

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский
Нижегородский государственный университет им. Н.И. Лобачевского»
(ННГУ)

Институт информационных технологий, математики и механики

ЛАБОРАТОРНАЯ РАБОТА

на тему:

«Битовые поля и множества»

Выполнил(а): студент(ка) группы _____
_____ / Фатехов К.Г./
Подпись

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

Нижний Новгород
2023

Содержание

Введение.....	3
1 Постановка задачи.....	4
2 Руководство пользователя.....	5
2.1 Приложение для демонстрации работы битовых полей	5
2.2 Приложение для демонстрации работы множеств	6
2.3 «Решето Эратосфена»	7
3 Руководство программиста	9
3.1 Описание алгоритмов	9
3.1.1 Битовые поля	9
3.1.2 Множества	11
3.1.3 «Решето Эратосфена»	13
3.2 Описание программной реализации	14
3.2.1 Описание класса TBitField	14
3.2.2 Описание класса TSet	17
Заключение	22
Литература	23
Приложения	24
Приложение А. Реализация класса TBitField	24
Приложение Б. Реализация класса TSet.....	24

Введение

Битовые поля актуальны в программировании, так как позволяют экономить память и оптимизировать работу с большим количеством флагов или настроек. Они позволяют использовать одну переменную для хранения нескольких булевых значений или числовых флагов, что уменьшает размер данных и упрощает их обработку.

Битовые поля особенно полезны при работе с микроконтроллерами или встроенными системами, где доступная память и ресурсы ограничены. Они позволяют эффективно использовать ограниченное количество битов памяти и сократить затраты на хранение данных.

Кроме того, битовые поля могут использоваться для создания протоколов связи или форматов данных, где каждый бит представляет определенное значение или состояние. Это позволяет упростить передачу и интерпретацию данных между различными системами.

В целом, использование битовых полей в программировании позволяет сократить затраты на память и улучшить производительность, особенно в случаях, когда необходимо работать с большим количеством флагов или настроек.

1 Постановка задачи

Цель: Разработать битовое множество на языке программирования C++. Битовое множество включает в себя 2 класса «TBitField» и «TBitSet». В этих классах нужно сделать реализацию основных операций с битовыми полями и множествами.

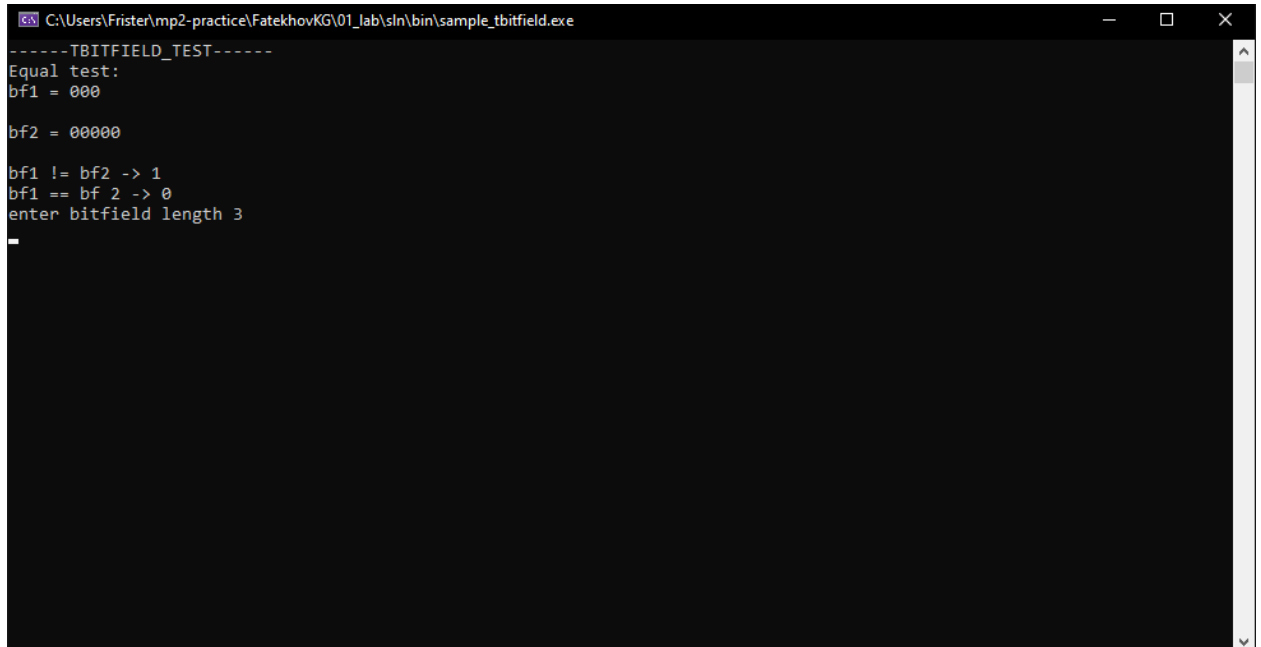
2. Задачи данной лабораторной работы:

- Разработка класса TBitField и реализация основных операций с битовыми полями в этом классе.
- Разработка класса TSet и реализация основных операций для работы с множествами в этом классе.
- Проверка и демонстрация работы разработанных классов с помощью приложений для работы с битовыми полями, множествами и решением задачи "Решето Эратосфена".
- Написание отчета о выполненной лабораторной работе.

2 Руководство пользователя

2.1 Приложение для демонстрации работы битовых полей

1. Запустим приложение с названием sample_tbitfield.exe. В результате появится следующее окно



```
C:\Users\Frister\mp2-practice\FatekhovKG\01_lab\sln\bin\sample_tbitfield.exe
-----TBITFIELD_TEST-----
Equal test:
bf1 = 000

bf2 = 00000

bf1 != bf2 -> 1
bf1 == bf 2 -> 0
enter bitfield length 3
_
```

Рис. 1. Основное окно программы

2. Это окно показывает работу основных функций работы с битовыми полями (сравнение на равенство, присваивание, длина, установка/очистка бита, сложение, пересечение, отрицание). Предлагается ввести значение битового поля длины 3. В результате будет выведено (рис. 2).
- 3.

```
C:\Users\Frister\mp2-practice\FatekhovKG\01_lab\sln\bin\sample_tbitfield.exe
-----TBITFIELD_TEST-----
Equal test:
bf1 = 000

bf2 = 00000

bf1 != bf2 -> 1
bf1 == bf 2 -> 0
enter bitfield length 3
1
1
0
Length of bf3 = 3
SetBit(2) for bf3: 111

ClrBit(2) for bf3: 110

GetBit(2) for bf3: 0
bf2 = bf3 -> 110

bf2 = bf3 | bf1 -> 110

bf2 = bf3 & bf1 -> 000

bf2 = ~bf1 -> 111

Для продолжения нажмите любую клавишу . . .
```

Рис. 2. Основное окно программы

2.2 Приложение для демонстрации работы множеств

1. Запустим приложение с названием sample_tset.exe. В результате появится следующее окно

```
C:\Users\Frister\mp2-practice\FatekhovKG\01_lab\sln\bin\sample_tset.exe
-----TSET_TEST-----
enter set length 6
Input your set (To finish, enter -1)
-
```

Рис. 3. Основное окно программы

2. Предлагается ввести значение множества длины 6. Будет выведено следующее (рис. 4). Это окно показывает работу основных функций работы с множествами (сравнение на равенство, добавление элемента, длина, удаление элемента, сложение, пересечение, отрицание).



```
-----TSET_TEST-----
enter set length 6
Input your set (To finish, enter -1)
0
1
3
-1
set1 = 100001
set2 = 110100

Equal test:
set1 != set2 -> 1
set1 == set2 -> 0
set3(s1).inselem(2) -> 101001

set3.maxpower -> 6
set4 = set3 -> 101001

set3.delelem(3)-> 100001

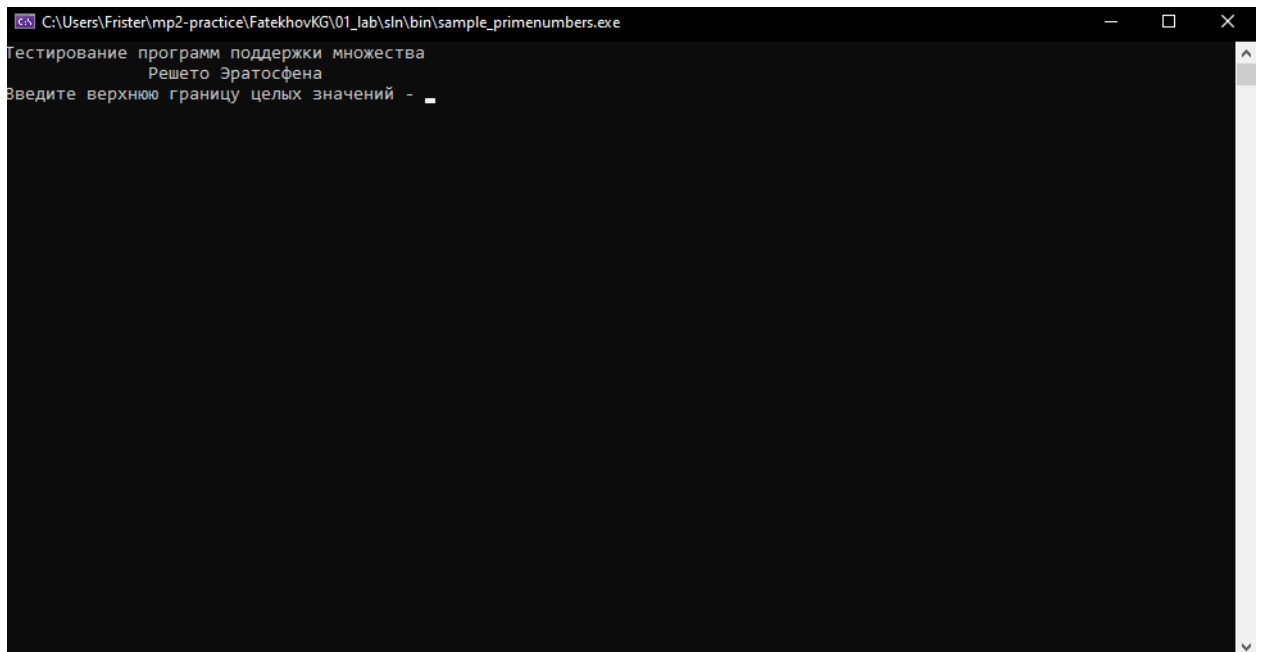
set3 = set4 + set2 -> 111101
set2 = set2 * set4 -> 100000
set1 = ~set3 -> 000010

Для продолжения нажмите любую клавишу . . .
```

Рис. 4. Основное окно программы

2.3 «Решето Эратосфена»

4. Запустим приложение с названием sample_tbitfield.exe. В результате появится окно, показанное ниже (рис. 5).



```
Тестирование программ поддержки множества
Решето Эратосфена
Введите верхнюю границу целых значений - _
```

Рис. 5. Основное окно программы

5. Предлагается ввести верхнюю границу целых значений для поиска простых чисел. В результате выведено множество в битовом представлении. Ниже выведены простые числа, входящие в множества с выбранной верхней границей.

```
C:\Users\Frister\mp2-practice\FatekhovKG\01_lab\sln\bin\sample_primenumbers.exe
Тестирование программ поддержки множества
Решето Эратосфена
Введите верхнюю границу целых значений - 75

Печать множества некрatных чисел
0011010100010100010100010000010100000100010100010000010000010100000100010100

Печать простых чисел
 2  3  5  7 11 13 17 19 23 29
31 37 41 43 47 53 59 61 67 71
73
В первых 75 числах 21 простых
Для продолжения нажмите любую клавишу . . .
```

Рис. 6. Основное окно программы

3 Руководство программиста

3.1 Описание алгоритмов

3.1.1 Битовые поля

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

Битовые поля предоставляют компактное представление данных, которое экономит память и ускоряет выполнение операций. Они обладают следующими преимуществами:

- 1) Экономия памяти
- 2) Ускорение выполнения операций
- 3) Удобство чтения и записи
- 4) Ясность кода
- 5) Повышение производительности

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

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

Операция объединения:

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

Пример:

0 1 1 1 1 1

1 0 0 0 1 1

1 1 1 1 1 1

Операция пересечения:

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

Пример:

0 1 1 1 1 0

1 0 0 1 0 1

0 0 0 1 0 0

Операция дополнения (отрицания):

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

Пример:

0 1 1 0 1 1

1 0 0 1 0 0

Операция ввода битового поля из консоли:

Операция позволяет ввести битовое поле из консоли. Для этого необходимо в консоль вводить 1 или 0 до тех пор, пока длина битового поля не будет равна установленной длине.

Пример:

0 0 0 0 – битовое поле длины 4

После ввода:

0 1 1 0

Операция вывода битового поля в консоль:

Операция вывода позволяет вывести битовое поле в консоль

Также в битовом поле можно установить или отчистить бит

Пример:

0 1 1 0 1 1

Результат установление 0 бита

1 1 1 0 1 1

Пример:

0 1 1 0 1 1

Результат удаления 2 бита:

0 1 0 0 1 1

Замечание: Введенный бит должен быть больше либо равен 0 и меньше длины битового поля

Операции сравнения

Операция равенства выведет 1, если два битовых поля равны 0 в противном случае. Операция, обратная операции равенства, выведет 0, если битовые поля равны, 1 в противном случае.

3.1.2 Множества

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

Множества представляют собой набор целых положительных чисел. В данной лабораторной работе множество реализовано при помощи битового поля, соответственно каждый бит которых интерпретируется элементом, равным индексом бита.

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

Пример множества максимальной длины 5: A: {0, 1, 2, 3, 4}

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

Операция объединения с множеством:

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

Пример:

A = {0,1,3}

B = {1,3,4}

Результат объединения множеств A|B:

$$A+B = \{0, 1, 3, 4\}$$

Операция пересечения с множеством:

Операция возвращает экземпляр класса, содержащий все уникальные элементы, которые есть во всех множествах.

Пример:

$$A = \{1, 2, 3\}$$

$$B = \{1, 3, 5\}$$

Результат пересечения множеств $A*B$:

$$A*B = \{3\}$$

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

Операция дополнения (отрицания):

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

Пример:

$$A = \{1, 2\}$$

Результат объединения множеств $\sim A$:

$$\sim A = \{3, 4, 5\}$$

Операция ввода множества из консоли:

Операция позволяет ввести множество из консоли. Для этого необходимо вводить элементы в консоль, но нельзя вводить элементы больше максимального элемента множества.

Пример:

$$A = \{\}$$

После ввода:

$$A = \{0, 2\}$$

Операция вывода битового поля в консоль:

Операция вывода позволяет вывести множество в консоль

Также в множестве можно добавить или убрать элемент.

Пример:

$A = \{1, 2, 3\}$

Результат добавления 4:

$A = \{1, 2, 3, 4\}$

Пример:

$A = \{0, 2, 3\}$

Результат удаления 0:

$A = \{2, 3\}$

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

3.1.3 «Решето Эратосфена»

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

Данный алгоритм реализован двумя способами, при помощи классов TSet.

Сначала необходимо ввести целое положительное число, которое будет верхней границей. После чего алгоритм заполняет все элементы классов равными 1.

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

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

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

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

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

3.2 Описание программной реализации

3.2.1 Описание класса TBitField

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

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

    // доступ к битам
    int GetLength(void) const;
    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;
    TBitField& operator=(const TBitField &bf);
    TBitField operator|(const TBitField &bf);
    TBitField operator&(const TBitField &bf);
    TBitField operator~(void);

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

Назначение: представление битового поля.

Поля:

BitLen — длина битового поля — максимальное количество битов.

pMem — память для представления битового поля.

MemLen — количество элементов для представления битового поля.

Методы:

```
int GetMemIndex(const int n) const;
```

Назначение: получение индекса элемента в памяти.

Входные параметры:

n - номер бита.

Выходные параметры:

Номер элемента в памяти.

TELEM GetMemMask (const int n) const;

Назначение: получение маски по индексу.

Входные параметры:

n - номер бита.

Выходные параметры:

Маска.

TBitField(int len);

Назначение: конструктор с параметром.

Входные параметры:

len – длина поля.

Выходные параметры:

Битовое поле.

TBitField(const TBitField &bf);

Назначение: конструктор копирования.

Входные параметры:

bf – ссылка на копируемое поле.

Выходные параметры:

Битовое поле.

void SetBit(const int n);

Назначение: установка бита.

Входные параметры:

n – индекс бита.

void ClrBit(const int n);

Назначение: очистка бита.

Входные параметры:

n – индекс бита.

int GetBit(const int n) const

Назначение: получение значения бита.

Входные параметры:

n – индекс бита.

Выходные параметры:

Значение бита.

int operator==(const TBitField &bf) const

Назначение: операция сравнения на равенство.

Входные параметры:

bf – ссылка на константное битовое поле

Выходные параметры:

Результат сравнения

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

Назначение: операция сравнения на неравенство.

Входные параметры:

bf – ссылка на константное битовое поле

Выходные параметры:

Результат сравнения

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

Назначение: операция присваивания.

Входные параметры:

bf – ссылка на константное битовое поле

Выходные параметры:

Ссылка на битовое поле.

TBitField operator|(const TBitField &bf);

Назначение: операция “или”.

Входные параметры:

bf – ссылка на константное битовое поле

Выходные параметры:

Битовое поле.

TBitField operator&(const TBitField &bf);

Назначение: операция “и”.

Входные параметры:

bf – ссылка на константное битовое поле

Выходные параметры:

Битовое поле.

TBitField operator~(void);

Назначение: операция отрицания.

Выходные параметры:

Битовое поле.

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

Назначение: перегрузка потокового ввода.

Выходные параметры:

istr – ссылка на поток ввода.

bf – ссылка на константное битовое поле

Выходные параметры:

Ссылка на поток ввода

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

Назначение: перегрузка потокового вывода.

Входные параметры:

ostr – ссылка на поток вывода.

bf – ссылка на константное битовое поле

Выходные параметры:

Ссылка на поток вывода.

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;
    TSet& operator=(const TSet &s);
    TSet operator+ (const int Elem);
```

```

TSet operator- (const int Elem);

TSet operator+ (const TSet &s);
TSet operator* (const TSet &s);
TSet operator~ (void);

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

```

Назначение: представление множества.

Поля:

MaxPower – длина множества.

BitField – битовое поле.

Методы:

```
TSet(int mp) ;
```

Назначение: конструктор с параметром.

Входные параметры:

mp – длина множества.

Выходные данные:

Множество.

```
TSet(const TSet &s) ;
```

Назначение: конструктор копирования.

Входные параметры:

s – ссылка на константное множество.

Выходные параметры:

Множество.

```
TSet(const TBitField &bf) ;
```

Назначение: конструктор преобразования типа.

Входные параметры:

bf – ссылка на константное битовое поле

Выходные данные:

Множество.

```
operator TBitField() ;
```

Назначение: преобразование типа к битовому полю.

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

Назначение: получение мощности множества.

Выходные параметры:

Мощность множества.

```
void InsElem(const int Elem);
```

Назначение: включение элемента в множество.

Входные параметры:

Elem – новый элемент множества.

```
void DelElem(const int Elem);
```

Назначение: удаление элемента из множества.

Входные параметры:

Elem – элемент множества.

```
int IsMember(const int Elem) const;
```

Назначение: проверка наличия элемента в множестве.

Входные параметры:

Elem – элемент множества.

Выходные параметры:

Результат проверки

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

Назначение: операция сравнения на равенство.

Входные элементы:

s – ссылка на константное множество.

Выходные параметры:

Результат сравнения.

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

Назначение: операция сравнения на неравенство.

Входные элементы:

s – ссылка на константное множество.

Выходные параметры:

Результат сравнения.

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

Назначение: операция присваивания.

Входные параметры:

s – ссылка на константное множество.

Выходные параметры:

Ссылка на множество.

TSet operator+ (const int Elem);

Назначение: операция объединения с элементом.

Входные параметры:

Elem – новый элемент множества.

Выходные параметры:

Множество.

TSet operator- (const int Elem);

Назначение: операция разницы с элементом.

Входные параметры:

Elem – элемент множества.

Выходные параметры:

Множество.

TSet operator+ (const TSet &s);

Назначение: операция объединения множеств.

Входные параметры:

s – ссылка на константное множество.

Выходные параметры:

Множество.

TSet operator* (const TSet &s);

Назначение: операция пересечения множеств.

Входные параметры:

s – ссылка на константное множество.

Выходные параметры:

Множество.

TSet operator~ (void);

Назначение: операция дополнения.

Выходные параметры:

Множество.

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

Назначение: перегрузка операции потокового ввода.

Входные параметры:

istr – ссылка на поток ввода.

bf – ссылка на константное множество.

Выходные параметры:

Ссылка на поток ввода.

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

Назначение: перегрузка операции потокового вывода.

Входные параметры:

ostr – ссылка на поток вывода.

bf – ссылка на константное множество.

Выходные параметры:

Ссылка на поток вывода.

Заключение

В результате выполнения работы «Битовые поля и множества» были изучены и продемонстрированы на практике принципы работы битовых полей и множеств. Удалось разработать классы с операциями над битовыми полями и множествами. С помощью тестов мы увидели что битовые поля и множества очень эффективны, так как они позволяют ускорить операции над множествами. На основе результатов тестов, можно полагать, что битовые поля и множества очень полезны. Они сильно облегчают работу с большими объёмами данных

Литература

1. . Сысоев А.В., Алгоритмы и структуры данных, лекция 03, 19 сентября.
2. Информация из нейронных сетей, в частности «CHAT GPT»

Приложения

Приложение А. Реализация класса TBitField

```
TBitField::TBitField(int len)
{
    if (len < 0) throw "length can't be negative";
    BitLen = len;
    MemLen = BitLen / (sizeof(TELEM) * 8) + 1;
    pMem = new TELEM[MemLen];
    for (int i = 0; i < MemLen; i++) pMem[i] = 0;
}

TBitField::TBitField(const TBitField& bf) {
    BitLen = bf.BitLen;
    MemLen = bf.MemLen;
    pMem = new TELEM[MemLen];
    for (int i = 0; i < MemLen; i++) pMem[i] = bf.pMem[i];
}

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

int TBitField::GetMemIndex(const int n) const {
    return (n / (sizeof(TELEM) * 8));
}

TELEM TBitField::GetMemMask(const int n) const {
    if ((n > BitLen) || (n < 0)) throw "Negative n";
    return 1 << (n & (sizeof(TELEM) * 8 - 1));
}

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

void TBitField::SetBit(const int n) {
    if ((n > BitLen) || (n < 0)) throw "Negative n";
    pMem[GetMemIndex(n)] = (GetMemMask(n) | pMem[GetMemIndex(n)]);
}

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

int TBitField::GetBit(const int n) const {
    if ((n > BitLen) || (n < 0)) throw "Negative n";
    if (pMem[GetMemIndex(n)] & GetMemMask(n)) return 1; else return 0;
}

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

    for (int i = 0; i < bf.BitLen; i++)
        if (bf.GetBit(i) == 0)
            ostr << 0;
        else ostr << 1;
    ostr << endl;
    return ostr;
}

TBitField& TBitField::operator=(const TBitField& bf) {
    BitLen = bf.BitLen;
    MemLen = bf.MemLen;
    delete[] pMem;
    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 // сравнение
{
    if (BitLen != bf.BitLen)
        return 0;
    int k = 0;
    for (int i = 0; i < MemLen; i++)
        if (pMem[i] != bf.pMem[i]) {
            return 0;
        }
    return 1;
}

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

TBitField TBitField::operator|(const TBitField& bf) // операция "или"
{
    int k;
    int z;
    int y;
    if (BitLen > bf.BitLen)
    {
        y = 0; // This is big
        k = BitLen;
        z = bf.BitLen;
    }
    else {
        y = 1; //bf is big
        k = bf.BitLen;
        z = BitLen;
    }
    TBitField a(k);
    for (int i = 0; i <= GetMemIndex(z); i++) a.pMem[i] = bf.pMem[i] | pMem[i];
    for (int i = (GetMemIndex(z) + 1); i < a.MemLen; i++) if (y == 1) a.pMem[i] = bf.pMem[i]; else a.pMem[i] =
pMem[i];
    return a;
}

TBitField TBitField::operator&(const TBitField& bf) // операция "и"
{
    int k;
    int z;
    if (BitLen > bf.BitLen) {
        k = BitLen;
        z = bf.BitLen;
    }

    else
    {
        k = bf.BitLen;
        z = BitLen;
    }

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

TBitField TBitField::operator~(void) // отрицание
{

```

```

        TBitField a(*this);
        for (int i = 0; i < (a.MemLen - 1); i++) a.pMem[i] = ~(a.pMem[i]);
        for (int i = ((a.MemLen - 1) * (sizeof(TELEM) * 8)); i < (a.BitLen); i++) {
            if (a.GetBit(i) == 1) a.ClrBit(i);
            else a.SetBit(i);
        }
        return a;
    }

// ВВОД/ВЫВОД

istream& operator>>(istream& istr, TBitField& bf) // ввод
{
    int tmp;
    for (int i = 0; i < bf.BitLen; i++) {
        istr >> tmp;
        if ((tmp != 0) && (tmp != 1)) {
            throw "The bit cannot take such a value";
        }

        if (tmp == 0) {
            bf.ClrBit(i);
        }
        else {
            bf.SetBit(i);
        }
    }
    return istr;
}

```

Приложение Б. Реализация класса TSet

```

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

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

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

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

int TSet::GetMaxPower(void) const // получить макс. к-во эл-тов
{
    return MaxPower;
}

int TSet::IsMember(const int Elem) const // элемент множества?
{
    if ((Elem < MaxPower) & (Elem >= 0))
        if (BitField.GetBit(Elem) == 1) return 1;
        else return 0; else throw "Negative Elem";
}

```

```

void TSet::InsElem(const int Elem) // включение элемента множества
{
    if ((Elem < MaxPower) & (Elem >= 0))
        BitField.SetBit(Elem); else throw "Negative n";
}

void TSet::DelElem(const int Elem) // исключение элемента множества
{
    if ((Elem < MaxPower) & (Elem >= 0))
        BitField.ClrBit(Elem); else throw "Negative n";
}

// теоретико-множественные операции

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 !(*this == s);
}

TSet TSet::operator+(const TSet& s) // объединение
{
    TSet a(BitField | s.BitField);
    return a;
}

TSet TSet::operator+(const int Elem) // объединение с элементом
{
    TSet a(*this);
    if ((Elem < MaxPower) & (Elem >= 0)) {
        a.InsElem(Elem);
    }
    else throw "Negative Elem";
    return a;
}

TSet TSet::operator-(const int Elem) // разность с элементом
{
    TSet a(*this);
    if ((Elem < MaxPower) & (Elem >= 0)) {
        a.DelElem(Elem);
    }
    else throw "Negative Elem";
    return a;
}

TSet TSet::operator*(const TSet& s) // пересечение
{
    TSet a(BitField & s.BitField);
    return a;
}

TSet TSet::operator~(void) // дополнение

```

```

{
    TSet a(~BitField);
    return a;
}

// перегрузка ввода/вывода

istream& operator>>(istream& istr, TSet& s) // ввод
{
    for (int i = 0; i < s.MaxPower; i++) {
        s.DelElem(i);
    }
    cout << "Input your set (To finish, enter -1)" << endl;
    int i;

    while (1) {
        istr >> i;
        if (i == -1) {
            return istr;
        }
        if ((i < 0) || (i > s.MaxPower)) {
            throw "OUTOFRANGE";
        }
        s.InsElem(i);
    }
    return istr;
}

ostream& operator<<(ostream& ostr, const TSet& s) // вывод
{
    cout << s.BitField;
    return ostr;
}

```