Základy programovania (IZP)
Cyklus, první práce s poli (1. cvičenie)

Ing. Pavol Dubovec

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

- Predmet správy pri písaní email-u: IZP – v čom treba pomôcť
- Ak sa dohodneme na termíne odovzdania prosím dodržiavajte ho. Po termíne sa súbory považujú za neodovzdané.

Ako ukladáme desatinné čísla v jazyku C?

- Desatinné čísla sú uložené v premenných s pohyblivou desatinnou čiarkou. Tento typ premenných nám umožňuje pracovať s číslami, ktoré majú desatinnú časť. V jazyku C používame dva hlavné dátové typy pre desatinné čísla
  - float – presnosť približne 7-8 číslic.
  - double – presnosť približne 15-16 číslic.
- Vlastnosti desatinných čísel
  - Zápis desatinných čísel do pamäte počítača má obmedzenú presnosť.
  - V zdrojovom kóde sa desatinné čísla vždy označujú desatinnou bodkou (napr. 3,14).

	Definícia premennej	Značka	Vstup	Výstup
float	`float f = 1;`	%f	`scanf("%f", &f);`	`printf("Hodnota float: %f\n", f);`
double	`double d = 1.0;`	%lf	`scanf("%lf", &d);`	`printf("Hodnota double: %f\n", d);`
int	`int i = 1.0;`	%d	`scanf("%d", &i);`	`printf("Hodnota int: %d\n", i);`

Aké funkcie nad desatinnými číslami poznáte?

Ako formátujeme výstup na zvolený počet desatinných miest?

- Ak chcete používať matematické funkcie, musíte pripojiť hlavičkový súbor
  - `#include <math.h>`
- Základní matematické funkce
  - `sqrt(x)`: Vracia druhú odmocninu z x.
  - `pow(x, y)`: Vracia x rozšírené na y.
  - `sin(x)`, `cos(x)`, `tan(x)`: Goniometrické funkcie, kde x je uhol v radiánoch.
  - `fabs(x)`: Vracia absolútnu hodnotu x.
- Na formátovanie výstupu použite príkaz printf so špecifikátorom formátu
  - `printf("%.2f", hodnota); // Výstup s 2 desatinnými miestami`
- Príklad:

```
double x = 9.0;
double y = 2.0;
double result;
result = sqrt(x); // Výpočet druhej odmocniny
printf("sqrt(%.1f) = %.2f\n", x, result);
```

Aké problémy pri práci s desatinnými číslami v jazyku C poznáte?

- Obmedzená presnosť
  - desatinné čísla sú reprezentované v binárnom formáte, čo môže viesť k chybám pri zaokrúhľovaní.
  - Napríklad číslo 0.1 nemá presnú binárnu reprezentáciu, čo môže viesť k nepresnostiam pri výpočtoch.
- Chyby pri zaokrúhľovaní
  - pri vykonávaní aritmetických operácií môže dôjsť k chybám pri zaokrúhľovaní.
  - Príklad: `float a = 0.1f; float b = 0.2f; float c = a + b;`
  - `// Očakávaný výsledok: 0.3, skutočný výsledok: 0.300000012`
- Problémy s porovnávaním
  - Priame porovnávanie desatinných čísel môže byť nespoľahlivé z dôvodu chýb pri zaokrúhľovaní.
  - Príklad:

```
float x = 0,1f;
if (x == 0.1f) { printf(„Zodpovedá\n“);}
else { printf(„Nezodpovedá“);
// Tento výsledok je pravdepodobnejší}
```

- Pretečenie a podtečenie
  - Desatinné čísla môžu pretekať alebo podtekať, ak sú príliš veľké alebo príliš malé.
  - Príklad:

```
float large = 1e38f * 10.0f; // Overflow
float small = 1e-38f / 10.0f; // Underflow
```

Čo to je cyklus?
Aké typy cyklov poznáme?

- V jazyku C sa cyklus používa na opakovanie bloku kódu viackrát, až kým nie je splnená špecifikovaná podmienka. Poznáme tri hlavné typy cyklov:
  - For cyklus – je najlepší, keď je počet iterácií známy.
  - While cyklus – je najlepší, keď počet iterácií nie je známy.
  - Do-While cyklus - Zaručuje, že cyklus sa vykoná vždy aspoň raz.
- For cyklus - For cyklus sa používa, keď je počet iterácií vopred známy.

```
#include <stdio.h>
int main() {
    for (int i = 0; i < 5; i++) {
        printf("Iterácia %d\n", i);
    }
    return 0;
}
```

Teda:

- **Inicializácia:** int i = 0 nastaví počiatočnú hodnotu.
- **Podmienka:** i < 5 kontroluje, či má cyklus pokračovať.
- **Inkrementácia/Dekrementácia:** i++ aktualizuje premennú cyklu.

- While cyklus – While cyklus sa používa, keď počet iterácií nie je vopred známy. Pokračuje, pokiaľ je podmienka pravdivá.

```
#include <stdio.h>
int main() {
    int i = 0;
    while (i < 5) {
        printf("Iterácia %d\n", i);
        i++;
    }
    return 0;
}
```

Teda:

- **Podmienka:** i < 5 sa kontroluje pred každou iteráciou.
- **Kód vo vnútri cyklu sa vykonáva, pokiaľ je podmienka pravdivá.**

- Cyklus do-while – cyklus do-while je podobný while cyklu, ale zaručuje, že kód vo vnútri cyklu sa vykoná aspoň raz.

```
#include <stdio.h>
int main() {
    int i = 0;
    do {
        printf("Iterácia %d\n", i);
        i++;
    } while (i < 5);
    return 0;
}
```

Teda:

- **Kód vo vnútri cyklu sa vykoná raz pred kontrolou podmienky.**
- **Podmienka**: i < 5 sa kontroluje po každej iterácii.

Čo to je statické pole?

Ako statické pole vytvárame?

Pole – vytvorenie, inicializácia a indexácia

- Pole je dátová štruktúra, ktorá uchováva viacero hodnôt rovnakého typu.
  - Syntax: `typ názov_pola[n];`
  - kde **typ** je dátový typ prvkov poľa a **n** je počet prvkov.
- Pole môžeme inicializovať pri jeho vytvorení, ak nešpecifikujeme všetky prvky, zvyšné budú inicializované na 0:
  - `int pole[5] = {1, 2}; // pole[2], pole[3], pole[4] budú 0`
- Prvky poľa sú prístupné pomocou indexov, ktoré začínajú od nuly. Príklad:

```
int pole[5] = {10, 20, 30, 40, 50};
printf("%d", pole[0]); // Výstup: 10
printf("%d", pole[4]); // Výstup: 50
```

- Cyklus cez pole

```
for (int i = 0; i < 5; i++) {
    printf("%d ", pole[i]);
}
```

Aké chyby statického poľa môžu vznikať?

- Jazyk C nekontroluje hranice poľa, čo môže viesť k nepredvídateľnému správaniu alebo pádu programu.
- Napríklad

```
int pole[5] = {1, 2, 3, 4, 5};
printf("%d", pole[10]); // Neplatný prístup do poľa – chyba
```

- Veľkosť poľa musí byť pri statických poliach známa pri kompilácii a nemôže sa meniť počas behu programu → riešenie dynamickou alokáciou pamäte.
- Jazyk C nepodporuje záporné indexy
- Napríklad

```
int pole[5] = {1, 2, 3, 4, 5};
printf("%d", pole[-2]); // Neplatný index – chyba
```

Praktické cvičenia

1. Výpočet koreňov kvadratickej rovnice
2. Je znak písmeno?
3. Určenie parity čísel
4. Faktoriál
5. Reverzácia poľa
6. Reverzácia poľa a max
7. Rezerzácia poľa a priemer (avg)

- **Úloha:** Napíšte program v jazyku C, ktorý vypočíta korene kvadratickej rovnice. Program by mal zobraziť, či má rovnica dva reálne korene, jeden reálny koreň, alebo žiadne reálne korene.
- **Podmienky:**

1. Kvadratická rovnica má tvar ($ax^2 + bx + c = 0$).
2. Diskriminant (D) sa vypočíta podľa vzorca ($D = b^2 - 4ac$).
3. Ak je diskriminant väčší ako 0, rovnica má dva reálne korene.
4. Ak je diskriminant rovný 0, rovnica má jeden reálny koreň.
5. Ak je diskriminant menší ako 0, rovnica nemá žiadne reálne korene.

Riešenie – výpočet koreňov kvadratickej rovnice

```c
#include <stdio.h>
#include <math.h>

int main(void) {
    int a, b, c, discriminant;
    double x1, x2;

    // Využi %f pre čítanie desatinných čísel
    printf("Select three integer numbers: ");
    scanf("%d %d %d", &a, &b, &c);

    // Test koeficientu a
    if (a == 0) { // if (!a)
        printf("Error: The supplied coefficient 'a' = 0!\n");
        return 1;
    }

    discriminant = b * b - 4 * a * c;
    if (discriminant < 0) {
        printf("The quadratic equation has no roots.\n");
    } else if (discriminant == 0) {
        x1 = -b / (2.0 * a);
        printf("The quadratic equation has only one root: x1=%f\n", x1);
    } else {
        x1 = (-b + sqrt(discriminant)) / (2 * a);
        x2 = (-b - sqrt(discriminant)) / (2 * a);
        printf("The quadratic equation has the following roots: x1=%f, x2=%f\n", x1, x2);
    }
    return 0;
}
```

Praktické cvičenie – je znak písmeno?

- **Úloha:** Napíš program, ktorý zistí, či je zadaný znak písmeno. Vyskúšajte si tiež, že znaky môžu byť porovnávané s číselnými hodnotami.

Ďalšie informácie:

- Znaky v ASCII tabuľke majú číselné hodnoty, napríklad ‘a’ má hodnotu 97.
- Program používa podmienky na kontrolu, či je znak v rozsahu veľkých alebo malých písmen.

Riešenie – je znak písmeno?

```c
#include <stdio.h>

int main(void) {
    char c;

    printf("Select a character: ");
    scanf("%c", &c);

    if ((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z')) {
        printf("Char '%c' is a letter!\n", c);
    } else {
        printf("Char '%c' is not a letter!\n", c);
    }

    if ('a' == 97) {
        printf("Character '%c' has ordinal value of '%d'!\n", 97, 'a');
    }

    return 0;
}
```

Praktické cvičenie – určenie parity čísel

- **Úloha:** Napíš program, ktorý zistí, či sú zadané celé čísla párne alebo nepárne. Program by mal načítať tri celé čísla a pre každé z nich vypísať, či je párne alebo nepárne.

Ďalšie informácie:

- Číslo je párne, ak je deliteľné dvomi bez zvyšku, inak je nepárne.
- Program používa operátor modulo (%) na zistenie zvyšku po delení.
- Použitie cyklu for umožňuje efektívne spracovanie ľubovoľného počtu vstupov.

Riešenie – určenie parity čísel

```c
#include <stdio.h>

int main(void) {
    int num_count;
    // Načítaj počet čítaných hodnôt
    printf("Select the number of supplied values: ");
    scanf("%d", &num_count);

    // 1) Začni na i=0
    // 2) Otestuj, či podmienka stále platí
    // 3) Vykonaj 1 iteráciu
    // 4) Aktualizuj iteračnú premennú
    // 5) Pokračuj krokom 2)
    for (int i = 0; i < num_count; i++) {
        int num;
        printf("Select the %d. number: ", i + 1);
        scanf("%d", &num);

        if (num % 2 == 0) {
            printf("Number %d is: even\n", num);
        } else {
            printf("Number %d is: odd\n", num);
        }
    }

    return 0;
}
```

- **Úloha:** Napíš program, ktorý vypočíta faktoriál zadaného celého čísla. Program by mal skontrolovať, či je číslo nezáporné, a potom vypočítať jeho faktoriál.
- **Ďalšie informácie:**
  - Faktoriál čísla (n) je definovaný ako súčin všetkých kladných celých čísel menších alebo rovných (n)
  - Faktoriál je definovaný len pre nezáporné čísla.

Riešenie – faktoriál

```c
#include <stdio.h>

int main(void) {
    int num;

    printf("Select an integer number: ");
    scanf("%d", &num);

    // Faktoriál je definovaný pre nezáporné čísla
    if (num < 0) {
        printf("Error: Factorial is defined only for non-negative numbers!\n");
        return 1;
    }

    // 1) Definujte a inicializujte akumulátor (int factorial = 1)
    // 2) Začnite iteráciu od hodnoty 0 (int i = 0)
    // 3) Skontrolujte, či je podmienka stále platná (i < num), ak nie - ukončite cyklus
    // 4) Vykonajte jednu iteráciu cyklu: aktualizujte hodnotu akumulátora (factorial *= i + 1)
    // 5) Aktualizujte iteračnú premennú (i++)
    // 6) Prejdi na krok 3)
    int factorial = 1;
    for (int i = 0; i < num; i++) {
        factorial *= i + 1;
    }

    printf("Factorial of %d = %d\n", num, factorial);

    return 0;
}
```

- **Úloha:** Napíš program, ktorý načíta päť celých čísel do poľa a vypíše ich v opačnom poradí.
- **Ďalšie informácie:**
  - Pole je dátová štruktúra, ktorá uchováva viacero hodnôt rovnakého typu.
  - Indexovanie poľa začína od 0 a končí pri veľkosti poľa mínus jeden.

Riešenie – reverzácia poľa

```c
#include <stdio.h>

int main(void) {
    // Statické polia sú definované ako <typ> <identifikátor>[<veľkosť>]
    // Polia sú indexované od 0, teda maximálny index je <veľkosť> - 1!
    int arr[5];

    // Premenná iterácia sa môže použiť na indexovanie poľa
    printf("Select 5 numbers: ");
    for (int i = 0; i < 5; i++) {
        scanf("%d", &arr[i]);
    }

    // Iterujte pole v opačnom poradí - začnite od konca
    // pole a zníženie iteračnej premennej
    printf("Reversed: ");
    for (int i = 4; i >= 0; i--) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

- **Úloha:** Napíš program, ktorý načíta päť celých čísel do poľa, vypíše ich v opačnom poradí a nájde najväčšie číslo v poli.
- **Ďalšie informácie:**
  - Na nájdenie najväčšieho čísla v poli je potrebné prejsť všetky prvky poľa a porovnať ich hodnoty.

Riešenie – reverzácia a max

```c
#include <stdio.h>

int main(void) {
    // Static arrays are defined as <type> <identifier>[<size>]
    // Arrays are indexed from 0, thus maximum index is <size> - 1!
    int arr[5];

    // The iteration variable can be used to index the array
    printf("Select 5 numbers: ");
    for (int i = 0; i < 5; i++) {
        scanf("%d", &arr[i]);
    }

    // Iterate the array in the reverse order - start at the end of the
    // array and decrease the iteration variable
    printf("Reversed: ");
    for (int i = 4; i >= 0; i--) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    int max = arr[0];
    for (int i = 1; i < 5; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
    }
    printf("%d is the maximum of the array values.\n", max);

    return 0;
}
```

Praktické cvičenie – reverzácia a priemer

- **Úloha:** Napíš program, ktorý načíta päť celých čísel do poľa, vypíše ich v opačnom poradí, nájde najväčšie číslo v poli a vypočíta priemer hodnôt v poli.

Riešenie – reverzácia a priemer

```c
#include <stdio.h>

int main(void) {
    int arr[5];

    printf("Select 5 numbers: ");
    for (int i = 0; i < 5; i++) {
        scanf("%d", &arr[i]);
    }

    printf("Reversed: ");
    for (int i = 4; i >= 0; i--) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    int max = arr[0];
    int sum = arr[0];
    for (int i = 1; i < 5; i++) {
        sum += arr[i];
        if (arr[i] > max) {
            max = arr[i];
        }
    }
    double avg = sum / 5.0;

    printf("%d is the maximum of the array values.\n", max);
    printf("%f is the average of the array values.\n", avg);

    return 0;
}
```