

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
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
_____ / Чистов А.Д. /
Подпись

Проверил: к.т.н., доцент каф. ВВиСП
_____ / Кустикова В.Д. /
Подпись

Нижний Новгород
2023

Содержание

Введение.....	3
1 Постановка задачи.....	4
2 Руководство пользователя.....	5
2.1 Приложение для демонстрации работы битовых полей.....	5
2.2 Приложение для демонстрации работы множеств.....	6
2.3 «Решето Эратосфено»	8
3 Руководство программиста	9
3.1 Описание алгоритмов	9
3.1.1 Битовые поля	9
3.1.2 Множества	9
3.1.3 «Решето Эратосфена»	9
3.2 Описание классов.....	10
3.2.1 Описание класса TBitField	10
3.2.2 Описание класса TSet	14
Заключение	18
Литература	19
Приложения	20
Приложение Б. Реализация класса TBitField.....	20
Приложение Б. Реализация класса TSet.....	24

Введение

Битовое поле — это структура данных, состоящая из одного или нескольких соседних битов, выделенных для определенных целей, так что любой отдельный бит или группа битов в структуре может быть установлен или проверен. 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. Битовые поля часто используются для управления флагами или настроек. Например, в программировании можно использовать битовые поля для представления различных состояний объекта или опций функции.

Теория множеств — учение об общих свойствах множеств — преимущественно бесконечных. Явным образом понятие множества подверглось систематическому изучению во второй половине XIX века в работах немецкого математика Георга Кантора. Активное применение аппарата теории множеств в современной науке приводит к необходимости создания соответствующих программных решений.

Битовые поля и множества могут быть использованы для оптимизации некоторых алгоритмов. Например, вы можете использовать битовые поля для представления булевого массива, что позволяет эффективно выполнять операции, такие как поиск, вставка и удаление элементов.

Важным преимуществом использования битовых операций является тот факт, что позволяют выполнять различные манипуляции с битами, такие как установка, сброс или инвертирование конкретных битов. Это может быть полезно, например, при работе с масками или фильтрами.

Множества поддерживают базовые операции, такие как объединение, пересечение и разность, что может быть очень удобным при работе с данными. Например, можно объединить два множества, чтобы получить новое множество, содержащее все элементы из обоих исходных множеств. В целом можно сказать, что битовые поля и множества имеют широкое применение в различных областях, особенно в программировании и компьютерных системах. Они позволяют эффективно использовать память и упрощают работу с большим количеством данных.

1 Постановка задачи

Цель – Изучить битовые поля и множества. Получить навык практического применения данных структур данных.

Задачи:

1. Изучить основные понятия и принципы работы с битовыми полями и множествами
2. Разработать программу, которая будет реализовывать операции над битовыми полями и множествами
3. Протестировать корректность выполнения программы на различных примерах
4. Применение полученных результатов для алгоритма «решето Эратосфена»
5. Сделать выводы о проделанной работе

2 Руководство пользователя

2.1 Приложение для демонстрации работы битовых полей

1. Запустить sample_bitfield.exe. В результате появится следующее окно (рис 1):

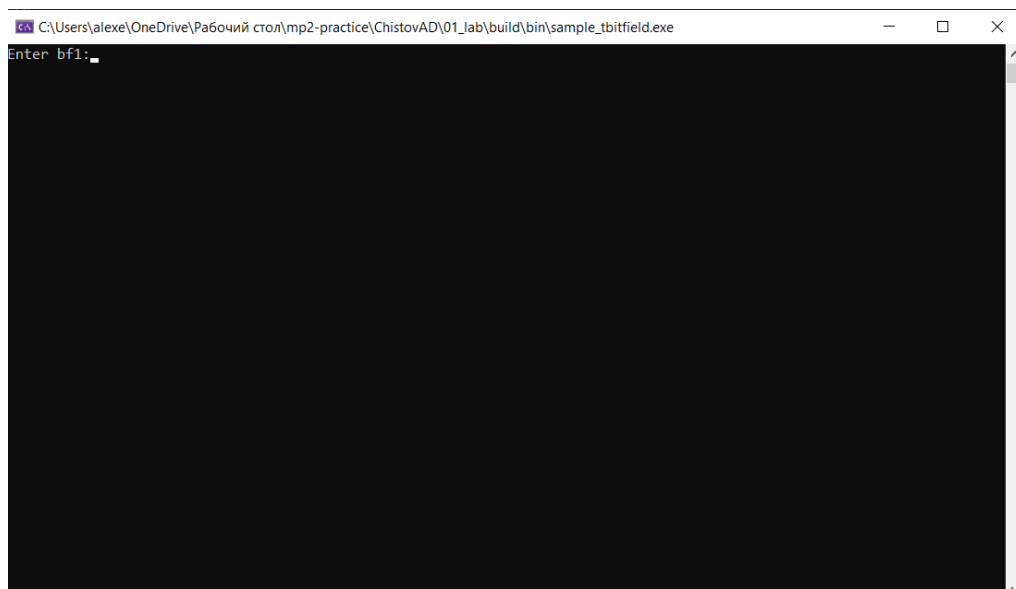


Рис.1. Основное окно приложения битовых полей

2. Далее вам необходимо ввести 2 битовое поле длины 5 (рис 2):

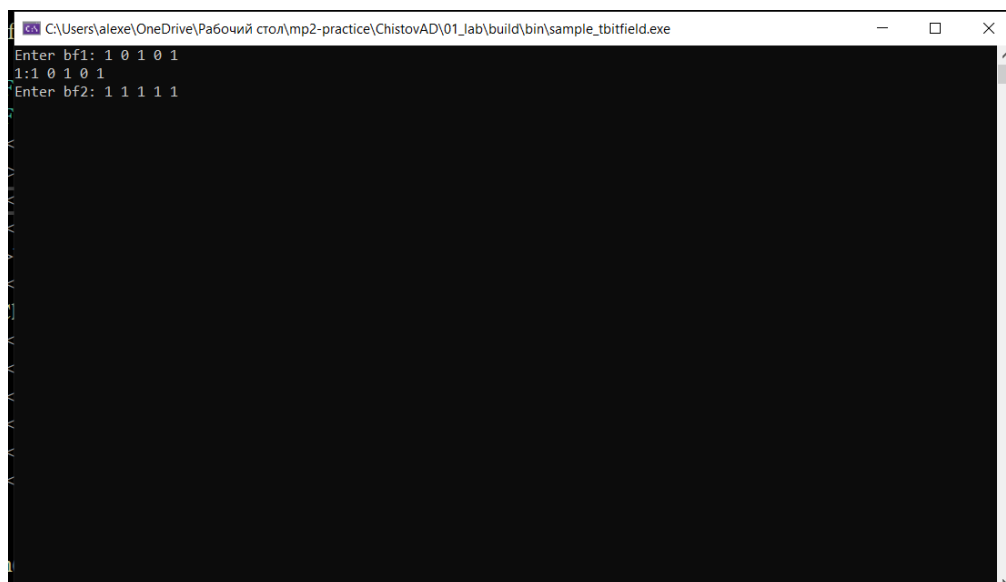
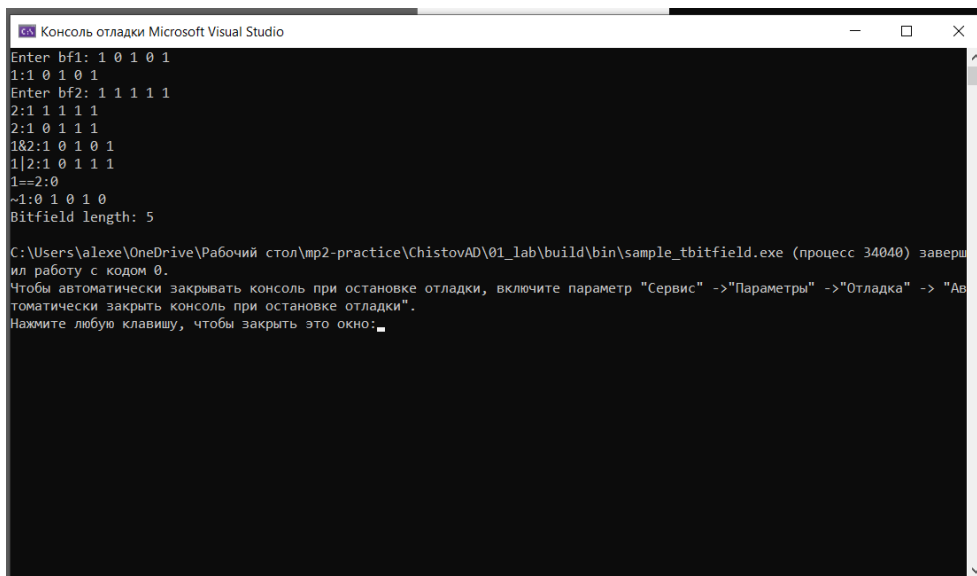


Рис.2. Ввод битовых полей

3. После вы получите результат работы программы с введенными битовыми полями(рис 3):



```
Консоль отладки Microsoft Visual Studio
Enter bf1: 1 0 1 0 1
1:1 0 1 0 1
Enter bf2: 1 1 1 1
2:1 1 1 1
2:1 0 1 1
1&2:1 0 1 0 1
1|2:1 0 1 1 1
1==2:0
~1:0 1 0 1 0
Bitfield length: 5

C:\Users\alexe\OneDrive\Рабочий стол\mp2-practice\ChistovAD\01_lab\build\bin\sample_tbitfield.exe (процесс 34040) завершил работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" -> "Параметры" -> "Отладка" -> "Автоматически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно: █
```

Рис.3.Результат работы программы

2.2 Приложение для демонстрации работы множеств

1. Запустить sample_bitfield.exe.В результате появится следующее окно (рис 4):

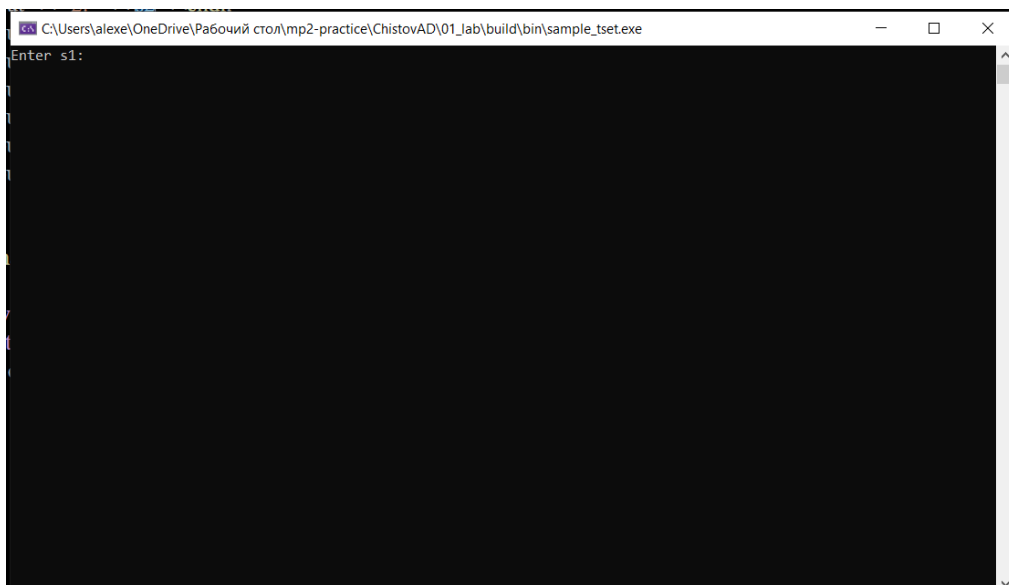
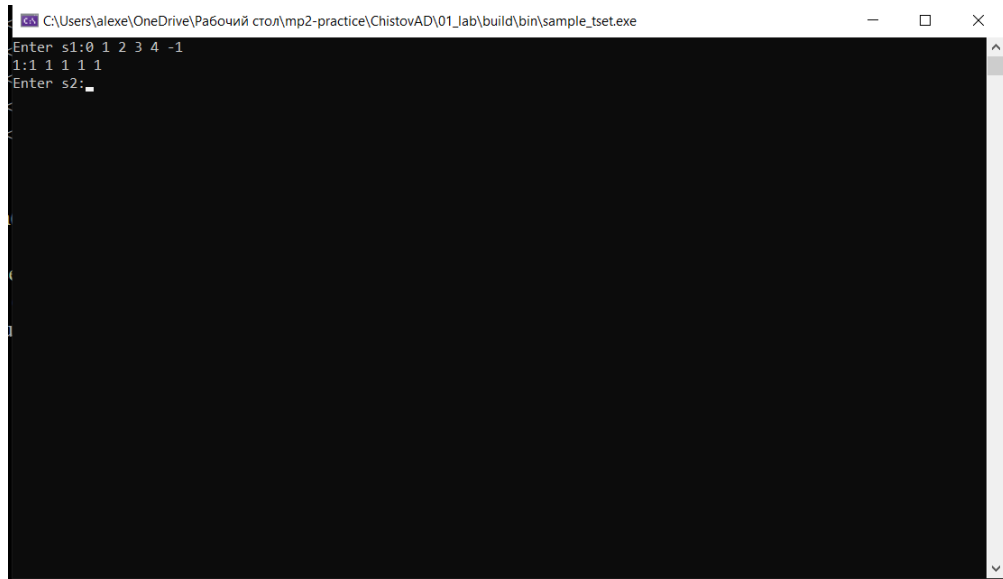


Рис 4. Основное окно работы множеств

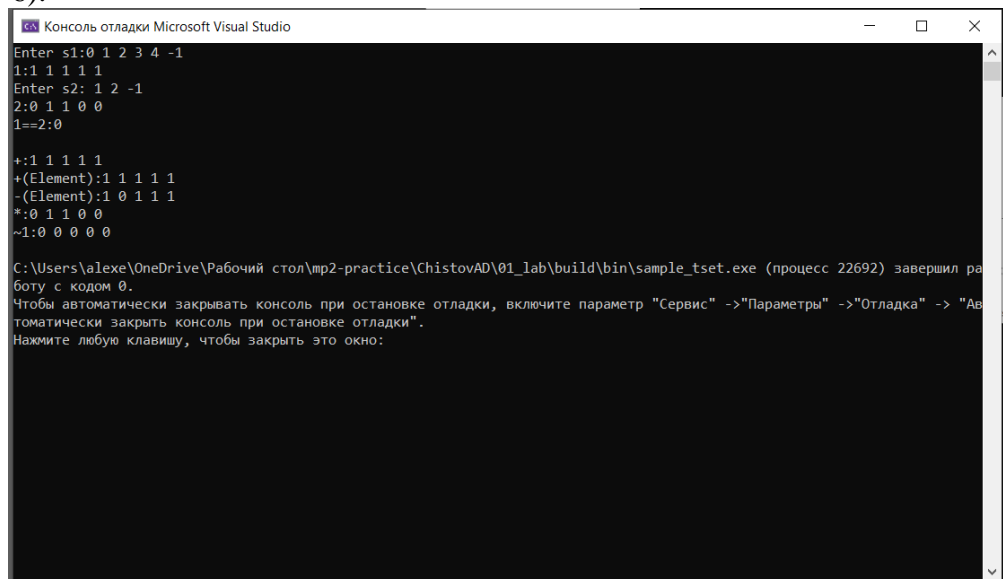
2. . Далее вам необходимо ввести 2 множества длины 5 (рис 5):



```
C:\Users\alexe\OneDrive\Рабочий стол\mp2-practice\ChistovAD\01_lab\build\bin\sample_tset.exe
Enter s1: 0 1 2 3 4 -1
1: 1 1 1 1 1
Enter s2:
2: 0 1 1 0 0
```

Рис.5. Ввод множеств

3. После вы получите результат работы программы с введенными множествами(рис 6):



```
Консоль отладки Microsoft Visual Studio
Enter s1: 0 1 2 3 4 -1
1: 1 1 1 1 1
Enter s2: 1 2 -1
2: 0 1 1 0 0
1==2: 0

+1: 1 1 1 1 1
+(Element): 1 1 1 1 1
-(Element): 1 0 1 1 1
*: 0 1 1 0 0
~1: 0 0 0 0 0

C:\Users\alexe\OneDrive\Рабочий стол\mp2-practice\ChistovAD\01_lab\build\bin\sample_tset.exe (процесс 22692) завершил работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" -> "Параметры" -> "Отладка" -> "Автоматически закрывать консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно:
```

Рис.6. Результат работы программы

2.3 Приложение «решето Эратосфена»

1. Запустите sample_primenumbers.exe. В результате появится следующее окно(рис 7):

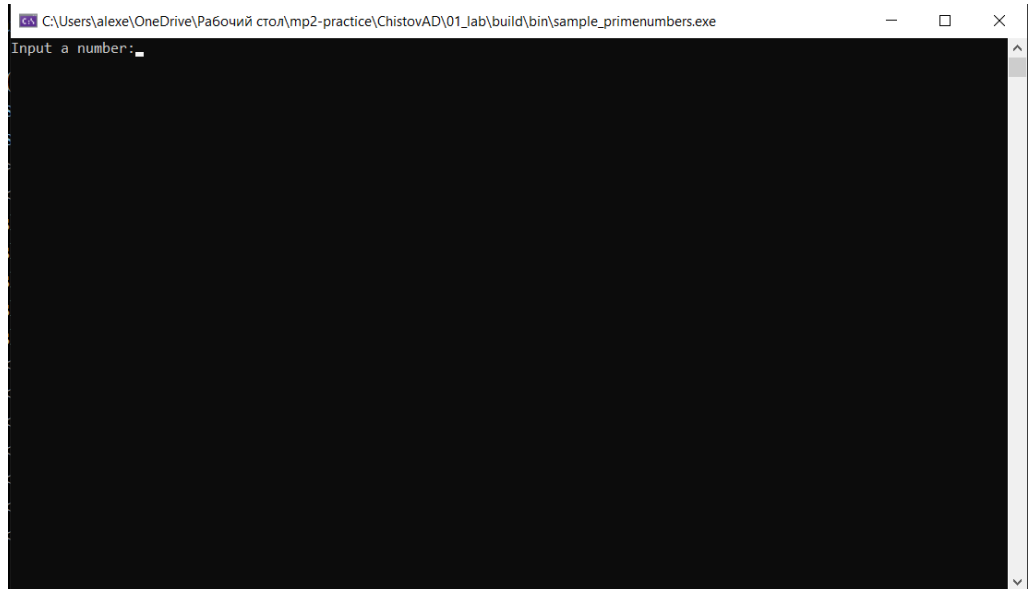


Рис.7. Основное окно приложения

2. Далее необходимо ввести целое положительное число для того, чтобы получить все простые числа до введенного (рис 8)

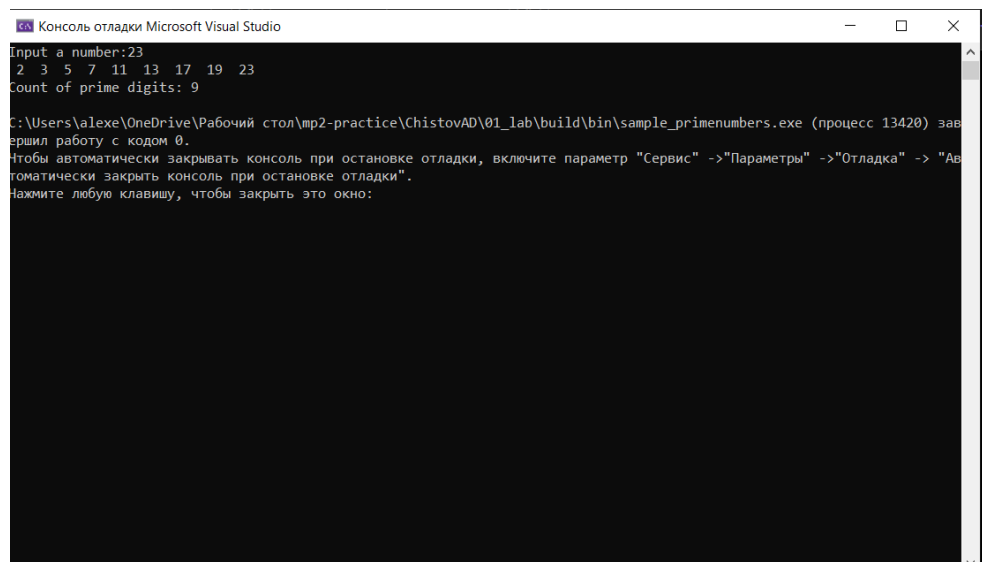


Рис.8.Результат работы приложения решето Эратосфена

3 Руководство программиста

3.1 Используемые алгоритмы

3.1.1 Битовые поля

Класс TbitField содержит в себе длину битового поля, память и количество элементов, содержащемуся в нем, где каждый элемент может быть равен 0 или 1. Ввиду последнего реализуются операции работы с ними (получение битовой маски, поставить элемент равным 0 и 1 и т.д.), сравнение на равенство и ввод, вывод и другие. Сама реализация операций представлена в конце лабораторной работы.

3.1.2 Множества

Класс множества (Tset) основан на классе битовых полей (TbitField). На нем основаны теоретико-множественные операции (объединение, пересечение и т.д.), получение максимальной длины, а также добавление и удаление элементов, сравнение на равенство и ввод, вывод. Tset содержит в себя битовое поле, а также максимальную длину множества. Реализация операций представлена в конце лабораторной работы.

3.1.3 Алгоритм «решето Эратосфена»

Решето Эратосфена — алгоритм нахождения всех простых чисел до некоторого целого числа n , который приписывают древнегреческому математику Эратосфену Киренскому. Как и во многих случаях, здесь название алгоритма говорит о принципе его работы, то есть решето подразумевает фильтрацию, в данном случае фильтрацию всех чисел за исключением простых. По мере прохождения списка нужные числа остаются, а ненужные (они называются составными) исключаются.

Алгоритм выполнения состоит в следующем:

- 1) У пользователя запрашивается целое положительное число
- 2) Заполнение множества
- 3) Проверка до квадратного корня и удаление кратных членов (данный шаг повторяется несколько раз пока остаются кратные числа)
- 4) Полученные элементы будут простыми числами

3.2 Описание классов

3.2.1 Класс TbitField

Объявление класса:

```
class TBitField
{
private:
    int BitLen;
    TELEM *pMem;
    int MemLen;

    int GetMemIndex(const int n) const;
    TELEM GetMemMask (const int n) const;
public:
    TBitField(int len);
    TBitField(const TBitField &bf);
    ~TBitField();

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

    // битовые операции
    int operator==(const TBitField &bf) const; // сравнение
    int operator!=(const TBitField &bf) const; // сравнение
    TBitField& operator=(const TBitField &bf); // присваивание
    TBitField operator|(const TBitField &bf); // операция "или"
    TBitField operator&(const TBitField &bf); // операция "и"
    TBitField operator~(void);             // отрицание

    friend istream &operator>>(istream &istr, TBitField &bf);
    friend ostream &operator<<(ostream &ostr, const TBitField &bf);
};
```

Поля:

BitLen – длина битового поля.

pMem – память для представления битового поля

MemLen – количество элементов Мем для представления битового поля

Конструкторы:

- **TBitField(int len);**

Назначение: конструктор с параметром, выделение памяти

Входные параметры:

len – длина битового поля

Выходные параметры:

Отсутствуют

- **TBitField(const TBitField &bf);**

Назначение: конструктор копирования. Создание экземпляра класса на основе другого экземпляра

Входные параметры:

&bf – ссылка, адрес экземпляра класса, на основе которого будет создан другой.

Выходные параметры:

Отсутствуют

Деструктор:

~TBitField() ;

Назначение: деструктор. Отчистка выделенной памяти

Входные и выходные параметры отсутствуют

Методы:

- **int GetMemIndex(const int n) const;**

Назначение: получение индекса элемента, где хранится бит.

Входные данные:

n – номер бита.

Выходные данные:

Индекс элемента, где хранится бит с номером **n**.

- **TELEM GetMemMask(const n) const;**

Назначение: получение битовой маски

Входные данные:

n – номер бита.

Выходные данные:

Элемент под номером **n**

- **int GetLength(void) const;**

Назначение: получение длины битового поля

Входные параметры отсутствуют

Выходные параметры: длина битового поля

- **void SetBit(const int n)**

Назначение: установить бит=1

Входные параметры:

n - номер бита, который нужно установить

Выходные параметры отсутствуют

- **void ClrBit(const int n);**

Назначение: отчистить бит (установить бит = 0)

Входные параметры:

n - номер бита, который нужно отчистить

Выходные параметры отсутствуют

- **int GetBit(const int n) const;**

Назначение: вывести бит (узнать бит)

Входные параметры:

n - номер бита, который нужно вывести (узнать)

Выходные параметры: бит (1 или 0, в зависимости есть установлен он, или нет)

Операторы:

- **int operator== (const TBitField &bf) const;**

Назначение: оператор сравнения. Сравнить на равенство 2 битовых поля

Входные параметры:

&bf – битовое поле, с которым мы сравниваем

Выходные параметры: 1 или 0, в зависимости равны они, или нет соответственно

- **int operator!= (const TBitField &bf) const;**

Назначение: оператор сравнения. Сравнить на равенство 2 битовых поля

Входные параметры:

&bf – битовое поле, с которым мы сравниваем

Выходные параметры: 1 или 0, в зависимости равны они, или нет соответственно

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

Назначение: оператор присваивания. Присвоить экземпляру *this экземпляр &bf

Входные параметры:

&bf – битовое поле, которое мы присваиваем

Выходные параметры: ссылка на экземпляр класса **TBitField**, *this

- **TBitField operator| (const TBitField &bf) ;**

Назначение: оператор побитового «ИЛИ»

Входные параметры:

&bf – битовое поле

Выходные параметры: Экземпляр класса , который равен { *this | bf }

- **TBitField operator&(const TBitField &bf) ;**

Назначение: оператор побитового «И»

Входные параметры:

&bf – битовое поле, с которым мы сравниваем

Выходные параметры: Экземпляр класса, который равен { *this & bf }

- **TBitFields operator~(void) ;**

Назначение: оператор инверсии

Входные параметры отсутствуют

Выходные параметры: Экземпляр класса, каждый элемент которого равен {~*this }

- **friend istream &operator>>(istream &istr, TBitFields &bf) ;**

Назначение: оператор ввода из консоли

Входные параметры:

&istr – буфер консоли

&bf – класс, который нужно ввести из консоли

Выходные параметры: Ссылка на буфер (поток) &istr

- **friend ostream &operator<<(ostream &ostr, const TBitFields &bf) ;**

Назначение: оператор вывода из консоли

Входные параметры:

&istr – буфер консоли

&bf – класс, который нужно вывести в консоль

Выходные параметры: Ссылка на буфер (поток) &istr

3.2.2 Класс TSet

Объявление класса:

```
class TSet
{
private:
    int MaxPower; // максимальная мощность множества
    TBitField BitField; // битовое поле для хранения хар-го вектора
public:
    TSet(int mp);
    TSet(const TSet &s); // 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 n); // включить элемент в множество
    void DelElem(const int n); // удалить элемент из множества
    int IsMember(const int n) const; // проверить наличие элемента в
    // множестве
    // теоретико-множественные операции
    int operator== (const TSet &s); // сравнение
    TSet& operator=(const TSet &s); // присваивание
    TSet operator+ (const int n); // включение элемента в множество
    TSet operator- (const int n); // удаление элемента из множества
    TSet operator+ (const TSet &s); // объединение
    TSet operator* (const TSet &s); // пересечение
    TSet operator~ (void); // дополнение
    friend istream &operator>>(istream &istr, TSet &bf);
    friend ostream &operator<<(ostream &ostr, const TSet &bf);
```

Назначение: представление множества чисел.

Поля:

MaxPower — максимальный элемент множества.

BitField — экземпляр битового поля, на котором реализуется множество.

Конструкторы:

- **TSet(int mp);**

Назначение: конструктор с параметром, выделение памяти

Входные параметры:

mp — максимальный элемент множества.

Выходные параметры: Отсутствуют

- **TSet(const TSet &s);**

Назначение: конструктор копирования. Создание экземпляра класса на основе другого экземпляра

Входные параметры:

&s – ссылка, адрес экземпляра класса, на основе которого будет создан другой.

Выходные параметры: Отсутствуют

Деструктор:

~TSet () ;

Назначение: деструктор. Отчистка выделенной памяти

Входные и выходные параметры отсутствуют

Методы:

- **int GetMaxPower(void) const;**

Назначение: получение максимального элемента множества

Входные параметры отсутствуют

Выходные параметры: максимальный элемент множества

- **void InsElem(const int Elem)**

Назначение: добавить элемент в множество

Входные параметры: Elem - добавляемый элемент

Выходные параметры отсутствуют

- **void DelElem(const int Elem)**

Назначение: удалить элемент из множества

Входные параметры: Elem - удаляемый элемент

Выходные параметры отсутствуют

- **int IsMember(const int Elem) const;**

Назначение: узнать, есть ли элемент в множестве

Входные параметры:

Elem - элемент, который нужно проверить на наличие

Выходные параметры: 1 или 0, в зависимости есть элемент в множестве, или нет

Операторы:

- **int operator==(const TSet &s) const;**

Назначение: оператор сравнения. Сравнить на равенство 2 множества

Входные параметры: &s – битовое поле, с которым мы сравниваем

Выходные параметры: 1 или 0, в зависимости равны они, или нет соответственно

- **int operator!=(const TSet &s) const;**

Назначение: оператор сравнения. Сравнить на равенство 2 множества

Входные параметры: &s – битовое поле, с которым мы сравниваем

Выходные параметры: 0 или 1, в зависимости равны они, или нет соответственно

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

Назначение: оператор присваивания. Присвоить экземпляру *this экземпляр &s

Входные параметры:

&s – множество, которое мы присваиваем

Выходные параметры: ссылка на экземпляр класса **TSet**, *this

- **TSet operator+(const TSet &bf);**

Назначение: оператор объединения множеств

Входные параметры: &s - множество;

Выходные параметры: Экземпляр класса, который равен { *this | s }

- **TSet operator*(const TSet &bf);**

Назначение: оператор пересечения множеств

Входные параметры: &s - множество;

Выходные параметры: Экземпляр класса, который равен { *this & s }

- **TBitField operator~(void);**

Назначение: оператор дополнение до Универса

Входные параметры: отсутствуют

Выходные параметры: Экземпляр класса, каждый элемент которого равен

{ ~*this }, т.е. если i элемент исходного экземпляра будет равен будет находится в множестве, то на выходе его не будет, и наоборот

- **friend istream &operator>>(istream &istr, TSet &s);**

Назначение: оператор ввода из консоли

Входные параметры:

&istr – буфер консоли

&s – класс, который нужно ввести из консоли

Выходные параметры: Ссылка на буфер (поток) &istr

- **friend ostream &operator<<(ostream &ostr, const TSet &s);**

Назначение: оператор вывода из консоли

Входные параметры:

&istr – буфер консоли

&s – класс, который нужно вывести в консоль

Выходные параметры: Ссылка на буфер (поток) &istr

- **operator TBitField();**

Назначение: вывод поля **BitField**

Входные параметры отсутствуют

Выходные данные: поле **BitField**

- **TSet operator+(const int Elem);**

Назначение: оператор объединения множества и элемента. Данный оператор аналогичен метод добавления элемента в множество

Входные параметры:

Elem - число

Выходные параметры: исходный экземпляр класса, содержащий Elem

- **TSet operator- (const int Elem);**

Назначение: оператор объединения множества и элемента. Данный оператор аналогичен методу удаления элемента из множества.

Входные параметры: Elem - число

Выходные параметры: исходный экземпляр класса, не содержащий Elem

Заключение

В результате данной лабораторной работы были изучены теоретические основы битовых полей и множеств, а также принципы их использования в программировании. На основе полученных знаний была разработана программа, которая реализует операции над битовыми полями и множествами. Были проведены эксперименты с различными наборами данных, чтобы проверить работу программы и изучить ее производительность. Проведенный анализ результатов показал, что использование битовых полей и множеств может быть очень полезным в решении определенных задач. Они позволяют эффективно работать с большим количеством данных, а также выполнять операции объединения, пересечения и разности между наборами элементов. В целом, лабораторная работа помогла понять основные принципы работы с битовыми полями и множествами, их преимущества и ограничения.

Литературы

1. <https://metanit.com/c/tutorial/6.7.php>
2. https://en.wikipedia.org/wiki/Bit_field
3. <https://informatics.msk.ru/mod/book/view.php?id=578&chapterid=306>
4. <https://learn.microsoft.com/ru-ru/cpp/cpp/cpp-bit-fields?view=msvc-170>

Приложения

Приложение А. Реализация класса TBitField

```
TBitField::TBitField(int len)
{
    if (len > 0) {
        BitLen = len;
        MemLen = ((len + bitsInElem - 1) >> shiftSize);
        pMem = new TELEM[MemLen];
        memset(pMem, 0, MemLen * sizeof(TELEM));
    }
    else if (len == 0)
    {
        BitLen = 0;
        MemLen = 0;
        pMem = nullptr;
    }
    else {
        throw ("Bitfield size less than 0");
    }
}

TBitField::TBitField(const TBitField& bf)
{
    BitLen = bf.BitLen;
    MemLen = bf.MemLen;
    if (MemLen) {
        pMem = new TELEM[MemLen];
        memcpy(pMem, bf.pMem, MemLen * sizeof(TELEM));
    }
    else {
        pMem = nullptr;
    }
}

TBitField::~TBitField()
{
    delete[] pMem;
}

int TBitField::GetMemIndex(const int n) const
```

```

{
    return n >> shiftSize;
}

TELEM TBitField::GetMemMask(const int n) const
{
    return 1 << (n & (bitsInElem - 1));
}

int TBitField::GetLength(void) const
{
    return BitLen;
}

void TBitField::SetBit(const int n)
{
    if (n >= BitLen || n < 0)
        throw("bit position is out of range");
    pMem[GetMemIndex(n)] |= GetMemMask(n);
}

void TBitField::ClrBit(const int n)
{
    if (n >= BitLen || n < 0)
        throw("bit position is out of range");
    pMem[GetMemIndex(n)] &= ~GetMemMask(n);
}

int TBitField::GetBit(const int n) const {
    if (n >= BitLen || n < 0)
        throw("bit position is out of range");
    return (pMem[GetMemIndex(n)] & GetMemMask(n)) ? 1 : 0;
}

TBitField & TBitField::operator=(const TBitField & bf)
{
    if (*this == bf) return *this;
    if (BitLen != bf.BitLen)
    {
        delete[] pMem;
        BitLen = bf.BitLen;
        MemLen = bf.MemLen;
    }
}

```

```

        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
{
    if (BitLen != bf.BitLen) return 0;
    for (int i = 0; i < MemLen; i++) {
        if (pMem[i] != bf.pMem[i]) {
            return 0;
        }
    }
    return 1;
}

int TBitField::operator!=(const TBitField& bf) const
{
    return !((*this) == bf);
}

TBitField TBitField::operator|(const TBitField& bf)
{
    int len = max(BitLen, bf.BitLen);
    TBitField tmp(len);
    for (int i = 0; i < tmp.MemLen; i++)
        tmp.pMem[i] = pMem[i] | bf.pMem[i];
    return tmp;
}

TBitField TBitField::operator&(const TBitField& bf)
{
    int len = max(BitLen, bf.BitLen);
    TBitField tmp(len);
    for (int i = 0; i < tmp.MemLen; i++) {
        tmp.pMem[i] = pMem[i] & bf.pMem[i];
    }
}

```

```

        return tmp;
    }

TBitField TBitField::operator~(void)
{
    TBitField tmp(BitLen);
    for (int i = 0; i < BitLen; i++)
        if (GetBit(i)==0)
            tmp.SetBit(i);
    return tmp;
}

istream& operator>>(istream& istr, TBitField& bf) {
    int answer;
    for (int i = 0; i < bf.BitLen; i++) {
        istr >> answer;
        if (answer == 0) {
            bf.ClrBit(i);
        }
        else if (answer == 1) {
            bf.SetBit(i);
        }
        else{
            throw ("The entered element does not match the bit field");
            break;
        }
    }
    return istr;
}

ostream& operator<<(ostream& ostr, const TBitField& bf)
{
    for (int i = 0; i < bf.GetLength(); ++i)
    {
        ostr << bf.GetBit(i) << " ";
    }
    return ostr;
}

```

Приложение Б. Реализация класса TSet

```
TSet::TSet(int mp) :MaxPower(mp), BitField(mp) {}

TSet::TSet(const TSet& s) :
BitField(s.BitField),MaxPower(s.GetMaxPower()) {}

TSet::TSet(const TBitField& bf) :MaxPower(bf.GetLength()), BitField(bf)
{}

int TSet::GetMaxPower(void) const { return MaxPower;}

TSet::operator TBitField(){ return BitField;}

int TSet::IsMember(const int Elem) const
{
    if (Elem >= MaxPower || Elem < 0) throw ("Elemet is out of
universe");
    return BitField.GetBit(Elem);
}

void TSet::InsElem(const int Elem)
{
    if (Elem >= MaxPower || Elem < 0) throw ("Elemet is out of
universe");
    return BitField.SetBit(Elem);
}

void TSet::DelElem(const int Elem)
{
    if (Elem >= MaxPower || Elem < 0) throw ("Elemet is out of
universe");
    return BitField.ClrBit(Elem);
}

TSet& TSet::operator=(const TSet& s)
{
    if (*this == s) return *this;
    MaxPower = s.MaxPower;
    BitField = s.BitField;
    return *this;
}

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



```

{
    if (MaxPower != s.GetMaxPower())
    { return 0;}
    return (BitField == s.BitField);
}

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

TSet TSet::operator+(const TSet& s)
{
    TSet tmp(max(MaxPower, s.GetMaxPower()));
    tmp.BitField = BitField | s.BitField;
    return tmp;
}

TSet TSet::operator+(const int Elem)
{
    if (Elem >= MaxPower || Elem < 0) throw ("Element is out of
universe");
    TSet tmp(*this);
    tmp.InsElem(Elem);
    return tmp;
}

TSet TSet::operator-(const int Elem)
{
    if (Elem >= MaxPower || Elem < 0) throw ("Element is out of
universe");
    TSet tmp(*this);
    tmp.DelElem(Elem);
    return tmp;
}

TSet TSet::operator*(const TSet& s)
{
    TSet tmp(max(MaxPower, s.GetMaxPower()));
    tmp.BitField = BitField & s.BitField;
    return tmp;
}

```

```

TSet TSet::operator~(void)
{
    TSet tmp(MaxPower);
    tmp.BitField = ~BitField;
    return tmp;
}

//

istream& operator>>(istream& istr, TSet& ts) {
    int inputElem;
    while (true) {
        istr >> inputElem;
        if (inputElem == -1) {
            break;
        }
        if (inputElem >= 0 && inputElem < ts.MaxPower) {
            ts.InsElem(inputElem);
        }
        else {
            cerr << "Invalid input. Element out of range." << endl;
        }
    }
    return istr;
}

ostream& operator<<(ostream& ostr, const TSet& s)
{
    const int x = s.MaxPower-1;
    for (int i = 0; i <= x; ++i)
    {
        ostr << s.IsMember(i) << " ";
    }
    return ostr;
}

```