Федеральное агентство по образованию Российской Федерации
Государственное образовательное учреждение
высшего профессионального образования
Нижегородский государственный университет им. Н.И. Лобачевского

Факультет вычислительной математики и кибернетики

Отчёт по лабораторной работе

Множества

Выполнил:
студент ф-та ИИТММ гр. 381908-01

Козел С. А.

Проверил:
ассистент каф. МОСТ, ИИТММ

Лебедев И.Г.

Нижний Новгород
2020 г.

Содержание

Введение

Множества (битовые поля) - некоторое количество бит, расположенных последовательно в памяти, значение которых процессор не способен прочитать из-за особенностей аппаратной реализации.

Постановка задачи

Цель данной работы - разработка структуры данных для битовых полей с использованием массива.

Выполнение работы предполагает решение следующих задач:

1. Реализация класса битового поля TBitField
2. Реализация класса для хранения битового поля TSet
3. Реализация методов для ввода/вывода структуры данных в файл
4. Публикация исходных кодов в личном репозиторий на GitHub.

Руководство пользователя

Пользователю нужно запустить файл mp2-lab1-set.exe.

Откроется консольное приложение для тестирования битовых полей.

Программа покажет функциональность каждой функции по средствам вывода данных в консоль.

Руководство программиста

Описание структуры программы

Программа состоит из следующих модулей:

- Приложение mp2-lab1-set
- Статическая библиотека TBitField:
  - TBitField.h – описание класса битового поля
- Статическая библиотека TSet:
  - TSet.h – описание класса работы с битовым полем
- Приложение main:
  - main.cpp – тестирование работы программы

Описание структур данных

Класс TBitField:

1. TBitField(int len) – конструктор по умолчанию;
2. TBitField(const TBitField &bf) – конструктор копирования;
3. ~TBitField() - деструктор;
4. int GetLength(void) const - получить длину (к-во битов);
5. void SetBit(const int n) - установить бит;
6. void ClrBit(const int n) - очистить бит;
7. int GetBit(const int n) const - получить значение бита;
8. bool operator==(const TBitField &bf) const - сравнение;
9. bool operator!=(const TBitField &bf) const – сравнение;
10. TBitField& operator=(const TBitField &bf) – присваивание;
11. TBitField operator|(const TBitField &bf) - операция "или";
12. TBitField operator&(const TBitField &bf) - операция "и";
13. TBitField operator~(void) – отрицание;
14. void InFile(std::string file_name) – прочитать с файла;
15. void FromFile(std::string file_name) – вывести в файл;
16. friend istream &operator>>(istream &istr, TBitField &bf) – перегрузка ввода;
17. friend ostream &operator<<(ostream &ostr, const TBitField &bf) – перегрузка вывода;

Класс TSet:

1. TSet(int mp) – конструктор по умолчанию;
2. TSet(const TSet &s) - конструктор копирования;
3. TSet(const TBitField &bf) - конструктор преобразования типа;
4. operator TBitField() - преобразование типа к битовому полю;
5. int GetMaxPower(void) const - максимальная мощность множества;
6. void InsElem(const int Elem) - включить элемент в множество
7. void DelElem(const int Elem) - удалить элемент из множества;
8. int IsMember(const int Elem) const - проверить наличие элемента в множестве;
9. int operator== (const TSet &s) const – сравнение;
10. int operator!= (const TSet &s) const – сравнение;
11. TSet& operator=(const TSet &s) – присваивание;
12. TSet operator+ (const int Elem) - объединение с элементом
13. TSet operator- (const int Elem) - разность с элементом;
14. TSet operator+ (const TSet &s) – объединение;
15. TSet operator* (const TSet &s) – пересечение;
16. TSet operator~ (void) – дополнение;

17. friend istream &operator>>(istream &istr, TSet &bf) – перегрузка ввода;
18. friend ostream &operator<<(ostream &ostr, const TSet &bf) – перегрузка ввода;
19. void InFile(std::string file_name) – прочитать с файла;
20. void FromFile(std::string file_name) – вывести в файл;
21. void ChangeElements(int n, int new_elem) – изменить элемент;
22. TSet getElements(int K) – получить элемент по позиции;

Описание алгоритмов

Принцип работы битового поля.

Битовые поля обеспечивают удобный доступ к отдельным битам данных. Они позволяют формировать объекты с длиной, не кратной байту. Что в свою очередь позволяет экономить память, более плотно размещая данные.

Битовое поле не может существовать само по себе. Оно может быть только элементом структуры или объединения.

Так же битовые поля имеют некоторые ограничения. Нельзя получить адрес переменной битового поля. Переменные битового поля не могут помещаться в массив. Переходя с компьютера на компьютер нельзя быть уверенным в порядке изменения битов (слева направо или справа налево). Любой код, использующий битовые поля, зависит от компьютера.

Эксперименты

```
Тестирование программ поддержки битового поля
            Решето Эратосфена
Введите верхнюю границу целых значений - 5

Печать множества некратных чисел
001101

Печать простых чисел
  2   3   5
В первых 5 числах 3 простых
File opened and rewritten!
File is open and read!
0011010
```

Заключение

При выполнении данной работы мною была полностью изучена и успешно реализована структура данных битовые поля.

Литература

1. https://ru.wikipedia.org/wiki/Битовое_поле
2. https://docs.microsoft.com/ru-ru/cpp/cpp/cpp-bit-fields?view=msvc-160&viewFallbackFrom=vs-2019
3. http://www.c-cpp.ru/books/bitovye-polya

Приложения

Приложение 1

TBitField.h

```
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 pMem для бита n       (#O2)
  TELEM GetMemMask (const int n) const; // битовая маска для бита n      (#O3)
public:
  TBitField(int len);                   //                               (#O1)
  TBitField(const TBitField &bf);       //                               (#П1)
  ~TBitField();                         //                                (#C)

  // доступ к битам
  int GetLength(void) const;      // получить длину (к-во битов)          (#O)
  void SetBit(const int n);       // установить бит                      (#O4)
  void ClrBit(const int n);       // очистить бит                        (#П2)
  int  GetBit(const int n) const; // получить значение бита              (#Л1)

  // битовые операции
  bool operator==(const TBitField &bf) const; // сравнение                 (#O5)
  bool operator!=(const TBitField &bf) const; // сравнение
  TBitField& operator=(const TBitField &bf); // присваивание              (#П3)
  TBitField  operator|(const TBitField &bf); // операция "или"            (#O6)
  TBitField  operator&(const TBitField &bf); // операция "и"              (#Л2)
  TBitField  operator~(void);                // отрицание                  (#C)

  void InFile(std::string file_name);
  void FromFile(std::string file_name);

  friend istream &operator>>(istream &istr, TBitField &bf);       //      (#O7)
  friend ostream &operator<<(ostream &ostr, const TBitField &bf); //      (#П4)
};
// Структура хранения битового поля
//   бит.поле - набор битов с номерами от 0 до BitLen
//   массив pMem рассматривается как последовательность MemLen элементов
//   биты в эл-тах pMem нумеруются справа налево (от младших к старшим)
// О8 Л2 П4 С2

#endif
```

```cpp
TBitField::TBitField(int len)
{
    if (len < 0)
        throw std::logic_error("Input error: invalide value of bitfield length in constructor");
    else
    {
        BitLen = len;
        MemLen = len / (sizeof(TELEM) * 8) + 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;
    pMem = NULL;
}

int TBitField::GetMemIndex(const int n) const // индекс Мем для бита n
{
    if (n < 0)
        throw std::out_of_range("Input error: invalide value of bitfield in GetMemIndex");
    else
        return(n / (sizeof(TELEM) * 8));
}

TELEM TBitField::GetMemMask(const int n) const // битовая маска для бита n
{
    if (n < 0)
        throw std::out_of_range("Input error: invalide value of bitfield in GetMemMask");
    else
    {
        int tmp = n % (sizeof(int) * 8);
        TELEM mask = 1 << tmp;
        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
{
    if (n < 0 || n > BitLen)
    {
        throw std::out_of_range("Input error: invalide value of bitfield in SetBit");
    }
    pMem[GetMemIndex(n)] |= GetMemMask(n);
}

void TBitField::ClrBit(const int n) // очистить бит
{
    if (n < 0 || n > BitLen)
    {
        throw std::out_of_range("Input error: invalide value of bitfield in ClrBit");
    }
    pMem[GetMemIndex(n)] &= ~GetMemMask(n);
}

int TBitField::GetBit(const int n) const // получить значение бита
{
    if (n < 0 || n > BitLen)
    {
        throw std::out_of_range("Input error: invalide value of bitfield in GetBit");
    }
    return pMem[GetMemIndex(n)] & GetMemMask(n);
}
 // битовые операции
```

```cpp
TBitField& TBitField::operator=(const TBitField& bf) // присваивание
{
    if (!(*this == bf))
    {
        if (MemLen != bf.MemLen)
        {
            delete[] pMem;
            MemLen = bf.MemLen;
            pMem = new TELEM[MemLen];
        }
        BitLen = bf.BitLen;
        for (int i = 0; i < MemLen; i++)
        {
            pMem[i] = bf.pMem[i];
        }
        return *this;
    }
    else return *this;
}

bool TBitField::operator==(const TBitField& bf) const // сравнение
{
    if (BitLen != bf.BitLen)
    {
        return false;
    }
    else
    {
        for (int i = 0; i < MemLen; i++)
        {
            if (pMem[i] != bf.pMem[i])
            {
                return false;
            }
            return true;
        }
    }
}

bool TBitField::operator!=(const TBitField& bf) const // сравнение
{
    if (!(*this == bf))
        return true;
    else return false;
}

TBitField TBitField::operator|(const TBitField& bf) // операция "или"
{
    int len = BitLen;
    if (bf.BitLen > len)
        len = bf.BitLen;
    TBitField temp(len);
    for (int i = 0; i < MemLen; i++)
    {
        temp.pMem[i] = pMem[i];
    }
    for (int i = 0; i < bf.MemLen; i++)
    {
        temp.pMem[i] |= bf.pMem[i];
    }

    return temp;
}

TBitField TBitField::operator&(const TBitField& bf) // операция "и"
{
    TBitField tmp(bf.BitLen > BitLen ? bf.BitLen : BitLen);

    for (int i = 0; i < MemLen; i++)
    {
        tmp.pMem[i] = pMem[i];
    }
    for (int i = 0; i < bf.MemLen; i++)
    {
        tmp.pMem[i] &= bf.pMem[i];
    }

    return tmp;
}
```

```cpp
TBitField TBitField::operator&(const TBitField& bf) // операция "и"
{
    TBitField tmp(bf.BitLen > BitLen ? bf.BitLen : BitLen);

    for (int i = 0; i < MemLen; i++)
    {
        tmp.pMem[i] = pMem[i];
    }
    for (int i = 0; i < bf.MemLen; i++)
    {
        tmp.pMem[i] &= bf.pMem[i];
    }

    return tmp;
}

TBitField TBitField::operator~(void) // отрицание
{
    TBitField tmp(*this);

    for (int i = 0; i < tmp.BitLen; i++)
    {
        if (tmp.GetBit(i))
        {
            tmp.ClrBit(i);
        }
        else
        {
            tmp.SetBit(i);
        }
    }
    return tmp;
}

void TBitField::InFile(std::string file_name)
{
    fstream fs;
    fs.open(file_name, fstream::in | fstream::out);
    if (!fs.is_open())
    {
        std::cout << "Error open file!" << std::endl;
    }
    else
    {
        for (int i = 0; i < GetLength(); i++)
        {
            if (GetBit(i))
            {
                fs << "1";
            }
            else
            {
                fs << "0";
            }
        }
        std::cout << "File opened and rewritten!" << std::endl;
    }
    fs.close();
}

void TBitField::FromFile(std::string file_name)
{
    fstream fs;
    fs.open(file_name, fstream::in | fstream::out);
    if (!fs.is_open())
    {
        std::cout << "Error open file!" << std::endl;
    }
    else
    {
        std::string str;
        fs >> str;
        TBitField tmp(str.size());
        for (int i = 0; i < str.size(); i++)
        {
            if (str[i] == '1')
            {
                tmp.SetBit(i);
            }
            else
            {
                tmp.ClrBit(i);
            }
        }
        *this = tmp;
        std::cout << "File is open and read!" << std::endl;
    }
    fs.close();
}
```

```
// ввод/вывод
istream& operator>>(istream& istr, TBitField& bf) // ввод
{
    char sym;
    do {
        istr >> sym;
    } while (sym != ' ');
    int i = 0;
    while (true)
    {
        istr >> sym;
        if (sym == '0')
            bf.ClrBit(i++);
        else if (sym == '1')
            bf.SetBit(i++);
        else break;
    }
    return istr;
}

ostream& operator<<(ostream& ostr, const TBitField& bf) // вывод
{
    int len = bf.GetLength();
    for (int i = 0; i < len; i++)
        if (bf.GetBit(i)) ostr << '1';
        else ostr << '0';
    return ostr;
}
```

Приложение 2

TSet.h

```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);

  void InFile(std::string file_name);
  void FromFile(std::string file_name);

  void ChangeElements(int n, int new_elem);
  TSet getElements(int K);
};
std::string DelBadSymb(std::string str);
int CountNumOfDig(const std::string& str);
void StrToArrStr(std::string str, std::string* arr);
int StrToInt(std::string str);
void ArrStrToArrInt(std::string* arrStr, int* arrInt, int len);
int FindMaxElem(int* arr, int len);

#endif
```

```cpp
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);
}
```

```cpp
TSet& TSet::operator=(const TSet& s) // присваивание
{
    if (!(*this == s))
    {
        BitField = s.BitField;
        MaxPower = s.MaxPower;
        return *this;
    }
    else return *this;
}

int TSet::operator==(const TSet& s) const // сравнение
{
    return BitField == s.BitField;
}

int TSet::operator!=(const TSet& s) const // сравнение
{
    return !(*this == s);
}

TSet TSet::operator+(const TSet& s) // объединение
{
    TSet tField(std::max(MaxPower, s.MaxPower));
    tField.BitField = BitField | s.BitField;
    return tField;
}

TSet TSet::operator+(const int Elem) // объединение с элементом
{
    TSet tField(*this);
    tField.BitField.SetBit(Elem);
    return tField;
}

TSet TSet::operator-(const int Elem) // разность с элементом
{
    TSet tField(*this);
    tField.BitField.ClrBit(Elem);
    return tField;
}

TSet TSet::operator*(const TSet& s) // пересечение
{
    TSet tField(max(s.MaxPower, MaxPower));
    tField.BitField = BitField & s.BitField;
    return tField;
}
```

```cpp
TSet TSet::operator~(void) // дополнение
{
    TSet tField(MaxPower);
    tField.BitField = ~BitField;
    return tField;
}

void TSet::InFile(std::string file_name)
{
    fstream fs;
    fs.open(file_name, fstream::in | fstream::out);
    if (!fs.is_open())
    {
        std::cout << "Error open file!" << std::endl;
    }
    else
    {
        fs << "{";
        for (int i = 0; i < BitField.GetLength(); i++)
        {
            if (BitField.GetBit(i))
            {
                if (i == BitField.GetLength() - 1)
                {
                    fs << i;
                }
                else
                {
                    fs << i;
                    fs << ", ";
                }
            }
        }
        fs << "}";
        std::cout << "File opened and rewritten!" << std::endl;
    }
    fs.close();
}
```

```cpp
void TSet::FromFile(std::string file_name)
{
    fstream fs;
    fs.open(file_name, fstream::in | fstream::out);
    if (!fs.is_open())
    {
        std::cout << "Error open file!" << std::endl;
    }
    else
    {
        std::string str;
        getline(fs, str);
        str = DelBadSymb(str);
        int NumOfDig = CountNumOfDig(str);
        std::string* arrStr = new std::string[NumOfDig];
        int* arrInt = new int[NumOfDig];
        StrToArrStr(str, arrStr);
        ArrStrToArrInt(arrStr, arrInt, NumOfDig);
        int Max = FindMaxElem(arrInt, NumOfDig);
        TSet res(Max + 1);
        for (int i = 0; i < NumOfDig; i++)
        {
            res.InsElem(arrInt[i]);
        }
        *this = res;
        delete[] arrStr;
        delete[] arrInt;
        std::cout << "File is open and read!" << std::endl;
    }
    fs.close();
}

void TSet::ChangeElements(int n, int new_elem)
{
    if (n == new_elem) {}           // Ecli nomer elementa = new element -> nichego ne delayem
    else {
        if ((n < 0) || (new_elem < 0 || new_elem > MaxPower))
        {
            throw std::out_of_range("Input error: invalide value of tset in ChangeElements");
        }
        else {
            int tmp = 0;
            int count = 0;
            for (int i = 0; count < n; i++)
            {
                if (BitField.GetBit(i))
                {
                    tmp = i;
                    count++;
                }
            }
            DelElem(tmp);
            InsElem(new_elem);
        }
    }
}
```

```cpp
TSet TSet::getElements(int K)
{
    TSet res(MaxPower + 1);
    for (int i = 0; i < BitField.GetLength(); i++)
    {
        if (BitField.GetBit(i))
        {
            if (i % K == 0)
            {
                res.InsElem(i); // std::cout << i << " ";
            }
        }
    }
    return res;
}

 // перегрузка ввода/вывода
istream& operator>>(istream& istr, TSet& s) // ввод
{
    string text;
    string tmp;
    getline(istr, tmp);
    int space = 1;
    for (int i = 0; i < tmp.length(); i++)
    {
        if (tmp[i] != '{' && tmp[i] != '}' && tmp[i] != ',')
        {
            text += tmp[i];                                  // Chistyui text
        }
        if (tmp[i] == ' ')
        {
            space++;
        }
    }

    string* delSpace = new string[space];
    int index = 0;
    for (int i = 0; i < text.length(); i++)
    {
        if (text[i] != ' ')
        {
            delSpace[index] += text[i];
        }
        else index++;
    }

    int* textConvertInt = new int[space];

    int maxElement = 0;
    for (int i = 0; i < space; i++)
    {
        textConvertInt[i] = StrToInt(delSpace[i]);
        if (textConvertInt[i] > maxElement)
        {
            maxElement = textConvertInt[i];
        }
    }

    TSet res(maxElement + 1);
    for (int i = 0; i < space; i++)
    {
        res.InsElem(textConvertInt[i]);
    }

    delete[] delSpace;
    delete[] textConvertInt;

    s = res;

    return istr;
}

ostream& operator<<(ostream& ostr, const TSet& s) // вывод
{
    cout << "{";
    for (int i = 0; i < s.BitField.GetLength(); i++)
    {
        if (s.BitField.GetBit(i))
        {
            if (i == s.BitField.GetLength() - 1)
            {
                ostr << i;
            }
            else
            {
                ostr << i;
                ostr << ", ";
            }
        }
    }
    cout << "}";

    return ostr;
}
```

```cpp
std::string DelBadSymb(std::string str)
{
    std::string tmp = "";
    for (int i = 0; i < str.length(); i++)
    {
        if (str[i] != '{' && str[i] != '}' && str[i] != ',')
        {
            tmp += str[i];
        }
    }
    return tmp;
}

int CountNumOfDig(const std::string& str)
{
    int result = 1;
    for (int i = 0; i < str.length(); i++)
    {
        if (str[i] == ' ')
        {
            result += 1;
        }
    }
    return result;
}

void StrToArrStr(std::string str, std::string* arr)
{
    int count = 0;
    for (int i = 0; i < str.length(); i++)
    {
        if (str[i] != ' ')
        {
            arr[count] += str[i];
        }
        else
            count++;
    }
}

int StrToInt(std::string str)
{
    int result = 0;
    for (int i = 0; i < str.length(); i++)
    {
        result = result * 10 + (str[i] - '0');
    }

    return result;
}

void ArrStrToArrInt(std::string* arrStr, int* arrInt, int len)
{
    for (int i = 0; i < len; i++)
    {
        arrInt[i] = StrToInt(arrStr[i]);
    }
}

int FindMaxElem(int* arr, int len)
{
    int max = 0;
    for (int i = 0; i < len; i++)
    {
        if (arr[i] > max)
        {
            max = arr[i];
        }
    }
    return max;
}
```

Приложение 3

Main.cpp

```
#ifndef USE_SET // Использовать класс TBitField

#include "tbitfield.h"

int main()
{
  int n, m, k, count;

  setlocale(LC_ALL, "Russian");
  cout << "Тестирование программ поддержки битового поля" << endl;
  cout << "              Решето Эратосфена" << endl;
  cout << "Введите верхнюю границу целых значений - ";
  cin  >> n;
  TBitField s(n + 1);
  // заполнение множества
  for (m = 2; m <= n; m++)
    s.SetBit(m);



  // проверка до sqrt(n) и удаление кратных
  for (m = 2; m * m <= n; m++)
    // если m в s, удаление кратных
    if (s.GetBit(m))
      for (k = 2 * m; k <= n; k += m)
        if (s.GetBit(k))
          s.ClrBit(k);
  // оставшиеся в s элементы - простые числа
  cout << endl << "Печать множества некратных чисел" << endl << s << endl;
  cout << endl << "Печать простых чисел" << endl;
  count = 0;
  k = 1;
  for (m = 2; m <= n; m++)
    if (s.GetBit(m))
    {
      count++;
      cout << setw(3) << m << " ";
      if (k++ % 10 == 0)
        cout << endl;
    }
  cout << endl;
  cout << "В первых " << n << " числах " << count << " простых" << endl;

  s.InFile(PathForFile);

  TBitField s_2(n + 1);
  s_2.FromFile(PathForFile);
  std::cout << s_2;
}
#else
```

```cpp
#include "tset.h"

int main()
{
  int n, m, k, count;

  setlocale(LC_ALL, "Russian");
  cout << "Тестирование программ поддержки множества" << endl;
  cout << "               Решето Эратосфена" << endl;
  cout << "Введите верхнюю границу целых значений - ";
  cin  >> n;
  TSet s(n + 1);
  // заполнение множества
  //for (m = 2; m <= n; m++)


  s.InsElem(2);
  s.InsElem(5);
  s.InsElem(7);
  s.InsElem(10);
  s.InsElem(11);
  s.InsElem(12);
  s.InsElem(17);
  s.InsElem(19);


  // проверка до sqrt(n) и удаление кратных

  std::cout << "\n";
  std::cout << s;
  std::cout << "\n";
  s.InFile(PathForFile);
  //s.getElements(5);
  std::cout << "\n";

  for (m = 2; m * m <= n; m++)
    // если м в s, удаление кратных
    if (s.IsMember(m))
      for (k = 2 * m; k <= n; k += m)
       if (s.IsMember(k))
         s.DelElem(k);
  // оставшиеся в s элементы - простые числа
  cout << endl << "Печать множества некратных чисел" << endl << s << endl;
  cout << endl << "Печать простых чисел" << endl;
  count = 0;
  k = 1;
  for (m = 2; m <= n; m++)
    if (s.IsMember(m))
    {
      count++;
      cout << setw(3) << m << " ";
      if (k++ % 10 == 0)
        cout << endl;
    }
  cout << endl;
  cout << "В первых " << n << " числах " << count << " простых" << endl;


  TSet s_2(n + 1);
  s_2.FromFile(PathForFile);


  std::cout << "\n";
  std::cout << s_2;
  s_2.ChangeElements(2, 18);
  std::cout << "\n";
  std::cout << "\n";
  std::cout << s_2;
  std::cout << "\n";
  s_2.getElements(5);

  TSet s_3(n + 1);
  s_3 = s_2.getElements(5);
  std::cout << s_3;

 // s_2.InFile(PathForFile);
 // s_2.getElements(2);
}

#endif
```