Základy programování (IZP)

6. počítačové cvičenie

Brno University of Technology, Faculty of Information Technology
Božetěchova 1/2. 612 66 Brno - Královo Pole
ilazur@fit.vut.cz

Prečo potrebujeme ukazovatele?

- Statická pamäť(stack) je obmedzená, pri programoch s predom neznámym počtom prvkov by sme ju mohli ľahko vyčerpať.
- Preto máme dynamickú pamäť(heap), z ktorej si zoberieme toľko, koľko potrebujeme. Ak k nej ale chceme pristupovať, musíme použiť ukazovatele.
- Taktiež pri volaní funkcií nechceme zbytočne kopírovať dáta, radšej funkcii ukážeme, kde v pamäti sú.

Dátový typ ukazovateľ

```
int *a;
int b;
int c;
```

Operátor referencie

```
a = &b; // a points on memory address of b
```

Operátor dereferencie

```
c = *a; // get value from memory address
```

Ako to vlastne funguje

```
#include <stdio.h>

int main() {
    int a, *p_a; // We have normal variable a and pointer on variable p_a
    a = 42;
    p_a = &a; // Put reference on variable a
    *p_a = 15; // Access variable a with pointer and dereference operator, set new value
    printf("New value of variable 'a' is %d\n", a);
    printf("And same value with pointer access: %d\n", *p_a);
    return 0;
}
```

Dátový typ ukazovateľa a premennej, na ktorú ukazuje, musia byť rovnaké

Ako to vlastne funguje

1. Načítajte zo STDIN 1 celé a jedno desatinné číslo
2. Vytvorte si ukazovatele na obe čísla
3. Pomocou ukazovateľov a dereferencie čísla sčítaje a výsledok vypíšte

```
#include <stdio.h>

int main() {

    // There will be your code

    return 0;
}
```

Ako to vlastne funguje

```
#include <stdio.h>

int main() {
    int a, *p_a;
    double b, *p_b;

    p_a = &a;
    p_b = &b;

    scanf("%d", p_a);
    scanf("%lf", &b);

    double sum = 0;
    sum = *p_a + *p_b;

    printf("%lf", sum);

    return 0;
}
```

Pole ako ukazovateľ

```
#include <stdio.h>
int main() {
    int arr[] = {0, 1, 2, 3, 4}, *p_arr;
    p_arr = arr; //arr in itself points on first elem of array
    printf("%d\n", arr[0]);
    printf("%d\n", *p_arr);

    // Print second element
    printf("%d\n", arr[1]);
    printf("%d\n", *(p_arr + 1));

    // Print last element
    printf("%d\n", arr[4]);
    printf("%d\n", *(p_arr + 4));

    for (int i = 0; i < 5; i++) {
        printf("%d\n", *p_arr);
        p_arr++;
    }

    return 0;
}
```

Pole ako ukazovateľ

1. Načítajte zo STDIN 5 celých čísiel do poľa
2. Ku každému číslu pripočítajte hodnotu 1 pomocou ukazovateľa a dereferencie
3. Vypíšte čísla v opačnom poradí

```
#include <stdio.h>

#define ARR_SIZE 5

int main() {
    int arr[ARR_SIZE], *p_arr;

    // scanf to array and add value 1 to every number

    for (int i = (ARR_SIZE - 1); i >= 0; i--) {
        printf("%d ", arr[i]);
    }

    printf("\n");
    return 0;
}
```

Pole ako ukazovateľ

```
#include <stdio.h>
#define ARR_SIZE 5
int main() {
    int arr[ARR_SIZE], *p_arr;

    p_arr = arr; // arr in itself is pointer to 0 elem, so we can set our
  pointer to it
    for (int i = 0; i < ARR_SIZE; i++) {
        scanf("%d", p_arr); // same as scanf("%d", &arr[i]);
        p_arr++; // Moving pointer to next element
    }
    p_arr = arr; // We have to reset pointer to 0 elem of array
    for (int i = 0; i < ARR_SIZE; i++, p_arr++) {
        (*p_arr)++; // (*p_arr) = (*p_arr) + 1; // arr[i] = arr[i] + 1;
    }

    for (int i = (ARR_SIZE - 1); i >= 0; i--) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

Odovzdávanie parametrov pomocou odkazu

1. Načítajte zo STDIN 2 celé čísla
2. Napíšte funkciu, ktorá zadané čísla podelí a výsledok vráti
3. Nezabudnite ošetriť prípad, kedy by mohlo dôjsť k deleniu 0

```
#include <stdio.h>

int main() {
    int dividend, divisor;
    double quotient;
    scanf("%d %d", &dividend, &divisor);

    // There will function call
    printf("%d / %d = %lf\n", dividend, divisor, quotient);

    return 0;
}
```

Odovzdávanie parametrov pomocou odkazu

```
#include <stdio.h>
int div(int dividend, int divisor, double *quotient) {
    if (divisor == 0) {
        return 0;
    }
    *quotient = (double)dividend / divisor;
    return 1;
}

int main() {
    int dividend, divisor;
    double quotient;
    scanf("%d %d", &dividend, &divisor);

    if (!div(dividend, divisor, &quotient)) {
        printf("Division by zero!\n");
        return 1;
    }
    printf("%d / %d = %lf\n", dividend, divisor, quotient);

    return 0;
}
```

Ukazovatele na štruktúry

1. Napíšte funkciu, ktorá zadané čísla v štruktúre zväčší o 1
2. Funkciu vypracujte tak, že bude využívať iba ukazovate

```
#include <stdio.h>

struct pair_t {
    int first;
    int second;
};

int main() {
    struct pair_t object, *p_object;
    object.first = 5;
    object.second = 2;

    p_object = &object;

    // There will be function call

    printf("%d %d\n", object.first, object.second);
    return 0;
}
```

Odovzdávanie parametrov pomocou odkazu 2

```
void add(struct pair_t *object) {
    object->first = object->first + 1;
    object->second = object->second + 1;
}
```

```
add(p_object);
```

Relácie

1. Navrhnite štruktúry pre definičný obor set_t a množinu pair_t. Definičný obor by mal obsahovať jednotlivé prvky a svoju veľkosť. Zadaný obor je 1, 2, 3, 4, 5. Množina by mala byť tvorená polom dvojíc, x a y.
2. Navrhnite funkciu ktorá overí, či je zadaná množina funkciou, inak povedané, či má každé x z definičného oboru nejaký obraz v obore hodnôt.

```
struct set_t {
    int itemA;
    int itemB;
};
struct set_t set;
```

```
typedef struct {
    int itemA;
    int itemB;
} simple_set_t;
simple_set_t set;
```

Relácie

```
#include <stdio.h>
#include <stdbool.h>
typedef struct {
    int items(5);
    int cardinality;
} set_t;

typedef struct {
    int  first ;
    int second;
} pair_t;

int main() {
    set_t set = {{ 1, 2, 3, 4, 5 }, 5};
    pair_t rel1(5) = { {1, 1}, {2, 2}, {3, 3}, {4, 4}, {5, 5} };
    pair_t rel2(5) = { {1, 3}, {3, 1}, {1, 2}, {2, 1}, {4, 4} };
    pair_t rel3(5) = { {1, 2}, {2, 3}, {3, 3}, {5, 1}, {4, 4} };

    // There will be function calls
    return 0;
}
```

Relácie

```
bool is_function(pair_t relation(), int rel_size, set_t set) {
    for (int i = 0; i < set.cardinality; i++) {
        int count = 0;
        for (int j = 0; j < rel_size; j++) {
            if (set.items(i) == relation(j).first)
            count++;
        }
        if (count != 1)
            return 0;
    }
    return 1;
}
```

Relácie

1. Rozšírte svoje riešenie predchádzajúceho príkladu o funkciu, ktorá nájde lokálne maximum a lokálne minimum a vráti dvojice prvkov, kde sa toto maximum a minimum nachádza

```
int main() {
    pair_t object, *p_object;
    int *p;
    object.first  = 5;
    object.second = 2;

    p_object = &object;
    p_object->first = 0;
    printf("%d, %d\n", object.first, object.second); // 0, 2

    p = &object.second;
    *p = 3;
    printf("%d, %d\n", object.first, object.second); // 0, 3

    return 0;
}
```

Relácie

```
int rel_minmax(pair_t rel(), int rel_size, pair_t *min, pair_t *max) {
    *min = rel(0);
    *max = rel(0);
    for (int i = 1; i < rel_size; i++) {
        if (rel(i).second < min->second) {
            *min = rel(i);
        } else if (rel(i).second > max->second) {
            *max = rel(i);
        }
    }

    return 0;
}
```

Stačí vypracovať jednu variantu

- Zo STDIN načítajte string s maximálnou dĺžkou 100. Vo funkcii načítajte dve celé čísla. Opäť vo funkcii vymeňte písmená na zadaných indexoch. Výsledok vypíšte. Všetky vami navrhnuté funkcie musia mať návratový typ void.
- Deklarujte si 5 prvkové pole a inicializujte si ho ľubovoľnými hodnotami. Pomocou ukazovateľov a jednoduchého porovnávania (bubble sort) pole zoraďte. Výsledok vypíšte. Nemusíte vytvárať žiadnu funkciu.
- Vytvorte si štruktúru pre trojrozmerný bod v priestore. Načítajte hodnoty pre dva body. Pomocou vami vytvorenej funkcie vypočítajte nový bod, ktorý bude ležať medzi týmito dvoma bodmi, postačí jednoduché sčítanie a podelenie súradníc dvoma. Navrhnutá funkcia bude mať návratový typ void.