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 1: «Простые классы»

Группа:	М8О-206Б-18, №27
Студент:	Шорохов Алексей Павлович
Преподаватель:	Журавлёв Андрей Андреевич
Оценка:	
Дата:	

Москва, 2019

Задание

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.

Адрес репозитория на GitHub

Код программы на C++

```
cmake_minimum_required(VERSION 3.2)

project(BitString)

add_executable(BitString
        Source.cpp
        BitString.cpp)

set_property(TARGET BitString PROPERTY CXX_STANDART 11)
```

BitString.cpp

```
#include "BitString.h"
#include <stdlib.h>
#include <iostream>
#include <string>
#include <vector>

BitString::BitString() {

   firstHalf = 0;
   secondHalf = 0;

}

BitString::BitString(unsigned long long first, unsigned long long second) {
   firstHalf = first;
   secondHalf = second;
}
```

```
void BitString::Enter() {
    std::string str;

    std::cout << '\n' << "Enter string" << '\n';
    std::cin >> str;

    std::string sec(str.size(), '0');
    std::vector<int> v;

    while (str != sec) {
        int a = 0;
        for (int i = 0; i < str.size(); i++) {
            a *= 10;
            a += str[i] - '0';
            str[i] = char('0' + a / 2);
            a %= 2;
        }
        v.push_back(a);
    }



    unsigned long long shs = 1;

    for (int i = 0; i < 64 && i < v.size(); i++) {
        secondHalf += v[i] * shs;
        shs *= 2;
    }

    unsigned long long fhs = 1;

    for (int i = 64; i < v.size(); i++) {
        firstHalf += v[i] * fhs;
        fhs *= 2;
    }
}

BitString BitString::_not() {

    BitString bs;

    bs.firstHalf = ~(firstHalf);
    bs.secondHalf = ~(secondHalf);

    return bs;
```

```
}


BitString BitString::_and(const BitString &bs) {

   BitString bs1;
   bs1.firstHalf = firstHalf & bs.firstHalf;
   bs1.secondHalf = secondHalf & bs.secondHalf;

   return bs1;

}


BitString BitString::_or(const BitString &bs) {

   BitString bs1;
   bs1.firstHalf = firstHalf | bs.firstHalf;
   bs1.secondHalf = secondHalf | bs.secondHalf;

   return bs1;

}


BitString BitString::_xor(const BitString &bs) {

   BitString bs1;
   bs1.firstHalf = firstHalf ^ bs.firstHalf;
   bs1.secondHalf = secondHalf ^ bs.secondHalf;

   return bs1;

}

void BitString::shiftLeft(unsigned long long n) {

   unsigned long long pow63 = 1;
   for (int i = 0; i < 63; i++) {
      pow63 *= 2;
   }

   for (int i = 0; i < n; i++) {                    //110100111 << 3 ==
100111000
      firstHalf = firstHalf << 1;
```

```
            if (secondHalf >= pow63) {
                firstHalf += 1;
            }
            secondHalf = secondHalf << 1;
        }
    }

    void BitString::shiftRight(unsigned long long n) {                    //110100111 >> 3
    == 000110100

        unsigned long long pow63 = 1;
        for (int i = 0; i < 63; i++) {
            pow63 *= 2;
        }
        for (int i = 0; i < n; i++) {
            secondHalf = secondHalf >> 1;
            if (firstHalf % 2 == 1) {
                secondHalf += pow63;
            }
            firstHalf = firstHalf >> 1;
        }


    }

    unsigned long long BitString::posBitNumber(){

        BitString bs1;
        bs1.firstHalf = firstHalf;
        bs1.secondHalf = secondHalf;

        unsigned long long number = 0;

        while (bs1.firstHalf != 0) {
            if (bs1.firstHalf % 2 == 1) number++;
            bs1.firstHalf /= 2;
        }

        while (bs1.secondHalf != 0) {
            if (bs1.secondHalf % 2 == 1) number++;
            bs1.secondHalf /= 2;
        }

        return number;
```

```
}

int BitString::compPosBitNumber(BitString &bs) {
    unsigned long long thisNumber = posBitNumber();
    unsigned long long bsNumber = bs.posBitNumber();

    if (thisNumber > bsNumber) return 0;
    if (thisNumber < bsNumber) return 1;
    else return 2;

}

void BitString::isArgInThis(const BitString &bs) {

    BitString lbs;
    lbs.firstHalf = firstHalf;
    lbs.secondHalf = secondHalf;
    BitString sbs;
    sbs.firstHalf = bs.firstHalf;
    sbs.secondHalf = bs.secondHalf;

    std::vector<int> vflbs;
    std::vector<int> vfsbs;

    std::vector<int> vslbs;
    std::vector<int> vssbs;


    while (lbs.firstHalf != 0) {
        vflbs.push_back(lbs.firstHalf % 2);
        lbs.firstHalf /= 2;
    }


    for (int i = vflbs.size(); i < 64 ; i++) {
        vflbs.push_back(0);
    }

    while (lbs.secondHalf != 0) {
        vslbs.push_back(lbs.secondHalf % 2);
        lbs.secondHalf /= 2;
    }
    for (int i = vslbs.size(); i < 64; i++) {
        vslbs.push_back(0);
    }
```

```
while (sbs.firstHalf != 0) {
    vfsbs.push_back(sbs.firstHalf % 2);
    sbs.firstHalf /= 2;
}

for (int i = vfsbs.size(); i < 64 ; i++) {
    vfsbs.push_back(0);
}


while (sbs.secondHalf != 0) {
    vssbs.push_back(sbs.secondHalf % 2);
    sbs.secondHalf /= 2;
}
for (int i = vssbs.size(); i < 64; i++) {
    vssbs.push_back(0);
}



for (int i = 0; i < vfsbs.size() && i < vflbs.size(); i++) {

    if (vfsbs[i] == 1 && vflbs[i] != 1) {
        std::cout << "NO\n";
        return;
    }
}



for (int i = 0; i < vssbs.size() && i < vslbs.size(); i++) {
    if (vssbs[i] == 1 && vslbs[i] != 1) {
        std::cout << "NO\n";
        return;
    }
}

std::cout << "YES\n";
```

```
}

void BitString::print() {

   BitString bs1;
   bs1.firstHalf = firstHalf;
   bs1.secondHalf = secondHalf;

   std::vector<int> v;

   while (bs1.firstHalf != 0) {
      v.push_back(bs1.firstHalf % 2);
      bs1.firstHalf /= 2;
   }

   for (int i = 0; i < 64 - v.size(); i++) {
      std::cout << 0;
   }

   for (int i = v.size() - 1; i >= 0; i--) {
      std::cout << v[i];
   }
   v.clear();

   std::cout << " ";

   while (bs1.secondHalf != 0) {
      v.push_back(bs1.secondHalf % 2);
      bs1.secondHalf /= 2;
   }
   for (int i = 0; i < 64 - v.size(); i++) {
      std::cout << 0;
   }

   for (int i = v.size() - 1; i >= 0; i--) {
      std::cout << v[i];
   }
   std::cout << '\n';

}
```

BitString.h

```
#include <iostream>
```

```
#include <string>

class BitString
{
public:

   BitString();
   BitString(unsigned long long first, unsigned long long second);

   void Enter();

   BitString _not();
   BitString _and(const BitString &bs);
   BitString _or(const BitString &bs);
   BitString _xor(const BitString &bs);

   void shiftLeft(unsigned long long n);
   void shiftRight(unsigned long long n);

   unsigned long long posBitNumber();
   int compPosBitNumber(BitString &bs);

   void isArgInThis(const BitString &bs);

   void print();

private:
   unsigned long long firstHalf;
   unsigned long long secondHalf;


};
```

Source.cpp

```
#include "BitString.h"

int main(int argc, char** argv) {

   BitString bs;

   bs.Enter();

   BitString bs1;
```

```
    bs1.Enter();

    bs.print();
    bs1.print();

    BitString bsTest = bs._not();

    std::cout << "not first number:\n";

    bsTest.print();

    bsTest = bs._and(bs1);

    std::cout << "first and second:\n";

    bsTest.print();

    bsTest = bs._or(bs1);

    std::cout << "first or second:\n";

    bsTest.print();

    bsTest = bs._xor(bs1);

    std::cout << "first xor second:\n";

    bsTest.print();

    std::cout << "Positive Bit Number of First is " << bs.posBitNumber() << '\n';

    if (bs.compPosBitNumber(bs1) == 0) {
        std::cout << "Bit Comparence of first and second shows that first is larger\n";
    } else if (bs.compPosBitNumber(bs1) == 1) {
        std::cout << "Bit Comparence of first and second shows that second is
larger\n";
    } else {
        std::cout << "Bit Comparence of first and second shows that they are
equal\n";
    }

    std::cout << "Is second in first? : ";
    bs.isArgInThis(bs1);

    int shift;
    std::cout << "Enter number of bits to shift first number left and second right : ";
    std::cin >> shift;

    bs.shiftLeft(shift);

    std::cout << "Shifted first left : \n";
    bs.print();

    bs1.shiftRight(shift);

    std::cout << "Shifted second right : \n";
    bs1.print();




    return 0;
}
```

<u>file01.test</u>
```
18446744073709551627
11
3
```

<u>file02.test</u>
```
18446744073709551615
156
10
```

<u>Результаты тестов</u>

<u>1</u>
```
Enter string
18446744073709551627

Enter string
11
0000000000000000000000000000000000000000000000000000000000000001
0000000000000000000000000000000000000000000000000000000000001011
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000001011
not first number:
1111111111111111111111111111111111111111111111111111111111111110
1111111111111111111111111111111111111111111111111111111111110100
```

```
first and second:
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000001011
first or second:
0000000000000000000000000000000000000000000000000000000000000001
0000000000000000000000000000000000000000000000000000000000001011
first xor second:
0000000000000000000000000000000000000000000000000000000000000001
0000000000000000000000000000000000000000000000000000000000000000
Positive Bit Number of First is 4
Bit Comparence of first and second shows that first is larger
Is second in first? : YES
Enter number of bits to shift first number left and second right : 3
Shifted first left :
0000000000000000000000000000000000000000000000000000000000001000
0000000000000000000000000000000000000000000000000000000001011000
Shifted second right :



Enter string
18446744073709551615

Enter string
156
0000000000000000000000000000000000000000000000000000000000000000
1111111111111111111111111111111111111111111111111111111111111111
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000010011100
not first number:
1111111111111111111111111111111111111111111111111111111111111111
0000000000000000000000000000000000000000000000000000000000000000
first and second:
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000010011100
first or second:
0000000000000000000000000000000000000000000000000000000000000000
1111111111111111111111111111111111111111111111111111111111111111
first xor second:
0000000000000000000000000000000000000000000000000000000000000000
1111111111111111111111111111111111111111111111111111111101100011
Positive Bit Number of First is 64
Bit Comparence of first and second shows that first is larger
Is second in first? : YES
Enter number of bits to shift first number left and second right : 10
```

```
Shifted first left :
0000000000000000000000000000000000000000000000000000001111111111
1111111111111111111111111111111111111111111111111111110000000000
Shifted second right :
```

Объяснение результатов

Программа получает на вход две строки, содержащие числа, которые далее преобразует в 128-битовые строки и выполняет преобразования, требуемые задание лабораторной работы.

Вывод

Были изучены основы ООП и заложен фундамент для будущей учебы и последующего применения знаний в работе.