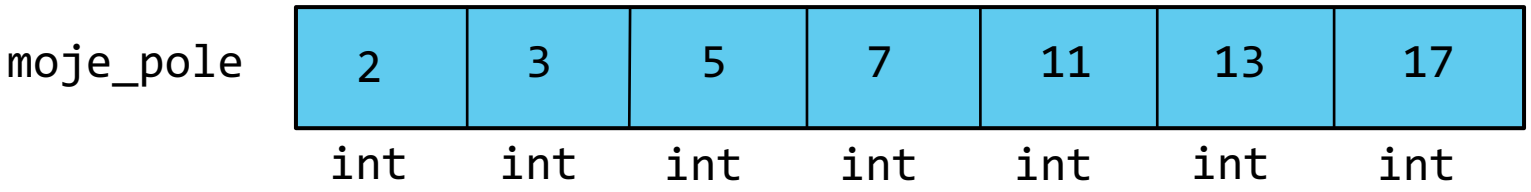
Pole a Funkce

Úvod do programování 1

Tomáš Kühr

(Jednorozměrné) pole

- Datová struktura
  - Lineární
  - Homogenní = prvky stejného datového typu
  - Statická = předem určený počet prvků
- Pole umožňuje pohodlně zpracovat větším množství dat



Vytvoření pole

- Pole je proměnná – před použitím je potřeba jej vytvořit
- bez inicializace:
  typ identifikátor[*velikost*];
- s inicializací:
  typ identifikátor[*velikost*] = {*inicializační hodnoty*};
- Inicializační hodnoty oddělujeme čárkou
- Hodnot může být i méně, než jaká je velikost pole
- V případně neuvedení velikosti se tato určí podle počtu uvedených hodnot
- Příklady:

```
int cisla[5];
int cisla[5] = { 1, 2, 3, 4, 5 };
int cisla[] = { 1, 2, 3, 4, 5 };
int cisla[10] = { 1, 2, 3, 4, 5 };
double desetinna[4];
double desetinna[] = { 1.1, 5.1, 3.2, 1e5 };
```

Používání pole

- K prvkům přistupujeme pomocí operátoru indexu [ ]
- Prvky mají indexy od 0 do velikosti pole – 1
- Příklad:

```
int i;
int pole[10];

for (i = 0; i < 10; i++){
    pole[i] = i + 1;
}

for (i = 0; i < 10; i++){
    printf("%i, ", pole[i]);
}
```

Velikost pole

- Velikost nelze z proměnné typu pole nijak zjistit
- Při práci s polem ji obvykle máme uloženou v nějaké proměnné
- V ANSI C:

```
#define VELIKOST 10
...
int i;
int pole[VELIKOST];
for (i = 0; i < VELIKOST; i++){
    pole[i] = i + 1;
}
```

- Od C99 funguje také:

```
int i;
const int velikost = 10;
int pole[velikost];
for (i = 0; i<velikost; i++){
    pole[i] = i + 1;
}
```

Textové řetězce

- Pole znaků ukončených znakem `'\0'`
- Velikost pole tedy je délka řetězce + 1
- Funkce pro práci s řetězci najdete v knihovně **string.h**
- Příklad vytvoření:

```
char retezec[] = "ahoj";
char retezec[] = { 'a', 'h', 'o', 'j', '\0' };
```

- Příklad využití zarážky:

```
char retezec[] = "ahoj";
char znak;
int i = 0;
while (znak = retezec[i]){
    printf("%c ", znak);
    i++;
}
```

Vybrané funkce ze string.h

- Délka řetězce: `strlen`
- Kopírování řetězce: `strcpy`, `strncpy`
- Spojení řetězců: `strcat`, `strncat`
- Porovnávání řetězců (lexikografické): `strcmp`, `strncmp`
- Hledání znaku vřetězci: `strchr` (zleva), `strrchr` (zprava)
- Hledání podřetězce v řetězci: `strstr`
- A mnoho dalších...

Funkce

- Relativně samostatná část programu
- Je ji možné opakovaně volat z libovolných částí programu
- Může mít své vstupy = parametry funkce
- I výstup = návratová hodnota funkce
- Příklady: `main`, `printf`, `scanf`, …

Vytvoření funkce

- Funkci nelze definovat uvnitř jiné funkce (včetně `main`)
- Funkci je potřeba definovat dříve, než ji voláte
- Obecně:

```
typ_výsledku jméno_funkce(typ_p1 jméno_p1, ... , typ_pN jméno_pN){
    příkazy_těla_funkce
    ...
}
```

- V těle funkce se mohou využívat identifikátory parametrů
- Parametry se chovají jako lokální proměnné
- Tělo funkce může obsahovat příkaz `return` pro nastavení výsledku
- Příkaz `return` ihned funkci ukončí

Příklad vytvoření funkce

- Funkce bez návratové hodnoty:

```
void vypis_max(int prvni, int druhe){
    int max;
    max = (prvni < druhe) ? druhe : prvni;
    printf("Maximum je %d.\n", max);
    return;
}
```

- Funkce s výsledkem:

```
int maximum(int prvni, int druhe){
    int max;
    max = (prvni < druhe) ? druhe : prvni;
    return max;
}
```

Použití funkce

- Pomocí operátoru volání funkce (kulaté závorky)
- Uvnitř závorek uvádíme hodnoty vstupů
- Tyto se pak vloží do parametrů
- Návratovou hodnotu můžeme, ale nemusíme použít
- Příklady:

```
int main(){
    int a = 5, b = 10;
    int vetsi = maximum(b, 3 * a);
    printf("Max: %d\n", maximum(vetsi, a + b));
    vypis_max(a*b, a + b);
    return 0;
}
```

Funkce pracující s polem

- Pole lze předat do funkce jako vstup
- Zatím ho ale neumíme vrátit z funkce jako výsledek
- Pozor, změny provedené v poli se projeví i mimo funkci
- Příklad:

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

int main() {
    int cisla[] = {1, 2, -7, 14};
    vypis_pole_cisel(cisla, 4);
    return 0;
}
```

Příklad s modifikací pole

```
#define DELKA 3
void plus10(int vstup[], int pocet){
    for (int i = 0; i < pocet; i++){
        vstup[i] += 10;
    }
}

int main(){
    int pole[] = { 10, 20, 30 };
    plus10(pole, 3);
    for (int i = 0; i < DELKA; i++){
        printf("%d, ", pole[i]);
    }
    return 0;
}
```

Deklarace funkce

- Hodí se především u větších programů
- Je komplikované /nepřehledné funkce definovat dříve, než jsou použity
- Někdy je toto dokonce nemožné
- Deklarace = popis funkce dostačující k tomu, aby překladač věděl, jak bude funkce používána
- Uvádí se zpravidla na začátku programu
- Udává jméno funkce, typy parametrů a návratové hodnoty
- Obecně:

```
typ_výsledku jméno_funkce(typ_p1 jméno_p1, ...);
typ_výsledku jméno_funkce(typ_p1, ...);
```

Příklad s deklarací

```
int soucet(int, int);
void novy_radek();

int soucet(int a, int b) {
    int soucet = a + b;
    printf("soucet a + b = %i", soucet);
    novy_radek();
    return soucet;
}

void novy_radek() {
   printf("\n");
}
```

Rozsah platnosti - lokální proměnná

- Definována uvnitř bloku (funkce, cyklus, …)
- Vzniká při vstupu programu do bloku
- Existuje po dobu vykonávání příkazů v bloku
- Zaniká při opuštění bloku
- Příklad:

```
// tady proměnná znak ještě neexistuje
while (text[i] != '\0'){
    char znak = text[i] - 'A' + 'a';
    printf("%c\n", znak);
}
// tady proměnná znak už neexistuje
```

Rozsah platnosti - globální proměnná

- Definována mimo jakékoli bloky
- Vzniká spuštěním programu
- Přístupná ze všech míst programu
- Lokální identifikátory mohou překrýt ty globální (i jiné lokální)
- Zaniká při ukončení programu
- Příklad:

```
int pocet = 10;  // proměnná přístupná odkudkoli

int main(){ ... }

...
```

Rozsah platnosti - statická proměnná

- Definujeme uvnitř bloku
- Pomocí klíčového slova `static`
- Vzniká (a je inicializována) při prvním vstupu do bloku
- Přístupná je pouze z daného bloku
- Zaniká ale až ukončením programu
- Příklad:

```
int porovnej(int prvni, int druhe){
    static int pocet = 0;
    pocet++;
    if (prvni < druhe) return -pocet;
    if (prvni == druhe) return 0;
    return pocet;
}
```

Pole – cvičení 4

Vytvořte program, který vyhledá v zadaném poli zadané číslo a vypíše jeho index (nebo informaci o tom, že číslo v poli nebylo nalezeno). Při testování můžete pole nechat načítat od uživatele. Program by měl být ale funkční bez ohledu na to, jaká čísla pole obsahuje...

Příklad použití:

```
Zadejte cislo: -5
Zadejte cislo: 17
Zadejte cislo: 3
Zadejte cislo: 2
Zadejte cislo: 8

Zadejte hledane cislo: 2
Index hledaneho cisla je: 3
Zadejte hledane cislo: 20
Hledane cislo nebylo nalezeno.
```

Pole – cvičení 5

Vytvořte program, který projde dané pole a spočítá, kolik čísel v tomto poli je v zadaném rozmezí (mezi zadanými 2 hodnotami). Při testování programu si zkuste nalezená čísla třeba i vypisovat do konzole.

Příklad použití:

```
Zadejte cislo: -5
Zadejte cislo: 17
Zadejte cislo: 3
Zadejte cislo: 2
Zadejte cislo: 8

Zadejte zacatek intervalu: 0
Zadejte konec intervalu: 10
Pocet cisel v intervalu je: 3
```

Funkce – cvičení 3

Vytvořte program, který po zadání typu tělesa (válec, kvádr, ...) a jeho rozměrů vypočítá objem a povrch tohoto tělesa. Jednotlivé výpočty objemů a povrchů vybraných těles naprogramujte pomocí funkcí. Načítání dat a výpisy pak ponechte v hlavní funkci programu.

Příklad použití:

```
Zadejte typ telesa (1=valec, 2=kvadr): 2
Zadejte prvni rozmer: 1
Zadejte druhy rozmer: 2
Zadejte treti rozmer: 3

Objem je: 6
Povrch je: 22
```