

Programování v jazyce C pro chemiky (C2160)

6. Funkce, struktury

Funkce

- Program v jazyce C je strukturován do funkcí
- Příkazy jazyka C nelze nikdy uvádět mimo funkce
- Program začíná funkcí *main()*, z ní lze volat další funkce, atd.
- Definice funkce obsahuje:
 - typ návratové hodnoty (*int*, *float*, *char* ...); pokud funkce žádnou hodnotu nevrací, uvedeme klíčové slovo **void**
 - jméno funkce (bez mezer, stejně jako názvy proměnných)
 - definice proměnných předávaných funkci (tzv. parametry) uvedené v závorkách; pokud funkce žádné parametry nepřijímá, použijeme opět **void**
 - kód funkce (tj. příkazy) ve složených závorkách

```
int myfunction(int n, float a)
{
    // Tady se uvedou prikazy funkce
}

// Nasledujici funkce nevraci zadnou hodnotu a neprijima zadne parametry
void otherfunction(void)
{
    // Kod funkce
}
```

Návratová hodnota funkce, příkaz return

- Příkaz **return** slouží k okamžitému opuštění funkce a zároveň ke specifikaci návratové hodnoty funkce
- Navracená hodnota se uvádí bezprostředně za příkazem **return** (může to být jakýkoli výraz, konstanta, jméno proměnné, ...)
- Voláme-li funkci, která vrací hodnotu, můžeme vrácenou hodnotu přiřadit do vhodné proměnné nebo ji přímo použít jako součást aritmetického výrazu, podmínky, cyklu a pod.
- Příkaz **return** lze použít kdekoliv v těle funkce

```
int square(int n)
{
    return n*n;    // Funkce vrací druhou mocninu n
}

int main(void)
{
    int a = 0, b = 9, c = 12;
    a = square(b);
    printf("Druha mocnina je: %i\n", a);
    if (square(c) > 20)
        printf("Druha mocnina z %i je vetsi nez 20\n", c);
    return 0;
}
```

Návratová hodnota funkce main()

- Funkce main() vrací hodnotu typu int, tato hodnota je předána operačnímu systému po skončení programu (v shellu můžeme tuto hodnotu získat výrazem \$? po vykonání příkazu)
- V Unixu je konvence, že při úspěšném průběhu vrací program hodnotu 0, při neúspěchu hodnotu různou od 0

```
int main(void)
{
    FILE *f = NULL;
    f = fopen("test1.txt", "r");
    if (f == NULL)
    {
        printf("Cannot open file!\n");
        return 1;
    }
    // Zde muze byt nejaky dalsi kod

    return 0;
}
```

Předání hodnot do funkce

- Typ a jména proměnných předávaných funkci uvádíme při definici funkce v závorce za jménem funkce, proměnné oddělujeme čárkami
- V jazyce C jsou proměnné **předávány hodnotou**, tj. předávané hodnoty (**argumenty**) jsou zkopírovány do nových proměnných (**parametrů**). Jakákoli změna jejich hodnoty ve volané funkci neovlivní hodnoty proměnných volající funkce (POZOR: **výjimkou jsou jakákoli pole**, tedy i řetězce!)

```
// Vytvoří se promenne a a b, do kterých se zkopírují předané hodnoty
int sum(int a, int b)
{
    int c = 0;
    c = a + b;
    // Zmena hodnoty promenne a nijak neovlivni hodnotu
    // promenne i, která byla předána jako argument
    a = 100;
    return c;
}

int main(void)
{
    int i = 5, j = 7, k = 0;
    k = sum(i, j);
    // Hodnoty promenných i a j zůstanou vždy nezmeneny, i pokud jsme
    // přiřazovali jakékoli hodnoty promenným a a b ve funkci sum()
}
```

Předání pole do funkce

- Pole (a tedy i řetězce) nejsou do funkce předávána hodnotou, tj. nedochází k vytvoření kopie daného pole, ale předané pole je pouze přístupné pod novým jménem proměnné (tzv. předávání odkazem). Jakákoli změna hodnoty některého z prvků pole ve volané funkci změní hodnotu i pro volající funkci.

```
#define MAX_VAL 5

void myfunction(int v[MAX_VAL])
{
    v[0] = 10;    // Zmeni hodnoty puvodniho pole values[]
    v[1] = 20;
}

int main(void)
{
    int values[MAX_VAL] = {1, 2, 3, 4, 5};
    myfunction(values);
    // Nasledujici prikaz vypise hodnoty 10 a 20
    // ktere byly do pole values[] prirazeny ve funkci myfunction()
    printf("Prvni dve hodnoty pole: %i, %i", values[0], values[1]);
    return 0;
}
```

Předání řetězce funkci

- Řetězce jsou také pole, proto také nejsou předávány hodnotou
- Pokud velikost předávaného pole neznáme (např. u řetězců inicializovaných řetězcovou konstantou), neuvádíme v hranatých závorkách žádnou velikost, abychom kompilátor nemátli

```
#define STRING_SIZE 100

void print_strings(char s1[STRING_SIZE], char s2[])
{
    printf("Řetězce předané funkci: %s, %s\n", s1, s2);
}

int main(void)
{
    char str[STRING_SIZE] = "";
    char file_name[] = "/var/logfile";

    scanf("%s", str);

    print_strings(str, file_name);

    return 0;
}
```

Pořadí definice funkcí, deklarace funkce

- V okamžiku kdy je určitá funkce volána, musí mít překladač základní informace o této funkci (její tzv. prototyp, tedy typ návratové hodnoty a typy parametrů)
- V praxi tedy definujeme nejdříve funkce, které budou volány, teprve potom funkce z nichž jsou volány; funkce `main()` bude tedy poslední
- Alternativním přístupem je použití **deklarace funkce**, která je tvořena hlavičkou funkce zakončenou středníkem; uvádí se na začátku programu; samotná definice funkce se může nacházet kdekoliv dále

```
int square(int n);    // Deklarace funkce, musí být zakončena středníkem
```

```
int main(void)
{
    int a = 0, b = 9;
    // Protože funkce square() již byla deklarována na začátku programu,
    // překladač ji zna a můžeme ji tedy zavolat
    // i když definována bude až později
    a = square(b);
    return a;
}
```

```
int square(int n)    // Definice funkce
{
    return n*n;
}
```


Globální a lokální proměnné

- Proměnné definované uvnitř funkce se nazývají **lokální** a mají platnost pouze v této funkci
- **Globální proměnné** se definují mimo definici funkce, zpravidla na začátku programu před definicí první funkce
- Globální proměnné jsou dostupné ve všech funkcích. Změna jejich hodnoty v jedné funkci se tedy projeví i ve všech ostatních.
- Globální proměnné jsou automaticky inicializovány hodnotou 0

```
// Globalni promenna:  
char filename[] = "/home/uzivatel/tmp/data.dat";  
  
void read_file(void)  
{  
    FILE *f = NULL; // Lokalni promenna  
    f = fopen(filename, "r");  
    // Zde by byl dalsi kod pro nacteni dat ze souboru  
}  
  
int main(void)  
{  
    read_file();  
    return 0;  
}
```

Struktury

- Struktury slouží k seskupení několika proměnných různých typů
- Strukturu definujeme následovně:

```
typedef struct
```

```
{
```

```
    // definice prvků (proměnných) struktury
```

```
} JMENO_STRUKTURY;
```

- Jméno struktury používáme k definici proměnných podobně, jako kdyby se jednalo o jméno základního typu (int, float,)
- Hodnoty jednotlivých proměnných struktury můžeme inicializovat pomocí složených závorek

```
// Definujeme strukturu vector_2d pro 2-dimenzionalni vektor
```

```
typedef struct
```

```
{
```

```
    float x;
```

```
    float y;
```

```
} vector_2d;
```

```
// Definice promenne vector a inicializace souradnic x a y
```

```
// hodnotmi 1.1 a 2.5
```

```
vector_2d vector = {1.1, 2.5};
```

Struktury

- K jednotlivým položkám (proměnným) struktury přistupujeme přes tečku:

jméno_struktury.jméno_proměnné

```
// Ukazka scitani dvou vektoru s vyuzitim struktur
```

```
typedef struct  
{  
    float x;  
    float y;  
} VECTOR_2D;
```

```
VECTOR_2D v1 = {0.0, 0.0};  
VECTOR_2D v2 = {0.0, 0.0};  
VECTOR_2D vsum = {0.0, 0.0};
```

```
printf("Zadejte souradnice prvnioho vektoru");  
scanf("%f %f", &v1.x, &v1.y);  
printf("Zadejte souradnice druhého vektoru");  
scanf("%f %f", &v2.x, &v2.y);
```

```
vsum.x = v1.x + v2.x;  
vsum.y = v1.y + v2.y;
```

```
printf("Vysledny vektor: %f, %f\n", vsum.x, vsum.y);
```

Struktury

- Struktury mohou obsahovat různé typy proměnných, včetně např. řetězcových proměnných nebo polí
- Pro načítání a výpis hodnot platí stejná pravidla jako pro běžné proměnné, tj. u funkcí `scanf()` a `fscanf()` uvádíme před názvem proměnné znak **&** s výjimkou řetězcových proměnných

```
// Ukazka slozitejsi struktury obsahujici retezcovou promennou
// a ukazka nacisti hodnot ze vstupu a vypisu na vystup

typedef struct
{
    char str[5];
    float energy_exp;
    float energy_calc;
} ENERGY_ITEM;

// Retezcovou promennou inicializujeme "", ostatni promenne 0.0
ENERGY_ITEM item = {"", 0.0, 0.0};

// Nacteme hodnoty ze vstupu, pred nazvy promennych davame & krome
// retezcovych promennych
scanf("%4s %f %f", item.str, &item.energy_exp, &item.energy_calc);

// Vypiseme hodnoty
printf("Hodnoty: %s, %f, %f\n", item.str, item.energy1, item.energy2);
```

Pole struktur

- Je možné vytvářet pole struktur, postup je podobný jako při vytváření polí základních typů (int, float, ...)
- Pole struktur obvykle inicializujeme tak, že inicializujeme první hodnotu první položky a překladač pak inicializuje všechny ostatní hodnotu nulou. U globálních proměnných není inicializace nutná, protože jsou vždy automaticky inicializovány nulou.

```
#define MAX_ITEMS 50
```

```
typedef struct  
{  
    char str[5];  
    float energy_exp;  
    float energy_calc;  
} ENERGY_ITEM;
```

```
ENERGY_ITEM items[MAX_ITEMS] = {{0}}; // Definujeme pole struktur
```

Pole struktur - načítání dat ze souboru

- Pole struktur se často využívají pro načítání dat ze souboru

```
#define MAX_ITEMS 50
typedef struct
{
    char str[5];
    float energy_exp;
    float energy_calc;
} ENERGY_ITEM;

ENERGY_ITEM items[MAX_ITEMS] = {{0}}; // Definujeme pole struktur
int count = 0; // Pocet nactenych polozek v poli items[]

// Nacteni ze souboru:
FILE *f = NULL;
// Zde musi byt kod pro otevreni souboru pro cteni (s identif. f)

while (fscanf(f, "%4s %f %f", items[count].str,
              &items[count].energy_exp,
              &items[count].energy_calc) == 3) {
    // Je-li count >= MAX_ITEMS vypiseme hlasku a vyskocime z cyklu
    count++;
}
// Zde se soubor uzavre
```

Dodržujte následující pravidla

- Při programování postupujte po malých krocích, pak vždy program přeložte a otestujte a dále pokračujte až po odstranění chyb. Např. v úloze 2 nejdříve program předělejte tak, aby používal struktury (a zkompilujte a otestujte). Teprve potom ho rozdělte do funkcí.
- Dodržujte následující uspořádání programu:
 - Vložení hlavičkových souborů pomocí `#include`
 - Definice všech symbolických konstant pomocí `#define`
 - Definice všech struktur (pomocí `typedef struct`)
 - Definice všech globálních proměnných
 - Deklarace funkcí (pokud je používáte)
 - Kód funkcí ve vhodném pořadí (použijeme-li deklarace, na pořadí nezáleží)
- Uvnitř funkce inicializujte proměnné při jejich definici vhodnou hodnotou.
- Jako globální definujte pouze ty proměnné, jejichž hodnoty budou používány ve více než jedné funkci. Všechny ostatní proměnné definujte jako lokální (např. řídicí proměnné cyklu, identifikátory souborů atd.).
- Velikost pole znaků pro řetězce zvolte tak aby se do něj vešly všechny načítané znaky + zakončovací znak `\0`
- Při načítání řetězce funkcí `fscanf()` vždy specifikujte šířku tak, aby nedošlo k překročení velikosti pole.
- Pro pojmenování funkcí nepoužívejte jména `read()` a `write()`, protože tyto funkce jsou již použity ve standardní knihovně a došlo by ke kolizi názvů.

Úlohy - část 1

1. Vytvořte program, který od uživatele vyžádá souřadnice dvou trojdimenzionálních vektorů a spočítá jejich vektorový součin. Výsledek vypíše na obrazovku. Pro reprezentaci vektorů použijte struktury.

nepovinná, 1 bod

Úlohy - část 2

2. Upravte program z předchozího cvičení (úloha 2 ze cvičení 5) tak, aby data načítal do pole struktur. Struktura bude obsahovat PDB kód komplexu a dvě hodnoty energií. Do výstupního souboru bude zapsán PDB kód a hodnoty energií (opět s obráceným pořadím řádků).

Program navíc rozdělte do následujících samostatných funkcí:

- ♦ funkce pro otevření vstupního souboru a načtení hodnot
- ♦ funkce pro otevření výstupního souboru a zápisu hodnot energií
- ♦ funkce `main()`

Řetězcové proměnné obsahující jména souborů definujte ve funkci `main()` jako lokální proměnné a předejte je příslušným funkcím jako argumenty. Proměnné inicializujte hodnotami "data1.dat" a "cv6u2.dat".

Funkce pro čtení a zápis souboru budou vracet hodnotu typu `int`. V případě úspěchu vrátí 0 a při neúspěchu 1. Ve funkci `main()` pak bude testována návratová hodnota těchto funkcí a v případě, že vrátí 1, bude celý program ukončen a vrátí hodnotu 1.

2 body

Úlohy - část 3

3. Vytvořte program, který načte zjednodušený PDB soubor `crambin_simple.pdb` (nacházející se v adresáři `/home/tootea/C2160/data`) do pole vhodných struktur. Tento soubor obsahuje informace o molekule proteinu *crambin*. Potom program zapíše data do jiného souboru tak, aby formát dat byl podobný načítanému PDB souboru (nemusí být přesně stejně formátovaný). Použijte podobný přístup jako v úloze 2 (globální pole struktur, oddělené funkce pro čtení a zápis).

3 body

ATOM	748	CG	TYR	44	11.895	12.742	14.274	C
------	-----	----	-----	----	--------	--------	--------	---

Každý řádek zjednodušeného PDB souboru se skládá z následujících položek:

1. Název záznamu – řetězec max. 6 znaků (zde **ATOM**)
2. Číslo atomu – celé kladné číslo (zde **748**)
3. Název atomu – řetězec max. 4 znaky (zde **CG**)
4. Zkratka názvu rezidua (tj. aminokyseliny) – řetězec max. 3 znaky (zde **TYR**)
5. Číslo rezidua – celé kladné číslo (zde **44**)
6. Kartézské souřadnice x, y a z udávající pozici atomu v prostoru v Angstromech – tři desetinná čísla (zde **11.895 12.742 14.274**)
7. Zkratka jména prvku – řetězec max. 2 znaky (zde uhlík **C**)

Úlohy - část 4

4. Do programu z úlohy 3 přidejte funkci, která analyzuje načtené pole struktur a vypíše celkový počet atomů a počet atomů jednotlivých prvků (dle poslední položky v řádku, stačí uvažovat prvky H, C, N, O, S).

nepovinná, 2 body