Московский Авиационный Институт
(Национальный Исследовательский Университет)

Кафедра 806 «Вычислительная информатика и программирование»
Факультет: «Информационные технологии и прикладная математика»

Лабораторная работа
Дисциплина: «Объектно-ориентированное программирование»
III семестр
Задание 2: «Операторы, литералы»

Группа:	М8О-208Б-18, №6
Студент:	Хитриков Артемий Юрьевич
Преподаватель:	Журавлёв Андрей Андреевич
Оценка:	
Дата:	14.10.2019

Москва, 2019

1. Задание

(*вариант № 6*): Создать класс *BitString* для работы с 96-битовыми строками. Битовая строка должна быть представлена двумя полями: старшая часть unsigned long long (uint64_t), младшая часть unsigned int (uint32_t).
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.

2. Адрес репозитория на GitHub

https://github.com/DeZellt/oop_exercise_02

3. Код программы на C++

lab2.cpp

```
#include <iostream>
#include <cstdint>
#include "bitstring.h"

int main() {

  uint32_t bitleft;
uint32_t bitright;
```

```
uint32_t incfirst;
uint64_t incsecond;

  bitstring literal_test = "8928 1179048"_bitstring;   std::cout << "Строка, сконструированная с помощью пользовательского литерала:" << literal_test << std::endl;

  bitstring a;
  std::cout << "Введите данные в следующем формате: *32-битовая строка* + *64-битовая строка*.\n";
std::cin >> a;

  bitstring f;
  std::cout << "Введите данные для второй строки в следующем формате: *32-битовая строка* + *64-битовая строка*.\n";
std::cin >> f;

  std::cout << "Сейчас будут выведены первая строка и вторая строка." << std::endl;
  std::cout << a << std::endl;
  std::cout << f << std::endl;

  std::cout << "Сейчас будут проведены стандартные битовые операции с вашей строкой." << std::endl;
  std::cout << a << " & " << f << " = " << (a & f) << std::endl;
std::cout << a << " | " << f << " = " << (a | f) << std::endl;
std::cout << a << " ^ " << f << " = " << (a ^ f) << std::endl;
std::cout << "~(" << a << ") = " << ~a << std::endl;

  std::cout << "Введите через пробел кол-во битов, на которое требуется сделать сдвиг" << std::endl;
  std::cin >> bitleft >> bitright;
  std::cout << "Сейчас будет проведен сдвиг на " << bitleft << " битов влево и на " << bitright << " битов вправо. После сдвига будет печататься состояние строки." << std::endl;   a.shiftLeft(bitleft);   std::cout << a << std::endl;   a.shiftRight(bitright);   std::cout << a << std::endl;
```

```
  std::cout << "А теперь проверим еще функции. Для того, чтобы проверить функцию включения, введите еще одно число в следующем формате: *32битовая строка* + *64-битовая строка*." << std::endl;
  std::cin >> incfirst >> incsecond;   bitstring inc{incfirst, incsecond};
std::cout << "Кол-во единиц равно " << a.countone() << std::endl;
std::cout << a << " == " << inc << "? Это " << ((a == inc) ? "правда" : "неправда") << std::endl;   std::cout << a << " > " << inc << "? Это " << ((a > inc) ? "правда" : "неправда")
<< std::endl;   std::cout << a << " < " << inc << "? Это " << ((a < inc) ? "правда" : "неправда")
<< std::endl;    std::cout << "В " << a << " включено " << inc.getF() << " " << inc.getS() << "? Это " << (a.include(inc) ? "правда" : "неправда") << std::endl;
  return 0;
}


#ifndef D_BITSTRING_H_
#define D_BITSTRIGN_H_ 1

#include <iostream>
#include <cstdint>
#include <sstream>

class bitstring {

private:
    uint32_t m_first;
uint64_t m_second;
void shiftLeftOne();
void shiftRightOne();
int32_t findOne() const;
int32_t findTwo() const;
    int32_t length() const;

public:
    bitstring(uint32_t first=0, uint64_t second=0);
```

```
    bitstring operator&(const bitstring& r) const;      bitstring operator|(const
bitstring& r) const;      bitstring operator^(const bitstring& r) const;
bitstring operator~() const;      void shiftLeft(int32_t i);      void
shiftRight(int32_t i);      int32_t countone() const;      bool equal(const
bitstring& r) const;      bool operator==(const bitstring& r) const;      bool
operator>(const bitstring& r) const;      bool operator<(const bitstring& r)
const;      friend std::ostream& operator<<(std::ostream& os, const
bitstring& bts);      friend std::istream& operator>>(std::istream& is,
bitstring& bts);      bool include(const bitstring& r) const;      uint64_t getS()
const;      uint32_t getF() const;
    void print();
};

bitstring operator""_bitstring(const char* str, size_t);

#endif
```

bitstring.cpp

```
#include "bitstring.h"
#include <iostream>
#include <cstdint>

bitstring::bitstring(uint32_t first, uint64_t second)
{      m_first = first;      m_second = second;
}

int32_t bitstring::findOne() const { // ищет длину второго слова
int32_t secondlen = 0;
    for (int32_t i = 0; i < 64; i++) if (((m_second >> i) & 1) == 1) secondlen = i;
return ++secondlen;
}

int32_t bitstring::findTwo() const {
    int32_t firstlen = 0;
    for (int32_t i = 0; i < 32; i++) if (((m_first >> i) & 1) == 1) firstlen = i;
return ++firstlen;
}
```

```
int32_t bitstring::length() const {return (findOne() + findTwo());}

bitstring bitstring::operator&(const bitstring& r) const {
bitstring result{};
   result.m_first = m_first & r.m_first;
result.m_second = m_second & r.m_second;     return
result;
}

bitstring bitstring::operator|(const bitstring& r) const { // побитовое "ИЛИ"
bitstring result{};
   result.m_first    =    m_first    |    r.m_first;
result.m_second  =  m_second  |  r.m_second;
return result;
}

bitstring bitstring::operator^(const bitstring& r) const { // побитовое "ИСКЛ
ИЛИ"
   bitstring result{};
   result.m_first = m_first ^ r.m_first;
   result.m_second = m_second ^ r.m_second;
return result;
}

bitstring bitstring::operator~() const { // побитовое "НЕ"
bitstring result{};     result.m_first = ~m_first;
   result.m_second = ~m_second;
return result;
}

void bitstring::shiftLeft(int32_t i) { // сдвиг на произвольное кол-во бит влево
for (int32_t k=0; k<i; k++) shiftLeftOne(); }

void bitstring::shiftLeftOne() { // сдвиг влево на один бит
const uint64_t max = 0x80000000;
   m_first <<= 1;
   if ((m_second & max) == max) m_first++;
m_second <<= 1;
```

```
}

void bitstring::shiftRight(int32_t i) { // сдвиг вправо на i бит
for (int32_t k=0; k<i; k++) shiftRightOne();
}

void bitstring::shiftRightOne() { // сдвиг вправо на один бит
m_second >>= 1;
   if ((m_first & 1) == 1) m_second+= 0x80000000;
m_first >>= 1;
}

int32_t bitstring::countone() const { // считает кол-во единиц
int32_t count = 0;     bitstring temp{m_first, m_second};     for
(int32_t i = 0; i < 96; i++) {         if ((temp.getS() & 1) == 1)
count++;         temp.shiftRight(1);
   }
   return count;
}

bool bitstring::operator==(const bitstring& r) const {return(countone()
== r.countone());} bool bitstring::operator>(const bitstring& r) const
{return(countone() > r.countone());} bool bitstring::operator<(const
bitstring& r) const {return(countone() < r.countone());}

std::ostream& operator<<(std::ostream& os, const bitstring& bts){
return os << bts.getF() << " " << bts.getS() << std::endl;
}

std::istream& operator>>(std::istream& is, bitstring& bts){
return is >> bts.m_first >> bts.m_second;
}

bool bitstring::include(const bitstring& r) const {return (((m_first & r.getF()) ==
r.getF()) && ((m_second & r.getS()) == r.getS()));}

bitstring operator""_bitstring(const char* str, size_t) {
std::stringstream ss(str);       bitstring result;       ss >>
result;     return result;
```

```
}

uint64_t bitstring::getS() const {return m_second;}
uint32_t bitstring::getF() const {return m_first;}
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.10.2) project(lab2)
set(CMAKE_CXX_STANDARD 14)
add_executable(lab2
 lab2.cpp
    bitstring.cpp)

set(CMAKE_CXX_FLAGS
    "${CMAKE_CXX_FLAGS} -Wall -Wextra")
```

4. Входные данные

Входные данные должны удовлетворять условиям, заданным лабораторной работе. Для данного варианта входные данные – две пары чисел, у обеих пар первое число формата unsigned int, второе unsigned long long.

Test01.txt

```
31 15
15 31
5 3
43 17
```

Test02.txt

```
0 1
13 169
4 2
0 0
```

Test03.txt

478 12839
1223 83829
3 1
119 1

5. Результаты выполнения тестов

Result_test01.txt

Строка, сконструированная с помощью пользовательского литерала:8928 1179048
Введите данные в следующем формате: *32-битовая строка* + *64-битовая строка*.
31 15
Введите данные для второй строки в следующем формате: *32-битовая строка* + *64-битовая строка*.
15 31
Сейчас будут выведены первая строка и вторая строка.
31 15
15 31
Сейчас будут проведены стандартные битовые операции с вашей строкой.
31 15 & 15 31 = 15 15
31 15 | 15 31 = 31 31
31 15 ^ 15 31 = 16 16
~(31 15) = 4294967264 18446744073709551600
Введите через пробел кол-во битов, на которое требуется сделать сдвиг
5 3
Сейчас будет проведен сдвиг на 5 битов влево и на 3 битов вправо. После сдвига будет печататься состояние строки.
992 480
3 3758096385
А теперь проверим еще функции. Для того, чтобы проверить функцию включения, введите еще одно число в следующем формате: *32-битовая строка* + *64-битовая строка*.
43 17
Кол-во единиц равно 9
31 15 == 43 17? Это неправда

31 15 > 43 17? Это правда
31 15 < 43 17? Это неправда
В 31 15 включено 43 17? Это неправда

Result_test02.txt

Строка, сконструированная с помощью пользовательского литерала:8928 1179048
Введите данные в следующем формате: *32-битовая строка* + *64-битовая строка*.
0 1
Введите данные для второй строки в следующем формате: *32-битовая строка* + *64-битовая строка*.
13 169
Сейчас будут выведены первая строка и вторая строка.
0 1
13 169
Сейчас будут проведены стандартные битовые операции с вашей строкой.
0 1 & 13 169 = 0 1
0 1 | 13 169 = 13 169
0 1 ^ 13 169 = 13 168
~(0 1) = 4294967295 18446744073709551614
Введите через пробел кол-во битов, на которое требуется сделать сдвиг
4 2
Сейчас будет проведен сдвиг на 4 битов влево и на 2 битов вправо. После сдвига будет печататься состояние строки.
0 16
0 0
А теперь проверим еще функции. Для того, чтобы проверить функцию включения, введите еще одно число в следующем формате: *32-битовая строка* + *64-битовая строка*.
0 0
Кол-во единиц равно 1
0 1 == 0 0? Это неправда
0 1 > 0 0? Это правда
0 1 < 0 0? Это неправда
В 0 1 включено 0 0? Это правда

Result_test03.txt

Строка, сконструированная с помощью пользовательского литерала:8928 1179048

Введите данные в следующем формате: *32-битовая строка* + *64-битовая строка*.

478 12839

Введите данные для второй строки в следующем формате: *32-битовая строка* + *64-битовая строка*.

1223 83829

Сейчас будут выведены первая строка и вторая строка.

478 12839

1223 83829

Сейчас будут проведены стандартные битовые операции с вашей строкой.

478 12839 & 1223 83829 = 198 549

478 12839 | 1223 83829 = 1503 96119

478 12839 ^ 1223 83829 = 1305 95570

~(478 12839) = 4294966817 18446744073709538776

Введите через пробел кол-во битов, на которое требуется сделать сдвиг

3 1

Сейчас будет проведен сдвиг на 3 битов влево и на 1 битов вправо. После сдвига будет печататься состояние строки.

3824 102712

239 6419

А теперь проверим еще функции. Для того, чтобы проверить функцию включения, введите еще одно число в следующем формате: *32-битовая строка* + *64-битовая строка*.

119 1

Кол-во единиц равно 14

478 12839 == 119 1? Это неправда

478 12839 > 119 1? Это правда

478 12839 < 119 1? Это неправда

В 478 12839 включено 119 1? Это неправда

6. Вывод

Перегрузка операторов служит полезным инструментом для любого программиста. Благодаря перегрузке код программы становится более читаемым. Пользовательские литералы, в свою очередь, задают точную настройку для строк, с которыми мы работаем – делает работу с программой более эффективной.