Министерство образования и науки Российской Федерации

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчет по лабораторной работе**

**Структура хранения множества**

**Выполнил**:студент группы 381606-2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Тимакин Н.Е.

Подпись

**Проверил**: к.ф.-м.н., доц.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Баркалов К.А.

Подпись

Нижний Новгород 2017

**Введение**

Множество – это важное базовое математическое понятие, необходимое для реализации многих алгоритмов. Но оно не представлено как структура данных средствами языка C++, и поэтому необходимо сделать такую структуру.

В силу особенностей работы современных вычислительных систем очень удобно представлять множество как характеристический вектор h={1,0,0,1,…,0,1} с длиной равной мощности данного множество, и i-ая компонента – 1, если i-ый элемент принадлежит множеству, и 0 в противном случае.

**Постановка задачи**

Необходимо реализовать структуру данных множество с помощью классов Битовое Поле, представляющее характеристический вектор, и Множество, которое использует Битовое Поле. Добиться успешного выполнения тестовой программы, которая ищет и выводит простые числа с помощью множества, и прохождения всех предоставленных тестов.

**Описание структуры программы**

Программа содержит в себе 4 проекта:

1. “bitfield” (содержит объявление и реализацию классов TBitField (битовое поле) и TSet (множество))
2. “gtest” (библиотека Google Test)
3. “sample\_prime\_numbers” (тестовая программа)
4. “test\_bitfield” (тесты для классов TBitField и TSet)

**bitfield**

Содержит 4 файла:

1. tbitfield.h
2. tbitfield.cpp
3. tset.h
4. tset.cpp

**tbitfield.h** – объявление класса TBitField, описание его полей и методов

Поля:

private:

int BitLen; - максимальное количество элементов

TELEM \*pMem; - память для представления битового поля

int MemLen; - текущее количество элементов

Методы:

int GetMemIndex(const int n) const; - получить индекс бита с номером n

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

public:

TBitField(int len); - конструктор приведения типа

TBitField(const TBitField &bf); - конструктор копирования

~TBitField(); - деструктор

int GetLength(void) const; - получить длину (количество битов)

void SetBit(const int n); - установить бит

void ClrBit(const int n); - очистить бит

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

int operator==(const TBitField &bf) const; - перегрузка операции сравнения

int operator!=(const TBitField &bf) const; - перегрузка операции не равно

TBitField& operator=(const TBitField &bf); - перегрузка оператора присваивания

TBitField operator|(const TBitField &bf); - перегрузка операции побитовое «или»

TBitField operator&(const TBitField &bf); - перегрузка операции побитовое «и»

TBitField operator~(void); - перегрузка операции отрицание

friend istream &operator>>(istream &istr, TBitField &bf); - перегрузка операции ввода из потока

friend ostream &operator<<(ostream &ostr, const TBitField &bf); - перегрузка операции вывода в поток

**tbitfield.cpp –** реализация методов класса TBitField

**tset.h** – объявление класса TSet, описание его полей и методов

Поля:

private:

int MaxPower; - максимальная мощность множества

TBitField BitField; - битовое поле для хранения характеристического вектора

Методы:

public:

TSet(int mp); - конструктор приведения типа

TSet(const TSet &s); - конструктор копирования

TSet(const TBitField &bf); - конструктор приведения типа

operator TBitField(); - преобразование типа к битовому полю

int GetMaxPower(void) const; - получить максимальную мощность

void InsElem(const int Elem); - вставить элемент в множество

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

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

int operator== (const TSet &s) const; - перегрузка операции сравнения

int operator!= (const TSet &s) const; - перегрузка операции не равно

TSet& operator=(const TSet &s); - перегрузка оператора присваивания

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

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

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

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

TSet operator~ (void); - дополнение

friend istream &operator>>(istream &istr, TSet &bf); - перегрузка операции ввода из потока

friend ostream &operator<<(ostream &ostr, const TSet &bf); - перегрузка операции вывода в поток

**tset.cpp –** реализация методов класса TSet

Проекты “gtest”, “sample\_prime\_numbers” и “test\_bitfield” были предоставлены полностью готовые для работы, поэтому их подробное описание не приводится.

**Описание алгоритмов**

**Класс TBitField**

int GetMemIndex(const int n) const {вернуть n / размер типа данных TELEM \* 8}

TELEM GetMemMask (const int n) const {вернуть 1 сдвинутую на (остаток от деления n на (размер типа данных TELEM \* 8)) бит}

TBitField(int len) {если len<0 , выбросить исключение. Присвоить значения размерам поля, выделить память}

TBitField(const TBitField &bf) {присвоить поля bf, выделить память под массив, присвоить значения каждой ячейке массива}

~TBitField() {освободить память}

int GetLength(void) const {вернуть максимальное количество элементов}

void SetBit(const int n) {если n<0 или n>максимального количества элементов, то выбросить исключение. pMem[GetMemIndex(n)] |= GetMemMask(n);}

void ClrBit(const int n) { если n<0 или n>максимального количества элементов, то выбросить исключение. pMem[GetMemIndex(n)] &= ~(GetMemMask(n));}

int GetBit(const int n) const { если n<0 или n>максимального количества элементов, то выбросить исключение. Если (pMem[GetMemIndex(n)] & GetMemMask(n)) == 0 , то вернуть 0, иначе вернуть 1}

int operator==(const TBitField &bf) const {если максимальные размеры не равны, вернуть 0. Если хоть один элемент с первого до предпоследнего не равен, вернуть 0. Если хотя бы один бит до размера в битах не совпадает в последнем элементе, вернуть 0. Вернуть 1}

int operator!=(const TBitField &bf) const {вернуть отрицание проверки на равенство}

TBitField& operator=(const TBitField &bf) {присвоить максимальную длину. Если текущие длины не равны, то присвоить текущую длину, освободить память, выделить новую. Присвоить поочерёдно каждый элемент}

TBitField operator|(const TBitField &bf) {из длин в битах \*this и bf выбрать наибольшую и наименьшую. Создать битовое поле наибольшей длины под результат. В цикле от 0 до минимальной длины: если i-ый бит \*this дизъюнкция i-ый бит bf дают 1, то установить в результат i-ый бит, иначе очистить в результате i-ый бит. В цикле от минимальной длины до максимальной: если\_1 длина в битах \*this больше или равна длине в битах bf, если\_2 i-ый бит \*this – 1, то установить в результат i-ый бит, иначе\_2 очистить в результате i-ый бит, иначе\_1, если\_3 i-ый бит bf – 1, то установить в результат i-ый бит, иначе\_3 очистить в результате i-ый бит. Вернуть результат}

TBitField operator&(const TBitField &bf) { из максимальных длин \*this и bf выбрать наибольшую и наименьшую. Создать битовое поле наибольшей длины под результат. В цикле от 0 до предпоследнего элемента pMem: в i-ую компоненту pMem результата записать результат конъюнкции i-ой компоненты pMem \*this и i-ой компоненты pMem bf. В цикле для каждого бита последнего элемента: если i-ый бит \*this конъюнкция i-ый бит bf дают 1, то установить в результат i-ый бит, иначе очистить в результате i-ый бит. Вернуть результат}

TBitField operator~(void) {создать битовое поле максимальной длины под результат. В цикле к каждому элементу pMem применить побитовое отрицание и присвоить в результат в соответствующем порядке. Вернуть результат}

friend istream &operator>>(istream &istr, TBitField &bf) {пока текущий символ 0 или 1, вводить символ из потока. Вернуть поток}

friend ostream &operator<<(ostream &ostr, const TBitField &bf) {в цикле от 0 до длины в битах: выводить в поток i-ый бит bf. Вернуть поток}

**Класс TSet**

TSet(int mp) {присвоить максимальной мощности значение mp}

TSet(const TSet &s) {вызвать конструктор копирования для битового поля с параметром s.BitField, присвоить максимальной мощности значение максимальной мощности s}

TSet(const TBitField &bf) { вызвать конструктор копирования для битового поля с параметром bf, присвоить максимальной мощности значение максимальной мощности bf}

operator TBitField() {вернуть битовое поле}

int GetMaxPower(void) const {вернуть максимальную мощность}

void InsElem(const int Elem) {вызвать метод установки бита в позицию Elem для битового поля}

void DelElem(const int Elem) { вызвать метод удаления бита из позиции Elem для битового поля }

int IsMember(const int Elem) const {вызвать метод GetBit(Elem) для битового поля и вернуть его значение}

int operator== (const TSet &s) const {вернуть результат сравнения битового поля \*this и битового поля s}

int operator!= (const TSet &s) const {вернуть результат проверки на неравенство битового поля \*this и битового поля s}

TSet& operator=(const TSet &s) {присвоить значения максимальной мощности и битового поля s в this. Вернуть \*this}

TSet operator+ (const int Elem) {создать множество под результат, равный данному, добавить элемент с номером Elem, вернуть результат}

TSet operator- (const int Elem) {создать множество под результат, равный данному, исключить элемент с номером Elem, вернуть результат}

TSet operator+ (const TSet &s) {вернуть результат операции «или» над битовыми полями s и \*this}

TSet operator\* (const TSet &s) {вернуть результат операции «и» над битовыми полями s и \*this }

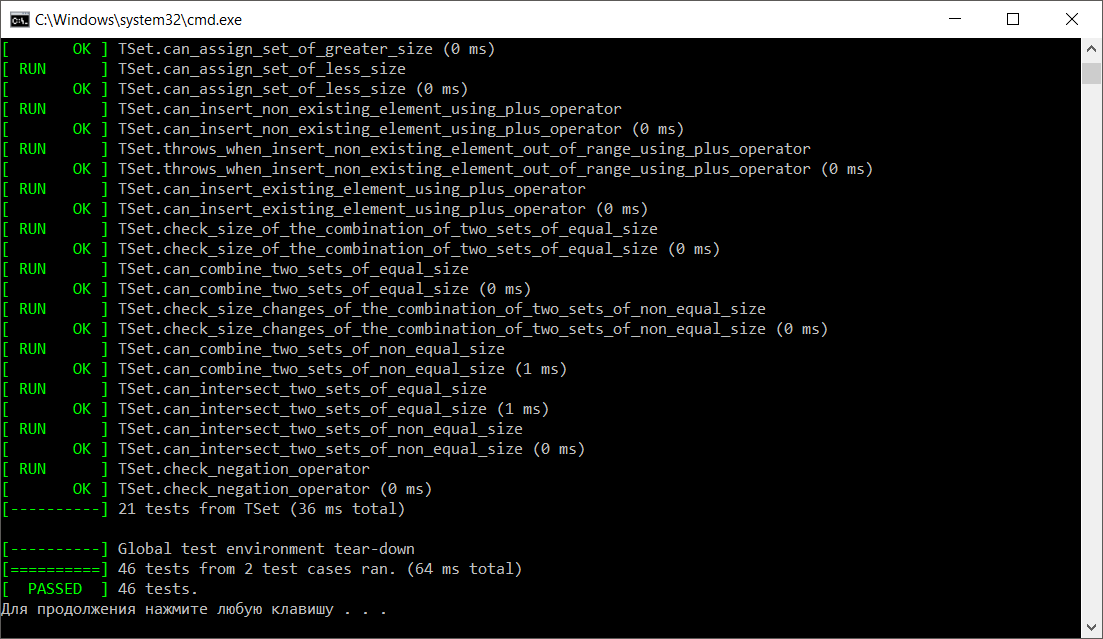
TSet operator~ (void) {вернуть результат операции «отрицание» над битовым полем}

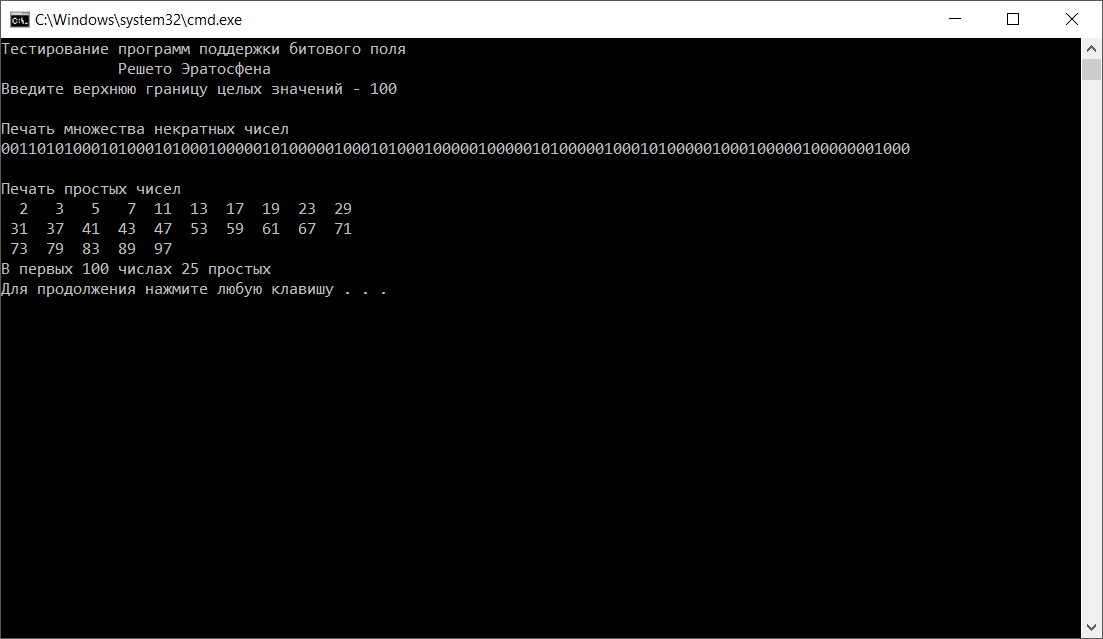
friend istream &operator>>(istream &istr, TSet &bf) {вводить числа в поток пока текущее больше или равно 0: если текущее равно 0, то очистить бит, иначе установить бит. Вернуть поток}

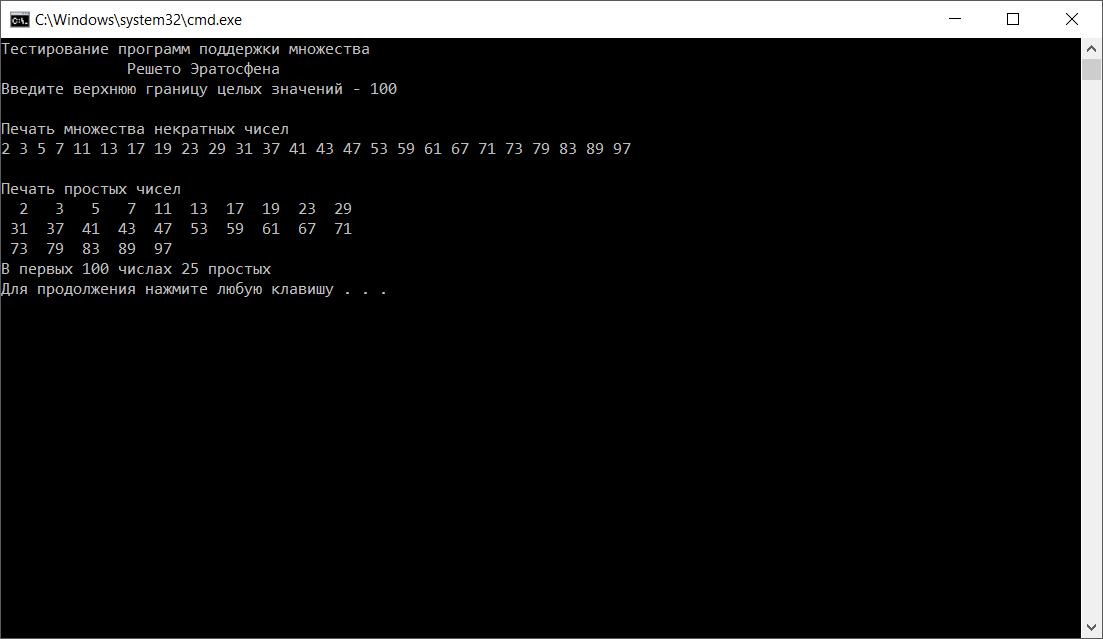
friend ostream &operator<<(ostream &ostr, const TSet &bf) {в цикл от 0 до длины битового поля: если i-ый элемент входит в множество, то вывести его в поток. Вернуть поток}

**Результаты**

Как видно из скриншотов ниже, все тесты выполняются корректно, и тестовая программа работает правильно в обоих вариантах: и с помощью битового поля, и с помощью множества.







**Вывод**

Все тестовые программы завершаются корректно, все задачи, поставленные ранее, выполнены, а значит разработку структуры данных «множество» можно считать успешной.

**Литература**

Брайан Керниган, Деннис Ритчи «Язык программирования Си»

Брюс Эккель «Философия С++. Введение в стандартный С++»

Стивен Прата «Язык программирования С++. Лекции и упражнения»

**Приложение**

**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

// 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(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**

#include "tbitfield.h"

TBitField::TBitField(int len)

{

if (len < 0) throw len;

BitLen = len;

MemLen = BitLen / (8 \* sizeof(TELEM)) + 1;

pMem = new TELEM[MemLen];

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

pMem[i] = 0;

}

}

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

{

BitLen = bf.BitLen;

MemLen = bf.MemLen;

pMem = new TELEM[MemLen];

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

{

return n / (sizeof(TELEM) \* 8);

}

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

{

return 1 << n % (sizeof(TELEM) \* 8);

}

// 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

{

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

throw n;

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

}

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

{

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

throw n;

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

}

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

{

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

throw n;

if ((pMem[GetMemIndex(n)] & GetMemMask(n)) == 0) return 0;

else return 1;

}

// битовые операции

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

{

BitLen = bf.BitLen;

if (MemLen != bf.MemLen) {

MemLen = bf.MemLen;

delete[] pMem;

pMem = new TELEM[MemLen];

}

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

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

}

return \*this;

}

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

{

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

int i = 0;

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

if (pMem[i] != bf.pMem[i]) return 0;

}

for (i = (MemLen - 1) \* sizeof(TELEM); 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"

{

int maxlen, minlen,i;

if (BitLen >= bf.BitLen){

maxlen = BitLen;

minlen = bf.BitLen;

}

else {

maxlen = bf.BitLen;

minlen = BitLen;

}

TBitField res(maxlen);

for (i = 0; i < minlen; i++) {

if ((GetBit(i) | bf.GetBit(i)) == 1) {

res.SetBit(i);

}

else {

res.ClrBit(i);

}

}

for (i = minlen; i < maxlen; i++) {

if (BitLen >= bf.BitLen) {

if (GetBit(i) == 1) {

res.SetBit(i);

}

else {

res.ClrBit(i);

}

}

else {

if (bf.GetBit(i) == 1) {

res.SetBit(i);

}

else {

res.ClrBit(i);

}

}

}

return res;

}

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

{

int maxlen, minlen, i;

if (BitLen >= bf.BitLen) {

maxlen = BitLen;

minlen = bf.BitLen;

}

else {

maxlen = bf.BitLen;

minlen = BitLen;

}

TBitField res(maxlen);

for (i = 0; i < minlen/(8 \* sizeof(TELEM)); i++) {

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

}

for (i = (minlen / (8 \* sizeof(TELEM))) \* 32; i < minlen; i++) {

if (GetBit(i) & bf.GetBit(i) == 1) res.SetBit(i);

else res.ClrBit(i);

}

return res;

}

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

{

TBitField res(BitLen);

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

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

}

return res;

}

// ввод/вывод

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

{

char sym;

int i = 0;

istr >> sym;

while ((sym == '0') || (sym == '1')) {

if (sym == '1') bf.SetBit(i);

else bf.ClrBit(i);

i++;

istr >> sym;

}

return istr;

}

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

{

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

ostr << bf.GetBit(i);

}

return ostr;

}

**tset.h**

#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**

#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 (BitField == s.BitField);

}

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

{

return (BitField != s.BitField);

}

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

{

return (BitField | s.BitField);

}

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

{

return (BitField & s.BitField);

}

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

{

return ~BitField;

}

// перегрузка ввода/вывода

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

{

/\*int i = 0;

istr >> i;

while (i >= 0) {

if (i == 0) s.BitField.ClrBit(i);

else s.BitField.SetBit(i);

istr >> i;

}\*/

int member, i = 0;

istr >> member;

while (member >= 0) {

if (member == 0)

s.BitField.ClrBit(i);

else

s.BitField.SetBit(i);

i++;

istr >> member;

}

return istr;

}

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

{

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

if (s.IsMember(i) == 1) ostr << i << " ";

}

return ostr;

}