

Základy programování (IZP)

Deváté počítačové cvičení

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

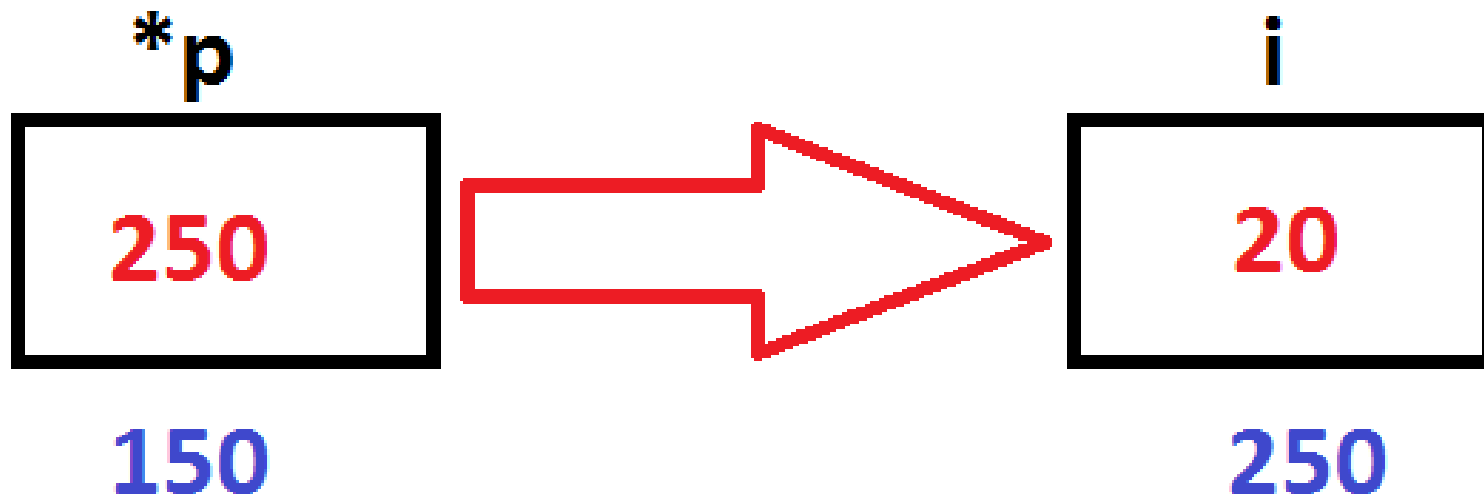


- **Dnešní cvičení je za 2 body**
- **Druhý projekt**
- **Práce s ukazateli**
- **Praktické příklady**

DRUHÝ PROJEKT

PRÁCE S UKAZATELI

```
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);
```



- 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); }  
}
```

```
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)
{
    p->pay = 1000;
}
```

- . (**tečka**) jindy

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

PŘÍKLAD

- Stáhněte si **kostru programu**
- Budete opravovat chyby a doplňovat funkcionalitu
- **Možnosti práce se soubory**
 - **Linux - terminál:** překlad pomocí `Makefile`
- **Kontrola práce s pamětí**
 - Je vhodné překládat s ladícími informacemi: `-g`

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

kde `main` je název binárního souboru

- Budeme používat strukturovaný datový typ **Person**

```
typedef struct {  
    unsigned year;  
    char* name; // jmeno  
} Person;
```

- Některé funkce jsou již v kostře implementovány, některé je třeba doplnit a upravit.
- Je třeba
 - U konstrukturu doplnit kontrolu úspěchu alokace
 - U destrukturu doplnit uvolnění paměti pro jméno
 - Provedení hluboké kopie osoby
 - Je nutné osobu (dst) inicializovat hodnotami ze src

```
typedef struct {  
    unsigned size; // velikost pole  
    Person* data; // pole  
} Array;
```

- Je třeba doplnit
 - U destruktoru uvolnění jednotlivých položek (uvolnění celého pole je v kostře)
- A dále doimplementovat
 - Seřazení prvků (algoritmus Selection sort)
 - Nalezení prvku s minimální hodnotou (binární vyhledávání v seřazeném poli)
 - Pro dvě výše uvedené funkce budete používat funkci `person_cmp`

- 1) Pole je rozděleno na **seřazenou část** a **neseřazenou část**.
- 2) V neseřazené části se nalezne minimum a vymění se s prvkem bezprostředně následujícím za seřazenou částí a tento prvek se do ní zahrne.
- 3) Na začátku má seřazená část délku nula, na konci má délku pole.

Neseřazená část Seřazená část Nejmenší prvek

6	60	42	200	4	3
3	60	42	200	4	6
3	4	42	200	60	6
3	4	6	200	60	42
3	4	6	42	60	200
3	4	6	42	60	200
3	4	6	42	60	200

```
int array_find_bin(PersonArray *a, Person *p);
```

- 1) Indexy začátku a konce vyhledávání nastavíme na index prvního a posledního prvku pole.
- 2) Spočítáme průměrnou hodnotu těchto indexů a převedeme na int
 - Tj. máme index prvku uprostřed pole
 - Na tomto indexu by se prvek mohl nacházet (**odhad**)
- 3) Pokud se na tomto indexu prvek nachází, končíme.
- 4) Pokud je prvek na indexu **větší** než hledaný, **index konce** vyhledávání nastavíme na **odhad - 1**
- 5) Pokud je prvek na indexu **menší** než hledaný, **index začátku** vyhledání nastavíme na **odhad + 1**
- 6) Zpět na bod 2

Mid point

Začátek vyhledávání

Konec vyhledávání

3	4	6	42	60	200
3	4	6	42	60	200

$$(0 + 5) / 2 = 2$$

$$(0 + 1) / 2 = 0$$

Prvek 3 nalezen

02-fib.c

REKURZE

- Implementujte program pro výpočet Fibbonaciho čísel
 - Algoritmus v komentáři zdrojového kódu
- Zjistěte, kolikrát byla funkce volána (výjimečně můžete použít globální proměnnou)
- Pokuste se zrychlit výpočet s použitím pole již vypočítaných výsledků

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181

- Implementujte program pro výpočet hodnoty faktoriálu
- $0! = 1$
- $1! = 1$
- $n! = n * n-1!$

Děkuji Vám za pozornost!