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. 8381 _____ **Переверзев Д.Е.**

Преподаватель _____ **Жангиров Т.Р.**

Санкт-Петербург

2019

Цель работы

Ознакомиться с методами вычисления выражения в префиксов форме и реализовать на практике.

Задание

Вариант 19

Рассматривается выражение следующего вида:

< выражение >::= < терм > | < терм > + < выражение > |
< терм > — < выражение >

< терм > ::= < множитель > | < множитель > * < терм >

< множитель > ::= < число > | < переменная > | (< выражение >) |
< множитель > ^ < число >

< число > ::= < цифра >

< переменная > ::= < буква >

Такая форма записи выражения называется ***инфиксной***.

Постфиксной (префиксной) формой записи выражения *aDb* называется запись, в которой знак операции размещен за (перед) операндами: *abD* (*Dab*).

Примеры

Инфиксная	***Постфиксная***	***Префиксная***
a-b	*a-b*	*a-b*
*a*b+c*	*ab*c+*	*+*abc*
a(b+c)*	*abc+**	**a+bc*
*a+b^c^d*e*	*abc^d^e*+*	*+a*^b^cde*.

Отметим, что постфиксная и префиксная формы записи выражений не содержат скобок.

Требуется:

б) вычислить как целое число значение выражения (без переменных), записанного в префиксной форме (задан текстовый файл *prefix*);

Выполнение работы

1. Создание функций:
   - void shift(int *&str, int ind, char simb, int len) - замняет 3 символа на 1;
   - int isnum(char simb)- определяет число или оператор передан ей;
   - int which_oper(int oper, int num1, int num2) - определят оператор;
   - int *step(int *expression, int *j, int *l) - один шаг преобразования;
   - int step_by_step(int *expression, int *len) - все шаги сразу;
   - int test(string str) - проверка принимаемой строки на корректность;
2. Основной алгоритм вычисления заключается в рассматривании с конца стека по три символа строки, если один из них это оператор, а два других - числа, то вместо эти трех значений записывается одно значение выражения(<число1><оператор><число2>). Дальше рассматриваются следующий набор из трех значений. Так продолжается, стек не будет содержать только один элемент.
3. Был разработан WEB GUI. Серверная часть была написана на node.js, для обработки строки были написаны методы на c++. Для клиентской части использовались язык разметки HTML, язык таблиц стилей CSS, JavaScript для обработки действий на странице и передачи данных без обновления страницы с помощью объекта XMLHttpRequest.
4. В зависимости от передаваемых аргументов при запуске программы выполняются разные действия:
   - gui — открытие WEB GUI;
   - console — продолжается работа с консолью;
   - step — выполнение преобразований по шагам;
   - all_steps — выполнение всех шагов преобразования сразу;
5. При работе с консолью реализован ввод выражения из файла.

Оценка эффективности алгоритма

Алгоритм имеет линейную сложность, так как алгоритм проходит по массиву только один раз.

Тестирование программы

Входная выражение	Строка после форматирования
+++2222	8
9	9
10	ERROR
-*++^-+*+5621370543	825032721
+-8*765	-29

Выводы

В ходе лабораторной работы был изучен способ вычисления выражения в префиксов форме. А также реализован WEB GUI.

Приложение А

Исходный код программы.

main.cpp

```
#include "lr_3_methods.h"
#include "lr_3_methods.cpp"

int main(int argc, char *argv[])
{
   bool flag_step=false;
   bool flag_gui=false;
   for(int i=1;i<argc;i++)
   {
      if(!strcmp("gui",argv[i]))
         flag_gui=true;
      if(!strcmp("step",argv[i]))
         flag_step=true;
      if(!strcmp("console",argv[i]))
         flag_gui=false;
      if(!strcmp("all_steps",argv[i]))
         flag_step=false;
   }

   if(flag_gui)
   {
      system("open http://163.172.163.152:3000/");
      return 0;
   }
   int i = 0;
   int *prefix_arr;
   int len;
   //input
   string prefix;


   while(true)
   {
      ifstream in("prefix.txt");

      cout<<"Read expression from file?(y/n): ";
      cout<<prefix;
      char y_n='y';
      cin>>y_n;
      if(y_n=='n')
         continue;
      getline(in,prefix);
```

```
    int err = test(prefix);
    if (!err)
    {
        i = 0;
        int j = 0;
        len = prefix.size();
        prefix_arr = new int[len];
        for (int elem : prefix)
            prefix_arr[j++] = elem;
        break;
    }
    else
    {
        system("clear");
        cout<<"#ERROR#\nWrong expression\nPlease repeat\n";

    }
}
//step
string ret;
if(flag_step)
{
    do{
        ret = " ";

        char y_n;
        cout<<"Continue?(y/n)\n";
        cin>>y_n;
        if(y_n=='n')
            break;
        prefix_arr = step(prefix_arr, &i, &len);
        for (int j = 0; j < len; j++)
        {
            if (isnum(prefix_arr[j]) == 0)
            {
                ret += (char)prefix_arr[j];
                ret += ' ';
            }
            else
            {
                ret += to_string(prefix_arr[j] - '0') + ' ';
            }
        }
        cout<<ret<<endl;
```

```
    }while(count(ret.begin(),ret.end(),' ')>2);
    return 0;
  }

  //all steps
  cout<<"Meaning of expression: "<<step_by_step(prefix_arr, &len)<<endl;



  return 0;
}
```

lr_3_methods.cpp

```
#include "lr_3_methods.h"
void shift(int *&str, int ind, int simb, int len)
{
  str[ind] = simb;
  for (; ind < len; ind++)
    str[ind - 2] = str[ind];
}

int isnum(char simb)
{
  if (simb == '*' || simb == '+' || simb == '^' || simb == '-')
    return 0;
  return 1;
}

int which_oper(int oper, int num1, int num2)
{
  switch (oper)
  {
  case '+':
    return '0' + (num1 - '0') + (num2 - '0');
    break;
  case '-':
    return '0' + (num1 - '0') - (num2 - '0');
    break;
  case '*':
    return '0' + (num1 - '0') * (num2 - '0');
    break;
  case '^':
    return '0' + pow((num1 - '0'), (num2 - '0'));
    break;

  default:
```

```
            break;
        }
        return '1';
    }

    int *step(int *expression, int *j, int *l)
    {
        int i = *j;
        int len = *l;
        if (len < 2)
            return expression;
        if (i == 0)
            i = len - 1;
        if (isnum(expression[i]) == 1 && isnum(expression[i - 1]) == 1 && isnum(expression[i - 2]) ==
0)
        {
            shift(expression, i, which_oper(expression[i - 2], expression[i - 1], expression[i]), len);
            len -= 2;
            i -= 2;
        }
        i--;
        *j = i;
        *l = len;
        return expression;
    }

    int step_by_step(int *expression, int *len)
    {
        int i = 0;
        int *ret = step(expression, &i, len);
        while (true)
        {
            ret = step(ret, &i, len);
            if (*len < 2)
                return (int)ret[0] - '0';
        }
    }

    int test(string str)
    {

        int num_quantity = 0;
        int oper_quantity = 0;
        for (int i = 0; i < str.length(); i++)
        {
            if (isnum(str[i]) == 1 && str[i] < 48 && str[i] > 57)
```

```
        {
            cout << "Error 4\n";
            return str[i];
        }
        if (isdigit(str[i]))
            num_quantity++;
        else
        {
            oper_quantity++;
            //3
            if (str[i] != '^' && str[i] != '+' && str[i] != '*' && str[i] != '-')
            {
                cout << "Error 3\n";
                return 3;
            }
        }
    }
    //2
    if (isdigit(str[0]) && str.length() > 1)
    {
        cout << "Error 2\n";
        return 2;
    }
    //1
    if (oper_quantity + 1 != num_quantity)
    {
        cout << "Error 1\n";
        return 1;
    }
    return 0;
}
```

lr_3_methods.h

```
#include <iostream>
#include <string>
#include <cmath>
#include <cstdlib>
#include <fstream>

using namespace std;

void shift(int *&str, int ind, char simb, int len);

int isnum(char simb);
```

```
int which_oper(int oper, int num1, int num2);

int *step(int *expression, int *j, int *l);

int step_by_step(int *expression, int *len);

int test(string str);
```