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-108Б-18, №6
Студент:	Васильева Василиса Евгеньевна
Преподаватель:	Журавлёв Андрей Андреевич
Оценка:	
Дата:	14.10.2019

Москва, 2019

1. Задание

Создать класс BitString для работы с 96-битовыми строками. Битовая строка должна быть представлена двумя полями: старшая часть unsigned long long, младшая часть unsigned int. 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/vasilisavasileva/oop_excercise_2

3. Код программы на C++

main.cpp

main.cpp

```
#include <iostream>
#include<locale>
#include "Bitstring.h"


int main() {

  setlocale(LC_ALL, "rus");
  int m, n;
  bool l;
  std::cout << "Введите значения строки 1\n";
  Bitstring BS1;
  std::cin >> BS1;
  std::cout << "Введите значения строки 2\n";
  Bitstring BS2;
  std::cin >> BS2;
  std::cout << "Введите количество битов для сдвига\n";
  std::cin >> n;
  std::cout << "Первая строка\n";
  std::cout << BS1 <<std::endl;
  std::cout << "Вторая строка\n";
  std::cout << BS2 <<std::endl;
  std::cout << "and\n";
  std::cout << (BS1 & BS2) <<std::endl;
  std::cout << "or\n";
  std::cout << (BS1 | BS2) <<std::endl;
  std::cout << "xor\n";
```

```
  std::cout << (BS1 ^ BS2) <<std::endl;
  std::cout << "not\n";
  std::cout <<  ~BS1 <<std::endl;
  std::cout << "BS1 shiftleft\n";
  std::cout << (BS1 << n) <<std::endl;
  std::cout << "BS2 shiftleft\n";
  std::cout << (BS2 << n) <<std::endl;
  std::cout << "BS1 shiftRight\n";
  std::cout << (BS1 >> n) <<std::endl;
  std::cout << "BS2 shiftRight\n";
  std::cout << (BS2 >> n) <<std::endl;
  std::cout << "count units BS1\n";
  std::cout << (BS1.counter()) <<std::endl;
  std::cout << "count units BS2\n";
  std::cout << (BS2.counter()) <<std::endl;
  std::cout << "comparing units\n";
  std::cout<< (BS1.compare(BS2)) <<std::endl;
  std::cout << "includes BS1 BS2\n";
  std::cout << (BS1.includes(BS2)) <<std::endl;
  //--------------123456789-123456789-123456789-123
  Bitstring BS4 = 10000000000000000000000000000000_bs;
  std::cout<<BS4<<std::endl;
  return 0;
}
```

Bitstring.h

```
#ifndef BITSTRING_H
#define BITSTRING_H
#include <inttypes.h>
#include <iostream>
class Bitstring {
 private:
  uint64_t b1;
  uint32_t b2;
 public:
  Bitstring();
  Bitstring(uint64_t b1, uint32_t b2);
  friend Bitstring operator& (const Bitstring& a, const Bitstring& b);
  friend Bitstring operator| (const Bitstring& a, const Bitstring& b);
  friend Bitstring operator^ (const Bitstring& a, const Bitstring& b);
  friend Bitstring operator~ (const Bitstring& a);
  friend Bitstring operator<< (const Bitstring& l, int m);
  friend Bitstring operator>> (const Bitstring& l, int m);
  int counter() const;
  friend bool operator< (const Bitstring& a, const Bitstring& b);
  friend bool operator> (const Bitstring& a, const Bitstring& b);
  friend bool operator== (const Bitstring& a, const Bitstring& b);
  bool compare(const Bitstring& bs2) const;
  bool includes(const Bitstring& bs2) const;
  friend std::istream& operator>>(std::istream& in, Bitstring& Bs);
  friend std::ostream& operator<<(std::ostream& out, const Bitstring&
Bs);
};
Bitstring operator "" _bs(const char* str);
#endif
```

Bitstring.cpp

```
#include "Bitstring.h"
#include <iostream>
#include <inttypes.h>
#include<cstring>

Bitstring::Bitstring() {
  this->b1 = 0;
  this->b2 = 0;
}

Bitstring::Bitstring(uint64_t b1, uint32_t b2) {
  this->b1 = b1;
  this->b2 = b2;
}

Bitstring operator&(const Bitstring& a, const Bitstring& b) {
  Bitstring bs3{ (a.b1)&(b.b1),(a.b2)&(b.b2) };
  return bs3;
}

Bitstring operator|(const Bitstring& a, const Bitstring& b) {
  Bitstring bs3 = Bitstring( (a.b1) | (b.b1),(a.b2) | (b.b2) );
  return bs3;
}

Bitstring operator^ (const Bitstring& a, const Bitstring& b) {
  Bitstring bs3{ (a.b1) ^ (b.b1),(a.b2) ^ (b.b2) };
  return bs3;
}

Bitstring operator~ (const Bitstring& a) {
  Bitstring bs3{ ~(a.b1),~(a.b2) };
  return bs3;
}

Bitstring operator<<(const Bitstring& l, int m) {
  uint32_t a, t2 = l.b2;
  uint64_t t1 = l.b1;
  a = 1;
  a <<= 31;
  for (int i = 0; i < m; i++) {
 if (((t2)&a) > 0) {
t1 <<= 1;
t2 <<= 1;
      t1 = t1 + 1;
    }
    else {
t1 <<= 1;
t2 <<= 1;
    }
}
  return Bitstring(t1,t2);
}
```

```
Bitstring operator>>(const Bitstring& l, int m) {
  uint64_t a, t1 = l.b1;
  uint32_t b, t2 = l.b2;
b = 1;
b <<= 31;
a = 1;
  for (int i = 0; i < m; i++) {
    if (((t1)&a) > 0) {
 t1 >>= 1;
      t2 >>= 1;
      t2 = t2 + b;
    }
    else {
      t1 >>= 1;
      t2 >>= 1;
}
  }
  return Bitstring(t1,t2);
}

int Bitstring::counter() const{
  uint64_t a = 1;
uint32_t b = 1;
uint64_t l;
uint32_t l1;
int count = 0;

for (int i = 0; i < 63; i++) {
    l = (this->b1)&a;
 if (l != 0) {
      ++count;
}
    a <<= 1;
  }
  for (int i = 0; i < 32; i++) {
    l1 = (this->b2)&b;
if (l1 != 0) {
      ++count;
    }
    b <<= 1;
  }
  return count;
}


bool Bitstring::compare(const Bitstring& bs2) const {
int a = this->counter();
  int b = bs2.counter();
if (a == b)
return true;
  return false;

}

bool Bitstring::includes(const Bitstring& bs2) const {
  if (((this->b1)&(bs2.b1)) == bs2.b1)
```

```
    if (((this->b2)&(bs2.b2)) == bs2.b2)
return true;
  return false;

}



bool operator< (const Bitstring& a, const Bitstring& b) {
if ((a.b1) < (b.b1))
return true;
else if ((a.b1) > (b.b1))
return false;
}


bool operator> (const Bitstring& a, const Bitstring& b) {
if ((a.b1) > (b.b1))
return true;
else if ((a.b1) < (b.b1))
return false;
if ((a.b1) == (b.b1))
    if ((a.b2) > (b.b2))
return true;
else
return false;
}

bool operator== (const Bitstring& a, const Bitstring& b) {
return (((a.b1) == (b.b1)) && ((a.b2) == (b.b2)));
}


std::ostream& operator<<(std::ostream& out, const Bitstring& Bs){
  uint64_t a = 1;
  a <<= 63;
  uint32_t b = 1;
b <<= 31;
for (int i = 0; i < 64; i++) {
out << ((a & Bs.b1) > 0);
a >>= 1;
}
  for (int i = 0; i < 32; i++) {
out << ((b & Bs.b2) > 0);
b >> 1;
}
  return out;
}

std::istream& operator>>(std::istream& in, Bitstring& Bs){
  in>>Bs.b1>>Bs.b2;
  return in;
}



Bitstring operator "" _bs(const char* str) {
```

```
  int size = strlen(str);
  if(size > 96) exit(1);
  uint64_t a = 0;
  uint32_t b = 0;
  int i = size-1;
  for(i; i>=size-32 && i >=0;i--){
    b += (str[i] - '0') << (size-1-i);
  }
  for(i; i>=0;i--){
    a += (str[i] - '0')<<(size-33- i);
  }
  Bitstring result = Bitstring(a, b);
  return result;
}
```

CmakeLists.txt

cmake_minimum_required(VERSION 2.8) # Проверка версии CMake.
 # Если версия установленой программы
 # старее указаной, произойдёт аварийный выход.

project(2lab) # Название проекта

set(SOURCE_EXE main.cpp) # Установка переменной со списком исходников для исполняемого файла

set(SOURCE_LIB Bitstring.cpp) # Тоже самое, но для библиотеки

add_library(bitstring STATIC ${SOURCE_LIB}) # Создание статической библиотеки с именем foo

add_executable(main ${SOURCE_EXE}) # Создает исполняемый файл с именем main

target_link_libraries(main bitstring)

4. Результаты выполнения тестов

Тест 1
Введите значения строки 1
1
1
Введите значения строки 2
1
1
Введите количество битов для сдвига
1

Первая строка
00000000000000000000000000000000000000000000000000000000000001000000000000000000000000000001
Вторая строка
00000000000000000000000000000000000000000000000000000000000001000000000000000000000000000001
and
00000000000000000000000000000000000000000000000000000000000001000000000000000000000000000001
or
00000000000000000000000000000000000000000000000000000000000001000000000000000000000000000001
xor
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
not
11111111111111111111111111111111111111111111111111111111011111111111111111111111111110
BS1 shiftleft
00000000000000000000000000000000000000000000000000000000000010000000000000000000000000000010
BS2 shiftleft
00000000000000000000000000000000000000000000000000000000000010000000000000000000000000000010
BS1 shiftRight
00000000000000000000000000000000000000000000000000000000000001000000000000000000000000000001
BS2 shiftRight
00000000000000000000000000000000000000000000000000000000000001000000000000000000000000000001
count units BS1
2
count units BS2
2
comparing units
1
includes BS1 BS2
1
00000000000000000000000000000000000000000000000000000000000010000000000000000000000000000000

Тест 2

Введите значения строки 1
0

3
Введите значения строки 2
0
11
Введите количество битов для сдвига
3
Первая строка
00000000000000000000000000000000000000000000000000000000000000000000000000000011
Вторая строка
00000000000000000000000000000000000000000000000000000000000000000000000000001011
and
00000000000000000000000000000000000000000000000000000000000000000000000000000011
or
00000000000000000000000000000000000000000000000000000000000000000000000000001011
xor
00000000000000000000000000000000000000000000000000000000000000000000000000001000
not
11111111111111111111111111111111111111111111111111111111111111111111111111111100
BS1 shiftleft
00000000000000000000000000000000000000000000000000000000000000000000000000011000
BS2 shiftleft
00000000000000000000000000000000000000000000000000000000000000000000000001011000
BS1 shiftRight
00000000000000000000000000000000000000000000000000000000000000000000000000000011
BS2 shiftRight
00000000000000000000000000000000000000000000000000000000000000000000000000001011
count units BS1
2
count units BS2
3
comparing units
0
includes BS1 BS2
0

00000000000000000000000000000000000000000000000000000000000010
00000000000000000000000000

Тест 3

Введите значения строки 1
11
66
Введите значения строки 2
4
8
Введите количество битов для сдвига
3
Первая строка
00000000000000000000000000000000000000000000000000000101100
00000000000000000000001000010
Вторая строка
00000000000000000000000000000000000000000000000000010000
00000000000000000000000001000
and
00000000000000000000000000000000000000000000000000000000000
00000000000000000000000000
or
00000000000000000000000000000000000000000000000000001111000
00000000000000000000001001010
xor
00000000000000000000000000000000000000000000000000001111000
00000000000000000000001001010
not
11111111111111111111111111111111111111111111111111101001111111
11111111111111110111101
BS1 shiftleft
00000000000000000000000000000000000000000000000000101100000
00000000000000000001000010000
BS2 shiftleft
00000000000000000000000000000000000000000000000000010000000
00000000000000000000001000000
BS1 shiftRight
00000000000000000000000000000000000000000000000000000101100
00000000000000000000001000010
BS2 shiftRight
00000000000000000000000000000000000000000000000000000010000
00000000000000000000000001000
count units BS1
5

count units BS2
2
comparing units
0
includes BS1 BS2
0
00000000000000000000000000000000000000000000000000000000000010
00000000000000000000000000000

5. Объяснение результатов работы программы

Программа просит на вход значения битовых строк, которые представляются четырьмя десятичными числами, по два на каждую строку. Литерал, описанный в моей программе, создает из входных данных два объекта типа Bitstring. Потом над объектом производятся действия согласно моему варианту. Эти действия совершаются при помощи перегруженных операторов, таких как and, not, xor и т.д.

6. Вывод

В ходе работы я познакомилась с пользовательскими литералами и принципом их работы. Так же я освоила перегрузку операторов и поняла, насколько более удобной становится работа с объектами классов при их применении.