**Bài 13: Bitset - Khái niệm cơ bản - Sử dụng thư viện chuẩn STL cho C/C++**

**Đăng bởi: Admin | Lượt xem: 7329 | Chuyên mục: C/C++**

**Khái niệm :**

1 bitset dùng để lưu trữ các bit (các phần tử chỉ có hai giá trị là 0 hoặc 1, đúng hoặc sai,..)

Một bitset là một mảng của bool nhưng mỗi giá trị Boolean không được lưu trữ riêng thay vào đó bistet tối ưu hóa không gian sao cho mỗi bool chỉ chiếm 1 bit. Do đó, không gian được lấy bởi bitset nhỏ hơn bool bs [N] và bs vector ( N). Tuy nhiên, một giới hạn của bitset là N phải được biết tại thời điểm biên dịch, tức là một hằng số (giới hạn này không có ở đó với vectơ và mảng động)

Vì bitset lưu trữ cùng một thông tin theo cách nén, thao tác trên bitset nhanh hơn so với mảng và vectơ. Chúng ta có thể truy cập từng bit bitset riêng lẻ với sự trợ giúp của toán tử lập chỉ mục mảng [] đó là bs [3] hiển thị bit ở chỉ số 3 của bit bs giống như một mảng đơn giản. Ghi nhớ bitset bắt đầu lập chỉ mục ngược của nó là 10110, 0 ở các chỉ số 0 và 3 trong khi 1 ở các chỉ số thứ 2 và thứ 4.

Chúng ta có thể xây dựng một bitset bằng cách sử dụng số nguyên cũng như chuỗi nhị phân thông qua các hàm tạo được hiển thị trong mã bên dưới. Kích thước của bitset được cố định tại thời điểm biên dịch, nghĩa là nó có thể được thay đổi khi chạy.

Hàm chính được xác định cho lớp bitset là toán tử [], đếm, kích thước, đặt, đặt lại và nhiều thứ khác được giải thích trong mã bên dưới -

// C++ program to demonstrate various functionality of bitset

#**include** <bits/stdc++.h>

**using** **namespace** std;

#**define** M 32

**int** **main**()

{

// default constructor initializes with all bits 0

bitset<M> bset1;

// bset2 is initialized with bits of 20

bitset<M> **bset2**(20);

// bset3 is initialized with bits of specified binary string

bitset<M> **bset3**(string("1100"));

// cout prints exact bits representation of bitset

cout << bset1 << endl; // 00000000000000000000000000000000

cout << bset2 << endl; // 00000000000000000000000000010100

cout << bset3 << endl; // 00000000000000000000000000001100

cout << endl;

// declaring set8 with capacity of 8 bits

bitset<8> set8; // 00000000

// setting first bit (or 6th index)

set8[1] = 1; // 00000010

set8[4] = set8[1]; // 00010010

cout << set8 << endl;

// count function returns number of set bits in bitset

**int** numberof1 = set8.count();

// size function returns total number of bits in bitset

// so there difference will give us number of unset(0)

// bits in bitset

**int** numberof0 = set8.size() - numberof1;

cout << set8 << " has " << numberof1 << " ones and "

<< numberof0 << " zeros\n";

// test function return 1 if bit is set else returns 0

cout << "bool representation of " << set8 << " : ";

**for** (**int** i = 0; i < set8.size(); i++)

cout << set8.test(i) << " ";

cout << endl;

// any function returns true, if atleast 1 bit

// is set

**if** (!set8.any())

cout << "set8 has no bit set.\n";

**if** (!bset1.any())

cout << "bset1 has no bit set.\n";

// none function returns true, if none of the bit

// is set

**if** (!bset1.none())

cout << "bset1 has some bit set\n";

// bset.set() sets all bits

cout << set8.set() << endl;

// bset.set(pos, b) makes bset[pos] = b

cout << set8.set(4, 0) << endl;

// bset.set(pos) makes bset[pos] = 1 i.e. default

// is 1

cout << set8.set(4) << endl;

// reset function makes all bits 0

cout << set8.reset(2) << endl;

cout << set8.reset() << endl;

// flip function flips all bits i.e. 1 <-> 0

// and 0 <-> 1

cout << set8.flip(2) << endl;

cout << set8.flip() << endl;

// Converting decimal number to binary by using bitset

**int** num = 100;

cout << "\nDecimal number: " << num

<< " Binary equivalent: " << bitset<8>(num);

**return** 0;

}

Kết quả :

00000000000000000000000000000000

00000000000000000000000000010100

00000000000000000000000000001100

00010010

00010010 has 2 ones and 6 zeros

bool representation of 00010010 : 0 1 0 0 1 0 0 0

bset1 has no bit set.

11111111

11101111

11111111

11111011

00000000

00000100

11111011

Decimal number: 100 Binary equivalent: 01100100

Đối với tập bitset, chức năng đặt lại và lật được xác định. Đặt hàm thiết lập tất cả các bit của bitset nếu không có đối số nào được cung cấp nếu không nó sẽ đặt bit có vị trí được đưa ra làm đối số. Theo cách tương tự, thiết lập lại và lật cũng hoạt động nếu chúng được gọi mà không có đối số, chúng thực hiện thao tác trên toàn bộ bitset và nếu một số vị trí được cung cấp làm đối số thì chúng chỉ thực hiện thao tác tại vị trí đó.

Đối với bitset, tất cả toán tử bitwise đều bị quá tải, chúng có thể được áp dụng trực tiếp vào bitet mà không cần chuyển đổi hay chuyển đổi, toán tử bị quá tải chính là &, |, == ,! = Và toán tử dịch chuyển <> giúp thao tác trên bitet dễ dàng.

Sử dụng toán tử trên được hiển thị trong mã dưới đây.

// C++ program to show applicable operator on bitset.

#**include** <bits/stdc++.h>

**using** **namespace** std;

**int** **main**()

{

bitset<4> **bset1**(9); // bset1 contains 1001

bitset<4> **bset2**(3); // bset2 contains 0011

// comparison operator

cout << (bset1 == bset2) << endl; // false 0

cout << (bset1 != bset2) << endl; // true 1

// bitwise operation and assignment

cout << (bset1 ^= bset2) << endl; // 1010

cout << (bset1 &= bset2) << endl; // 0010

cout << (bset1 |= bset2) << endl; // 0011

// left and right shifting

cout << (bset1 <<= 2) << endl; // 1100

cout << (bset1 >>= 1) << endl; // 0110

// not operator

cout << (~bset2) << endl; // 1100

// bitwise operator

cout << (bset1 & bset2) << endl; // 0010

cout << (bset1 | bset2) << endl; // 0111

cout << (bset1 ^ bset2) << endl; // 0101

}

Kết quả :

0

1

1010

0010

0011

1100

0110

1100

0010

0111

0101

**Các hàm thông dụng trong bitset:**

**1 . Hàm All :**

Trả về True nếu tất cả các thành phần trong bitset đều là 1 và trả về false nếu có 1 phần tử khác 1

Ví dú :

// bitset::all

#**include** <iostream> // std::cin, std::cout, std::boolalpha

#**include** <bitset> // std::bitset

**int** **main** ()

{

std::bitset<8> foo;

std::cout << "Please, enter an 8-bit binary number: ";

std::cin >> foo;

std::cout << std::boolalpha;

std::cout << "all: " << foo.all() << '\n';

std::cout << "any: " << foo.any() << '\n';

std::cout << "none: " << foo.none() << '\n';

**return** 0;

}

Kết quả trả về :

Please, enter an 8-bit binary number: 11111111

all: true

any: true

none: false

**2 . Hàm Any :**

Trả về true nếu trong bitset có bất kì phần tử nào có giá trị là 1 và trả false trong trường hợp còn lại

Ví dụ :

// bitset::any

#**include** <iostream> // std::cin, std::cout

#**include** <bitset> // std::bitset

**int** **main** ()

{

std::bitset<16> foo;

std::cout << "Please, enter a binary number: ";

std::cin >> foo;

**if** (foo.any())

std::cout << foo << " has " << foo.count() << " bits set.\n";

**else**

std::cout << foo << " has no bits set.\n";

**return** 0;

}

Kết quả trả về :

Please, enter a binary number: 10110

0000000000010110 has 3 bits set.

**3. Hàm count :**

Hàm sẽ trả về số lượng phần tử bằng 1 trong bitset

Ví dụ :

// bitset::count

#**include** <iostream> // std::cout

#**include** <string> // std::string

#**include** <bitset> // std::bitset

**int** **main** ()

{

std::bitset<8> **foo** (std::string("10110011"));

std::cout << foo << " has ";

std::cout << foo.count() << " ones and ";

std::cout << (foo.size()-foo.count()) << " zeros.\n";

**return** 0;

}

Kết quả trả về :

10110011 has 5 ones and 3 zeros

**4. Hàm size :**

Trả về số lượng của các phần tử trong bitset

Code mẫu :

// bitset::size

#**include** <iostream> // std::cout

#**include** <bitset> // std::bitset

**int** **main** ()

{

std::bitset<8> foo;

std::bitset<4> bar;

std::cout << "foo.size() is " << foo.size() << '\n';

std::cout << "bar.size() is " << bar.size() << '\n';

**return** 0;

}

Kết quả trả về :

foo.size() **is** 8

bar.size() **is** 4