МИНИСТЕРСТВО НАУКИ И ВЫСШЕГООБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

**ЛАБОРАТОРНАЯ РАБОТА**

на тему:

**«Битовые поля и множества»**

**Выполнил(а):** студент(ка) группы \_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Хохлов А.Д./

Подпись

**Проверил:** к.т.н, доцент каф. ВВиСП

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Кустикова В.Д./

Подпись

Нижний Новгород  
2023

Содержание

[Введение 3](#_Toc149043756)

[1 Постановка задачи 4](#_Toc149043757)

[2 Руководство пользователя 5](#_Toc149043758)

[2.1 Приложение для демонстрации работы битовых полей 5](#_Toc149043759)

[2.2 Приложение для демонстрации работы множеств 6](#_Toc149043760)

[2.3 «Решето Эратосфено» 7](#_Toc149043761)

[3 Руководство программиста 9](#_Toc149043762)

[3.1 Описание алгоритмов 9](#_Toc149043763)

[3.1.1 Битовые поля 9](#_Toc149043764)

[3.1.2 Множества 11](#_Toc149043765)

[3.1.3 «Решето Эратосфена» 13](#_Toc149043766)

[3.2 Описание программной реализации 14](#_Toc149043767)

[3.2.1 Описание класса TBitField 14](#_Toc149043768)

[3.2.2 Описание класса TSet 17](#_Toc149043769)

[Заключение 22](#_Toc149043770)

[Литература 23](#_Toc149043771)

[Приложения 24](#_Toc149043772)

[Приложение А. Реализация класса TBitField 24](#_Toc149043773)

[Приложение Б. Реализация класса TSet 26](#_Toc149043774)

# Введение

В современном мире информационных технологий большую роль играют операции с большими объемами данных. Одной из важных операций является эффективная работа с битовыми множествами. Битовое множество - это структура данных, которая позволяет компактно хранить и оперировать набором битов. Такая структура данных может быть использована в различных областях: от алгоритмов сжатия данных до обработки больших объемов информации.

Актуальность и применяемость данной лабораторной работы состоит в том, что она позволяет познакомиться с основными принципами работы с битовыми множествами и реализовать их на языке программирования C++. Использование битовых множеств позволяет существенно ускорить операции с большими объемами данных и снизить требования по памяти.

Таким образом, данная лабораторная работа является актуальной и полезной для студентов и специалистов в области информационных технологий, которые имеют необходимость эффективно работать с битами и битовыми множествами.

# Постановка задачи

Цель: Целью данной лабораторной работы является создание битового множества на языке программирования C++. В рамках работы необходимо разработать классы TBitField и TSet, которые будут предоставлять функциональность для работы с битовыми полями и множествами. Основной задачей является реализация основных операций с битовыми множествами, таких как установка, сброс, проверка наличия бита, а также операции с множествами, такие как объединение, пересечение и разность.

2. Задачи данной лабораторной работы:

- Разработка класса TBitField, который будет предоставлять функциональность для работы с битовыми полями.

- Реализация основных операций с битовыми полями, включая установку, сброс и проверку наличия определенного бита.

- Определение класса TSet, который будет предоставлять функциональность для работы с множествами

- Реализация основных операций с множествами, включая объединение, пересечение и разность.

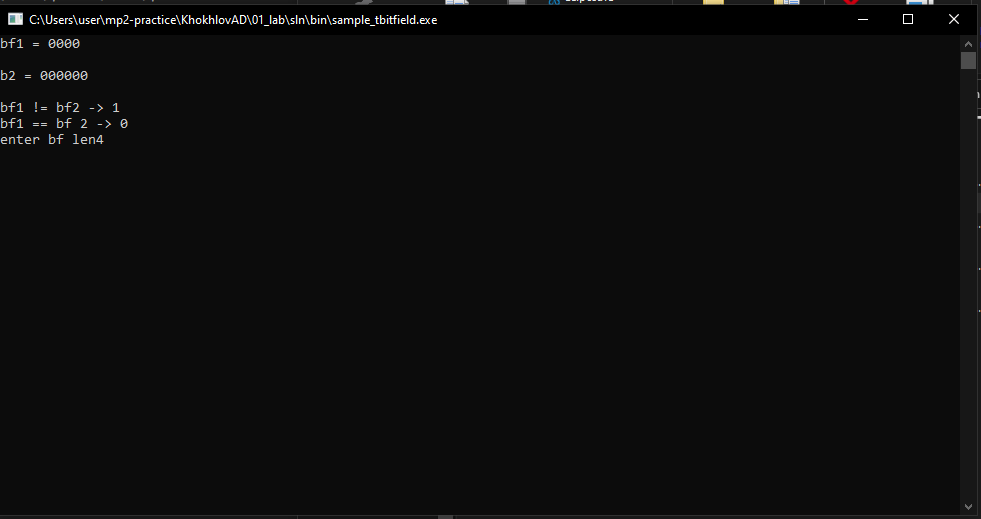
- Проверка и демонстрация работы разработанных классов с помощью приложений для работы с битовыми полями, множествами и решением задачи "Решето Эратосфена".

- Написание отчета о выполненной лабораторной работе, включая описание алгоритмов, программной реализации и результатов работы.

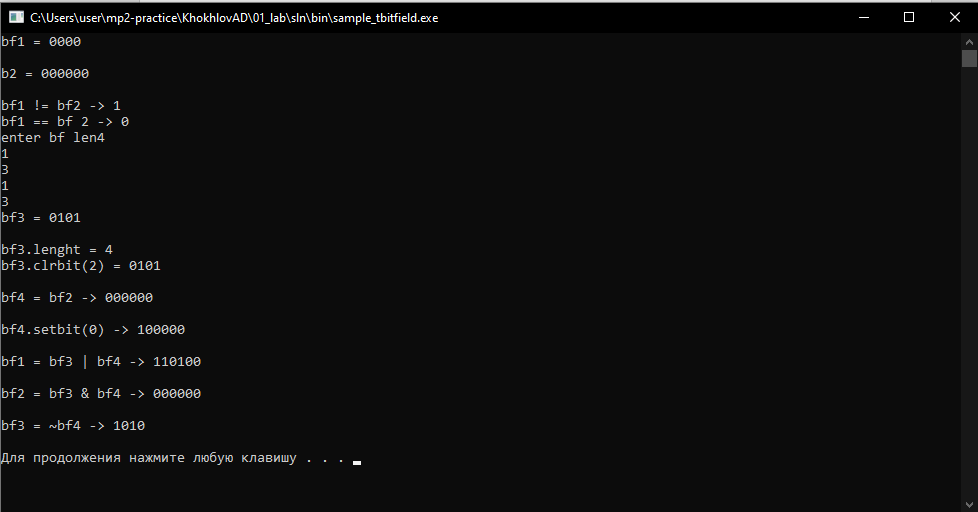
# Руководство пользователя

## Приложение для демонстрации работы битовых полей

1. Запустите приложение с названием sample\_tbitfield.exe. В результате появится окно, показанное ниже (рис. 1).



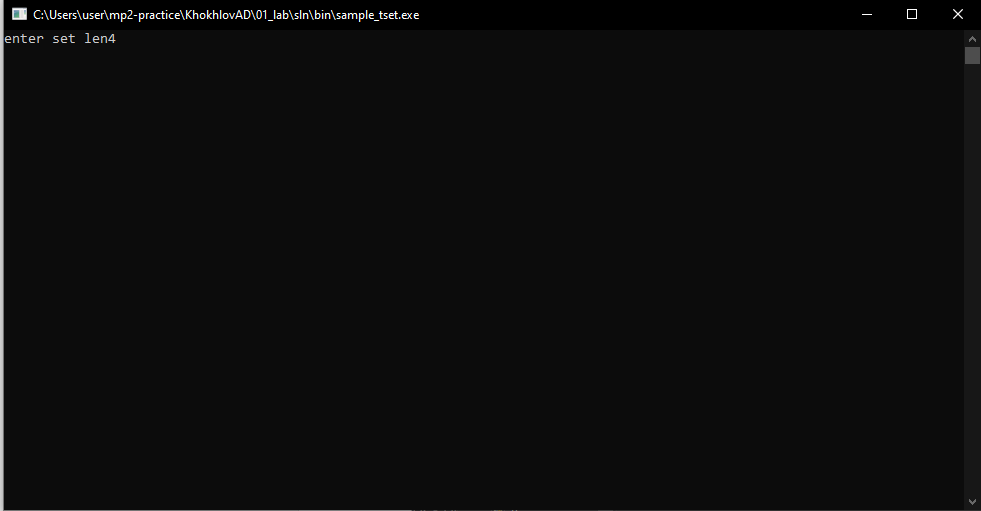
1. Основное окно программы
2. Это окно показывает работу основных функций работы с битовыми полями (сравнение на равенство, присваивание, длина, установка/очистка бита, сложение, пересечение, отрицание). Для продолжения введите значение битового поля длины 4. В результате будет выведено (рис. 2). Для выхода нажмите любую клавишу.



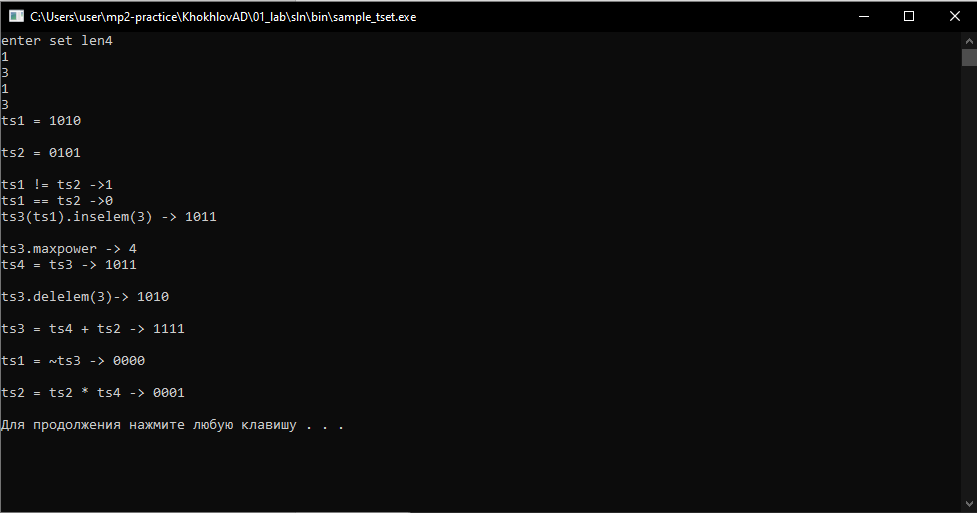
1. Основное окно программы

## Приложение для демонстрации работы множеств

1. Запустите приложение с названием sample\_tset.exe. В результате появится окно, показанное ниже (рис. 3).



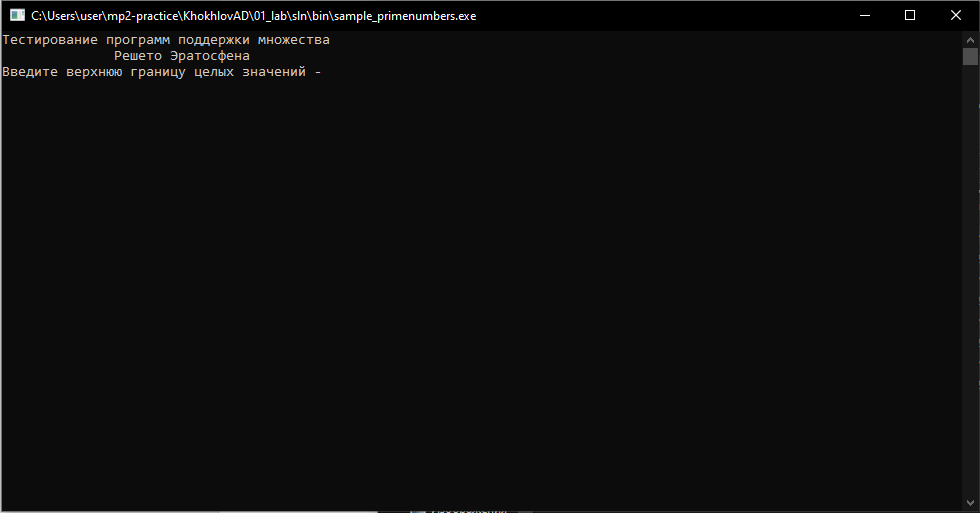
1. Основное окно программы
2. Введите значение множества длины 4. Будет выведено следуюшщее (рис. 4). Это окно показывает работу основных функций работы с множествами (сравнение на равенство, добавление элемента, длина, удаление элемена, сложение, пересечение, отрицание). Для завершения программы нажмите любую клавишу.



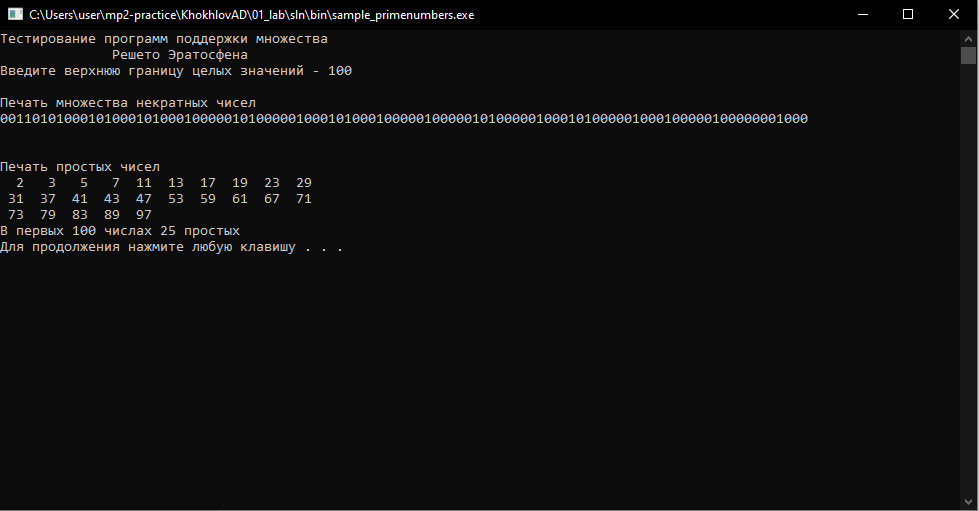
1. Основное окно программы

## «Решето Эратосфено»

1. Запустите приложение с названием sample\_tbitfield.exe. В результате появится окно, показанное ниже (рис. 5).



1. Основное окно программы
2. Введите верхнюю границу целых значений для поиска простых чисел. В результате выведено множество в битовом представлении. Ниже выведен список простых чисел (рис. 6). Для завершения программы нажмите любую клавишу.



1. Основное окно программы

# Руководство программиста

## Описание алгоритмов

### Битовые поля

1. Начало работы узазано в пункте [2.1](#_Приложение_для_демонстрации_1)
2. Описание методов и полей класса указано в пункте [3.2.1](#_Описание_класса_TBitField)

Битовые поля представляют собой набор чисел, каждый бит которых интепретируется элементом, равным индексом бита . Битовые поля обеспечивают удобный доступ к отдельным битам данных. Они позволяют формировать объекты с длинной, не крастной байту, что в свою очередь позволяет экономить память, более плотно размещая данные.

Битовое поле хранится в виде класса с полями: массив беззнаковых целых чисел, каждое из которых имеет размер 32 бита, максимамальный элемент(количество битов), количество беззнаковых целых чисел, которые образуют битовое поле.

Пример битового поля длинны 5: 01100

Битовое поле поддерживает операции объединения, пересечения, дополнение (отрицание), сравнения , ввода и вывода.

Операция объединения:

Операция возвращает экземпляр класса , каждый бит которого равен 1, если он есть хотя бы в 1 классе, которые объединяем, и 0 в противном случае.

Пример:

0 1 1 0 1 1

1 0 0 0 1 1

1 1 1 0 1 1

Операция пересечения:

Операция возвращает экземпляр класса , каждый бит которого равен 1, если он есть в каждом классе, и 0 в противном случае.

Пример:

0 1 1 0 1 1

1 0 0 0 1 1

0 0 0 0 1 1

Операция дополнения (отрицания):

Операция возвращает экземпляр класса , каждый бит которого равен 0, если он есть исходном классе, и 1 в противном случае.

Пример:

0 1 1 0 1 1

1 0 0 1 0 0

Операция ввода битового поля из консоли:

Операция позволяет ввести битовое поле из консоли. Для этого необходимо в консоль ввести 1 или 0 столько раз, какова длинна вводимого битового поля.

Пример:

0 0 0 0 0 0

После ввода:

0 1 1 0 1 1

Операция вывода битового поля в консоль:

Операция вывода позволяет вывести битовое поле в консоль

Также в битовом поле можно установить или отчистить бит

Пример:

0 1 1 0 1 1

Результат установление 1 бита:

1 1 1 0 1 1

Пример:

0 1 1 0 1 1

Результат удаления 2 бита:

0 0 1 0 1 1

Замечание: Введенный бит должен быть больше 0 и меньше длинны битового поля

Операции сравнения

Операция равенства выведет 1 , если два битовых поля равны, или каждые их биты совпадают, 0 в противном случае. Операция , обратная операции равенства, выведет 0, если хотя бы два бита совпадают, 1 в противном случае.

### Множества

1. Начало работы узазано в пункте [2.2](#_Приложение_для_демонстрации)
2. Описание методов и полей класса указано в пункте [3.2.2](#_Описание_класса_TSet)

Множества представляют собой набор цулых положительных чисел.В данной лаборатрной работе множество реализовано при помощи битового поля, соответственно каждый бит которых интепретируется элементом, равным индексом бита . Битовые поля обеспечивают удобный доступ к отдельным битам данных. Они позволяют формировать объекты с длинной, не крастной байту, что в свою очередь позволяет экономить память, более плотно размещая данные. Создание множества через битовые поля может сильно сократить использование памяти.

Множество поле хранится в виде класса с полями: Битовое поле, максимальный элемент множества.

Пример множества максимальной длинны 5: A: { 1, 2, 3, 5}

Множество поддерживает операции объединения, пересечения, дополнение (отрицание), сравнения , ввода и вывода.

Операция объединения с монжеством:

Операция возвращает экземпляр класса, содержащий все уникальные элементы из двух классов.

Пример:

A = {1, 2, 3}

B = {1,3,5}

Результат объединения множеств A|B:

A+B = {1, 2, 3, 5}

Операция объединения с множеством:

Операция возвращает экземпляр класса, содержащий все уникальные элементы, которые находятся в каждом классе.

Пример:

A = {1, 2, 3}

B = {1,3,5}

Результат объединения множеств A|B:

A\*B = {3}

Операции объединения и пересечения также работают с целым числом, они будут эквивалентны функциям добавления или удаления элемента в (из) множество (множества)

Операция дополнения (отрицания):

Операция возвращает экземпляр, класса содержащий элементы, которых нет в исходном классе, не большие максимальному элементу.

Пример:

A = {1, 2, 3}

Результат объединения множеств ~A:

~A = {4, 5}

Оп

Операция ввода множества из консоли:

Операция позволяет ввести множество из консоли. Для этого необходимо в консоль ввести сначала количество элементов в множестве, а дальше сами элементы, причём элементы не могут превышать максимального элемента множества.

Пример:

A = {}

После ввода:

A = {1, 2, 3}

Операция вывода битового поля в консоль:

Операция вывода позволяет вывести множество в консоль

Также в множестве можно добавить или убрать элемент.

Пример:

A = {1, 2, 3}

Результат добавления 4:

A = {1, 2, 3, 4}

Пример:

A = {1, 2, 3}

Результат удаления 2:

A = {1, 3}

Замечание: Введенное число должно быть больше 0 и меньше максимального элемента множества

### «Решето Эратосфена»

1. Начало работы указано в пункте [2.3](Решето_Эратосфено#_)

Данный алгоритм реализован двумя способами, при помощи классов TSet.

Сначала необходимо ввести целое положительное число, до которого будет . После чего алгоритм заполняет все элементы классов равными 1.

После чего алгоритм , начинает перебирать все числа от 2 до N.

1. Если это число есть в нашем множестве, то мы переходим к шагу 2, иначе к шагу 3.

2. Это число, и дальше все кратные ему числа удаляются из нашего множества.

3. Выбирается следующее число. Если это число больше N, то алгоритм заканчивается, иначе – переход к шагу 1.

После чего выводятся все числа от 1 до N с идентификатором (0, если оно не простое, 1 если простое), затем выводятся все простые числа и их количество.

## Описание программной реализации

### Описание класса TBitField

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;

TELEM GetMemMask (const int n) const;

public:

TBitField(int len);

TBitField(const TBitField &bf);

void SetBit(const int n);

void ClrBit(const int n);

int GetBit(const int n) const

bool operator==(const TBitField &bf) const;

bool operator!=(const TBitField &bf) const;

const TBitField& operator=(const TBitField &bf);

TBitField operator|(const TBitField &bf);

TBitField operator&(const TBitField &bf);

TBitField operator~(void);

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

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

};

Назначение: представление битового поля.

Поля:

BitLen – длина битового поля – максимальное количество битов.

pMem – память для представления битового поля.

MemLen – количество элементов для представления битового поля.

Методы:

int GetMemIndex(const int n) const;

Назначение: получение индекса элемента в памяти.

Входные параметры:

n - номер бита.

Выходные параметры:

Номер элемента в памяти.

TELEM GetMemMask (const int n) const;

Назначение: получение маски по индексу.

Входные параметры:

n - номер бита.

Выходные параметры:

Маска.

TBitField(int len);

Назначение: конструктор с параметром.

Входные параметры:

len – длина поля.

TBitField(const TBitField &bf);

Назначение: конструктор копирования.

Входные параметры:

bf – ссылка на копируемое поле.

void SetBit(const int n);

Назначение: установка бита.

Входные параметры:

n – индекс бита.

Выходные парамытры:

void.

void ClrBit(const int n);

Назначение: очистка бита.

Входные параметры:

n – индекс бита.

Выходные парамытры:

void.

int GetBit(const int n) const

Назначение: получение значения бита.

Входные параметры:

n – индекс бита.

Выходные параметры:

Значение бита.

bool operator==(const TBitField &bf) const

Назначение: перегрузка операции сравнения на равенство.

Входные параметры:

bf – ссылка на константное битовое поле

Выходные параметры:

Результат сравнения

bool operator!=(const TBitField &bf) const;

Назначение: перегрузка операции сравнения на неравенство.

Входные параметры:

bf – ссылка на константное битовое поле

Выходные параметры:

Результат сравнения

const TBitField& operator=(const TBitField &bf);

Назначение: перегрузка операции присваивания.

Входные параметры:

bf – ссылка на константное битовое поле

Выходные параметры:

Ссылка на битовое поле.

TBitField operator|(const TBitField &bf);

Назначение: перегрузка операции “или”.

Входные параметры:

bf – ссылка на константное битовое поле

Выходные параметры:

Битовое поле.

TBitField operator&(const TBitField &bf);

Назначение: перегрузка операции “и”.

Входные параметры:

bf – ссылка на константное битовое поле

Выходные параметры:

Битовое поле.

TBitField operator~(void);

Назначение: перегрузка операции отрицания.

Выходные параметры:

Битовое поле.

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

Назначение: перегрузка потокового ввода.

Выходные параметры:

istr – ссылка на поток ввода.

bf – ссылка на константное битовое поле

Выходные параметры:

Ссылка на поток вводы

ostream &operator<<(ostream &ostr, const TBitField &bf);

Назначение: перегрузка потокового вывода.

Входные параметры:

ostr – ссылка на поток вывода.

bf – ссылка на константное битовое поле

Выходные параметры:

Ссылка на поток вывода.

### Описание класса TSet

class TSet

{

private:

int MaxPower;

TBitField BitField;

public:

TSet(int mp);

TSet(const TSet &s);

TSet(const TBitField &bf);

operator TBitField();

int GetMaxPower(void) const;

void InsElem(const int Elem);

void DelElem(const int Elem);

int IsMember(const int Elem) const;

bool operator== (const TSet &s) const;

bool operator!= (const TSet &s) const;

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

TSet operator+ (const int Elem);

TSet operator- (const int Elem);

TSet operator+ (const TSet &s);

TSet operator\* (const TSet &s);

TSet operator~ (void);

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

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

};

Назначение: представление множества.

Поля:

MaxPower – длина множества.

BitField – битовое поле.

Методы:

TSet(int mp);

Назначение: конструктор с параметром.

Входные параметры:

mp – длина множества.

TSet(const TSet &s);

Назначение: конструктор копирования.

Входные параметры:

s – ссылка на константное множество.

TSet(const TBitField &bf);

Назначение: конструктор преобразования типа.

Входные параметры:

bf – ссылка на константное битовое поле

operator TBitField();

Назначение: преобразование типа к питовому полю.

int GetMaxPower(void) const;

Назначение: получение максимальной мощности множества.

Выходные параметры:

Максимальная мощность множества.

void InsElem(const int Elem);

Назначение: включение элемента в множество.

Входные параметры:

Elem – новый элемент множества.

void DelElem(const int Elem);

Назначение: удаление элемента из множества.

Входные параметры:

Elem – элемент множества.

int IsMember(const int Elem) const;

Назначение: проверка наличия элемента в множестве.

Входные параметры:

Elem – элемент множества.

Выходные параметры:

Результат проверки

bool operator== (const TSet &s) const;

Назначение: перегрузка операции сравнения на равенство.

Входные элементы:

s – ссылка на константное множество.

Выходные параметры:

Результат сравнения.

bool operator!= (const TSet &s) const;

Назначение: перегрузка операции сравнения на неравенство.

Входные элементы:

s – ссылка на константное множество.

Выходные параметры:

Результат сравнения.

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

Назначение: перегрузка операции присваивания.

Входные параметры:

s – ссылка на константное множество.

Выходные параметры:

Ссылка на множество.

TSet operator+ (const int Elem);

Назначение: перегрузка операции объединения с элементом.

Входные параметры:

Elem – новый элемент множества.

Выходные параметры:

Множество.

TSet operator- (const int Elem);

Назначение: перегрузка операции разницы с элементом.

Входные параметры:

Elem – элемент множества.

Выходные параметры:

Множество.

TSet operator+ (const TSet &s);

Назначение: перегрузка операции объединения множеств.

Входные параметры:

s – ссылка на константное множество.

Выходные параметры:

Множество.

TSet operator\* (const TSet &s);

Назначение: перегрузка операции пересечения множеств.

Входные параметры:

s – ссылка на константное множество.

Выходные параметры:

Множество.

TSet operator~ (void);

Назначение: перегрузка операции дополнения.

Выходные параметры:

Множество.

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

Назначение: перегрузка операции потокового ввода.

Входные параметры:

istr – ссылка на поток ввода.

bf – ссылка на константное множество.

Выходные параметры:

Ссылка на поток ввода.

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

Назначение: перегрузка операции потокового вывода.

Входные параметры:

ostr – ссылка на поток вывода.

bf – ссылка на константное множество.

Выходные параметры:

Ссылка на поток вывода.

# Заключение

В ходе выполнения работы "Битовые поля и множества" были изучены и практически применены концепции битовых полей и множеств.

Были достигнуты следующие результаты:

1. Были изучены теоретические основы битовых полей и множеств.

2. Была разработана программа, реализующая операции над битовыми полями и множествами. В ходе экспериментов была оценена эффективность работы этих операций и сравнена с другими подходами. Результаты показали, что использование битовых полей и множеств позволяет существенно сократить объем памяти и ускорить операции над множествами.

3. Были проанализированы полученные результаты и сделаны выводы о преимуществах и ограничениях использования битовых полей и множеств. Оказалось, что эти структуры данных особенно полезны при работе с большими объемами данных, где компактность представления и эффективность операций являются ключевыми факторами.

# Литература

1. . Сысоев А.В., Алгоритмы и структуры данных, лекция 03, 19 сентября.

# Приложения

## Приложение А. Реализация класса TBitField

#define BITS\_IN\_ONE\_MEM (sizeof(TELEM) \* 8)

TBitField::TBitField(int len)

{

if (len < 0) throw "Negative len";

BitLen = len;

MemLen = (BitLen-1) / BITS\_IN\_ONE\_MEM + 1;

pMem = new TELEM[MemLen];

for (int i = 0; i < MemLen; i++) pMem[i] = 0;

}

TBitField::TBitField(const TBitField& bf)

{

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

{

return (n / BITS\_IN\_ONE\_MEM);

}

TELEM TBitField::GetMemMask(const int n) const

{

if ((n > BitLen) || (n < 0)) throw "Negative n";

return (1 << (n % BITS\_IN\_ONE\_MEM));

}

int TBitField::GetLength(void) const

{

return BitLen;

}

void TBitField::SetBit(const int n)

{

if ((n >= BitLen) || (n < 0)) throw "Negative n";

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

}

void TBitField::ClrBit(const int n)

{

if ((n > BitLen) || (n < 0)) throw "Negative n";

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

}

int TBitField::GetBit(const int n) const

{

if ((n > BitLen) || (n < 0)) throw "Negative n";

if (pMem[GetMemIndex(n)] & GetMemMask(n)) return 1; else return 0;

}

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

{

delete[] pMem;

this->BitLen = bf.BitLen;

MemLen = bf.MemLen;

pMem = new TELEM[MemLen];

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

return \*this;

}

bool TBitField::operator==(const TBitField& bf) const

{

if (BitLen != bf.BitLen)

return 0;

int k = 0;

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

if (pMem[i] != bf.pMem[i]) {

return 0;

}

return 1;

}

bool TBitField::operator!=(const TBitField& bf) const

{

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

}

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

{

int k;

int z;

int y;

if (BitLen > bf.BitLen)

{

y = 0; // This is big

k = BitLen;

z = bf.BitLen;

}

else {

y = 1; //bf is big

k = bf.BitLen;

z = BitLen;

}

TBitField a(k);

for (int i = 0; i <= GetMemIndex(z); i++) a.pMem[i] = bf.pMem[i] | pMem[i];

for (int i = (GetMemIndex(z) + 1); i < a.MemLen; i++) if (y == 1) a.pMem[i] = bf.pMem[i]; else a.pMem[i] = pMem[i];

return a;

}

TBitField TBitField::operator&(const TBitField& bf)

{

int k;

int z;

if (BitLen > bf.BitLen) {

k = BitLen;

z = bf.BitLen;

}

else

{

k = bf.BitLen;

z = BitLen;

}

TBitField a(k);

for (int i = 0; i <= GetMemIndex(k); i++) a.pMem[i] = pMem[i] & bf.pMem[i];

return a;

}

TBitField TBitField::operator~(void)

{

TBitField a(\*this);

for (int i = 0; i < (a.MemLen - 1); i++) a.pMem[i] = ~(a.pMem[i]);

for (int i = ((a.MemLen - 1) \* BITS\_IN\_ONE\_MEM); i < (a.BitLen); i++) {

if (a.GetBit(i) == 1) a.ClrBit(i);

else a.SetBit(i);

}

return a;

}

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

{

int tmp;

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

istr >> tmp;

if ((tmp != 0) && (tmp != 1)) {

throw "The bit cannot take such a value";

}

if (tmp == 0) {

bf.ClrBit(i);

}

else {

bf.SetBit(i);

}

}

return istr;}

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

{

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

if (bf.GetBit(i) == 0)

ostr << 0;

else ostr << 1;

ostr << endl;

return ostr;

}

## Приложение Б. Реализация класса TSet

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

{

MaxPower = mp;

}

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

{

MaxPower = s.GetMaxPower();

}

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

{

MaxPower = bf.GetLength();

}

TSet::operator TBitField()

{

return BitField;

}

int TSet::GetMaxPower(void) const

{

return MaxPower;

}

int TSet::IsMember(const int Elem) const

{

if ((Elem < MaxPower) & (Elem >= 0))

if (BitField.GetBit(Elem) == 1) return 1;

else return 0; else throw "Negative Elem";

}

void TSet::InsElem(const int Elem)

{

if ((Elem < MaxPower) & (Elem >= 0))

BitField.SetBit(Elem); else throw "Negative n";

}

void TSet::DelElem(const int Elem)

{

if ((Elem < MaxPower) & (Elem >= 0))

BitField.ClrBit(Elem); else throw "Negative n";

}

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

{

MaxPower = s.MaxPower;

BitField = s.BitField;

return \*this;

}

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

{

return (BitField == s.BitField);

}

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

{

return !(\*this == s);

}

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

{

TSet a(BitField | s.BitField);

return a;

}

TSet TSet::operator+(const int Elem)

{

TSet a(\*this);

if ((Elem < MaxPower) & (Elem >= 0)) {

a.InsElem(Elem);

}

else throw "Nrgative Elem";

return a;

}

TSet TSet::operator-(const int Elem)

{

TSet a(\*this);

if ((Elem < MaxPower) & (Elem >= 0)) {

a.DelElem(Elem);

}

else throw "Nrgative Elem";

return a;

}

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

{

TSet a(BitField & s.BitField);

return a;

}

TSet TSet::operator~(void)

{

TSet a(~BitField);

return a;

}

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

{

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

s.DelElem(i);

}

cout << "Input your set (To finish, enter -1)" << endl;

int i;

while (1) {

istr >> i;

if (i == -1) {

return istr;

}

if ((i < 0) || (i > s.MaxPower)) {

throw "OUTOFRANGE";

}

s.InsElem(i);

}

return istr;}

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

{

cout << s.BitField;

return ostr;

}