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
_____ / Лысов И.М. /
Подпись

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

Нижний Новгород
2023

Оглавление

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 [Error! Reference source not found.]**Error! Reference source not found.**.

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 1.

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.

Множества поддерживают базовые операции, такие как объединение, пересечение и разность. 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 Цели и задачи

Цель: изучить битовые поля и множества. Получить навык практического применения данных структур данных.

Задачи:

1. Изучить основные 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 и тестах.
4. Применить полученные знания и реализовать алгоритм «решето Эратосфена» отбора простых элементов из заданного множества элементов.
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

На этом шаге необходимо ввести количество элементов в битовом поле (n). Далее это число будет использоваться для второго поля как (n+1) элементов в битовом поле.

2. После ввода числа элементов во множестве программа требует ввести соответствующие битовые значения для первого поля (рис. 2).

Рис. 2. Ввод значений первого битового поля

3. После ввода значений первого битового поля, программа выводит значение этого поля. Далее программа просит ввести значения второго битового поля, причем для (n+1) элементов (рис. 3).

Рис. 3. Вывод значений первого поля и ввод значений второго битового поля

4. После нажатия клавиши Enter программа выводит результаты операций работы с битовыми полями. Используются следующие операции (“или”, “и”, проверка равны ли два поля, отрицание первого битового поля) (рис. 4).

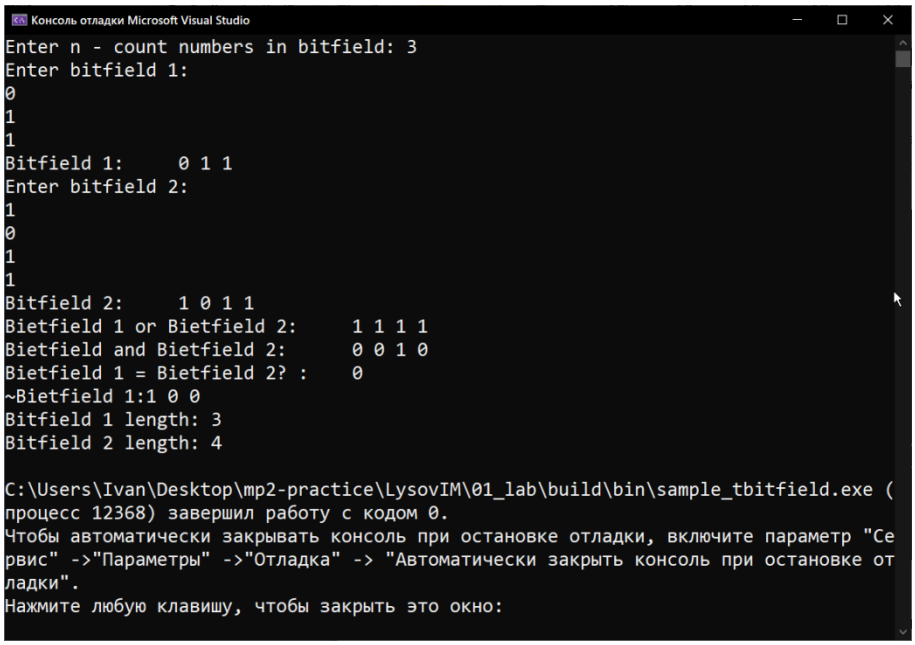

Рис. 4. Результат работы программы

2.2 Приложение для демонстрации работы множеств

1. Запустить sample_tset.exe. В результате появится следующее окно (рис. 5).

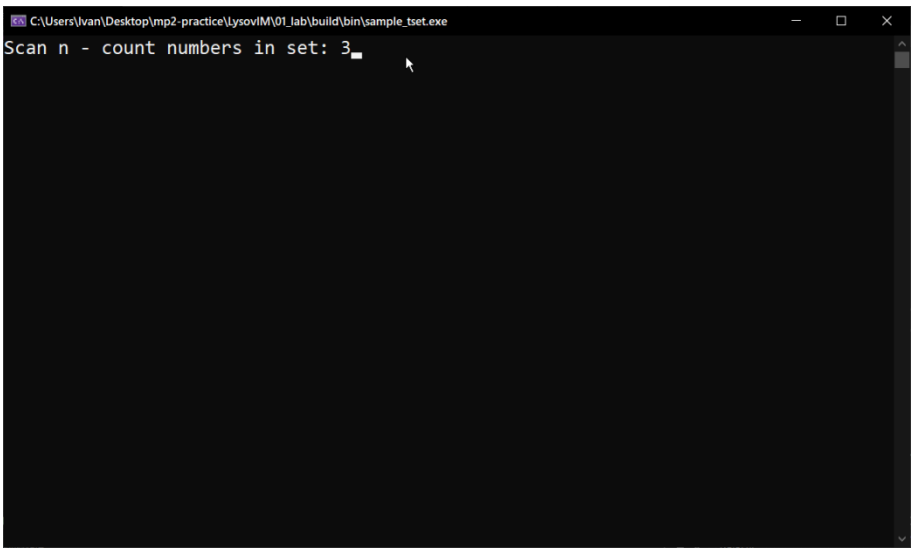

Рис. 5. Основное меню приложения для работы с множествами

На этом шаге необходимо ввести количество элементов в битовом поле (n). Далее это число будет использоваться для второго поля как (n+1) элементов в битовом поле.

2. После ввода числа элементов во множестве программа требует ввести соответствующие значения для первого множества (рис. 6).

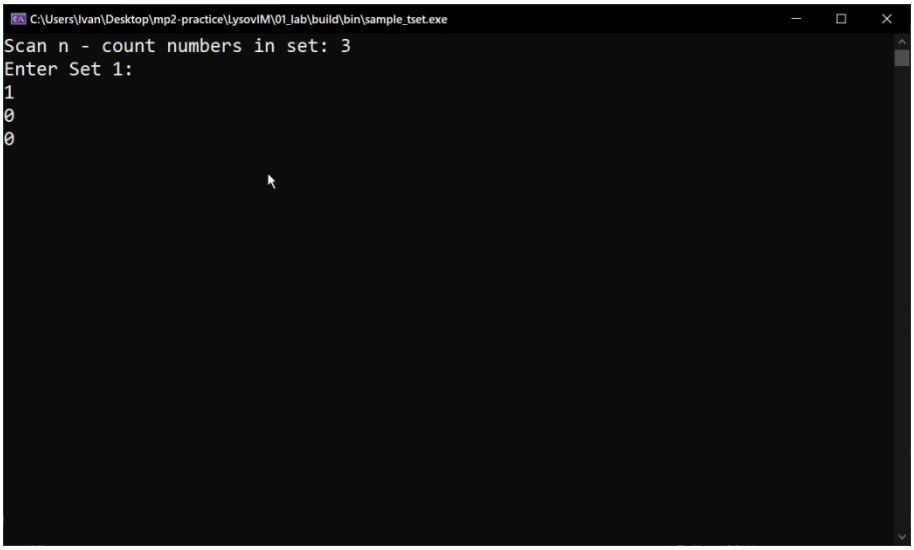

Рис. 6. Ввод значений первого множества

3. После ввода значений первого множества, программа выводит значение этого множества, с помощью бит (0 – нет элемента во множестве, 1 – есть элемент в 9 множестве). Далее программа просит ввести значения второго множества, причем для (n+1) элементов (рис. 7).

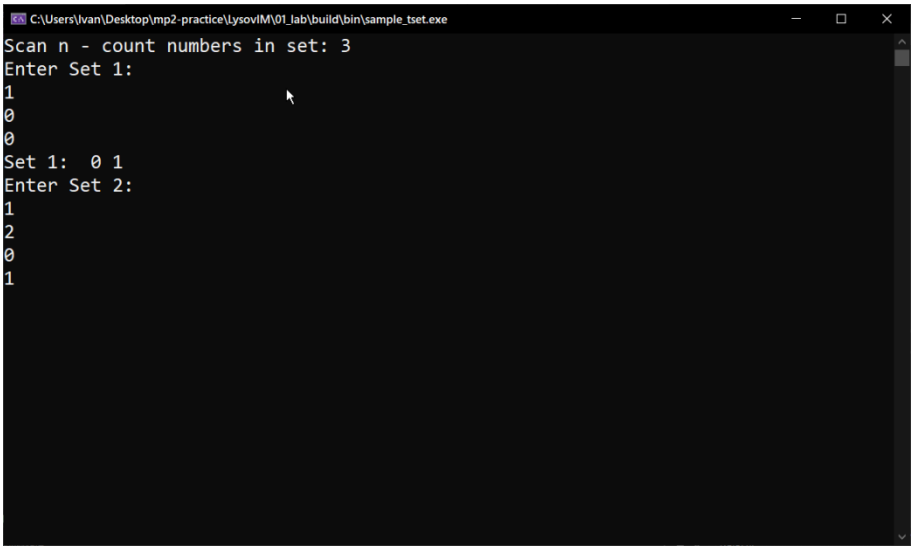

Рис. 7. Вывод значений первого множества и ввод значений второго множества

4. После нажатия клавиши Enter программа выводит результаты операций работы с битовыми полями. Используются следующие операции (проверка на равенства двух множеств, множество 1 || множество 2, множество 1 + элемент (0), множество 1 – элемент (1), множество 1 & множество 2, отрицание первого множества) (рис. 8).

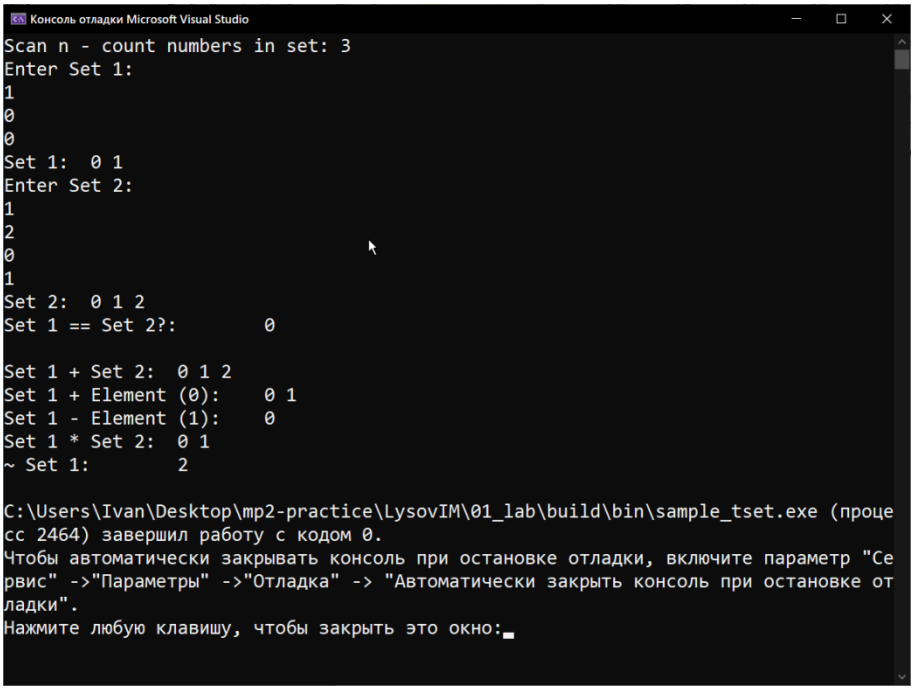

Рис. 8. Результат работы программы

2.3 Приложение «решето Эратосфена»

1. Запустите sample_primenumbers.exe. В результате появится следующее окно (рис. 9).

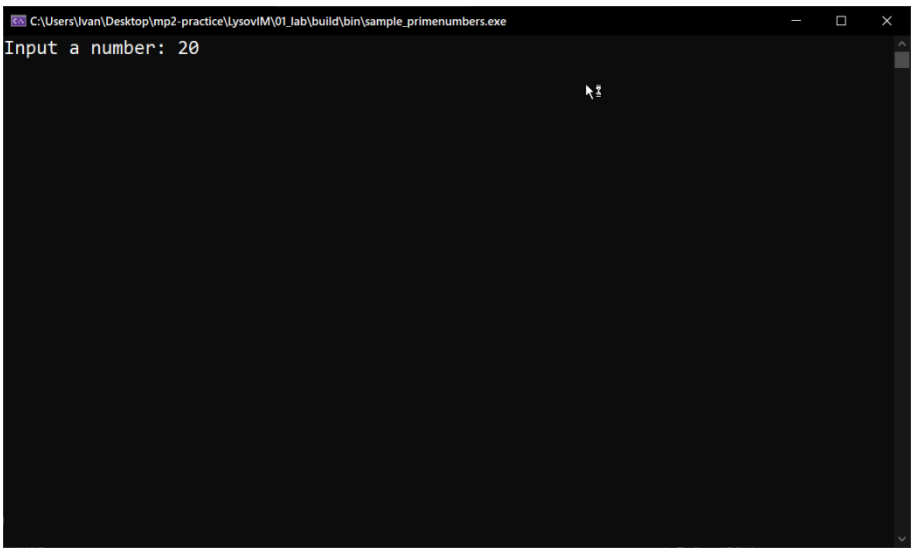

Рис. 9. Основное окно приложения

2. Далее необходимо ввести целое положительное число для того, чтобы получить все простые числа до введенного числа (рис. 10).

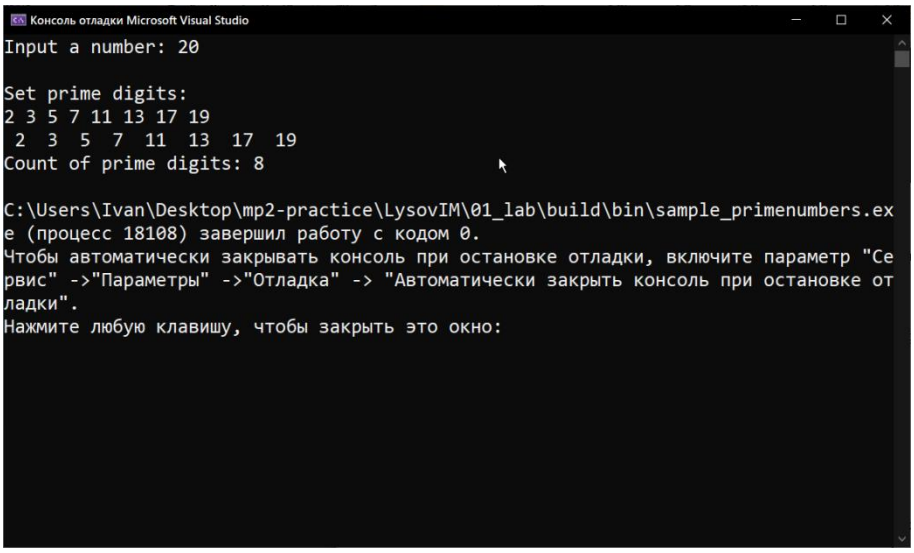

Рис. 10. Результат работы алгоритма «решето Эратосфена»

3 Руководство программиста

3.1 Использованные алгоритмы

3.1.1 Битовое поле

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.

С битовыми полями можно реализовать операции: взятие маски, получение значения бита, очистка бита, установка бит 1.

На основе битовых полей можно реализовать следующие побитовые операции: побитовое «ИЛИ», побитовое «И», побитовое «НЕ».

- **Операция побитовое «ИЛИ»**

Входные данные: битовые поля.

Выходные данные: битовое поле, после применении операции (если хотя бы в одном поле есть 1, то после применения операции в результате будет 1).

A	1	0	0
B	0	1	0
A|B	1	1	0

- **Операция побитовое «И»**

Входные данные: битовые поля.

Выходные данные: битовое поле, после применении операции (если хотя бы в одном поле есть 0, то после применения операции в результате будет 0).

A	1	0	0
B	1	1	0
A&B	1	0	0

- **Операция побитовое «НЕ»**

Входные данные: битовое поле.

Выходные данные: битовое поле, после применении операции будет инвертированное поле.

A	1	0	0
~A	0	1	1

3.1.2 Множества

Множества – набор исходных элементов. Но в данной работе множества основаны на основе битовых полях. В работе основаны теоретико-множественные операции

(объединение, пересечение, отрицание и т.д.), получение максимальной длины, а также добавление и удаление элементов, сравнение на равенство и ввод, вывод.

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

- **Операция объединения множеств**

Входные данные: 2 множества.

Выходные данные: множество, равное объединению множеств, содержащее все элементы из двух множеств.

A	0	1	1
B	0	2	1
A&B	0	1	-

- **Операция пересечения множеств**

После применения операции получится множество, содержащее все элементы, которые имеются в обоих множествах.

Входные данные: 2 множества.

Выходные данные: множество, равное пересечению множеств, содержащее все элементы, которые содержатся в обоих множествах множеств.

A	0	1	1
B	0	2	1
A+B	0	1	2

- **Операция дополнения множества**

Входные данные: множество.

Выходные данные: множество, равное дополнению исходного множества.

A	0	1	1
U	0	1	2
~A	0	1	2

- **Операция сравнения множеств**

Входные данные: 2 множества.

Выходные данные: 0, если множества не равны (равны) и равны (не равны).

A	**1**	**1**	**2**
B	**0**	**1**	**2**
A==B	**0**		
A!=B	**1**		

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 (n)

2) Заполнение множества от 1 до n.

3) Проверка до квадратного корня ($\sqrt{n}$) и удаление кратных членов.

4) Полученные элементы будут простыми числами.

3.2 Описание классов

3.2.1 Класс TBitField

Объявление 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; // индекс в pMem 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
  const 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)
};
```

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 экземпляр класса на основе имеющегося экземпляра.

Входные параметры: **bf** – ссылка, адрес экземпляра класса, на основе которого будет создан другой.

Выходные параметры: отсутствуют.

Деструктор:

- **~TBitField();**

Назначение: очистка 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 значение бита 1.

Входные параметры: **n** – номер бита, который нужно установить.

Выходные параметры: отсутствуют.

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

Назначение: установить значение бита 0.

Входные параметры: **n** – номер бита, который нужно отчистить.

Выходные параметры: отсутствуют.

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

Назначение: вывести значение бита.

Входные параметры: **n** – номер бита, который нужно вывести (узнать).

Выходные параметры: значение бита.

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, если равны то 1, 0 если нет.

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

Назначение: оператор сравнения. Сравнить на неравенство 2 битовых поля.

Входные параметры: **bf** – битовое поле, с которым мы сравниваем.

Выходные параметры: 1 или 0, в зависимости неравны 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 }.**

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

Назначение: оператор инверсии.

Входные параметры**:** отсутствуют.

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

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

Назначение: оператор ввода из консоли.

Входные параметры: **istr** – буфер консоли и **bf -** класс, который нужно ввести из консоли.

Выходные параметры: ссылка на буфер (поток) **istr.**

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

Назначение: оператор вывода из консоли.

Входные параметры: **istr** – буфер консоли и **bf** - класс, который нужно вывести в консоль.

Выходные параметры: ссылка на буфер (поток) **istr**.

3.2.2 Класс TSet

```
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
  const 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);
};
```

Поля:

MaxPower – максимальный элемент множества.

BitField – экземпляр битового поля, на котором реализуется множество.

Конструкторы:

- **TSet(int mp);**

Назначение: конструктор с параметром и 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: очистка выделенной памяти.

Входные и выходные параметры**:** отсутствуют.

Методы:

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

Назначение: получить максимальный элемент множества.

Входные параметры: отсутствуют.

Выходные параметры: максимальный элемент множества.

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

Назначение: добавить элемент во 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: проверить наличие элемента в множестве.

Входные параметры: **`Elem`** – элемент, который нужно проверить на наличие.

Выходные параметры: 1 или 0, в зависимости от наличия элемента в множестве.

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.

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

Назначение: оператор сравнения. Сравнить на неравенство 2 множества.

Входные параметры: **`s`** – битовое поле, с которым мы сравниваем.

Выходные параметры**:** 1 или 0, в зависимости неравны они или нет.

- **`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 Универсального множества.

Входные параметры: отсутствуют.

Выходные параметры: экземпляр класса, каждый элемент которого равен **`{~*this}`**, т.е. если i элемент исходного экземпляра будет равен будет находиться во 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 и **`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 и **`&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 во 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 метод удаления элемента из множество.

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. 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.

Литература

1. Бытовые поля [https://metanit.com/c/tutorial/6.7.php?ysclid=lozbqtqq6160734321].
2. Битовые множества (bitset) [https://studfile.net/preview/1872018].

Приложение

Реализация класса TSet:

```
#include "tset.h"

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

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

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

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

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

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


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

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

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

istream& operator>>(istream& istr, TSet& s)
{
    int elem;
    for (int i = 0; i < s.MaxPower; i++) {
        istr >> elem;
        if (elem < 0)
            break;
        s.InsElem(elem);
    }
    return istr;
}

ostream& operator<<(ostream& ostr, const TSet& s)
{
    for (int i = 0; i < s.MaxPower; i++)
    {
        if (s.IsMember(i))
            ostr << i << " ";

    }
    return ostr;
}
```

Реализация класса TBitField:

```
#include "tbitfield.h"
const int bitsInElem = 32;
const int shiftSize = 5;//Значение для сдвига

TBitField::TBitField(int Len)//Конструктор по умолчанию
{
    if (Len > 0) {
        BitLen = Len;// длина битового поля - макс. к-во битов
        MemLen = ((Len + bitsInElem - 1) >> shiftSize); // к-во эл-тов Мем для
представления бит.поля
        pMem = new TELEM[MemLen];//Массив для предоставления битового поля
        memset(pMem, 0, MemLen * sizeof(TELEM));//Функция копирования одного объекта в
другой заданное количество раз
    }
    else if (Len < 0)
    {
        throw ("Field less than 0");
    }
    else if (Len == 0) {
        MemLen = 0;// к-во эл-тов Мем для представления бит.поля
        BitLen = 0;// длина битового поля - макс. к-во битов
        pMem = nullptr;//Значение пустого указателя
    }
}

TBitField::TBitField(const TBitField& bf) //Конструктор копирования
{
    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("The bit is not in the range");
    pMem[GetMemIndex(n)] |= GetMemMask(n);
}

void TBitField::ClrBit(const int n)
{
    if (n >= BitLen || n < 0)
```

```
        throw("The bit is not in the range");
    pMem[GetMemIndex(n)] &= ~GetMemMask(n);
}

int TBitField::GetBit(const int n) const {
    if (n >= BitLen || n < 0)
        throw("The bit is not in the range");
    return (pMem[GetMemIndex(n)] & GetMemMask(n)) ? 1 : 0;
}

const 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
{
    return !((*this) == bf);
}

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;
}

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

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

TBitField TBitField::operator~()
{
    TBitField obj(BitLen);
    for (int i = 0; i < BitLen; i++)
        if (GetBit(i) == 0)
```

```
            obj.SetBit(i);
    return obj;
}

istream& operator>>(istream& istr, TBitField& bf) {
    int ans;
    for (int i = 0; i < bf.BitLen; i++) {
        istr >> ans;
        if (ans == 0) {
            bf.ClrBit(i);
        }
        else if (ans == 1) {
            bf.SetBit(i);
        }
        else {
            throw ("The Element doesn't match the bitfield");
            break;
        }
    }
    return istr;
}

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