COMSW 1003-1

Introduction to Computer Programming in C

Lecture 4

Spring 2011

Instructor: Michele Merler

http://www1.cs.columbia.edu/~mmerler/comsw1003-1.html

Announcements

- HW 1 is due on Monday, February 14th at the beginning of class, no exceptions
- Read so far: PCP Chapters 1 to 4
- Reading for next Wednesday: PCP Chapter 5

Review – Access CUNIX

http://www1.cs.columbia.edu/~bert/courses/1003/cunix.html

1) Enable windowing environment
   - X11, Xming, X-Server

2) Launch SSH session (login with UNI and password)
   - Terminal, Putty

3) Launch Emacs
   `$ emacs &`

4) Open/create a file, than save it with .c extension

5) Compile source code into executable with gcc

Review - Compiling your C code

- GCC : **GNU Compiler Collection**
- When you invoke GCC, it normally does preprocessing, compilation, assembly and linking

  - Basic Command
    - gcc myProgram.c
    - ./a.out

Run compiled program (executable)

  - More advanced options

    - gcc –Wall –o myProgram myProgram.c
    - ./myProgram

Review - Compiling your C code

- GCC : **GNU Compiler Collection**
- When you invoke GCC, it normally does preprocessing, compilation, assembly and linking

  - Basic Command
    - gcc myProgram.c
    - ./a.out

Run compiled program (executable)

Display all types of warnings, not only errors

Specify name of the executable

- gcc –Wall –o myProgram myProgram.c
- ./myProgram

Run compiled program (executable)

Review: C Syntax

- Statements
  - one line commands
  - always end with ;
  - can be grouped between { }

- Comments

  // single line comment

  /* multiple lines comments
  */

Review : Variables and types

- **Variables** are placeholders for values

```
int x = 2;
x = x + 3; // x value is 5 now
```

- In C, variables are divided into **types**, according to how they are **represented in memory** (always represented in binary)
  - **int** **4 bytes, signed/unsigned**
  - **float** **4 bytes, decimal part + exponent**
  - **double** **8 bytes**
  - **char** **1 byte, ASCII Table**

Review : Casting

- Casting is a method to correctly use variables of different types together
- It allows to treat a variable of one type as if it were of another type in a specific context
- When it makes sense, the compiler does it for us automatically

- Implicit (automatic)

```
int x =1;
float y = 2.3;
x = x + y;
```

x= 3 compiler automatically casted (=converted) y to be an integer just for this instruction

- Explicit (non-automatic)

```
char c = 'A' ;
int x = (int) c;
```

Explicit casting from char to int. The value of x here is 65

Today

- Operators
- printf()
- Binary logic

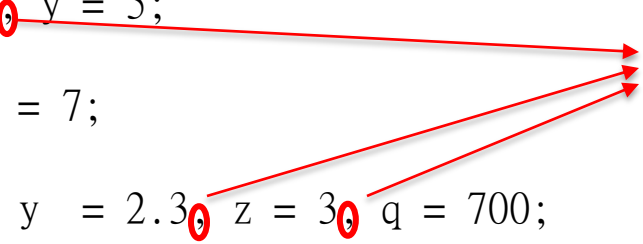
Operators

- Assignment `=`
- Arithmetic `*  /  %  +  -`
- Increment `++  --  +=  -=`
- Relational `<  <=  >  >=  ==  !=`
- Logical `&&  ||  !`
- Bitwise `&  |  ~  ^  <<  >>`
- Comma `,`

Operators – Assignment and Comma

```
int x = 3;

x = 7;
```

```
int x, y = 5;

x = y = 7;

float y  = 2.3, z = 3, q = 700;
```



The comma operator allows us to perform multiple assignments/declarations

```
int i,j,k;

k = (i=2, j=3);

printf( "i = %d, j = %d, k = %d\n" ,i,j,k);
```

Operators - Arithmetic

*	/	%	+	-

- Arithmetic operators have a **precedence**

```
int x;

x = 3 + 5 * 2 - 4 / 2;
```

- We can use parentheses () to impose our precedence order

```
int x;

x = (3 + 5) * (2 - 4) / 2;
```

- % returns the module (or the remainder of the division)

```
int x;

x = 5 % 3;  // x = 2
```

- We have to be careful with integer vs. float division : remember automatic casting!

```
int x = 3;
float y;

y = x / 2; // y = 1.00
```

```
float y;

y = 1 / 2;  // y = 0.00
```

Operators - Arithmetic

*	/	%	+	-

- Arithmetic operators have a **precedence**

```
int x;
x = 3 + 5 * 2 - 4 / 2;
```

- We can use parentheses () to impose our precedence order

```
int x;
x = (3 + 5) * (2 - 4) / 2;
```

- % returns the module (or the remainder of the division)

```
int x;
x = 5 % 3;  // x = 2
```

- We have to be careful with integer vs. float division : remember automatic casting!

```
int x = 3;
float y;
y = x / 2; // y = 1.00
```

Possible fixes:
1)`float x = 3;`
2)`y = (float) x /2;`
Then `y = 1.50`

```
float y;
y = 1 / 2;  // y = 0.00
```

Possible fix: `y = 1.0/2;`
Then `y = 0.50`

Operators – Increment/Decrement

++	--	+=	-=

```
int x = 3, y, z;
```

`x++;` ⟶ x is incremented at the end of statement

`++x;` ⟶ x is incremented at the beginning of statement

```
y = ++x + 3; // x = x + 1; y = x + 3;

z = x++ + 3; // z = x + 3; x = x + 1;

x -= 2;      // x = x - 2;
```

Operators - Relational

<	<=	>	>=	==	!=

- Return 0 if statement is false, 1 if statement is true

```
int x = 3, y = 2, z, k, t;

z = x > y;          // z = 1

k = x <= y;         // k = 0

t = x != y;         // t = 1
```

Operators - Logical

&&	||	!

- A variable with value 0 is false, a variable with value !=0 is true

```
int x = 3, y = 0, z, k, t, q = -3;

z = x && y;     // z = 0;     x is true but y is false

k = x || y;     // k = 1;     x is true

t = !q;         // t = 0;     q is true
```

Operators - Bitwise

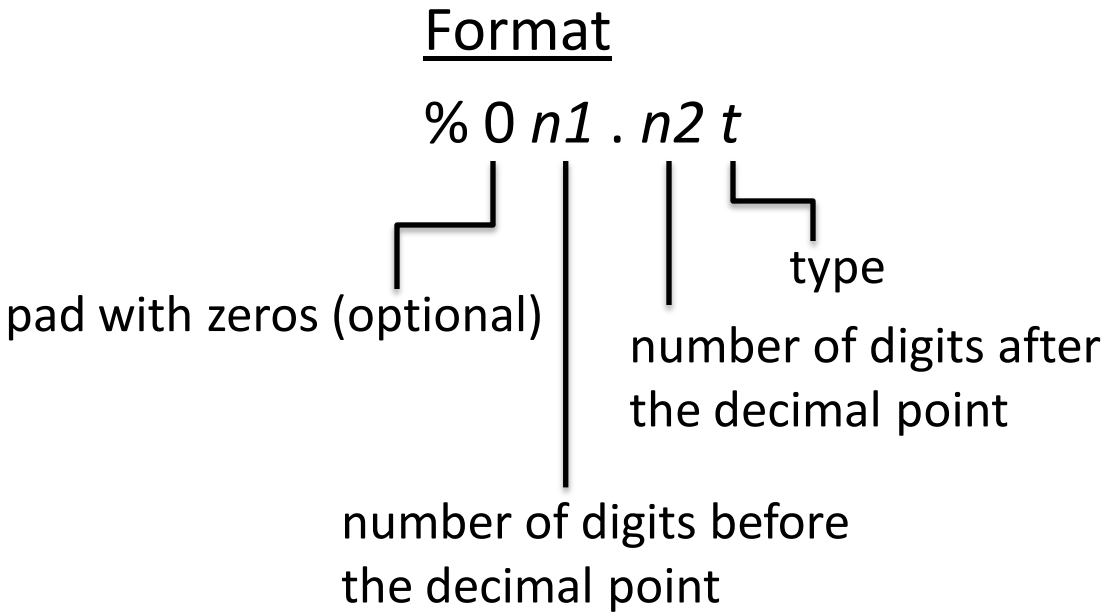
- Work on the binary representation of data
- Remember: computers store and see data in binary format!

```
int x, y, z , t, q, s, v;

x = 3;                                  00000000000000000000000000000011
y = 16;                                 00000000000000000000000000010000

z = x << 1;  equivalent to  z = x · 2^1 00000000000000000000000000000110

t = y >> 3;  equivalent to  t = y · 2^-3 00000000000000000000000000000010

q = x & y;                              00000000000000000000000000000000

s = x | y;                              00000000000000000000000000010011

v = x ^ y;                              00000000000000000000000000010011
```

XOR

printf

- `printf` is a function used to print to standard output (command line)
- Syntax:
  `printf("format1 format2 …", variable1, variable2,…);`
- Format characters:
  - %d or %i integer
  - %f float
  - %lf double
  - %c char
  - %u unsigned
  - %s string



printf

```c
#include <stdio.h>

int main() {

   int a,b;
   float c,d;
   a = 15;
   b = a / 2;

   printf("%d\n",b);
   printf("%3d\n",b);
   printf("%03d\n",b);

   c = 15.3;
   d = c / 3;
   printf("%3.2f\n",d);

   return(0);

}
```

Output:

```
7
  7
007

5.10
```

printf

Escape sequences

\n	newline
\t	tab
\v	vertical tab
\f	new page
\b	backspace
\r	carriage return

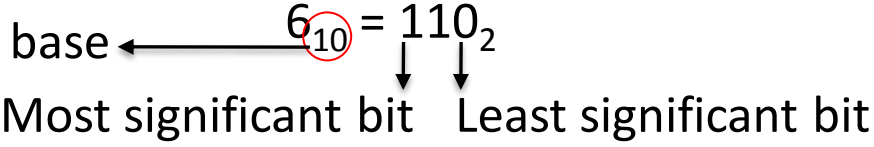
Binary Logic

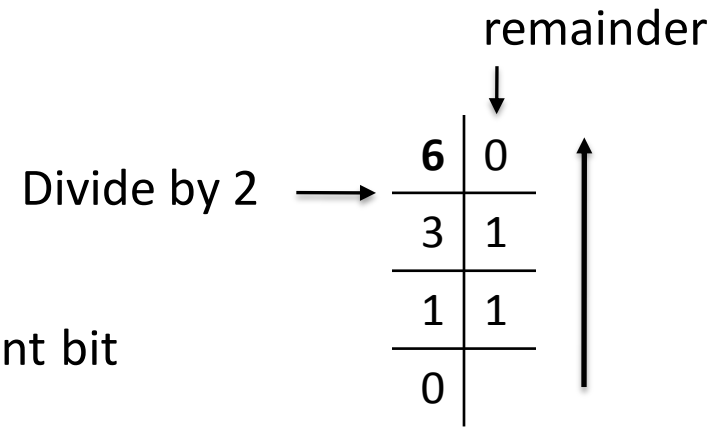
- 1 = true, 0 = false
- Decimal to binary conversion

$$6_{10} = 110_2$$

Binary Logic

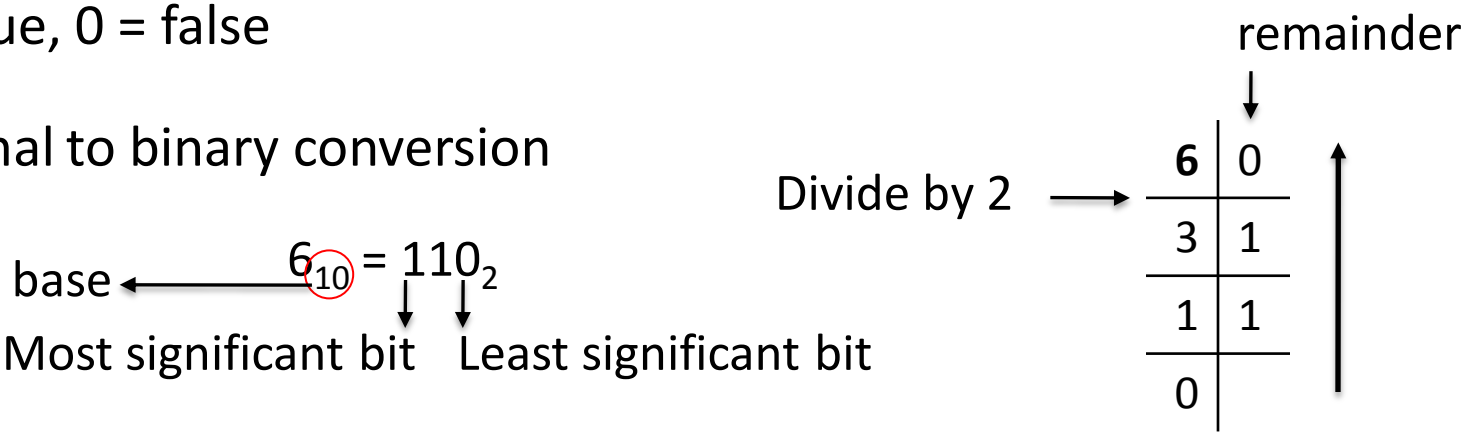
- 1 = true, 0 = false
- Decimal to binary conversion





Binary Logic

- 1 = true, 0 = false
- Decimal to binary conversion

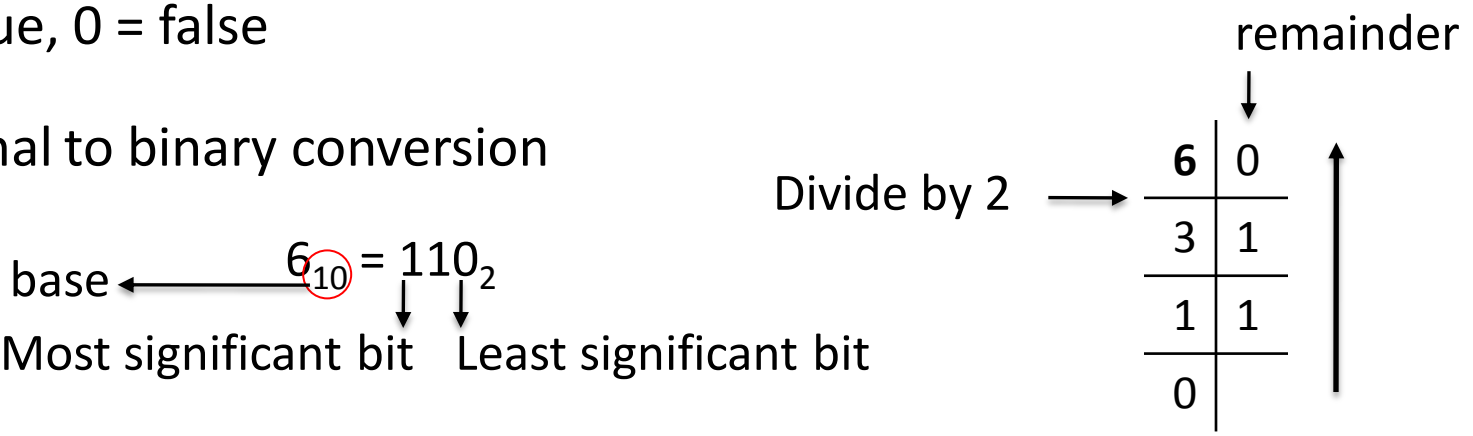


- Binary to decimal conversion

$11001_2 = 1x2^0 + 0x2^1 + 0x2^2 + 1x2^3 + 1x2^4 = 25$

Binary Logic

- 1 = true, 0 = false
- Decimal to binary conversion



- Binary to decimal conversion

$11001_2 = 1x2^0 + 0x2^1 + 0x2^2 + 1x2^3 + 1x2^4 = 25$

- AND
  v = x & y

x	y	v
0	0	0
0	1	0
1	0	0
1	1	1

- OR
  v = x | y

x	y	v
0	0	0
0	1	1
1	0	1
1	1	1

- NOT
  v = !x

x	v
0	1
1	0

- EXOR
  v = x ^ y

x	y	v
0	0	0
0	1	1
1	0	1
1	1	0

Homework 1 review

HOW TO COMPRESS/UNCOMPRESS folders in UNIX

- Compress folder ~/COMS1003/HW1 to HW1.tar.gz

  tar -zcvf HW1.tar.gz ~/COMS1003/HW1

- Uncompress HW1.tar.gz to folder ~/COMS1003/HW1new

  tar -zxvf HW1.tar.gz -C ~/COMS1003/HW1new

  (note: ~/COMS1003/HW1new must exist already)