

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МОЭВМ**

**ОТЧЕТ**  
**по лабораторной работе №3**  
**по дисциплине «Алгоритмы и структуры данных»**  
**ТЕМА: СТЕКИ И ОЧЕРЕДИ**

Студент гр. 7381

\_\_\_\_\_

Павлов А.П.

Преподаватель

\_\_\_\_\_

Фирсов М.А.

Санкт-Петербург

2018

### **Цель работы.**

Ознакомиться с такими структурами данных, как стек и очередь и методами работы с ними.

### **Задание.**

Вариант 8.

В заданном текстовом файле F записано логическое выражение (ЛВ) в следующей форме:

$\langle \text{ЛВ} \rangle ::= \text{true} \mid \text{false} \mid (\neg \langle \text{ЛВ} \rangle) \mid (\langle \text{ЛВ} \rangle \wedge \langle \text{ЛВ} \rangle) \mid (\langle \text{ЛВ} \rangle \vee \langle \text{ЛВ} \rangle)$

где знаки  $\neg$ ,  $\wedge$  и  $\vee$  обозначают соответственно отрицание, конъюнкцию и дизъюнкцию. Вычислить (как Boolean) значение этого выражения.

Программа должна считать из файла логическое выражение, используя структуру данных- стек, посчитать ее значение.

### **Основные теоретические положения.**

Стек — абстрактный тип данных, представляющий собой список элементов, организованных по принципу LIFO (англ. last in — first out, «последним пришёл — первым вышел»).

### **Ход работы.**

### **Описание алгоритма.**

### **Общий алгоритм.**

Программа посимвольно считывает входные данные и проверяет их корректность, помещая логические операнды (!, |, &) в стек для операндов, а логические высказывания (true, false) — в стек для логических высказываний. Если входные данные корректны, то вычисляется само логическое выражение, в противном случае выводиться сообщение о некорректности входных данных, стеки очищаются, и программа завершается.

### **Алгоритм вычисления логического выражения.**

Программа вытаскивает из стека операнд элемент, если стек не пуст. Если элемент логическое И (&) или логическое ИЛИ (|), то вытаскиваются из стека 2 логических высказываний и считается результат, последний помещается с стек логических высказываний. Если логический операнд —

логическое отрицание (!), то из стека логических высказываний достается один элемент, считается выражение и результат помещается в стек логических высказываний. Данный алгоритм повторяется до тех пор, пока стек операнд не пуст. В конце программа вытаскивает элемент из стека логических высказываний, который является окончательным результатом.

### **Описаний функций и структур данных.**

Функция **bool analyzer**(Stack<char>& stackOperand, Stack<bool>& stackLogik).

#### Аргументы:

stackOperand – стек логических операндов.

stackLogik – стек логических выражений.

#### Возвращаемое значение:

false – если логическое выражение не корректно.

true – если логическое выражение корректно.

#### Описание:

Функция проверяет на корректность логическое выражение, помещает операнды и логические высказывания в стеки.

Функция **bool calc**(Stack <char> & stackOperand, Stack <bool>& stackLogik)

#### Аргументы:

stackOperand – стек логических операндов.

stackLogik – стек логических выражений.

#### Возвращаемое значение:

false – если логическое выражение – ложь.

true – если логическое выражение – истина.

#### Описание:

Итеративно вычисляет лежащее в стеках логическое выражение.

Функция **char \* readStr**(char \* bufferStr, char ch, int length)

#### Аргументы:

char \* bufferStr – указатель на символьный массив.

char ch – считанный ранее символ.

int length – длина символьного массива

Возвращаемое значение:

char \* bufferStr – указатель на символьный массив.

Описание:

Функция посимвольно считывает данные из входного потока и помещает их в строку.

Функция **bool check()**

Аргументы:

Функция ничего не принимает.

Возвращаемое значение:

true – если считанный символ является одним из следующей последовательности символов (' ', '&', '|', '\n', 'EOF').

false – в ином случае.

Функция **char \* demonstrate(bool tmp, char \* str)** служит для демонстрации работы программы.

Функция **int main()**

Описание:

В главной функции программы происходит вызов функции **analyzer**. Если функция возвращает true, то вызывается функция **calc**, которая считает логическое выражение. Если выражение – истина, то выводится сообщение "Logical expression is true!", иначе – "Logical expression is false!". В противном случае выводится сообщение "Incorrect logical expression!".

### **Используемые структуры данных.**

Шаблонный класс Stack.

Класс содержит в себе указатель на массив типа Type, количество элементов в стеке и размер стека. А также методы: 2 конструктора, деструктор, методы push, pull, isEmpty и pop.

Иллюстрация стека приведена на рис.1. По рисунку видно, что в стеке лежит 3 буквы, первая на выход буква 'с'.

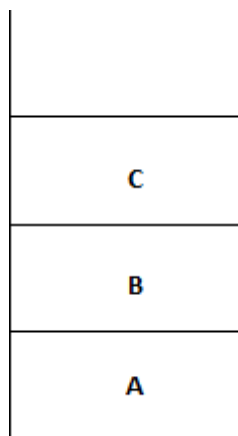


Рисунок 1 – иллюстрация стека.

### Тестирование программы:

Были написаны 6 тестов для данной программы, а также 2 скрипта для тестирования и компиляции программы. Тестирование программы представлено на рис. 2.

Исходные данные:  $!(\text{true} | (\text{false} \& (!\text{true})))$

Результат работы программы:

Reading logical expression...

Push element: !

Push element: 1

Push element: |

Push element: 0

Push element: &

Push element: !

Push element: 1

Calculating logical expression...

Pop element: !

Pop element: 1

Push element: 0

!true = false

-----

Pop element: &

Pop element: 0

Pop element: 0

Push element: 0

false&>false = false

-----

Pop element: |

Pop element: 0

Pop element: 1

Push element: 1

false|true = true

-----

Pop element: !

Pop element: 1

Push element: 0

!true = false

-----

Pop element: 0

false

Logical expression is false!

```
Test 1:  
(!true)  
Logical expression is false!  
  
Test 2:  
(true|false)  
Logical expression is true!  
  
Test 3:  
(!(true|(false&true)))  
Logical expression is false!  
  
Test 4:  
(true|(false|true)  
Incorrect logical expression!  
  
Test 5:  
(true&(false|(!true)))  
Logical expression is false!  
  
Test 6:  
((!true)|(true&false))  
Logical expression is false!
```

Рисунок 2 – тестирование

## **Выводы.**

В результате работы была изучена структура данных – стек. А также созданы методы для работы с этой структурой данных. Был использован алгоритм вычисления логических выражений на основе стека.

## ИСХОДНЫЙ КОД:

### Приложение А: код файла stack.hpp

```
#include <iostream>
#define TEST
template <typename Type>
class Stack
{
private:
    Type *array;           //stack pointer
    size_t size;           //maximum number of items in the stack
    int topElem;           //current stack item number
public:
    Stack();
    Stack(size_t size);
    ~Stack();              //destructor
    void push(Type);        //put an item at the top of the stack
    Type pop();             //remove an item from the top of the stack and
return it
    bool isEmpty();        //check for stack emptiness
};

//Stack constructor
template <typename Type>
Stack<Type>::Stack(){
    array = nullptr;
    size = 0;
    topElem = 0;
}

template <typename Type>
Stack<Type>::Stack(size_t maxSize){
    array = new Type[maxSize]; //allocate stack memory
    size = maxSize;           //stack size initialization
    topElem = 0;              //initializing the current element to zero
}

//destructor
template <typename Type>
Stack<Type>::~~Stack(){
    delete [] array; //remove stack
}

//add item to stack function
template <typename Type>
void Stack<Type>::push(Type value){
#ifdef TEST
    std::cout<<"Push element: "<<value<<std::endl;
#endif
    array[topElem] = value;
    ++topElem;
}
```



```

//If the stack is full, the amount of allocated memory increases in 2 times
if (topElem == size) {
    size_t new_size = size * 2;
    //Copying the contents of the stack to a new memory area
    Type *new_array = new Type[new_size];
    for (size_t index = 0; index < size; ++index)
        new_array[index] = array[index];
    delete[] array; //remove old data

    size = new_size;
    array = new_array;
}
}

//function to remove an item from the stack
template <typename Type>
Type Stack<Type>::pop()
{
    if(isEmpty()){
        std::cout<<"Error: Stack is empty"<<std::endl;
        exit(1);
    }
#ifdef TEST
    std::cout<<"Pop element: "<<array[topElem-1]<<std::endl;
#endif
    return array[--topElem];
}

template <typename Type>
bool Stack<Type>::isEmpty(){
    if(!topElem)
        return true;
    return false;
}

```

### **Приложение Б: код файла main.cpp**

```

#include <iostream>
#include <string>
#include "stack.hpp"
#define TEST

char * readStr(char * bufferStr, char ch, int length){//reading string for false
of true
    bufferStr[0] = ch;
    int i;
    for(i = 1; i < (length - 1); i++)
        bufferStr[i] = std::cin.get();
    bufferStr[i] = '\0';
    return bufferStr;
}

bool check(){

```

```

        char ch = std::cin.get();
        if(ch == ')' || ch == '&' || ch == '|' || ch == '\n' || ch == EOF || ch == '
    '){
            std::cin.putback(ch);
            return true;
        }
        else return false;
    }

char * demonstrate(bool tmp, char * str){//function for demonstration
    if(tmp)
        strcpy(str, "true");
    else
        strcpy(str, "false");
    return str;
}
//logical expression validation function
bool analyzer(Stack <char> & stackOperand, Stack <bool>& stackLogik){
    char ch;
    char * bufferStr = nullptr;
    while((ch = std::cin.get()) == ' '){//ignore spaces
        ;
    }
    if(ch == 't')
    {
        bufferStr = new char[6];
        if(!strcmp(readStr(bufferStr, ch, 5), "true"))
        {
            delete [] bufferStr;
            if(check()){
                stackLogik.push(true);
                return true;
            }
            return false;
        }
        else{
            delete [] bufferStr;
            return false;
        }
    }
    else if(ch == 'f')
    {
        bufferStr = new char[6];
        if(!strcmp(readStr(bufferStr, ch, 6), "false"))
        {
            delete [] bufferStr;
            if(check()){
                stackLogik.push(false);
                return true;
            }
            return false;
        }
    }
}

```

```

        else{
            delete [] bufferStr;
            return false;
        }
    }
    else if(ch == '('){
        while((ch = std::cin.get()) == ' ')//ignore spaces
            ;
        if(ch == '!')
        {
            stackOperand.push(ch);
            if(analyzer(stackOperand, stackLogik))
            {
                while((ch = std::cin.get()) == ' ')//ignore spaces
                    ;
                if(ch == ')')
                    return true;
                else return false;
            }
            else return false;
        }
    }
    else{
        std::cin.putback(ch);
        if(analyzer(stackOperand, stackLogik))
        {
            while((ch = std::cin.get()) == ' ')//ignore spaces
                ;
            if(ch == '&' || ch == '|')
            {
                stackOperand.push(ch);
                if(analyzer(stackOperand, stackLogik))
                {
                    while((ch = std::cin.get()) == ' ')
                        ;
                    if(ch == ')')
                        return true;
                    else return false;
                }
                else return false;
            }
            else return false;
        }
        else return false;
    }
}
else return false;
}

```

```

bool calc(Stack <char> & stackOperand, Stack <bool>& stackLogik){
#ifdef TEST
    std::cout<<"Calculating logical expression..."<<std::endl;

```

```

#endif
    char * str1 = new char[6];//for
    char * str2 = new char[6];//
    char * str3 = new char[6];//
    bool tmp1, tmp2;          //demonstate
    char operand;
    bool result;
    while(!stackOperand.isEmpty()){
#ifdef TEST
        std::cout<<"-----"<<std::endl;
#endif
        if((operand = stackOperand.pop()) == '&'){
            result = ((tmp1 = stackLogik.pop()) & (tmp2 =
stackLogik.pop()));
            stackLogik.push(result);
#ifdef TEST
                std::cout<<demonstate(tmp1,
str1)<<operand<<demonstate(tmp2, str2)<< '='<<demonstate(result,
str3)<<std::endl;
#endif
            }
            else if(operand == '|'){
                result = ((tmp1 = stackLogik.pop()) | (tmp2 =
stackLogik.pop()));
                stackLogik.push(result);
#ifdef TEST
                    std::cout<<demonstate(tmp1,
str1)<<operand<<demonstate(tmp2, str2)<< '='<<demonstate(result,
str3)<<std::endl;
#endif
            }
            else{
                result = !(tmp1 = stackLogik.pop());
                stackLogik.push(result);
#ifdef TEST
                    std::cout<<operand<<demonstate(tmp1,
str1)<< '='<<demonstate(result, str3)<<std::endl;
#endif
            }
        }
        if(!stackLogik.isEmpty()){
#ifdef TEST
            std::cout<<"-----"<<std::endl;
#endif
            result = stackLogik.pop();
#ifdef TEST
                std::cout<<demonstate(result, str3)<<std::endl;
#endif
        }
        delete [] str1;
        delete [] str2;

```

```

        delete [] str3;
        return result;
    }

int main(){
    char ch;
    while((ch = std::cin.get()) == ' '){//ignore spaces
        ;
    }
    if(ch == '\n' || ch == EOF){ //if input is empty
        std::cout<<"Input is empty"<<std::endl;
        return 0;
    }
    else
        std::cin.putback(ch);
    Stack <char> stackOperand(2);//operand stack initialization
    Stack <bool> stackLogik(2); //stack initialization for logical
statements: true, false
#ifdef TEST
    std::cout<<"Reading logical expression..."<<std::endl;
#endif
    if(analyzer(stackOperand, stackLogik)){
        if(calc(stackOperand, stackLogik))
            std::cout<<"Logical expression is true!"<<std::endl;
        else
            std::cout<<"Logical expression is false!"<<std::endl;
    }
    else
        std::cout<<"Incorrect logical expression!"<<std::endl;
}

```

#### **Приложение В: код файла compile.bat**

```

@echo off
cl /EHsc .\Source\main.cpp

```

#### **Приложение В: код файла runTests.bat**

```

@echo off
cl /EHsc .\Source\main.cpp
echo Test 1:
type .\Tests\Test1.txt
echo.
main < .\Tests\Test1.txt
echo.
echo Test 2:
type .\Tests\Test2.txt
echo.
main < .\Tests\Test2.txt
echo.
echo Test 3:
type .\Tests\Test3.txt
echo.
main < .\Tests\Test3.txt
echo.
echo Test 4:

```

```
type .\Tests\Test4.txt
echo.
main < .\Tests\Test4.txt
echo.
echo Test 5:
type .\Tests\Test5.txt
echo.
main < .\Tests\Test5.txt
echo.
echo Test 6:
type .\Tests\Test6.txt
echo.
main < .\Tests\Test6.txt
echo.
```