**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №3**

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

**Тема: «Стеки и очереди»**

Студент гр. 7382 Дрозд А. С.

Преподаватель Фирсов М. А.

Санкт-Петербург

2018

**Задание.**

Проверить, является ли содержимое заданного текстового файла F

правильной записью формулы следующего вида:

< формула > ::= < терм > | < терм > + < формула > | < терм > - < формула >

< терм > ::= < имя > | ( < формула > ) | [ < формула > ] | { < формула > }

< имя > ::= x | y | z

Если не является, то указать номер ошибочной позиции.

**Пояснение задания.**

На вход программе подаётся имя файла. Итогом завершения программы должно быть выражение, дающее понять, что является ли выражение в файле правильной записью формулы или нет.

**Описание алгоритма.**

Программа просит пользователя ввести ключ(число). Если пользователь введет единицу, то программа попросит ввести путь до файла, в котором находится проверяемое выражение и считает строку от туда. Если пользователь введет другое число, то программа попросит ввести выражение вручную. После считывания программа начнет обработку строки. Обработкой строки занимается функция **are\_correct**. Эта функция посимвольно проверяет введенное выражение. Каждый символ может относиться к одной из групп: знак, открывающаяся скобка, закрывающаяся скобка, переменная или неизвестный знак. В зависимости от того, в какую группу относиться проверяемый символ, функция проверяет следующий символ и предыдущие скобки.

**Описание структур**

В программе использовалось одну структуру:

* struct elem

В структуре находится 2 указателя на структуры и символ. Указатель struct elem\* head указывает на предыдущую структуру списка. Указатель struct elem\* tail указывает на следующую структуру списка. Символ char sim содержит символ данного элемента.

**Описание функций**

В программе использовалось 7 функций:

* **int are\_correct(char\* str,char\*\* simbol)**

Принимает указатель на строку и указатель на этот указатель, и проверяет, является ли строка **str** формулой. Если строка — это бинарное коромысло возвращает 0, в обратном случае возвращает число от 1 до 8. Каждое из чисел показывает из-за чего функция преждевременно закончила работу. В \*simbol остается указатель на последний проверенный символ строки.

* **int IsEmpty(struct elem\* stack)**

Принимает указатель на структуру элемента стека. Если этот указатель пуст возвращает 0, в обратном случае 1.

* **int push(struct elem\*\* stack,char ch)**

Принимает указатель на указатель на структуру элемента стека и символ. Функция создает стек со структурой, в которую вставляет символ **ch**, если стек пустой. Если стек не пуст структура с символом вставляется в конец стека.

* **char top(struct elem\*\* stack)**

Принимает указатель на указатель на структуру элемента стека. Функция возвращает символ из последней структуры стека. Если стек пуст возвращает символ «$».

* **int pop(struct elem\*\* stack)**

Принимает указатель на указатель на структуру элемента стека. Функция удаляет последний элемент стека. Если в стеке один элемент, удаляет стек.

* **int destroy(struct elem\*\* stack)**

Принимает указатель на указатель на структуру элемента стека. Функция высвобождает всю использованную на стек память.

* **int main()**

Главная функция программы. Первоначально просит ввести ключ. В зависимости от введенного ключа функция будет по разному считывать строку. Если ключ единица, то считает из файла, путь к которому вы введете, если не единица считает из терминала. После пойдет обработка строки и ее проверка на то, является ли она формулой. В зависимости от данных, полученных при прохождения проверки функция выведет на экран является ли выражение формулой. Если выражение не формула, можно будет увидеть на каком символе допущена ошибка.

**Тестирование.**

|  |  |
| --- | --- |
| Вводимые данные | Результат: |
| 2  ((x+[]+z)) | Выражение не является формулой.  Ошибка на 6-ом символе.  Верно введено:  ((x+[ |
| 2  ({[([{x}+y]-z)+x]-y}+z) | Выражение является формулой. |
| 2  (x&y) | Выражение не является формулой.  Ошибка на 3-ом символе.  Верно введено:  (x |
| 2  ((x+y))) | Выражение не является формулой.  Ошибка на 9-ом символе.  Верно введено:  ((x+y))) |
| 1  Tests/file1.txt  (в файле находится <(x+y)>) | Выражение является формулой. |

**Вывод.**

В процессе выполнения лабораторной работы была изучена структура данных – стек. А также созданы методы для работы с этой структурой данных. Был использован синтаксический анализатор с использованием стека.

**Приложение 1. Код программы.**

* **[test.sh](https://github.com/makometr/AiSD/pull/37/files" \l "_blank)**

gcc ./Sourse/main.c

echo -e '\_\_\_\_\_\_\_\nTest 1:'

cat ./Tests/test1.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./a.out < Tests/test1.txt

echo -e ''

echo -e '\_\_\_\_\_\_\_\nTest 2:'

cat ./Tests/test2.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./a.out < ./Tests/test2.txt

echo -e ''

echo -e '\_\_\_\_\_\_\_\nTest 3:'

cat ./Tests/test3.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./a.out < ./Tests/test3.txt

echo -e ''

echo -e '\_\_\_\_\_\_\_\nTest 4:'

cat ./Tests/test4.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./a.out < ./Tests/test4.txt

echo -e ''

echo -e '\_\_\_\_\_\_\_\nTest 5:'

cat ./Tests/test5.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./a.out < ./Tests/test5.txt

echo -e ''

echo -e '\_\_\_\_\_\_\_\nTest 6:'

cat ./Tests/test6.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./a.out < ./Tests/test6.txt

echo -e ''

echo -e '\_\_\_\_\_\_\_\nTest 7:'

cat ./Tests/test7.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./a.out < ./Tests/test7.txt

echo -e ''

echo -e '\_\_\_\_\_\_\_\nTest 8:'

cat ./Tests/test8.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./a.out < ./Tests/test8.txt

* **main[.](https://github.com/makometr/AiSD/pull/37/files" \l "_blank)c**

#include <stdio.h>

#include <stdlib.h>

struct elem //Структура элемента стека

{

struct elem\* head;

struct elem\* tail;

char sim;

};

int IsEmpty(struct elem\* stack) //Функция проверяющая стек на пустоту

{

if(stack==NULL)

return 0;

return 1;

}

int push(struct elem\*\* stack,char ch) //Функция добавляющая элемент в конец стека

{

if(IsEmpty(\*stack)==0) //Если стек пуст,создаем структуру,заполняем ее,запихиваем в стек

{

struct elem \*a=(struct elem\*)calloc(1,sizeof(struct elem));

\*stack=a;

a->head=NULL;

a->tail=NULL;

a->sim=ch;

}

else //В обратном случае доходим до конца стека и новую заполненную структуру запихиваем в конец стека

{

struct elem\* el=\*stack;

struct elem \*a=(struct elem\*)calloc(1,sizeof(struct elem));

while(el->tail!=NULL)

el=el->tail;

el->tail=a;

a->head=el;

a->tail=NULL;

a->sim=ch;

}

}

char top(struct elem\*\* stack) //Функция возвражающая символ последнего элемента стека

{

if(IsEmpty(\*stack)==0) //Если стек пустой позвращаем символ $

return '$';

struct elem\* el=\*stack;

while(el->tail!=NULL)

el=el->tail;

return el->sim;

}

int pop(struct elem\*\* stack) //функция удаляющая последний элемент стека

{

if(IsEmpty(\*stack)==0)

return 1;

struct elem\* el=\*stack;

while(el->tail!=NULL) //доходим до конца стека

el=el->tail;

if(el->head!=NULL) //если в стеке более 1 елемента

{

el->head->tail=NULL;

free(el);

}

else //если элемент последний стек обнуляется

{

free(el);

\*stack=NULL;

}

return 0;

}

int destroy(struct elem\*\* stack) //Функция высвобождающая всю память стека

{

if(IsEmpty(\*stack)==0)

return 0;

struct elem\* el=\*stack;

while(el->tail!=NULL)

el=el->tail;

while(el->head!=NULL)

{

el=el->head;

free(el->tail);

}

free(el);

}

int are\_correct(char\* str,char\*\* simbol) //Функция, проверяющая на корректность введенную формулу

{

struct elem\* stack=NULL;

\*simbol=str; //simbol служит счетчиком пройденной строки

int chac=0; //Симол для проверок

while(\*\*simbol!='\0') //Пока не пройдем строку

{

chac=1; //Нужен для проверки , попал ли simbol хоть в один if

if(\*\*simbol=='(' || \*\*simbol=='[' || \*\*simbol=='{') //Если открывающиеся скобки

{

printf("Найдена открывающаяся скобка %c.За ней должна следовать переменная<x;y;z> или открывающаяся скобка <(;{;[>.\n",\*\*simbol);

if(\*(\*simbol+1)=='(' || \*(\*simbol+1)=='x' || \*(\*simbol+1)=='y' || \*(\*simbol+1)=='z' || \*(\*simbol+1)=='{' || \*(\*simbol+1)=='[')//проверка на корректность следующего символа

{

push(&stack,\*\*simbol);

printf(" Следующий символ корректен.\n");

}

else

{

printf(" Следующий символ не корректен.\n Программа закрывается.\n Выводится результат.\n");

destroy(&stack);

return 1;

}

chac=0;

}

if(\*\*simbol==')') //Если закрывающаяся скобка

{

printf("Найдена закрывающаяся скобка %c.За ней должен следовать знак<+;-> или закрывающаяся скобка <);};]>.\n",\*\*simbol);

printf("Так же необходимо,чтобы предыдущая скобка была (.\n");

if((top(&stack)=='(') &&(\*(\*simbol+1)==')' || \*(\*simbol+1)=='}' || \*(\*simbol+1)==']' || \*(\*simbol+1)=='+' || \*(\*simbol+1)=='-' || \*(\*simbol+1)=='\0'))//проверка на корректность следующего символа

{

pop(&stack);

printf(" Следующий символ корректен.\n Предыдущая скобка корректна.\n");

}

else

{

printf(" Следующий символ или предыдущая скобка не корректен.\n Программа закрывается.\n Выводится результат.\n");

destroy(&stack);

return 2;

}

chac=0;

}

if(\*\*simbol=='}') //Если закрывающаяся скобка

{

printf("Найдена закрывающаяся скобка %c.За ней должен следовать знак<+;-> или закрывающаяся скобка <);};]>.\n",\*\*simbol);

printf("Так же необходимо,чтобы предыдущая скобка была {.\n");

if((top(&stack)=='{') &&(\*(\*simbol+1)==')' || \*(\*simbol+1)=='}' || \*(\*simbol+1)==']' || \*(\*simbol+1)=='+' || \*(\*simbol+1)=='-' || \*(\*simbol+1)=='\0'))//проверка на корректность следующего символа

{

pop(&stack);

printf(" Следующий символ корректен.\n Предыдущая скобка корректна.\n");

}

else

{

printf(" Следующий символ или предыдущая скобка не корректен.\n Программа закрывается.\n Выводится результат.\n");

destroy(&stack);

return 3;

}

chac=0;

}

if(\*\*simbol==']') //Если закрывающаяся скобка

{

printf("Найдена закрывающаяся скобка %c.За ней должен следовать знак<+;-> или закрывающаяся скобка <];);}>.\n",\*\*simbol);

printf("Так же необходимо,чтобы предыдущая скобка была [.\n");

if(top(&stack)=='[' &&(\*(\*simbol+1)==')' || \*(\*simbol+1)=='}' || \*(\*simbol+1)==']' || \*(\*simbol+1)=='+' || \*(\*simbol+1)=='-' || \*(\*simbol+1)=='\0'))//проверка на корректность следующего символа

{

printf(" Следующий символ корректен.\n Предыдущая скобка корректна.\n");

pop(&stack);

}

else

{

printf(" Следующий символ или предыдущая скобка не корректен.\n Программа закрывается.\n Выводится результат.\n");

destroy(&stack);

return 4;

}

chac=0;

}

if(\*\*simbol=='+' || \*\*simbol=='-') //Если знак

{

printf("Найден знак %c.За ней должна следовать переменная<x;y;z> или открывающаяся скобка <(;{;[>.\n",\*\*simbol);

if(\*(\*simbol+1)=='(' || \*(\*simbol+1)=='x' || \*(\*simbol+1)=='y' || \*(\*simbol+1)=='z' || \*(\*simbol+1)=='{' || \*(\*simbol+1)=='[')//проверка на корректность следующего символа

{

printf(" Следующий символ корректен.\n");

chac=0;

}

else

{

printf(" Следующий символ не корректен.\n Программа закрывается.\n Выводится результат.\n");

destroy(&stack);

return 5;

}

}

if(\*\*simbol=='x' || \*\*simbol=='y' || \*\*simbol=='z') //Если переменная

{

printf("Найдена переменная %c.За ней должен следовать знак<+;-> или закрывающаяся скобка <);};]>.\n",\*\*simbol);

if(\*(\*simbol+1)==')' || \*(\*simbol+1)==']' || \*(\*simbol+1)=='}' || \*(\*simbol+1)=='+' || \*(\*simbol+1)=='-' || \*(\*simbol+1)=='\0')//проверка на корректность следующего символа

{

printf(" Следующий символ корректен.\n");

chac=0;

}

else

{

printf(" Следующий символ не корректен.\n Программа закрывается.\n Выводится результат.\n");

destroy(&stack);

return 6;

}

}

if(chac!=0) //Если не вошел ни в один if

{

printf("Найден неопределенный в задании символ %c. Программа закрывается.\n Выводится результат.\n",\*\*simbol);

return 7;

}

\*simbol=\*simbol+1;

}

printf("Программа закончила обработку строки.\n");

if(IsEmpty(stack)!=0) //если остались не закрытые скобки

{

printf("Не все открытые скобки были закрыты.\n Программа закрывается.\n Выводится результат.\n");

destroy(&stack);

return 8;

}

printf(" Программа закрывается.\n Выводится результат.\n");

return 0;

}

int main() //Основная функция

{

printf("Введите 1 , если хотите считать строчку из файла, в обратном случае введите любое другое число.\n");

int key; //Переменная является ключем для метода считывания, а после используется как счетчик

scanf("%d",&key);

char str[1000]; //Строка для считывания и обработки

str[0]='1';

char\* stop=str; //Указатель для избавления от \n, и проверки на считывания строки из файла

while(str[0]!='\n')

scanf("%c",&str[0]);

if(key==1) //Если требуется считывание из файла

{

FILE\* file;

key=0;

printf("Введите путь к файлу от текущей дирректории.\n");

fgets(str,1000,stdin);

stop=str;

while(\*stop!='\n' && key<1000) //Избавление от \n

{

stop++;

key++;

}

\*stop='\0';

file = fopen (str,"r"); //Открытие файла

if (file == NULL) //Если файл не открыт

{

printf("Файл %s не был открыт.",str);

return -1;

}

stop = fgets (str,sizeof(str),file); //Считывание строки из файла

if(stop==NULL) //Если строка не считалась

{

printf("Строка файла не считалась.");

return -1;

}

if(fclose(file)==EOF) //Если файл не закрыт

{

printf("Файл не смог закрыться");

return -1;

}

}

else //Если строка вводится вручную

{

printf("Введите выражение.\n\n<формула> ::= <терм>|<терм>+<формула> | <терм>-<формула>\n<терм> ::= <имя> | (<формулa>) | [<формула>] | {<формула>}\n<имя> ::= x | y | z\n\n");

printf("В выражении не должно быть более 100 символов, иначе функция не будет считать выражением первые 1000 символов.\n\n");

fgets(str,1000,stdin); //Считывание

}

stop=str;

key=0;

while(\*stop!='\n' && key<1000) //Избавление от \n

{

stop++;

key++;

}

\*stop='\0';

int result=are\_correct(str,&stop); //Проверка строки

if(result==0) //Если проверка пройдена успешно

printf("\nВыражение является формулой.\n");

else //Если не успешно

{

if(str[0]!='(' && str[0]!='[' && str[0]!='{' && str[0]!='x' && str[0]!='y' && str[0]!='z') //Если ошибка на первом же символе

printf("\nВыражение не является формулой.\nОшибка на %d-ом символе.\nНичего не введено верно.\n",1);

else //В остальных случаях

{

printf("\nВыражение не является формулой.\nОшибка на %d-ом символе.\n",stop-str+2);

str[stop-str+1]='\0'; //Обрезка строки для вывода во куда строка была верна

printf("Верно введено: \n%s\n",str); //Вывод строки

}

}

}