Základy programovania (IZP)
Vnorené cykly (4. cvičenie)

Ing. Pavol Dubovec

Vysoké Učení Technické v Brně, Fakulta informačních technologií
Božetěchova 1/2. 612 66 Brno- Královo Pole
idubovec@fit.vutbr.cz

Prečo používame funkcie?
Čo to je parameter, argument, návratový typ?

Teoretické opakovanie – funkcie

- Umožňujú rozdelenie programu na menšie, spravovateľné časti. Každá funkcia má názov, návratový typ a zoznam parametrov.
- Deklarácia slúži na zavedenie identifikátora a na poskytnutie opisu jeho typu. Môže to byť typ, objekt alebo funkcia. Deklarácia je prostriedok, pomocou ktorého je kompilátor schopný prijať odkazy na uvedený identifikátor. Končí středníkem (bodkočiarkou).
- Definícia slúži na vytvorenie inštancie alebo implementáciu určeného identifikátora. Ide o informáciu, ktorú linker vyžaduje na prepojenie odkazov na uvedené entity.

```c
// Deklarácia funkcie
int sucet(int a, int b);
// Hlavná funkcia
int main() {
    int vysledok = sucet(5, 3);
    printf("Súčet: %d\n", vysledok);
    return 0;
}
```

```c
// Definícia funkcie
int sucet(int a, int b) {
    return a + b;
}
```

Čo to je parameter, argument, návratový typ?

Teoretické opakovanie – návratový typ a parametre 1/2

- Návratový typ určuje **typ hodnoty**, ktorú **funkcia vráti**. Ak funkcia nevracia hodnotu, použije sa `void`. Určuje, aký **typ dát** môžeme očakávať **po vykonaní funkcie**.
- Parametre sú **premenné**, ktoré funkcia **prijíma**. Môžu byť rôznych typov a musia byť špecifikované **v deklarácii aj definícii**. Parametre umožňujú funkciám pracovať s **rôznymi vstupmi a vracať rôzne výstupy**.
- Parametre **sú platné iba v rámci tela funkcie**. Mimo funkcie **sú nedostupné**.
- Pri **volaní** funkcie sa **hodnoty argumentov** priradia **k parametrom**.
- Argumenty sú hodnoty, ktoré odovzdávame funkcii **pri jej volaní**.

```c
// Deklarácia funkcie, ktorá vypočíta priemer a vypíše výsledok
void vypisPriemer(double x, double y);
// Hlavná funkcia
int main() {
    double a = 5.0;
    double b = 10.0;
    // Volanie funkcie s argumentmi a, b
    vypisPriemer(a, b);

    return 0;
}
```

```c
// Definícia funkcie
void vypisPriemer(double x, double y) {
    // Vypočíta priemer dvoch čísel
    double priemer = (x + y) / 2;
    // Vypíše výsledok
    printf("Priemer z %.2f a %.2f je %.2f\n", x,
    y, priemer);
}
```

- Funkcie v C môžu vracať rôzne typy hodnôt, ako sú int, float, double, char, bool (#include stdbool.h), alebo void, ak nevracajú žiadnu hodnotu.
- Pri odovzdávaní poľa funkcii sa pole a jeho dĺžka odovzdávajú ako dva samostatné parametre. Funkcia nemôže priamo vrátiť pole, pretože pole je v C odovzdávané ako ukazovateľ na prvý prvok.
- Pole je v C reprezentované ako ukazovateľ na prvý prvok. Ak by sme sa pokúsili vrátiť pole z funkcie, vrátili by sme ukazovateľ na lokálnu premennú, ktorá by po skončení funkcie prestala existovať, čo by viedlo k neplatnému ukazovateľu → Namiesto toho môžeme vrátiť ukazovateľ na dynamicky alokované pole (cvičenie 9) alebo použiť ukazovateľ ako argument funkcie, aby sme modifikovali pôvodné pole.

Prečo by sme mali používať zanorené cykly?
Aké problémy môžeme riešiť pomocou zanorených cyklov?

Teoretické opakovanie – zanorené cykly (nested loops)

- Cyklus vnútri iného cyklu. Vonkajší cyklus sa vykoná raz pre každú iteráciu vnútorného cyklu.
- Je možné použiť rôzne typy cyklov: for, while, do-while.
- Každý typ cyklu môže byť zanorený do iného typu cyklu.
- Môžu byť použité pre zložitejšie úlohy, ako je práca s viacrozmernými poľami.
- Pri použití zanorených slučiek je dôležité dbať na efektivitu, aby nedochádzalo k zbytočnému spomaleniu programu.

```c
int main() {
    int matica[2][2][2] = {{{1, 2}, {3, 4}}, {{5, 6}, {7, 8}}};
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            for (int k = 0; k < 2; k++) {
                printf("matica[%d][%d][%d] = %d\n", i, j, k, matica[i][j][k]);
            }
        }
    }
    return 0;
}
```

Kedy by ste použili break, continue v cykle? Môžete uviesť príklad situácie, kde by každý z týchto riadiacich príkazov bol užitočný?

- Príkaz break sa používa na okamžité ukončenie cyklu alebo príkazu switch. Keď sa príkaz break objaví vo vnútri cyklu, cyklus sa ukončí a riadenie sa presunie na ďalší príkaz nasledujúci za cyklom.

```
for (int i = 0; i < 10; i++) {
    if (i == 4) {
        break; // Ukončí cyklus ak i == 4
    }
    printf("%d\n", i);
}
```

- Príkaz continue sa používa na preskočenie aktuálnej iterácie cyklu a pokračovanie v ďalšej iterácii. Keď sa vyskytne príkaz continue, zvyšný kód vo vnútri cyklu pre aktuálnu iteráciu sa preskočí a cyklus pokračuje ďalšou iteráciou.

```
for (int i = 0; i < 10; i++) {
    if (i == 4) {
        continue; // Preskočí iteráciu ak i == 4
    }
    printf("%d\n", i);
}
```

Praktické cvičenie – zadanie na funkcie 1/2

1. **Zadanie:** Napíšte funkciu `is_alpha`, ktorá skontroluje, či je daný znak abecedný (`a-z`, `A-Z`). Použite túto funkciu v programe, ktorý načíta znak od používateľa a vypíše, či je abecedný alebo nie.
2. **Zadanie:** Napíšte funkciu `is_name`, ktorá skontroluje, či reťazec obsahuje iba abecedné znaky. Použite túto funkciu v programe, ktorý načíta reťazec od používateľa a skontroluje, či obsahujú iba abecedné znaky.
3. **Zadanie:** Napíšte program, ktorý načíta meno a priezvisko od používateľa. Skontrolujte, či meno a priezvisko obsahujú iba abecedné znaky pomocou funkcie `is_name`. Ak obsahujú neabecedné znaky, program vypíše chybovú správu a skončí.

Ďalšie informácie:

- Knižnica `<ctype.h>` obsahuje užitočné funkcie na prácu s znakmi.
- Reťazce sú v jazyku C reprezentované ako pole znakov ukončené nulovým znakom (`\0`). Pri práci s reťazcami je dôležité pamätať na tento ukončovací znak.
- Funkcie `scanf` a `printf` sú základné nástroje na načítanie vstupu od používateľa a výpis výstupu v jazyku C. Pri používaní `scanf` je dôležité špecifikovať maximálnu dĺžku vstupu, aby sa predišlo pretečeniu pamäte.

Praktické cvičenie – zadania na funkcie 2/2

1. **Zadanie**: Napíšte funkciu `get_max`, ktorá nájde maximálnu hodnotu v zadanom poli celých čísel. Použite túto funkciu v programe, ktorý načíta pole celých čísel od používateľa a vypíše maximálnu hodnotu.
2. **Zadanie**: Napíšte funkciu `get_sum`, ktorá vypočíta súčet prvkov v zadanom poli celých čísel. Použite túto funkciu v programe, ktorý načíta pole celých čísel od používateľa a vypíše súčet prvkov.
3. **Zadanie**: Napíšte funkciu `smaller_than`, ktorá skontroluje, či všetky hodnoty v prvom poli sú menšie ako hodnoty na zodpovedajúcich indexoch v druhom poli. Použite túto funkciu v programe, ktorý načíta dve polia celých čísel od používateľa a skontroluje, či prvé pole má menšie hodnoty ako druhé pole.

Ďalšie informácie:

- Polia sú v jazyku C reprezentované ako sekvencie prvkov rovnakého typu. Pri práci s poľami je dôležité správne spracovať ich dĺžku a indexy.

Praktické cvičenie – vnorené cykly

1. **Zadanie**: Napíšte funkciu `is_in_array`, ktorá skontroluje, či je hodnota v zadanom poli celých čísel. Použite túto funkciu v programe, ktorý načíta pole celých čísel od používateľa a skontroluje, či obsahuje zadanú hodnotu.
2. **Zadanie:** Napíšte program, ktorý načíta dve polia celých čísel od používateľa. Skontrolujte, koľko hodnôt sa nachádza v oboch poliach bez zohľadnenia duplicitných hodnôt.
3. **Zadanie:** Napíšte program, ktorý načíta dve polia celých čísel od používateľa. Skontrolujte, koľko hodnôt sa nachádza v oboch poliach so zohľadnením duplicitných hodnôt.
4. **Zadanie:** Napíšte program, ktorý načíta dve polia celých čísel od používateľa. Skontrolujte, koľko hodnôt sa nachádza v oboch poliach so zohľadnením duplicitných hodnôt pomocou volania funkcie `is_in_array`.

Ďalšie informácie:

- Vnorené cykly umožňujú iteráciu cez viacrozmerné štruktúry alebo vykonávanie komplexných porovnaní. Každý vnorený cyklus by mal mať svoju vlastnú iteračnú premennú, aby sa predišlo konfliktom

Praktické cvičenie – množiny

1. ***Zadanie**: Napíšte funkciu is_in_set, ktorá skontroluje, či je hodnota v zadanej množine celých čísel.
2. ***Zadanie:** Napíšte funkciu is_set, ktorá skontroluje, či pole predstavuje množinu, t.j. či každá hodnota je jedinečná.
3. ***Zadanie:** Napíšte funkciu is_sorted_set, ktorá skontroluje, či pole je efektívnou reprezentáciou množiny, t.j. či každá hodnota je jedinečná a hodnoty sú zoradené.
4. ***Zadanie:** Napíšte program, ktorý načíta dve množiny celých čísel od používateľa. Skontrolujte, či každá množina obsahuje jedinečné hodnoty pomocou funkcie is_set. Ak množiny obsahujú duplicitné hodnoty, vypíšte chybové hlásenie a ukončí program.
5. ***Zadanie:** Napíšte program, ktorý načíta dve množiny celých čísel od používateľa. Vypočítajte a vypíšte prienik, zjednotenie a súčin množín.

Ďalšie informácie:

- Množiny sú kolekcie jedinečných prvkov. Efektívna reprezentácia množiny znamená, že prvky sú jedinečné a zoradené, čo umožňuje rýchlejšie operácie ako prienik a zjednotenie.