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

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

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

You can also have unsigned values.

<table>
<thead>
<tr>
<th>Type</th>
<th>Size in bits</th>
<th>range</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char</td>
<td>8</td>
<td>[0, 255] = [0, 2^8-1]</td>
</tr>
<tr>
<td>unsigned short</td>
<td>16</td>
<td>[0, 65535] = [0, 2^16-1]</td>
</tr>
<tr>
<td>unsigned int</td>
<td>32</td>
<td>[0, 4,294,967,295] = [0, 2^32-1]</td>
</tr>
<tr>
<td>unsigned long</td>
<td>32</td>
<td>[0, 4,294,967,295] = [0, 2^32-1]</td>
</tr>
</tbody>
</table>
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_name[16];`  
  `  int score;`  
  `};`  

• & is used to dereference a pointer  
  – int count;  
  – int *countPtr;  

• Array: similar to java
  Example: `int A[10];`  

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: …}`

• 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  
        • #else  
        • #endif
  – Comments
    `/* any text until */`

Pointers

• Pointers ---- variables that contain memory addresses as their values.
  – int count;
  – int *countPtr;

• & is used to dereference a pointer  
  – int count;
  – int *countPtr = &count;

A comprehensive example--link list

```c
#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("TEXT>");
    scanf("%s", line);
    pnode = pnode->pnext;
  } while (pnode->pnext != NULL);
  return 0;
}
```

Pointers example

<table>
<thead>
<tr>
<th>variables</th>
<th>addresses</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>0xbffff4f0</td>
<td>3</td>
</tr>
<tr>
<td>j</td>
<td>0xbffff4f4</td>
<td>-99</td>
</tr>
<tr>
<td>count</td>
<td>0xbffff4f8</td>
<td>12</td>
</tr>
<tr>
<td>countPtr</td>
<td>0xbffff4fc</td>
<td>0xbffff4f8</td>
</tr>
<tr>
<td>?? countPtr++;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>?? countPtr++;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
binary representation (1)
signed integer

- Sign and magnitude
  1 bit sign + (n-1) bit magnitude.
  \((47)_{10} = (0010 1111)_2 \quad \text{and} \quad (-47)_{10} = (1010 1111)_2\)

- 2's complement
  It can be computed by \(2^{n-1} + x \mod 2^n\)
  \((47)_{10} = (0010 1111)_2 \quad \text{and} \quad (-47)_{10} = (1010 1111)_2\)
  \((-47)_{10} = (-0010 1111) \mod (1 0000 0000)\)
  \((-47)_{10} = (0010 1111) \mod (1 0000 0000)\)

binary representation (2)
IEEE 754 floating-point standard

- Single precision, double precision
- Normalized significand – omit the first leading 1
- Biased exponent. \(\text{exponent} + 127\)
  \((-1)^s \times (1 + \text{Significand}) \times 2^{\text{(Exponent-Bias)}}\)