Fakulta elektrotechniky
a informatiky

Programovanie

Prog LS-2022/2023

Cvičenie č. 4

- Čo je smerník
- Smerník ako argument funkcie
- Smerník a polia resp. reťazce

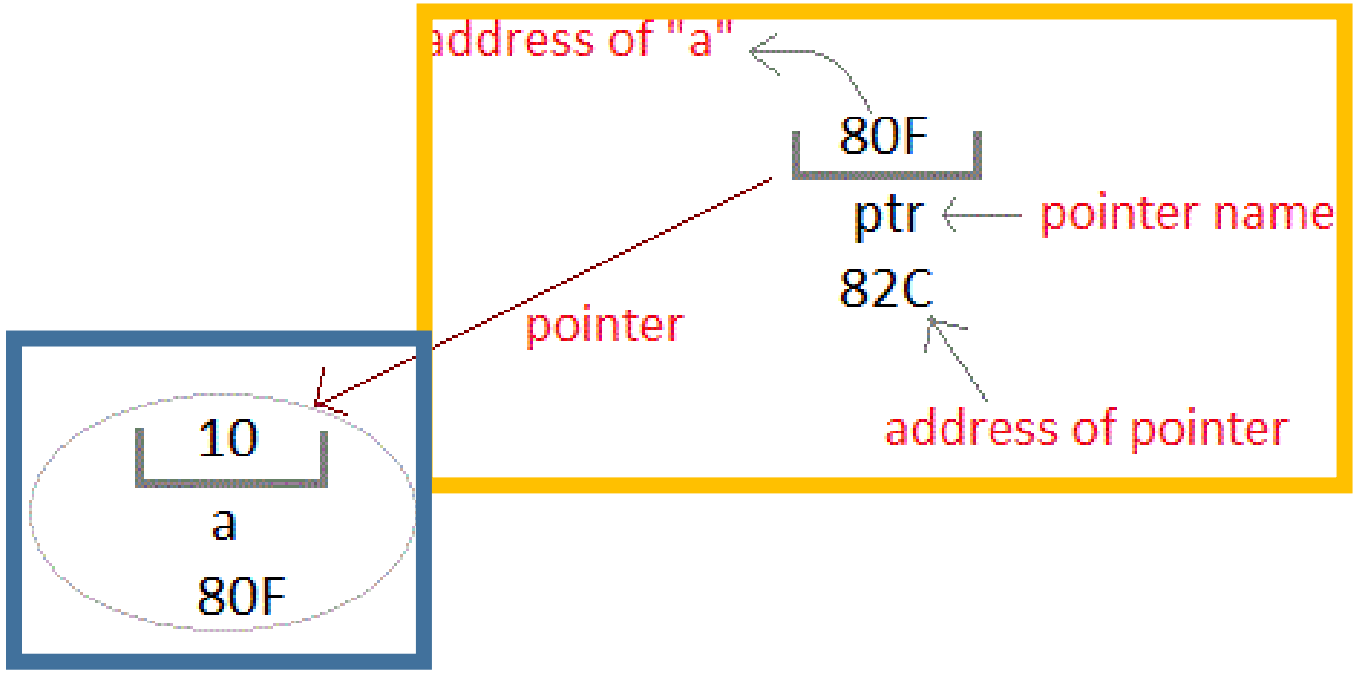
Ing. Ondrej Kováč, PhD.

Čo je to smerník?

- **Smerník je údajový typ, ktorý obsahuje adresu, ktorá ukazujeme na miesto v pamäti.**
- Ak deklarujeme premennú, pred ktorú dáme znak *****, dáme prekladaču najavo, že v tejto premennej budú uložené adresy na premenné.
- Ak pred premennú dáme znak **&** získame jej adresu

```
int a = 10;
```

```
int* ptr = &a;
```



Čo je to smerník?

```c
#include <stdio.h>
int main()
{
  int a = 5;          // Priradenie hodnoty 5 do premennej a.
  int* ptr_a = &a;    // Priradenie adresy prem. a do ptr_a.
                      // (typ smernik)
  int c = *ptr_a;     // Priradenie obsahu do prem c.
                      // Tento obsah sa nachádza na adrese kam
                      // ukazuje smerník ptr_a.
  printf(">> a = %d \n", a);
  printf(">> *ptr_a = %d \n", *ptr_a);
  printf(">> ptr_a = %p \n", ptr_a);
  printf(">> &a = %p \n",  &a);
  printf(">> c = %d \n", c);
  return 0;
}
```

```
ab123cd@zapfei:CV1$ gcc ptr.c -o PTR
ab123cd@zapfei:CV1$ ./PTR
>> a = 5
>> *ptr_a = 5
>> ptr_a = 0x7ffdebd89bf8
>> &a = 0x7ffdebd89bf8
>> c = 5
```

Smerník ako argument funkcie

- Výsledok vykonania ľubovoľnej funkcie môže byť vrátený pomocou jej návratovej hodnoty.
- K tomuto účelu slúži príkaz **retrun**.
- Mnoho funkcií ale je typu **void**, teda bez návratovej hodnoty a stále dokážu "vrátiť" výsledok, napr. funkcia **scanf**. načíta znak alebo reťazec do určenej premennej.

- *Majme jednoduchú funkciu, ktorá vynásobí číslo, ktoré jej bude zaslané hodnotou "-1". Teda v prípade kladného čísla dostaneme číslo záporné a naopak v prípade záporného čísla zasa vo výsledku dostaneme číslo kladné.*

```
int a = 5;
int b = invert(a);
printf(">> b = -1 x a = -1 x %d = %d\n", a, b);
```

```
ab123cd@zapfei:CV1$ gcc invert.c -o INVERT
ab123cd@zapfei:CV1$ ./INVERT

>> b = -1 x a = -1 x 5 = -5
```

Smerník ako argument funkcie

- *Majme jednoduchú funkciu, ktorá vynásobí číslo, ktoré jej bude zaslané hodnotou "-1". Teda v prípade kladného čísla dostaneme číslo záporné a naopak v prípade záporného čísla zasa vo výsledku dostaneme číslo kladné.*

- Riešenie tohto príkladu môže byť dvojaké. Asi sa zhodneme na tom, že požadovaná funkcia by mohla vyzerať takto:

```
int invert(int cislo)
{
  return -1*cislo;
}
```

Smerník ako argument funkcie

- *Majme jednoduchú funkciu, ktorá vynásobí číslo, ktoré jej bude zaslané hodnotou "-1". Teda v prípade kladného čísla dostaneme číslo záporné a naopak v prípade záporného čísla zasa vo výsledku dostaneme číslo kladné.*

- Riešenie tohto príkladu môže byť dvojaké. Asi sa zhodneme na tom, že požadovaná funkcia by mohla vyzerať takto:

```
int invert(int cislo)
{
  return -1*cislo;
}
```

- Jedná sa teda o „klasickú“ funkciu s návratovou hodnotou typu **int**.

- Túto funkciu však môžeme navrhnúť aj tak, aby bola bez

Smerník ako argument funkcie

- Funkciu „invert“ však môžeme navrhnúť aj tak aby bola bez návratovej hodnoty.

```
void invert(int* cislo)
{
   *cislo = *cislo * (-1);
}
```

Smerník ako argument funkcie

- Funkciu „invert“ však môžeme navrhnúť aj tak aby bola bez návratovej hodnoty.

```
void invert(int* cislo)
{
   *cislo = *cislo * (-1);
}
```

- Môžeme si všimnúť, že funkcia nemá návratovú hodnotu. V jej argumente je tentokrát smerník, ktorý ukazuje na miesto v pamäti kde je uložená premenná, ktorú chceme invertovať.

- Funkciu **main** je v tomto prípade potrebné trochu upraviť.

Smerník ako argument funkcie

- Funkciu „invert“ však môžeme navrhnúť aj tak aby bola bez návratovej hodnoty.

```
void invert(int* cislo)
{
   *cislo = *cislo * (-1);
}
```

- Funkciu **main** je v tomto prípade potrebné trochu upraviť.

```
int main()
{
  int a = 5;
  int b = a;       // Do b je skopírovaná hodnota z premennej a
  invert(&b);   // Funkcii invert sa pošle iba adresa premennej b
  printf(" >> b = -1 x a = -1 x %d = %d\n", a, b);
  return 0;
}
```

Pozor na nedorozumenia!

VOID - ZERO (0) -NULL

- Častokrát sa môžeme stretnúť s označením NULL, Void alebo nula - 0. Je potrebné mať na pamäti, že

Non-zero value

0

null

undefined

Pozor na nedorozumenia!

VOID - ZERO (0) -NULL

- Častokrát sa môžeme stretnúť s označením NULL, Void alebo nula - 0. Je potrebné mať na pamäti, že:

  - V jazyku C **void** znamená absenciu dátového typu. **Aj smerníky môžu byť typu void.** Taký smerník môže **obsahovať adresu premennej ľubovoľného typu**. Určite ale obsahuje adresu, na ktorej sa môžu nachádzať dáta! Môžeme ho použiť vtedy ak do neho počas behu programu, v rôznom čase, potrebujeme uložiť referenciu na rôzne údajové typy. **Smerník typu void má v jazyku C svoj význam, ale pre bežné potreby ho používať nebudeme.**

Pozor na nedorozumenia!

VOID - ZERO (0) -NULL

- Častokrát sa môžeme stretnúť s označením NULL, Void alebo nula - 0. Je potrebné mať na pamäti, že:
  - **Smerník, ktorý obsahuje hodnotu NULL neukazuje nikam do pamäte**. Občas budeme tento smerník používať a overovať ako návratovú hodnotu funkcie. Napríklad vtedy ak sa nepodarí otvoriť súbor pre zápis alebo čítanie. V takom prípade vhodným testom predídeme chybe programu. **Tiež je možné smerník inicializovať hodnotou NULL, teda vytvorená premenná je definovaná, ale neukazuje nikam do pamäte.**

Pozor na nedorozumenia!

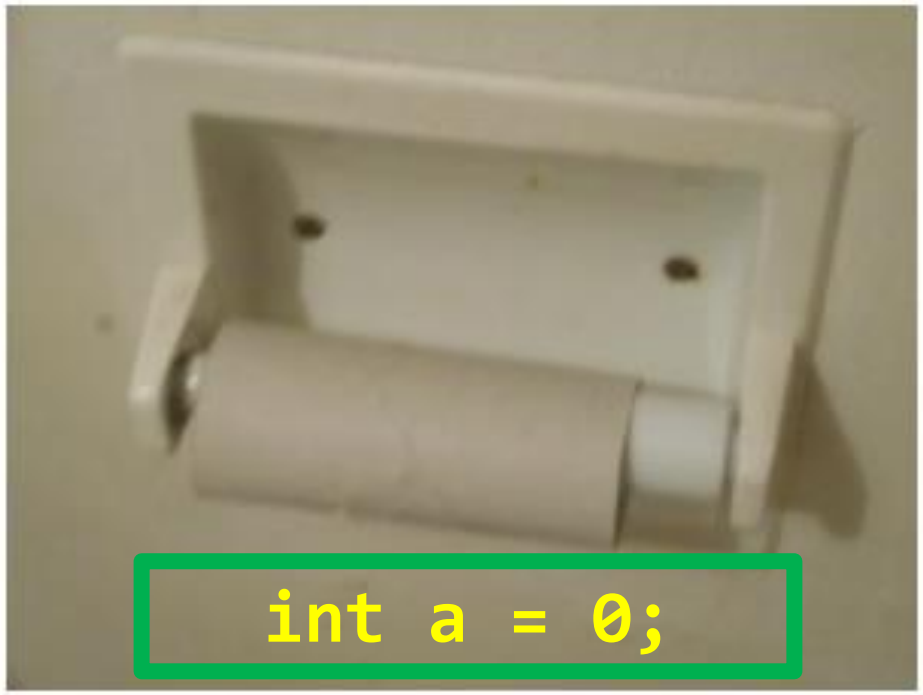
VOID - ZERO (0) -NULL

- Častokrát sa môžeme stretnúť s označením NULL, Void alebo nula - 0. Je potrebné mať na pamäti, že:
  - **Smerník, ktorý obsahuje hodnotu NULL neukazuje nikam do pamäte**. Občas budeme tento smerník používať a overovať ako návratovú hodnotu funkcie. Napríklad vtedy ak sa nepodarí otvoriť súbor pre zápis alebo čítanie. V takom prípade vhodným testom predídeme chybe programu. **Tiež je možné smerník inicializovať hodnotou NULL, teda vytvorená premenná je definovaná, ale neukazuje nikam do pamäte.**
  - Pri smerníkoch je možné inicializovať aj hodnotou **0** to sa tiež považuje za **NULL**. (ale pozor!)

Pozor na nedorozumenia!

VOID - ZERO (0) -NULL

- Častokrát sa môžeme stretnúť s označením NULL, Void alebo nula - 0. Je potrebné mať na pamäti, že:
  - Pri smerníkoch je možné inicializovať aj hodnotou **0** to sa tiež považuje za **NULL**. (ale pozor!)
  - Je potrebné mať na pamäti, že pri normálnej premennej napr. **int 0** znamená **hodnotu 0**.



Pozor na nedorozumenia!

Nedefinovaný stav

- (Ne)poznáme ešte jeden stav ...
  - Ak premennú deklarujeme ale neinicializujeme, nevieme čo v nej bude. Bude v nej náhodná hodnota, ktorú v pamäti zanechal iný program alebo iná funkcia nášho programu.

```
int a; //netušíme čo je uložené v a
       radšej použijeme
       int a = 0;
```

Smerník a 1R polia resp. reťazce

- V prípade polí (teda aj reťazcov) sa za **adresu považuje samotný názov tohto poľa.**
- **V prípade klasických 1R polí je potrebné funkcii spoločne s adresou zaslať aj veľkosť poľa**. To je dané tým, že obyčajné 1R pole na svojom konci nemá ukončovací znak - terminátor.

```
void funkcia_nad_1R_polom(int* pole, int dlzka);
```

- **Reťazec je 1R pole, ktoré je zakončené terminátorom '\0‘.**
- Preto stačí, ak sa funkcii v argumente posiela iba **názov poľa** a už nie je potrebné zasielať aj jeho dĺžku.

Smerník a 1R polia resp. reťazce

- V prípade polí (teda aj reťazcov) sa za adresu považuje samotný názov tohto poľa.
- Reťazec je 1R pole, ktoré je zakončené terminátorom '\0‘.
- Preto stačí, ak sa funkcii v argumente posiela iba názov poľa a už nie je potrebné zasielať aj jeho dĺžku.
- V bloku kódu je zobrazený jednoduchý kód na zámenu veľkých písmen za malé a naopak (medzery a iné znaky okrem písmen nie sú ošetrené).

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
void change(char* retazec){
  int d = strlen(retazec);
  for (int i = 0; i<d; i++)
  {
   if (islower(retazec[i]))
        retazec[i] = retazec[i] + ('A'-'a');
   else
        retazec[i] = retazec[i] - ('A'-'a');
  }
}
int main(){
  char Retazec[100] = "AhoJSveT";
  printf("\n>> Vstupny retazec --> %s\n",Retazec);
  change(Retazec);
  printf("\n>> Vystupny retazec --> %s\n",Retazec);
}
```

Smerník a 1R polia resp. reťazce

- V bloku kódu je zobrazený jednoduchý kód na zámenu veľkých písmen za malé a naopak (medzery a iné znaky okrem písmen nie sú ošetrené).

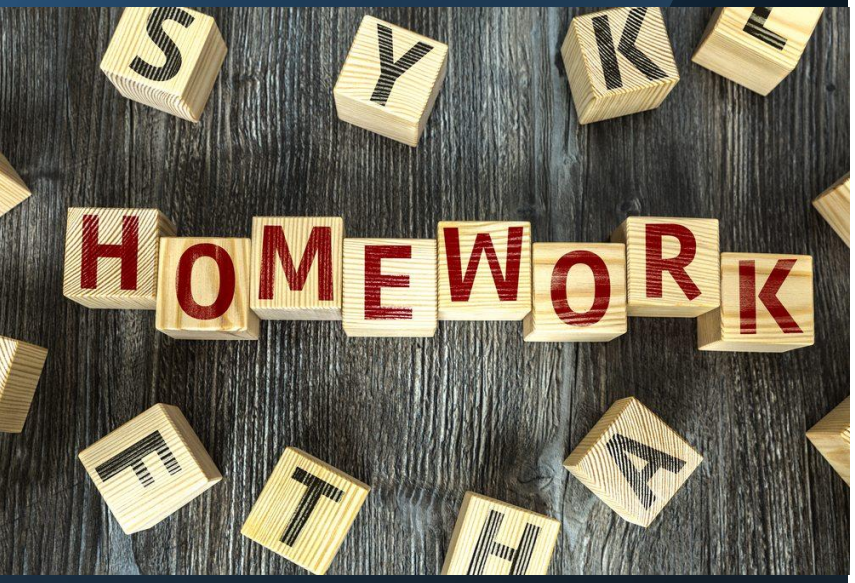
```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
void change(char* retazec){
  int d = strlen(retazec);
  for (int i = 0; i<d; i++)
  {
   if (islower(retazec[i]))
        retazec[i] = retazec[i] + ('A'-'a');
   else
        retazec[i] = retazec[i] - ('A'-'a');
  }
}
int main(){
  char Retazec[100] = "AhoJSveT";
  printf("\n>> Vstupny retazec --> %s\n",Retazec);
  change(Retazec);
  printf("\n>> Vystupny retazec --> %s\n",Retazec);
}
```

```
ab123cd@zapfei:CV1$ ./RETAZEC

>> Vstupny retazec --> AhoJSveT

>> Vystupny retazec --> aHOjsVEt
```

Hlasovanie na odreagovanie pred samostatnou prácou

- Ktorá situácia by Vás nahnevala viac?

Samostatná práca



Úloha 1.

Napíšte program (aritmetika.c) pre výpočet základných aritmetických operácií "+", "-", "*", "/"

- Operácia bude zadaná ako argument príkazového riadku!
- Vstupné čísla a, b budú vyžiadané od používateľa programom. (printf, scanf)
- Všetky funkcie okrem main budú typu **void**.
- Kostra kódu je zobrazená v bloku vpravo.
  - Odporúčaný postup je taký, že
    - najprv si sfunkčnite funkcie,
    - potom si spravíte zadávanie čísel,
    -nakoniec vyriešite zadávanie operácie z príkazového riadku!
- Úloha je postavená tak, že musíte doplniť kostru a nesmiete nič čo je v nej uvedené zmeniť alebo vymazať!

Úloha 2.

Vytvorte vlastnú funkciu pre porovnávanie dvoch reťazcov.

- Reťazce budú funkcii odovzdané cez smerník (teda adresa ako to je v príklade vyššie).
- Funkcia vráti hodnotu **1 ak budú reťazce zhodné** a hodnotu **0 v opačnom prípade**.
- Funkcia bude mať názov **StrCmp**.
- Kostra programu je vpravo.
- Okrem **stdio.h** sa zakazuje použitie iných .h súborov