Lecture - II
Basics of C Programming

Summary of Last Class
• Basics of UNIX:
  - logging in, changing password
  - text editing with vi, emacs and pico
  - file and directory operations
  - file/dir permissions and changing them

Processes
• ps: list currently active user processes
• ps aux: list all active processes in long format
• kill n: kill process with id=n
• kill -9 n: force to kill
• CTRL-z: push to background
• fg: bring to foreground (also fg n: bring nth process)
• top: system utilization information
• time command: calculate time for a given command

C vs. Java
• C is procedural, not object oriented
  - C has no objects, interfaces or packages
  - A program only consists of functions and data
• C is compiled, not interpreted
  - Translated directly to assembly language
  - Faster, less portable and very hard to debug.
• C has no array bounds, null pointer or cast checks
  - You have to detect and handle all problems yourself
• C has no garbage collector
  - You have to do all of the memory management yourself

C vs. Java (cont.)
• C has pointers
  - Similar to Java references but...
  - ...they can be used in calculations (pointer arithmetic)
  - Allows you to use the location of data in computations (not just the value)
  - Useful, powerful and a debugging nightmare!
• Compared to Java, C is a low-level language
  - You can and must do everything yourself
  - Suitable for low-level software like device-drivers, communication libraries, operating systems, etc.
• You can implement anything in C!
  - No limits!

C vs. Java (cont.)
• A Java program consists of:
  - Several classes, one class per file.
  - A main method in one of these classes.
  - External class libraries (jar files).
• A C program consists of:
  - Several functions in any number of files.
  - A main function in one of these files.
  - Possibly some header files.
  - External libraries with their own header files.
C can be quite complex

- This program computes and prints the first 800 decimals of PI:

```c
#include <stdio.h>
long a=10000,b,c=2800,d,e,f[2801],g;
int main(){
    for(;b-c;)
    f[++b]=a/5;
    for(;d=0,g=c*2;c-=14,printf("%.4d",e+d/a),e=d%a)
        for(b=c;d+=f[b]*a,f[b]=d%--g,d/=g--,--b;d*=b);
}
```

Basic C Program

```c
#include <stdio.h>
main() {
    
}
```

Basic C Program

```c
prog1.c:
...
    main() {
        
    }
...

gcc prog1.c ==> a.out
gcc prog1.c -o prog1 ==> prog1
make prog1 ==> prog1
```

Print to the screen (stdout)

```c
#include <stdio.h>
main() {
    printf("Hello, CSC4304 Class!\n");
}
```

Header Files

- The C compiler works in 3 phases:
  1. Pre-process source files
  2. Compile source files into object files
  3. Link object files into an executable
- `#include <stdio.h>` means "include the contents of standard file
  `stdio.h` here"
  - Standard files are usually located in directory `/usr/include`
  - `/usr/include/stdio.h` may contain `#include` statements itself...
- You can use `#include` to include your own files into each other:
  - `#include "myfile.h"` means "include file `myfile.h` (from the current
directory) here"
  - Included files usually have extension "`.h" (header)"

Read argument and print

```c
#include <stdio.h>
// take arguments from stdin
main(int argc, char* argv[])
{
    printf("Hello, %s\n", argv[1]);
}
```
Basic Data Types

- **Basic Types**
  - `char`: character - 1 byte
  - `short`: short integer - 2 bytes
  - `int`: integer - 4 bytes
  - `long`: long integer - 4 bytes
  - `float`: floating point - 4 bytes
  - `double`: double precision floating point - 8 bytes

- **Formatting Template**
  - `%d`: integers
  - `%f`: floating point
  - `%c`: characters
  - `%s`: string
  - `%x`: hexadecimal
  - `%u`: unsigned int

Formatting

```c
#include <stdio.h>
main()
{
    char var1;
    float f;

    printf("Enter a character:");
    scanf("%c", &var1);
    printf("You have entered character:%c. ASCII value=%d, Address=%x\n", var1, var1, &var1);
    f = (float)var1;
    printf(".. And its float value would be: %.2f\n", f);
}
```

Formatting (cont.)

```c
#include <stdio.h>
ter main(void) {
    int val = 5;
    char c = 'a';
    char str[] = "world";

    printf("Hello world\n");
    printf("Hello %d World\n", val);
    printf("Hello %c World\n", c);
    printf("Hello %s World\n", str);
    printf("Hello %lu\n", (unsigned long)val);
    return 0;
}
```

Arrays

- **Defining an array is easy:**
  ```c
  int a[3]; /* a is an array of 3 integers */
  ```
- **Array indexes go from 0 to n-1:**
  ```c
  a[0] = 5; a[1] = 4; a[2] = 3; /*合法*/
  if (x = a[0]); /* What is the value of x? */
  ```
- **Beware:** in this example a[3] does not exist, but your compiler will not complain if you use it.
  - But your program may have a very strange behavior...
- **You can create multidimensional arrays:**
  ```c
  int matrix[3][2];
  ```
Strings

- A string is an array of characters:
  ```c
  char hello[13]="Hello, world\n";
  ```

- Unlike in Java, you must decide in advance how many characters can be stored in a string.
- You cannot change the size of the array afterwards.
- Beware: strings are always terminated by a NULL character: `\0`
  - For example, "Hello" is string of 6 characters: `H e l l o \0`

Manipulating Strings

- There are standard function to manipulate strings:
  - `strcpy(destination, source)` will copy string `source` into string `destination`:
    ```c
    char a[15] = "Hello, world\n";
    char b[10];
    strcpy(b,a);
    ```
    ```c
    # Attention: `strcpy` does not check that destination is large enough to accommodate source.
    ```

- `char s[10];
  strcpy(s,a);` /* This will get you in BIG trouble */

Manipulating Strings (cont.)

- Instead of `strcpy` it is always better to use `strncpy`:
  - `strncpy` takes one more parameter to indicate the maximum number of characters to copy:
    ```c
    char a[15] = "Hello, world\n";
    char c[10];
    strncpy(c,a,9); /* Why 9 instead of 10? */
    ```

Comparison Operators

- The following operators are defined for basic data types:
  ```c
  if (a == b) {...} 
  if (a != b) {...} 
  if (a < b) {...} 
  if (a <= b) {...} 
  if (a > b) {...} 
  if (a >= b) {...} 
  if ((x&b) && (x&c)) {...} /* logical AND */ 
  if ((x&b) || (x&c)) {...} /* logical OR */ 
  ```

- There is no boolean type in C. We use integers instead:
  - 0 means FALSE
  - Any other value means TRUE

```c
int x;
if (x) {...} /* Equivalent to: if (x!=0) {...} */
if (!x) {...} /* Equivalent to: if (x==0) {...} */
```
Classical Bugs

- Do not confuse '=' and '=='

```c
if (x=y) {...}  // This is correct C but it means something different */
if (x==y) { /* always execute */}
if (x!=0) { /* never executed */}
```

- Do not confuse '&' and '&&'

```c
if (x&y) {...}  // This is correct C but it means something different */
if (x|y) {...}  // This is correct C but it means something different */
```

Loops

```c
while (x>0){
    ...
}
do{
    ...
} while (x>0);
for (x=0; x<3; x++) {...}
```

Functions

- In C, functions can be defined in two ways:

```c
int foo() {
    /* function foo returns an int */
    return 10;
}
void bar(int p1, double p2) {
    /* function bar returns nothing */
    ...
}
```

- Calling a function is easy:

```c
int i = foo();  /* call function foo() */
bar(2, -4.21);  /* call function bar() */
```

Exercises

1. Write a program which defines an integer, a float, a character and a string, then displays their values and their sizes on screen. /*use the sizeof() function*/

2. Write a program which computes and displays fib(n), where n is a parameter taken from command line:

   fib(0) = 0, fib(1) = 1
   If n > 1 then fib(n) = fib(n-1) + fib(n-2)

Summary

- C Basics
  - C vs Java
  - Writing to stdout
  - Taking arguments
  - Reading from stdin
  - Basic data types
  - Formatting
  - Arrays and Strings
  - Comparison Operators
  - Loops
  - Functions

Hmm.

Acknowledgments

- Advanced Programming in the Unix Environment by R. Stevens
- The C Programming Language by B. Kernighan and D. Ritchie
- Understanding Unix/Linux Programming by B. Molay
- Lecture notes from B. Molay (Harvard), T. Kuo (UT-Austin), G. Pierre (Vrije), M. Matthews (SC), and B. Knicki (WPI).