

САА – Упражнение 3

Структура от данни стек

Стекът е линейна структура от данни, в която обработката на информация става само от едната страна, наречена връх (дъното не е достъпно при наличие на елементи в стека). Организацията на стека е LIFO – Last In First Out – т.е. последния въведен елемент излиза първи.

Стекът може да бъде реализиран по два начина:

- Статично – чрез масив;
- Динамично.

Пример за приложение на стека при проверка на баланса на скобите в даден израз – за дадения израз проверката се прави чрез създаване на празен стек и четене последователно на елементите на израза. Ако елементът е отваряща скоба: ‘(’, ‘{’, ‘[’ тя се записва в стека. Ако елементът е затваряща скоба - ‘)’, ‘}’ или ‘]’ се проверява символа на върха на стека:

1. ако той е съответната отваряща скоба се изтрива от стека (това означава, че има правилно съответствие между отваряща и затваряща скоба);
2. ако стекът е празен или пък символът не е съответната отваряща скоба, това означава, че има грешка в разполагането на скобите.

Задачи:

1. Реализирайте програмно алгоритъма за проверка на баланс на скобите в даден израз като използвате динамична реализация на стек.

Дефинирайте структурата, която ще използвате за съхраняване на данни в стека и указател към върха на стека.

```
struct bracket
{
    char symbol;
    struct bracket * prev;
} * sp = NULL;
```

В главната функция организирайте потребителския вход – въвеждането на израза за проверка. За всеки от въведените символи правете проверка дали не е отваряща скоба – ако е така, използвайте функция, за да добавите елемента в стека.

Във функцията за добавяне на елемент в стека можете да използвате следния код:

```
ptr = (struct bracket *)malloc(sizeof(struct bracket));
if (!ptr)
    return 0;
```

```
ptr->symbol = c;  
ptr->prev = sp;  
sp = ptr;  
return 1;
```

Ако въведения елемент е затваряща скоба използвайте функция за проверка на елемента на върха на стека:

```
int checkStack (char c)  
{  
    char mirror;  
    struct bracket * ptr;  
    switch (c)  
    {  
        case ')' :  
            mirror = '(';  
            break;  
        case '}' :  
            mirror = '{';  
            break;  
        case ']' :  
            mirror = '[';  
            break;  
    }  
    if (sp && sp->symbol == mirror)  
    {  
        ptr = sp;  
        sp = sp->prev;  
        free (ptr);  
        return 1;  
    }  
    else  
        return 0;  
}
```