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 3822Б1ФИ2
_____ / Холин К.И
Подпись

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

Нижний Новгород
2023

Содержание

Введение

В C++ иногда возникают такие ситуации,когда информацию об объекте достаточно хранить в формате состояний(статусов),представляющих из себя 0 и 1. На этом основывается проект Множества,который использует интерфейс битовых полей для реализации работы с теоретико-множественными операциями. Это самый оптимальный вариант,поскольку он даёт нам возможность использовать не всю предоставляемую типом данных память,а только его часть. Обращение к определённому биту позволяет нам узнать его состояние для выполнения конкретной задачи. Например,чтобы проверить элемент на принадлежность множеству в нашем случае. Битовые поля в этом случае играют важную роль.

1 Постановка задачи

Цель – реализовать классы: TSet и TBitField

Задачи:

1. Класс для работы с множествами должен поддерживать эффективное хранение данных.
2. Написать следующие операции для работы с битовыми полями: установить бит в 1,установить бит в 0,получить значение бита,сравнить два битовых поля,сложить и инвертировать,вывести битовое поле требуемого формата и ввести битовое поле.
3. Добавить вспомогательные операции получения бита,маски бита,длины битового поля.
4. Написать следующие операции для работы с множествами: вставка элемента,удаление,проверка наличия,сравнение множеств,объединение множеств,пересечение,разность,копирование,вычисление мощности множества,вывод элементов множества требуемого формата и ввод.
5. Добавить вспомогательные операции для получения мощности множества.

2 Руководство пользователя

2.1 Приложение для демонстрации работы битовых полей

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

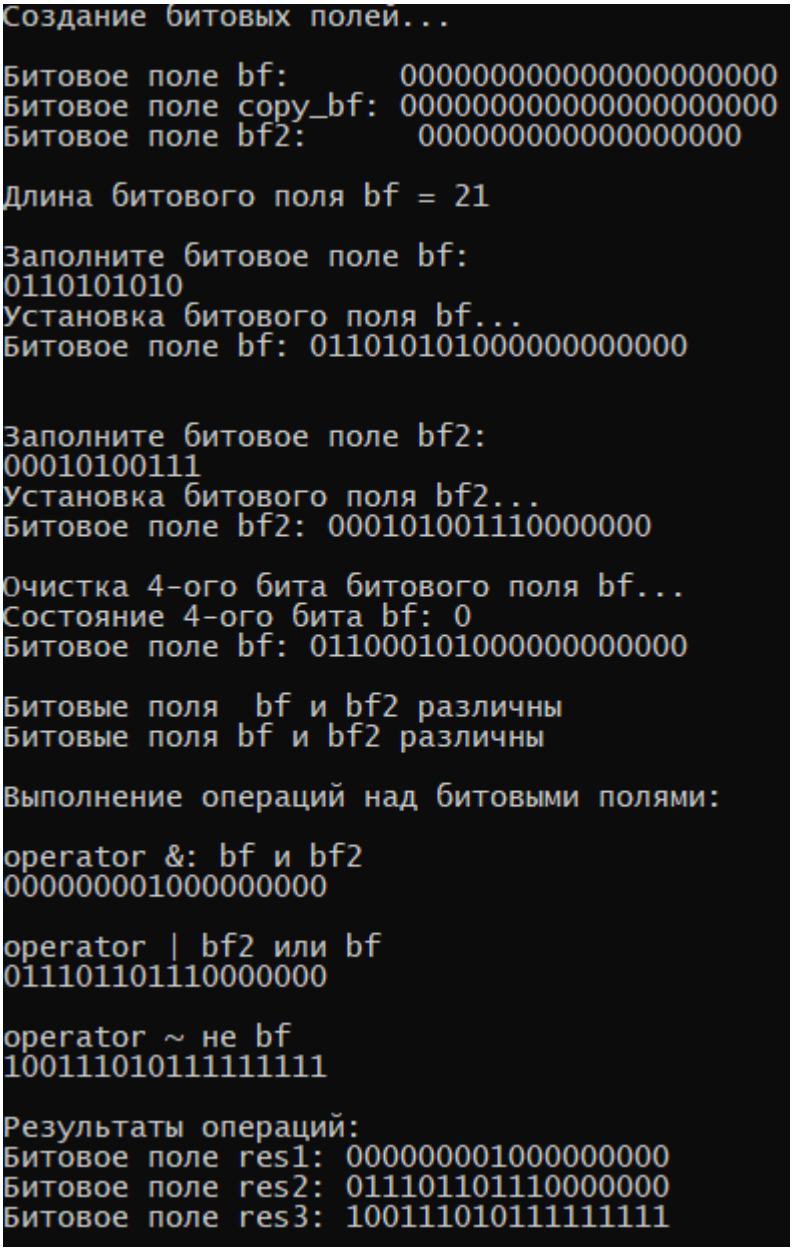


Рис. 1. Основное окно программы

2. На первом шаге создаются 3 битовых поля(рис.2)

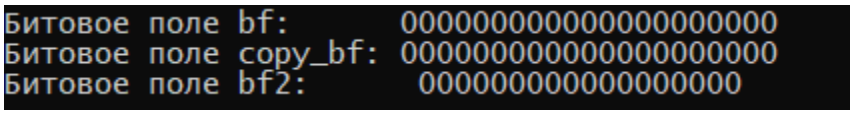


Рис.2 Создание битовых полей

3. На следующем шаге выполняется установка битового поля bf и выводится его длина(рис.3)

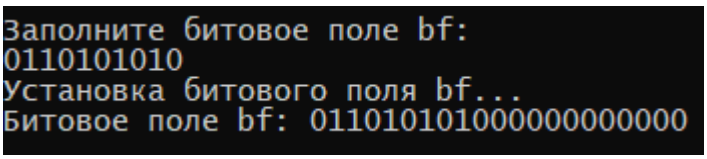


Рис.3 Установка битового поля bf с выводом длины

4. Далее выполняется установка битового поля bf2(рис.4)

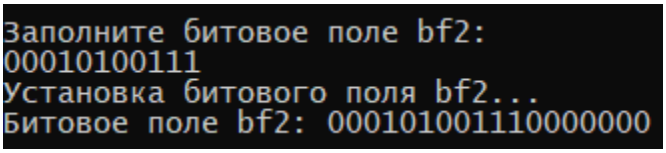


Рис.4 Установка битового поля bf2

5. На 5 шаге удаляется бит с номером 4 из битового поля bf(рис.5)

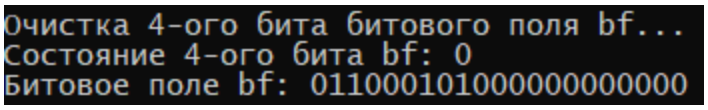


Рис.5 Удаление 4-го бита битового поля bf2

6. В первой строке проверяется операция равенства битовых полей bf и bf2,а во второй- операция неравенства(рис.6)

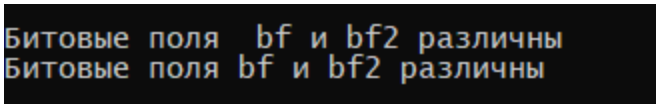


Рис.6 Сравнение битовых полей

7. На данном этапе выполняются различные операции с битовыми полями(рис.7)

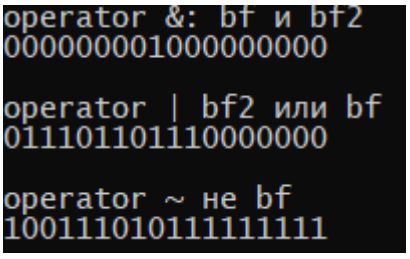


Рис.7 Операции над битовыми полями

8. На завершающем шаге выводятся результаты вычислений(рис.8)

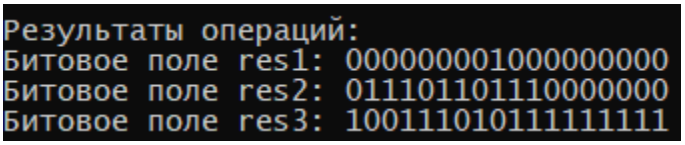


Рис.8 Результаты вычислений

2.2 Приложение для демонстрации работы множеств

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

```
Создание множеств...

Представление мощностей множеств:
Мощность множества A = 11
Мощность множества B = 21
Мощность множества копии B = 21
Мощность множества C = 16

Ввод элементов множества A:
Введите элементы множества A
How many element do you want enter?
5
1 2 3 4 5
Continue?(1/0)0

Ввод элементов множества B:
Введите элементы множества B
How many element do you want enter?
10
2 4 6 8 10 12 14 16 0 3
Continue?(1/0)0

Ввод элементов множества C:
Введите элементы множества C
How many element do you want enter?
2
4 15
Continue?(1/0)0

Сравнение множеств A и  B:

Проверка на равенство:
Множества A и B разной мощности

Проверка на неравенство:
Множества A и B разной мощности

Выполнение теоретико-множественных операций над множествами:

Объединение множества A с элементом 9
A= {1,2,3,4,5,9,}

Разность множества B с элементом 14
B= {0,2,3,4,6,8,10,12,16,}

Объединение множества A с множеством B
res1= {0,1,2,3,4,5,6,8,9,10,12,16,18,19,}

Пересечение множества A с множеством C
res2= {4,}

Дополнение к множеству C
res3= {0,1,2,3,5,6,7,8,9,10,11,12,13,14,}

Разность множеств A и C
res4= {1,2,3,5,9,}

Множества A,B,C
A= {1,2,3,4,5,9,}
B= {0,2,3,4,6,8,10,12,16,}
C= {0,1,2,3,5,6,7,8,9,10,11,12,13,14,}
```

Рис.1 Окно основной программы

2. По началу множества пустые,так как в них не содержится элементов. Далее представлены мощности множеств(рис.2)

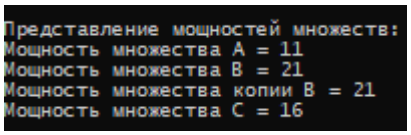


Рис.2 Мощности множеств A,B,C

3. На третьем этапе представлен процесс заполнения множеств элементами, введёными с клавиатуры(рис.3)

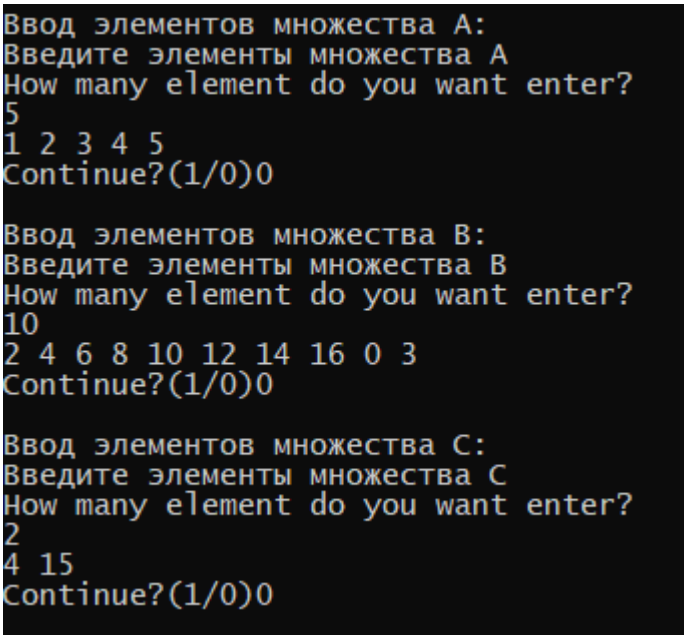


Рис.3 Заполнение множеств A,B,C

4. В этом случае сравниваются два множества на равенство и неравенство(рис.4)

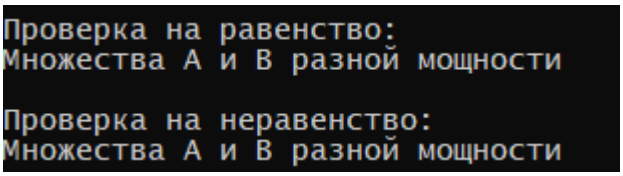


Рис.4 Сравнение множеств A и B

5. На рис.5 приведены основные операции с множествами(рис.5)

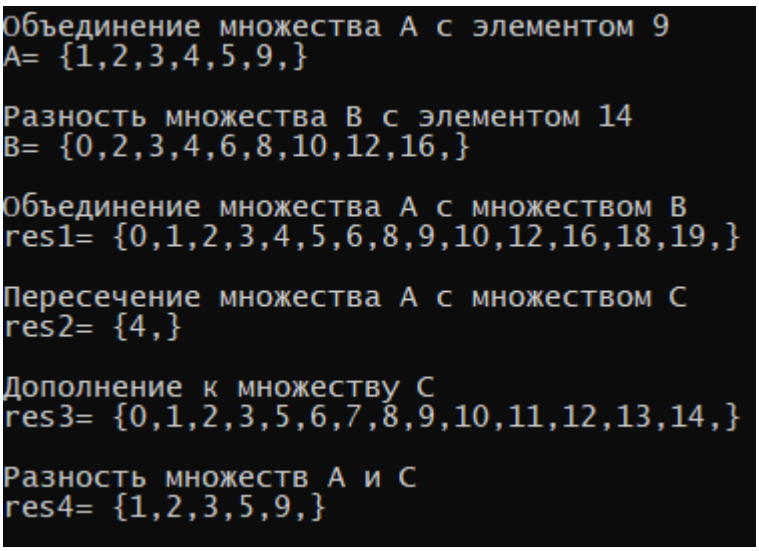


Рис.5 Основные операции с множествами A,B,C

6. В завершение были выведены множества A,B,C, которые принимали участие в программе(рис.6)

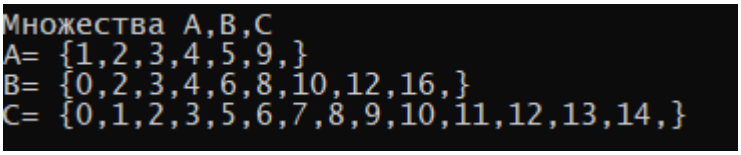


Рис.6 Вывод множеств A,B,C

2.3 «Решето Эратосфена»

1. Откройте приложение sample_primenumbers.exe.В результате появится окно ниже(рис.1)

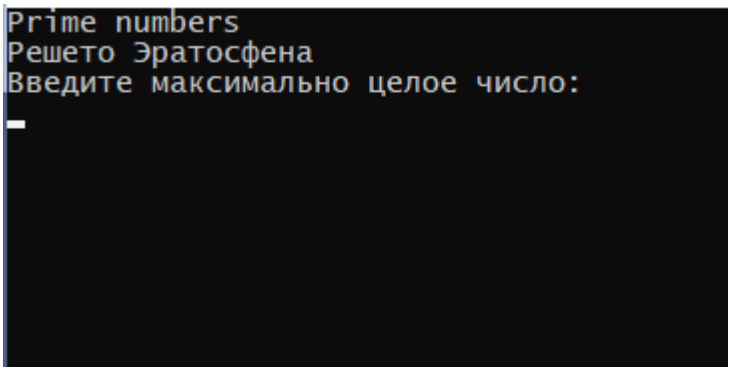
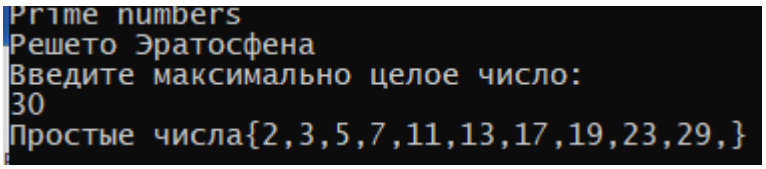


Рис.1 Окно основной программы

2. Вам будет необходимо ввести число ,до которого будут выведены все простые числа на экран. Для примера введём число 30 и посмотрим на результат(рис.2)



3. Рис.2 Все простые числа от 2 до 30

3 Руководство программиста

3.1 Описание алгоритмов

3.1.1 Битовые поля

Битовые поля представляют из себя характеристические массивы,где индексы каждого элемента – это элементы множества. Каждое битовое поле задаётся длиной(унивёрс битов),количеством единиц памяти(кол-во характеристических массивов) и памятью для их хранения. Элемент битового поля может находиться в двух состояниях: 1 и 0. 1- элемент содержится в множестве,а 0 – элемент не содержится в множестве. Данный алгоритм позволяет реализовать интерфейс для работы с множествами.

Описания методов:
Конструктор инициализатор инициализирует поля BitLen,MemLen и pMem.
Принимает параметр len ,которое получает поле BitLen. Далее вычисляется количество элементов памяти для хранения битового поля и на основании MemLen выделяется память под массив для хранения битового поля. На последнем шаге используется функция memset для инициализации нулями.

Конструктор копирования выполняет поверхностное копирование объекта. Значения полей переданного объекта копируются в объект,для которого вызывается конструктор копирования, в целях создания новой копии объекта.

Деструктор освобождает выделенную память из-под массива pMem и устанавливает значения полей объекта в 0.

.

Метод GetMemMask по заданной позиции бита генерирует для него битовую маску. Возвращается объект класса TBitField

Метод GetLength возвращает целое неотрицательное число бит. Это число называется длиной битового поля.

Метод SetBit по заданной позиции бита меняет его состояние с 0 на 1. Если состояние бита было изначально 1,то изменения состояния не произойдёт.

Метод ClrBit по заданной позиции бита меняет его состояние с 1 на 0. Если состояние бита было изначально 0,то изменения состояния не произойдёт.

Метод GetBit возвращает целое неотрицательное число. Это число называется состоянием бита, которое может быть либо 0,либо 1.

Операция == сравнивает поэлементно два битовых поля и в качестве результата возвращает True,если битовые поля равны и False ,если не равны.

Операция != сравнивает поэлементно два битовых поля и в качестве результата возвращает True,если битовые поля не равны и False ,если равны.

Операция = присваивает значения полей переданного объекта в параметры метода объекту,для которого эта операция была вызвана. Возвращается изменённый объект класса TBitField с новыми значениями полей,для которого была вызвана операция =.

Операция ~ инвертирует значения битов в случае 1 на 0 и в случае 0 на 1. В результате возвращается объект с инвертированными значениями битов.

Операция | выполняет побитовое сложение между двумя битовыми полями. Если хотя бы один из битов объектов принимает значение 1,то результирующий бит будет равен 1. Иначе 0. Возвращается новый объект как результат сложения битовых полей.

Операция | выполняет побитовое сложение между двумя битовыми полями. Если хотя бы один из битов объектов принимает значение 1,то результирующий бит будет равен 1. Иначе 0. Возвращается новый объект как результат сложения битовых полей.

Операция >> считывает строку из стандартного потока ввода cin и поэлементно устанавливает значения битов битового поля. Вводится строка

из 0 и 1. Каждый элемент строки последовательно принимается методами ClrBit и SetBit соответственно для установки значения бита в битовом поле.

Операция << выводит значения битов битового поля на экран в формате «10101010…» , используя метод GetBit для считывания состояния бита.

3.1.2 Множества

Множества по идее наследуются от класса TBitField. Множество – это класс TSet,реализованный на основе класса TBitField. Работа TSet заключается в том,что он использует класс TBitField как инструмент для создания множеств и осуществления теоретико-множественных операций. Максимальная мощность множества – это и есть длина битового поля. Таким образом, главная роль отводится классу TBitField,который и отвечает за техническую часть работы множеств.

Описания методов:

Конструктор инициализатор инициализирует поля MaxPower и BitField. Принимаемый параметр mp используется для инициализации полей MaxPower и BitField(в списке инициализации конструктора). В результате определяется мощность множества и создаётся объект класса BitField – интерфейс для работы с множествами.

Конструктор копирования копирует значения полей переданного объекта для инициализации объекта,для которого этот конструктор был вызван.

Конструктор преобразования типа выполняет неявное преобразование из типа объекта класса TBitField в тип TSet.

Оператор преобразования TBitField() выполняет неявно преобразование из объекта класса TSet в объект класса TBitField.

Метод GetMaxPower возвращает целое неотрицательное число. Это число называется мощностью множества, равной длине битового поля.

Метод InsElem реализует вставку элемента в множество. Элемент представляет из себя целое неотрицательное число. Внутри метода InsElem используется метод SetBit для установки значения бита битового поля. Ничего не возвращает.

Метод DelElem исключает элемент из множества. Внутри метода DelElem используется метод ClrBit для обнуления значения бита. Ничего не возвращает.

.

Метод IsMember делает проверку на принадлежность элемента множеству. Внутри метода IsMember используется метод GetBit,чтобы считать состояние бита и проверить его значение. В случае 1 – элемент принадлежит множеству. В противном случае не принадлежит множеству. Ничего не возвращает.

Операция == проверяет битовые поля множеств на равенство. Выполняется та же операция сравнения,что и для класса TBitField. В случае True множества одинаковы. В противном случае различны.

Операция != проверяет битовые поля множеств на неравенство. Выполняется та же операция сравнения,что и для класса TBitField. В случае True множества различны. В противном случае одинаковы.

Операция = реализует присваивание полей объекта класса,для которого была вызвана операция =, и переданного объекта. Возвращается изменённый объект класса TSet с новыми значениями полей,для которого была вызвана операция =.

Операция + ,принимающая элемент для вставки, осуществляет объединение множества с элементом. Внутри тела операции + содержится метод InsElem,который вставляет элемент на позицию Elem множества. Возвращается копия объекта с добавленным элементом Elem.

Операция -,принимающая элемент для удаления, вычитает из множества элемент Elem. Внутри тела операции – содержится метод DelElem,который исключает элемент из позиции Elem множества.. Возвращается копия объекта с исключённым элементом Elem.

Операция + отвечает за объединение множеств. Внутри тела операции создаётся новый объект класса TBitField,для которого выполняется побитовое сложение, реализованное в классе TBitField. Возвращается объект класса TSet как результирующий.

Операция - отвечает за пересечение множеств. Внутри тела операции создаётся новый объект класса TBitField,для которого выполняется побитовое умножение, реализованное в классе TBitField. Возвращается объект класса TSet как результирующий.

Операция ~ - это дополнение к множеству. Она исключает текущие элементы множества и добавляет все элементы универса без исключённых элементов в множество. Создаётся объект класса TBitField и применяется операция ~,реализованная в классе TBitField. Возвращается объект класса TSet как результирующий.

Операция – реализует разность множеств A и B. Эта операция исключает такие элементы из множества A,которые есть в множестве B. Создаются 2 объекта класса TBitField – один под результат,другой – инвертированный объект класса TBitField. Результатом является побитовое умножение неинвертированного объекта и инвертированного. Возвращается объект класса TSet как результирующий.

Операция >> считывает некоторое количество введённых элементов из стандартного потока ввода cin и осуществляет вставку элементов с помощью метода InsElem в множество.

Операция << выводит элементы множества в формате {e1,e2,….,en},где ej – это элемент множества и j = 1 до n. Открывается фигурная скобка {. Далее делается проверка на принадлежность элемента множеству. Непринадлежащие элементы нас не интересуют и не выводятся,поэтому выводятся только принадлежащие элементы множества через символ «,». Фигурная скобка закрывается.

3.1.3 «Решето Эратосфена»

Решето Эратосфена – это алгоритм, позволяющий найти все простые числа до заданного числа n. Суть этого алгоритма заключается в следующем:

1. Выписать подряд все числа от 2 до n
2. Пусть у нас есть переменная p=2 –первое простое число
3. Зачёркиваем все числа,кратные 2p,3p,4p…
4. Находим первое простое число в списке,большее p. Присваиваем его p
5. Повторяем шаги 3 и 4.

Данный алгоритм позволяет легко и быстро найти все простые числа.

3.2 Описание программной реализации

3.2.1 Описание класса TBitField

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

  // методы реализации
  int   GetMemIndex(const int n) const;
  TELEM GetMemMask (const int n) const;

  int BitsInMem = 16;
  int shiftsize = 4;

public:
  TBitField(int len);
  TBitField(const TBitField &bf);
  ~TBitField();

  // доступ к битам
  int GetLength(void) const;
  void SetBit(const int n);
  void ClrBit(const int n);
  int  GetBit(const int n) const;
  // битовые операции
  int operator==(const TBitField &bf) const;
  int 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& obj);
  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

GetMemIndex	**GetMemMask**	**GetLength**	**SetBit**
Назначение: Получение индекса элемента памяти	Назначение: Получение битовой маски по номеру бита	Назначение: Получение длины битового поля	Назначение: Установить бит в единицу
Входные Параметры- n- Номер бита	Входные Параметры- n- Номер бита	Входные Параметры: Отсутствуют	Входные Параметры: n- Номер бита
Выходные параметры Номер элемента памяти	Выходные Параметры: Битовая маска	Выходные параметры: Длина битового поля	Выходные параметры: отсутствуют

ClrBit	**GetBit**
Назначение: Установить бит в ноль	Назначение: Получение значения бита
Входные Параметры: n- Номер бита	Входные Параметры: n- Номер бита
Выходные параметры: отсутствуют	Выходные параметры: Получение значения бита (0 или 1)

Операции

Вывод **Operator<<**	**Ввод** **Operator>>**
Назначение:	Назначение:
Вывод битового поля	ввод битового поля
Входные параметры:	Входные параметры:
ostream& ostr-ссылка на поток.	Istream& istr-Ссылка на поток,
const TBitField& bf	TBitFitField& bf-
Константная ссылка	неконстантная ссылка
на битовое поле	на битовое поле
Выходные параметры:	Выходные параметры:
поток с	поток с введённой
битовым полем формата	битовой строкой
(1010101 и т.д)	

Конструкторы/деструктор

Конструктор инициализатор	**Конструктор копирования**
Назначение:	Назначение:
Создание	Копирование
битового поля	битовых полей
Входные параметры:	Входные Параметры:
Len-Длина	Const TBitField& bf –

битового поля	Константная ссылка на битовое поле
Выходные	Выходные
параметры:	параметры:
Отсутствуют	отсутствуют

Деструктор

Назначение:

Освобождение памяти

Входные

параметры:

отсутствуют

Выходные

параметры:

3.2.2 Описание класса TSet

```
class TSet
{
private:
  int MaxPower;
  TBitField BitField;
public:
  TSet(int mp);
  TSet(const TSet &s);
  TSet(const TBitField &bf);
  operator TBitField();
  // доступ к битам
  int GetMaxPower(void) const;
  void InsElem(const int Elem);
  void DelElem(const int Elem);
  int IsMember(const int Elem) const;
  // теоретико-множественные операции
  int operator== (const TSet &s) const;
  int 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);
  TSet operator-(const TSet& obj);
```

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

Битовые поля:

```
MaxPower – максимальная мощность множества
TBitField – битовое поле
```

Методы

GetMaxPower	**InsElem**	**DelElem**	**IsMember**
Назначение: Получение мощности множества Входные параметры: Отсутствуют Выходные Параметры: Мощность множества	Назначение: Добавление элемента в множество Входные параметры: Elem- добавляемый элемент Выходные параметры: отсутствуют	Назначение: Исключение элемента из множества Входные Параметры: Elem– удаляемый элемент Выходные параметры: отсутствуют	Назначение: Проверка на принадлежность Входные параметры: Elem – элемент для проверки Выходные параметры: Значение бита (0 или 1)

Операции

Равенство(==) Operator==

Назначение:

Проверка на равенство двух множеств

Входные параметры:

s- множество

Выходные параметры:

Целое число (0 или 1)

Неравенство(!=) Operator!=

Назначение:

Проверка на неравенство двух множеств

Входные параметры:

s – множество

Выходные параметры:

Целое число (0 или 1)

Присваивание(=)

Назначение:

Присвоение значений полей одного объекта класса другому

Входные параметры:

s – множество

Выходные параметры:

Ссылка на объект своего класса TSet

Объединение с элементом Operator+

Назначение:

Побитовое сложение элемента множества с элементом

Входные параметры:

Elem- добавляемый

Пересечение с элементом operator&

Назначение:

Побитовое умножение соответствующего элемента множества с элементом

Входные параметры:

Elem– добавляемый элемент

Пересечение множеств Operator&

Назначение:

Побитовое умножение элементов двух множеств

Входные параметры:

s – множество

элемент

Выходные параметры: Результирующее множество

Выходные параметры: Результирующее множество

Выходные параметры: Результирующее множество

Разность с элементом operator-
Назначение: Исключение соответствующего элемента множества
Входные параметры: Elem – вычитаемый элемент
Выходные параметры: Результирующее множество

Дополнение к множеству operator~
Назначение: Инвертировать значения битового поля. Это и будет дополнение к множеству.
Входные параметры: отсутствуют
Выходные параметры: Результирующее множество

Вывод Operator<<
Назначение:
Вывод элементов множества в формате({e1,e2,…,en})
Входные параметры: Ostream& ostr- ссылка на поток, Const TSet& s- константная ссылка на объект класса TSet
Выходные параметры: Поток с множеством формата(A={e1,e2,…,en} и т.д)

Ввод Operator>>
Назначение:

Заполнение множества элементами

Входные параметры: Istream& istr – ссылка на поток,TSet& s – ссылка на объект класса TSet

Конструктор инициализатор

Назначение: Создание множеств

Входные параметры: Mp – мощность множества

Выходные параметры: Отсутствуют

Конструктор Копирования

Назначение: Копирование множеств

Входные параметры: s – множество

Выходные параметры: Отсутствуют

Конструктор преобразования типа:

Назначение: Преобразование из TBitField в TSet

Входные параметры: Bf – Битовое поле

Выходные параметры: Отсутствуют

Оператор преобразования Operator TBitField()

Назначение: Преобразование из TSet в TBitField

Входные

параметры:	
Отсутствуют	
Выходные параметры:	
Объект класса TBitField	

Заключение

По результатам лабораторной работы были реализованы классы TSet и TBitField,а также написаны приложения и тесты для проверки работоспособности реализации. К лабораторной работе был составлен полный отчёт по теме со всеми подробными описаниями.

Литература

1. Битовые поля и операции над ними с.33
2. Битовые поля. Урок 32
3. Битовые поля раздел Битовые поля

Приложения

Приложение А. Реализация класса TSet

```
TSet::TSet(int mp) : BitField(mp)
{
    MaxPower = mp;
}

// конструктор копирования
TSet::TSet(const TSet &s) : BitField(s.GetMaxPower())
{
    MaxPower = s.GetMaxPower();
    BitField = s.BitField;
}

// конструктор преобразования типа
TSet::TSet(const TBitField &bf): BitField(bf)
{
    MaxPower = bf.GetLength();
    BitField = bf;

}

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

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
{
    if (Elem < 0 || Elem >= MaxPower) {
        throw "element not exist";
    }
    BitField.SetBit(Elem);
}

void TSet::DelElem(const int Elem) // исключение элемента множества
{
    if (Elem < 0 || Elem >= MaxPower) {
        throw "element not exist";
    }
    BitField.ClrBit(Elem);
}
```

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

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

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

TSet TSet::operator+(const TSet &s) // объединение
{
    if (*this == s) {
        return *this;
    }
    TBitField res(1);
    res = BitField | s.BitField;
    return TSet(res);
}

TSet TSet::operator+(const int Elem) // объединение с элементом
{
    if (Elem < 0 || Elem >= MaxPower) {
        throw "element not exist";
    }
    if (IsMember(Elem)) {
        return TSet(*this);
    }
    TBitField res(BitField);
    res.SetBit(Elem);
    return TSet(res);
}

TSet TSet::operator-(const int Elem) // разность с элементом
{
    if (Elem < 0 || Elem >= MaxPower) {
        throw "element not exist";
    }
    if (!IsMember(Elem)) {
        return TSet(*this);
    }
    TBitField res(BitField);
    res.ClrBit(Elem);
    return TSet(res);
}
```

```
TSet TSet::operator-(const TSet& obj) {
    TBitField res(1);
    TBitField inv(obj.BitField);
    res = BitField & (~inv);
    return TSet(res);
}

TSet TSet::operator*(const TSet &s) // пересечение
{
    if (*this == s) {
        return *this;
    }
    TBitField res(1);
    res = BitField &s.BitField;
    return TSet(res);
}

TSet TSet::operator~(void) // дополнение
{
    TBitField tmp(*this);
    tmp = ~tmp;
    return TSet(tmp);
}

 // перегрузка ввода/вывода

istream& operator>>(istream& istr, TSet& bf) {
    unsigned int e = 1;
    size_t count;
    cout << "How many element do you want enter?" << endl;
    cin >> count;
    int i = 0;
    while (i < count) {
        istr >> e;
        bf.InsElem(e);
        i++;
    }
    return istr;
}

ostream& operator<<(ostream &stream, const TSet &obj) // вывод
{
    size_t i, n;
    stream << "{";
    n = obj.MaxPower;
    for (i = 0; i < n; i++) {
        if (obj.IsMember(i)) {


            stream << i << ",";
        }
    }
    stream << "}";
    return stream;
}
```

Приложение Б. Реализация класса TBitField

```
TBitField::TBitField(int len)
{
    if (len < 0) {
        throw "Negative length";
    }
    BitLen = len;
    MemLen = ((len + BitsInMem - 1) >> shiftsize);//количество участков памяти под хранение элементов 1-N
    pMem = new TELEM[MemLen];//создать характеристический массив
    memset(pMem, 0, MemLen * sizeof(TELEM));//заполнить MemLen кусков нулями
}

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

TBitField::~TBitField()
{
    delete[] pMem;
    MemLen = 0;
    BitLen = 0;
}

int TBitField::GetMemIndex(const int n) const // индекс Мем для бита n
{
    return n >> shiftsize;
}

TELEM TBitField::GetMemMask(const int n) const // битовая маска для бита n
{
    return 1 << (n & (BitsInMem-1));
}

// 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 "Negative length";
    }
    pMem[GetMemIndex(n)] = pMem[GetMemIndex(n)] | GetMemMask(n);
}

void TBitField::ClrBit(const int pos) // очистить бит
```

```cpp
void TBitField::ClrBit(const int pos) // очистить бит
{
    if (pos < 0 || pos >= BitLen) {
        throw "Negative length";
    }
    pMem[GetMemIndex(pos)] = pMem[GetMemIndex(pos)] & ~GetMemMask(pos);
}

int TBitField::GetBit(const int n) const // получить значение бита
{
    if (n < 0 || n >= BitLen) {
        throw "Negative length";
    }
    int test = pMem[GetMemIndex(n)] & GetMemMask(n);
    return (pMem[GetMemIndex(n)] & GetMemMask(n) );
}

// битовые операции

const TBitField& TBitField::operator=(const TBitField &bf) // присваивание
{
    if (*this == bf) {
        return *this;
    }

    BitLen = bf.BitLen;
    if (MemLen != bf.MemLen) {
        MemLen = bf.MemLen;
        TELEM* p = new TELEM[MemLen];
        delete[] pMem;
        pMem = p;
    }
    memcpy(pMem, bf.pMem, MemLen * sizeof(TELEM));
    return *this;
}

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

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

TBitField TBitField::operator|(const TBitField &bf) // операция "или"
{


        if (BitLen != bf.BitLen) {
            TELEM* p = new TELEM[bf.MemLen];
            memcpy(p, pMem, MemLen * sizeof(TELEM));
            delete[] pMem;
            BitLen = bf.BitLen;
            MemLen = bf.MemLen;
            pMem = p;
        }
        TBitField tmp(*this);
        for (size_t i = 0; i < bf.MemLen; i++) {
            tmp.pMem[i] = tmp.pMem[i] | bf.pMem[i];
        }
        return tmp;

}

TBitField TBitField::operator&(const TBitField &bf) // операция "и"
{
    if (BitLen != bf.BitLen) {
        TELEM* p = new TELEM[bf.MemLen];
        memcpy(p, pMem, bf.MemLen * sizeof(TELEM));
        delete[] pMem;
        BitLen = bf.BitLen;
        MemLen = bf.MemLen;
        pMem = p;
    }
    TBitField tmp(*this);
    for (size_t i = 0; i < bf.MemLen; i++) {
        tmp.pMem[i] = tmp.pMem[i] & bf.pMem[i];
    }

    return tmp;
}

TBitField TBitField::operator~(void) // отрицание
{
    TBitField tbf = (*this);
    for (int i = 0; i < BitLen; i++)
    {
        if (tbf.GetBit(i))
            tbf.ClrBit(i);
        else
            tbf.SetBit(i);
    }
    return tbf;
}


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

//Ввод
istream& operator>>(istream& istr, TBitField& obj) {
    string BitField;
    istr >> BitField;

    for (int i = 0; i < BitField.length(); i++) {
        if (BitField[i] == '1') {
            obj.SetBit(i);
        }
        else {
            obj.ClrBit(i);
        }
    }
    return istr;
}
```