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).

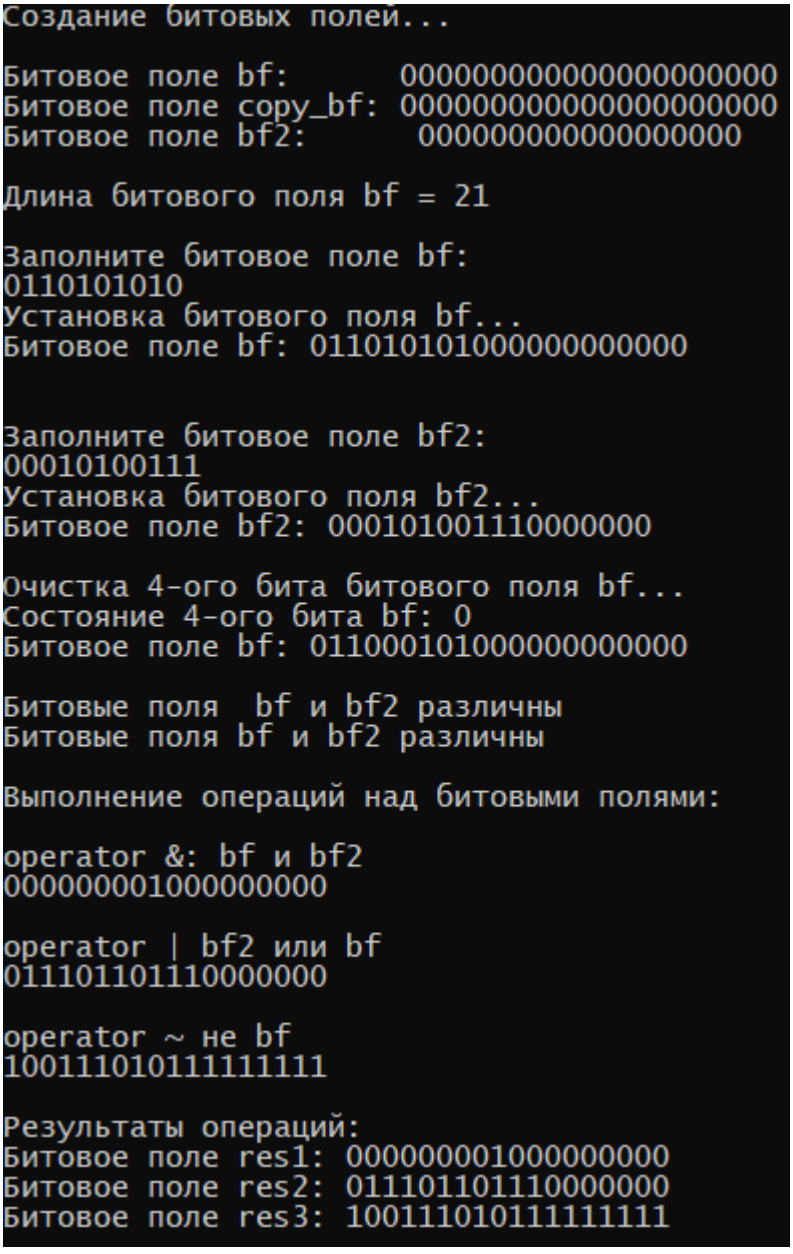


Рис. 1. Основное окно программы

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

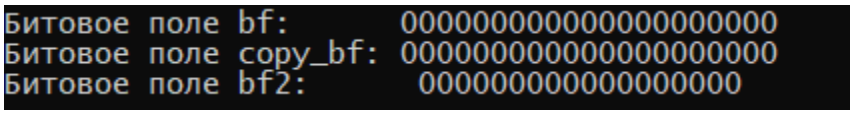


Рис.2 Создание битовых полей

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

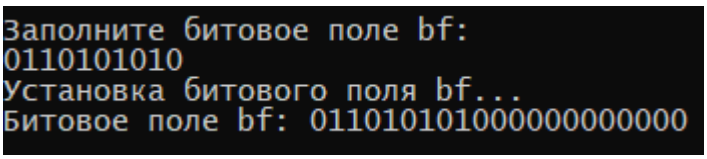


Рис.3 Установка битового поля bf с выводом длины

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

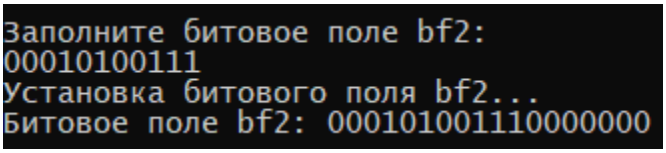


Рис.4 Установка битового поля bf2

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

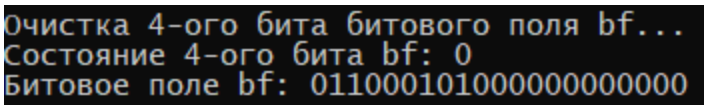


Рис.5 Удаление 4-го бита битового поля bf2

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

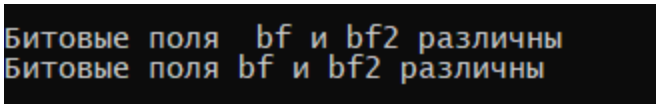


Рис.6 Сравнение битовых полей

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

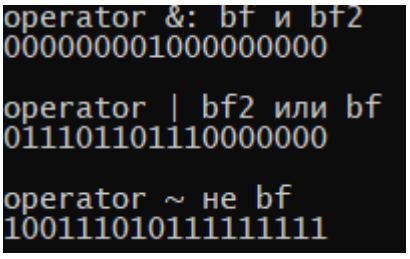


Рис.7 Операции над битовыми полями

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

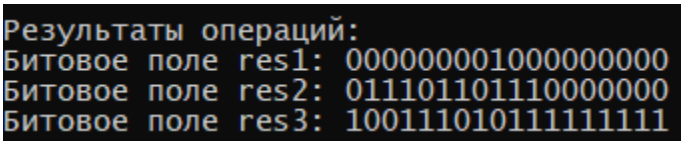


Рис.8 Результаты вычислений

2.2 Приложение для демонстрации работы множеств

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

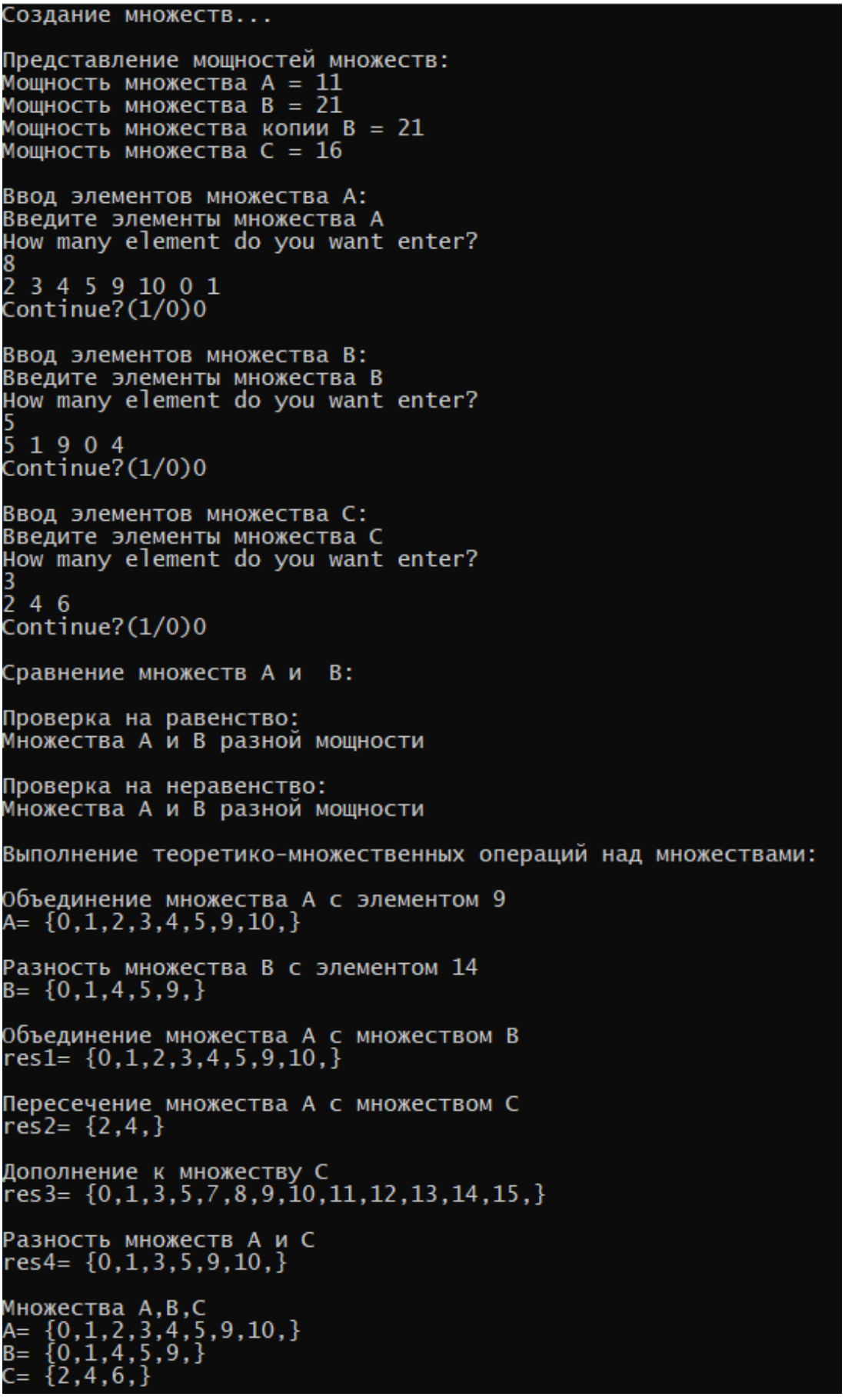


Рис.1 Окно основной программы

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

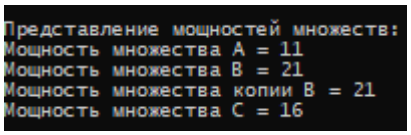


Рис.2 Мощности множеств A,B,C

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

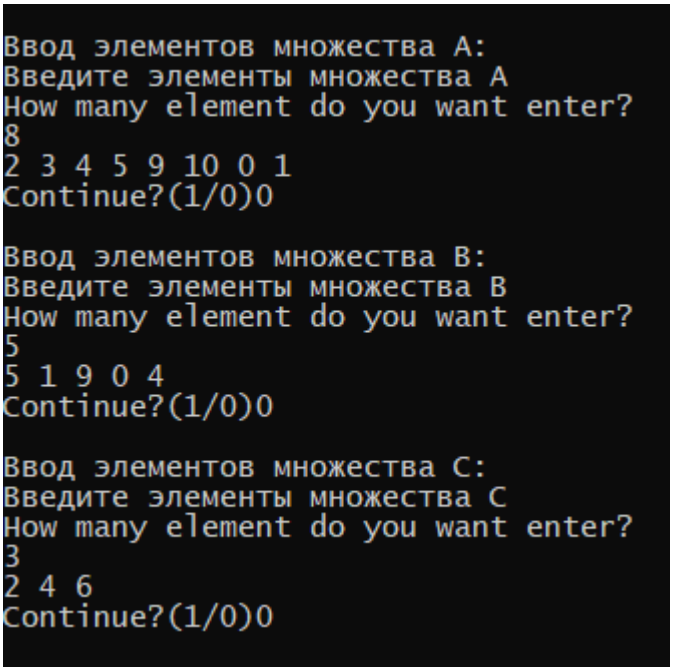


Рис.3 Заполнение множеств A,B,C

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

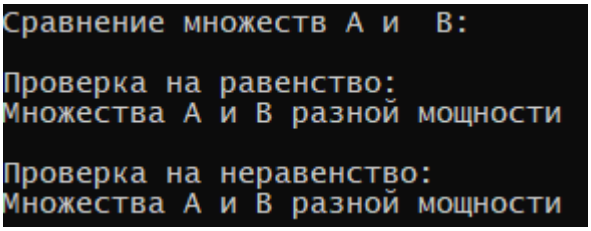


Рис.4 Сравнение множеств A и B

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

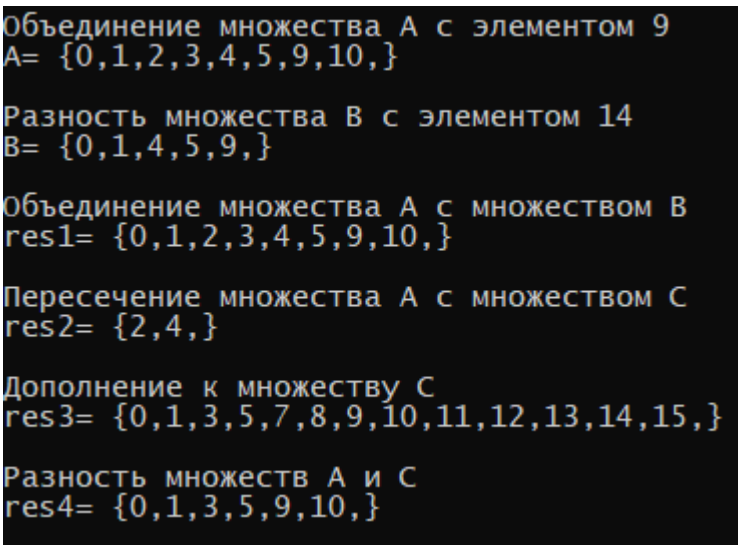


Рис.5 Основные операции с множествами A,B,C

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

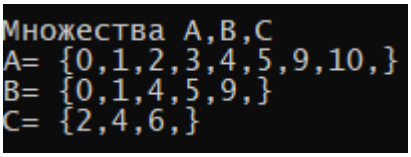


Рис.6 Вывод множеств A,B,C

2.3 «Решето Эратосфена»

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

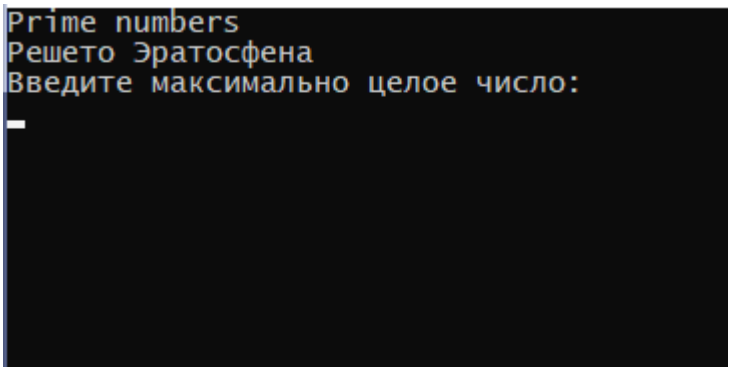
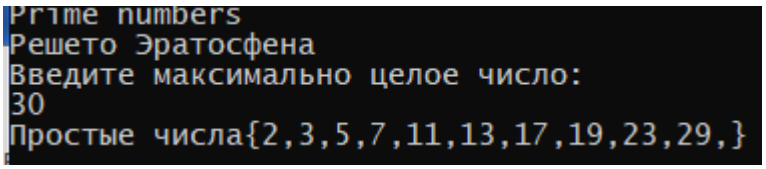


Рис.1 Окно основной программы

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 последовательность нулей и единиц. 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.

Пусть дано множество A:

A = { 0, 2, 3 , 4, 6 }

index	0	1	2	3	4	5	6	7
bits	1	0	1	1	1	0	1	0

Битовое поле:

1	0	1	1	1	0	1	0

Описания методов:

Конструктор инициализатор

Входные параметры: len – длина битового поля.

Выходные параметры: отсутствуют.

Алгоритм:

1. Вычисляется количество элементов памяти для хранения битового поля.

2. Выделяется память под массив для хранения битового поля размера MemLen.

3. Функция memset инициализацирует память нулевыми значениями.

Конструктор копирования.

Входные данные: константная ссылка на объект типа TBitField.

Выходные данные: отсутствуют.

Алгоритм:

Копирует значения полей переданного объекта в поля текущего объекта.

Деструктор.

Входные данные: отсутствуют.

Выходные данные: отсутствуют

Алгоритм:

1. Освобождает выделенную память из-под массива pMem.
2. Устанавливает значения полей объекта в 0.

.

GetMemMask

Входные данные: n – номер бита

Выходные данные: число типа TELEM,где TELEM – тип элементов массива pMem.

Алгоритм:

Выполняет побитовый сдвиг единицы влево на (n & (BitsInMem-1) бит, где BitsInMem – это количество битов одной единице памяти.

0	0	n	…	…	0	0	0

GetLength

Входные данные: отсутствуют.

Выходные данные: BitLen – длина битового поля.

Алгоритм:

Возвращает длину битового поля.

SetBit

Входные данные: n – номера бита.

Выходные данные: отсутствуют.

Алгоритм:

1. Проверка номера бита на границы битового поля.

2. Выполняем операцию побитового сложения с элементом и битовой маской по номеру бита n. Используются методы GetMemIndex и GetMemMask соответственно.
3. Результат сложения записывается по номеру элемента в памяти. Используется метод GetMemIndex.

ClrBit

Выходные данные: n – номер бита.

Входные данные: отсутствуют.

Алгоритм:

1. Проверка номера бита на границы битового поля.
2. Выполняем операцию побитового умножения с элементом и битовой маской(битовая маска инвертируется с помощью операции ~) Используются методы GetMemIndex и GetMemMask соответственно.
3. Результат умножения записывается по номеру элемента в памяти.

GetBit

Входные данные: n – номер бита

Выходные данные: значение бита с номером n

Алгоритм:

Возвращает значение бита с номером n.

Operator!=

Входные данные: константная ссылка на тип TBitField

Выходные данные: число: 0 или 1

Алгоритм:

1. Выполняет побитовое сравнение значений битовых полей.
2. Возвращает 0 – битовые поля не не равны, или 1 – битовые поля не равны

Operator==

Входные данные: константная ссылка на тип TBitField

Выходные данные: число 0 или 1

Алгоритм:

1.Выполняет побитовое сравнение значений битовых полей.

2 .Возвращает 0 – битовые поля не равны, или 1 – битовые поля равны

Operator=

Входные данные: константная ссылка на объект типа TBitField

Выходные данные: *this – ссылка на объект себя

Алгоритм:

1. Проверка на самоприсваивание.
2. Проверка на равенство длин битовых полей. Иначе память перевыделяется.
3. Копирование значений элементов переданного объекта в текущий объект.
4. Возвращается *this.

Operator&

Входные данные: константная ссылка на объект типа TBitField

Выходные данные: объект типа TBitField

Алгоритм:

1.Создаётся копия текущего объекта

2. Побитовое умножение битовых полей. Если оба значения бита равны 1,результат равен 1 . Иначе если хотя бы 1 ноль,то результирующий бит равен 0.

3. Возвращается объект типа TBitField как результат побитового умножения.

BF1	1	1	1	1	0	1	0
BF2	0	1	1	1	0	0	0
BF1&BF2	0	1	1	1	0	0	0

Operator~

Входные данные: отсутствуют

Выходные данные: объект типа TBitField

Алгоритм:

1. Проверка на равенство размером. Иначе создаётся временный объект, выделяется память под нужный размер, значения старой памяти копируются в новую и старая память удаляется.
2. Указатель на старую память получает значение указателя на выделенную память и временный объект удаляется.

3. Выполняется побитовое сложение. Если при сложении значений битовых полей оба бита равны 0,то результирующее значение 0. Если хотя бы 1 единица,то результирующее значение 1.
4. Возвращается объект типа TBitField как результат инвертации.

BF1	1	1	0	1	0	1	0
~BF1	0	0	1	0	1	0	1

Operator|

Входные параметры: константная ссылка на объект типа TBitField

Выходные параметры: объект типа TBitField

Алгоритм:

1. Проверка на равенство размеров. Иначе создаётся временный объект, выделяется память под нужный размер, значения старой памяти копируются в новую и старая память удаляется.
2. Указатель на старую память получает значение указателя на выделенную память и временный объект удаляется.
3. Выполняется побитовое сложение. Если при сложении значений битовых полей оба бита равны 0,то результирующее значение 0. Если хотя бы 1 единица,то результирующее значение 1.
4. Возвращается объект типа TBitField как результат побитового сложения.

BF1	1	1	0	1	0	1	0
BF2	0	0	1	1	0	0	0
BF1|BF2	1	1	1	1	0	1	0

Operator>>

Входные параметры: istr - ссылка на стандартный поток ввода, неконстантная ссылка на объект типа TBitField.

Выходные параметры: istr – ссылка на стандартный поток ввода.

1. Вводится посл-ть из 0 и 1

2. Если значение 0,то вызывается метод ClrBit и соответствующее значение бита битового поля устанавливается в 0. В противном случае вызывается метод SetBit.
3. Возвращается ссылка на стандартный поток istr.

Operator<<

Входные данные: ostr – ссылка на стандартный поток вывода,константная ссылка на объект типа TBitField.

Выходные данные:ostr - ссылка на стандартный поток вывода

Алгоритм:

1. Открывается фигурная скобка {
2. Выполняется проверка на принадлежность i-того элемента множества. Если i-тый элемент принадлежит множеству,то выводится на консоль с помощью метода GetBit. Иначе счётчик увечивается на 1 и итерация повторяется
3. Закрывается фигурная скобка }.
4. Возвращается ссылка на стандартный поток вывода.

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

Описания методов:

Конструктор инициализатор.

Входные параметры: mp- максимальная мощность множества

Выходные параметры: отсутствуют

Алгоритм:

1. Поле MaxPower инициализируется значением mp.

2. Присваивание полю BitField значения bf..

Конструктор копирования.

Входные значения: константная ссылка на объект типа TSet

Выходные значения: отсутствуют

Алгоритм:

Выполняется копирование значений полей переданного объекта в текущий объект.

Конструктор преобразования:

Входные данные: константная ссылка на объект типа TBitField

Выходные данные: отсутствуют

Алгоритм:

1. MaxPower инциализируется значением len.

2. Присваивание TBitField и bf.

Operator TBitField()

Входные параметры: отсутствуют

Выходные параметры: объект типа TBitField

Алгоритм:

1.Создаётся новый объект типа TBitField с вызовом конструктора копирования.

2. Возвращается новый объект типа TBitField.

GetMaxPower

Входные данные: отсутствуют

Выходные данные: MaxPower- максимальная мощность множества.

Алгоритм:

Возвращается максимальная мощность множества.

InsElem

Входные данные: elem- элемент включения во множество

Выходные данные: отсутствуют

Алгоритм:

Вызывает метод Setbit с переданным параметром elem.

DelElem

Входные параметры: elem – элемент для исключения из множества

Выходные параметры: отсутствуют

Алгоритм:

Вызывается метод ClrBit с переданным параметром elem.

IsMember

Входные параметры: elem- элемент для проверки на принадлежность множеству

Выходные параметры: число: 0 или 1

Алгоритм:

1. Вызывает метод GetBit с переданным параметром elem.
2. Возвращает результат вызова метода GetBit. 1 – элемент принадлежит множеству,0 – не принадлежит.

Operator==

Входные параметры: константная ссылка на объект типа TSet

Выходные параметры: число: 0 или 1

Алгоритм:

1. Выполняется сравнение битовых полей двух множеств.
2. Возвращается 0 – множества равны,0 – не равны

Operator!=

Входные параметры: константная ссылка на объект типа TSet

Выходные параметры: число: 0 или 1

Алгоритм:

1.Выполняется сравнение битовых полей двух множеств.

2.Возвращается 0 – множества не не равны. 1 – не равны

Operator=

Входные параметры: константная ссылка на объект типа TSet

Выходные параметры: *this – ссылка на объект себя.

Алгоритм:

1. Проверка на самоприсваивание.
2. Присваивание BitField и bf.
3. Возвращается *this

Operator+

Входные параметры: elem – элемент для объединения с множеством

Выходные параметры: новый объект TSet с объёдинённым элементом

Алгоритм:

1. Создаётся новый объект TBitField – копия BitField Вызывается метод SetBit с переданным параметром elem.
2. Возвращается конструктор преобразования с результирующим объектом.

A = { 1 , 2 , 3} MaxPower = 6, elem = 5

A` = {1,2,3,5}

A	0	1	1	3	0	0
A+elem	0	1	1	1	0	1

Operator-

Входные параметры: elem – элемент для вычитания из множества

Выходные параметры: новый объект TSet с удалённым элементом

Алгоритм:

1.Создаётся новый объект TBitField – копия BitField

2.Вызывается метод ClrBit с переданным параметром elem.

3. Возвращается конструктор преобразования TSet с результирующим объектом.

A = { 0 , 1 , 2,4,5} MaxPower = 6, elem = 5

A` = {0,1,2,4}

A	1	1	1	0	1	1
A-	1	1	1	0	1	0

elem						

Operator+

Входные параметры: константная ссылка на объект типа TSet

Выходные данные: новый объект TSet с результатом объединения множеств.

Алгоритм:

1. Создаётся новый объект TBitField – копия BitField
2. Выполняется побитовое сложение двух битовых полей.
3. Возвращается конструктор преобразования TSet с результирующим объектом.

A= {1,2,4,5},MaxPower = 6

B = {0,3} MaxPower = 6

A|B = {0,1,2,3,4,5}

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

Operator~

Входные данные: отсутствуют

Выходные данные: новый объект типа TSet

Алгоритм:

1. Создаётся новый объект типа TBitField – копия BitField
2. Применяется операция ~для битового поле.
3. Возвращается конструктор преобразования TSet с результирующим объектом.

A = {1,4,5} MaxPower = 6

~A= { 0,2,3}

A	0	1	1	0	1	1
~A	1	0	0	1	0	0

Operator&

Входные параметры: константная ссылка на объект типа TSet

Выходные параметры: новый объект типа TBitField с объединением множеств.

Алгоритм:

1. Создаётся новый объект типа TBitField – копия BitField.

2. Выполняет побитовое умножение битовых полей.

3. Возвращается конструктор преобразования TSet с результирующим объектом.

A= {1,2,4,5,0},MaxPower = 6

B = {0,3,1} MaxPower = 6

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

Operator-

Входные данные: константная ссылка на объект типа TSet

Выходные параметры: новый объект типа TBitField с исключёнными элементами.

Алгоритм:

1. Создаётся новый объект 1 типа TBitField – копия BitField.

2. Создаётся новый объект 2 типа TBitField – копия переданного объекта.

3. Выполняется побитовое умножение объекта 1 и инвертированного объекта 2.

4. Возвращается конструктор преобразования TSet с результирующим объектом.

A= {1,2,3,5},MaxPower = 6

B = {2,3,4,0} MaxPower = 6

A	0	1	1	1	0	1
B	1	0	1	1	1	0
~B	0	1	0	0	0	1
A-B	0	1	0	0	0	1

Operator>>

Входные данные: : istr - ссылка на стандартный поток ввода, неконстантная ссылка на объект типа TBitField.

Выходные данные: ссылка на стандартный поток ввода

Алгоритм:

1. Вводится кол-во элементов, которое необходимо добавить в множество.
2. Вводятся последовательно некоторые числа(элементы множества)
3. Вызывается метод InsElem.
4. Возвращается ссылка на стандартный поток ввода

Operator<<

Входные данные:

Выходные данные:

Алгоритм:

1. Открывается фигурная скобка {
2. Получаем размер множества.
3. В цикле делаем проверку на принадлежность i-того элемента множеству.
4. Если i-тый элемент принадлежит множеству,то выводится на экран i-тый элемент. Иначе следующая итерация.
5. Закрывается фигурная скобка.
6. Возвращается ссылка на стандартный поток вывода.

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.

GetMemIndex

Назначение: Получение индекса элемента памяти

Входные Параметры- n- Номер бита

Выходные параметры Номер элемента памяти

GetMemMask

Назначение: Получение битовой маски по номеру бита

Входные Параметры- n- Номер бита

Выходные Параметры: Битовая маска

GetLength

Назначение: Получение длины битового поля

Входные Параметры: Отсутствуют

Выходные параметры: Длина битового поля

SetBit

Назначение: Установить бит в единицу

Входные Параметры: n- Номер бита

Выходные параметры: отсутствуют

ClrBit

Назначение: Установить бит в ноль

Входные Параметры: n- Номер бита

Выходные параметры: отсутствуют

GetBit

Назначение: Получение значения бита

Входные Параметры: n- Номер бита

Выходные параметры: Получение значения бита (0 или 1)

Операции

Вывод
Operator<<

Назначение:

Вывод битового поля

Входные параметры:

ostream& ostr-ссылка на поток.

const TBitField& bf

Константная ссылка на битовое поле

Выходные параметры:

поток с битовым полем формата (1010101 и т.д)

Ввод
Operator>>

Назначение:

ввод битового поля

Входные параметры:

Istream& istr-Ссылка на поток,

TBitFitField& bf-

неконстантная ссылка на битовое поле

Выходные параметры:

поток с введённой битовой строкой

Конструкторы/деструктор

Конструктор инициализатор

Назначение:

Создание битового поля

Входные параметры:

Len-Длина битового поля

Выходные параметры:

Отсутствуют

Конструктор копирования

Назначение:

Копирование битовых полей

Входные Параметры:

Const TBitField& bf –

Константная ссылка на битовое поле

Выходные параметры:

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	Назначение: Добавление элемента в множество	Назначение: Исключение элемента из множества	Назначение: Проверка на принадлежность
Входные параметры: Отсутствуют	Входные параметры: Elem- добавляемый элемент	Входные Параметры: Elem– удаляемый элемент	Входные параметры: Elem – элемент для проверки
Выходные Параметры: Мощность множества	Выходные параметры: отсутствуют	Выходные параметры: отсутствуют	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 элемент

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

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;
}
```

```
//создание множеств
TSet A(10 + 1);
TSet B(20 + 1);
TSet C(15 + 1);
TSet copy_B(B);
TSet res1(1), res2(1), res3(1),res4(1);
cout << endl;
cout << "Представление мощностей множеств:" << endl;
//мощности множеств
cout << "Мощность множества A = " << A.GetMaxPower() << endl;
cout << "Мощность множества B = " << B.GetMaxPower() << endl;
cout << "Мощность множества копии B = " << copy_B.GetMaxPower() << endl;
cout << "Мощность множества C = " << C.GetMaxPower() << endl;
cout << endl;
int choice = 1;

//заполнение множеств
cout << "Ввод элементов множества A:" << endl;
cout << "Введите элементы множества A" << endl;
while (choice == 1) {
    cin >> A;
    cout << "Continue?(1/0)";
    cin >> choice;
}
choice = 1;
cout << endl;
cout << "Ввод элементов множества B:" << endl;
cout << "Введите элементы множества B" << endl;
while (choice == 1) {
    cin >> B;
    cout << "Continue?(1/0)";
    cin >> choice;
}
choice = 1;
cout << endl;
cout << "Ввод элементов множества C:" << endl;
cout << "Введите элементы множества C" << endl;
while (choice == 1) {
    cin >> C;
    cout << "Continue?(1/0)";
    cin >> choice;
}
//проверка тройного присваивания
/*A = B = copy_B;*/
/*A = copy_B = B;*/

//проверка на равенство множеств
cout << endl;
cout << "Сравнение множеств A и  B: " << endl;
cout << endl;
cout << "Проверка на равенство:" << endl;
if (A == B) {
    cout << "Множества A и B равной мощности" << endl;
}
else {
```

```
        cout << "Множества A и B разной мощности" << endl;
    }
    cout << endl;
    //проверка на неравенство множеств
    cout << "Проверка на неравенство:" << endl;
    if (A != B) {
        cout << "Множества A и B разной мощности" << endl;
    }
    else {
        cout << "Множества A и B равной мощности" << endl;
    }
    cout << endl;
    //теоретико-множественные операции над множествами
    cout << "Выполнение теоретико-множественных операций над множествами: " << endl;
    cout << endl;
    cout << "Объединение множества A с элементом 9" << endl;
    A = A + 9;
    cout << "A= " << A << endl;
    cout << endl;
    cout << "Разность множества B с элементом 14" << endl;
    B = B - 14;
    cout << "B= " << B << endl;
    cout << endl;

    cout << "Объединение множества A с множеством B" << endl;
    res1 = A + B;
    cout << "res1= " << res1 << endl;
    cout << endl;

    cout << "Пересечение множества A с множеством C" << endl;
    res2 = A * C;
    cout << "res2= " << res2 << endl;
    cout << endl;

    cout << "Дополнение к множеству C" << endl;
    res3 = ~C;
    cout << "res3= " << res3 << endl;
    cout << endl;

    cout << "Разность множеств A и C" << endl;
    res4 = A - C;
    cout << "res4= " << res4 << endl;
    cout << endl;

    //Множества A,B,C
    cout << "Множества A,B,C" << endl;
    cout << "A= " << A << endl;
    cout << "B= " << B << endl;
    cout << "C= " << C << endl;

    return 0;
}
```

```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;
}
```

```
    setlocale(LC_ALL, "rus");
    cout << "Создание битовых полей..." << endl;
    TBitField bf(20 + 1);//оригинал
    TBitField copy_bf(bf);//копия
    TBitField bf2(17 + 1);
    TBitField res1(1), res2(2), res3(3);
    cout << endl;
    cout << "Битовое поле bf: " <<  "     "<< bf << endl;
    cout << "Битовое поле copy_bf: " << copy_bf << endl;
    cout << "Битовое поле bf2: " << "     " << bf2<< endl;
    cout << endl;

    cout << "Длина битового поля bf = " << bf.GetLength() << endl;
    cout << endl;

    cout << "Заполните битовое поле bf: " << endl;
    cin >> bf;
    cout << "Установка битового поля bf..." << endl;
    cout << "Битовое поле bf: " << bf << endl;
    cout << endl;

    //проверка бита на принадлежность
    bool status_bit = bf.GetBit(16);
    cout << endl;

    cout << "Заполните битовое поле bf2: " << endl;
    cin >> bf2;
    cout << "Установка битового поля bf2..." << endl;
    cout << "Битовое поле bf2: " << bf2 << endl;
    cout << endl;

    cout << "Очистка 4-ого бита битового поля bf..." << endl;
    bf.ClrBit(4);//установить 4-ый бит в 0.
    cout << "Состояние 4-ого бита bf: " << bf.GetBit(4) << endl;
    cout << "Битовое поле bf: " << bf << endl;
    cout << endl;

    if (bf == bf2) {
        cout << "Битовые поля bf и bf2 одинаковы" << endl;
    }
    else {
        cout << "Битовые поля  bf и bf2 различны" << endl;
    }

    if (bf != bf2) {
        cout << "Битовые поля bf и bf2 различны" << endl;
    }
    else {
        cout << "Битовые поля bf и bf2 одинаковы" << endl;
    }
    cout << endl;

    ////проверка на операции к себе
    //bf = bf | bf;
    //copy_bf = copy_bf & copy_bf;
    //bf2 = ~bf2;

    cout << "Выполнение операций над битовыми полями: " << endl;
    cout << endl;

  cout << endl;
  //проверка операций
  cout << "operator &: bf и bf2" << endl;
  res1 = bf & bf2;
  cout << res1 << endl;
  cout << endl;
  cout << "operator | bf2 или bf" << endl;
  res2 = bf2 | bf;
  cout << res2 << endl;
  cout << endl;
  cout << "operator ~ не bf" << endl;
  res3 = ~bf;
  cout << res3 << endl;

  //проверка тройного присваивания
  /*bf = bf2 = copy_bf;*/
  cout << endl;
  cout << "Результаты операций: " << endl;
  //вывод на экран
  cout << "Битовое поле res1: " << res1 << endl;
  cout << "Битовое поле res2: " << res2 << endl;
  cout << "Битовое поле res3: " << res3 << endl;

  return 0;
}
```