МИНИСТЕРСТВО НАУКИ И ВЫСШЕГООБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Национальный исследовательский
Нижегородский государственный университет им. Н.И. Лобачевского»
(ННГУ)**

Институт информационных технологий, математики и механики

ЛАБОРАТОРНАЯ РАБОТА

на тему:

«Постфиксная форма записи арифметических выражений»

Выполнил(а): студент группы 3822Б1ФИ2
_____ / Хохлов А.Д./
Подпись

Проверил: к.т.н, доцент каф. ВВиСП
_____ / Кустикова В.Д./
Подпись

Нижний Новгород
2023

Содержание

Введение

В современном мире информационных технологий большую роль играют арифметические операции. Одной из важных операций является эффективная работа с постфиксными формами записи. Постфиксная форма записи играет важную роль в многих областях информатики.

Знание и понимание структуры и принципов хранения помогают оптимизировать использование памяти и увеличивать эффективность вычислений.

Таким образом, данная лабораторная работа является актуальной и полезной для студентов и специалистов в области информационных технологий, которые имеют необходимость эффективно работать с битами и битовыми множествами.

1 Постановка задачи

Цель: Целью данной лабораторной работы является создание структуры хранения и перевода инфиксной формы в постфиксную на языке программирования C++. В рамках работы необходимо разработать классы Stack, Lexems и Arifmetics, которые будут предоставлять функциональность для работы с инфиксной формой записи. Основной задачей является реализация основных операций с инфиксной формой записи в постфиксную.

Задачи данной лабораторной работы:

- Разработка класса Stack.
- Реализация основных операций с стэком: push, pop, check, проверка на пустоту, очистка.
- Определение класса Lexems.
- Реализация основных операций с лексемами, включая разбиение на константы, переменные и операторы.
- Определение класса Arifmetics.
- Реализация основных операций для перевода инфиксной формы в постфиксную.
- Проверка и демонстрация работы разработанных классов с помощью приложений.
- Написание отчета о выполненной лабораторной работе, включая описание алгоритмов, программной реализации и результатов работы.

2 Руководство пользователя

2.1 Приложение для демонстрации работы стека

1. Запустите приложение с названием sample_tstack.exe. В результате появится окно, показанное ниже (рис. 1).

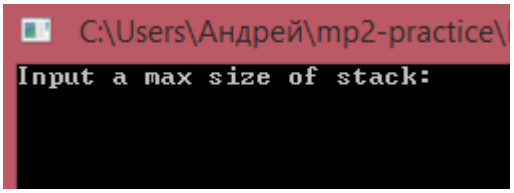

Рис. 1. Основное окно программы

2. Это окно показывает работу основных функций работы со стеком (проверка на пустоту, получение верхнего элемента, удаление элемента). Для продолжения введите максимальный размер стека, количество элементов и сами элемета. В результате будет выведено (рис. 2). Для выхода нажмите любую клавишу.

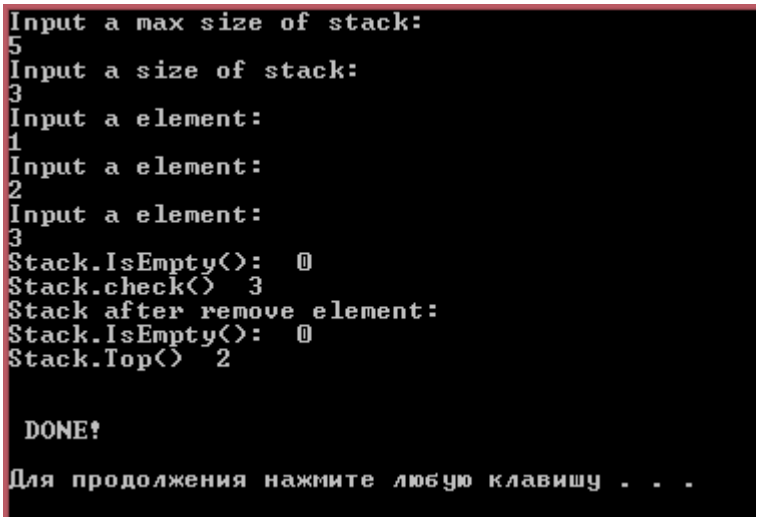

Рис. 2. Основное окно программы

2.2 Приложение для демонстрации работы перевода в постфиксную форму

1. Запустите приложение с названием sample_postfix_form.exe. В результате появится окно, показанное ниже (рис. 3). Это окно показывает работу основных функций перевода инфиксной формы в постфиксную.

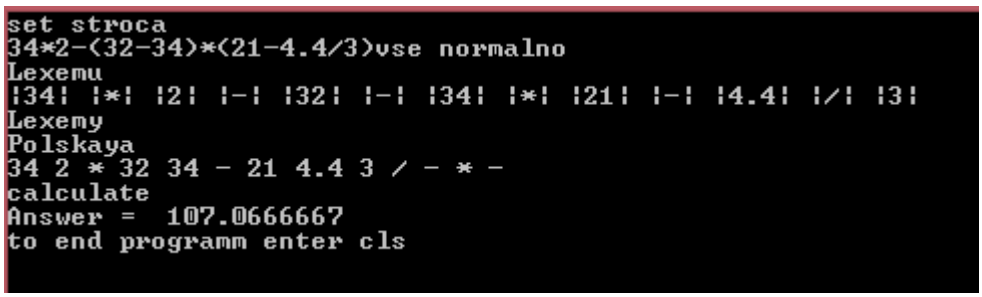

Рис. 3. Основное окно программы

3 Руководство программиста

3.1 Описание алгоритмов

3.1.1 Стек

Стек — это способ организации данных в компьютерной программе, при котором элементы добавляются и удаляются с одного конца (вершины стека), а доступ к элементам осуществляется только через другой конец (основание стека). Таким образом, стек работает по принципу "последним пришел — первым вышел" (LIFO).

Операция добавления в конец:

Операция добавляет новый элемент в вершину стека.

Пример:

V = {1, 2, 3, 4, 5}

Добавление нового элемента:

elem = 2

V + elem= {1, 2, 3, 4, 5, 2}

Операция удаления элемента с верхушки стека:

Операция удаляет последний элемент с верхушки стека.

Пример:

V = {1, 2, 3, 4, 5}

После удаления:

V = {1, 2, 3, 4}

Операция проверки верхушки стека:

Операция возвращает последний элемент

Пример:

V = {1, 2, 3, 4, 5}

Результат выполнения операции:

A = 5

Операция проверки на пустоту:

Операция показывает наличие элементов в стеке. Возвращает 1 если он пустой, в противном случае 0.

Пример:

V = {0, 1, 2, 3, 4}

Результат выполнения операции:

0

3.1.2 Постфиксная форма

Постфиксная форма арифметического выражения — это формат записи математического выражения, в котором операторы стоят после всех операндов. Например, вместо записи "3 + 4" используется "3 4 +". Постфиксная форма позволяет более ясно выразить порядок выполнения операций, поскольку операторы указываются после всех операндов.

Алгоритм, о котором ты говоришь, принимает на вход строку, которая содержит арифметическое выражение, и хэш-таблицу, в которой хранятся операнды этого выражения. Затем алгоритм использует математические правила для определения приоритета операций: сначала он выполняет операции в скобках, затем умножение и деление, а затем сложение и вычитание.

Пример перевода инфиксной формы в постфиксную:

Инфиксная форма: A+(B+C)

Постфиксная форма: ABC++

Операция получения инфиксной записи:

Возвращает исходное выражение.

Операция получения постфиксной записи:

Алгоритм:

- Проходим исходную строку;
- При нахождении **числа**, заносим его в выходную строку;
- При нахождении **оператора**, заносим его в стек;
- Выталкиваем в выходную строку из стека все операторы, имеющие приоритет выше рассматриваемого;
- При нахождении открывающейся скобки, заносим её в стек;
- При нахождении закрывающей скобки, выталкиваем из стека все операторы до открывающейся скобки, а открывающуюся скобку удаляем из стека.

Пример: x - y / (5 * z) + 10

Результат: x y 5 z * / - 10 +

3.2 Описание программной реализации

3.2.1 Описание класса Stack

```
class Stack {
private:
      ValType * arr;
      int SM;
      int ind;
public:
      Stack(int Size_Max);
      ~Stack();
      Stack(const Stack& a);
      void push(const ValType a);
      ValType pop();
      ValType check();
      bool IsEmpty();
      int Size();
      void Clear();
      void Print();
      int getInd();
      int GetSizeMax();
};
```

Назначение: представление стека.

Поля:

SM – максимальный размер стека.

ind – текущее колличество элементов.

arr – память для представления стека.

Методы:

Stack(int Size_Max)

Назначение: конструктор с параметром.

Входные параметры:

~Stack();

Назначение: деструктор.

Stack(const Stack& a);

Назначение: конструктор копирования.

Выходные параметры:

a – копируемый стек.

void push(const ValType a);

Назначение: добавление элемента в начало.

Выходные параметры:

a – новый элемент.

ValType pop();

Назначение: удаление элемента с верхушки стека.

ValType check();

Назначение: получение значения элемента с верхушки стека.

Выходные параметры:

Значение элемента.

bool IsEmpty();

Назначение: проверка стека на пустоту.

Выходные параметры:

Результат проверки.

int Size();

Назначение: получение текущего размера стека.

Выходные параметры:

Значение максимального размера стека.

```
void Clear();
```
Назначение: очистка стека.

```
void Print();
```
Назначение: вывод стека.

```
int getInd();
```
Назначение: получение значения текущего количества элементов.

Выходные параметры:

Значение текущего количества элементов.

```
int GetSizeMax();
```
Назначение: получение максимального размера стека.

Выходные параметры:

Значение максимального размера стека.

3.2.2 Описание класса Lexems

```
class Lexems {
private:
      int length;
      int Type;
      double number;
      char Name[5];
public:
      void Printvar();
      int GetType();
      char* GetName();
      double GetNumber();
      int GetLength();
      void SetType(int i);
      void SetChar(char* a);
      void SetNumber(double a);
      void SetLength(int i);
};
```

Назначение: хранение структуры данных лексем.

Поля:

```
Length – длина.
Type – тип лексемы.
Number – константа.
Name – название лексемы.
```

Методы:

```
void Printvar();
```
Назначение: печать переменной.

`int GetType();`

Назначение: получение типа лексемы.

Выходные параметры: номер типа лексемы.

`char* GetName();`

Назначение: получение имени лексемы.

Выходные параметры: название лексемы.

`double GetNumber();`

Назначение: получение константы.

Выходные параметры: константа.

`int GetLength();`

Назначение: получение длины лексемы.

Выходные параметры: длина лексемы.

`void SetType(int i);`

Назначение: установка типа лексемы.

Входные параметры:

`i` – индекс типа лексемы.

`void SetChar(char* a);`

Назначение: установка имени лексемы.

Входные параметры:

`a` – имя.

`void SetNumber(double a);`

Назначение: установка константы.

Входные параметры:

`a` – константа.

`void SetLength(int i);`

Назначение: установка длины лексемы.

Входные параметры:

`i` – длина лексемы.

3.2.3 Описание класса Arifmetics

```
class Arifmetics {
private:
      const char priority[6] = { ')','(','+','-','*','/' };
      const int m_priority[6] = { -1,0,1,1,2,2 };
      bool error = false;
      char onestring[1000];
      Stack<int> a;
      Stack <int> b;
      Lexems * c;
      Lexems * d;
public:
      Lexems* GetPolish() {
            return d;
      }
      char* GetOnestring();
      int  SetVariable();
      double calculate();
      int Num(int i);
      Arifmetics();
      void Lexem();
      void Polsky();
      void PrintString();
      int getPriority(int z);
      bool Check();
      void PrintPolsky();
      void PrintLexem();
      void SetStroka();
      void PushStroka(char* a);
};
```

Назначение: перевод арифметического выражения в постфиксную запись.

Поля:

`const char priority[6]` – операторы.

`const int m_priority[6]` – приоритет операторов.

`char onestring[1000]` – строка констант.

`Stack<int> a` – стек для операторов.

`Stack <int> b` – стек для операторов.

`Lexems * c` – массив лексем.

`Lexems * d` – массив лексем.

Методы:

```
Lexems* GetPolish();
```

Назначение: перевод выражения в постфиксную форму.

Выходные параметры: постфиксная запись.

```
char* GetOnestring();
```

Назначение: получение строки лексем.

Выходные параметры: строка лексем.

`int  SetVariable();`

Назначение: установка переменных.

Выходные параметры: успешность установки.

`double calculate();`

Назначение: вычисление выражения в постфиксной форме.

Выходные параметры: результат вычисления.

`int Num(int i);`

Назначение: возвращает элемент строки констант по индексу.

Входные параметры:

`i` – индекс.

Выходные параметры: константа.

`Arifmetics();`

Назначение: конструктор по умолчанию.

`void Lexem();`

Назначение: присваивание приоритета лексем.

`void Polsky();`

Назначение: перевод в постфиксную запись.

`void PrintString();`

Назначение: печать выражения.

`int getPriority(int z);`

Назначение: получение значения приоритета.

Входные параметры:

`z` – индекс.

Выходные параметры: значение приоритета.

`bool Check();`

Назначение: проверка на корректность записи.

Выходные параметры: результат проверки.

`void PrintPolsky();`

Назначение: печать постфиксной формы.

`void PrintLexem();`

Назначение: печать лексем.

```
void SetStroka();
```
Назначение: установка строки.

```
void PushStroka(char* a);
```
Назначение: добавление строки в стек.

Заключение

В ходе выполнения работы "Постфиксная форма записи арифметических выражений" были изучены и практически применены концепции стека и перевода инфиксной формы в постфиксную.

Были достигнуты следующие результаты:

1. Были изучены теоретические основы стека и алгоритма перевода в постфиксую форму

2. Была разработана программа, реализующая необходимые операции. В ходе экспериментов была оценена эффективность работы этих операций и сравнена с другими подходами.

3. Были проанализированы полученные результаты и сделаны выводы о преимуществах и ограничениях использования стека. Оказалось, что эти структуры данных особенно полезны при работе с большими объемами данных, где компактность представления и эффективность операций являются ключевыми факторами.

Литература

1. Сысоев А.В., Алгоритмы и структуры данных, лекция 07, 17 октября.

Приложения

Приложение А. Реализация класса Stack

```
template <class ValType>
int Stack <ValType> ::GetSizeMax() {
      return SM;
}
template <class ValType>
Stack <ValType> ::Stack(int Size_Max)
{
      SM = Size_Max;
      ind = -1;
      arr = new ValType[SM];
}

template <class ValType>
Stack <ValType> ::~Stack()
{
      delete[] arr;
}
template <class ValType>
Stack <ValType> ::Stack(const Stack& a)
{
      SM = a.SM;
      arr = new ValType[SM];
      for (int i = 0; i <= ind; i++)
            arr[i] = a.arr[i];
}

template <class ValType>
void Stack <ValType> ::push(const ValType a) {
      if (ind + 1 >= SM) {
            Stack <ValType> b(*this);
            delete[] arr;
            SM = SM * 2;
            arr = new ValType[SM];
            for (int i = 0; i <= ind; i++) arr[i] = b.arr[i];
            arr[++ind] = a;
      }
      else
            arr[++ind] = a;
}
template <class ValType>
ValType Stack <ValType> ::pop() {
      if (ind == -1) throw "negative"; else
            return arr[ind--];
}
template <class ValType>
ValType Stack <ValType> ::check() {
      if (ind == -1) return 0; else
            return arr[ind];
}

template <class ValType>
bool  Stack <ValType> ::IsEmpty() {
      if (ind == -1) return 1; else return 0;
}

template <class ValType>
int  Stack <ValType> ::Size() {
      return (ind + 1);
```

```
}
template <class ValType>
void Stack <ValType> ::Clear() {
      ind = -1;
}

template <class ValType>
void Stack <ValType> ::Print() {
      for (int i = 0; i <= ind; i++) cout << arr[i] << endl;
}
template <class ValType>
int Stack <ValType> ::getInd() {
      return ind;
}
```

Приложение Б. Реализация класса Lexems

```
void Lexems::Printvar() {
      for (int i = 0; i < strlen(Name); i++) cout << Name[i];
      cout << endl;
}
void Lexems::SetLength(int i) {
      length = i;
}
int Lexems::GetLength() {
      return length;
}

double Lexems::GetNumber() {
      return number;
}
char* Lexems::GetName() {
      return Name;
}
void Lexems::SetNumber(double a) {
      number = a;
}

void Lexems::SetChar(char* a) {
      if (strlen(a) < 5)
            strcpy(Name, a); else cout << "slischom big name peremennoy";
}
void Lexems::SetType(int i) {
      Type = i;
}

int Lexems::GetType() {
      return Type;
}
```

Приложение В. Реализация класса Arifmetics

```
void Arifmetics::SetStroka() {
      cout << "set stroca" << endl;
      cin >> onestring;
}
void Arifmetics::PushStroka(char* a) {
      strcpy(onestring, a);
      cout << endl;
}
char* Arifmetics::GetOnestring() {
      return onestring;
```

```
}
int Arifmetics::SetVariable() {
      int t = 0;
      for (int i = 0; i < c[0].GetLength(); i++)

            if (c[i].GetType() == -3) {
                  t = 1;
                  double k;
                  int y = 0;

                  for (int j = 0; j < i; j++) {

                        if ((c[j].GetType() == -3) && (!strcmp(c[j].GetName(),
c[i].GetName()))) {
                              c[i].SetNumber(c[j].GetNumber());
                              y = 1;
                              break;
                        }

                        if ((c[j].GetType() == -3) && (!strcmp(c[j].GetName(),
&c[i].GetName()[1]))) {
                              c[i].SetNumber(-1 * c[j].GetNumber());
                              y = 1;
                              break;
                        }
                        if       ((c[j].GetType()       ==       -3)       &&
(!strcmp(&c[j].GetName()[1], c[i].GetName()))) {
                              c[i].SetNumber(-1 * c[j].GetNumber());
                              y = 1;
                              break;
                        }
                  }

                  if (y == 0) {
                        if ((c[i].GetName())[0] != '-') {
                              cout << "vvedite peremennuu ";
                              c[i].Printvar();
                              cin >> k;
                        }
                        else {
                              cout << "vvedite peremennuu ";
                              char a[10] = "";
                              strcpy(a, c[i].GetName());
                              string b(a);
                              b.erase(0, 1);
                              int i = 0;
                              while (i < b.length()) {
                                    a[i] = b[i];
                                    i++;
                              }
                              a[i] = '\0';
                              c[i].SetChar(a);
                              c[i].Printvar();
                              cin >> k;
                              k = k * (-1);
                        }
                        c[i].SetNumber(k);
                  }
```

```
            }
      return t;
}
void Arifmetics::PrintLexem() {
      cout << endl;
      cout << "Lexemu " << endl;
      for (int i = 0; i < c[0].GetLength(); i++)
            switch (c[i].GetType()) {
            case 1: {
                  cout << "+ ";
                  break;
            }
            case 2: {
                  cout << "|-| ";
                  break;
            }
            case 3: {
                  cout << "|*| ";
                  break;
            }
            case 4: {
                  cout << "|/| ";
                  break;
            }
            case -2: {
                  cout << "|";
                  cout.precision(10);
                  cout << c[i].GetNumber();
                  cout << "| ";
                  break;
            }
            case -3: {
                  cout << c[i].GetName() << " ";
                  break;
            }
            {
            default:
                  break;
            }
            }
      cout << endl << "Lexemy" << endl;
}
void Arifmetics::PrintPolsky() {
      for (int i = 0; i < d[0].GetLength(); i++)
            switch (d[i].GetType()) {
            case 1: {
                  cout << "+ ";
                  break;
            }
            case 2: {
                  cout << "- ";
                  break;
            }
            case 3: {
                  cout << "* ";
                  break;
            }
            case 4: {
                  cout << "/ ";
                  break;
            }
            case -2: {
                  cout << d[i].GetNumber() << " ";
```

```
                  break;
            }
            case -3: {
                  cout << d[i].GetName() << " ";
                  break;
            }
            {
            default:
                  break;
            }
            }
}
void Arifmetics::Polsky() {
      cout << "Polskaya" << endl;
      int i = 0;
      int q = 0;
      while (i < c[0].GetLength()) {
            switch (c[i].GetType())
            {
            case -3: {
                  d[q++] = c[i];
                  break;
            }
            case -2: {
                  d[q++] = c[i];
                  break;
            }
            case -1: {
                  int z = b.getInd();
                  while (b.check() != 0)
                        d[q++].SetType(b.pop());
                  b.pop();

                  break;
            }
            case 0: {
                  b.push(0);
                  break;
            }
            case 1: { //+
                  int z = 0;
                  if (b.check() < 2) b.push(1); else while (b.check() >= 2) {
                        d[q++].SetType(b.pop());
                        z = 1;
                  }
                  if (z == 1) b.push(1);
                  break;
            }
            case 2: { //-
                  int z = 0;
                  if ((b.check() < 2) || (b.IsEmpty())) b.push(2); else while
(b.check() >= 2) {
                        d[q++].SetType(b.pop());
                        z = 1;
                  }
                  if (z == 1) b.push(2);
                  break;
            }
            case 3: {
                  b.push(3);
                  break;
            }
            case 4: {
```

```
                  b.push(4);
                  break;
            }

            default:
                  break;
            }
            i++;
      }

      while (!b.IsEmpty()) d[q++].SetType(b.pop());
      d[0].SetLength(q);
}
int Arifmetics::Num(int i) {
      return onestring[i] - '0';
}
Arifmetics::Arifmetics() : a(1000), b(1000) {
      c = new Lexems[1000];
      d = new Lexems[1000];
}
void Arifmetics::PrintString() {
      for (int i = 0; i < strlen(onestring); i++) cout << onestring[i];
}

bool Arifmetics::Check() {
      int i = 0;
      while ((i < strlen(onestring)) && (error != true)) {

            switch (getPriority(i))
            {
            case 0:
            {
                  a.push(0);
                  if (i > 0) if (getPriority(i - 1) == -2) {
                        error = true;
                        cout << "error on the " << i << "position";
                  }
                  break;
            }
            //(onestring(i+1)=='.')
            case 1:
                  if (i > 0) if ((onestring[i - 1] == '.')) {
                        error = true; cout << "error on the " << i <<
"position";
                  }
                  {              if (i < strlen(onestring) - 1)
                        if ((getPriority(i + 1) >= 1) || (getPriority(i + 1)
== -1) || (onestring[i + 1] == '.')) { error = true; cout << "error on the "
<< i + 2 << "position"; }
                        else a.push(1); else {
                              error = true; cout << "error on the " << i + 1
<< "position";
                        }
                  break;
                  }
            case 2:
            {
                  if (i > 0) if (onestring[i - 1] == '.') {
                        error = true; cout << "error on the " << i <<
"position";
                  }
                  if (i < strlen(onestring) - 1)
```

```
                        if ((getPriority(i + 1) >= 1) || (getPriority(i + 1)
== -1) || (onestring[i + 1] == '.')) {
                              error = true; cout << "error on the " << i + 2
<< "position";
                        }
                        else a.push(2); else {
                              error = true; cout << "error on the " << i + 1
<< "position";
                        }
                  break;
            }

            case -1:
            {
                  if (i < strlen(onestring) - 1)  if (getPriority(i + 1) == -
2) { error = true; cout << "error on the " << i + 2 << "position"; }

                  int i = 0;
                  while ((a.check() != 0) && (!a.IsEmpty()))
                        i = a.pop();
                  if ((a.check() == 0) && (!a.IsEmpty()))
                        i = a.pop();
                  else { error = true; cout << "error more ')' "; }
                  break;
            }

            case -2:
            {
                  if (i > 0) if (getPriority(i - 1) == -3) if (!(onestring[i -
1] == '.')) {
                        error  =  true;  cout  <<  "error  on  the  "  <<  i  <<
"position";
                  }
                  else a.push(-2);
                  else a.push(-2);
                  else if (onestring[i] == '.') { error = true; cout << "error
on the " << i + 1 << "position"; }
                  break;
            }
            case -3:
            {

                  if (i > 0) if ((getPriority(i - 1) == -2) || (onestring[i -
1] == '.') || (onestring[i + 1] == '.')) {
                        error  =  true;  cout  <<  "error  on  the  "  <<  i  +  2  <<
"position";
                  }
                  else a.push(-2);
                  break;
            }
            default:
            {
                  error = true;
                  cout << "unchoun char, position " << i + 1;
            }
            }
            ++i;
      }
      int h = a.getInd();
      if (error != true) {
            for (int i = 0; i <= h; i++)
                  if (a.pop() == 0) {
                        error = true; cout << "error more '(' ";
```

```
                }
        }
        if (!error) return true; else return false;
}

int Arifmetics::getPriority(int z) {
        for (int i = 0; i < 6; i++)
                if (onestring[z] == priority[i]) return m_priority[i];
        if ((!isalpha(onestring[z])) || (onestring[z] == '.')) {
                if    (!isalpha(onestring[z])    &&    (!isdigit(onestring[z]))    &&
((onestring[z] != '.'))) return -40;
                return -2;
        }
        else return -3;
}

void Arifmetics::Lexem() {
        int i = 0;
        if (error == false) {
                int q = 0;
                while (i < strlen(onestring)) {

                        switch (getPriority(i))
                        {
                        case 2: {
                                if   (onestring[i]   ==   '*')   c[q].SetType(3);   else
c[q].SetType(4);
                                ++q;
                                ++i;
                                break;
                        }
                        case 1: {
                                int z = 0;
                                if (onestring[i] == '+') {
                                        if ((i == 0) || (getPriority(i - 1) == 0)) {
                                                if (getPriority(i + 1) == -3) {
                                                        z = 1;
                                                        int k = 0;
                                                        int d = i;
                                                        ++i;
                                                        while (getPriority(i) == -3) {
                                                                k++;
                                                                i++;
                                                        }
                                                        char a[20] = "";
                                                        strncpy(a, &onestring[d], k + 1);
                                                        a[k + 1] = '\0';
                                                        c[q].SetType(-3);
                                                        c[q].SetChar(a);
                                                        q++;
                                                }
                                                else {
                                                        z = 1;
                                                        int k = 0;
                                                        int d = i;
                                                        ++i;
                                                        while (getPriority(i) == -2) {
                                                                k++;
                                                                i++;
                                                        }
                                                        char a[20] = "";
                                                        strncpy(a, &onestring[d], k + 1);
                                                        a[k + 1] = '\0';
```

```
                        double res = atof(a);
                        c[q].SetType(-2);
                        c[q].SetNumber(res);
                        q++;
                  }
            }
            else c[q].SetType(1);


      }
      else {
            if ((i == 0) || (getPriority(i - 1) == 0)) {

                  if (getPriority(i + 1) == -3) {
                        z = 1;
                        int k = 0;
                        int d = i;
                        ++i;
                        while (getPriority(i) == -3) {
                              k++;
                              i++;
                        }
                        char a[20] = "";
                        strncpy(a, &onestring[d], k + 1);
                        a[k + 1] = '\0';
                        c[q].SetType(-3);
                        c[q].SetChar(a);
                        q++;
                  }
                  else {
                        z = 1;
                        int k = 0;
                        int d = i;
                        ++i;
                        while (getPriority(i) == -2) {
                              k++;
                              i++;
                        }
                        char a[20] = "";
                        strncpy(a, &onestring[d], k + 1);
                        a[k + 1] = '\0';
                        double res = atof(a);
                        c[q].SetType(-2);
                        c[q].SetNumber(res);
                        q++;
                  }
            }
            else c[q].SetType(2);

      }
      if (z == 0) {
            ++q;
            ++i;
      }
      break;
}
case 0: {
      c[q].SetType(0);
      ++q;
      ++i;
      break;
}
case -1: {
```

```
                        c[q].SetType(-1);
                        ++q;
                        ++i;
                        break;
                  }
                  case -3: {
                        c[q].SetType(-3);
                        int z = 1;
                        while     ((getPriority(++i)     ==     -3)&(i     !=
strlen(onestring))) ++z;
                        char name[10] = "";
                        strncpy(name, &onestring[i - z], z);
                        c[q].SetChar(name);
                        q++;
                        break;
                  }
                  case -2: {
                        int z = 1;
                        c[q].SetType(-2);
                        double res = Num(i);
                        int y = 0;

                        while ((i != strlen(onestring) && (getPriority(++i) ==
-2) && (y == 0)))
                              if (onestring[i] == '.')
                                    y = 1;
                              else
                              {
                                    res = res * 10 + Num(i);
                                    ++z;
                              }

                        int u = i - 1;
                        while ((i != strlen(onestring)) && (getPriority(i) ==
-2) && (y == 1)) {
                              res = res * 10 + Num(i);
                              ++z;
                              ++i;
                        }
                        if (z != 1)
                              res = res / (pow(10, i - u - 1));
                        c[q].SetNumber(res);
                        ++q;
                        break;
                  }

                  default:
                        break;
                  }


            }
            c[0].SetLength(q);
      }

}
double Arifmetics::calculate() {
      cout << "Answer =  ";
      Stack <double> f(d[0].GetLength());
      int i = 0;
      while (i < d[0].GetLength()) {
            double one;
            double two;
```

```
            switch (d[i].GetType())
            {
            case 1: {

                  one = f.pop();
                  two = f.pop();
                  f.push(one + two);
                  i++;
                  break;
            }
            case -2: {
                  f.push(d[i++].GetNumber());
                  break;
            }
            case -3: {
                  f.push(d[i++].GetNumber());
                  break;
            }
            case 2: {
                  one = f.pop();
                  two = f.pop();
                  f.push(two - one);
                  i++;
                  break;
            }
            case 3: {
                  one = f.pop();
                  two = f.pop();
                  f.push(one*two);
                  i++;
                  break;
            }
            case 4: {
                  one = f.pop();
                  two = f.pop();
                  f.push(two / one);
                  i++;
                  break;
            }
            default:
                  break;
            }
      }
      return f.pop();

}
```