

Ch. 15 Bitwise Operations in C++

CS 2212 INTERMEDIATE PROGRAMMING/C++

Reading hex, OCT and returning to dec

Programs - C/C++ - D7 HexIO Demo/hexIOdemo.cpp - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help



```
hexIOdemo.cpp
2+ * hexIOdemo.cpp
7 #include <iostream>
8 #include <iomanip>
9 using namespace std;
10
11 int main( void )
12 {
13     unsigned short ui, xi;
14     cout << "\n Please enter an unsigned int: ";
15     cin >> dec >> ui; //dec is default, manipulator is shown for demonstration purposes.
16     cout << " = " << ui << " in decimal and = "
17         << hex << ui << " in hex, and = " //Hexadecimal (hex) manipulator used.
18         << oct << ui << " in octal.\n" << dec; //Then Octal manipulator (oct). Don't forget to put it back to decimal (dec).
19
20     cout << "\n Please enter an unsigned int in hex: ";
21     cin >> hex >> xi; // hmmm, what will cin read in next time?
22     cout << " = " << xi << " in decimal and = "
23         << hex << xi << " in hex, and = "
24         << oct << xi << " in octal.\n\n" << dec;
25
26     cout << "\n Please enter an unsigned int - oops, I left it in hex: ";
27     cin >> xi; // hmmm, what will cin read in next time?
28
29     cout << " = " << xi << " in decimal and = "
30         << hex << xi << " in hex, and = "
31         << oct << xi << " in octal.\n\n";
32
33     //what would happen if your forgot to put back in dec?
34     cout << " = " << xi << " Oops! Left it in octal.\n\n" << dec;
35
36     return 0;
37 }
```

← Error

← Error

Sample Output from hexiodemo.cpp

Programs - C/C++ - D7 HexIO Demo/hexIODemo.cpp - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

hexIODemo.cpp

```
20 cout << "\n Please enter an unsigned int in hex: ";
21 cin >>hex >>xi ; // hmmm, what will cin read in next time?
22 cout << " = " <<xi << " in decimal and = "
```

Problems Tasks Console Properties

<terminated> (exit value: 0) D7 HexIO Demo.exe [C/C++ Application] C:\Eclipse C CPP\Programs\D7 HexIO Demo\Debug\D7 HexIO Demo.exe

Please enter an unsigned int: 65535
= 65535 in decimal and = ffff in hex, and = 177777 in octal.

Please enter an unsigned int in hex: F1eb
= 61931 in decimal and = f1eb in hex, and = 170753 in octal.

Please enter an unsigned int - oops, I left it in hex: 23456
= 65535 in decimal and = ffff in hex, and = 177777 in octal.
= 177777 Oops! Left it in octal.

Maximum value

Upper and lower case accepted in hex.

Hex of 23456 exceeds the maximum value, so C++ gives you the max value.

Decoding an IP Address

Problem: Read a 32-bit internet IP address. Convert it and print it in the four-part dotted form that is customary for IP Addresses.

Input: A 32-bit integer in hex.

- For example: fa1254b9

Output: The dotted form of the address.

- For this example, the output is: 250.18.84.185

NOTE: The shift operation does not change the number of bits in the number.

If you start with a 32-bit number **and shift it 10 places to the left**, the result is a 32-bit number that has **lost its 10 high-order bits** and has had **10 zero bits inserted on the right side**.

Solution: Masking - bitwise

The mask. We need a mask to isolate the rightmost 8 bits of a long integer. We can write the constant with or without leading zeros, as 0xffL or 0x000000ffL, but we need the L to make it a long integer (32 bits).

0000 0000 0000 0000 0000 0000 1111 1111 is the mask.

To format the output so that all 8 hex digits are displayed, it is necessary to set a field width of 8 columns and to set the fill character to '0'. **The fill character will stay set until explicitly changed. BUT**, the field width must be set separately for each thing printed.

The address contains four values; each number is 8 bits (we will show it as a number between 0 to 255).

The byte mask lets us isolate the rightmost 8 bits from the address. In this box, we store each successive 8-bit field of the number into a `f#` variables.

part1 is from the leftmost 8 bits – shift 24 places

To get part1, the first field of the IP address, we shift the address 24 places to the right so that all but the first 8 (leftmost) bits are discarded.

Then we take the result of that number and the mask.

1111 10100001 0010 0101 0100 1011 1001 is the binary of fa1254b9 and in ipAddress

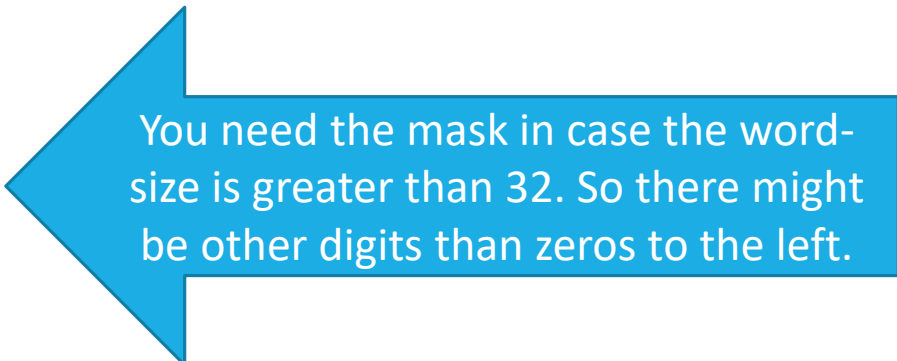
Shifting 24 to the right, leaves leftmost 8 bits

0000 0000 0000 0000 0000 0000 1111 1010

0000 0000 0000 0000 0000 0000 1111 1111 is the mask.

ipAddress & mask is

1111 1010 which is the decimal value $128 + 64 + 32 + 16 + 8 + 2 = 250$



You need the mask in case the word-size is greater than 32. So there might be other digits than zeros to the left.

part2 and part3 use smaller shifts and no shift for part4

To get part2 and part3, we shift the input by smaller amounts and mask out everything except the 8 bits on the right.

part2 is the result of shift right 16 times and mask.

0001 0010 and in decimal $16 + 2 = 18$

part3 is the result of shift right 8 times and mask.

0101 0100 and in decimal $64 + 16 + 4 = 84$

Since part4 is supposed to be the rightmost 8 bits of the IP address, we can mask the input directly, without shifting.

part4 is 1011 1001 and in decimal $128 + 32 + 16 + 8 + 1 = 185$

The IP address is 250.18.84.185 viola!

iomanip formatting output in C++

www.cplusplus.com/reference/iomanip/

Google UNH Bookmarks Apple Pi Robotics Pages

<iomanip>

IO Manipulators

Header providing parametric manipulators:

These go to the stream using >> or << direction operators.

fx Parametric manipulators

setiosflags	Set format flags (function)
resetiosflags	Reset format flags (function)
setbase	Set basefield flag (function)
setfill	Set fill character (function)
setprecision	Set decimal precision (function)
setw	Set field width (function)
get_money C++11	Get monetary value (function)
put_money C++11	Put monetary value (function)
get_time C++11	Get date and time (function)
put_time C++11	Put date and time (function)

Notice that non-parametric manipulators are declared directly in <ios>.

II. Applying Bit Operations

- ENCRYPTION – TRANSPOSITION CIPHER AT THE BITWISE LEVEL

```
8 #include <iostream>
9 using namespace std;
10 #define AE 0xE000 // bits 15:13 end up as 11:9 AE is 1110 0000 0000 0000
11 #define BE 0x1F00 // bits 12:8 end up as 4:0 BE is 0001 1111 0000 0000
12 #define CE 0x00F0 // bits 7:4 end up as 15:12 CE is 0000 0000 1111 0000
13 #define DE 0x000F // bits 3:0 end up as 8:5 DE is 0000 0000 0000 1111
14 unsigned short encrypt( unsigned short n );
15 int main( void )
16 {
17     short in;
18     unsigned short crypt;
19     cout <<"Enter a short number to encrypt: ";
20     cin >>in;
21     // Cast the int to unsigned before calling encrypt.
22     crypt = encrypt( (unsigned short) in );
23     cout <<"\nThe input number in base 10 (decimal) is: " <<in <<" \n"
24         <<"The input number in base 16 (hexadecimal) is: " <<hex <<in <<" \n\n"
25         <<"The encrypted number in base 10 is: " <<dec <<crypt <<"\n"
26         <<"The encrypted number in base 16 is: " <<hex <<crypt <<"\n\n";
27 }
```

```
28 // -----
29 unsigned short
30 encrypt( unsigned short n )
31 {
32     unsigned short a, b, c, d; // for the four parts of n
33     a = (n & AE) >> 4; // Isolate bits 15:13, shift to positions 11:9.
34     b = (n & BE) >> 8; // Isolate bits 12:8, shift to positions 4:0.
35     c = (n & CE) << 8; // Isolate bits 7:4, shift to positions 15:12.
36     d = (n & DE) << 5; // Isolate bits 3:0, shift to positions 8:5.
37     return c | a | d | b ; // Pack the four parts back together.
38 }
39
```

Output from encryption demo

Problems Tasks Console Properties

<terminated> (exit value: 0) D9EncryptionDemo.exe [C/C++ Application] C:\Eclipse C CPE

Enter a short number to encrypt: 23456

The input number in base 10 (decimal) is: 23456

The input number in base 16 (hexadecimal) is: 5ba0

The encrypted number in base 10 is: 42011

The encrypted number in base 16 is: a41b

Entered: 23456 hex 5ba0 binary 0101 1100 1010 0000

Walk through processing

`a = (n & AE) >> 4;`

```
      n = 0101 1100 1010 0000
& AE is 1110 0000 0000 0000
      -----
      0100 0000 0000 0000
>>4   0000 0100 0000 0000 is a
```

`b = (n & BE) >> 8;`

```
      n = 0101 1100 1010 0000
& BE is 0001 1111 0000 0000
      -----
      0001 1100 0000 0000
>> 8   0000 0000 0001 1100 is b
```

Walk through processing

`c = (n & CE) << 8;`

n =	0101	1100	1010	0000
& CE is	0000	0000	1111	0000

0000 0000 1010 0000

<<8 1010 0000 0000 0000 is c

`d = (n & DE) << 5;`

n =	0101	1100	1010	0000
& DE is	0000	0000	0000	1111

0000 0000 0000 0000

<<5 0000 0000 0000 0000 is d

Walk through processing

```
return c | a | d | b ; //use logical or to combine the 4 pieces
```

The order of the variables
doesn't matter. Why?

1010	0000	0000	0000	is c
0000	0100	0000	0000	is a

1010 0100 0000 0000 c | a

1010	0100	0000	0000	c a
0000	0000	0000	0000	is d

1010 0100 0000 0000 is c | a | d

1010	0100	0000	0000	is c a d
0000	0000	0001	1100	is b

Encrypted number: 1010 0100 0001 1100 is c | a | d | b

That's all!
