БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

ИИТ БГУИР

Программное обеспечение информационных технологий

Отчет

По лабораторной работе №3

«Переопределение операторов. Дружественность»

по курсу «ООТПиСП»

Выполнил студент группы 381064 Сорока А.А.

Проверил преподаватель: Лашкевич Е.М.

Минск 2015 г

**Цель работы** – научиться переопределять операции для классов, использовать переопределенных операторов для реализации более привычной и удобной формы записи манипулирования над ними, использовать дружественных функций.

**Задание А4.** Определите в классе String следующие операции:

String& operator=(const char\*);//копирование строк

String& operator=(const String&);//копирование строк

String& operator+=(const char\*);//объединение строк

String& operator+=(const String&);//объединение строк

char operator[](int);//получить символ

**Код программы:**

Main.h:

#include <iostream>

#include <string>

using namespace std;

class String

{

public:

String();

~String();

void SetString(string);

string GetString();

String& operator= (const String&);

String& operator= (char\*);

String& operator+= (const String&);

String& operator+= (char\*);

char& operator[] (int);

private:

string str;

};

Main.cpp:

#include "main.h"

int main(int argc, char \*argv[])

{

String a, b;

a.SetString("abc");

cout << "default " << a.GetString() << endl;;

b.SetString(" b string");

a = b;

cout << "after =b " << a.GetString() << endl;;

char \*c = " from c";

a = c;

cout << "after =c " << a.GetString() << endl;

a += b;

cout << "after +b " << a.GetString() << endl;;

a += c;

cout << "after +c " << a.GetString() << endl;

cout << "char position 3 " << a[3] << endl;

system("pause");

return 0;

}

String::String()

{

}

String::~String()

{

}

void String::SetString(string tmp)

{

str = tmp;

}

string String::GetString()

{

return str;

}

String& String::operator= (const String& tmp)

{

if (this == &tmp) return \*this;

str = tmp.str;

return \*this;

}

String& String::operator+= (const String& tmp)

{

if (this == &tmp) return \*this;

str += tmp.str;

return \*this;

}

String& String::operator= (char \* a)

{

string tmp(a);

str = tmp;

return \*this;

}

String& String::operator+= (char \* a)

{

string tmp(a);

str += tmp;

return \*this;

}

char& String::operator[] (int pos)

{

char tmp = '\0';

if (pos < str.length())

{

return str[pos];

}

else

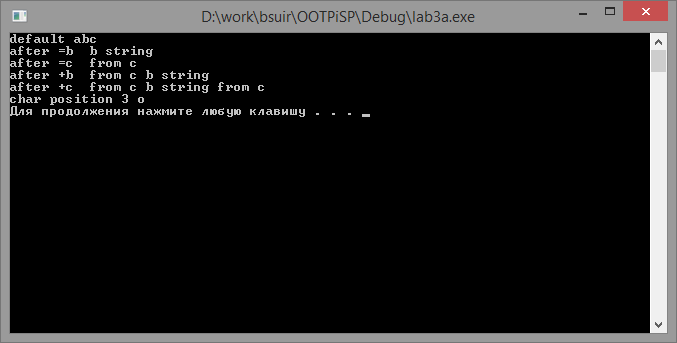
{

return tmp;

}

}

**Скриншот выполнения программы:**



**Задание В3.**Составить описание класса, обеспечивающего представление матрицы произвольного размера с возможностью изменения числа строк и столбцов, вывода на экран подматрицы любого размера и всей матрицы.

**Код программы:**

Main.h:

#include <iostream>

using namespace std;

class Matrix

{

public:

Matrix();

Matrix(int, int);

~Matrix();

void SetValue(int);

void SetValue(int, int, int);

void SetValue(int \*\*, int, int, int);

void ViewMatrix();

void ViewMatrix(int, int, int ,int);

void NewMatrix(int, int);

private:

int \*\*matrix;

int m, n; //размеры матрицы

void FreeMatrix(int \*\*, int, int);

int \*\* InitMatrix(int, int); // создает и зполняет нулями матрицу при создании

};

Main.cpp:

#include "main.h"

int main(int argc, char \*argv[])

{

Matrix m(3,5);

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 5; j++)

{

m.SetValue(i, j, i\*j + 1);

}

}

cout << "Matrix: " << endl;

m.ViewMatrix();

cout << " " << endl << "Matrix from 1,1 1,4: " <<endl;

m.ViewMatrix(1, 1, 4, 3);

cout << " " << endl << "Change to matrix 4x3: " << endl;

m.NewMatrix(4, 3);

m.ViewMatrix();

cout << " " << endl;

system("pause");

return 0;

}

Matrix::Matrix()

{

m = 0;

n = 0;

}

Matrix::Matrix(int a, int b) //конструктор, создаем матрицу M x N

{

m = a;

n = b;

matrix = InitMatrix(m, n);

SetValue(0);

}

Matrix::~Matrix() // подчистим за собой динамику

{

FreeMatrix(matrix, m, n);

}

int\*\* Matrix::InitMatrix(int m, int n)

{

int \*\*matr = new int\*[m];

for (int i = 0; i < m; i++)

{

matr[i] = new int[n];

}

return matr;

}

void Matrix::SetValue(int value)

{

for (int i = 0; i < m; i++)

{

for (int j = 0; j < n; j++)

{

matrix[i][j] = value;

}

}

}

void Matrix::SetValue(int \*\* matr, int a, int b, int value)

{

for (int i = 0; i < a; i++)

{

for (int j = 0; j < b; j++)

{

matr[i][j] = value;

}

}

}

void Matrix::SetValue(int a, int b, int value)

{

if (a < m && b < n)

{

matrix[a][b] = value;

}

}

void Matrix::ViewMatrix()

{

for (int i = 0; i < m; i++)

{

for (int j = 0; j < n; j++)

{

cout << matrix[i][j] << " ";

}

cout << endl;

}

}

void Matrix::ViewMatrix(int x1, int y1, int x2, int y2)

{

if (x1 >= 0 && x1 < n && y1 >= 0 && y1 < m && x1 >= 0 && x1 < n && y1 >= 0 && y1 < m)

{

if (y1 <= y2 && x1 <= x2)

{

for (int i = y1; i < y2; i++)

{

for (int j = x1; j < x2; j++)

{

cout << matrix[i][j] << " ";

}

cout << endl;

}

}

else

{

cout << "bad parametrs" << endl;

}

}

else

{

cout << "bad parametrs" << endl;

}

}

void Matrix::NewMatrix(int a, int b)

{

if (a > 0)

{

int \*\*newMatrix = InitMatrix(a, b);

SetValue(newMatrix, a, b, 0);

for (int i = 0; i < a; i++)

{

for (int j = 0; j < b; j++)

{

if (i < m && j < n)

{

newMatrix[i][j] = matrix[i][j];

}

}

}

FreeMatrix(matrix, m, n);

matrix = newMatrix;

m = a;

n = b;

}

else

{

cout << "bad parametrs" << endl;

}

}

void Matrix::FreeMatrix(int \*\* matr, int a, int b)

{

for (int i = 0; i < m; i++)

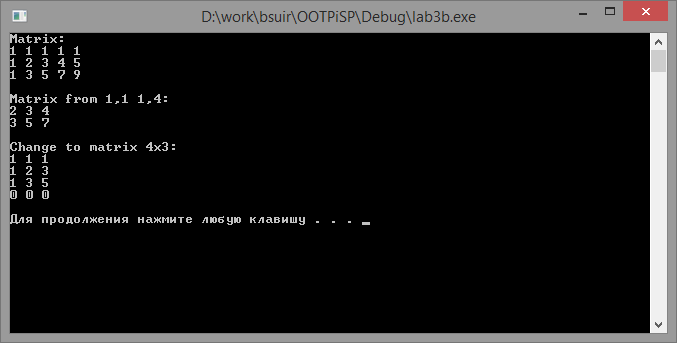
{

delete[] matr[i];

}

}

**Скриншот выполнения программы:**



**Вывод**: В ходе выполнения лабораторной работы я научился переопределять операции для классов, использовать переопределенных операторов для реализации более привычной и удобной формы записи манипулирования над ними, использовать дружественных функций.