Министерство Образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования «Нижегородский Государственный Университет им. Н.И. Лобачевского» «Институт Информационных Технологий Математики и Механики»

**Отчет**

**По лабораторной работе №6**

**«Таблицы»**

Выполнил:

Студент группы 0826-1

Смертин Д. С.

Проверил:

Доцент кафедры МОСТ

Сысоев А. В.

г. Нижний Новгород

2017 г.

Оглавление

[Введение 3](#_Toc483579017)

[Постановка задачи 4](#_Toc483579018)

[Руководство пользователя 5](#_Toc483579019)

[Руководство программиста 6](#_Toc483579020)

[Заключение 8](#_Toc483579021)

[Литература 9](#_Toc483579022)

[Приложение 10](#_Toc483579023)

# Введение

Таблицей называется последовательность записей. Запись может состоять из нескольких полей. Одно из полей должно задавать имя записи (ключ), остальные поля образуют тело записи. Таблицы нужны, для того, чтобы была возможность обращаться к данным по имени. Такой способ более удобен пользователю. Возможные операции над таблицей: вставка новой или удаление старой записи, поиск записи по ключу.

В данной лабораторной работе рассматриваются 6 вариантов реализации таблиц: линейная и упорядоченная на массиве или списке, хэш-таблица c двумя вариантами разрешения коллизий: метод цепочек и открытая адресация.

# Постановка задачи

Цель данной лабораторной работы – разработать программу, в которой будет реализовано три вида таблиц: линейная, упорядоченная (на массиве или списке) и хэш-таблица. Результатом выполнения программы является сама реализация тип-классов, которая включает в себя реализации методов добавления или удаления записи, поиска записи и извлечения данных по ключу.

Объектами, хранящимися в таблицах, должны быть полиномы. Также должна быть реализована «алгебра полиномов» - вычисление выражения, результатом и переменными которого являются полиномы.

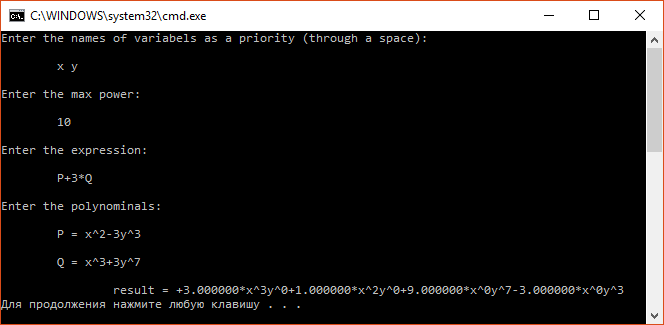
К программе (классам) нужно написать тесты, которые будут подтверждать корректность ее(их) работы.

# Руководство пользователя

Чтобы запустить программу, откройте *“TableOfPolynominal.exe”*. Запустится консольная программа, в которой последовательно вводятся:

1. Переменные, используемые в выражениях полиномов (через пробел);
2. «Максимальная степень», кратная 10. Если реальная максимальная степень в вашем полиноме меньше 10, то вводится 10, если она больше 9, но меньше 100, то вводится 100, и так далее;
3. Инфиксное выражение, переменными которого являются полиномы. Имена полиномов должны быть латинскими заглавными буквами.
4. Выражения полиномов, которые состоят из объявленных раннее переменных;

Программа выводит результат вычисления введенного выражения в виде полинома.



Чтобы программа работала корректно, используйте те же конструкции ввода, что и в примере.

# Руководство программиста

Структура программы

Код программы содержится в трёх проектах: “*TableOfPolynominal*”, “*TableOfPolynominal\_test”* и *“gtest”.*

“*TableOfPolynominal*” содержит заголовочные файлы “*TLineTable.h*”, “*TOrderTable.h*”, “*THashTable.h*”, в которых реализованы одноименные шаблон-классы, “T*Polynominal.h*”, “T*Postfix.h*”, “TStack*.h*”, файлы исходного кода: “*TPolynominal.cpp*”, “T*Postfix.cpp*” и “*Source.cpp*”. “*TableOfPolynominal\_test”* содержит исходные файлы тестов, которые работают на основе библиотеки “*gtest.lib”*, которая образуется в результате сборки проекта *“gtest”*.

В “T*Polynominal.h*” объявлены структуры *“TMonom”* и *“TLink”*, и класс *“TPolynominal”*. В них объявлены (где-то реализованы) их методы и поля.

В “TStack*.h*” реализован одноименный класс-шаблон.

В “T*Postfix.h*” и “T*Postfix.cpp*” соответственно объявлен и реализован класс для работы с арифметическими выражениями.

В “T*Polynominal.cpp*” реализованы методы класса *“TPolynominal”* и объявлены статистические поля.

В “*Source.cpp*” написан код, демонстрирующий работу с таблицами*.*

Алгоритмы

**Структура “TLine”**

Структура «запись» включает в себя два поля: имя и объект.

**Класс “TLineTable”**

Данный класс-тип представляет собой массив записей. В полях класса хранится размер таблицы, количество записей, массив записей.

**Алгоритм вставки записи в линейной таблице**

Вставка записи в линейную таблицу происходит с помощью метода AddLine(“key”, “значение”). Имени записи присваивается значение “key”, а объекту “значение”. Она помещается в первую свободную ячейку массива.

**Алгоритм поиска записи в линейной таблице**

Алгоритм метода SearchOfLineByName(“key”) проходит по всем заполненным ячейкам массива записей, сравнивая нужный “key” с именем записи. Если такая запись найдена, то выводится её позиция в массиве, если нет, то выводится «-1»

**Алгоритм удаления записи в линейной таблице**

В методе DeleteLine(“key”) сначала идет поиск записи в таблице. Если запись найдена, то эта ячейка заменяется на последнюю заполненную. После этого последняя ячейка удаляется.

**Класс “TOrderTable”**

Данный класс-тип представляет собой упорядоченный массив записей. В полях класса хранится размер таблицы, количество записей, массив записей. Записи упорядочиваются по первой букве имени по алфавиту.

**Алгоритм вставки записи в линейной таблице**

Вставка записи в линейную таблицу происходит с помощью метода AddLine(“key”, “значение”). Имени записи присваивается значение “key”, а объекту “значение”. Она помещается в первую ячейку массива, где значение ключа будет больше предыдущих значений имени. Ячейки, стоящие после вставленной, сдвигаются.

**Алгоритм поиска записи в линейной таблице**

Алгоритм метода SearchOfLineByName(“key”) проходит по всем заполненным ячейкам массива записей, сравнивая нужный “key” с именем записи. Если такая запись найдена, то выводится её позиция в массиве, если нет, то выводится «-1»

**Алгоритм удаления записи в линейной таблице**

В методе DeleteLine(“key”) сначала идет поиск записи в таблице. Если запись найдена, то эта ячейка удаляется. После этого последующие ячейки сдвигаются к месту удаления.

**Структура “THashLine”**

Данная структура включает в себя следующие поля: имя и объект записи, количество записей в цепочке, указатель на следующую запись в цепочке.

**Класс “THashTable”**

Данный класс-тип представляет собой массив записей. В полях класса хранится количество записей и массив записей. Обращение к элементам массива происходит напрямую по индексу, полученному в результате работы метода Hash(“key”).

**Метод Hash(“key”)**

Данный метод преобразует имя записи в индекс, по которому идет обращение к элементам массива. Так как таблица, в данной лабораторной работе, нужна для хранения полиномов, то нам достаточно хранить 35 «главных» записей (по количеству латинских заглавных букв и цифр). Побочные записи хранятся в цепочках (списках), основанных на «главных» записях.

**Алгоритм вставки записи в линейной таблице**

Вставка записи в линейную таблицу происходит с помощью метода AddLine(“key”, “значение”). Имени записи присваивается значение “key”, а объекту “значение”. Она помещается в ячейку массива с номером, сгенерированным методом Hash(“key”). Если эта ячейка уже занята, то запись вставляется в конец цепочки этой же ячейки.

**Алгоритм поиска записи в линейной таблице**

Алгоритм метода IsFound(“key”) обращается к элементу массива с номером, сгенерированным методом Hash(“key”). Если она пустая, то результат – false, если нет, то проходимся по цепочке сравнивая имена нужной записи с именами записей в цепочке. Если запись с таким именем найдена, то результируем true, если нет, то false.

**Алгоритм удаления записи в линейной таблице**

Алгоритм метода DeleteLine(“key”) обращается к элементу массива с номером, сгенерированным методом Hash(“key”), находит запись в цепочке и удаляет её. Указатель на следующую запись предыдущей записи в цепочке переставляется на последующую запись после удаленной.

# Заключение

В процессе выполнения лабораторной работы изучены такие структуры данных, как линейные, упорядоченные и хэш-таблицы, выполнена их реализация в виде классов.

В каждом из таких классов реализованы методы поиска, добавления и удаления записей.

Реализована «алгебра полиномов» - вычисление выражения, результатом и переменными которого являются полиномы.

К программе были написаны тесты, подтверждающие корректность её работы.

# Литература

1. Герберт Шилдт. Полный справочник по C++ = C++: The Complete Reference. — 4-е изд. — М.: Вильямс, 2011. — С. 800. — ISBN 978-5-8459-0489-8.
2. Бьёрн Страуструп. Программирование: принципы и практика использования C++, исправленное издание = Programming: Principles and Practice Using C++. — М.: Вильямс, 2011. — С. 1248. — ISBN 978-5-8459-1705-8.
3. Майкл Мейн, Уолтер Савитч. Структуры данных и другие объекты в C++ = Data Structures and Other Objects Using C++. — 2-е изд. — М.: Вильямс, 2002. — 832 с. — ISBN 0-201-70297-5.

# Приложение

***“TLineTable.h”***

#pragma once;

#include <string>

#define MAX\_SIZE 100000

using namespace std;

template <class T>

struct TLine

{

string nameKey;

T pValue;

};

template <class T>

class TLineTable

{

private:

TLine<T>\* line;

size\_t numOfLine;

size\_t countOfLine;

public:

TLineTable()

{

numOfLine = NULL;

countOfLine = NULL;

line = nullptr;

}

TLineTable(size\_t \_numOfLine)

{

if (\_numOfLine > MAX\_SIZE || \_numOfLine < NULL)

throw \_numOfLine;

numOfLine = \_numOfLine;

line = new TLine<T>[\_numOfLine];

countOfLine = NULL;

}

~TLineTable(){}

bool IsFull()

{

return (countOfLine == numOfLine);

}

bool IsEmpty()

{

return (countOfLine == NULL);

}

int SearchOfLineByName(string \_nameKey)

{

if (IsEmpty())

return -1;

for (size\_t i = 0; i < countOfLine; i++)

{

if (line[i].nameKey == \_nameKey)

return i;

}

return -1;

}

int SearchOfLineByValue(T \_pValue)

{

for (size\_t i = 0; i < countOfLine; i++)

{

if (line[i].pValue == \_pValue)

return i;

}

return -1;

}

void AddLine(string \_nameKey, T \_pValue)

{

if (IsFull())

throw "Is Full";

line[countOfLine].nameKey = \_nameKey;

line[countOfLine].pValue = \_pValue;

countOfLine++;

}

void DeleteLine(string \_nameKey)

{

if (IsEmpty())

throw "Is Empty";

int pos = SearchOfLineByName(\_nameKey);

if (pos == -1)

throw "Is Not Found";

line[pos].nameKey = line[countOfLine-1].nameKey;

line[pos].pValue = line[countOfLine-1].pValue;

line[countOfLine].nameKey = '\0';

countOfLine--;

}

T GetValue(string \_nameKey)

{

if (IsEmpty())

throw "Is Empty";

int pos = SearchOfLineByName(\_nameKey);

if (pos == -1)

throw "Is Not Found";

return line[pos].pValue;

}

};

***“TOrderTable.h”***

#pragma once;

#include <string>

#define MAX\_SIZE 100000

using namespace std;

template <class T>

struct TLine

{

string nameKey;

T pValue;

};

template <class T>

class TOrderTable

{

private:

TLine<T>\* line;

size\_t numOfLine;

size\_t countOfLine;

public:

TOrderTable()

{

numOfLine = NULL;

countOfLine = NULL;

line = nullptr;

}

TOrderTable(size\_t \_numOfLine)

{

if (\_numOfLine > MAX\_SIZE || \_numOfLine < NULL)

throw \_numOfLine;

numOfLine = \_numOfLine;

line = new TLine<T>[\_numOfLine];

countOfLine = NULL;

}

~TOrderTable(){}

bool IsFull()

{

return (countOfLine == numOfLine);

}

bool IsEmpty()

{

return (countOfLine == NULL);

}

int SearchOfLineByName(string \_nameKey)

{

if (IsEmpty())

return -1;

for (size\_t i = 0; i < countOfLine; i++)

{

if (line[i].nameKey == \_nameKey)

return i;

}

return -1;

}

int SearchOfLineByValue(T \_pValue)

{

for (size\_t i = 0; i < countOfLine; i++)

{

if (line[i].pValue == \_pValue)

return i;

}

return -1;

}

void AddLine(string \_nameKey, T \_pValue)

{

if (IsFull())

throw "Is Full";

TLine<T> temp;

int pos = 0;

while (\_pValue <= line[pos].pValue)

pos++;

temp = line[pos];

line[pos].nameKey = \_nameKey;

line[pos].pValue = \_pValue;

countOfLine++;

int flag = 1;

TLine<T> tmp;

for (size\_t i = pos + 1; i < countOfLine; i++)

{

if (flag)

{

tmp = line[i];

line[i] = temp;

flag--;

}

else

{

temp = line[i];

line[i] = tmp;

flag++;

}

}

}

void DeleteLine(string \_nameKey)

{

if (IsEmpty())

throw "Is Empty";

int pos = SearchOfLineByName(\_nameKey);

if (pos == -1)

throw "Is Not Found";

for (size\_t i = pos; i < countOfLine - 1; i++)

{

line[i].nameKey = line[i + 1].nameKey;

line[i].pValue = line[i + 1].pValue;

}

line[countOfLine].nameKey = '\0';

countOfLine--;

}

T GetValue(string \_nameKey)

{

if (IsEmpty())

throw "Is Empty";

int pos = SearchOfLineByName(\_nameKey);

if (pos == -1)

throw "Is Not Found";

return line[pos].pValue;

}

};

***“THashTable.h”***

#pragma once;

#include "TPolynominal.h"

#define MaxSizeOfTable 36

template <class T>

struct THashLine

{

size\_t numOfLine;

std::string nameOfLine;

T /\*TPolynominal \*/value;

THashLine<T>\* pNextLine;

THashLine()

{

numOfLine = NULL;

}

};

template <class T>

class THashTable

{

THashLine<T> line[MaxSizeOfTable];

size\_t countOfLine;

size\_t Hash(std::string \_nameOfLine)

{

return ((int)(\_nameOfLine[0]) - 48);

}

public:

THashTable()

{

countOfLine = NULL;

}

bool IsFull()

{

return (countOfLine == MaxSizeOfTable);

}

bool IsEmpty()

{

return (countOfLine == NULL);

}

bool IsFound(std::string \_nameOfLine)

{

size\_t hashKey = Hash(\_nameOfLine);

if (line[hashKey].numOfLine == 0) return false;

if (line[hashKey].nameOfLine == \_nameOfLine)

{

return true;

}

else

{

THashLine<T>\* p = new THashLine<T>;

p = line[hashKey].pNextLine;

for (size\_t i = 0; i < line[hashKey].numOfLine - 1; i++)

{

if (p->nameOfLine == \_nameOfLine)

return true;

p = p->pNextLine;

}

}

return false;

}

void AddLine(std::string \_nameOfLine, T /\*TPolynominal \*/ \_polinom)

{

if (IsFull())

throw "Is Full";

size\_t hashKey = Hash(\_nameOfLine);

if (line[hashKey].numOfLine == 0)

{

line[hashKey].nameOfLine = \_nameOfLine;

line[hashKey].value = \_polinom;

}

else

{

line[hashKey].pNextLine = new THashLine<T>;

THashLine<T>\* p = new THashLine<T>;

p = line[hashKey].pNextLine;

for (size\_t i = 0; i < line[hashKey].numOfLine-1; i++)

{

p->pNextLine = new THashLine<T>;

p = p->pNextLine;

}

p->nameOfLine = \_nameOfLine;

p->value = \_polinom;

}

line[hashKey].numOfLine++;

countOfLine++;

}

void DeleteLine(std::string \_nameOfLine)

{

if (IsEmpty())

throw "Is Empty";

if (IsFound(\_nameOfLine) == false)

throw "Is Not Found";

size\_t count = 0;

size\_t hashKey = Hash(\_nameOfLine);

if (line[hashKey].numOfLine == 1)

{

T \_polinom;

line[hashKey].nameOfLine = "";

line[hashKey].value = \_polinom;

}

else

{

THashLine<T>\* p = new THashLine<T>;

p = line[hashKey].pNextLine;

while (p->nameOfLine != \_nameOfLine)

{

p = p->pNextLine;

count++;

}

THashLine<T>\* pl = new THashLine<T>;

pl = line[hashKey].pNextLine;

for (size\_t i = 0; i < count - 2; i++)

{

pl = pl->pNextLine;

}

pl->pNextLine = p->pNextLine;

delete p;

}

line[hashKey].numOfLine--;

countOfLine--;

}

T /\*TPolynominal\*/ GetPolinominal(std::string \_nameOfLine)

{

if (IsEmpty())

throw "Is Empty";

if (IsFound(\_nameOfLine) == false)

throw "Is Not Found";

size\_t hashKey = Hash(\_nameOfLine);

if (line[hashKey].nameOfLine == \_nameOfLine)

{

return line[hashKey].value;

}

else

{

THashLine<T>\* p = new THashLine<T>;

p = line[hashKey].pNextLine;

for (size\_t i = 0; i < line[hashKey].numOfLine-1; i++)

{

if (p->nameOfLine == \_nameOfLine)

return p->value;

p = p->pNextLine;

}

}

}

};

***“Source.cpp”***

#include <iostream>

#include "TPolynominal.h"

#include "TPostfix.h"

#include "THashTable.h"

#include "TLineTable.h"

#include "TOrderTable.h"

using namespace std;

void main()

{

string strVar;

string infixStr;

string\* arrVarPol;

//double argPol[1] = { 2 };

cout << "Enter the names of variabels as a priority (through a space):\n\n\t";

getline(cin, strVar);

//strVar = "x";

cout << "\n";

cout << "Enter the max power:\n\n\t";

cin >> TPolynominal::Power/\* = 10\*/;

cout << "\n";

cout << "Enter the expression:\n\n\t";

cin >> infixStr /\*= "P+Q"\*/;

cout << "\n";

TPostfix expression(infixStr);

expression.ToPostfix();

arrVarPol = expression.GetNameOfPolynominals();

THashTable<TPolynominal> tableOfPol;

cout << "Enter the polynominals:\n\n\t";

for (size\_t i = 0; i < expression.GetNumOfVar(); i++)

{

string ce = "0123456789.";

int j = 0;

while (j < arrVarPol[i].length())

{

if (ce.find(arrVarPol[i][j]) != std::string::npos)

j++;

else

break;

}

if (j == arrVarPol[i].length())

{

TPolynominal \_p(arrVarPol[i]);

tableOfPol.AddLine(arrVarPol[i], \_p);

}

else

{

cout << arrVarPol[i] << " = ";

string strPol/\* = "x^2"\*/;

cin >> strPol;

cout << "\n\t";

TPolynominal pol(strPol, strVar);

tableOfPol.AddLine(arrVarPol[i], pol);

}

}

TPolynominal result = expression.CalculatePolynom(tableOfPol);

cout << "\tresult = " << result.ToString() << endl;

system("PAUSE");

}