**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**ОТЧЕТ**

**по лабораторной работе №3**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: «Стеки и очереди»**

|  |  |  |  |
| --- | --- | --- | --- |
| Студент гр. 7381 | |  | Вологдин М.Д. |
| Преподаватель |  |  | Фирсов М.А. |

Санкт-Петербург

2018

**Задание:**

5-д. Правильная скобочная конструкция с тремя видами скобок определяется как

< текст > ::= < пусто > | < элемент > < текст >

< элемент > ::= < символ > | ( < текст > ) | [ < текст > ] | { < текст > }

где < символ > − любой символ, кроме ( , ) , [ , ] , { , }. Проверить, является ли текст, содержащийся в заданном файле *F*, правильной скобочной конструкцией; если нет, то указать номер ошибочной позиции.

**Пояснение задания:**

На вход программе подаётся строка, состоящая из скобок и прочих символов. Требуется проверить правильность расстановки скобок.

**Описание алгоритма:**

1. Проходим по всей строке, от 1го символа к последнему.
   1. Если это открывающая скобка, то кладём ее в стек.
   2. Если это закрывающая скобка и стек не пуст
      1. Удаляем верхний элемент стека, если там лежит подходящая открывающая скобка
      2. Иначе выводим ошибку и завершаем программу.
2. Если стек оказался пустым, то скобки расставлены верно.

**Описание структур**:

* Struct node  
  – Элемент стека, включающий значение и указатель на следующий элемент
* Struct Stack  
  – Сам стек, включающий верхний элемент стэка и размер.

**Описание функций**:

* Struct Stack init()  
  – Инициация стека, функция возвращает пустой стек.
* Void push(struct Stack \*stk, char simv)  
  – Помещение элемента в стек, функция принимает указатель на стек и элемент.
* Char pop(struct Stack \*stk)  
  – Удаление верхнего элемента из стека, функция принимает указатель на стек и возвращает верхний элемент стека.
* Char top(struct Stack \*stk)  
  – Просмотр верхнего элемента, функция принимает указатель на стек и возвращает верхний элемент стека.
* Int IsEmpty(struct Stack \*stk)  
  – Проверка стека на пустоту, функция принимает указатель на стек и возвращает размер стека.
* Int isopen(char bkt)  
  – Проверка скобки на открывающую, функция принимает символ и возвращает True, если скобка открывающая и False во всех других случаях.
* Int isclose(char bkt)  
  – Проверка скобки на закрывающую, функция принимает символ и возвращает True, если скобка закрывающая и False во всех других случаях.
* Int isbkt(char bkt)  
  – Проверка символа на скобку, функция принимает символ и возвращает True, если это скобка и False во всех других случаях.

**Тестирование** **программы**:

Для проверки программы были созданы тесты с корректными и некорректными данными.

Рассмотрим первый тест подробнее. На вход программе подаётся строка «(this is[simple][{test}text{with}]brackets[])». Как мы видим из вывода, программа поочередно кладёт в стек каждую открывающую скобку, а затем удаляет ее, как только к ней встретилась парная. После чего, тк стек стал пустым, программа вывела «currect».

В таблице ниже приведены результаты тестирования.

|  |  |
| --- | --- |
| (this is[simple][{test}text{with}]brackets[]) | push (  push [  pop [  push [  push {  pop {  push {  pop {  pop [  push [  pop [  pop (  correct |
| {This[is(text)]} | correct |
| (){}{[](}) | 8 - wrong position |
| ( java } c ) python [ scala ( ruby < assembler } c++ ] pascal ] java script < php < java ] c ( python ( scala [ ruby ) assembler ] c++ ( pascal ) java script } php { java ) c [ python ) scala ( ruby > assembler ] c++ { pascal ] java script } php | 7 - wrong position |
| (Look){at[this[text]] | wrong, there are not enough closing brackets at the end |

**Вывод:**

В результате работы была освоена тема «Стек и очередь» на примере реализации стека и функций для работы с ним.

**Приложение А**

**Исходный код**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

#include <stdio.h>

#include <string.h>

#include <stddef.h>

#include <stdlib.h>

struct node

{

char value;

struct node \*next;

};

struct Stack

{

struct node \*head;

int size;

};

//Functions for working with the stack

struct Stack init()

{

struct Stack stack = {NULL, 0};

return stack;

}

void push(struct Stack \*stk, char simv)

{

struct node \*node = (struct node\*)malloc(sizeof(struct node));

node->value = simv;

node->next = stk->head;

stk->head = node;

stk->size++;

}

char pop(struct Stack \*stk)

{

struct node \*node = stk->head;

stk->head = stk->head->next;

char res = node->value;

free(node);

stk->size--;

return res;

}

char top(struct Stack \*stk)

{

return (stk->head ? stk->head->value : 0);

}

int IsEmpty(struct Stack \*stk)

{

return (!stk->size);

}

int isopen(char bkt)

{

return (bkt == '(' || bkt == '[' || bkt == '{');

}

int isclose(char bkt)

{

return (bkt == ')' || bkt == ']' || bkt == '}');

}

int isbkt(char bkt)

{

return (isopen(bkt) || isclose(bkt));

}

int main()

{

struct Stack stack = init();

char str[500];

//Reading data

fgets(str,500,stdin);

for (int i=0; i<strlen(str);i++)

{

char c = str[i];

if (isbkt(c))

{

//If the bracket is open, then push

if (isopen(c))

{

//add tabs

for (int j=0;j<stack.size;j++)

printf(" ");

printf("push %c\n",c);

push(&stack,c);

}

else

{

//If the bracket covers the one on top of the stack, then pop, else error

if (!IsEmpty(&stack))

switch (top(&stack))

{

case '(':

if (c == ')')

{

//add tabs

for (int j=0;j<stack.size-1;j++)

printf(" ");

printf("pop %c\n", top(&stack));

pop(&stack);

}

else

{

printf("%d - wrong position\n",i);

return 0;

}

break;

case '[':

if (c == ']')

{

//add tabs

for (int j=0;j<stack.size-1;j++)

printf(" ");

printf("pop %c\n", top(&stack));

pop(&stack);

}

else

{

printf("%d - wrong position\n",i);

return 0;

}

break;

case '{':

if (c == '}')

{

//add tabs

for (int j=0;j<stack.size-1;j++)

printf(" ");

printf("pop %c\n", top(&stack));

pop(&stack);

}

else

{

printf("%d - wrong position\n",i);

return 0;

}

break;

default:

printf("%d - wrong position\n",i);

return 0;

}

else

{

printf("%d - wrong position",i);

return 0;

}

}

}

}

printf("%s", IsEmpty(&stack) ? "correct\n" : "wrong, there are not enough closing brackets at the end\n");

return 0;

}