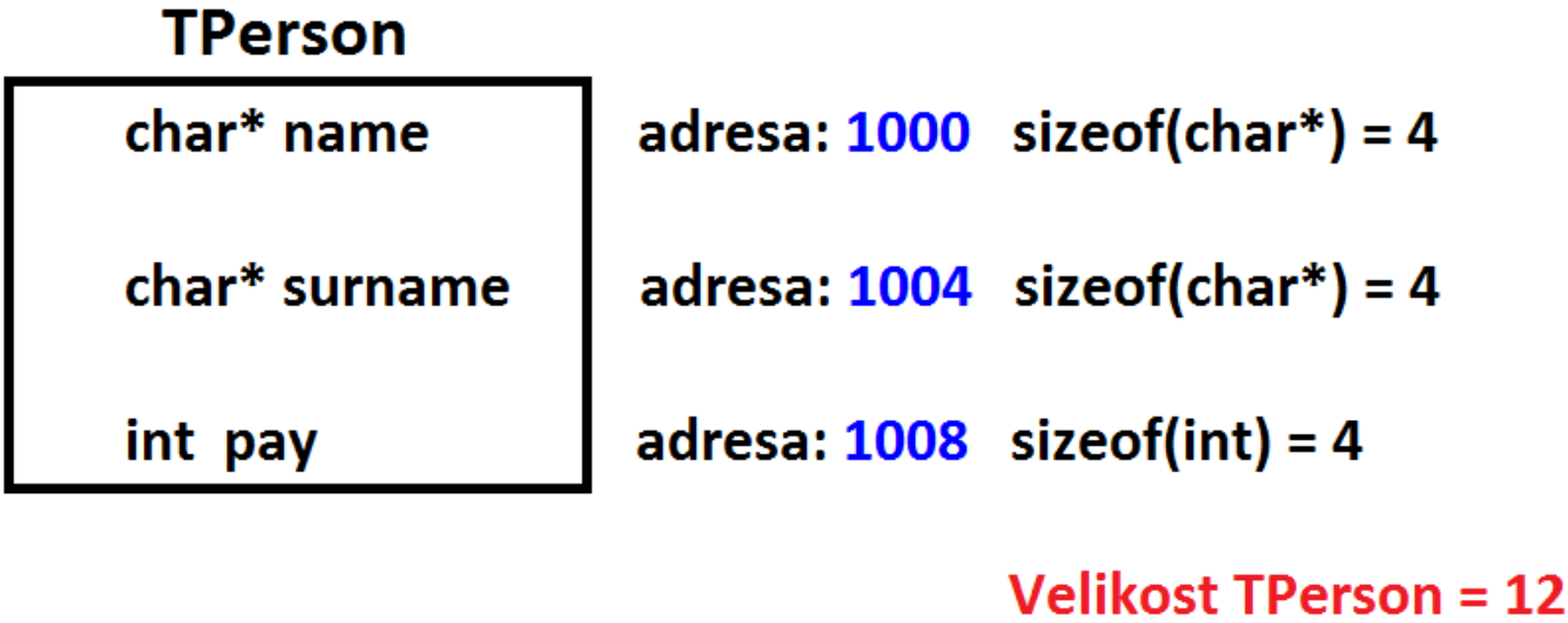
- Klíčové slovo **struct**

```
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);   }
```

- 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);   }
```

- Velikosti datových typů závisí na architektuře



# Struktury – demo: alokace paměti

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

}
```

```
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;
  }
}
```

- 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)** jindy

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

# PŘÍKLADY

- Stáhněte si **připravené kostry programů:**
- Budete pouze doplňovat části kódu
- V přiloženém ZIP archivu se nachází soubor **`.c`** a **`.h`** a také **`Makefile`**
- **http://www.fit.vutbr.cz/~inecasova/web/cv11.zip**
- **Možnosti práce se soubory**
  - **Windows – Code::Blocks:** vytvořit nový projekt, pravým tlačítkem klik na projekt → Add files…
  - **Linux – terminál:** překlad pomocí `Makefile` (případně samozřejmě též Code::Blocks)
- **Kontrola práce s pamětí**

```
valgrind --leak-check=full ./main
```

# STRUKTURY

# Typ Object

- Budeme používat strukturovaný datový typ **`Object`**

```
typedef struct {
  int id;        // ID
  char* name;    // jmeno
} Object;
```

- Vašim úkolem bude implementovat
  - **konstruktor**, který bude objekt inicializovat
  - **destruktor**, který bude uvolňovat použitou paměť
  - funkci, která **vymění obsah** dvou objektů
  - funkci, **která zkopíruje obsah objektu**
- Kostry jsou v souboru **`struct.c`**

# Typ Object – několik rad

- K alokaci paměti pro řetězec použijte funkci **`malloc()`**
- Po alokaci kontrolujte, jak dopadla:

```
int* pole = malloc(2*sizeof(int));
if (pole == NULL) {
  fprintf(stderr, "Chyba alokace!\n");
  return -1;
}
```

- Pro záměnu dat objektů budete potřebovat nějaký **pomocný objekt**
- Ve funkci **`item_copy()`** je možné použít konstruktor **`object_ctor()`**

# POLE STRUKTUR

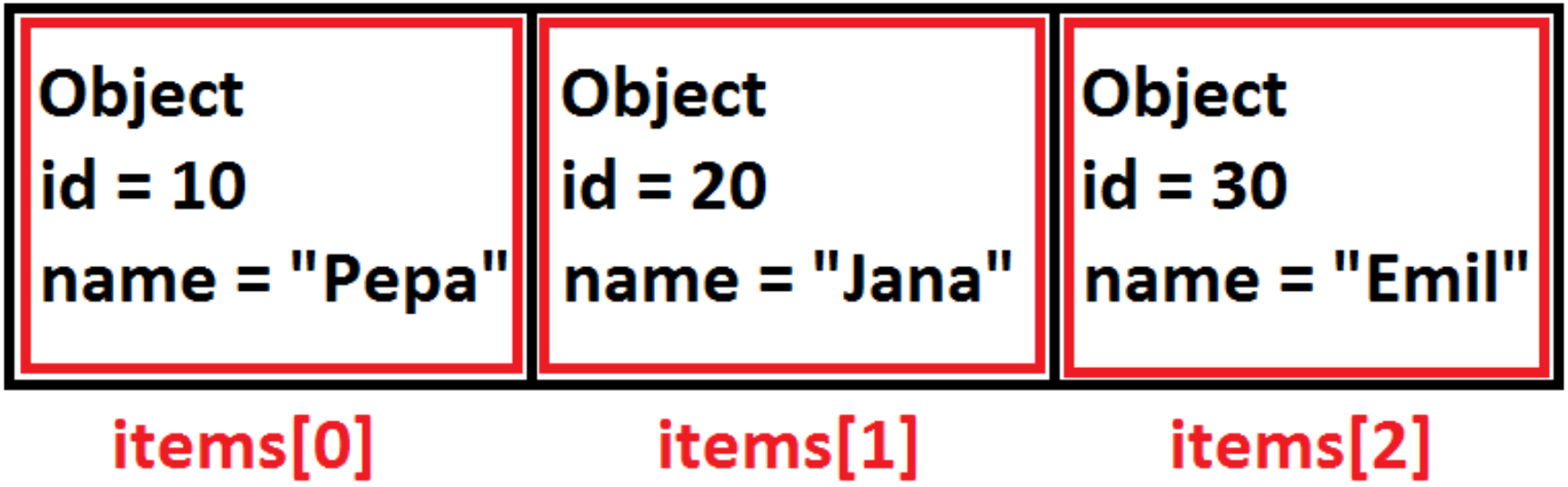
- Nyní vytvoříme pole objektů typu **`Object`**

```
typedef struct {
  unsigned size; // velikost pole
  Object* items; // pole objektu
} Array;
```

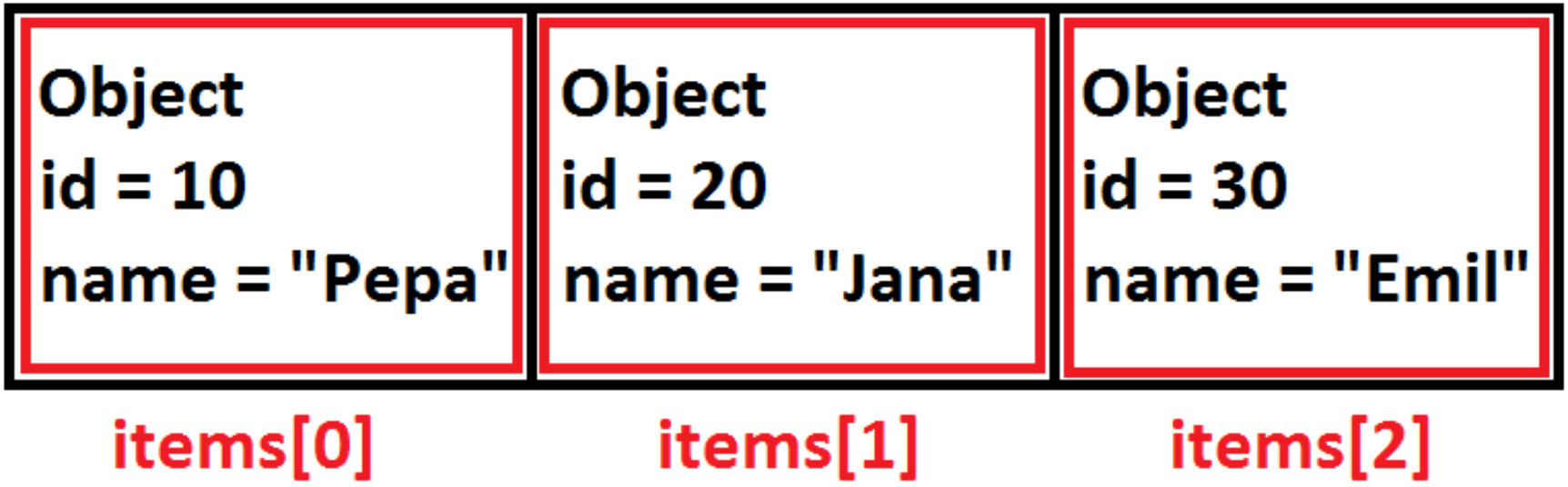
- Vašim úkolem bude implementovat
  - **Konstruktor** a **destruktor**
  - Funkci pro **vložení objektu** na daný index
  - Funkci pro **nalezení jména/ID**
- Kostry jsou v souboru **`array.c`**
- **Ostatní funkce za DÚ**

# Pole struktur – ilustrace

- Pole objektů: **`Object* items;`**
  - Na každém indexu pole je struktura **`Object`**



- Pole objektů: **`Object* items;`**
  - Na každém indexu pole je struktura **`Object`**




**Array**
- **items – pole objektů**
- **size = 3**

Děkuji Vám za pozornost!