**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 1**

Тема: Простые классы на языке С++

Студент: Пермяков Никита Александрович

Группа: 80-207

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. Постановка задачи

* Создать класс BitString для работы с 96-битовыми строками.
* Битовая строка должна быть представлена двумя полями:

1) старшая часть unsigned long long,

2) 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.

.

1. Описание программы

Класс BitString

* Инициализация с двумя полями
  + unsigned long long
  + unsigned int
* Перегруженные операторы
  + сложения
  + произведения
  + сравнения
  + вычитания
  + отрицания
* Функция вывода сообщения со списком команд
* Функция main
* Функция проверки хранимых значений
* Функция сравнения количества битов
* Функция вывода значений
* Функция включения
* Функции
  + сдвиг влево
  + сдвиг вправо
  + циклический влево
  + циклический вправо

Класс TestCase

* тесты на длину вводимой строки
* тесты на размер вводимой строки
* тесты на допустимые символы

Подсчет результата с помощью перегруженных функций

Вывод результатов

1. Набор тестов

01011010001

01101100010

1

2

3

4

5

1

6

4

7

3

8

2

9

10

11

12

111010000011010

011100111001110

11

10

9

8

7

7

4

6

9

5

0

4

3

2

1

12

1. Результаты выполнения тестов

root@dell:/mnt/c/Users/permi/source/repos/draft/3# ./a.out

Введите битовую строку:

01011010001

Введите битовую строку:

01101100010

1 - сложить битовые строки

2 - перемножить битовые строки

3 - сложить по модулю битовые строки

4 - выполнить побитовое отрицание строк

5 - выполнить побитовый сдвиг вправо строк

6 - выполнить побитовый сдвиг влево строк

7 - выполнить циклический сдвиг строк вправо

8 - выполнить циклический сдвиг строк влево

9 - узнать состояние строк

10 - запустить тесты внутреннего представления

11 - очистить экран

12 - выйти

1

Сумма равна: 01111110011

2

Произведение равно: 01001000000

3

Сумма по модулю равна: 00110110011

4

Побитовое отрицание первой строки равно: 10100101110

Побитовое отрицание второй строки равно: 10010011101

5

На сколько двигать строки?

1

Результаты побитового сдвига:

первая строка - 01010010111

вторая строка - 01001001110

6

На сколько будем двигать строки?

4

Результаты побитового сдвига:

первая строка - 01011100000

вторая строка - 00111010000

7

На сколько будем двигать строки?

3

Результаты побитового сдвига:

первая строка - 11010100101

вторая строка - 10110010011

8

На сколько будем двигать строки?

2

Результаты побитового сдвига:

первая строка - 10010111010

вторая строка - 01001110110

9

Первая строка 10100101110

Вторая строка 10010011101

10

Result test at length - OK

Result test at size - OK

11

12

root@dell:/mnt/c/Users/permi/source/repos/draft/3# ./a.out

Введите битовую строку:

111010000011010

Введите битовую строку:

011100111001110

1 - сложить битовые строки

2 - перемножить битовые строки

3 - сложить по модулю битовые строки

4 - выполнить побитовое отрицание строк

5 - выполнить побитовый сдвиг вправо строк

6 - выполнить побитовый сдвиг влево строк

7 - выполнить циклический сдвиг строк вправо

8 - выполнить циклический сдвиг строк влево

9 - узнать состояние строк

10 - запустить тесты внутреннего представления

11 - очистить экран

12 - выйти

10

Result test at length - OK

Result test at size - OK

9

Первая строка 111010000011010

Вторая строка 011100111001110

8

На сколько будем двигать строки?

7

Результаты побитового сдвига:

первая строка - 000110101110100

вторая строка - 110011100111001

7

На сколько будем двигать строки?

4

Результаты побитового сдвига:

первая строка - 101011101000001

вторая строка - 111001110011100

6

На сколько будем двигать строки?

9

Результаты побитового сдвига:

первая строка - 011010000000000

вторая строка - 001110000000000

5

На сколько двигать строки?

0

Результаты побитового сдвига:

первая строка - 111010000011010

вторая строка - 011100111001110

4

Побитовое отрицание первой строки равно: 000101111100101

Побитовое отрицание второй строки равно: 100011000110001

3

Сумма по модулю равна: 100110111010100

2

Произведение равно: 000001000100001

1

Сумма равна: 100111111110101

12

1. Листинг программы

/\*

Пермяков Никита

github nikit34

Создать класс BitString для работы с 96-битовыми строками.

Битовая строка должна быть представлена двумя полями:

1) старшая часть unsigned long long,

2) 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.

\*/

#include<iostream>

#include<string>

#include<assert.h>

const unsigned int UNIT\_N = 28;

const unsigned int LENGTH\_A = 20;

const unsigned int LENGTH\_B = 10;

const unsigned int SIZE\_A = 64;

const unsigned int SIZE\_B = 32;

class TestCase {

public:

template <typename Ta, typename Tb>

void test\_field\_length(Ta a, Tb b) const {

int length\_a = std::to\_string(a).length();

int length\_b = std::to\_string(b).length();

if (length\_a < length\_b)

std::swap(length\_a, length\_b);

assert(("\nTest Error: overflow for [a b] by [length]\n", length\_a < LENGTH\_A && length\_b < LENGTH\_B));

std::cout << "\nResult test at length - OK" << std::endl;

};

template <typename Ta, typename Tb>

void test\_field\_size(Ta a, Tb b) const {

int size\_a = std::to\_string(a).size();

int size\_b = std::to\_string(b).size();

if (size\_a < size\_b)

std::swap(size\_a, size\_b);

assert(("\nTest Error: overflow for [a b] by [size]\n", size\_a < SIZE\_A && size\_b < SIZE\_B));

std::cout << "\nResult test at size - OK" << std::endl;

};

};

class BitString {

public:

BitString() {

this->unit = nullptr;

this->count = 0;

this->bits = { 1, 1 };

};

struct fields {

unsigned long long part\_large;

unsigned int part\_small;

} bits;

void recard\_fields() {

std::string str;

for(unsigned int i = 0; i < this->count; ++i)

str.push\_back(this->unit[i]);

if(this->count > LENGTH\_B - 1) {

const std::string str\_large = '1' + str.substr(0, this->count - LENGTH\_B + 1);

this->bits.part\_large = std::stoull(str\_large, nullptr, 10);

const std::string str\_small = '1' + str.substr(this->count - LENGTH\_B + 1, LENGTH\_B - 1);

this->bits.part\_small = std::stoul(str\_small, nullptr, 10);

}

else {

this->bits.part\_small = std::stoul('1' + str.substr(0, this->count), nullptr, 10);

}

};

class Error {

public:

Error() {

std::cout << std::endl << "Error: ";

};

Error(const std::string &msg) : Error() {

std::cout << msg << std::endl;

};

};

const int sizeBitString() const {

return this->count;

};

template <typename T>

void valid\_input\_length(T a) {

int length\_a = a.length();

assert(("\nTest Error: overflow for [a] by [length]\n", length\_a <= UNIT\_N));

}

void enter() {

std::string str;

std::cout << "Введите битовую строку: " << std::endl;

std::cin >> str;

this->count = str.size();

this->valid\_input\_length(str);

for(unsigned int i = 0; i < this->count; i++)

if (str[i] != '0' && str[i] != '1')

throw BitString::Error();

this->unit = new unsigned char[this->count];

for(unsigned int i = 0; i < this->count; i++)

this->unit[i] = str[i];

this->recard\_fields();

};

void show() {

for (unsigned int i = 0; i < this->count; i++)

std::cout << this->unit[i];

};

std::string notBit() {

std::string str;

for(unsigned int i = 0; i < this->count; i++) {

if(this->unit[i] == '0')

this->unit[i] = '1';

else

if(this->unit[i] == '1')

this->unit[i] = '0';

str += this->unit[i];

}

return str;

};

std::string andBit(BitString b) {

std::string str;

for (unsigned int i = 0; i < this->count; i++) {

if(this->unit[i] == '0' && b.unit[i] == '1')

str += '0';

else

if(this->unit[i] == '1' && b.unit[i] == '0')

str += '0';

else

str += this->unit[i];

}

return str;

};

std::string orBit(BitString b) {

std::string str;

for(unsigned int i = 0; i < this->count; i++) {

if(this->unit[i]== '0' && b.unit[i] == '1')

str += '1';

else

if(this->unit[i] == '1' && b.unit[i] == '0')

str += '1';

else

str += this->unit[i];

}

return str;

};

std::string xorBit(BitString b) {

std::string str;

for(unsigned int i = 0; i < this->count; i++) {

if(this->unit[i] == '0' && b.unit[i] == '1')

str += '1';

else

if(this->unit[i] == '1' && b.unit[i] == '0')

str += '1';

else

if(this->unit[i] == '0' && b.unit[i] == '0')

str += '0';

else

if(this->unit[i] == '1' && b.unit[i] == '1')

str += '0';

}

return str;

}

std::string rightShift(unsigned int &n) {

std::string str;

for (unsigned int i = 0; i < this->count; i++)

str += this->unit[i];

str.erase(this->count - n, n);

str.insert(0, n, '0');

return str;

}

std::string leftShift(unsigned int n) {

std::string str;

for (unsigned int i = 0; i < this->count; i++)

str += this->unit[i];

str.erase(0, n);

for (unsigned int i = 0; i < n; i++)

str.append("0");

return str;

}

std::string rightCycleShift(unsigned int n) {

std::string str;

for (unsigned int i = count - n; i < this->count; i++)

str += this->unit[i];

for (unsigned int i = 0; i < this->count - n; i++)

str += this->unit[i];

return str;

}

std::string leftCycleShift(unsigned int n) {

std::string str;

for (unsigned int i = n; i < this->count; i++)

str += this->unit[i];

for (unsigned int i = 0; i < n; i++)

str += this->unit[i];

return str;

}

std::string operator+(BitString b) {

return (\*this).orBit(b);

}

std::string operator\*(BitString b) {

return (\*this).andBit(b);

}

std::string operator/(BitString b) {

return (\*this).xorBit(b);

}

std::string operator-() {

return (\*this).notBit();

}

friend std::ostream& operator<< (std::ostream& cout, BitString item) {

cout << item.unit;

return cout;

}

private:

unsigned char\* unit;

unsigned int count;

};

int main() {

setlocale(LC\_ALL, "RUS");

TestCase tests;

BitString a, b;

try {

a.enter();

b.enter();

if (a.sizeBitString() != b.sizeBitString())

throw BitString::Error();

int key = 0;

bool flag\_loop = true;

while(flag\_loop) {

std::cout << std::endl

<< " 1 - сложить битовые строки" << std::endl

<< " 2 - перемножить битовые строки" << std::endl

<< " 3 - сложить по модулю битовые строки " << std::endl

<< " 4 - выполнить побитовое отрицание строк" << std::endl

<< " 5 - выполнить побитовый сдвиг вправо строк" << std::endl

<< " 6 - выполнить побитовый сдвиг влево строк " << std::endl

<< " 7 - выполнить циклический сдвиг строк вправо " << std::endl

<< " 8 - выполнить циклический сдвиг строк влево " << std::endl

<< " 9 - узнать состояние строк" << std::endl

<< " 10 - запустить тесты внутреннего представления" << std::endl

<< " 11 - очистить экран" << std::endl

<< " 12 - выйти"<<std::endl;

std::cin >> key;

switch(key) {

case 1: {

std::cout << std::endl << "Сумма равна: " << a + b << std::endl << std::endl;

a.recard\_fields();

break;

}

case 2: {

std::cout << std::endl << "Произведение равно: " << a \* b << std::endl << std::endl;

a.recard\_fields();

break;

}

case 3: {

std::cout << std::endl << "Сумма по модулю равна: " << a / b << std::endl << std::endl;

a.recard\_fields();

break;

}

case 4: {

std::cout << std::endl << "Побитовое отрицание первой строки равно: " << -a << std::endl

<< "Побитовое отрицание второй строки равно: " << -b << std::endl << std::endl;

a.recard\_fields();

break;

}

case 5: {

std::cout << " На сколько двигать строки?" << std::endl;

unsigned int n;

std::cin >> n;

if (n < 0)

throw BitString::Error("Invalid shift value");

std::cout << "Результаты побитового сдвига:" << std::endl

<< " первая строка - " << a.rightShift(n) << std::endl

<< " вторая строка - " << b.rightShift(n) << std::endl << std::endl;

a.recard\_fields();

b.recard\_fields();

break;

}

case 6: {

std::cout << " На сколько будем двигать строки?" <<std::endl;

unsigned int n;

std::cin >> n;

if (n < 0)

throw BitString::Error();

std::cout << "Результаты побитового сдвига:" << std::endl

<< " первая строка - " << a.leftShift(n) << std::endl

<< " вторая строка - " << b.leftShift(n) << std::endl << std::endl;

a.recard\_fields();

b.recard\_fields();

break;

}

case 7: {

std::cout << " На сколько будем двигать строки?" <<std::endl;

unsigned int n;

std::cin >> n;

if (n < 0)

throw BitString::Error();

std::cout << "Результаты побитового сдвига:" << std::endl

<< " первая строка - " << a.rightCycleShift(n) << std::endl

<< " вторая строка - " << b.rightCycleShift(n) << std::endl << std::endl;

a.recard\_fields();

b.recard\_fields();

break;

}

case 8: {

std::cout << " На сколько будем двигать строки?" <<std::endl;

unsigned int n;

std::cin >> n;

if (n < 0)

throw BitString::Error();

std::cout << "Результаты побитового сдвига:" << std::endl

<< " первая строка - " << a.leftCycleShift(n) << std::endl

<< " вторая строка - " << b.leftCycleShift(n) << std::endl << std::endl;

a.recard\_fields();

b.recard\_fields();

break;

}

case 9: {

std::cout << "Первая строка "; a.show(); std::cout << std::endl;

std::cout << "Вторая строка "; b.show(); std::cout << std::endl <<std::endl;

break;

}

case 10: {

tests.test\_field\_length(a.bits.part\_large, a.bits.part\_small);

tests.test\_field\_size(a.bits.part\_large, a.bits.part\_small);

break;

}

case 11: { system("cls"); break; }

case 12: { flag\_loop = false; break; }

}

}

}

catch(...){

BitString::Error e("Cout value invalid");

}

return 0;

}