Data Types, Operators And Bit Manipulations

Topics

1. What is Data Type?
2. Data Types In C
3. Operators
4. Bit Manipulations
5. Problems

Data Types

Each variable in C (or any other language) has an associated data type. Each data type requires different amounts of memory and has some specific operations, which can be performed over it.

Let us briefly describe them one by one:

a. **char**: The most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers.
b. **int**: As the name suggests, an int variable is used to store an integer.
c. **float**: It is used to store decimal numbers (numbers with floating point value) with single precision.
d. **double**: It is used to store decimal numbers (numbers with floating point value) with double precision.

Different data types also have different ranges up to, which they can store, numbers. These ranges may vary from compiler to compiler. Below is list of ranges along with the memory requirement and format specifiers on 32-bit gcc compiler

Data Types in C & C++

Data Type	Memory (bytes)
short int	2
unsigned short int	2
unsigned int	4
int	4
long int	4
unsigned long long int	8
long long int	8
signed char	1
unsigned char	1
float	4
double	8
long double	12
bool	1

Comparison of a float with a value in C

- float a = 0.7;
- uses the double constant 0.7 to create a single precision number (losing some precision) whereas:
- if (a == 0.7): will compare two double precision numbers (a is promoted first).
- The precision that was lost when turning the double 0.7 into the float **a** is not regained when promoting a back to **a** double.
- If you change all those 0.7 values to 0.7f (to force float rather than double), or if you just make a double, it will work fine.

You can see this in action with the following code:

```
#include <stdio.h>
int main (){
   float f = 0.7;    // double converted to float
   double d1 = 0.7;  // double kept as double
   double d2 = f;    // float converted back to double

   printf ("double:          %.30f\n", d1);
   printf ("double from float: %.30f\n", d2);
   return 0;
}
```

which will output something like:
double: 0.699999999999999955591079014994
double from float: 0.699999988079071044921875000000

Important References:

1. https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html
2. www.pitt.edu/~juy9/142/slides/L3-FP_Representation.pdf
3. http://www3.ntu.edu.sg/home/ehchua/programming/java/datarepresentation.html
4. http://stackoverflow.com/questions/12683215/c-sizeof-with-bool

Operators

Arithmetic Operators
+, -, *, /, %
post-increment (x++)
pre-increment (++x)
post-decrement (x--)
pre-decrement (--x)
These are used to perform arithmetic/mathematical operations on operands.

Relational Operators
== , != , >, <, >=, <=
Relational operators are used for comparison of two values.

Logical Operators (&&, ||, !)

Bitwise Operators (&, |, ^, ~, >> and <<)

Assignment Operators (=, +=, -=, *=, etc)

Other Operators (conditional, comma, sizeof, address, redirecton)

Important References

1. http://www.programiz.com/c-programming/precedence-associativity-operators
2. https://www.cs.cf.ac.uk/Dave/PERL/node35.html

Bit Manipulations

1. **& (bitwise AND):** Takes two numbers as operand and does AND on every bit of two numbers. The result of AND is 1 only if both bits are 1.
2. **| (bitwise OR):** Takes two numbers as operand and does OR on every bit of two numbers. The result of OR is 1 any of the two bits is 1.
3. **^ (bitwise XOR):** Takes two numbers as operand and does XOR on every bit of two numbers. The result of XOR is 1 if the two bits are different.
4. **~ (bitwise NOT):** Takes one number and inverts all bits of it

Truth Table

A	b	&	|	^	~a
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

- **<< (left shift):** Takes two numbers, left shifts the bits of first operand, the second operand decides the number of places to shift.
  - "A<< x" implies shifting the bits of A to the left by x positions.
  - The first x bits are lost this way. The last x bits have 0.
  - Example:
    A = 29 (11101) and x = 2,
    So A << 2 means
    0011101 << 2 = 1110100 (116)
  - **A << x is equal to multiplication by 2^x**.

- **>> (right shift):** Takes two numbers, right shifts the bits of first operand, the second operand decides the number of places to shift.
  - "A >> x" implies shifting the bits of A to the right by x positions.
  - The last x bits are lost this way.
  - Example :
    A = 29 (11101) and x = 2,
    So A >> 2 means
    0011101 >> 2 = 0000111 = 7
  - **A >> x is equal to division by 2^x**

Bit Tricks

- **x & (x-1)** will clear the lowest set bit of x
- **x & ~(x-1)** extracts the lowest set bit of x (all others are clear). Pretty patterns when applied to a linear sequence.
- **x | (x + 1) = x** with the lowest cleared bit set.
- **x | ~(x + 1) =** extracts the lowest cleared bit of x (all others are set).

Important References

1. https://www.topcoder.com/community/data-science/data-science-tutorials/a-bit-of-fun-fun-with-bits/
2. https://www.hackerearth.com/notes/bit-manipulation/
3. http://www.catonmat.net/blog/low-level-bit-hacks-you-absolutely-must-know/
4. http://www.geeksforgeeks.org/category/bit-magic/

Problems

1. Check if ith bit is set in a number. LSB is considered as 0th bit.
   Solution: n & (1<<i) != 0 or (n>>i) & 1 != 0

2. Check if a given number is power of 2 or not
   Solution: n & n-1

3. Check if 2 given two signed integers are of the same sign.
   Solution: a XOR b >= 0

4. Write a program to print Binary representation of a given number.
   Solution: Extract bit by bit - loop or recursion

5. You are given an array with size n, which contains elements from range 1 to n. One element is repeated and one element is missing. Find these 2 elements.
   Solution:
   a. Mathematical: Sum and Product
   b. Using XOR
      i. Let a and b be the 2 numbers to be found.
      ii. XOR the given array with the numbers 1...n, call it X
      iii. You have: X = a^b
      iv. Extract a set bit from X, which means that, that bit is different in a and b.
      v. Based on this bit number, separate the array into 2 parts - One having numbers with this bit set and the other having numbers with this bit unset. Call this SetA-0 and SetA-1
      vi. Do step (v) for the numbers in the range 1..n and call them SetB-0 and SetB-1.
      vii. a = SetA-0 ^ SetB-0 , b = SetA-1 ^ SetB-1
      viii. Check for which is missing and which is repeated, using the array.

6. Find the number of set bits in a given integer
   Solution: (n & n-1) -> unsets the Lesat Set Bit -> repeat untill n becomes 0.

7. Compute pow(a,n)
   Solution:
```
for (int i = 0; i <= log(n); i++) {
        if(n&(1<<i))
                ans = ans*a;
        a=a*a;
}
```

8. Given 2 bit positions: x,y, return a number with those bits set.
   Solution: 1<<x | 1<<y

9. Construct a number with x set bits, followed by y unset bits. 1 <= x,y <= 10.
   Example: x=3, y=5 -> 11100000
   x=2, y=1 -> 110
   Solution: (1<<x) - 1 << y