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:

«Вычисление 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

Целью данной лабораторной работы является создание структуры хранения и перевода инфиксной формы в постфиксную на языке программирования C++. В рамках работы необходимо разработать классы TStack и TPostfix, 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 TStack.
- Реализация основных операций с стеком: push, pop, check, проверка на пустоту, очистка.
- Реализация основных операций с лексемами, включая разбиение на константы, переменные и операторы.
- Определение класса TPostfix.
- 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).

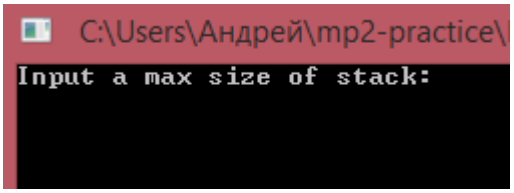

Рис. 1. Основное окно программы

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.

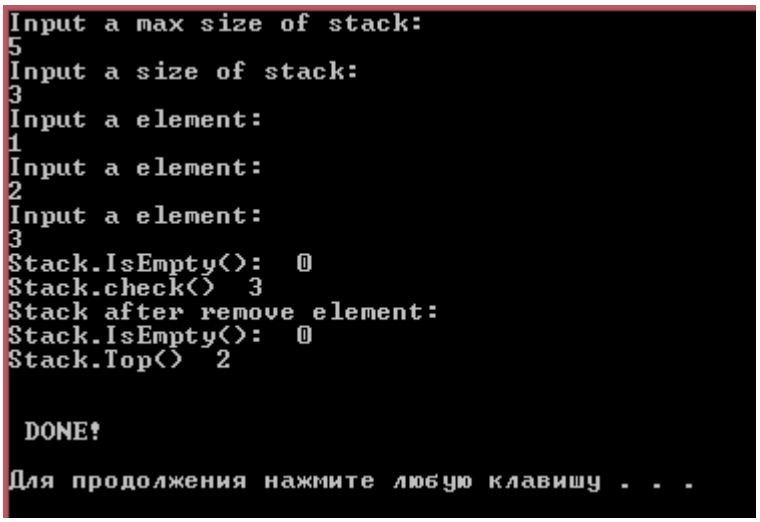

Рис. 2. Основное окно программы

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).

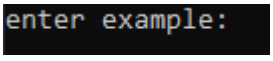

Рис. 3. Основное окно программы

2. Ведите пример. Это окно покажет основные функции класса TPostfix (инфиксная запись, результат проверки на корректность, постфиксная запись, результат вычисления) (рис. 4).

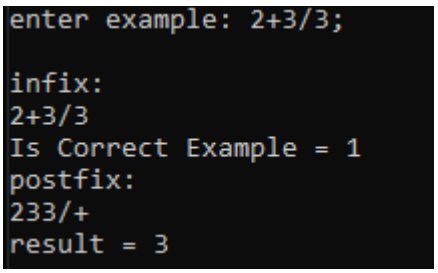

Рис. 4. Основное окно программы

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, которое строится по следующим правилам:

Исходное выражение просматривается слева направо до конца

1) операнды по мере появления помещаются в стек.

2) символы операций помещаются в стек 2 по следующим правилам.

2.1) при появлении закрытой скобки, элементы изымаются последовательно из стека 2 и переносятся в стек 1 пока стек 2 не станет пустым или пока не встретится открытая скобка.

2.2) левая скобка кладется в стек 2.

2.3) если текущая операция полученная при обходе выражения имеет более низкий приоритет чем операция на вершине стека 2, то все операции приоритет у которых больше или равен приоритету текущего перекладываются из стека 2 в стек 1.

По завершении выражения из стека 2 перекладываются в стек 1 пока он не будет пуст.

Пример: x - y / (5 * z) + 10

Результат: x y 5 z * / - 10 +

Операция разбиения инфиксной строки на лексемы:

Разбивает входную строку на вектор элементов строки.

Пример: 2+3/2+7.2

Результат: |2|+|3|/|2|+|7|.|2|

Операция перевода в постфиксную форму:

Работает по правилам описанным в начале.

Пример: x - y / (5 * z) + 10

Результат: x y 5 z * / - 10 +

Операция вычисления постфиксной формы:

1) пока не достигнут конец входной последовательности читаем очередную лексему

1.1) если прочитан операнд, то значение операнда кладется в стек.

1.2) если прочитана операция, то из стека изымаются значения двух последних операндов, выполняет операцию в порядке обратном взятия из стека операндов, результат кладется в стек.

2) если достигнут конец входной последовательности, то результат хранится в стеке.

Пример: |0|1|2|-|-1|*|+|2|0.5|0.5|+|/|-|

Результат: -1

3.2 Описание программной реализации

3.2.1 Описание класса Stack

```
class TStack {
private:
      ValType* elems;
      int MaxSize;
      int top;
public:
      TStack(int MaxSize = 10);
      TStack(const TStack<ValType>& s);
      ~TStack();
      bool isEmpty()const;
      bool isFull()const;
      ValType Top()const;
      void push(const ValType& elem);
      ValType Pop();
};
```

Назначение: представление стека.

Поля:

MaxSize – максимальный размер стека.

top – текущее колличество элементов.

elems – память для представления стека.

Методы:

```
TStack(int MaxSize = 10);
```

Назначение: конструктор с параметром.

Входные параметры:

MaxSize – максимальный размер стека.

```
TStack(const TStack<ValType>& s);
```

Назначение: конструктор копирования.

Входные параметры:

S – стек.

```
~TStack();
```
Назначение: деструктор.

```
void push(const ValType a);
```
Назначение: добавление элемента в начало.

Выходные параметры:

a – новый элемент.

```
ValType pop();
```
Назначение: удаление элемента с верхушки стека.

```
ValType Top();
```
Назначение: получение значения элемента с верхушки стека.

Выходные параметры:

Значение элемента.

```
bool IsEmpty();
```
Назначение: проверка стека на пустоту.

Выходные параметры:

Результат проверки.

```
bool IsFull();
```
Назначение: проверка стека на полноту.

Выходные параметры:

Результат проверки.

3.2.2 Описание класса TPostfix

```
class TPostfix
{
private:
      string infix;
      vector<string> postfix;
      vector<char> lexems;
      map<string, int> priority;
      map<string, double> operands;

      void Parse();
      void ToPostfix();
      void setOperands();
      bool IsCorrect()const;
      bool IsCorrectOperator()const;
      bool IsCorrectOperands()const;
      bool IsOperator(int ind)const;
      bool IncorrectSybol()const;
```

```
public:
      TPostfix(const string& infx);
      string GetInfix()const { return infix; }
      string GetPostfix()const;

      double Calculate();
      double Calculate(map<string, double> opernds);

      void PrintInfix();
      void PrintPostfix();
};
```

Назначение: перевод арифметического выражения в постфиксную запись.

Поля:

`string infix` – инфиксная запись.

`vector<string> postfix` – постфиксная запись.

`vector<char> lexems` – лексемы.

`map<string, int> priority` – приоритет операций.

`map<string, double> operands` – карта операндов.

Методы:

`TPostfix(string infx);`

Назначение: конструктор с параметром.

Входные параметры:

`infx` – инфиксная запись выражения.

`string GetInfix()const;`

Назначение: получение инфиксной строки.

Выходные параметры:

Строка.

`string GetPostfix()const;`

Назначение: получение постфиксной строки.

Выходные параметры:

Строка.

`void setOperands();`

Назначение: получение значений операндов через консоль ввода.

`void Parse();`

Назначение: разбиение входное строки на лексемы.

`void ToPostfix();`

Назначение: перевод в постфиксную запись.

`double Calculate();`

Назначение: вычисление постфиксной записи.

Выходные параметры:

Результат вычисления.

```
double Calculate(map<string, double> opernds);
```
Назначение: вычисление постфиксной записи.

Выходные параметры:

Результат вычисления.

```
bool IsCorrect()const;
```
Назначение: проверка корректности ввода выражения.

Выходные параметры:

Результат проверки.

```
bool IsCorrectOperator()const;
```
Назначение: проверка корректности ввода операторов.

Выходные параметры:

Результат проверки.

```
bool IsCorrectOperands()const;
```
Назначение: проверка корректности ввода операндов.

Выходные параметры:

Результат проверки.

```
bool IsOperator(int ind)const;
```
Назначение: проверка является ли элемент оператором.

Выходные параметры:

Результат проверки.

```
bool IncorrectSybol()const;
```
Назначение: проверка на наличие некорректных символов.

Выходные параметры:

Результат проверки.

```
void PrintInfix();
```
Назначение: вывод инфиксной записи.

```
void PrintPostfix();
```

Назначение: вывод постфиксной записи.

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 TStack

```
template <class ValType>
TStack<ValType>::TStack(int MaxSize)
{
	if (MaxSize < 0)
		throw std::exception("invalid size");
	top = -1;
	this->MaxSize = MaxSize;
	elems = new ValType[MaxSize];
}

template <class ValType>
TStack<ValType>::TStack(const TStack<ValType>& s)
{
	MaxSize = s.MaxSize;
	top = s.top;
	elems = new ValType[MaxSize];
	for (int i = 0; i <= top; i++)
		elems[i] = s.elems[i];
}

template <class ValType>
TStack<ValType>::~TStack()
{
	if (elems != nullptr)
		delete[] elems;
}

template <class ValType>
bool TStack<ValType>::isEmpty()const
{
	return top == -1;
}

template <class ValType>
bool TStack<ValType>::isFull()const
{
	return top + 1 == MaxSize;
}

template <class ValType>
ValType TStack<ValType>::Top()const
{
	if (top == -1)
		throw std::exception("stack is empty");
	return elems[top];
}

template <class ValType>
void TStack<ValType>::push(const ValType& elem)
{
	if (top + 1 == MaxSize)
		throw std::exception("stack is full");
	elems[++top] = elem;
}

template <class ValType>
ValType TStack<ValType>::Pop()
{
```

```
      if (top--  == -1)
            throw std::exception("stack is empty");
      return elems[top + 1];
}
```

Приложение В. Реализация класса TPostfix

```
TPostfix::TPostfix(const string& infx) {
      if (infx == "")
            throw std::exception("empty infix");
      infix = infx;
      std::string::iterator end_pos = std::remove(infix.begin(), infix.end(),
' ');
      infix.erase(end_pos, infix.end());
      priority = { {"*", 3}, {"/", 3},
            {"+", 2}, {"-", 2},
            {"(", 1}, {"=", 0}};
      ToPostfix();
}

void TPostfix::Parse() {
      for (char c : infix)
            lexems.push_back(c);
}

string TPostfix::GetPostfix()const
{
      string tmp="";
      for (int i = 0; i < postfix.size(); i++)
      {
            tmp += postfix[i];
      }
      return tmp;
}

void TPostfix::PrintInfix()
{
      std::cout << infix;
}
void TPostfix::PrintPostfix()
{
      std::cout << GetPostfix();
}

bool TPostfix::IsCorrect()const
{
      return IsCorrectOperands() && IsCorrectOperator() && !IncorrectSybol();
}

bool TPostfix::IsOperator(int ind)const
{
      if ( lexems[ind] == '*' || lexems[ind] == '/' ||
            lexems[ind] == '+' || lexems[ind] == '-' )
            return true;
      return false;
}


bool TPostfix::IsCorrectOperands()const
{
      int flag=0;
      int ind = 0;
      while (ind < lexems.size() - 1)
```

```
	{
		if (lexems[ind] == ')' && !IsOperator(ind + 1))
			return false;
		if (lexems[ind] == '(' && IsOperator(ind + 1))
			return false;
		if(IsOperator(ind) || lexems[ind] == '(' || lexems[ind] == ')')
			ind++;
		if(lexems[ind] == '.' && IsOperator(ind +1))
			return false;
		int flag = 0;
		while(isdigit(lexems[ind]) || lexems[ind]=='.')
		{
			ind++;
			if (ind >= lexems.size())
				break;
			if (lexems[ind ] == '.')
			{
				flag++;
			}
			if (isalpha(lexems[ind]))
				return false;
		}
		if (flag > 1)
			return false;
		if (ind >= lexems.size())
			break;
		while(isalpha(lexems[ind]) && !lexems[ind] != '('
					&& !lexems[ind] != ')')
		{
			if (ind >= lexems.size() - 1)
				break;
			if (isdigit(lexems[ind+1]) || (lexems[ind + 1] == '.'))
				return false;

			ind++;
		}
	}
	return true;
}

bool TPostfix::IsCorrectOperator()const
{
	TStack<char> brackets;
	int flag = 0;
	for (char i : lexems)
	{
		switch (i)
		{
		case '(':
		{
			brackets.push('(');
			flag++; break;
		}
		case ')':
		{
		if (flag == 0)
			return false;
		if (brackets.isEmpty())
			return false;
		brackets.Pop();
		flag++;
		break;
		}
```

```
            case '*': case '/': case '+': case '-': case ',':
            {
                  if (flag == 0 || flag == lexems.size()-1)
                        return false;
                  if (IsOperator(flag - 1) || IsOperator(flag+1) ||
                        lexems[flag-1] == '.' || lexems[flag+1] == '.'
                        || lexems[flag+1] == ')' || lexems[flag - 1] == '(')
                        return false;
                  flag++;
                  break;
            }
            default: {flag++; break; }
            }
      }
      if (!brackets.isEmpty())
            return false;

      return true;
}

bool TPostfix::IncorrectSybol()const
{
      for (int i = 0; i < lexems.size(); i++)
            if (!IsOperator(i) && !isdigit(lexems[i]) &&
                  !isalpha(lexems[i]) && lexems[i] != '.' &&
                  lexems[i] != '(' && lexems[i] != ')')
                  return true;
      return false;
}

void TPostfix::ToPostfix()
{
      Parse();
      if (!IsCorrect())
      {
            throw std::exception("invalid infix");
      }
      TStack<string> st2(lexems.size());
      int i = 0;
      while (i < lexems.size())
      {
            string tmp;
            switch (lexems[i])
            {
            case '(':
            {
                  tmp = lexems[i];
                  st2.push(tmp);
                  i++;
                  break;
            }
            case ')':
            {
                  while (st2.Top() != "(")
                  {
                        postfix.push_back(st2.Pop());
                  }
                  st2.Pop();
                  i++;
                  break;
            }
            case '+': case '-': case '*': case '/':
            {
```

```
                int flag = 0;
                tmp = lexems[i];
                if (!st2.isEmpty())
                      while (flag == 0)
                      {
                            if (priority[st2.Top()] >= priority[tmp])
                            {
                                  postfix.push_back(st2.Pop());
                                  if (st2.isEmpty())
                                        break;
                            }
                            else if (isalpha(lexems[i])
                                   || isdigit(lexems[i]))
                                  flag++;
                            else
                                  flag++;
                      }
                st2.push(tmp);
                i++;
                break;
          }
          default:
          {
                string operand = "";
                int flag = 0;
                while (isdigit(lexems[i]))
                {
                      operand += lexems[i];
                      i++;
                      if (i >= lexems.size() - 1)
                            break;
                      if(lexems[i] == '.' || isdigit(lexems[i]))
                      {
                            operand += lexems[i];
                            i++;
                      }
                }
                if (operand != "")
                {
                      postfix.push_back(operand);
                      break;
                }
                else
                {
                      while (isalpha(lexems[i]))
                      {
                            operand += lexems[i];
                            i++;
                            if (i >= lexems.size())
                                  break;
                      }
                      postfix.push_back(operand);
                }
                break;
          }
          }
    }
    while (!st2.isEmpty())
    {
          postfix.push_back(st2.Pop());
    }
}
```

```
void TPostfix::setOperands()
{
	for (int i = 0; i < postfix.size() ; i++)
	{
		if (postfix[i] == "*" || postfix[i] == "/"
			|| postfix[i] == "+" || postfix[i] == "-")
			continue;
		if (operands.find(postfix[i]) != operands.end())
			continue;
		if (!isdigit(postfix[i][0]))
		{
			double b;
			cout << postfix[i] << " = ";
			cin >> b;
			operands.insert({ postfix[i], b });
			continue;
		}
		operands.insert({ postfix[i], stod(postfix[i]) });
	}
}

double TPostfix::Calculate()
{
	setOperands();
	return Calculate(operands);
}

double TPostfix::Calculate(map<string, double> oprnds)
{
	operands = oprnds;
	int i = 0;
	TStack<double> st(postfix.size());
	while (i < postfix.size())
	{
		bool flag = true;
		if (postfix[i] == "+")
		{
			double b = st.Pop(), a = st.Pop();
			st.push(a + b);
			i++;
			flag = false;
			continue;
		}
		if (postfix[i] == "*")
		{
			double b = st.Pop(), a = st.Pop();
			st.push(a * b);
			i++;
			flag = false;
			continue;
		}
		if (postfix[i] == "-")
		{
			double b = st.Pop(), a = st.Pop();
			st.push(a - b);
			i++;
			flag = false;
			continue;
		}
		if (postfix[i] == "/")
		{
			double b = st.Pop(), a = st.Pop();
			if (b == 0.0) throw std::exception("Division by zero");
```

```
                st.push(a / b);
                i++;
                flag = false;
                continue;
            }
            if (flag)
            {
                st.push(operands[postfix[i]]);
                i++;
            }
        }
        return st.Pop();
}
```