Základy programování (IZP)

7. počítačové cvičenie

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

Opakovanie - 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 + i)); //printf("%d\n", arr[i]);
    }

    return 0;
}
```

Vynásobte prvky poľa konštantou

1. Načítajte zo STDIN 1 celé číslo
2. Vytvorte si funkciu, ktorá prvky daného poľa vynásobí konštantou

```
#include <stdio.h>
#define ARR_SIZE 10

int main() {
    int array(ARR_SIZE) = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

    int value;
    scanf("%d", &value);

    // There will be function call

    for (int i = 0; i < ARR_SIZE; i++) {
        printf("%d ", array(i)); // printf("%d ", *(array + i));
    }

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

Vynásobte prvky poľa konštantou

```
void array_multiply(int *arr, int value) {
    for (int i = 0; i < ARR_SIZE; i++) {
        arr[i] *= value; // arr[i] = arr[i] * value;
    }
}
```

```
array_multiply(array, value);
```

Vložte do poľa nový prvok na zadaný index

1. Načítajte zo STDIN 2 celé čísla, index a hodnotu
2. Vytvorte si funkciu, ktorá na daný index vloží novú hodnotu a zvyšné prvky posunie nahor, pričom posledný prvok z pola vypadne

```
#include <stdio.h>
#define ARR_SIZE 10

int main() {
    int array[ARR_SIZE] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

    int indx, value;
    scanf("%d %d", &indx, &value);

    // There will be function call

    for (int i = 0; i < ARR_SIZE; i++) {
        printf("%d ", *(array + i));
    }
    printf("\n");
    return 0;
}
```

Vložte do poľa nový prvok na zadaný index

```
int array_insert(int arr(), int position, int value) {
    if (position < 0 || position >= ARR_SIZE) {
        return 0;
    }

    for (int i = ARR_SIZE - 1; i > position; i--) {
        arr(i) = arr(i - 1); //*(arr + i) = *(arr + i - 1);
    }

    arr(position) = value; //*(arr + position) = value;
    return 1;
}
```

```
array_insert(array, indx, value);
```

Vytvorte funkciu pre parsovanie argumentov

1. Vytvorte si dátovú štruktúru pre ukladanie hodnôt argumentov. Vstupné argumenty budú jeden prepínač, jedna číselná hodnota a jeden textový vstup.
2. Inicializujte si túto štruktúru na globálnej úrovni.

Vytvorte funkciu pre parsovanie argumentov

```
#include <stdio.h>
#include <stdbool.h>

typedef struct {
    bool xflag;
    char *svalue;
    int  nvalue;
} config_t;

config_t config = {
    .xflag = false,
    .svalue = NULL,
    .nvalue = 10
};
```

Vytvorte funkciu pre parsovanie argumentov

1. Vytvorte funkciu, ktorá argumenty spracuje a uloží do globálnej štruktúry.

```
const char *usage = "syntax: %s (-x|-y) (-n COUNT) -s STR\n"
    "-x and -y are optional and mutually exclusive\n"
    "-s STR - mandatory, STR is a string\n"
    "-n COUNT - optional, COUNT is non-negative integer (default: 10)\n";

int main(int argc, char *argv()) {
    // There will be function call

    printf("Parsed arguments:\n");
    printf("  xflag: %d\n", config.xflag);
    printf("  svalue: %s\n", config.svalue);
    printf("  nvalue: %d\n\n", config.nvalue);

    return 0;
}
```

Vytvorte funkciu pre parsovanie argumentov

```
int parse_args(int argc, char **argv) {
    for (int arg = 1; arg < argc; arg++) {
        if (strcmp(argv(arg), "-x") == 0) {
            config.xflag = true;
        } else if (strcmp(argv(arg), "-n") == 0) {
            if (n_set || arg + 1 >= argc) {
                return 4;
            }
            config.nvalue = atoi(argv(++arg));
        } else if (strcmp(argv(arg), "-s") == 0) {
            if (config.svalue != NULL || arg + 1 >= argc) {
                return 5;
            }
            config.svalue = argv(++arg);
            return 1;
        } else {
            return 6;
        }
    }
    return 1;
}
```

Vytvorte funkciu pre parsovanie argumentov

```
    if (parse_args(argc, argv) != 1) {
        fprintf(stderr, usage, argv[0]);
        return 1;
    }
```

Inicializácia pamäte

1. Navrhnite funkciu, ktorá vám alokuje požadovanú pamäť pre vektor o danom počte prvkov.
2. Navrhnite funkciu, ktorá inicializuje hodnoty vektora

```
#include <stdio.h>
#include <stdlib.h>

typedef struct {
    int *items;
    int size;
} vector_t;

int main() {
    vector_t v1;
    // Construction function call
    // Init function call

    for (int i = 0; i < v1.size; i++) {
        printf("%d , ", v1.items(i));
    }
    return 0;
}
```

Inicializácia pamäte

malloc(počet prvkov * sizeof(dátový typ prvkov))

```
int vector_ctor(vector_t *v, unsigned int size) {
    v->items = malloc(size * sizeof(int));
    // int arr(size); - It is very close to static alocated array
    // After malloc you can use same syntax for accessing dynamic array
  as in case of static array

    if (v->items == NULL) {
        // Allocation failed.
        return 0;
    }

    v->size = size;
    return 1;
}
```

```
vector_ctor(&v1, 5);
```

Vynásobenie hodnôt vektoru

1. Navrhnite funkciu, ktorá hodnoty daného vektoru vynásobí konštantou

```
int main() {
    vector_t v1;

    if (!vector_ctor(&v1, 5)) {
        printf("Vector construction failed!\n");
        return 1;
    }
    vector_init(&v1);

    // There will be function call

    return 0;
}
```

Vynásobenie hodnôt vektoru

```
void vector_scalar_multiply(vector_t *v, int scalar) {
    for (int i = 0; i < v->size; i++) {
        v->items(i) *= scalar; // v->items(i) = v->items(i) * scalar;
    }
}
```

```
vector_scalar_multiply(&v1, 2);
```

Uvoľnenie pamäte

1. Navrhnite funkciu, ktorá uvoľní alokovanú pamäť

```
#include <stdio.h>
#include <stdlib.h>

typedef struct {
    int *items;
    int size;
} vector_t;

int main() {
    vector_t v1;

    if (!vector_ctor(&v1, 5)) {
        printf("Vector construction failed!\n");
        return 1;
    }
    vector_init(&v1);

    // There will be function call

    return 0;
}
```

Uvoľnenie pamäte

Každý malloc si vyžaduje svoje free

```
void vector_dtor(vector_t *v) {
    if (v->items != NULL) {
        free(v->items);
        v->items = NULL;
    }
    v->size = 0;
}
```

```
vector_dtor(&v1);
```

Stačí vypracovať jednu variantu

- Zo STDIN načítajte string s maximálnou dĺžkou 100. Vo funkcii overte, či sa tam nachádza vami daný podstring. Ak áno, vymažte ho a výsledok vypíšte. Príklad, na vstupe je string "abcdefgh", vami daný podstring je "cde", výsledok teda bude "abfgh".
- Definujte si 10 prvkové pole a inicializujte si ho ľubovoľnými hodnotami. Vo vami navrhnutej funkcii upravte hodnoty tak, že hodnota prvku bude súčtom hodnôt jeho susedov. Nezabudnite ošetriť krajné prípady. Výsledok vypíšte pomocou funkcie.
- Alokujte si pamäť pre dva ľubovoľne dlhé vektory. Inicializujte ich hodnoty a potom vo funkcii urobte súčet hodnôt vektorov do jedného z nich. Výsledok vypíšte a nezabudnite na korektnú prácu s dynamickou pamäťou.