

Základy programování (IZP)

Páté počítačové cvičení

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



17.10.2016

- **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>

- **Nezapomeňte** se ve **WISu** přihlásit na všechny **4 termíny**:
 - 3 projekty IZP
 - 1 dokumentace ke třetímu projektu
- **Pozor**: přihlašujte se ke **správnému asistentovi!**
- Přihlašování začalo **10. 10. 2016 v 8:00**
- Přihlašování končí 30. 10. 2016 ve 22:00

- **Seznámení se zadáním prvního projektu**
- **Jak by měl vypadat zdrojový kód? Ukázka**
- **Příklady**
 - Načítání znaků ze standardního vstupu po znacích
 - Datový typ pole (😊 😊 😊)
 - Ukazatel, reference, dereference
 - Předávání parametrů funkcím – hodnotou, odkazem
 - Podpůrné funkce (převod číselné soustavy, vyhledání maxima apod.).

1. PROJEKT – PRÁCE S TEXTEM

- Základní informace
 - 31. 10. Obhajoba projektu
 - Měli byste mít hotovou implementaci 😊
 - 6.11. Odevzdání projektu do WISu
 - Jméno souboru: **proj1.c**
- Zadání viz Wiki stránky
- **POZOR, v projektu nelze použít**
 - Hlavičkový soubor **string.h**
 - Funkce **malloc** a **free** (dynam. alokace paměti)
 - Funkci **scanf** a jeho varianty
 - Funkci **atoi**
 - Práce s dočasnými soubory – funkce **fopen**, **fclose**, atd.

- **Co se hodnotí?** (Pořadí podle důležitosti)
 - přeložitelnost zdrojového souboru,
 - formát zdrojového souboru (členění, zarovnání, komentáře, vhodně zvolené identifikátory),
 - dekompozice problému na podproblémy (vhodné funkce, vhodná délka funkcí a parametry funkcí),
 - správná volba datových typů, případně tvorba nových typů,
 - správná funkcionality převodu dat a
 - ošetření chybových stavů.

- Převody
 - **BIN** → **TEXT**, výstup jsou:
 - adresy vstupních bajtů
 - hexadecimální kódování a
 - textovou reprezentaci obsahu
 - **TEXT** → **BIN**, výstup jsou:
 - hexadecimální kódování bajtů
- Překlad programu

```
gcc -std=c99 -Wall -Wextra -Werror  
proj1.c -o proj1
```

- **-Werror** → všechna varování interpretuje jako chyby!

- Spuštění

```
./proj1
```

```
./proj1 [-s M] [-n N]
```

```
./proj1 -x
```

```
./proj1 -S N
```

```
./proj1 -r
```

BIN → TEXT

ADDR HEX |STRING|

-s M (skip)

-n N (max. počet
přečtených bajtů)

BIN → TEXT (hex)

BIN → TEXT

(string, délka \geq N)

TEXT (hex) → BIN

JAK BY MĚL VYPADAT ZDROJOVÝ KÓD?

- **Dekomponujte problém na podproblémy** → tvorba vlastních funkcí
- Vhodně **pojmenovávejte** funkce a proměnné
- Dbejte na **přehlednost kódu** – odsazení!
 - Tělo funkce
 - Příkazy v if, else if, else
 - Těla cyklů
- Pište **komentáře**
 - Vhodné je psát komentáře ke každé definici funkce, a také k určitým částem kódu

```
// potrebne knihovny
#include <stdio.h>
#include <stdlib.h>

// definice funkci (dekompozice problemu)

// Hlavni program
int main(int argc, char** argv)
{
    // zpracovani argumentu prikazove radky
    // provedeni danych akci
    return 0;
}
```

```
/**
```

```
 * Funkce scita dve cela cisla.
```

```
 * @param a Prvni cislo
```

```
 * @param b Druhe cislo
```

```
 * @return Vraci soucet dvou cisel.
```

```
 */
```

```
int soucet(int a, int b)
```

```
{
```

```
    return a + b;
```

```
}
```

- V jazyce C můžete používat **3 implicitní streamy**, které reprezentují vstup a výstup
 - `stdin` standardní vstup, třeba z klávesnice
 - `stdout` standardní výstup, třeba na monitor
 - `stderr` standardní chybový výstup
- **Chybová hlášení programů** je vhodné (a v projektu nutné) vypisovat na standardní chybový výstup pomocí funkce `fprintf()`

```
#include <stdio.h>

fprintf(stream, format_retezec, dalsi_param);
fprintf(stderr, "Chybove hlaseni\n");
fprintf(stderr, "Malo parametru: %d", ecode);
```

Napište program, který načítá jednotlivé znaky ze standardního vstupu.

- Vypíše **číselnou a znakovou reprezentaci** jednotlivých znaků.
- Nakonec **vypíše počet načtených znaků**.

Napište program, který načítá jednotlivé znaky ze standardního vstupu.

- Vypíše **číselnou a znakovou reprezentaci** jednotlivých znaků.
- Nakonec **vypíše počet načtených znaků**.
- Zadávání **EOF**
 - Windows: CTRL + Z
 - Linux: CTRL + D

| Dec | Hex | Char | Dec | Hex | Char | Dec | Hex | Char | Dec | Hex | Char |
|-----|-----|------------------|-----|-----|-------|-----|-----|------|-----|-----|------|
| 0 | 00 | Null | 32 | 20 | Space | 64 | 40 | @ | 96 | 60 | ` |
| 1 | 01 | Start of heading | 33 | 21 | ! | 65 | 41 | A | 97 | 61 | a |
| 2 | 02 | Start of text | 34 | 22 | " | 66 | 42 | B | 98 | 62 | b |
| 3 | 03 | End of text | 35 | 23 | # | 67 | 43 | C | 99 | 63 | c |
| 4 | 04 | End of transmit | 36 | 24 | \$ | 68 | 44 | D | 100 | 64 | d |
| 5 | 05 | Enquiry | 37 | 25 | % | 69 | 45 | E | 101 | 65 | e |
| 6 | 06 | Acknowledge | 38 | 26 | & | 70 | 46 | F | 102 | 66 | f |
| 7 | 07 | Audible bell | 39 | 27 | ' | 71 | 47 | G | 103 | 67 | g |
| 8 | 08 | Backspace | 40 | 28 | (| 72 | 48 | H | 104 | 68 | h |
| 9 | 09 | Horizontal tab | 41 | 29 |) | 73 | 49 | I | 105 | 69 | i |
| 10 | 0A | Line feed | 42 | 2A | * | 74 | 4A | J | 106 | 6A | j |
| 11 | 0B | Vertical tab | 43 | 2B | + | 75 | 4B | K | 107 | 6B | k |
| 12 | 0C | Form feed | 44 | 2C | , | 76 | 4C | L | 108 | 6C | l |
| 13 | 0D | Carriage return | 45 | 2D | - | 77 | 4D | M | 109 | 6D | m |
| 14 | 0E | Shift out | 46 | 2E | . | 78 | 4E | N | 110 | 6E | n |
| 15 | 0F | Shift in | 47 | 2F | / | 79 | 4F | O | 111 | 6F | o |
| 16 | 10 | Data link escape | 48 | 30 | 0 | 80 | 50 | P | 112 | 70 | p |
| 17 | 11 | Device control 1 | 49 | 31 | 1 | 81 | 51 | Q | 113 | 71 | q |
| 18 | 12 | Device control 2 | 50 | 32 | 2 | 82 | 52 | R | 114 | 72 | r |
| 19 | 13 | Device control 3 | 51 | 33 | 3 | 83 | 53 | S | 115 | 73 | s |
| 20 | 14 | Device control 4 | 52 | 34 | 4 | 84 | 54 | T | 116 | 74 | t |
| 21 | 15 | Neg. acknowledge | 53 | 35 | 5 | 85 | 55 | U | 117 | 75 | u |
| 22 | 16 | Synchronous idle | 54 | 36 | 6 | 86 | 56 | V | 118 | 76 | v |
| 23 | 17 | End trans. block | 55 | 37 | 7 | 87 | 57 | W | 119 | 77 | w |
| 24 | 18 | Cancel | 56 | 38 | 8 | 88 | 58 | X | 120 | 78 | x |
| 25 | 19 | End of medium | 57 | 39 | 9 | 89 | 59 | Y | 121 | 79 | y |
| 26 | 1A | Substitution | 58 | 3A | : | 90 | 5A | Z | 122 | 7A | z |
| 27 | 1B | Escape | 59 | 3B | ; | 91 | 5B | [| 123 | 7B | { |
| 28 | 1C | File separator | 60 | 3C | < | 92 | 5C | \ | 124 | 7C | |
| 29 | 1D | Group separator | 61 | 3D | = | 93 | 5D |] | 125 | 7D | } |
| 30 | 1E | Record separator | 62 | 3E | > | 94 | 5E | ^ | 126 | 7E | ~ |
| 31 | 1F | Unit separator | 63 | 3F | ? | 95 | 5F | _ | 127 | 7F | □ |

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 |

| 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, spojitě místo v paměti

| 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, spojitě místo v paměti
- **Deklarace staticky:** `int moje_pole[6];`

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

| 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, spojitě 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);`

| 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, spojitě 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

| Skutečná adresa | 1634 | 1635 | 1636 | 1637 | 1638 | 1639 |
|--------------------|------|------|------|------|------|------|
| Index | 0 | 1 | 2 | 3 | 4 | 5 |
| Hodnota | 'h' | 'e' | 'l' | 'l' | 'o' | '\0' |

| Skutečná adresa | 1634 | 1635 | 1636 | 1637 | 1638 | 1639 |
|--------------------|------|------|------|------|------|------|
| Index | 0 | 1 | 2 | 3 | 4 | 5 |
| Hodnota | 'h' | 'e' | 'l' | 'l' | 'o' | '\0' |

- **Řetězce:** pole typu `char` zakončená nulovým znakem `'\0'`

| Skutečná adresa | 1634 | 1635 | 1636 | 1637 | 1638 | 1639 |
|--------------------|------|------|------|------|------|------|
| Index | 0 | 1 | 2 | 3 | 4 | 5 |
| Hodnota | 'h' | 'e' | 'l' | 'l' | 'o' | '\0' |

- **Řetězce:** pole typu `char` zakončená nulovým znakem `'\0'`
- **Pozor:** `char pole[5];`
 - Není zde místo pro ukončovací nulu (`'\0'`)
- **Pozor:** je nutné hlídat meze pole!

- **Otázka:** Jaký je rozdíl mezi následujícími inicializacemi pole?

```
char array_1[] = {'h', 'e', 'l', 'l', 'o', '\\0'};  
char array_2[] = "hello";
```

- **Otázka:** Jaký je rozdíl mezi následujícími inicializacemi pole?

```
char array_1[] = {'h', 'e', 'l', 'l', 'o', '\\0'};  
char array_2[] = "hello";
```

- **Odpověď:** Obě inicializace jsou ekvivalentní, ale u `array_2` je koncová nula přidána automaticky.

- **Otázka:** Jaký je rozdíl mezi následujícími inicializacemi pole?

```
char array[] = "hello";  
char *array  = "goodbye";
```

- **Otázka:** Jaký je rozdíl mezi následujícími inicializacemi pole?

```
char array[] = "hello";  
char *array = "goodbye";
```

- **Odpověď:**
 - **hello** je uloženo v poli 6 znaků, hodnoty pole **lze** libovolně měnit, kopie je uložena na zásobníku
 - **goodbye** je řetězcový literál, který **nelze** měnit

- **Definujte:**
 - Pole typu `int` o 5 prvcích a naplňte jej hodnotami
 - Řetězec s libovolným textem

- **Definujte:**

- Pole typu `int` o 5 prvcích a naplňte jej hodnotami
- Řetězec s libovolným textem

```
int array[5] = {5, 10, -10, 2, -2};  
char *str    = "Hello world!";
```

- Napište program, který
 - vypočítá sumu všech prvků v poli `array`
 - vypíše počet znaků v řetězci `str`

UKAZATELE

- **Proměnná**
- **Ukazatel (pointer)**
- **Velikost ukazatele**
- **Adresa proměnné**
- **Hodnota z adresy**
- **NULL**

- **Proměnná** – pojmenované místo v paměti, ve kterém uchováváme data

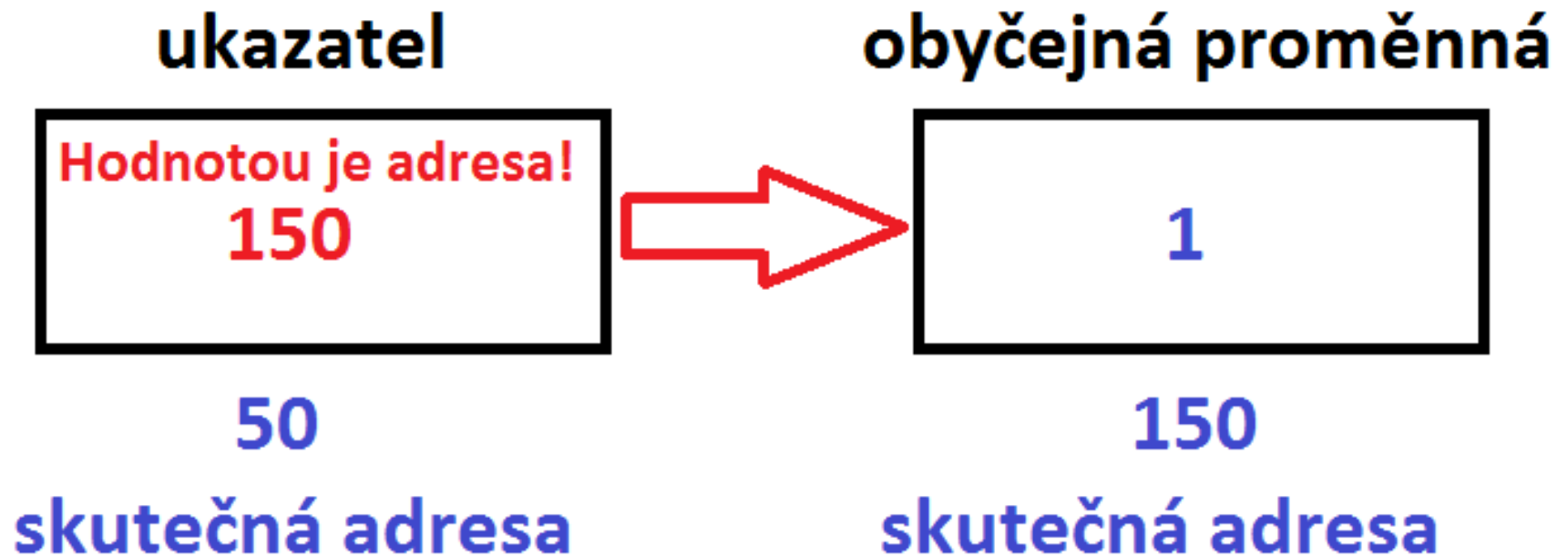
- **Proměnná** – pojmenované místo v paměti, ve kterém uchováváme data
- **Ukazatel (pointer)** – proměnná, která uchovává adresu nějakého místa v paměti
 - Říkáme, že ukazatel ukazuje na místo, které je určeno touto adresou

- **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)

- **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 **&**

- **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 *****

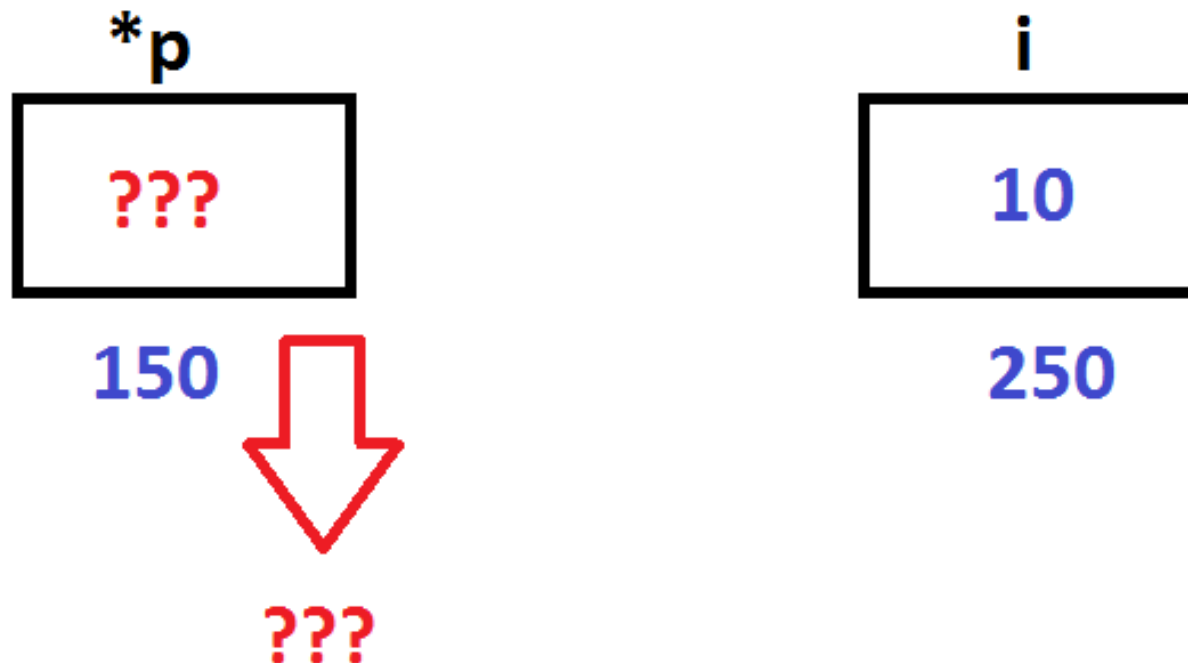
- **Proměnná** – pojmenované místo v paměti, ve kterém uchováváme data
- **Ukazatel (pointer)** – proměnná, která uchovává adresu nějakého místa v paměti
 - Říkáme, že ukazatel ukazuje na místo, které je určeno touto adresou
- **Velikost ukazatele** – závisí na tom, kolikabitový máme procesor/překladač (16, 32, 64 bitů)
- **Adresa proměnné** – referenční operátor **&**
- **Hodnota z adresy** – dereferenční operátor *****
- **NULL** – používá se pro inicializaci ukazatelů – říká, že ukazatel nikam neukazuje



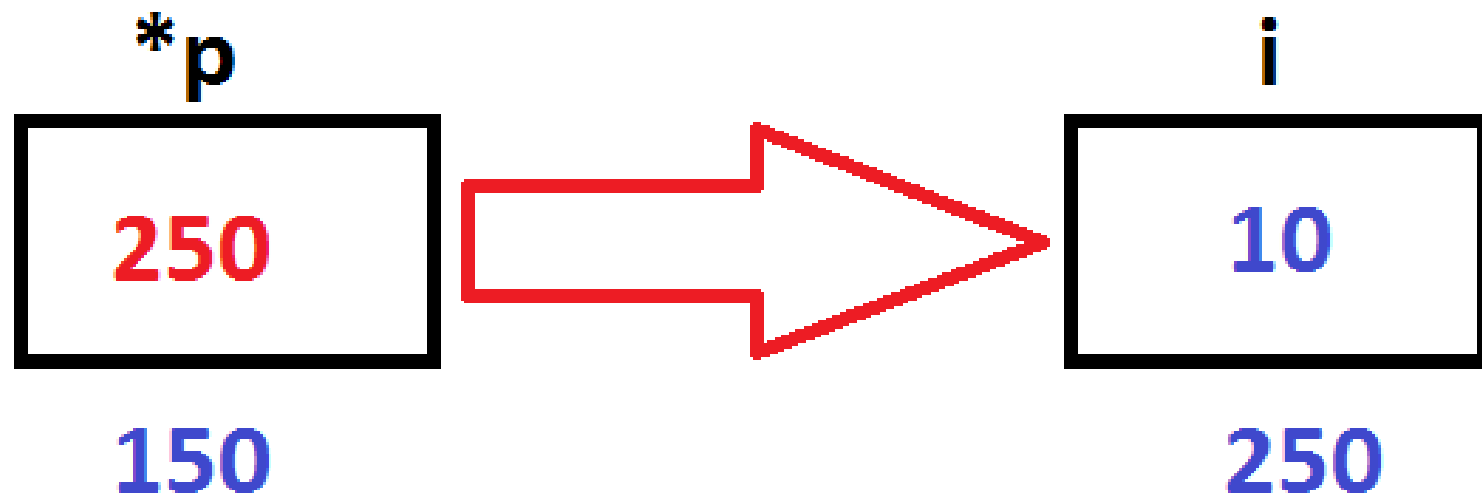
```
int i = 10;  
int *p;  ///  
p = &i;  ///  
*p = 20; ///  
printf("Hodnota promenne i: %d\n", i); ///  

```

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



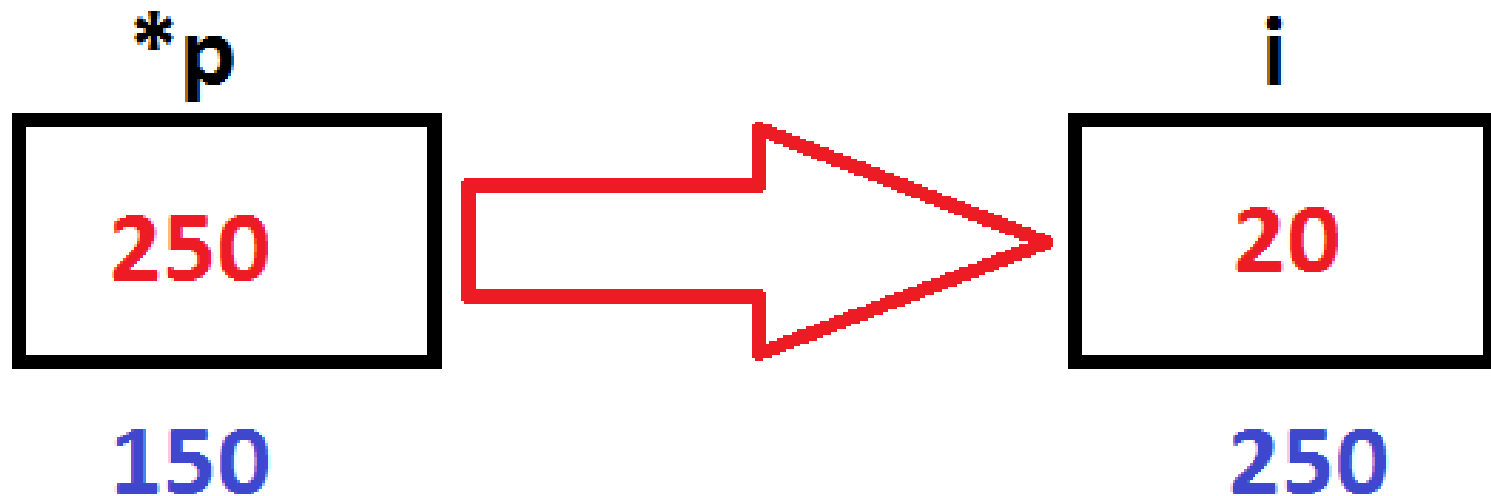
```
int i = 10;
int *p; // ukazatel p není inicializován!
p = &i; // ukazatel p ukazuje na i
*p = 20;
printf("Hodnota promenne i: %d\n", i);
```



```
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);
```



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

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



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

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

```
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 -->
```

```
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)
```

FUNKCE

- Do teď jsme vše zapisovali do funkce
`int main(...) {...}`
- Nyní začneme vytvářet vlastní funkce
- Obecně může **deklarace funkce** vypadat např. takto:

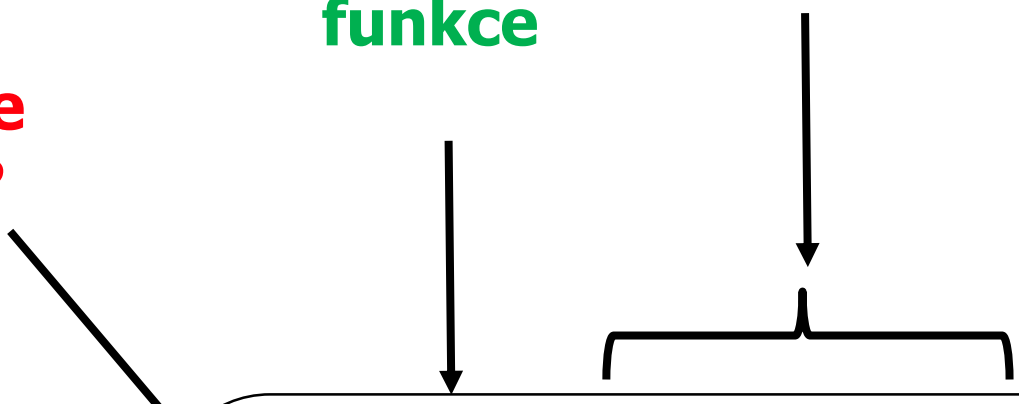
```
int max(int a, int b); // hlavička
```

- **int** - datový typ, který funkce vrací
 - pokud nic nevrací, použijeme typ **void**
- **max** – název funkce
- **int a, int b** – parametry, které funkce očekává
 - všimněte si datových typů
 - proměnné **a, b** jsou deklarovány a lze je použít v těle funkce

Co
chceme
získat?

Název
funkce

Co funkce potřebuje
pro výpočet?



```
int max(int a, int b)
{
    // Co musí funkce
    provést, aby dosáhla
    výsledku
}
```

- Minule jsme zkoušeli porovnávat dva řetězce. Používali jsme k tomu funkci `strcmp()`, která byla deklarována takto

```
int strcmp(char* s1, char* s2);
```


- Parametry můžeme funkcím předávat
 - **Hodnotou** (vytvoříme lokální kopii proměnné)
 - **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**, definujeme následující funkci

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

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

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

- Napište dvě verze funkce **max**, která určí větší ze dvou celých čísel.
 - **Verze 1:** funkce vrátí větší ze dvou čísel (**int**)
 - **Verze 2:** funkce nic nevrací (**void**)

- Naše funkce **max** se dá z funkce **main** zavolat následovně (předání parametrů hodnotou)

```
int main()  
{  
    ...  
    int a = 5;  
    int b = 10;  
    // v promenne vetsi bude  
    // výsledek volani  
    int vetsi = max(a,b) ;  
}
```

Napište funkci, která převede **číslo v osmičkové soustavě**, reprezentované jako **řetězec**, do **desítkové soustavy**.

Napište funkci, která převede **číslo v osmičkové soustavě**, reprezentované jako **řetězec**, do **desítkové soustavy**.

Převod mezi soustavami:

$$70_8 = 7 \cdot 8^1 + 0 \cdot 8^0 = 56 + 0 = 56_{10}$$

Napište funkci, která převede **číslo v osmičkové soustavě**, reprezentované jako **řetězec**, do **desítkové soustavy**.

Převod mezi soustavami:

$$70_8 = 7 \cdot 8^1 + 0 \cdot 8^0 = 56 + 0 = 56_{10}$$

Převod písmeno → číslice pomocí ASCII:

$$7 = '7' - '0' = 55 - 48 = 7$$

Lze to takto řešit? (Bez znalosti délky řetězce?)

POZNÁMKY PŘEVOD ŘETĚZCE NA ČÍSLO, KNIHOVNA <CTYPE.H>

- Několik funkcí (a hlavičkových souborů), které se mohou hodit
- Knihovna `#include <stdlib.h>`
- Funkce `int atoi(const char* str);`
 - Převádí řetězce na čísla, vrací získané číslo
 - POZOR: je nutné funkci předat opravdu číslo, ne nesmysl

```
int cislo = atoi("123abc");  
printf("prevedene cislo: %d", cislo);  
// prevedene cislo: 123
```

- Pokud předáte opravdu nesmysl, vrátí 0
- Bezpečnější varianta: `strtol()`

- Funkce `long int strtol (const char* str, char** endptr, int base);`
 - `str` – řetězec k převodu
 - `endptr` – ukazatel na poslední znak, který zbyl po konverzi
 - `if(*endptr != '\0') { // chyba }`
 - `base` – základ, v tomto případě desítková soustava (tedy 10)

Funkce vrací číslo > 0 nebo 0

- Zkontroluje, zda je znak **c** bílý znak

```
int isspace(int c);
```

- Zkontroluje, zda je znak **c** číslice

```
int isdigit(int c);
```

- Zkontroluje, zda je znak **c** písmeno

```
int isalpha(int c);
```

- Více informací o **ctype.h**
 - `man ctype.h`
 - Internet

DOMÁCÍ ÚKOLY

Implementuje funkci, která ověří, že **číslo x patří do intervalu $[a,b]$** .

Funkce vrátí

- 1 pokud se číslo x nachází v intervalu $[a,b]$
- 0 jinak

Implementujte funkci, která **ověří, zda je číslo dělitelné jiným číslem beze zbytku**. Pokud ano, funkce vrátí 1, jinak 0.

Nápověda: k řešení použijte operátor modulo (zbytek po celočíselném dělení), v C %

$$10 \% 5 = 0$$

$$11 \% 6 = 5$$

atp...

Implementujte funkci, která **vypočítá absolutní hodnotu** čísla typu **double** a vypočítanou absolutní hodnotu vrátí.

Implementujte funkci, která vybere **maximální hodnotu** z pole typu `int`, které má `n` členů. Funkce maximální hodnotu vrátí.

Napište funkci, která vypočítá **n-tý prvek Fibonacciho posloupnosti**.

Fibonacciho posloupnost je definována:

$$f(0) = 0$$

$$f(1) = 1$$

$$f(n) = f(n-1) + f(n-2)$$

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, atd.

- Napište funkci, která **překopíruje obsah** jednoho pole do druhého.
- Funkce bude přijímat **tři parametry**:
 - Zdrojové pole
 - Cílové pole
 - Počet prvků zdrojového pole

- Napište funkci, která **vrací délku zadaného řetězce**.
- Řetězec předejte odkazem

Děkuji Vám za pozornost!

- Vyhledání znaku `c` v řetězci `s`, vrací ukazatel na vyhledaný znak, jinak NULL

```
char* strchr(char* s, int c);
```

- Kopie řetězce `src` do řetězce `dst`, vrací ukazatel na `dst`

```
char* strcpy(char* dst, char* src);
```

- Spojení řetězců `dst` a `src`, výsledek v `dst`

```
char* strcat(char* dst, char* src);
```

- Lexikografické porovnání řetězců `s1` a `s2`

```
int strcmp(char* s1, char* s2);
```