CSE 378, Winter 2003

Machine Organization and Assembly Language Programming

- Section AA Thursday 12:30-1:20 MGH 287
- Section AB Thursday 1:30-2:20 MUE 155

- **Gang Zhao**
- **email** galaxy@cs.washington.edu
- **Office hour**: Thursday, 10:00-12:00, Sieg 226b

C vs. Java

- Java program
  - collection of classes
  - class containing main method is starting class
  - running java StartClass invokes StartClass.main method
  - JVM loads other classes as required
- C program
  - collection of functions, no classes/objects
  - one function - main() - is starting function
  - running executable (default name a.out) starts main function
  - typically, single program with all user code linked into, but can be dynamic libraries(.dll, .so)

- Today:
  - The basic of C for your java-native programmers
    - Forget about the objects/classes
    - Pointers are just the memory addresses.
  - A little more words on binary representation
    - Integers
    - Floating point

C vs. Java: simple example

- Java:

```
public class hello {
    public static void main( String[] args ) {
        System.out.println( "hello world! " );
    }
}
```

- C:

```
#include <stdio.h>
int main( int argc, char *argv[] ) {
    printf( "hello world!" );
    return 0;
}
```

Some facts about C

- C is early-70s, procedural language; (Java is mid-90s, high-level Object-Oriented (OO) language)
- Both high-level and low-level language
  - OS: from user interface to kernel to device driver
- Better control of low-level mechanisms
  - memory allocation, specific memory locations
- Performance sometimes better than Java
  - usually more predictable
- Java hides many details needed for writing code, but in C you need to be careful because:
  - memory management responsibility left to you
  - explicit initialization and error detection left to you
  - generally, more lines of (your) code for the same functionality
  - more room for you to make mistakes

Data types in C

- **sizes and limits (may vary for machine)**

Type	Size in bits	range
char	8	[-128, 127] = [-2^7, 2^7-1]
short	16	[-32768, 32767] = [-2^15, 2^15-1]
int	32	[-2,147,483,648, 2,147,483,647]=[-2^31, 2^31-1]
long	32	[-2,147,483,648, 2,147,483,647]=[-2^31, 2^31-1]
float	32	about -10^38 to 10^38
double	64	about -10^308 to 10^308

- **You can also have unsigned values.**

Type	Size in bits	range
unsigned char	8	[0, 255] = [0, 2^8-1]
unsigned short	16	[0, 65535] = [0, 2^16-1]
unsigned int	32	[0, 4,294,967,295]=[0, 2^32-1]
unsigned long	32	[0, 4,294,967,295]=[0, 2^32-1]

Data types in C (con't)

- Note that there is not a **boolean** data type, you may use int or char to emulate it.
- **struct** is used to declare a new data types.

Example:
```
struct record {
        char student_name[8];
        char course_sname[16];
        int score; };

struct record one_record;
```

  - Basically struct means grouping variables together.
- **Array**: similar to java

Example:
```
int A[10];
A[3] = 5;
```

Pointers(1)

- Pointers ---- variables that contain memory addresses as their values.
  - int count;
  - int * countPtr;
- & is used to dereference a pointer
  - int count;
  - int *countPtr = &count;

Control structure in C

- Looping
  - for `for(i = 0; i < 1000; i++) { …. }`
  - while `while(i <= 1000) { … }`
  - do … while `do { …. } while (stop_me == 'y')`
- branch
  - if/else
```
if ( condition == 1) { … }
else { …. }
```
  - switch … case
```
switch ( number ) {
    case 0: ….; break;
    case 1: ….; break;
    …
    default: …}
```

Pointers(2) example

```
int i, j, count;
int * countPtr;

i = 3;
j = -99;
count = 12;
countPtr = &count;
```

variables	addresses	values
	…	…
i	0xbffff4f0	3
j	0xbffff4f4	-99
count	0xbffff4f8	12
countPtr	0xbffff4fc	0xbffff4f8
	…	…

?? count++;
countPtr++;

Preprocessor and comments

- **C Preprocessor**
  - define new macros
    - #define MAXVALUE 100
  - include files with C code (typically, "header" files end with `.h`)
    - #include "filename.h"
  - conditionally compile parts of file
    - #ifdef name
    - code segment 1
    - #else
    - code segment 2
    - #endif
- **Comments**

/* any text until */

A comprehensive example--link list

```
#include <stdio.h>
#include <malloc.h>

#define TEXT_MAX 128

typedef struct tag_node {
  struct tag_node *pnext;
  char text[TEXT_MAX];
} NODE;

NODE * new_node(char *ptext) {
  NODE *pnode = (NODE
      *)malloc(sizeof(NODE));
  strcpy(pnode->text, ptext);
  return pnode;
}

int main(int argc, char **argv) {
  NODE *proot = new_node("");
  NODE *pnode = proot;
  char line[TEXT_MAX];

  printf("TEXT>");
  scanf("%s", line);
  while (line[0] != '.') {
    pnode->pnext = new_node(line);
    pnode = pnode->pnext;
    pnode->pnext = NULL;
    printf("TEXT>");
    scanf("%s", line);
  }
  pnode = proot;
  do {
    printf("%s\n", pnode->pnext->text);
    pnode = pnode->pnext;
  } while (pnode->pnext != NULL);
}
```

binary representation(1) signed integer

- Sign and magnitude

  1 bit sign + (n-1) bit magnitude.

  $$(47)_{10} = (0010\ 1111)_2 \xrightarrow{\text{sign and magnitude}} (0010\ 1111)_2$$

  $$(-47)_{10} = (-0010\ 1111)_2 \xrightarrow{\text{sign and magnitude}} (1010\ 1111)_2$$

- 2’s complement

  It can be computed by $(2^{n+1} + x) \bmod 2^{n+1}$

  $$(47)_{10} = (0010\ 1111)_2$$

  $$\xrightarrow{2's\ \text{complement}} ((1\ 0000\ 0000)_2 + (0010\ 1111)_2) \bmod (1\ 0000\ 0000)_2$$

  $$= (0010\ 1111)_2$$

  $$(-47)_{10} = (-0010\ 1111)_2$$

  $$\xrightarrow{2's\ \text{complement}} ((1\ 0000\ 0000)_2 + (-0010\ 1111)_2) \bmod$$

  $$(1\ 0000\ 0000)_2 = (1101\ 0001)_2$$

binary representation(2)

IEEE 754 floating-point standard

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
s	exponent								significand																						
1 bit	8 bits								23 bits																						

- Single precision, double precision
- Normalized significand – omit the first leading 1
- Biased exponent. -- exponent + 127

$$(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent-Bias})}$$