МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

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**

**«Структура хранения множества»**

**Выполнил:** студент группы 381706-02

Колесин Андрей Сергеевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Руководитель:**

Ассистент кафедры МОСТ

Шестакова Наталья Валерьевна

\_\_\_\_\_\_\_\_\_\_\_ Подпись

Нижний Новгород

2018

Содержание

[Введение 3](#__RefHeading___Toc16593_4200717104)

[Постановка задачи 4](#__RefHeading___Toc4252_3123410279)

[Руководство пользователя 5](#__RefHeading___Toc4254_3123410279)

[Руководство программиста 7](#__RefHeading___Toc4256_3123410279)

[Структуры данных 7](#__RefHeading___Toc4258_3123410279)

[Описание структуры программы 8](#__RefHeading___Toc4260_3123410279)

[Описание функций и процедур, их алгоритмов 10](#__RefHeading___Toc4262_3123410279)

[Заключение 11](#__RefHeading___Toc4264_3123410279)

[Список используемой литературы 12](#__RefHeading___Toc4266_3123410279)

[Приложение 13](#__RefHeading___Toc4268_3123410279)

# Введение

**Множество** — одно из ключевых понятий математики; это математический объект, сам являющийся набором, совокупностью, собранием каких-либо объектов, которые называются элементами этого множества и обладают общим для всех их характеристическим свойством.

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. Вместе с тем лишь в отдельных языках программирования предусмотрены встроенные средства для работы с множествами (примером может служить язык Pascal в реализации фирмы Borland). 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

В рамках лабораторной работы ставится задача создания программных средств, поддерживающих эффективное хранение множеств, удовлетворяющих следующим допущениям (множество конечное, то есть число элементов конечно; универсом являются целые положительные числа), и выполнение основных операций над множествами:

* включение элемента в множество;
* исключение элемента из множества;
* проверка наличия элемента в множестве;
* сложение множеств;
* пересечение множеств;
* разность множеств;
* копирование множества;
* вычисление мощности множества.

Программные средства должны содержать:

* класс Множество;
* 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 2^31 (потому что именно столько чисел содержит тип данных unsigned int).

# Руководство пользователя

Программа производит операции над множествами, которые пользователь ввел с клавиатуры.

1. Пользователь должен ввести количество элементов в универсе.(Рис. 1)

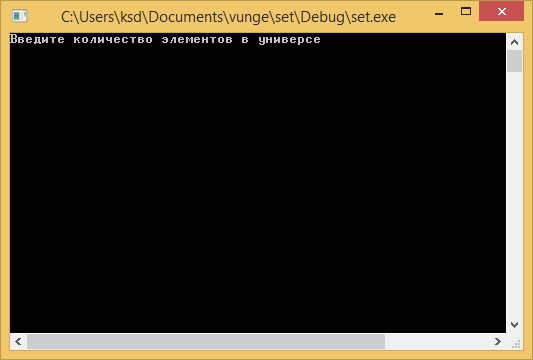


Рис. 1

1. Пользователь должен ввести два множества, над которыми буду проводиться операции. Для ввода множества нужно ввести номера элементов, принадлежащих множеству, а затем ввести -1 (Рис. 2)

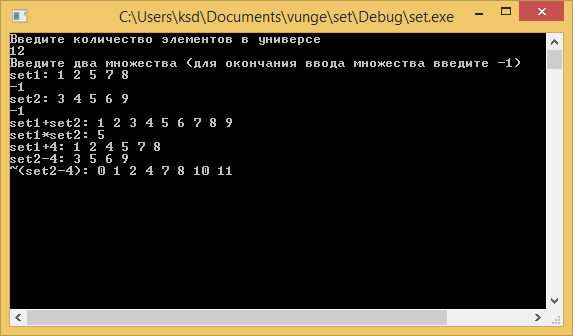


Рис. 2

1. Программа выводит результаты операций над введенными множествами.
   1. Объединение двух множеств.
   2. Пересечение двух множеств.
   3. Добавление к первому множеству элемента с номером 4.
   4. Удаление из второго множества элемента с номером 4.
   5. Дополнение множества, полученного путем удаления из второго множества элемента с номером 4.

# Руководство программиста

## Структуры данных

**Математическая структура** S = (M1,…,Mk;p1,…,ps) есть одно или несколько множеств , элементы M1,…,Mk, которых находятся в некоторых отношениях p1,…,ps.

* M1,…,Mk – базисные множества структуры.
* Каждое отношение pi есть двоичная функция, аргументами которой являются элементы базисного множества. Если аргументы функции находятся в отношении, то значение pi = ИСТИНА.

**Структура данных** — модель данных в виде математической структуры.

Понятие множества, или совокупности, принадлежит к числу простейших математических понятий; оно не определяется, но может быт математической структуры основано на понятии множества, само множество не может быть определено, как структура данных.

В дальнейших рассуждениях мы будем опираться на следующее описание множества, вытекающее из сделанных выше допущений.

Каждому множеству A = {a1, a2, …, an} ⊂ U = {u1, …, uk} поставим в соответствие характеристический вектор α = (α1, α2, …, αk), где

1. k – мощность U;
2. αi = 1, если ui ∈ A;

αi = 0, если ui ∉ A;

Тогда множество это S=(M;p), где M=(a1,a2,…,an) — множество индексов элементов, которые находятся в отношении p(ai), если элемент с индексом ai принадлежит множеству A, i∈{1,...,n}. Для множества определены операции: проверка наличия элемента, добавление элемента, удаления элемента, теоретико-множественные операции.

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, аi – элементы множества, n – мощность множества. Множество всех возможных элементов называется Универс и обычно обозначается U.

## Описание структуры программы

Программа состоит из двух классов: Tset и TBitField. И главного файла main.cpp

1. TbitField - Класс битового поля

#ifndef \_\_BITFIELD\_H\_\_

#define \_\_BITFIELD\_H\_\_

#include <iostream>

using namespace std;

typedef unsigned int TELEM;

class TBitField

{

private:

int BitLen; // длина битового поля - макс. к-во битов

TELEM \*pMem; // память для представления битового поля

int MemLen; // к-во эл-тов Мем для представления бит.поля ; в битах после BitLen должны всегда оставаться нули

// методы реализации

int GetMemIndex(const int n) const; // индекс в pМем для бита n (#О2)

TELEM GetMemMask(const int n) const; // битовая маска для бита n (#О3)

public:

TBitField();

TBitField(int len); // (#О1)

TBitField(const TBitField &bf); // (#П1)

~TBitField(); // (#С)

// доступ к битам

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

void SetBit(const int n); // установить бит (#О4)

void ClrBit(const int n); // очистить бит (#П2)

int GetBit(const int n) const; // получить значение бита (#Л1)

// битовые операции

int operator==(const TBitField &bf) const; // сравнение (#О5)

int operator!=(const TBitField &bf) const; // сравнение

TBitField& operator=(const TBitField &bf); // присваивание (#П3)

TBitField operator|(const TBitField &bf); // операция "или" (#О6)

TBitField operator&(const TBitField &bf); // операция "и" (#Л2)

TBitField operator~(void); // отрицание (#С)

friend istream &operator>>(istream &istr, TBitField &bf); // (#О7)

friend ostream &operator<<(ostream &ostr, const TBitField &bf); // (#П4)

};

1. Tset – класс множества.

#ifndef \_\_SET\_H\_\_

#define \_\_SET\_H\_\_

#include "tbitfield.h"

class TSet

{

private:

int MaxPower; // максимальная мощность множества

TBitField BitField; // битовое поле для хранения характеристического вектора

public:

TSet(int mp);

TSet(const TSet &s); // конструктор копирования

TSet(const TBitField &bf); // конструктор преобразования типа

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

// доступ к битам

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

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

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

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

// теоретико-множественные операции

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

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

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

TSet operator+ (const int Elem); // объединение с элементом

// элемент должен быть из того же универса

TSet operator- (const int Elem); // разность с элементом

// элемент должен быть из того же универса

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

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

TSet operator~ (void); // дополнение

friend istream &operator>>(istream &istr, TSet &bf);

friend ostream &operator<<(ostream &ostr, const TSet &bf);

};

#endif

## Описание функций и процедур, их алгоритмов

Для хранения битового поля используется массив unsigned int, размер которого равен 32 битам.

Получение размера массива для хранения битового поля из len элементов:

MemLen = (len + 31) >> 5;

Получение маски для n-ного бита:

mask = 1 << (n % 32);

сначала вычисляется номер бита в элементе массива n % 32

затем единица побитово сдвигается на n % 32 бит

Получение индекса бита, т. е. номера элемента массива, в котором расположен данный бит:

n >> 5

Включение n-ного бита:

int index = GetMemIndex(n);

Получаем элемент массива, который нужно изменить

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

и побитово складываем его с маской, таким образом нужный бит включаются, а остальлные не меняются.

Выключение n-ного бита:

int index = GetMemIndex(n);

Получаем элемент массива, который нужно изменить

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

побитово умножаем его с инвертированной маской, таким образом нужный бит выключаются, а остальлные не меняются.

Проверка включенности n-ного бита:

int index = GetMemIndex(n);

Получаем элемент массива, в котором находится нужный бит

pMem[index] & GetMemMask(n)

побитово умножаем его с маской, таким образом если бит был выключен результат будет равен нулю, если включен – маске, которая не равна нулю.

# Заключение

В процессе работы были достигнуты все поставленные цели, а именно: были созданы программные средства, поддерживающие хранение множеств и выполнены основные операции над множествами: включение элемента в множество, исключение элемента из множества, проверка наличия элемента в множестве, сложение множеств, пересечение множеств, разность множеств, копирование множества, вычисление мощности множества

# Список используемой литературы

1. <https://ru.wikipedia.org/wiki/Множество>
2. Лабораторный практикум: Учебно-методическое пособие / Мееров И.Б. [и др.] – Нижний Новгород: Нижегородский госуниверситет , 2017. – 105с.

# Приложение

main.cpp:

#include<iostream>

#include "tset.h"

#include<cstdlib>

#include <ctime>

using namespace std;

int main(){

srand(time(0));

setlocale(LC\_ALL, "");

cout << "Введите количество элементов в универсе" << endl;

int n;

cin >> n;

TSet set1(n);

TSet set2(n);

cout << "Введите два множества (для окончания ввода множества введите -1)" << endl;

cout << "set1: ";

cin >> set1;

cout << "set2: ";

cin >> set2;

cout << "set1+set2: " << (set1 + set2) << endl;

cout << "set1\*set2: " << (set1\*set2) << endl;

cout << "set1+4: " << (set1 + 4) << endl;

cout << "set2-4: " << set2 - 4 << endl;

cout << "~(set2-4): " << ~(set2 - 4) << endl;

cin.get();

cin.get();

return 0;

}

tbitfield.h:

#ifndef \_\_BITFIELD\_H\_\_

#define \_\_BITFIELD\_H\_\_

#include <iostream>

using namespace std;

typedef unsigned int TELEM;

class TBitField

{

private:

int BitLen; // длина битового поля - макс. к-во битов

TELEM \*pMem; // память для представления битового поля

int MemLen; // к-во эл-тов Мем для представления бит.поля ; в битах после BitLen должны всегда оставаться нули

// методы реализации

int GetMemIndex(const int n) const; // индекс в pМем для бита n (#О2)

TELEM GetMemMask(const int n) const; // битовая маска для бита n (#О3)

public:

TBitField();

TBitField(int len); // (#О1)

TBitField(const TBitField &bf); // (#П1)

~TBitField(); // (#С)

// доступ к битам

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

void SetBit(const int n); // установить бит (#О4)

void ClrBit(const int n); // очистить бит (#П2)

int GetBit(const int n) const; // получить значение бита (#Л1)

// битовые операции

int operator==(const TBitField &bf) const; // сравнение (#О5)

int operator!=(const TBitField &bf) const; // сравнение

TBitField& operator=(const TBitField &bf); // присваивание (#П3)

TBitField operator|(const TBitField &bf); // операция "или" (#О6)

TBitField operator&(const TBitField &bf); // операция "и" (#Л2)

TBitField operator~(void); // отрицание (#С)

friend istream &operator>>(istream &istr, TBitField &bf); // (#О7)

friend ostream &operator<<(ostream &ostr, const TBitField &bf); // (#П4)

};

// Структура хранения битового поля

// бит.поле - набор битов с номерами от 0 до BitLen

// массив pМем рассматривается как последовательность MemLen элементов

// биты в эл-тах pМем нумеруются справа налево (от младших к старшим)

// О8 Л2 П4 С2

#endif

tbitfield.cpp:

// ННГУ, ВМК, Курс "Методы программирования-2", С++, ООП

//

// tbitfield.cpp - Copyright (c) Гергель В.П. 07.05.2001

// Переработано для Microsoft Visual Studio 2008 Сысоевым А.В. (19.04.2015)

//

// Битовое поле

#include "tbitfield.h"

TBitField::TBitField()

{

int len = 1;

MemLen = (len + 31) >> 5;

BitLen = len;

pMem = new TELEM[MemLen];

if (pMem == nullptr){

throw "Не хватает памяти";

}

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

pMem[i] = 0;

}

}

TBitField::TBitField(int len)

{

if (len<0) throw "Количество бит не может быть отрицательным";

MemLen = (len + 31) >> 5;

BitLen = len;

pMem = new TELEM[MemLen];

if (pMem == nullptr){

throw "Не хватает памяти";

}

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

pMem[i] = 0;

}

}

TBitField::TBitField(const TBitField &bf) // конструктор копирования

{

MemLen = bf.MemLen;

BitLen = bf.BitLen;

pMem = new TELEM[MemLen];

if (pMem == nullptr){

throw "Не хватает памяти";

}

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

pMem[i] = bf.pMem[i];

}

}

TBitField::~TBitField()

{

delete pMem;

}

int TBitField::GetMemIndex(const int n) const // индекс Мем для бита n

{

if (n >= BitLen) throw "Слишком большой индекс";

if (n<0) throw "Индекс должен быть больше нуля";

return n >> 5;

}

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

{

TELEM mask = 1 << (n % 32);

return mask;

}

// доступ к битам битового поля

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

{

return BitLen;

}

void TBitField::SetBit(const int n) // установить бит

{

int index = GetMemIndex(n);

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

}

void TBitField::ClrBit(const int n) // очистить бит

{

int index = GetMemIndex(n);

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

}

int TBitField::GetBit(const int n) const // получить значение бита

{

int index = GetMemIndex(n);

return (pMem[index] & GetMemMask(n)) != 0;

}

// битовые операции

TBitField& TBitField::operator=(const TBitField &bf) // присваивание

{

if (bf.MemLen>MemLen){

MemLen = bf.MemLen;

delete pMem;

pMem = new TELEM[MemLen];

if (pMem == nullptr){

throw "Не хватает памяти";

}

}

BitLen = bf.BitLen;

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

pMem[i] = bf.pMem[i];

}

}

int TBitField::operator==(const TBitField &bf) const // сравнение

{

if (BitLen != bf.BitLen) return 0;

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

if (GetBit(i) != bf.GetBit(i)) return 0;

}

return 1;

}

int TBitField::operator!=(const TBitField &bf) const // сравнение

{

return !((\*this) == bf);

}

TBitField TBitField::operator|(const TBitField &bf) // операция "или"

{

const TBitField \*max = this;

const TBitField \*min = &bf;

const TBitField\* tmp;

if (max->BitLen < min->BitLen){

tmp = max;

max = min;

min = tmp;

}

TBitField res(max->BitLen);

for (int i = 0; i<min->MemLen; i++){

res.pMem[i] = min->pMem[i];

}

for (int i = 0; i<max->MemLen; i++){

res.pMem[i] |= max->pMem[i];

}

return res;

}

TBitField TBitField::operator&(const TBitField &bf) // операция "и"

{

const TBitField \*max = this;

const TBitField \*min = &bf;

const TBitField\* tmp;

if (max->BitLen < min->BitLen){

tmp = max;

max = min;

min = tmp;

}

TBitField res(max->BitLen);

for (int i = 0; i<min->MemLen; i++){

res.pMem[i] = min->pMem[i] & max->pMem[i];

}

return res;

}

TBitField TBitField::operator~(void) // отрицание

{

TBitField res(BitLen);

for (int i = 0; i<MemLen - 1; i++){

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

}

for (int i = (MemLen - 1) \* 32; i<BitLen; i++){

if (GetBit(i) == 0){

res.SetBit(i);

}

}

return res;

}

// ввод/вывод

istream &operator>>(istream &istr, TBitField &bf) // ввод

{

return istr;

}

ostream &operator<<(ostream &ostr, const TBitField &bf) // вывод

{

for (int i = 0; i<bf.GetLength(); i++){

ostr << bf.GetBit(i);

}

return ostr;

}

tset.h:

// ННГУ, ВМК, Курс "Методы программирования-2", С++, ООП

//

// tset.h - Copyright (c) Гергель В.П. 07.05.2001

// Переработано для Microsoft Visual Studio 2008 Сысоевым А.В. (19.04.2015)

//

// Множество

#ifndef \_\_SET\_H\_\_

#define \_\_SET\_H\_\_

#include "tbitfield.h"

class TSet

{

private:

int MaxPower; // максимальная мощность множества

TBitField BitField; // битовое поле для хранения характеристического вектора

public:

TSet(int mp);

TSet(const TSet &s); // конструктор копирования

TSet(const TBitField &bf); // конструктор преобразования типа

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

// доступ к битам

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

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

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

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

// теоретико-множественные операции

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

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

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

TSet operator+ (const int Elem); // объединение с элементом

// элемент должен быть из того же универса

TSet operator- (const int Elem); // разность с элементом

// элемент должен быть из того же универса

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

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

TSet operator~ (void); // дополнение

friend istream &operator>>(istream &istr, TSet &bf);

friend ostream &operator<<(ostream &ostr, const TSet &bf);

};

#endif

tset.cpp:

// ННГУ, ВМК, Курс "Методы программирования-2", С++, ООП

//

// tset.cpp - Copyright (c) Гергель В.П. 04.10.2001

// Переработано для Microsoft Visual Studio 2008 Сысоевым А.В. (19.04.2015)

//

// Множество - реализация через битовые поля

#include "tset.h"

TSet::TSet(int mp) : BitField(mp)

{

MaxPower = mp;

}

// конструктор копирования

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

{

MaxPower = s.MaxPower;

}

// конструктор преобразования типа

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

{

MaxPower = bf.GetLength();

}

TSet::operator TBitField()

{

return BitField;

}

int TSet::GetMaxPower(void) const // получить макс. к-во эл-тов

{

return MaxPower;

}

int TSet::IsMember(const int Elem) const // элемент множества?

{

return BitField.GetBit(Elem);

}

void TSet::InsElem(const int Elem) // включение элемента множества

{

BitField.SetBit(Elem);

}

void TSet::DelElem(const int Elem) // исключение элемента множества

{

BitField.ClrBit(Elem);

}

// теоретико-множественные операции

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

{

BitField = s.BitField;

MaxPower = s.MaxPower;

return \*this;

}

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

{

return this->BitField == s.BitField;

}

int TSet::operator!=(const TSet &s) const // сравнение

{

return this->BitField != s.BitField;

}

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

{

TSet res(this->BitField | s.BitField);

return res;

}

TSet TSet::operator+(const int Elem) // объединение с элементом

{

TSet res(\*this);

res.InsElem(Elem);

return res;

}

TSet TSet::operator-(const int Elem) // разность с элементом

{

TSet res(\*this);

res.DelElem(Elem);

return res;

}

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

{

TSet res(this->BitField & s.BitField);

return res;

}

TSet TSet::operator~(void) // дополнение

{

TSet res(\*this);

res.BitField = ~res.BitField;

return res;

}

// перегрузка ввода/вывода

istream &operator>>(istream &istr, TSet &s) // ввод

{

s = TSet(s.MaxPower);

int a;

istr >> a;

while (a != -1){

s.InsElem(a);

istr >> a;

}

return istr;

}

ostream& operator<<(ostream &ostr, const TSet &s) // вывод

{

bool f = true;

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

if (s.IsMember(i)){

ostr << i << " ";

f = false;

}

}

if (f) ostr << "Пустое множество";

return ostr;

}