МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Нижегородский государственный университет им. Н.И. Лобачевского»**

**Национальный исследовательский университет**

**Институт информационных технологий, математики и механики**

**Кафедра математического обеспечения и суперкомпьютерных технологий**

**ЛАБОРАТОРНАЯ РАБОТА**

**«»**

**Выполнил:** студент группы 381706-2

Гаврюшова Варвара Андреевна

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Руководитель:**

Лебедев Илья Геннадьевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

Нижний Новгород

2018

Оглавление

[Введение 3](#_Toc528661297)

[Постановка задачи 3](#_Toc528661298)

[Основные понятия и определения 3](#_Toc528661299)

[Операции над множествами 4](#_Toc528661300)

[Задача 4](#_Toc528661301)

[Условия и ограничения 4](#_Toc528661302)

[Руководство пользователя 5](#_Toc528661303)

[Руководство программиста 13](#_Toc528661305)

[Описание структуры программного комплекса 13](#_Toc528661306)

[Описание структуры данных 13](#_Toc528661307)

[Описание алгоритмов 13](#_Toc528661308)

[Заключение 15](#_Toc528661309)

[Литература 15](#_Toc528661310)

[Приложение 16](#_Toc528661311)

# 

# Введение

Теория множеств – учение об общих свойствах множеств – преимущественно бесконечных. Явным образом понятие множества подверглось систематическому

изучению во второй половине XIX века в работах немецкого математика Георга Кантора.

Влияние теории множеств на развитие современной математики очень велико. Прежде всего, теория множеств явилась фундаментом ряда новых математических дисциплин (теории функций действительного переменного, общей топологии, общей алгебры, функционального анализа и др.).

Постепенно теоретико-множественные методы находят всё большее применение и в

классических частях математики. Например, в области математического анализа они широко применяются в качественной теории дифференциальных уравнений, вариационном

исчислении, теории вероятностей и др.

Активное применение аппарата теории множеств в современной науке приводит к

необходимости создания соответствующих программных решений.

Программная реализация множества может выполняться различными способами (в

соответствии с требованиями конкретной задачи или с общих позиций) и обычно тесно

связана с использованием битовых операций в выбранном языке программирования.

Данная работа посвящена изучению одного из возможных подходов к хранению и

обработке множеств

# Постановка задачи

## Основные понятия и определения

Понятие множества, или совокупности, принадлежит к числу простейших математических понятий; оно не определяется, но может быть пояснено при помощи примеров. Так, можно говорить о множестве всех книг, составляющих данную библиотеку, множестве всех точек данной линии, множестве всех решений данного уравнения.

Книги данной библиотеки, точки данной линии, решения данного уравнения являются элементами соответствующего множества.

Чтобы определить множество, достаточно указать характеристическое свойство элементов, т. е. такое свойство, которым обладают все элементы этого множества и только они.

Может случиться, что данным свойством не обладает вообще ни один предмет; тогда говорят, что это свойство определяет пустое множество.

То, что данный предмет X есть элемент множества М, записывают так: X ∈ М (читают: X принадлежит множеству М).

Если каждый элемент множества А является в то же время элементом множества В, то множество А называется подмножеством, или частью, множества В. Это записывают так:

A ⊂ В или В ⊃ А. Таким образом, подмножеством данного множества В является и само множество В. Пустое множество, по определению, считают подмножеством всякого множества.

Одной из определяющих характеристик множества является его мощность. В рамках данной работы рассматриваются множества, содержащие конечное число элементов. В этом случае понятие мощности определяется как количество элементов множества.

Для таких множеств в математике принята следующая форма записи:

A = {a₁, a₂, ..., aₙ}, где А –множество, аₖ – элементы множества, n – мощность множества.

Множество всех возможных элементов называется Универс и обычно обозначается U.

## Операции над множествами

Пусть заданы два множества A = {a₁, a₂, ..., aₙ} и B = {b₁, b₂, ..., bₘ}. Рассмотрим следующие основные операции над множествами:

* Включение элемента в множество: A ∪{b} = {a₁, a₂, ..., aₙ, b}
* Исключение элемента из множества: A\{aₖ} = {a₁, a₂, ..., aₖ₋₁, …, aₖ₊₁, aₙ}
* Сумма (объединение) множеств. Суммой множеств А и В называется множество С, каждый элемент которого есть элемент хотя бы одного из множеств-слагаемых А и В.
* Пересечение множеств. Пересечением множеств А и В называется множество С, каждый элемент которого принадлежит обоим множествам А и В.
* Разность (дополнение) множеств. Разностью множеств А и В называется множество С, каждый элемент которого является элементом А и не является элементом В.
* Вычисление мощности множества.

## Задача

В рамках лабораторной работы ставится задача создания программных средств,

поддерживающих эффективное хранение множеств, удовлетворяющих указанным выше

допущениям, и выполнение основных операций над множествами:

* включение элемента в множество;
* исключение элемента из множества;
* проверка наличия элемента в множестве;
* объединение множеств;
* пересечение множеств;
* сравнение множеств;
* дополнение множества;

Программные средства должны содержать:

* класс Множество;
* класс Битовое поле;
* тестовое приложение, демонстрирующее использование основных операций с

множествами.

## Условия и ограничения

Сделаем следующие основные допущения:

1. Условимся рассматривать в дальнейшем конечные (см. выше) множества, состоящие из элементов произвольного типа.

2. Элементы множества проиндексированы (каждому элементу соответствует уникальный индекс).

3. Множество индексов элементов составляет непрерывный диапазон целых значений.

4. Будем считать размер множества конечным числом, не превышающим 2³¹.

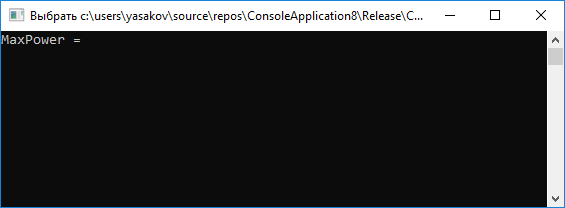
5. Нумерация бит в битовой строке – слева направо.

6. Нумерация элементов в массиве – слева направо, биты элемента – справа налево.

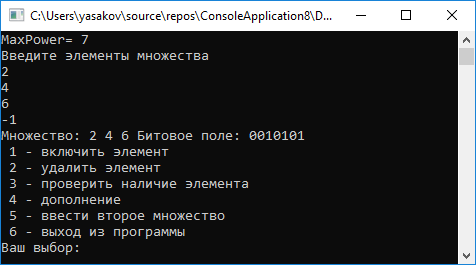
# Руководство пользователя

Возможности программы:

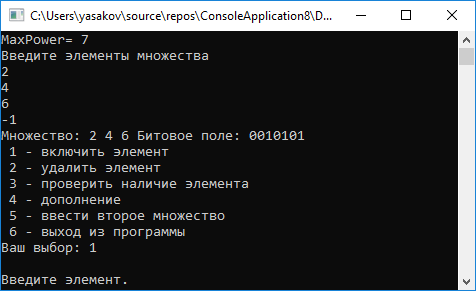
1. Ввод и вывод множества.
2. Включить элемент в множество.
3. Удалить элемент из множества.
4. Проверить наличие элемента в множестве.
5. Сравнить два множества.
6. Объединение двух множеств.
7. Пересечение двух множеств.
8. Дополнение множества.
9. При запуске, пользователю будет предложено ввести мощность множества.



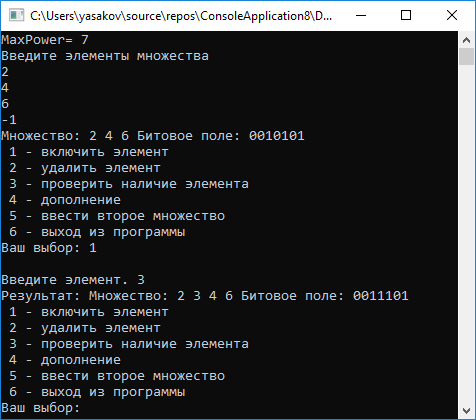
1. Далее необходимо ввести элементы множества. Чтобы прекратить вводить элементы, нужно ввести отрицательное число. После ввода всех элементов выведет множество и его битовое поле. Далее пользователю будет предложены некоторые операции над множеством (1, 2, 3, 4), ввести второе множество (5) или выйти из программы (6).



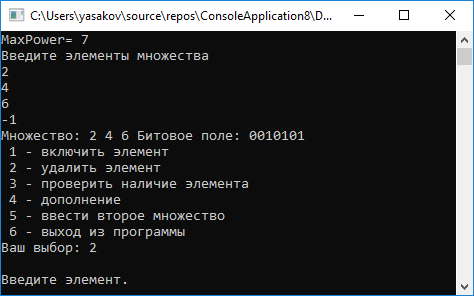
1. Если пользователь выбирает «1», программа попросит ввести элемент, который нужно будет включить в множество.



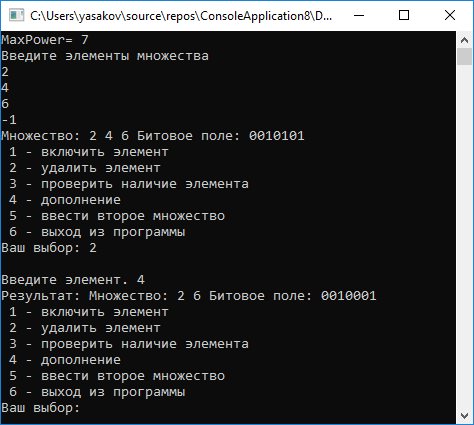
После ввода элемента программы выведет множество и его битовое поле. Далее см. пункт 4.



1. Если пользователь выбирает «2», программа попросит ввести элемент, который нужно будет удалить из множества.



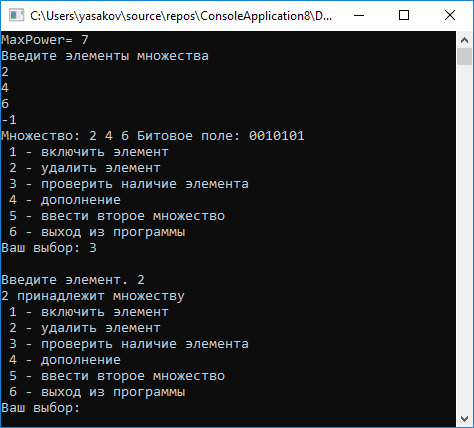
После ввода элемента программы выведет множество и его битовое поле. Далее см. пункт 4.



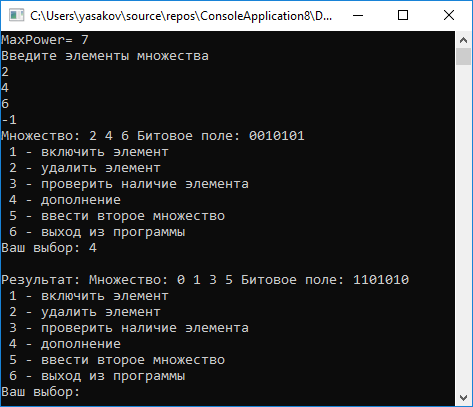
1. Если пользователь выбирает «3», программа попросит ввести элемент, с которым нужно будет провести проверку на наличие его в множестве.

# 

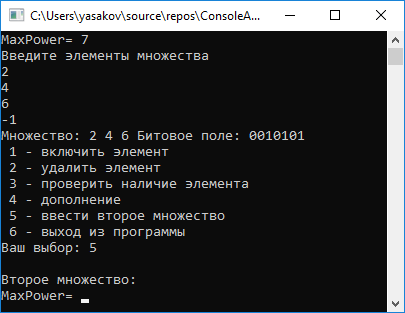
После ввода элемента программы выведет принадлежит или нет элемент множеству. Далее см. пункт 4.



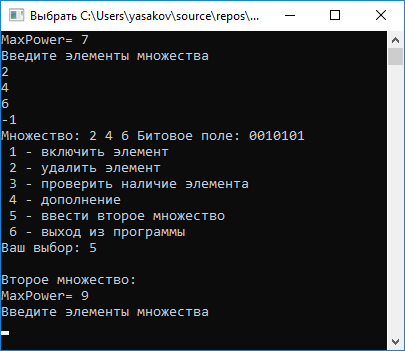
1. Если пользователь выбирает «4», программа выведет множество и его битовое поле. Далее см. пункт 4.



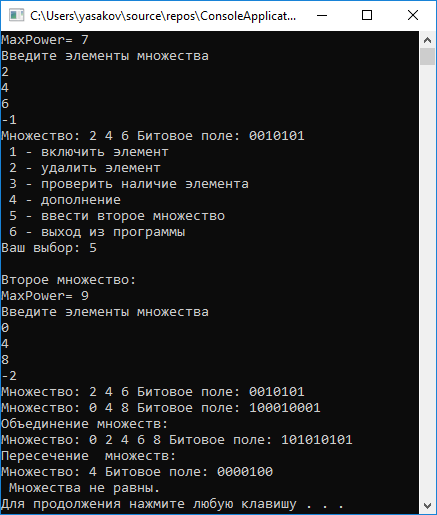
1. Если пользователь выбирает «5», программа попросит ввести мощность второго множества.



После ввода мощности пользователь должен будет ввести элементы множества.



После ввода элементов на экран будут выведены результаты операций: сравнения, объединения и пересечения. Далее см. пункт 4.



# Руководство программиста

## Описание структуры программного комплекса

* TBitField.h, TBitField.cpp – модуль с классом, реализующим операции над

Битовыми полями;

* TSet.h, TSet.cpp – модуль с классом, реализующим обработку Множеств;
* Main.cpp – модуль программы тестирования

## Описание структуры данных

Базисное множество: ячейка битового поля.

Отношения следования между состояниями (установить/сбросить бит).

Для работы с Битовым полем предлагается реализовать следующие операции:

* установить бит (в единицу);
* очистить бит (в ноль);
* получить значение бита;
* сравнить два битовых поля;
* выполнить операцию “логическое или” для двух битовых полей;
* выполнить операцию “логическое и” для двух битовых полей;
* выполнить операцию “логическое отрицание” для битового поля

Для работы с Множеством предлагается реализовать следующие операции:

* включение элемента в множество;
* исключение элемента из множества;
* проверка наличия элемента в множестве;
* сравнение множеств;
* объединение множеств;
* пересечение множеств
* дополнение множества;

## Описание алгоритмов

Класс Битовое поле(файл BitField.h):

int GetMemIndex(const int n) const; // индекс в pМем для бита n

TELEM GetMemMask (const int n) const; // битовая маска для бита n

Доступ к битам:

int GetLength(void) const; // получить длину (к-во битов)

void SetBit(const int n); //установить бит

void ClrBit(const int n); // очистить бит

int GetBit(const int n) const; // получить значение бита

Битовые операции:

Сравнение:

Если размеры массивов разные, то возвращаемое значение – 0

Цикл по элементам массива

Если элементы не равны, то возвращаемое значение – 0

Возвращаем 1

Присваивание:

Если элементы не равны, то присваиваем размер битового поля, размер массива и значения элементов массива (если нужно, то освобождаем и выделяем память)

Операция "или":

Создаем новый элемент с наибольшим из размеров битового поля

Применяем операцию "или" к параллельным элементам массива

Операция "и":

Создаем новый элемент с наименьшим из размеров битового поля

Применяем операцию "и" к параллельным элементам массива

Отрицание:

Создаем новый элемент с наименьшим из размеров битового поля

Применяем операцию отрицание к каждому элементу массива

Класс Множество (файл Set.h)

Доступ к битам:

int GetMaxPower(void) const; // максимальная мощность множества

void InsElem(const int n); // включить элемент в множество

void DelElem(const int n); // удалить элемент из множества

int IsMember(const int n) const; // проверить наличие элемента в множестве

Теоретико-множественные операции:

Сравнение:

Если битовые поля равны, то возвращаемое значение – 1

Иначе - 0

Присваивание:

Присваиваем мощность и битовое поле

# Заключение

В процессе работы были изучена задача создания программных средств,

поддерживающих эффективное хранение множеств, удовлетворяющих указанным выше

допущениям, и выполнение основных операций над множествами.

# Литература

1. Лабораторный практикум. Составители: Барышева И.В., Мееров И.Б., Сысоев А.В., Шестакова Н.В. Под редакцией Гергеля В.П. Учебно-методическое пособие. – Нижний Новгород: Нижегородский госуниверситет, 2017. – 105 с.
2. URL:

<https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82%D0%BE%D0%B2%D1%8B%D0%B5_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%B8>

Википедия. Битовые операции.

# Приложение

TSet.h

#pragma once

#include "TBitField.h"

#include <iostream>

using namespace std;

class TSet

{

private:

int MaxPower; // максимальная мощность множества

TBitField BitField; // битовое поле для хранения хар-го вектора

public:

TSet(int mp);

TSet(const TSet &s);

TSet(const TBitField &x);

~TSet();

operator TBitField(); // преобразование типа к битовому полю

void InsElem(const int n); // включить элемент в множество

void DelElem(const int n); // удалить элемент из множества

int IsMember(const int n) const; // проверить наличие элемента в множестве

int operator== (const TSet &s); // сравнение

TSet& operator=(const TSet &s); // присваивание

TSet operator+ (const int n); // включение элемента в множество

TSet operator-(const int n); // удаление элемента из множества

TSet operator+ (const TSet &s); // объединение

TSet operator\* (const TSet &s); // пересечение

TSet operator~ (void); // дополнение

friend istream &operator >> (istream &in, TSet &x);

friend ostream &operator<<(ostream &out, const TSet &x);

};

TSet.cpp

#include "stdafx.h"

#include <stdio.h>

#include "TSet.h"

#include "TBitField.h"

#include <iostream>

using namespace std;

TSet::TSet(int tmp) : BitField(tmp)

{

MaxPower = tmp;

}

TSet::TSet(const TSet &s, const TBitField &x) : BitField(x)

{

MaxPower = s.MaxPower;

}

TSet::TSet(const TSet &s) :BitField(s.BitField)

{

MaxPower = s.MaxPower;

}

TSet::TSet(const TBitField &bf) : BitField(bf)

{

MaxPower = bf.GetLenght();

}

TSet::operator TBitField()

{

return BitField;

}

void TSet::InsElem(const int n)

{

BitField.SetBit(n);

}

void TSet::DelElem(const int n)

{

BitField.ClrBit(n);

}

int TSet::IsMember(const int n) const

{

if (BitField.GetBit(n) == 0)

return 0;

else

return 1;

}

int TSet::operator== (const TSet &s)

{

if (MaxPower != s.MaxPower)

return 0;

if (BitField == s.BitField)

return 1;

else return 0;

}

TSet& TSet::operator=(const TSet &s)

{

MaxPower = s.MaxPower;

BitField = s.BitField;

return \*this;

}

TSet TSet::operator+ (const int n)

{

if (IsMember(n) == 1)

{

return \*this;

}

else

{

BitField.SetBit(n);

return \*this;

}

}

TSet TSet::operator-(const int n)

{

if (IsMember(n) == 0)

{

return \*this;

}

else

{

BitField.ClrBit(n);

return \*this;

}

}

TSet TSet::operator+ (const TSet &s)

{

int len = MaxPower;

if (s.MaxPower>MaxPower)

len = s.MaxPower;

TSet tmp(len);

for (int i = 0; i<s.MaxPower; i++)

{

if (s.IsMember(i))

tmp.InsElem(i);

}

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

{

if (IsMember(i))

{

if (tmp.IsMember(i) == 0)

tmp.InsElem(i);

}

}

return tmp;

}

TSet TSet::operator\* (const TSet &s)

{

TSet tmp(BitField & s.BitField);

return tmp;

}

TSet TSet::operator~ (void)

{

TSet res(\*this);

res.BitField = ~res.BitField;

return res;

}

istream &operator >> (istream &in, TSet &x)

{

int n=0;

cout << "Введите элементы множества " << "\n";

while (n >= 0)

{

in >> n;

if (n >= 0)

x = x + n;

}

return in;

}

ostream &operator<<(ostream &out, const TSet &x)

{

cout << "Множество: ";

for (int i = 0; i < x.MaxPower; i++)

{

if (x.BitField.GetBit(i))

out << i << " ";

}

cout << "Битовое поле: ";

out << x.BitField;

cout << "\n";

return out;

}

TSet::~TSet()

{

}

TBitField.h

#pragma once

#include <cstdio>

#include <iostream>

using namespace std;

typedef unsigned int TELEM;

class TBitField

{

private:

int BitLen; // размер битового поля

int MemLen;

TELEM \*pMEM;

int GetMemIndex(const int n) const; // индекс в pМем для бита n

TELEM GetMemMask(const int n) const; // битовая маска для бита n

public:

TBitField();

TBitField(int N);

TBitField(const TBitField &x);

~TBitField();

int GetLength(void) const;

void SetBit(const int n);

void ClrBit(const int n);

int GetBit(const int n) const;

int operator==(const TBitField &x);

TBitField& operator=(const TBitField &x);

TBitField operator|(const TBitField &x);

TBitField operator&(const TBitField &x);

TBitField operator~(void);

friend istream &operator>>(istream &istr, TBitField &x);

friend ostream &operator<<(ostream &ostr, const TBitField &x);

};

TBitField.cpp

#include "stdafx.h"

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "TBitField.h"

#include <iostream>

using namespace std;

typedef unsigned int TELEM;

TBitField::TBitField(int len)

{

if (len>0)

{

BitLen = len;

MemLen = (len + 31) >> 5;

pMem = new TELEM[MemLen];

if (pMem != NULL)

{

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

{

pMem[i] = 0;

}

}

}

}

TBitField::TBitField(const TBitField& x)

{

BitLen = x.BitLen;

MemLen = (BitLen + 31) >> 5;

pMem = new TELEM[MemLen];

if (pMem != NULL)

{

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

{

pMem[i] = x.pMem[i];

}

}

if (pMem == NULL)

{

cout << "Завершение работы. ";

return;

}

}

TBitField::~TBitField(void)

{

delete[]pMem;

pMem = NULL;

}

int TBitField::GetMemIndex(const int n) const

{

return(n >> 5);

}

TELEM TBitField::GetMemMask(const int n) const

{

return(1 << (n & 31));

}

int TBitField::GetLenght(void) const

{

return BitLen;

}

int TBitField::GetBit(const int n) const

{

if ((n >= 0)&(n<BitLen))

return pMem[GetMemIndex(n)] & GetMemMask(n);

else

return 0;

}

void TBitField::SetBit(const int n)

{

if ((n >= 0)&(n<BitLen))

pMem[GetMemIndex(n)] |= GetMemMask(n);

}

void TBitField::ClrBit(const int n) const

{

if ((n >= 0)&(n<BitLen))

pMem[GetMemIndex(n)] &= ~GetMemMask(n);

}

TBitField& TBitField::operator=(const TBitField &x)

{

if (&x != this)

{

BitLen = x.BitLen;

if (MemLen != x.MemLen)

{

delete[]pMem;

MemLen = x.MemLen;

pMem = new TELEM[MemLen];

}

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

pMem[i] = x.pMem[i];

}

return \*this;

}

int TBitField::operator==(const TBitField &x)

{

if (MemLen == x.MemLen)

{

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

{

for (int j = 0; j < x.MemLen; j++)

{

if (pMem[i] != x.pMem[j])

{

return 0;

}

}

}

return 1;

}

else

return 0;

}

TBitField TBitField::operator|(const TBitField &x)

{

int len = BitLen;

if (x.BitLen <= len)

len = x.BitLen;

TBitField temp(len);

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

temp.pMem[i] = pMem[i];

for (int i = 0; i < x.MemLen; i++)

temp.pMem[i] |= x.pMem[i];

return temp;

}

TBitField TBitField::operator&(const TBitField &x)

{

int len = BitLen;

if (x.BitLen <= len)

len = x.BitLen;

TBitField temp(len);

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

temp.pMem[i] = pMem[i];

for (int i = 0; i < x.MemLen; i++)

temp.pMem[i] &= x.pMem[i];

return temp;

}

TBitField TBitField::operator~(void)

{

int len = BitLen;

TBitField temp(len);

temp = \*this;

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

temp.pMem[i] = ~temp.pMem[i];

return temp;

}

istream &operator>>(istream &in, TBitField &x)

{

cout << "Enter max size of bit field " << endl;

in >> x.BitLen;

x.MemLen = (x.BitLen + 31) >> 5;

x.pMem = new TELEM[x.MemLen];

int i = 0;

char c;

cout << "Enter bit field " << endl;

if (x.pMem != NULL)

{

for (int j = 0; j < x.MemLen; j++)

{

x.pMem[j] = 0;

}

do

{

in >> c;

if (c == '1')

x.SetBit(i);

i++;

} while (((c == '1') || (c == '0')) && (i<x.BitLen));

}

return in;

}

ostream &operator<<(ostream &out, const TBitField &x)

{

for (int i = 0; i < x.BitLen; i++)

if (x.GetBit(i))//1

out << '1';

else

out << '0';

return out;

}Main.cpp

#include "stdafx.h"

#include "TBitField.h"

#include "TSet.h"

#include <iostream>

using namespace std;

typedef unsigned int TELEM;

int Menu()

{

cout << " 1 - включить элемент \n 2 - удалить элемент \n 3 - проверить наличие элемента \n 4 - дополнение \n 5 - ввести второе множество \n 6 - выход из программы \nВаш выбор: ";

int k;

cin >> k;

cout << endl;

return k;

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

setlocale(0, "");

int b1;

cout << "MaxPower= ";

cin >> b1;

TSet T1(b1);

cin >> T1;

cout << T1;

while (1)

{

switch (Menu())

{

case 1:

{

cout << "Введите элемент. ";

cin >> b1;

T1 = T1 + b1;

cout << "Результат: ";

cout << T1;

break;

}

case 2:

{

cout << "Введите элемент. ";

cin >> b1;

T1 = T1 - b1;

cout << "Результат: ";

cout << T1;

break;

}

case 3:

{

cout << "Введите элемент. ";

cin >> b1;

if (T1.IsMember(b1))

cout << b1 << " принадлежит множеству \n";

else

cout << b1 << " не принадлежит множеству \n";

break;

}

case 4:

{

cout << "Результат: ";

cout << ~T1;

break;

}

case 5:

{

int b2;

cout << "Второе множество: "<< "\n";

cout << "MaxPower= ";

cin >> b2;

TSet T2(b2);

cin >> T2;

cout << T1;

cout << T2;

cout << "Объединение множеств: " << endl;

cout << (T1 + T2);

cout << "Пересечение множеств: " << endl;

cout << (T1 \* T2);

if (T1 == T2)

cout << " Множества равны. " << endl;

else

cout << " Множества не равны. " << endl;

system("pause");

return 0;

}

default:

{

return 0;

}

}

}

system("pause");

return 0;

}