Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский Нижегородский государственный университет им. Н.И.Лобачевского»

Институт Информационных технологий, математики и механики

Фундаментальная информатика и информационные технологии

Отчет по лабораторной работе №1

**Работа с битовыми полями и множествами**

Выполнил:

Студент гр. 3821Б1ФИ3

Косарев Е.А.

Проверил:

заведующий лабораторией суперкомпьютерных технологий и высокопроизводительных вычислений

Лебедев И.Г.

Нижний Новгород

2022

Содержание

[Введение 3](#_Toc85632832)

[Постановка задачи 4](#_Toc85632833)

[Руководство пользователя 5](#_Toc85632834)

[Руководство программиста 18](#_Toc85632835)

[Описание структуры кода программ 18](#_Toc85632836)

[Описание класса TBitField 18](#_Toc85632837)

[Описание класса TSet 19](#_Toc85632838)

[Заключение 20](#_Toc85632840)

Приложение 1…………………………………………………...………………………21

Приложение 2………………………………………………………….………………33

# **Введение**

Битовое поле – набор нулей и единиц заданного размера. Основная идея использования битового поля заключается в экономии памяти. В качестве битового поля мы можем использовать массив. В частности, рассматривая массив int, экономия памяти составляет в 32 раза. Работая с битовым полем, безусловно, нам нужно знать какие существуют побитовые операции и что это такое. Побитовые операции — операции, производимые над цепочками битов. Выделяют два типа побитовых операций: логические операции и побитовые сдвиги. К логическим побитовым операциям относятся: логическое и, логическое или, отрицание, исключающее или (в данной работе не представлена). Операторы сдвига  <<  и  >>  сдвигают биты в переменной влево или вправо на указанное число. Сдвиг влево равносилен умножению числа на два в какой-то степени, сдвиг вправо равносилен делению на два в какой-то степени. Наличие таких операций позволяет работать с битовыми полями. Также следует дать определение такому термину, как битовая маска. Битовая маска – набор нулей и одной единственной единицы. Битовое поле представляет собой отдельный инструментарий, с помощью которого можно решать некие задачи. Например, сортировка массива без использования сравнений, сохранение информации о состоянии объектов, реализация множества и другие.

Множество – набор различных элементов. Они могут быть следующих видов: пустое, одноэлементное, универсальное. Пустое множество – множество, не содержащее ни одного элемента. Одноэлементное множество – множество, состоящее только из одного элемента. Универсально множество – множество, содержащее все мыслимые объекты. Одной из определяющих характеристик множества является его мощность. Мощность – количество элементов множества. Над множествами мы можем проводить следующие действия: добавление/удаление элемента, объединение множеств, пересечение множеств, дополнение множества, вычисление мощности множества, сравнение множеств, проверить является ли какой-либо элемент элементом некоторого множества. Возникает вопрос: «А как нам могут помочь битовые поля в реализации класса «Множество»?». С помощью битового поля мы можем представить какие элементы входят в множество. Так как в множестве все элементы различны, то каждый из них имеет свой уникальный номер. Тем самым в битовом поле мы можем хранить информацию о содержащихся в множестве элементах. Если бит равен 1, то это говорит о том, что элемент принадлежит данному множеству, в противном случае не принадлежит.

# **Постановка задачи**

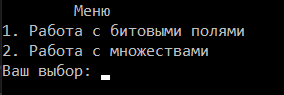
**Цель данной работы** — разработка структуры данных для хранения множеств с использованием битовых полей, а также освоение таких инструментов разработки программного обеспечения, как система контроля версий [Git](https://git-scm.com/book/ru/v2) и фрэймворк для разработки автоматических тестов [Google Test](https://github.com/google/googletest).

Выполнение работы предполагает решение следующих задач:

1. Реализация класса битового поля TBitField согласно заданному интерфейсу.
2. Реализация класса множества TSet согласно заданному интерфейсу.
3. Обеспечение работоспособности тестов и примера использования.
4. Реализация нескольких простых тестов на базе Google Test.
5. Публикация исходных кодов в личном репозитории на GitHub.

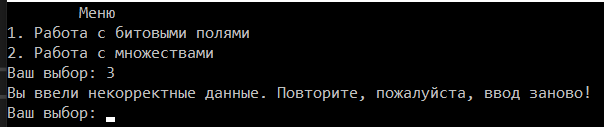
# Руководство пользователя

В данной программе реализована работа с битовыми полями и с множеством. Соответственно, при запуске программы появится меню, пользователю необходимо выбрать с чем он хочет работать: с битовыми полями или с множествами (рис.1).



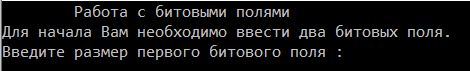
*Рис.1. Главное меню.*

Пользователь может выбрать либо первый пункт, либо второй. В случае выбора несуществующего пункта, необходимо будет повторить ввод (рис.2).



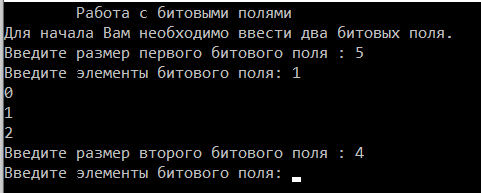
*Рис.2. Выбор несуществующего пункта меню.*

После выбора первого пункта появится меню работы с битовыми полями(рис.3).



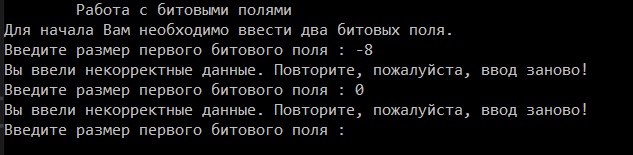
*Рис.3. Меню работы с битовыми полями.*

Для начала пользователю необходимо ввести два битовых поля. Нужно ввести размеры битовых полей и элементы. Элементами битового поля могут быть только 0 или 1. При вводе элемента, отличного от 0 или 1, ввод элементов поля прекращается, и по умолчанию незаполненные элементы становятся равными 0. Ввод второго битового поля осуществляется аналогично вводу первому битовому полю. Пример работы программы приведен на рисунке 4.



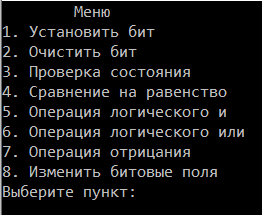
*Рис.4. Работа с битовыми полями. Ввод битовых полей.*

Так как размер битового поля должен быть больше нуля, то при вводе отрицательного значения или равного нулю, пользователь получит сообщение о том, что данные введены некорректно и нужно ввести их заново (рис.5).



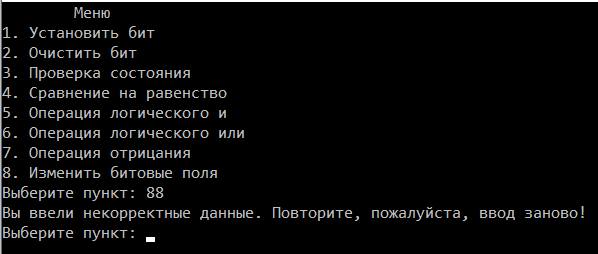
*Рис.5. Работа с битовыми полями. Некорректный ввод данных.*

После корректного ввода битовых полей появится следующее меню, в котором представлены всевозможные операции, которые можно проводить с битовыми полями (рис.6).



*Рис.6. Работа с битовыми полями. Операции над битовыми полями.*

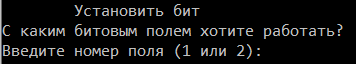
Как и в предыдущих меню, при выборе несуществующего пункта, пользователь получит сообщение о том, что данные введены некорректно и нужно будет выбрать пункт заново.



*Рис.7. Работа с битовыми полями. Некорректный ввод пункта меню.*

**Пункт – 1. Установить бит.**

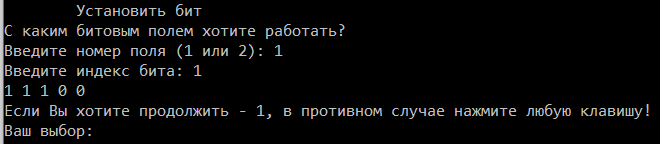
Сначала пользователю необходимо выбрать битовое поле, с которым он будет работать (рис.8).



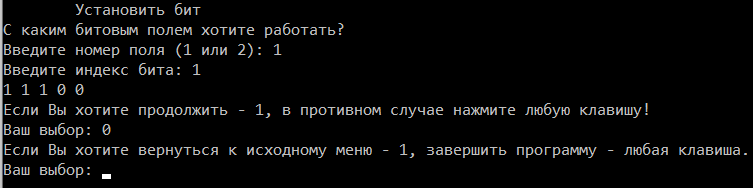
*Рис.8. Работа с битовыми полями. Установить бит. Выбор битового поля.*

При выборе несуществующего поля, появится сообщение о том, что ввод некорректен, снова нужно будет повторить ввод. Аналогично будет происходить и в других пунктах.

После выбора пользователю необходимо ввести индекс бита, который надо установить. Соответственно, при вводе значения меньше 0 или больше либо равного размеру битового поля возникнет ошибка, пользователю необходимо будет повторить ввод заново. Например, установим первый бит (изначально наше поле выглядело так: 10100). Пример выполнения программы представлен на рисунке 9.



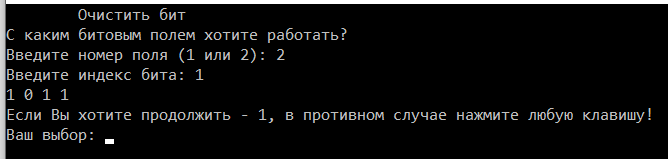
*Рис.9. Работа с битовыми полями. Установление бита.*

На рисунке 9 представлена фраза «Если Вы хотите продолжить – 1, в противном случае нажмите любую клавишу!». Соответственно, если пользователь хочет дальше продолжить работу с битовыми полями, то он должен выбрать цифру 1, а если не хочет, то, например, выбрать цифру 0. При выборе цифры 1 пользователь вернется к меню, представленном на рисунке 6. При выборе цифры 0 или же любой другой клавиши, отличной от 1, появится следующее сообщение: «Если Вы хотите вернуться к исходному меню – 1, завершить программу – любая клавиша». При выборе единицы вернемся к меню, представленному на рисунке 1, при выборе любой другой клавиши программа завершит свою работу. Пример работы программы представлен на рисунке 10.

*Рис.10. Завершение программы, либо переход к главному меню.*

**Пункт – 2. Очистить бит.**

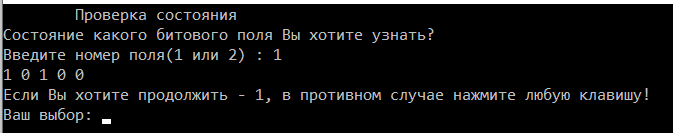
Последовательность действий аналогична выполнению первого пункта. Пример работы представлен на рисунке 11. Изначально наше поле имело вид 1111, очищаем первый бит, получаем соответствующий результат.



*Рис.11. Работа с битовыми полями. Очистка бита.*

**Пункт – 3. Проверка состояния.**

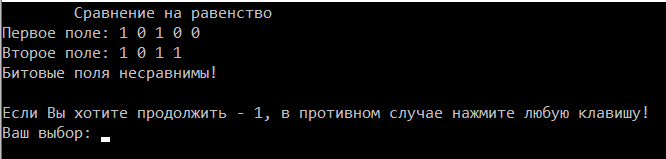
В этом пункте пользователь может проверить состояние первого или второго битового поля. Ему нужно выбрать конкретное поле, после чего на экране появится необходимая ему информация (рис.12).



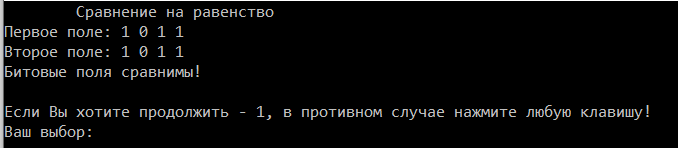
*Рис.12. Работа с битовыми полями. Проверка состояния.*

**Пункт – 4. Сравнение на равенство.**

В данном пункте можно убедиться в равенстве битовых полей. На нашем конкретном примере программа работает следующим образом (рис.13). В нашем случае поля различны, поэтому появляется сообщение «Битовые поля несравнимы!». Если бы поля имели одинаковый размер и одинаковые элементы, то пользователь получил бы сообщение следующего характера «Битовые поля сравнимы!».



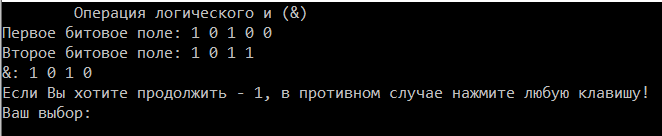
*Рис.13. Работа с битовыми полями. Сравнение на равенство. Битовые поля несравнимы.*



*Рис.14. Работа с битовыми полями. Сравнение на равенство. Битовые поля сравнимы.*

**Пункт – 5. Операция логического и (&).**

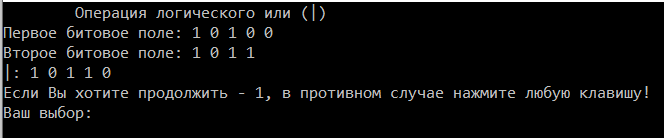
В данном пункте пользователю не нужно вводить никакие значения. На экране лишь будет представлен результат выполнения операции &. Пример выполнения программы представлен на рисунке 15.



*Рис.15. Работа с битовыми полями. Операция логического и (&).*

**Пункт – 6. Операция логического или (|)**

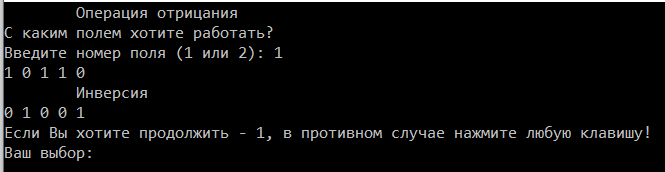
Последовательность действий аналогична выполнению пункта 5 (рис.16).



*Рис.16. Работа с битовыми полями. Операция логического или (|).*

**Пункт – 7. Операция отрицания.**

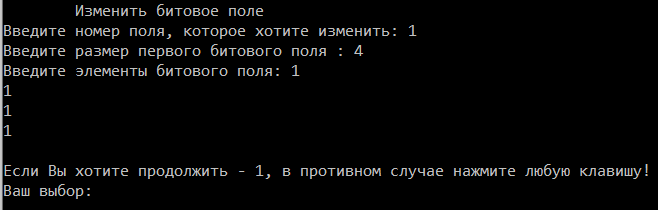
Сначала пользователю необходимо выбрать поле, с которым он будет работать. Далее будет выполнения операция отрицания, после чего на экране появится результат выполнения операции. Пример выполнения программы представлен на рисунке 17.



*Рис.17. Работа с битовыми полями. Операция отрицания.*

**Пункт – 8. Изменить битовое поле.**

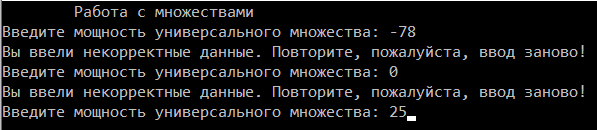
В данном пункте пользователь может заменить существующее поле на новое. Сначала нужно выбрать какое поле необходимо изменить. Далее пользователю необходимо корректно ввести размер битового поля и элементы битового поля (рис.18).



*Рис.18. Работа с битовыми полями. Изменить битовое поле.*

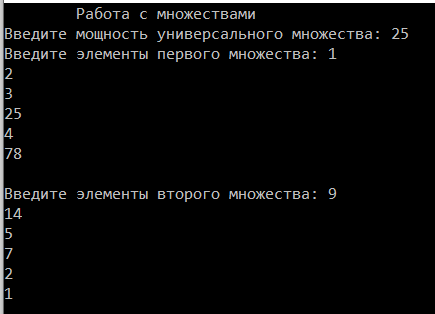
Все операции, которые можно проводить над битовыми полями описаны. Теперь перейдем к описанию работы с множествами.

Итак, после выбора пункта 2 в меню, представленном на рисунке 1, пользователю необходимо будет указать мощность универсального множества. Как известно, она не может быть меньше нуля и равной нулю тоже. При некорректном вводе данных пользователь получит сообщение о том, что значения введены неверно, нужно будет повторить ввод заново. Пример выполнения программы представлен на рисунке 19.



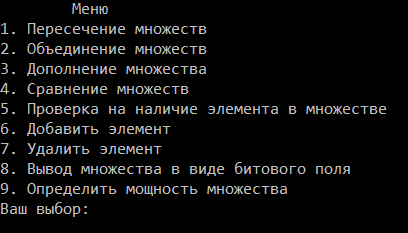
*Рис.19. Работа с множествами. Создание универсального множества.*

Далее пользователю необходимо будет задать два множества. Ввод элементов множества прекращается при вводе числа, не входящего в диапазон от (0, N], где N – мощность универсального множества (рис.20).

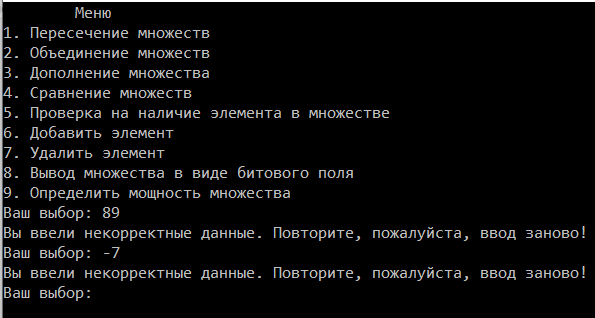


*Рис.20. Работа с множествами. Ввод элементов множеств.*

После создания универсального множества и двух множеств, на экране появляется меню, в котором прописаны какие действия пользователь может выполнить с множествами (рис.21(а)).



*Рис.21(а). Работа с множествами. Меню.*

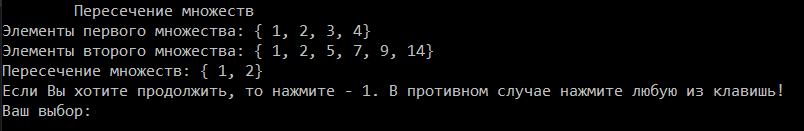


*Рис.21(б). Работа с множествами. Меню.*

Меню состоит из 9 пунктов. Разберемся в выполнении каждого из пунктов. Аналогично предыдущему, при выборе несуществующего пункта, пользователь получит сообщение о том, что данные были введены некорректно и нужно будет заново ввести номер пункта (рис.21(б)).

**Пункт – 1. Пересечение множеств.**

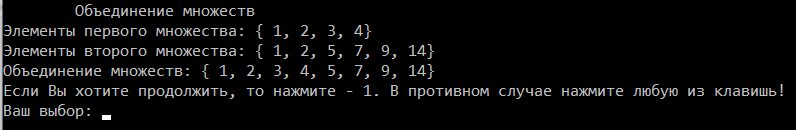
В данном пункте пользователю не нужно будет вводить какие-то новые значения. На экране появится результат пересечения двух, введенных ранее, множеств. На нашем конкретном примере программа работает следующим образом (рис.22).



*Рис.22. Работа с множествами. Пересечение множеств.*

**Пункт – 2. Объединение множеств.**

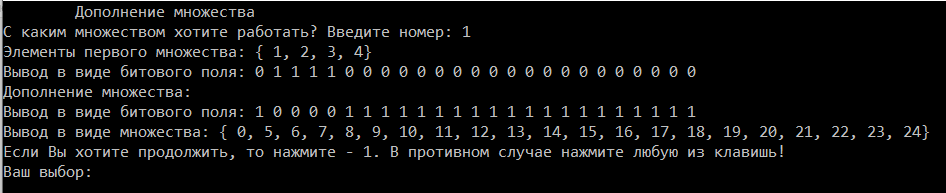
После выполнения данного пункта, на экране появится результат объединения двух множеств (рис.23).



*Рис.23. Работа с множествами. Объединение множеств.*

**Пункт – 3. Дополнение множества.**

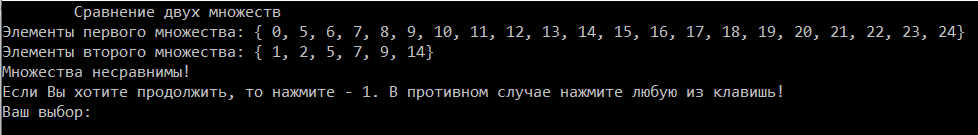
Сначала пользователю необходимо будет выбрать поле, с которым он будет работать. После чего выполнится операция дополнения и на экране появится результат (рис.24).



*Рис.24. Работа с множествами. Дополнение множества.*

**Пункт – 4. Сравнение множеств.**

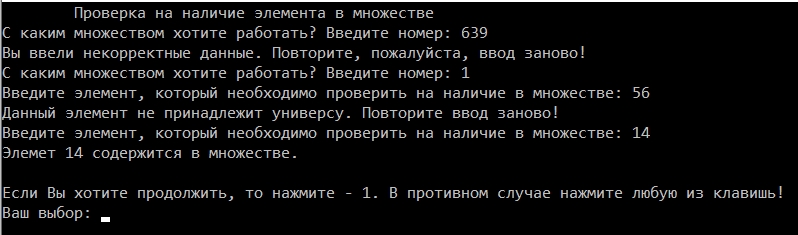
В данном пункте можно убедиться в равенстве двух множеств. На нашем конкретном примере программа работает следующим образом (рис.25). Если элементы первого множества будут совпадать с элементами второго множества, то на экране появится сообщение следующего характера: «Множества сравнимы!».



*Рис.25. Работа с множествами. Сравнение множеств.*

**Пункт – 5. Проверка на наличие элемента в множестве.**

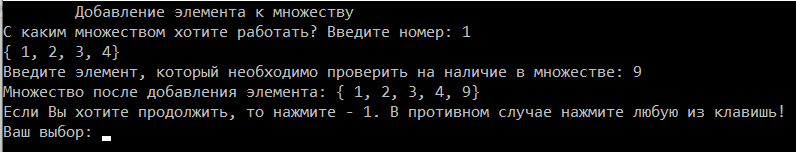
Сначала пользователю необходимо ввести номер множества, в котором он хочет проверить наличие того или иного элемента. После чего необходимо ввести значение элемента. Важно, что этот элемент должен принадлежать универсальному множеству. Если элемент не содержится в множестве, то на экране появится сообщение: «Элемент n не содержится в множестве.». Пример выполнения программы представлен на рисунке 26.



*Рис.26. Работа с множествами. Проверка на наличие элемента в множестве.*

**Пункт – 6. Добавить элемент.**

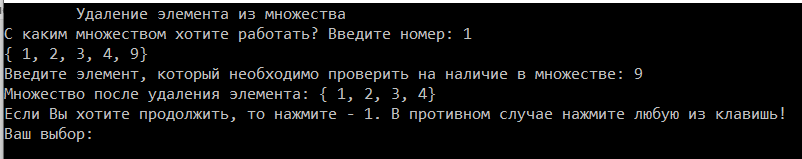
Сначала пользователь выбирает с каким полем будет работать. Далее вводит значение элемента, который хочет добавить в множество. Важно, что этот элемент должен принадлежать универсальному множеству.



*Рис.27.Работа с множествами. Добавление элемента к множеству.*

**Пункт – 7. Удаление элемента.**

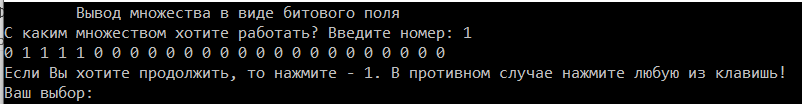
Последовательность действий аналогична пункту 6. Пример выполнения программы представлен на рисунке 28.



*Рис.28. Работа с множествами. Удаление элемента.*

**Пункт – 8. Вывод множества в виде битового поля.**

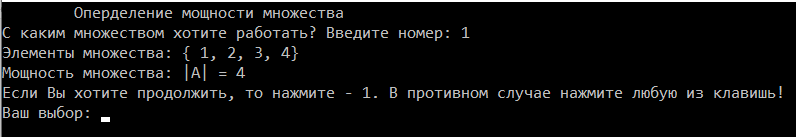
Пункт выполняется следующим образом (рис.29).



*Рис.29. Работа с множествами. Вывод множества в виде битового поля.*

**Пункт – 9. Определить мощность множества.**

Пользователю необходимо выбрать номер поля. После чего на экране появится мощность этого множества (рис.30).



*Рис.30. Работа с множествами. Определение мощности множества.*

# **Руководство программиста**

Описание структуры программы

Программа состоит из пяти файлов:

1. Заголовочный файл – TBitField.h
2. Заголовочный файл – TSet.h
3. Файл реализации – TBitField.cpp
4. Файл реализации – TSet.cpp
5. Файл с прикладной программой – Source.cpp

Описание класса TBitField

Поля:

1. int BitLen – длина битового поля
2. int\* pMem – массив
3. int MemLen – размер массива

Методы:

1. private:

* int GetMemIndex(const int n) const; - получение индекса
* int GetMemMask(const int n) const; - получение маски

1. public:

* TBitField(int len); - конструктор инициализатор
* TBitField(const TBitField& elem); - конструктор копирования
* void SetBit(const int n); - установить бит
* void ClsBit(const int n); - очистить бит
* int GetBit(int n); - проверка состояния бита
* int FieldLen(); - определение длины битового поля
* TBitField& operator=(const TBitField& elem); - перегрузка операции присваивания
* bool operator ==(const TBitField& elem); - перегрузка операции сравнения
* void operator ~(); - перегрузка операции дополнения
* TBitField operator|(const TBitField& elem); - перегрузка операции логическое или
* TBitField operator&(const TBitField& elem); - перегрузка операции логического и
* friend istream& operator>>(istream& is, TBitField& elem); - перегрузка операции типа >>
* friend ostream& operator<<(ostream& os, TBitField& elem); - перегрузка операции типа <<
* ~TBitField(); - деструктор

Описание класса TSet

Поля:

1. int MaxPower; - мощность множества
2. TBitField BitField; - битовое поле

Методы:

1. TSet(int mp); - конструктор инициализатор
2. TSet(const TSet& s); - конструктор копирования
3. TSet(TBitField& bf); - конструктор преобразования
4. operator TBitField(); - оператор преобразования типа
5. int InsElem(const int n); - функция добавления элемента
6. int DelElem(const int n); - функция удаления элемента
7. TSet operator+(const int n); - перегрузка операции + (добавление элемента в множество)
8. TSet operator-(const int n); - перегрузка операции – (удаление элемента из множества)
9. TSet operator~(); - перегрузка операции дополнения
10. TSet operator+(TSet& elem); - перегрузка операции + (объединение множеств)
11. оTSet operator\*(TSet& elem); - перегрузка операции \* (пересечение множеств)
12. int CheckElem(const int n); -
13. void PrintElem(int n); - функция вывода множества
14. int DeterminePower(); - функция определения мощности множества
15. bool operator == (const TSet& elem); - перегрузка операции сравнения
16. friend istream& operator>>(istream& is, TSet& elem); - перегрузка операции типа >>
17. friend ostream& operator<<(ostream& os, TSet& elem); -перегрузка операции типа <<

**Заключение**

**В ходе лабораторной работы была разработана** структура данных для хранения множеств с использованием битовых полей, а также освоение таких инструментов разработки программного обеспечения, как система контроля версий [Git](https://git-scm.com/book/ru/v2) и фрэймворк для разработки автоматических тестов [Google Test](https://github.com/google/googletest).

И были решены следующие задачи:

1. Реализация класса битового поля TBitField согласно заданному интерфейсу.
2. Реализация класса множества TSet согласно заданному интерфейсу.
3. Обеспечение работоспособности тестов и примера использования.
4. Реализация нескольких простых тестов на базе Google Test.
5. Публикация исходных кодов в личном репозитории на GitHub.

**Приложение 1**

Начало формы

Конец формы

src/tbitfield.cpp :

static const int FAKE\_INT = -1;

static TBitField FAKE\_BITFIELD(1);

TBitField::TBitField(int len) //конструктор инициализации

{

if (len > 0)

{

BitLen = len;

MemLen = (len - 1) / (sizeof(TELEM) \* 8) + 1;

pMem = new TELEM[MemLen];

for (int i = 0; i < MemLen; i++)

{

pMem[i] = 0;

}

}

else

{

throw '!';

}

}

TBitField::TBitField(const TBitField& bf) // конструктор копирования

{

if (bf.pMem == 0)

{

BitLen = 0;

pMem = 0;

MemLen = 0;

}

else

{

MemLen = bf.MemLen;

pMem = new TELEM[MemLen];

BitLen = bf.BitLen;

for (int i = 0; i < MemLen; i++)

{

pMem[i] = bf.pMem[i];

}

}

}

TBitField::~TBitField()

{

if (pMem != 0)

{

delete[] pMem;

BitLen = 0;

pMem = 0;

MemLen = 0;

}

}

int TBitField::GetMemIndex(const int n) const // индекс Мем для бита n

{

if (n >= 0)

{

return n >> (int)(log2(8 \* sizeof(TELEM)));

}

else

{

throw '!';

}

}

TELEM TBitField::GetMemMask(const int n) const // битовая маска для бита n

{

if (n >= 0)

{

return 1 << (n & (sizeof(TELEM) \* 8 - 1));

}

else

{

throw '!';

}

}

// 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)

{

pMem[GetMemIndex(n)] = pMem[GetMemIndex(n)] | GetMemMask(n);

}

else

{

throw '!';

}

}

void TBitField::ClrBit(const int n) // очистить бит

{

if (n >= 0 && n < BitLen)

{

pMem[GetMemIndex(n)] = pMem[GetMemIndex(n)] & (~GetMemMask(n));

}

else

{

throw '!';

}

}

int TBitField::GetBit(const int n) const // получить значение бита

{

if (n >= 0 && n < BitLen)

{

return (pMem[GetMemIndex(n)] & GetMemMask(n)) != 0;

}

else

{

throw '!';

}

}

// битовые операции

TBitField& TBitField::operator=(const TBitField& bf) // присваивание

{

if (this == &bf)

{

return \*this;

}

if (pMem != 0)

{

delete[] pMem;

}

if (bf.pMem == 0)

{

BitLen = 0;

pMem = 0;

MemLen = 0;

}

else

{

BitLen = bf.BitLen;

MemLen = (bf.BitLen - 1) / (sizeof(TELEM) \* 8) + 1;

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 false;

}

for (int i = 0; i < BitLen; i++)

{

if (GetBit(i) != bf.GetBit(i))

{

return false;

}

}

return true;

}

int TBitField::operator!=(const TBitField& bf) const // сравнение

{

if (BitLen != bf.BitLen)

{

return true;

}

for (int i = 0; i < BitLen; i++)

{

if (GetBit(i) != bf.GetBit(i))

{

return true;

}

}

return false;

}

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

{

TBitField res(std::max(BitLen, bf.BitLen));

for (int i = 0; i < std::min(BitLen, bf.BitLen); i++)

{

if ((GetBit(i) == 1) || (bf.GetBit(i) == 1))

{

res.SetBit(i);

}

}

for (int i = std::min(BitLen, bf.BitLen); i < std::max(BitLen, bf.BitLen); i++)

{

if (BitLen > bf.BitLen)

{

if (GetBit(i) == 1)

{

res.SetBit(i);

}

}

else

{

if (bf.GetBit(i) == 1)

{

res.SetBit(i);

}

}

}

return res;

}

TBitField TBitField::operator&(const TBitField& bf) // операция "и"

{

TBitField res(std::max(BitLen, bf.BitLen));

for (int i = 0; i < std::min(BitLen, bf.BitLen); i++)

{

if ((GetBit(i) == 1) && (bf.GetBit(i) == 1))

{

res.SetBit(i);

}

}

return res;

}

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

{

TBitField res(BitLen);

for (int i = 0; i < BitLen; i++)

{

if (GetBit(i) == 1)

{

res.ClrBit(i);

}

else

{

res.SetBit(i);

}

}

return res;

}

istream& operator>>(istream& istr, TBitField& bf) // ввод

{

/\*char str[1000] = {'2'};

int i = 0;

istr >> str;

while (str[i] != '2')

{

i = i + 1;

}

bf.BitLen = i;

bf.MemLen = (i - 1) / (sizeof(TELEM) \* 8) + 1;

delete[] bf.pMem;

bf.pMem = new TELEM[bf.MemLen];

std::cout << str << ' ' << bf.BitLen << ' ' << bf.MemLen << '/n';

for (i = 0; i < bf.BitLen; i++)

{

b.str[i] = s[i];

}

b.str[b.len] = 0;

return istr;\*/

}

ostream& operator<<(ostream& ostr, const TBitField& bf) // вывод

{

if (bf.pMem == 0)

{

ostr << "";

return ostr;

}

for (int i = 0; i < bf.BitLen; i++)

{

if (bf.GetBit(i) == 1)

{

ostr << "1";

}

else

{

ostr << "0";

}

}

return ostr;

}

**Приложение 2**

src/tset.cpp :

static TBitField FAKE\_BITFIELD(1);

static TSet FAKE\_SET(1);

//конструктор инициализации

TSet::TSet(int mp) : BitField(1)

{

if (mp > 0)

{

MaxPower = mp;

BitField = TBitField(MaxPower);

}

else

{

throw '!';

}

}

// конструктор копирования

TSet::TSet(const TSet& s) : BitField(1)

{

MaxPower = s.MaxPower;

BitField = s.BitField;

}

// конструктор преобразования типа

TSet::TSet(const TBitField& bf) : BitField(1)

{

MaxPower = bf.GetLength();

BitField = bf;

}

TSet::operator TBitField() // преобразование типа к битовому полю

{

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 (BitField.GetBit(Elem) == 1)

{

return true;

}

return false;

}

void TSet::InsElem(const int Elem) // включение элемента множества

{

if (Elem >= 0)

{

BitField.SetBit(Elem);

}

else

{

throw '!';

}

}

void TSet::DelElem(const int Elem) // исключение элемента множества

{

if (Elem >= 0)

{

BitField.ClrBit(Elem);

}

else

{

throw '!';

}

}

// теоретико-множественные операции

TSet& TSet::operator=(const TSet& s) // присваивание

{

if (this == &s)

{

return \*this;

}

MaxPower = s.MaxPower;

BitField = s.BitField;

return \*this;

}

int TSet::operator==(const TSet& s) const // сравнение

{

if ((MaxPower == s.MaxPower) && (BitField == s.BitField))

{

return true;

}

return false;

}

int TSet::operator!=(const TSet& s) const // сравнение

{

if ((MaxPower != s.MaxPower) || (BitField != s.BitField))

{

return true;

}

return false;

}

TSet TSet::operator+(const TSet& s) // объединение

{

TSet res(std::max(MaxPower, s.MaxPower));

res.BitField = BitField | s.BitField;

return res;

}

TSet TSet::operator+(const int Elem) // объединение с элементом

{

TSet res(std::max(Elem, MaxPower));

res.InsElem(Elem);

return res;

}

TSet TSet::operator-(const int Elem) // разность с элементом

{

TSet res(MaxPower);

res.DelElem(Elem);

return res;

}

TSet TSet::operator\*(const TSet& s) // пересечение

{

TSet res(std::max(MaxPower, s.MaxPower));

res.BitField = BitField & s.BitField;

return res;

}

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

{

TSet res(MaxPower);

res.BitField = ~BitField;

return res;

}

// перегрузка ввода/вывода

istream& operator>>(istream& istr, TSet& s) // ввод

{

return istr;

}

ostream& operator<<(ostream& ostr, const TSet& s) // вывод

{

return ostr;

}