**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 2**

Тема: Перегрузка операторов в С++

Студент: Хренов Геннадий

Группа: 80-207

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2019

1. Постановка задачи

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. Оперции 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.

1. Описание программы

Программа получает на вход 2 числа типа Bitstring, состоящие из 2 полей каждое(unsigned long long, unsigned long long, unsigned int, unsigned int). После вычислений программа выводит результаты операций: &,|,~,^,~, =, выполненные с помощью перегрузки операторов. Далее программа демонстрирует работу литерала, который приводит строку в нужный тип Bitstring.

Операторы & | ^ ~ = > < я перегрузил через дружественные функции, аргументы которых типа Bitstring. Сами функции проделывают данную операцию с составляющими unsigned long long и unsigned int, и возвращают переменную типа Bitstring. Также я сделал пользовательский литерал для работы с константами \_BS. Перед литералом пишется строка с разделителем «.», где перед точкой — первая часть unsigned long long, после точки — unsigned int. Часть до точки я преобразовываю в long long функцией atoll. Часть после точки записываю в отдельный массив чаров и преобразовываю в int функцией atoi.

1. Набор testcases

1)682 240 341 15

2)9999 1 8888 2

3) 0 0 0 0

1. Результаты выполнения тестов.

1)operation on bitstrings

enter 2 Bitstring(ull, ui, ull, ui)

682 240 341 15

A:

0000000000000000000000000000000000000000000000000000001010101010 00000000000000000000000011110000

B:

0000000000000000000000000000000000000000000000000000000101010101 00000000000000000000000000001111

A & B:

0000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000

A | B:

0000000000000000000000000000000000000000000000000000001111111111 00000000000000000000000011111111

A ^ B

0000000000000000000000000000000000000000000000000000001111111111 00000000000000000000000011111111

~A:

1111111111111111111111111111111111111111111111111111110101010101 11111111111111111111111100001111

A equal B (by number of units)

<12.128> with literal <\_BS> turn into class Bitstring:

0000000000000000000000000000000000000000000000000000000000001100 00000000000000000000000010000000

2)enter 2 Bitstring(ull, ui, ull, ui)

9999 1 8888 2

A:

0000000000000000000000000000000000000000000000000010011100001111 00000000000000000000000000000001

B:

0000000000000000000000000000000000000000000000000010001010111000 00000000000000000000000000000010

A & B:

0000000000000000000000000000000000000000000000000010001000001000 00000000000000000000000000000000

A | B:

0000000000000000000000000000000000000000000000000010011110111111 00000000000000000000000000000011

A ^ B

0000000000000000000000000000000000000000000000000000010110110111 00000000000000000000000000000011

~A:

1111111111111111111111111111111111111111111111111101100011110000 11111111111111111111111111111110

A is bigger than B (by number of units)

<12.128> with literal <\_BS> turn into class Bitstring:

0000000000000000000000000000000000000000000000000000000000001100 00000000000000000000000010000000

3)enter 2 Bitstring(ull, ui, ull, ui)

0 0 0 0

A:

0000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000

B:

0000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000

A & B:

0000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000

A | B:

0000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000

A ^ B

0000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000

~A:

1111111111111111111111111111111111111111111111111111111111111111 11111111111111111111111111111111

A equal B (by number of units)

<12.128> with literal <\_BS> turn into class Bitstring:

0000000000000000000000000000000000000000000000000000000000001100 00000000000000000000000010000000

1. Листинг программы

//Хренов Геннадий М8О-207Б

//реализовать класс Bitstring с перегрузкой операторов

#include<iostream>

using namespace std;

class Bitstring{

private:

unsigned long long x;

unsigned int y;

public:

Bitstring(unsigned long long part64, unsigned int part32): x(part64), y(part32)

{}

friend const Bitstring operator&(const Bitstring& w, const Bitstring& q);

friend const Bitstring operator|(const Bitstring& w, const Bitstring& q);

friend const Bitstring operator^(const Bitstring& w, const Bitstring& q);

friend const Bitstring operator~(const Bitstring& w);

friend Bitstring operator ""\_BS(const char\* i);

friend bool operator==(const Bitstring& w,const Bitstring& q);

friend bool operator>(const Bitstring& w,const Bitstring& q);

friend bool operator<(const Bitstring& w,const Bitstring& q);

friend int Count\_1 (unsigned long long x, unsigned int y);

void print(const Bitstring q) {

for(int i = 63; i >= 0; --i){

printf("%llu", (q.x >> i)&1);

}

cout << " ";

for(int i = 31; i >= 0; --i){

printf("%u", (q.y >> i)&1);

}

cout << '\n';

}

};

int Count\_1 (unsigned long long x, unsigned int y){

int res = 0;

while (x) {

res++;

x &= x-1; // Забираем младшую единичку.

}

while (y) {

res++;

y &= y-1; // Забираем младшую единичку.

}

return res;

}

const Bitstring operator&(const Bitstring& w, const Bitstring& q){

return Bitstring(w.x & q.x, w.y & q.y);

}

const Bitstring operator|(const Bitstring& w, const Bitstring& q){

return Bitstring(w.x | q.x, w.y | q.y);

}

const Bitstring operator^(const Bitstring& w, const Bitstring& q){

return Bitstring(w.x ^ q.x, w.y ^ q.y);

}

const Bitstring operator~(const Bitstring& w){

return Bitstring(~w.x, ~w.y);

}

bool operator==(const Bitstring& w,const Bitstring& q){

return (Count\_1(w.x, w.y) == Count\_1(q.x, q.y));

}

bool operator>(const Bitstring& w,const Bitstring& q){

return (Count\_1(w.x, w.y) > Count\_1(q.x, q.y));

}

bool operator<(const Bitstring& w,const Bitstring& q){

return (Count\_1(w.x, w.y) < Count\_1(q.x, q.y));

}

Bitstring operator ""\_BS(const char\* i){

int n = 0;

char integ[32];

while (i[n] != '.'){

++n;

}

for (int j = n + 1; j < strlen(i); j++){

integ[j - (n + 1)] = i[j];

}

return Bitstring(atoll(i),atoi(integ));

}

void Shiftleft(unsigned long long x, unsigned int y,int n){

x << n;

y << n;

}

void Shiftright(unsigned long long x, unsigned int y,int n){

x >> n;

y >> n;

}

void compare\_1(unsigned long long a, unsigned int b,unsigned long long x, unsigned int y){

if (Count\_1(a, b) == Count\_1(x, y)){

printf("equal\_1\n");

} else printf("not\_equal\_1\n");

}

void inclusion\_check(unsigned long long a, unsigned int b,unsigned long long x, unsigned int y){

if ((a | x) == a && (b | y) == b){

printf("it`s inclusion\n");

} else printf("it`s not inclusion\n");

}

int main() {

unsigned long long p1,p3;

unsigned int p2,p4;

cout << "operation on bitstrings\n";

cout << "enter 2 Bitstring(ull, ui, ull, ui)\n";

cin >> p1 >> p2 >> p3 >> p4;

Bitstring d1(p1,p2);

Bitstring d2(p3,p4);

Bitstring answ(0,0);

cout << "A:\n" << endl;

answ.print(d1);

cout << "B:\n" << endl;

answ.print(d2);

cout << "A & B:\n" << endl;

answ.print(d1 & d2);

cout << "A | B:\n" << endl;

answ.print(d1 | d2);

cout << "A ^ B" << endl;

answ.print(d1 ^ d2);

cout << "~A:\n" << endl;

answ.print(~d1);

if (d1 == d2){

cout << "A equal B (by number of units)\n";

}

if (d1 > d2){

cout << "A is bigger than B (by number of units)\n";

}

if (d1 < d2){

cout << "A is smaller B (by number of units)\n";

}

cout << " <12.128> with literal <\_BS> turn into class Bitstring:\n";

answ.print(12.128\_BS);

return 0;

}

6 Выводы:

Перегрузка операторов действительно важная и удобная вещь, с помощью которой можно определить операторы для своего типа данных, работающих как нам нужно. Перегрузить операторы можно как через дружеские функции, так и с помощью методов класса. Но стоит помнить, что некоторые операторы перегрузить нельзя: “::” “.” “.\*” “sizeof”,“typeid”.

СПИСОК ЛИТЕРАТУРЫ

1. Перегрузка операторов С++ [Электронный ресурс]. URL:<https://metanit.com/cpp/tutorial/5.14.php>

(дата обращения: 29.09.2019).

2. Битовые операции С++ [Электронный ресурс]. URL:

<http://www.c-cpp.ru/books/bitovye-operatory>

(дата обращения: 29.09.2019).