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:

**«Матрица на основе вектора»**

**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](#_Toc149043756)

[1 Постановка задачи 4](#_Toc149043757)

[2 Руководство пользователя 5](#_Toc149043758)

[2.1 Приложение для демонстрации работы векторов 5](#_Toc149043759)

[2.2 Приложение для демонстрации работы матриц 6](#_Toc149043760)

[3 Руководство программиста 9](#_Toc149043762)

[3.1 Описание алгоритмов 9](#_Toc149043763)

[3.1.1 Векторы 9](#_Toc149043764)

[3.1.2 Матрицы 11](#_Toc149043765)

[3.2 Описание программной реализации 14](#_Toc149043767)

[3.2.1 Описание класса TBitField 14](#_Toc149043768)

[3.2.2 Описание класса TSet 17](#_Toc149043769)

[Заключение 22](#_Toc149043770)

[Литература 23](#_Toc149043771)

[Приложения 24](#_Toc149043772)

[Приложение А. Реализация класса TBitField 24](#_Toc149043773)

[Приложение Б. Реализация класса TSet 26](#_Toc149043774)

# Введение

Программа "Матрица на основе шаблонных векторов" является мощным инструментом для работы с математическими матрицами. Она основана на использовании шаблонных векторов, что позволяет достичь гибкости и универсальности в работе с различными типами данных.

Данная программа предоставляет возможность создания и манипулирования матрицами произвольного размера. Она позволяет выполнить такие операции, как создание пустой матрицы, заполнение ее элементов, выполнение арифметических операций с матрицами (сложение, вычитание, умножение).

Преимущество программы "Матрица на основе шаблонных векторов" состоит в том, что она способна работать с различными числовыми типами данных (целочисленными, вещественными и т. д.), а также с пользовательскими типами данных, благодаря применению шаблонных векторов. Это позволяет легко вписаться в широкий спектр задач, связанных с матричными вычислениями.

Кроме того, программа обладает простым и интуитивно понятным интерфейсом, что делает ее использование легким даже для начинающих пользователей. Она также имеет высокую производительность и оптимизирована для работы с большими объемами данных.

# Постановка задачи

Цель: Разработать матрицу на основе шаблонного вектора на языке программирования C++. Матрица включает в себя 2 класса «TVector» и «TMatrix» В этих классах нужно сделать реализацию основных операций с векторами и матрицами.

2. Задачи данной лабораторной работы:

- Разработка класса TVector и реализация основных операций для работы с ними.

- Разработка класса TMatrix и реализация основных операций для работы с ними.

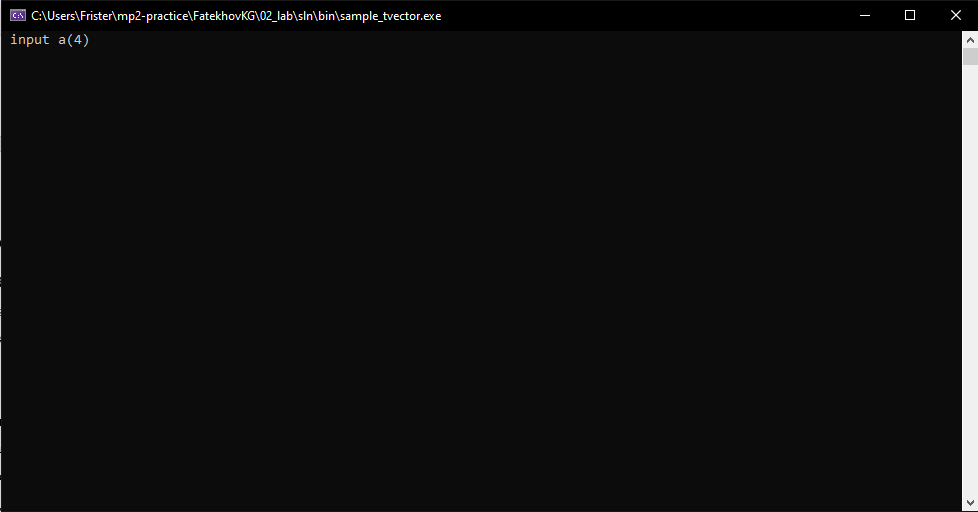
- Проверка и демонстрация работы разработанных классов с помощью приложений для работы с матрицами и векторами.

- Написание отчета о выполненной лабораторной работе.

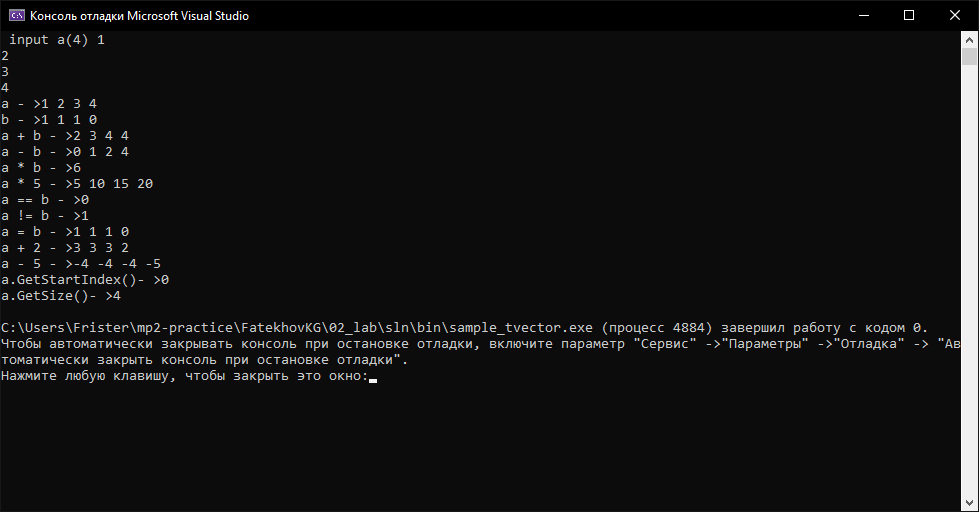
# Руководство пользователя

## Приложение для демонстрации работы векторов

1. Запустим приложение с названием sample\_tvector.exe. В результате появится следующее окно



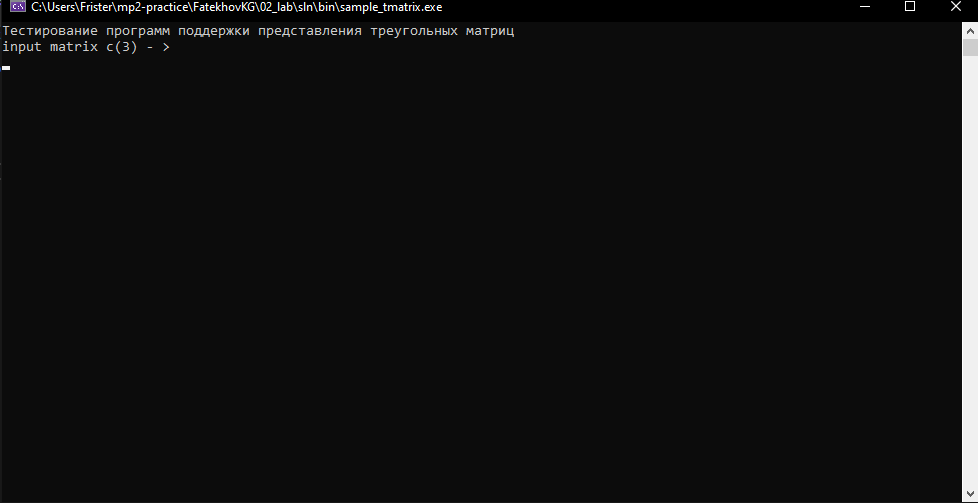
1. Основное окно программы
2. Это окно показывает работу основных функций работы с векторами (сравнение на равенство, присваивание, длина, сложение, скалярное умножение). Предлагается ввести вектор длины 4. В результате будет выведено (рис. 2).
3. Окно демонстрирует все операции связанные с векторами, описанные в данной программе.



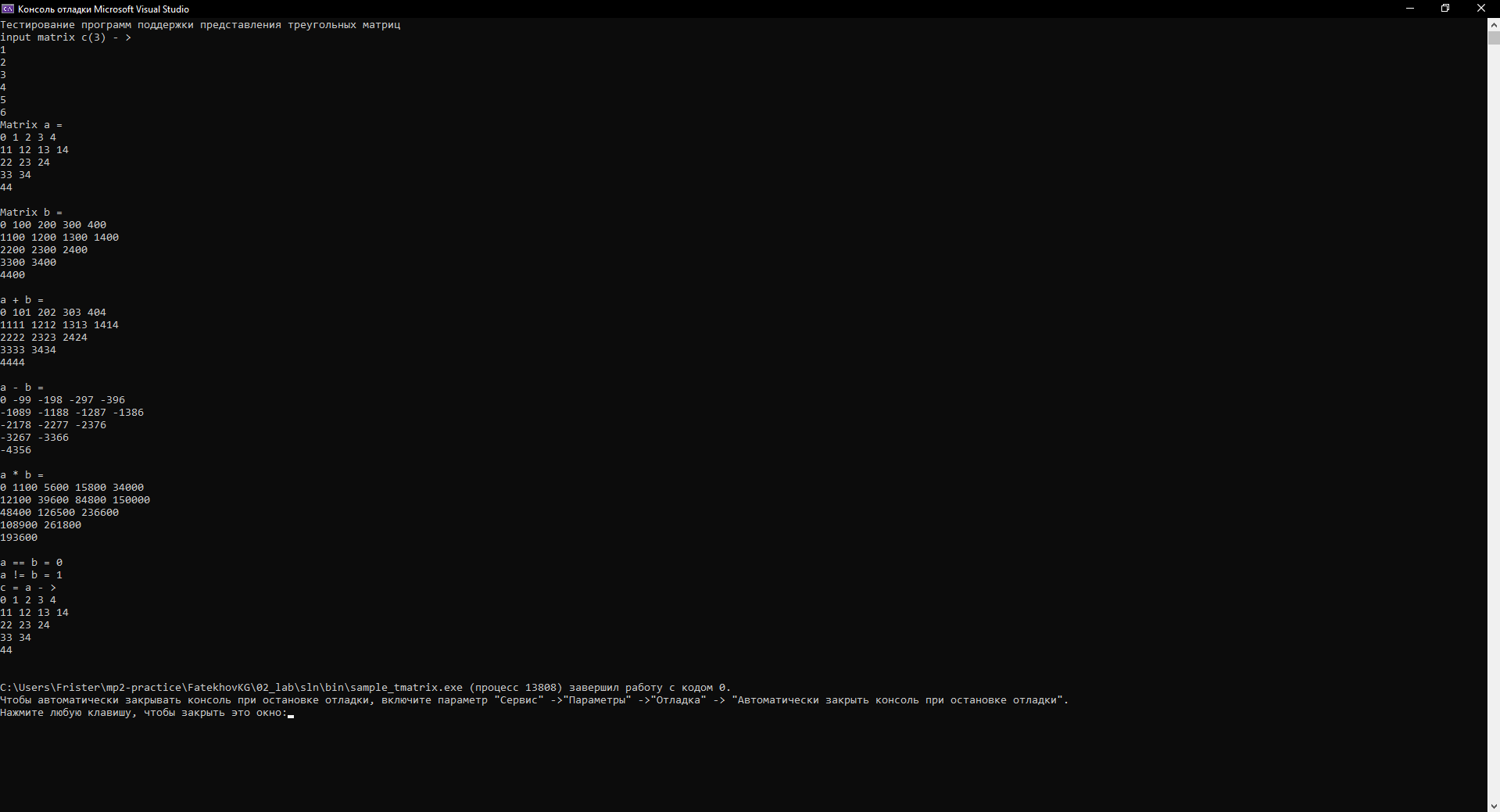
1. Основное окно программы

## Приложение для демонстрации работы матриц

1. Запустим приложение с названием sample\_tmatrix.exe. В результате появится следующее окно



1. Основное окно программы
2. Предлагается ввести матрицу размера 3. Будет выведено следующее (рис. 4). Это окно показывает работу основных функций работы с матрицами (умножение и сложение матриц).



1. Основное окно программы

# Руководство программиста

## Описание алгоритмов

### Векторы

1. Начало работы указано в пункте [2.1](#_Приложение_для_демонстрации_1)
2. Описание методов и полей класса указано в пункте [3.2.1](#_Описание_класса_TBitField)

Векторы в программировании представляют собой упорядоченные наборы элементов одного типа, которые могут изменяться в размере. Они представляют собой динамические структуры данных и часто используются для хранения и манипуляции коллекциями элементов. Они обладают следующими преимуществами:  
1. Динамический размер: Векторы позволяют добавлять и удалять элементы, что позволяет динамически изменять их размер, в отличие от статических массивов.

2. Быстрый доступ к элементам: Векторы обеспечивают быстрый доступ к элементам за счет использования индексов. Это позволяет быстро получать доступ к любому элементу вектора.

3. Поддержка итераций: Векторы позволяют легко выполнять итерации по элементам с помощью циклов, что делает их удобными для обработки больших объемов данных.

4. Универсальность: Векторы поддерживают элементы любого типа данных, что делает их полезными для хранения и обработки различных структур данных.

Операция GetSize():

Операция возвращает размер вектора.

Пример:

1 2 3

3

Операция GetStartIndex():

Операция возвращает индекс первого элемента вектора.

Пример:

Для v(3,1)

1

Операция индексирования:

Операция возвращает элемент вектора стоящий в указанной позиции.

Пример:

1 2 3

[2]

3

Операция ввода вектора из консоли:

Операция позволяет ввести вектор с консоли. Вводим элементы вектора, пока не достигнем установленной длины.

Пример:

0 0 0 0 – вектор длины 4

После ввода:

2 3 3 3

Операция вывода вектора в консоль:

Операция вывода позволяет вывести вектор в консоль

Операция сравнения:

Операция возвращает 0, когда вектора не равны и 1 в противном случае

Пример:

1 1 1

0 0 0

0

Скалярные операции:

Операции возвращают экземпляр вектора, все элементы которого изменились на указанное число.

Пример:

1 2 3

+ 3

4 5 6

Операция прибавление вектора:

Операция возвращает экземпляр класса, результат сложения двух векторов.

Пример:

1 1 1

2 2 2

3 3 3

Операция умножения векторов:

Операция возвращает число, результат умножения двух векторов скалярно.

Пример:

1 1 1

2 2 2

6

### Матрицы

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

Матрица представляет собой двумерный массив, состоящий из элементов одного типа данных. Она представляет собой таблицу, где каждый элемент находится в определенной строке и столбце. Матрицы используются для хранения и обработки данных, таких как числа, символы, строки и т. д.

Матрица реализована на основе шаблонного класса вектор.

Матрица поддерживает наследуемые поля вектора, а также его методы.

Операция сложения матриц:

Операция возвращает экземпляр класса, результат сложения матриц.

Пример:

A = 0 1 2 3 4

11 12 13 14

22 23 24

33 34

44

B = 0 100 200 300 400

1100 1200 1300 1400

2200 2300 2400

3300 3400

4400

Результат сложения матриц A и B:

A+B = 0 101 202 303 404

1111 1212 1313 1414

2222 2323 2424

3333 3434

4444

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 ~A:

~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

### «Решето Эратосфена»

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

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.

## Описание программной реализации

### 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 вводы

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.

### 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;

}