**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 2**

Тема: Перегрузка операторов в С++

Студент: Ефимов Александр

Группа: 80-201

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2019

1. Постановка задачи

**Создать класс** **BitString** для работы с 128-битовыми строками. Битовая строка должна быть представлена двумя полями типа unsigned long long. Должны быть реализованы все традиционные операции для работы с битами: and, or, xor, not. Реализовать сдвиг влево shiftLeft и сдвиг вправо shiftRight на заданное количество битов. Реализовать операцию вычисления количества единичных битов, операции сравнения по количеству единичных битов. Реализовать операцию проверки включения.

Операции and, or, xor, not, >>, << , сравнения (на равенство, больше и меньше) должны быть выполнены в виде перегрузки операторов.

Необходимо реализовать пользовательский литерал для работы с константами типа **BitString**.

1. Описание программы

Программа разбита на три исходных файла: объявление класса и функций в *ByteString.h*, их определения в *ByteString.cpp*, а также созданное элементарное меню в *main.cpp*.

В результате библиотека *ByteString* определяет работу над 128-битовыми строками, представленных через две переменные формата *unsigned long long*, при помощи

* перегруженных *rvalue* операторов
  + **&** (AND),
  + **|** (OR),
  + **^** (XOR),
  + **<<** (сдвиг влево),
  + **>>** (сдвиг вправо);
* *lvalue* операторов &=, |=, **^=** ~ (инверсия битов), <<=, >>=;
* Bool операторы (==, >, <)
* Операторы ввода (cin) и вывода (cout) (для lvalue и rvalue переменных)

А также пользовательский литерал “”\_b, преобразующий строку в битовую строку.

В *main.cpp* содержится простое меню для работы с библиотечными методами, а также функция *printList*, выдающая список этих строк.

1. Набор testcases

При запуске программы создается строка с значением 101101 (ведущие нули опущены) для демонстрации работы пользовательского литерала. Следующие списки чисел последовательно вводятся в меню. Они позволяют вызвать каждую (или большинство) функций для проверки работоспособности.

|  |  |  |
| --- | --- | --- |
| № | **test\_01.txt** | **test\_02.txt** |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | 2  1  2  010010  1  2  1111111111  2  3  1  5  4  2  2  5  5  3  2  3  4  5  1  3  2  7  6  7  3  3  3  6  4  4  6  6  6  5  4  0 | 1  2  1111111  1  2  11000000000000  1  1  1 31  2  3  1  4  2  6  2  2  2  3  3  6  7  7  2  6  2  1  2  6  9  2  6  7  12  2  9  2  5  1  3  12  2  6  5  4  9  2  0 |

1. Результаты выполнения тестов.

Изначально тесты были не пройдены, так как результат не в тесте 1 вводе № 8 не соответствовал ожидаемым (байтовые строки выводились в неверном формате), но, после исправления обнаруженной ошибки, все тесты вывели ожидаемые результаты.

* **test\_01Result.txt**

1. Add byte string

2. Print list of byte strings

3. Select byte string to manipulate

4. Delete selected byte\_string

5. Add operation on byte strings to list

6. Operate on selected byte string

7. Compare two byte strings

8. Print the list again

9. Clear list of byte strings

---------------

Menu choice: 2

1. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00101101

---------------

Menu choice: 1

Input format (1 for numbers, 2 for string): 2

Input byte string in bits: 010010

---------------

Menu choice: 1

Input format (1 for numbers, 2 for string): 2

Input byte string in bits: 1111111111

---------------

Menu choice: 2

1. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00101101

2. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00010010

3. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000011 11111111

---------------

Menu choice: 3

Type number of string (0 to deselect): 1

---------------

Menu choice: 5

1. Shift bits left result

2. Shift bits right result

3. AND of two byte strings

4. OR of two byte strings

5. XOR of two byte strings

Any other number returns to main menu automatically

Choice: 4

\* 1. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00101101

2. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00010010

3. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000011 11111111

Select a second byte string: 2

---------------

Menu choice: 2

\* 1. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00101101

2. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00010010

3. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000011 11111111

4. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00111111

---------------

Menu choice: 5

1. Shift bits left result

2. Shift bits right result

3. AND of two byte strings

4. OR of two byte strings

5. XOR of two byte strings

Any other number returns to main menu automatically

Choice: 5

\* 1. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00101101

2. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00010010

3. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000011 11111111

4. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00111111

Select a second byte string: 3

---------------

Menu choice: 2

\* 1. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00101101

2. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00010010

3. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000011 11111111

4. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00111111

5. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000011 11010010

---------------

Menu choice: 3

Type number of string (0 to deselect): 2

---------------

Menu choice: 3

Type number of string (0 to deselect): 4

---------------

Menu choice: 5

1. Shift bits left result

2. Shift bits right result

3. AND of two byte strings

4. OR of two byte strings

5. XOR of two byte strings

Any other number returns to main menu automatically

Choice: 3

1. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00101101

2. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00010010

3. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000011 11111111

\* 4. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00111111

5. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000011 11010010

Select a second byte string: 3

---------------

Menu choice: 2

1. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00101101

2. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00010010

3. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000011 11111111

\* 4. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00111111

5. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000011 11010010

6. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00111111

---------------

Menu choice: 7

1. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00101101

2. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00010010

3. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000011 11111111

\* 4. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00111111

5. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000011 11010010

6. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00111111

Select a second byte string: 6

Selected byte strings are equal

---------------

Menu choice: 7

1. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00101101

2. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00010010

3. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000011 11111111

\* 4. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00111111

5. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000011 11010010

6. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00111111

Select a second byte string: 3

Second string is bigger than the other

---------------

Menu choice: 3

Type number of string (0 to deselect): 3

---------------

Menu choice: 6

1. AND operation on selected byte string with another

2. OR operation on selected byte string with another

3. XOR operation on selected byte string with another

4. Check inclusion of another string in selected

5. Check if amount of 1's in byte strings is equal

6. Return amount of 1's in the string

7. Shift bits left

8. Shift bits right

9. Flip bits on selected bit string

Any other number returns to main menu automatically

Choice: 4

1. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00101101

2. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00010010

\* 3. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000011 11111111

4. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00111111

5. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000011 11010010

6. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00111111

Select a second byte string: 4

Selected string includes another string

---------------

Menu choice: 6

1. AND operation on selected byte string with another

2. OR operation on selected byte string with another

3. XOR operation on selected byte string with another

4. Check inclusion of another string in selected

5. Check if amount of 1's in byte strings is equal

6. Return amount of 1's in the string

7. Shift bits left

8. Shift bits right

9. Flip bits on selected bit string

Any other number returns to main menu automatically

Choice: 6

Selected strings 10 of 1's

---------------

Menu choice: 6

1. AND operation on selected byte string with another

2. OR operation on selected byte string with another

3. XOR operation on selected byte string with another

4. Check inclusion of another string in selected

5. Check if amount of 1's in byte strings is equal

6. Return amount of 1's in the string

7. Shift bits left

8. Shift bits right

9. Flip bits on selected bit string

Any other number returns to main menu automatically

Choice: 5

1. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00101101

2. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00010010

\* 3. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000011 11111111

4. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00111111

5. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000011 11010010

6. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00111111

Select a second byte string: 4

Selected strings do not have the same amount of 1's

---------------

Menu choice: 0

* **test\_02Result.txt**

1. Add byte string

2. Print list of byte strings

3. Select byte string to manipulate

4. Delete selected byte\_string

5. Add operation on byte strings to list

6. Operate on selected byte string

7. Compare two byte strings

8. Print the list again

9. Clear list of byte strings

---------------

Menu choice: 1

Input format (1 for numbers, 2 for string): 2

Input byte string in bits: 1111111

---------------

Menu choice: 1

Input format (1 for numbers, 2 for string): 2

Input byte string in bits: 11000000000000

---------------

Menu choice: 1

Input format (1 for numbers, 2 for string): 1

Input byte string in two numbers: 1

31

---------------

Menu choice: 2

1. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00101101

2. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 01111111

3. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00110000 00000000

4. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000001

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00011111

---------------

Menu choice: 3

Type number of string (0 to deselect): 1

---------------

Menu choice: 4

Selected byte string deleted

---------------

Menu choice: 2

\* 1. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 01111111

2. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00110000 00000000

3. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000001

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00011111

---------------

Menu choice: 6

1. AND operation on selected byte string with another

2. OR operation on selected byte string with another

3. XOR operation on selected byte string with another

4. Check inclusion of another string in selected

5. Check if amount of 1's in byte strings is equal

6. Return amount of 1's in the string

7. Shift bits left

8. Shift bits right

9. Flip bits on selected bit string

Any other number returns to main menu automatically

Choice: 2

\* 1. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00000000 01111111

2. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00110000 00000000

3. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000001

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00011111

Select a second byte string: 2

---------------

Menu choice: 2

\* 1. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00110000 01111111

2. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00110000 00000000

3. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000001

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00011111

---------------

Menu choice: 3

Type number of string (0 to deselect): 3

---------------

Menu choice: 6

1. AND operation on selected byte string with another

2. OR operation on selected byte string with another

3. XOR operation on selected byte string with another

4. Check inclusion of another string in selected

5. Check if amount of 1's in byte strings is equal

6. Return amount of 1's in the string

7. Shift bits left

8. Shift bits right

9. Flip bits on selected bit string

Any other number returns to main menu automatically

Choice: 7

Shift by: 7

---------------

Menu choice: 2

1. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00110000 01111111

2. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00110000 00000000

\* 3. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 10000000

00000000 00000000 00000000 00000000 00000000 00000000 00001111 10000000

---------------

Menu choice: 6

1. AND operation on selected byte string with another

2. OR operation on selected byte string with another

3. XOR operation on selected byte string with another

4. Check inclusion of another string in selected

5. Check if amount of 1's in byte strings is equal

6. Return amount of 1's in the string

7. Shift bits left

8. Shift bits right

9. Flip bits on selected bit string

Any other number returns to main menu automatically

Choice: 2

1. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00110000 01111111

2. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00110000 00000000

\* 3. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 10000000

00000000 00000000 00000000 00000000 00000000 00000000 00001111 10000000

Select a second byte string: 1

---------------

Menu choice: 2

1. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00110000 01111111

2. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00110000 00000000

\* 3. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 10000000

00000000 00000000 00000000 00000000 00000000 00000000 00111111 11111111

---------------

Menu choice: 6

1. AND operation on selected byte string with another

2. OR operation on selected byte string with another

3. XOR operation on selected byte string with another

4. Check inclusion of another string in selected

5. Check if amount of 1's in byte strings is equal

6. Return amount of 1's in the string

7. Shift bits left

8. Shift bits right

9. Flip bits on selected bit string

Any other number returns to main menu automatically

Choice: 9

---------------

Menu choice: 2

1. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00110000 01111111

2. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00110000 00000000

\* 3. 11111111 11111111 11111111 11111111 11111111 11111111 11111111 01111111

11111111 11111111 11111111 11111111 11111111 11111111 11000000 00000000

---------------

Menu choice: 6

1. AND operation on selected byte string with another

2. OR operation on selected byte string with another

3. XOR operation on selected byte string with another

4. Check inclusion of another string in selected

5. Check if amount of 1's in byte strings is equal

6. Return amount of 1's in the string

7. Shift bits left

8. Shift bits right

9. Flip bits on selected bit string

Any other number returns to main menu automatically

Choice: 7

Shift by: 0

---------------

Menu choice: 6

1. AND operation on selected byte string with another

2. OR operation on selected byte string with another

3. XOR operation on selected byte string with another

4. Check inclusion of another string in selected

5. Check if amount of 1's in byte strings is equal

6. Return amount of 1's in the string

7. Shift bits left

8. Shift bits right

9. Flip bits on selected bit string

Any other number returns to main menu automatically

Choice: 8

Shift by: 12

---------------

Menu choice: 2

1. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00110000 01111111

2. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00110000 00000000

\* 3. 00000000 00001111 11111111 11111111 11111111 11111111 11111111 11111111

11110111 11111111 11111111 11111111 11111111 11111111 11111111 11111100

---------------

Menu choice: 5

1. Shift bits left result

2. Shift bits right result

3. AND of two byte strings

4. OR of two byte strings

5. XOR of two byte strings

Any other number returns to main menu automatically

Choice: 1

1. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00110000 01111111

2. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00110000 00000000

\* 3. 00000000 00001111 11111111 11111111 11111111 11111111 11111111 11111111

11110111 11111111 11111111 11111111 11111111 11111111 11111111 11111100

Select byte string: 3

Shift by: 12

---------------

Menu choice: 2

1. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00110000 01111111

2. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00110000 00000000

\* 3. 00000000 00001111 11111111 11111111 11111111 11111111 11111111 11111111

11110111 11111111 11111111 11111111 11111111 11111111 11111111 11111100

4. 11111111 11111111 11111111 11111111 11111111 11111111 11111111 01111111

11111111 11111111 11111111 11111111 11111111 11111111 11000000 00000000

---------------

Menu choice: 6

1. AND operation on selected byte string with another

2. OR operation on selected byte string with another

3. XOR operation on selected byte string with another

4. Check inclusion of another string in selected

5. Check if amount of 1's in byte strings is equal

6. Return amount of 1's in the string

7. Shift bits left

8. Shift bits right

9. Flip bits on selected bit string

Any other number returns to main menu automatically

Choice: 5

1. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00110000 01111111

2. 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000 00000000 00110000 00000000

\* 3. 00000000 00001111 11111111 11111111 11111111 11111111 11111111 11111111

11110111 11111111 11111111 11111111 11111111 11111111 11111111 11111100

4. 11111111 11111111 11111111 11111111 11111111 11111111 11111111 01111111

11111111 11111111 11111111 11111111 11111111 11111111 11000000 00000000

Select a second byte string: 4

Selected strings have the same amount of 1's

---------------

Menu choice: 9

List clear

---------------

Menu choice: 2

The list is currently empty

---------------

Menu choice: 0

1. Листинг программы

* **ByteString.h**

#ifndef BYTE\_STRING\_H

#define BYTE\_STRING\_H

#include <string>

#include <iostream>

class ByteString

{

public:

friend bool CompareOnes(const ByteString&, const ByteString&);

friend std::istream& operator >> (std::istream&, ByteString& );

friend std::ostream& operator << (std::ostream&, ByteString& );

friend std::ostream& operator << (std::ostream&, ByteString&&);

friend ByteString operator "" \_b(const char\*, std::size\_t);

ByteString() : msp(0), lsp(0) {}

ByteString(long long a, long long b) : msp(a), lsp(b) {}

// = operator is default initialized

//Operations on bytes, returning the copy

ByteString operator & (const ByteString&) const;

ByteString operator | (const ByteString&) const;

ByteString operator ^ (const ByteString&) const;

ByteString operator >> (const size\_t&) const;

ByteString operator << (const size\_t&) const;

//Operations on existing ByteString

ByteString& operator &= (const ByteString&);

ByteString& operator |= (const ByteString&);

ByteString& operator ^= (const ByteString&);

ByteString& operator ~ (void);

ByteString& operator >>= (const size\_t&);

ByteString& operator <<= (const size\_t&);

//Comparison

bool operator < (const ByteString&) const;

bool operator > (const ByteString&) const;

bool operator == (const ByteString&) const;

bool includes (const ByteString&) const;

//Calculating amount of 1's

size\_t Ones() const;

private:

unsigned long long msp, lsp; // As in - Most Significant Part, Least Significant Part;

};

bool CompareOnes(const ByteString&, const ByteString&);

std::istream& operator >> (std::istream& is, ByteString& rhs);

std::ostream& operator << (std::ostream&, ByteString&);

std::ostream& operator << (std::ostream&, ByteString&&);

ByteString operator "" \_b(const char\*, std::size\_t);

#endif

* **ByteString.cpp**

#include <string>

#include <exception>

#include "ByteString.h"

const unsigned long long L\_BYTE = 9223372036854775808U; // Value of last bit of ULLong type

const size\_t B\_STR\_SIZE = 128;

//==========================================================In-class functions

//Operations on bytes, returning the copy

ByteString ByteString::operator & (const ByteString& rhs) const

{

ByteString res(msp & rhs.msp, lsp & rhs.lsp);

return res;

}

ByteString ByteString::operator | (const ByteString& rhs) const

{

ByteString res(msp | rhs.msp, lsp | rhs.lsp);

return res;

}

ByteString ByteString::operator ^ (const ByteString& rhs) const

{

ByteString res(msp ^ rhs.msp, lsp ^ rhs.lsp);

return res;

}

ByteString ByteString::operator >> (const size\_t& by) const

{

ByteString res = \*this;

if (by > 63)

{

res.lsp = res.msp;

res.msp = 0;

res.lsp >>= by - 64;

}

else

{

res.lsp >>= by;

res.lsp |= res.msp << (64 - by);

res.msp >>= by;

}

return res;

}

ByteString ByteString::operator << (const size\_t& by) const

{

ByteString res = \*this;

if (by > 63)

{

res.msp = res.lsp;

res.lsp = 0;

res.msp <<= by - 64;

}

else

{

res.msp <<= by;

res.msp |= res.lsp >> (64 - by);

res.lsp <<= by;

}

return res;

}

//Operations on existing ByteString

ByteString& ByteString::operator &= (const ByteString& rhs)

{

msp &= rhs.msp;

lsp &= rhs.lsp;

return \*this;

}

ByteString& ByteString::operator |= (const ByteString& rhs)

{

msp |= rhs.msp;

lsp |= rhs.lsp;

return \*this;

}

ByteString& ByteString::operator ^= (const ByteString& rhs)

{

msp ^= rhs.msp;

lsp ^= rhs.lsp;

return \*this;

}

ByteString& ByteString::operator >>= (const size\_t& by)

{

if (by > 63)

{

lsp = msp;

msp = 0;

lsp >>= by - 64;

}

else

{

lsp >>= by;

lsp |= msp << (64 - by);

msp >>= by;

}

return \*this;

}

ByteString& ByteString::operator <<= (const size\_t& by)

{

if (by > 63)

{

msp = lsp;

lsp = 0;

msp <<= by - 64;

}

else

{

msp <<= by;

msp |= lsp >> (64 - by);

lsp <<= by;

}

return \*this;

}

ByteString& ByteString::operator ~ (void)

{

msp = ~msp;

lsp = ~lsp;

return \*this;

}

//Comparison

bool ByteString::operator < (const ByteString& other) const

{

if (msp != other.msp) return msp < other.msp;

return lsp < other.lsp;

}

bool ByteString::operator > (const ByteString& other) const

{

if (msp != other.msp) return msp > other.msp;

return lsp > other.lsp;

}

bool ByteString::operator == (const ByteString& other) const

{ return msp == other.msp && lsp == other.lsp; }

bool ByteString::includes(const ByteString& rhs) const

{

for (size\_t i = 0; i < 64; ++i)

if (( (rhs.msp >> i & 1) == 1 ) && ( (msp >> i & 1) != 1 )) return false;

for (size\_t i = 0; i < 64; ++i)

if (( (rhs.lsp >> i & 1) == 1 ) && ( (lsp >> i & 1) != 1 )) return false;

return true;

}

//Calculating amount of 1's

size\_t ByteString::Ones() const

{

size\_t ones{ 0 };

for (size\_t i = 0; i < 64; ++i)

if ((msp >> i & 1) == 1) ++ones;

for (size\_t i = 0; i < 64; ++i)

if ((lsp >> i & 1) == 1) ++ones;

return ones;

}

//==========================================================Outside class

bool CompareOnes(const ByteString& lhs, const ByteString& rhs)

{

if (lhs.Ones() == rhs.Ones()) return true;

return false;

}

std::istream& operator >> (std::istream& is, ByteString& rhs)

{

std::string str;

is >> str;

if (str.size() > B\_STR\_SIZE) throw std::runtime\_error("Input ByteString overflow");

if (str.size() == 0)

{

rhs.msp = rhs.lsp = 0;

return is;

}

size\_t size = str.size();

unsigned long long msp{ 0 }, lsp{ 0 };

if (size <= B\_STR\_SIZE / 2)

{

for (size\_t i = 0; i < size; ++i)

{

lsp <<= 1;

if (str[i] - '0') lsp |= 1;

}

}

else

{

size\_t i;

for (i = 0; i < size - B\_STR\_SIZE / 2; ++i)

{

msp <<= 1;

if (str[i] - '0') msp |= 1;

}

for (; i < size; ++i)

{

lsp <<= 1;

if (str[i] - '0') lsp |= 1;

}

}

rhs.msp = msp;

rhs.lsp = lsp;

return is;

}

std::ostream& operator << (std::ostream& os, ByteString& rhs)

{

for (size\_t i = 0; i < 64; ++i)

{

if (!(i % 8) && i) os << " " << std::flush;

os << ( (rhs.msp << i) & L\_BYTE ? 1 : 0);

}

std::cout << std::endl;

for (size\_t i = 0; i < 64; ++i)

{

if (!(i % 8) && i) os << " " << std::flush;

os << ((rhs.lsp << i) & L\_BYTE ? 1 : 0);

}

return os;

}

std::ostream& operator << (std::ostream& os, ByteString&& rhs)

{

for (size\_t i = 0; i < 64; ++i)

{

os << ((rhs.msp << i) & L\_BYTE ? 1 : 0);

if (!(i % 8) && i) os << " " << std::flush;

}

std::cout << std::endl;

for (size\_t i = 0; i < 64; ++i)

{

os << ((rhs.lsp << i) & L\_BYTE ? 1 : 0);

if (!(i % 8) && i) os << " " << std::flush;

}

return os;

}

ByteString operator "" \_b(const char\* str, std::size\_t size) //find out the difference with std:: and without

{

if (size > B\_STR\_SIZE) throw std::runtime\_error("Input ByteString overflow");

ByteString res;

if (size == 0)

{

return res;

}

if (size <= B\_STR\_SIZE / 2)

{

for (size\_t i = 0; i < size; ++i)

{

res.lsp <<= 1;

if (str[i] - '0') res.lsp |= 1;

}

}

else

{

size\_t i;

for (i = 0; i < size - B\_STR\_SIZE / 2; ++i)

{

res.msp <<= 1;

if (str[i] - '0') res.msp |= 1;

}

for (; i < size; ++i)

{

res.lsp <<= 1;

if (str[i] - '0') res.lsp |= 1;

}

}

return res;

}

* **main.cpp**

/\*

Вариант 11:

Создать класс BitString для работы с 128-битовыми строками. Битовая строка должна быть представлена двумя полями типа unsigned long long.

Должны быть реализованы все традиционные операции для работы с битами: and, or, xor, not.

Реализовать сдвиг влево shiftLeft и сдвиг вправо shiftRight на заданное количество битов.

Реализовать операцию вычисления количества единичных битов, операции сравнения по количеству единичных битов.

Реализовать операцию проверки включения.

Операции and, or, xor, not, >>, << , сравнения (на равенство, больше и меньше) должны быть выполнены в виде перегрузки операторов.

Необходимо реализовать пользовательский литерал для работы с константами типа BitString.

Ефимов Александр Владимирович

M8O-201Б

\*/

#include <list>

#include <string>

#include <exception>

#include "ByteString.h"

using namespace std;

struct list\_empty : public std::exception

{

const char\* what() const throw ()

{

return "The list is currently empty";

}

};

void printList(list<ByteString>& l, list<ByteString>::iterator& sel)

{

int num = 1;

for (auto i = l.begin(); i != l.end(); ++i)

{

if (i == sel) cout << "\* " << num << ". ";

else cout << " " << num << ". ";

cout << \*i << endl;

++num;

}

cout << flush;

}

int main()

{

list<ByteString> listByte;

listByte.emplace\_back("101101"\_b);

ByteString temp;

int k = 1, num1, num2,

i = 0;

list<ByteString>::iterator it, it2;

it = it2 = listByte.end();

cout

<< "\n1. Add byte string\n"

<< "2. Print list of byte strings\n"

<< "3. Select byte string to manipulate\n"

<< "4. Delete selected byte\_string\n"

<< "5. Add operation on byte strings to list\n"

<< "6. Operate on selected byte string\n"

<< "7. Compare two byte strings\n"

<< "8. Print the list again\n"

<< "9. Clear list of byte strings\n"

<< endl;

do

{

cout << "\n---------------\nMenu choice: ";

try

{

cin >> k;

if (!cin) throw runtime\_error("Failed to get input");

switch (k)

{

case 1:

cout << "Input format (1 for numbers, 2 for string): ";

if (cin >> k)

{

if (k == 1)

{

cout << "Input byte string in two numbers: ";

cin >> num1 >> num2;

if (!cin) throw runtime\_error("Failed to get input");

listByte.emplace\_back(num1, num2);

}

else if (k == 2)

{

cout << "Input byte string in bits: ";

if (cin >> temp) listByte.push\_back(temp);

else

{

k = 1;

throw runtime\_error("Failed to get input");

}

}

k = 1;

}

else

{

k = 1;

throw runtime\_error("Failed to get input");

}

break;

case 2:

if (listByte.empty()) throw list\_empty();

printList(listByte, it);

break;

case 3:

if (listByte.empty()) throw list\_empty();

it = listByte.begin();

cout << "Type number of string (0 to deselect): ";

cin >> i;

if (cin && i > 0 && (unsigned)i <= listByte.size())

{

--i;

advance(it, i);

}

else if (i == 0) it = listByte.end();

else throw runtime\_error("Incorrect number given");

break;

case 4:

if (listByte.empty()) throw list\_empty();

if (it != listByte.end())

{

it = listByte.erase(it);

cout << "Selected byte string deleted" << endl;

}

else throw runtime\_error("Invalid element selected");

break;

case 5:

if (listByte.empty()) throw list\_empty();

if (listByte.size() < 2) throw runtime\_error("Less than 2 strings in list");

cout

<< "1. Shift bits left result\n"

<< "2. Shift bits right result\n"

<< "3. AND of two byte strings\n"

<< "4. OR of two byte strings\n"

<< "5. XOR of two byte strings\n"

<< "Any other number returns to main menu automatically\n"

<< "Choice: ";

cin >> k;

if (!cin) throw runtime\_error("Failed to get input");

if (k >= 1 && k <= 2)

{

printList(listByte, it);

cout << "Select byte string: ";

if (cin >> i && i > 0 && (unsigned)i <= listByte.size())

{

--i;

it = listByte.begin();

advance(it, i);

}

else throw runtime\_error("Incorrect input");

}

else if (k <= 5 && k >= 3)

{

printList(listByte, it);

if (it == listByte.end())

{

cout << "Select first byte string: ";

if (cin >> i && i > 0 && (unsigned)i <= listByte.size())

{

--i;

it = listByte.begin();

advance(it, i);

}

else throw runtime\_error("Incorrect input");

}

if (listByte.size() > 2)

{

cout << "Select a second byte string: ";

it2 = listByte.end();

do

{

it2 = listByte.end();

if (cin >> i && i > 0 && (unsigned)i <= listByte.size())

{

--i;

it2 = listByte.begin();

advance(it2, i);

}

else throw runtime\_error("Incorrect input");

if (it2 == it) cout << "The same string was selected, select another: ";

} while (it2 == listByte.end() || it2 == it);

}

else

{

for (it2 = listByte.begin(); it2 != listByte.end() && it2 == it; ++it2) {}

cout << "Selecting the only byte string left" << endl;

}

}

switch (k)

{

case 1:

cout << "Shift by: ";

cin >> num1;

if (!cin) throw runtime\_error("Failed to get input");

listByte.push\_back(\*it << num1);

break;

case 2:

cout << "Shift by: ";

cin >> num1;

if (!cin) throw runtime\_error("Failed to get input");

listByte.push\_back(\*it >> num1);

break;

case 3:

listByte.push\_back(\*it & \*it2);

break;

case 4:

listByte.push\_back(\*it | \*it2);

break;

case 5:

listByte.push\_back(\*it ^ \*it2);

break;

default:

break;

}

k = 5;

break;

case 6:

if (listByte.empty()) throw list\_empty();

if (it == listByte.end()) throw runtime\_error("No byte string selected");

cout

<< "1. AND operation on selected byte string with another\n"

<< "2. OR operation on selected byte string with another\n"

<< "3. XOR operation on selected byte string with another\n"

<< "4. Check inclusion of another string in selected\n"

<< "5. Check if amount of 1's in byte strings is equal\n"

<< "6. Return amount of 1's in the string\n"

<< "7. Shift bits left\n"

<< "8. Shift bits right\n"

<< "9. Flip bits on selected bit string\n"

<< "Any other number returns to main menu automatically\n"

<< "Choice: ";

cin >> k;

if (!cin) throw runtime\_error("Failed to get input");

if (k <= 5 && k >= 1)

{

if (listByte.size() < 2) throw runtime\_error("Less than 2 byte strings in list");

if (listByte.size() > 2)

{

printList(listByte, it);

it2 = listByte.end();

cout << "Select a second byte string: ";

do

{

if (it2 == it) cout << "The same byte string selected, select another: " << endl;

it2 = listByte.end();

if (cin >> i && i > 0 && (unsigned)i <= listByte.size())

{

--i;

it2 = listByte.begin();

advance(it2, i);

}

else throw runtime\_error("Incorrect input");

} while (it2 == listByte.end() || it2 == it);

}

else

{

for (it2 = listByte.begin(); it2 != listByte.end() && it2 == it; ++it2) {}

cout << "Selecting the only byte string left" << endl;

}

}

switch (k)

{

case 1:

\*it &= \*it2;

break;

case 2:

\*it |= \*it2;

break;

case 3:

\*it ^= \*it2;

break;

case 4:

cout << "Selected string " << (it->includes(\*it2) ? "includes" : "does not include") << " another string" << endl;

break;

case 5:

cout << "Selected strings " << (CompareOnes(\*it, \*it2) ? "have" : "do not have") << " the same amount of 1's" << endl;

break;

case 6:

cout << "Selected strings " << it->Ones() << " of 1's" << endl;

break;

case 7:

cout << "Shift by: ";

cin >> num1;

if (!cin) throw runtime\_error("Failed to get input");

\*it <<= num1;

break;

case 8:

cout << "Shift by: ";

cin >> num1;

if (!cin) throw runtime\_error("Failed to get input");

\*it >>= num1;

break;

case 9:

~\*it;

break;

default:

break;

}

k = 6;

break;

case 7:

if (listByte.size() < 2) throw runtime\_error("Less than 2 strings in list");

printList(listByte, it);

if (it == listByte.end())

{

cout << "Select first byte string: ";

if (cin >> i && i > 0 && (unsigned)i <= listByte.size())

{

--i;

it = listByte.begin();

advance(it, i);

}

else throw runtime\_error("Incorrect input");

}

if (listByte.size() > 2)

{

cout << "Select a second byte string: ";

do

{

it2 = listByte.end();

if (cin >> i && i > 0 && (unsigned)i <= listByte.size())

{

--i;

it2 = listByte.begin();

advance(it2, i);

}

else throw runtime\_error("Incorrect input");

if (it2 == it) cout << "The same string was selected, select another: ";

} while (it2 == listByte.end() || it2 == it);

}

else

{

for (it2 = listByte.begin(); it2 != listByte.end() && it2 == it; ++it2) {}

cout << "Selecting the only vector left" << endl;

}

if (\*it == \*it2)

{

cout << "Selected byte strings are equal" << endl;

break;

}

cout << (\*it > \* it2 ? "First" : "Second") << " string is bigger than the other" << endl;

break;

case 8:

cout

<< "1. Add byte string\n"

<< "2. Print list of byte strings\n"

<< "3. Select byte string to manipulate\n"

<< "4. Delete selected byte\_string\n"

<< "5. Add operation on byte strings to list\n"

<< "6. Operate on selected byte string\n"

<< "7. Compare two byte strings\n"

<< "8. Print the list again\n"

<< "9. Clear list of byte strings\n"

<< endl;

break;

case 9:

if (listByte.empty()) throw list\_empty();

listByte.clear();

it = it2 = listByte.end();

cout << "List clear" << endl;

break;

case 0:

break;

default:

cout << "Incorrect menu choice" << endl;

}

}

catch (list\_empty& err)

{

cout << err.what() << endl;

}

catch (runtime\_error& err)

{

cerr << err.what() << endl;

}

catch (...)

{

cerr << "Exception thrown, but undefined. Please open an issue at GitHub." << endl;

}

} while (k);

return 0;

}

1. Вывод

Перегруженные операторы позволяют значительно облегчить работу между переменными пользовательского типа, а также упростить внешний вид программы за счет использования уже известных программисту операнд (см. суммирование строк в **<string>**) вместо функций типа *sum()*, *div()* и т.д. Тоже самое можно сказать и про пользовательские литералы.

1. Список литературы
2. Статьи по компьютерным наукам [Электронный ресурс]. URL: <https://www.geeksforgeeks.org/> (дата обращения: 28.09.2019).