Game Programing Gems

[1.10]간단하고 빠른 비트 배열

2015 – 07 – 02 NHN NEXT 남세현

간단하고 빠른 비트 배열

비트 단위 연산

장점	단점
빠른 연산	에러, 실수 가능성
효율적인 용량	컴퓨터의 워드 크기에 의존

STL를 이용한 비트 배열

1. std::bitset<N>
2. std::vector<bool>

STL를 이용한 비트 배열(1)

1. std::bitset<N>
   - Compile때 N에 의해 고정됨.
   - 크기는 [N / word 사이즈]
     - 컴파일러마다 다를 수 있음

std::bitset 기본 사용법

```cpp
#include <bitset>
int _tmain(int argc, _TCHAR* argv[])
{
    std::bitset<5> myBitset;
    std::bitset<1024> bigBitset;
    std::cout << "size of myBitset(5) = " << sizeof(myBitset)
        << "\nsize of bigBitset(1024) = " << sizeof(bigBitset) << std::endl;

    myBitset = 5;               // 00101
    std::cout << myBitset.to_string() << std::endl;
    std::cout << myBitset.to_ulong() << std::endl;
    std::cout << myBitset.to_ullong() << std::endl;

    myBitset = 66;              // 00010
    std::cout << myBitset.to_string() << std::endl;
    std::cout << myBitset.to_ulong() << std::endl;
    std::cout << myBitset.to_ullong() << std::endl;

    myBitset[0] = 1;            // 00011
    // myBitset[0] = 123111313; // 00011
    // myBitset[0] = true;      // 00011

    std::cout << myBitset.to_string() << std::endl;
    std::cout << myBitset.to_ulong() << std::endl;
    std::cout << myBitset.to_ullong() << std::endl;

    myBitset[1] = false;    // 00001
    // myBitset[1] = 0;     // 00001

    std::cout << myBitset.to_string() << std::endl;
    std::cout << myBitset.to_ulong() << std::endl;
    std::cout << myBitset.to_ullong() << std::endl;
```

```
size of myBitset(5) = 4
size of bigBitset(1024) = 128
00101
5
5
00010
2
2
00011
3
3
00001
1
1
```

std::bitset 응용 (1)

```
enum MySetting
{
    MS_IS_MONSTER   = 0,
    MS_HAS_HP       = 1,
    MS_HI           = 2,
    MS_HAS_NO_IDEA  = 3,

    MS_MAX          = 4
};

std::bitset<MySetting::MS_MAX> mySettingBitset;

mySettingBitset[MySetting::MS_IS_MONSTER] = true;
mySettingBitset[MySetting::MS_HAS_HP] = false;
mySettingBitset[MySetting::MS_HI] = false;
mySettingBitset[MySetting::MS_HAS_NO_IDEA] = true;
```

std::bitset 응용 (2)

```
class boo
{
public:
    enum BooSetting
    {
        BS_1 = 0,
        BS_2 = 1,
        BS_3 = 2,
        BS_MAX = 3
    };

    bool GetAttribute(enum BooSetting attribute)
    {
        assert(attribute != BS_MAX);
        return m_Attributes[attribute];
    }

    void SetAttribute(enum BooSetting attribute, bool value)
    {
        assert(attribute != BS_MAX);
        m_Attributes[attribute] = value;
    }

    void ResetAttribute()
    {
        m_Attributes.reset();
    }

private:
    std::bitset<BooSetting::BS_MAX> m_Attributes;
};
```

```
boo foo;

foo.SetAttribute(boo::BooSetting::BS_1, 0);
foo.SetAttribute(boo::BooSetting::BS_3, true);

if (true == foo.GetAttribute(boo::BooSetting::BS_3))
    std::cout << "HELLO WORLD" << std::endl;
```

std::bitset 참고

```cpp
// bitset operators
#include <iostream>       // std::cout
#include <string>         // std::string
#include <bitset>         // std::bitset

int main ()
{
  std::bitset<4> foo (std::string("1001"));
  std::bitset<4> bar (std::string("0011"));

  std::cout << (foo^=bar) << '\n';       // 1010 (XOR,assign)
  std::cout << (foo&=bar) << '\n';       // 0010 (AND,assign)
  std::cout << (foo|=bar) << '\n';       // 0011 (OR,assign)

  std::cout << (foo<<=2) << '\n';        // 1100 (SHL,assign)
  std::cout << (foo>>=1) << '\n';        // 0110 (SHR,assign)

  std::cout << (~bar) << '\n';           // 1100 (NOT)
  std::cout << (bar<<1) << '\n';         // 0110 (SHL)
  std::cout << (bar>>1) << '\n';         // 0001 (SHR)

  std::cout << (foo==bar) << '\n';       // false (0110==0011)
  std::cout << (foo!=bar) << '\n';       // true  (0110!=0011)

  std::cout << (foo&bar) << '\n';        // 0010
  std::cout << (foo|bar) << '\n';        // 0111
  std::cout << (foo^bar) << '\n';        // 0101

  return 0;
}
```

Output:

```
1010
0010
0011
1100
0110
1100
0110
0001
0
1
0010
0111
0101
```

STL를 이용한 비트 배열(2)

2. std::vector<bool>
   - Runtime때 dynamic하게 사용
   - 일반적인 std::vector와 같음
   - Bit단위 연산을 해야 하므로, std::vector<bool>은 specialized version, 즉 라이브러리 측에서 따로 구현해놓음.

std::vector<bool> 기본 및 추가된 함수

```
std::cout << std::boolalpha;
std::vector<bool> mask;

mask.push_back(true);
mask.push_back(false);
mask.push_back(false);
mask.push_back(true);

std::cout << "mask contains:";
for (unsigned i = 0; i < mask.size(); i++)
    std::cout << ' ' << mask.at(i);
std::cout << '₩n';

mask.flip();

std::cout << "mask contains:";
for (unsigned i = 0; i < mask.size(); i++)
    std::cout << ' ' << mask.at(i);
std::cout << '₩n';
```

```
mask contains: true false false true
mask contains: false true true false
```

Endian 문제

From 20.5/3 (ISO/IEC 14882:2011)

> When converting between an object of class bitset and a value of some integral type, bit position pos corresponds to the bit value 1 << pos.

요약 : 신경 안써도 된다.

Network로 보낼 땐?

1. unsigned char 배열로 이루어 져 있다고 생각하기.

std::bitset<N> -> unsigned char[(N/8 + 1)]

- 패키징(serializaton)만 잘 해서 보내면 됨.

결론

C 스타일로 비트 연산 해서 관리하지 말고

- Int bit = 0;
  bit &= 1 << 3

C++ 스타일로 관리하자.

- std::bitset(N) myBitset;
  myBitset[3] = true;

C vs C++? 느려봤짜 얼마나 느리겠노