

Московский Авиационный Институт  
(Национальный исследовательский Университет)

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

**Лабораторная работа  
по курсу «ООП»**

**Тема:**  
**Операторы, литералы.**

Студент:	Козлов А.Д.
Группа:	М80-206Б-18
Преподаватель:	Журавлев А.А.
Вариант:	7
Оценка:	
Дата:	

Москва  
2019

## 1. Код программы на языке C++:

### bigString.hpp:

```
#ifndef BIGSTRING_HPP
```

```
#define BIGSTRING_HPP
```

```
struct BigString{
```

```
    BigString();
```

```
    BigString(unsigned long long n1, unsigned long long n2);
```

```
    BigString(const BigString&);
```

```
    ~BigString();
```

```
    void ReadBS(std::istream&);
```

```
    void WriteBS(std::ostream&);
```

```
    void set_lstring(unsigned long long n);
```

```
    void set_rstring(unsigned long long n);
```

```
    unsigned long long get_lstring() const;
```

```
    unsigned long long get_rstring() const;
```

```
    char get_bit(int) const;
```

```
    void print_all_bits();
```

```
    BigString operator &(const BigString& s) const;
```

```
    BigString operator |(const BigString& s) const;
```

```
    BigString operator ^(const BigString& s) const;
```

```
    BigString operator !() const;
```

```
    BigString& operator<<(int);
```

```
    BigString& operator>>(int);
```

```
    bool operator ==(const BigString& s) const;
```

```
    bool operator !=(const BigString& s) const;
```

```
    bool operator >(const BigString& s) const;
```

```
    bool operator <(const BigString& s) const;
```

```
    const BigString& operator =(const BigString& s);
```

```
    int num_of_units() const;
```

```
    BigString* comparison(BigString& s);
```

```
    bool is_include(const BigString& s) const;
```

```
private:
```

```
    unsigned long long lString;
```

```
    unsigned long long rString;
```

```
};
```

```
unsigned long long pow_m(int, int);
```

```
BigString operator "" _0XtoBS(const char*, size_t);
```

```
#endif
```

### **bigString.cpp:**

```
#include <iostream>
```

```
#include "../includes/bigString.hpp"
```

```
BigString::BigString():lString(0), rString(0){}
```

```
BigString::BigString(unsigned long long n1, unsigned long long n2):lString(n1),  
rString(n2){}
```

```
BigString::BigString(const BigString& s) {  
    this->lString = s.get_lstring();  
    this->rString = s.get_rstring();  
    std::cout << "Конструктор копирования\n";  
}
```

```
BigString::~BigString(){} 
```

```
void BigString::ReadBS(std::istream& in) {  
    in >> this->lString;  
    in >> this->rString;  
}
```

```
void BigString::WriteBS(std::ostream& out) {  
    out << "Bits: ";  
    for(int i = 0; i < 128; ++i) {  
        out << this->get_bit(i);  
    }  
    out << "\n";  
    out << "Первое поле числа: ";  
    out << this->lString;  
    out << " ";  
    out << "Второе поле поле числа: ";  
    out << this->rString;  
    out << "\n";  
}
```

```
void BigString::set_lstring(unsigned long long n) {  
    this->lString = n;  
}
```

```
void BigString::set_rstring(unsigned long long n) {  
    this->rString = n;  
}
```

```
unsigned long long BigString::get_lstring() const {
```

```

    return this->lString;
}

unsigned long long BigString::get_rstring() const {
    return this->rString;
}

char BigString::get_bit(int index) const{
    if(index >= 0 && index < 128) {
        unsigned long long mask = 0x8000000000000000; // первый бит 1 остальные
0
        char bit;
        if(index < 64) {
            mask = mask >> index;
            if(this->lString & mask) {
                bit = '1';
            } else {
                bit = '0';
            }
        } else {
            mask = mask >> (index - 64);
            if(this->rString & mask) {
                bit = '1';
            } else {
                bit = '0';
            }
        }
        return bit;
    }
}

void BigString::print_all_bits() {
    for(int i = 0; i < 128; ++i) {
        std::cout << this->get_bit(i);
    }
    std::cout << "\n";
}

BigString BigString::operator &(const BigString& s) const{
    BigString res;
    res.set_lstring(this->lString & s.get_lstring());
    res.set_rstring(this->rString & s.get_rstring());
    return res;
}

BigString BigString::operator |(const BigString& s) const{
    BigString res;

```

```

    res.set_lstring(this->lString | s.get_lstring());
    res.set_rstring(this->rString | s.get_rstring());
    return res;
}
BigString BigString::operator ^(const BigString& s) const{
    BigString res;
    res.set_lstring(this->lString ^ s.get_lstring());
    res.set_rstring(this->rString ^ s.get_rstring());
    return res;
}
BigString BigString::operator !() const{
    BigString res;
    res.set_lstring(~(this->lString));
    res.set_rstring(~(this->rString));
    return res;
}
BigString& BigString::operator <<(int shift) {
    unsigned long long Mask_firstBit = 0x8000000000000000; // первый бит 1
    остальные 0
    BigString res;
    int bit;
    for(int i = 0; i < shift; ++i) {
        if(this->rString & Mask_firstBit) {
            bit = 1;
        }
        else {
            bit = 0;
        }
        this->rString = this->rString << 1;
        this->lString = this->lString << 1;
        this->lString = this->lString | bit;
    }
    return *this;
}
BigString& BigString::operator >>(int shift) {
    unsigned long long Mask_lastBit = 1; // последний бит 1 остальные 0
    int bit;
    for(int i = 0; i < shift; ++i) {
        if (this->lString & Mask_lastBit) {
            bit = 1;
        }
        else {
            bit = 0;
        }
        this->lString = this->lString >> 1;
    }

```

```

        this->rString = this->rString >> 1;
        if (bit) {
            this->rString = this->rString | 0x8000000000000000;
        }
    }
    return *this;
}

bool BigString::operator ==(const BigString& s) const{
    return this->lString == s.get_lstring() && this->rString == s.get_rstring();
}

bool BigString::operator !=(const BigString& s) const{
    return !(this->lString == s.get_lstring() && this->rString == s.get_rstring());
}

bool BigString::operator >(const BigString& s) const{
    if(this->lString > s.get_lstring()) {
        return true;
    } else if(this->lString == s.get_lstring()) {
        if(this->rString > s.get_rstring()) {
            return true;
        }
    }
    return false;
}

bool BigString::operator <(const BigString& s) const{
    if(this->rString < s.get_rstring()) {
        return true;
    } else if(this->rString == s.get_rstring()) {
        if(this->lString < s.get_lstring()) {
            return true;
        }
    }
    return false;
}

const BigString& BigString::operator =(const BigString& s) {
    this->lString = s.get_lstring();
    this->rString = s.get_rstring();
    return s;
}

int BigString::num_of_units() const {
    unsigned long long mask = 0x8000000000000000;
    int counter = 0;
    for(int i = 0; i < 64; ++i) {
        if (this->lString & mask) {
            counter++;
        }
    }
}

```

```

    }
    mask = mask >> 1;
}
mask = 0x8000000000000000;
for(int i = 0; i < 64; ++i) {
    if (this->rString & mask) {
        counter++;
    }
    mask = mask >> 1;
}
return counter;
}

```

```

BigString* BigString::comparison(BigString& s) {
    int value1 = this->num_of_units();
    int value2 = s.num_of_units();
    if (value1 >= value2) {
        return this;
    }
    else {
        return &s;
    }
}

```

```

bool BigString::is_include(const BigString& s) const {
    if(((this->lString & s.get_lstring()) == this->get_lstring()) || ((this->rString &
s.get_rstring()) == this->get_rstring())) {
        return true;
    }
    else {
        return false;
    }
}

```

```

unsigned long long pow_m(int a, int b) {
    unsigned long long res = 0;
    for(int i = 0; i < b; ++i) {
        if(res == 0) {
            res = 1;
        }
        res = res*a;
    }
    return res;
}

```

```

BigString operator ""_0XtoBS(const char* str, size_t size) {
    BigString res;
    if(size != 37) {
        std::cout << "He to1\n";
        return res;
    }
    char sym;
    unsigned long long n = 0;
    int k = 15;
    for(int i = 0; i < 37; ++i) {
        std::cout << "k: " << k << std::endl;
        sym = str[i];
        std::cout << i << ") Sym : " << sym << std::endl;
        if((sym >= '0' && sym <= '9')) {
            n = n + (sym - '0') * pow_m(16, k);
            --k;
        } else if(sym == 'A') {
            n = n + 10 * pow_m(16, k);
            --k;
        } else if(sym == 'B') {
            n = n + 11 * pow_m(16, k);
            --k;
        } else if(sym == 'C') {
            n = n + 12 * pow_m(16, k);
            --k;
        } else if(sym == 'D') {
            n = n + 13 * pow_m(16, k);
            --k;
        } else if(sym == 'E') {
            n = n + 14 * pow_m(16, k);
            --k;
        } else if(sym == 'F') {
            n = n + 15 * pow_m(16, k);
            --k;
        } else if(sym == ' ') {
            res.set_lstring(n);
            k = 15;
        } else if(sym == 'X') {
            n = 0;
            k = 15;
        } else {
            res.set_lstring(0);
            res.set_rstring(0);
            return res;
        }
    }
}

```



```
}  
res.set_rstring(n);  
return res;  
}
```

**main.cpp:**

```
#include <iostream>
#include <fstream>
#include "../includes/bigString.hpp"

void separator();
void meny();
unsigned long long get_num_in_file(std::ifstream& fin);

int main(int argc, char* argv[]) {
    BigString str1;
    BigString str2;
    int key;
    std::cout << "Ввод:\n\t1)Консоль\n\t2)Файл\n";
    std::cin >> key;
    std::cout << "=====\n";
    if(key == 1) {
        std::cout << "Первая строка: ";
        str1.ReadBS(std::cin);
        std::cout << "Вторая строка: ";
        str2.ReadBS(std::cin);
    } else if(key == 2) {
        if(argc < 2) {
            std::cout << "Не указан файл!\n";
            return 1;
        }
        std::ifstream fin(argv[1]);
        if(!fin.is_open()) {
            std::cout << "Не удалось открыть файл!\n";
            return 2;
        }
        str1.ReadBS(fin);
        str2.ReadBS(fin);
        fin.close();
    } else {
        std::cout << "Стандартное действие\n\t1)---0XF800000000000000\n\t2)---
0X1800000000000000\n";
        str1.set_lstring(0XF800000000000000);
        str1.set_rstring(0X1800000000000000);
        str2.set_lstring(0XF800000000000000);
        str2.set_rstring(0X1800000000000000);
    }
    char R = '!';
    int shift = 0;
    std::cout << "=====\n";
```

```

while(R != EOF) {
    meny();
    std::cout << "=====\n";
    std::cin >> R;
    if(R == '1') {
        str1.WriteBS(std::cout);
    } else if(R == '2') {
        str2.WriteBS(std::cout);
    } else if(R == '3') {
        std::cin >> shift;
        str1 >> shift;
        str1.WriteBS(std::cout);
        str2 >> shift;
        str2.WriteBS(std::cout);
    } else if(R == '4') {
        str1 << shift;
        str1.WriteBS(std::cout);
        str2 << shift;
        str2.WriteBS(std::cout);
    } else if(R == '5') {
        str1.WriteBS(std::cout);
        str2.WriteBS(std::cout);
        (str1 & str2).WriteBS(std::cout);
    } else if(R == '6') {
        (str1 | str2).WriteBS(std::cout);
    } else if(R == '7') {
        (str1 ^ str2).WriteBS(std::cout);
    } else if(R == '8') {
        (!str1).WriteBS(std::cout);
        (!str2).WriteBS(std::cout);
    } else if(R == '9') {
        std::cout << "str1 : " << str1.num_of_units();
    } else if(R == 'A') {
        (str1.comparison(str2))->WriteBS(std::cout);
    } else if(R == 'B') {
        if(str1.is_include(str2)) {
            std::cout << "str1 включено в str2\n";
        } else {
            std::cout << "str1 не включено в str2\n";
        }
    } else if(R == 'C') {
        if(str1 == str2) {
            std::cout << "Равны\n";
        } else {
            std::cout << "Не равны\n";
        }
    }
}

```

```

    }
} else if(R == 'D') {
    if(str1 != str2) {
        std::cout << "Не равны\n";
    } else {
        std::cout << "Равны\n";
    }
} else if(R == 'E') {
    if(str1 > str2) {
        std::cout << "str1 больше str2\n";
    } else {
        std::cout << "str1 не больше str2\n";
    }
} else if(R == 'F') {
    if(str1 < str2) {
        std::cout << "str1 меньше str2\n";
    } else {
        std::cout << "str1 не меньше str2\n";
    }
} else {
    break;
}
std::cout << "=====\n";
}
return 0;
}

```

```

void meny() {
    std::cout << "1)Print str1" << std::endl;
    std::cout << "2)Print str2" << std::endl;
    std::cout << "3)shiftLeft" << std::endl;
    std::cout << "4)shiftLeft" << std::endl;
    std::cout << "5)AND" << std::endl;
    std::cout << "6)OR" << std::endl;
    std::cout << "7)XOR" << std::endl;
    std::cout << "8)NOT" << std::endl;
    std::cout << "9)num_of_units" << std::endl;
    std::cout << "A)comparison" << std::endl;
    std::cout << "B)is_include" << std::endl;
    std::cout << "C)==" << std::endl;
    std::cout << "D)!=" << std::endl;
    std::cout << "E)>" << std::endl;
    std::cout << "F)<" << std::endl;
}

```

```

unsigned long long get_num_in_file(std::ifstream& fin) {
    unsigned long long num = 0;
    char ch;
    while(fin.get(ch)) {
        if (ch != ' ' && ch != '\n' && ch != EOF) {
            num = num * 10 + (ch - '0');
        }
        else {
            break;
        }
    }
    return num;
}

```

### **CmakeLists.txt:**

```
cmake_minimum_required(VERSION 3.1)
```

```
project(oop_exercise_01)
```

```
add_executable(oop_exercise_01 main.cpp bigString.cpp)
```

### **2. Ссылка на репозиторий на GitHub.**

[https://github.com/ArtemKD/oop\\_exercise\\_02](https://github.com/ArtemKD/oop_exercise_02)

### **3. Набор testcases.**

test\_01.txt:

```

12868426913406380000 5193880475836733000
12918000598641058000 11398999006967146000

```

test\_02.txt:

```

11093947366930840000 12809315413885479000
8591448501686567000 7091160512692316000

```

test\_03.txt:

```

5647118480700488000 15870949720163300000
13670168416534604000 8221050084544006000

```

test\_04.txt:

```

11023188960974280000 10712243251891943000
15260885590755232000 11658364395506348000

```

test\_05.txt:

```

17490701296889190000 1907006900119359500
696737765169393700 17953908319343747000

```

#### 4. Результаты выполнения тестов.

Пример для теста test\_01.txt:

12868426913406380000 519388 0475836733000

12918000598641058000 1139по8999006967146000

menu:

#### 5. Объяснение результатов работы программы

- 1) При запуске программы oop\_exercise\_02 выводится меню с выбором формы заполнения полей объекта класса BigString
- 2) Если выбран ключ 1, то ввод выполняется со стандартного потока ввода, если выбран ключ 2, то выполняется ввод из файла указанного файла, указанного в аргументах.
- 3) Эти файлы должны иметь вид название вида test\_??.txt. Программа должна получать значения объекта класса BigString из этого текстового файла.
- 3) Программа выполняет операции: &, |, ^, !, для строк полученных из тестов или стандартного вывода.
- 4) Побитовый сдвиг строк str1 и str2 вычисляется с помощью перегрузки операторов >> и <<, затем выводится в выбранный поток вывода с помощью функции WriteBS.
- 5) Вычисление количества единиц в строке осуществляется с помощью функции num\_of\_units; функция возвращает значение типа int и выводится в стандартный вывод с помощью std::cout.
- 6) Сравнение строк по количеству единиц осуществляется с помощью функции comparison, которая использует предыдущую функцию num\_of\_units; функция возвращает указатель на объект класса BigString, затем с помощью метода класса WriteBS выводим слово с наибольшим количеством единиц в стандартный вывод.
- 7) Проверка включения осуществляется с помощью функции is\_include.
- 8) Операции сравнения возвращают значения true и false.
- 9) Реализован пользовательский литерал, который выделяет из строки 2 числа, записанных в 16-ной системе счисления, для записи их в поля объекта класса с типом unsigned long long.

#### 6. Вывод.

Выполняя данную лабораторную я получил опыт работы с перегрузками операторов и ознакомился с принципом создания пользовательского литерала. Создал класс, соответствующий варианту моего задания, реализовал для него битовые операции: «И»(&), «ИЛИ»(|), «ОТРИЦАНИЕ»(!), «Исключающее или»(^), а также операции побитового сдвига(<< и >>) и операции сравнения (<, >, ==, !=).

