Lecture 1
C Programming Language
Summary of Lecture 1

- General Information
- Homework Schedule
- Introduction to UNIX
- Introduction to C
- Simple C Programs and Examples
General Information

- News Group: columbia.spring.cs3101-secl
- Instructor: Aya Aner, Office 725 CEPSR, phone: 939-7121, e-mail: aya@cs.columbia.edu
  Office hours: Tuesday 11:30-12:30, Thursday 11:30-12:30, or by appointment

TA: Aya Aner
Text: An Introduction to ANSI C on UNIX, Paul Wang (available at Papyrus)
A good reference is Kernighan & Ritchie, The C Programming Language.

- Consult web page, general information link.
Prerequisites

- Prior programming experience
- Knowledge of a UNIX editor (e.g. vi, emacs, pico)
- CUNIX account - YOU CANNOT PARTICIPATE IN THIS CLASS WITHOUT IT !!
- Understanding that this is a 3000 level class, with 4 homework assignments over four weeks.
# Syllabus

- **Lecture 1** 1/18 introduction, simple C programs
- **Lecture 2** 1/20 control flow, data types
- **Lecture 3** 1/25 streams, preprocessor, more control
- **Lecture 4** 1/27 strings, scoping, debugging
- **Lecture 5** 2/1 arrays, pointers, dynamic memory
- **Lecture 6** 2/3 more on pointers, casting, makefile
- **Lecture 7** 2/8 structures, linked lists
- **Lecture 8** 2/10 recursion, unions

- Review Session 2/15

- Final Exam 2/17 (Thursday)
Homework Schedule

• All homework assignments are due exactly a WEEK after assignment:
  Electronic submission by 10:00am
  Hardcopy submission at 10:00am (in class).

• HW1: Assigned on Lecture 1
• HW2: Assigned on Lecture 3
• HW3: Assigned on Lecture 5
• HW4: Assigned on Lecture 7
UNIX System Shell

- We interact with UNIX through shell commands
- Three popular shells
  - sh  Bourne shell  $prompt
  - ksh  Korn shell  $prompt
  - csh  C-shell  %prompt
- UNIX file system
- Examples of shell commands:
  - %ls - list files in current directory
  - %man command - manual for command
  - %cd dir - enter directory dir
  - %pwd - name current directory
  - %cp src dest - copy src to dest
  - %mv src dest - move src to dest
  - %rm f1 - remove file f1
  - %rmdir dir - remover directory dir
  - %mkdir dir - create a new directory dir
  - %rm * - BEWARE !!! Removes all files!!!
# More Shell Commands

<table>
<thead>
<tr>
<th>Usage</th>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpr &lt;arg&gt;</td>
<td>lpr -Pslrp p1.c a.out &amp;</td>
<td>Print file</td>
</tr>
<tr>
<td>&lt;command&gt;&amp;</td>
<td></td>
<td>Run program in the background.</td>
</tr>
<tr>
<td>^z</td>
<td>^z</td>
<td>Stop running</td>
</tr>
<tr>
<td>bg</td>
<td>bg</td>
<td>Continue running - in background</td>
</tr>
<tr>
<td>jobs</td>
<td>jobs</td>
<td>List jobs running</td>
</tr>
<tr>
<td>kill %n</td>
<td>kill 1245</td>
<td>Kill job no. %n</td>
</tr>
<tr>
<td>gcc</td>
<td>gcc p1.c</td>
<td>Compile code, executable is a.out</td>
</tr>
<tr>
<td>chmod &lt;arg&gt;</td>
<td>chmod g-r p1.c</td>
<td>Change permissions to a file</td>
</tr>
</tbody>
</table>
Important Features

- In order to compile and run your c program you need to have access to the compiler (a program) “gcc” or “cc” at:
  /opt/local/bin/gcc and
  /opt/local/bin/cc

- When compiling your program myprog.c:
  %gcc myprog.c
  You create an executable a.out

- In order to run any program its path needs to be defined in your PATH variable (try %echo $PATH to view the list of paths). This variable is usually initialized in your .profile file - add the working directory (for example “aya/W 3101-01/hw1”) to the paths list in PATH using the “:” symbol for concatenation.
  Add the path for the compiler too.
Important Features

• Running your program a.out using an input file and an output file:
  `%a.out < input.file > output.file`
• `chmod`: u - user, g - group, o - other
  r - read, w - write, x - execute
• `%more` / `less` `filename` - view file
• `%echo $DISPLAY`
  `my_machine:0.0`
• `%export`
  `DISPLAY=dynasty.cs.columbia.edu:0.0`

• Note to the beginning programmer:
  `% emacs p1.c`
  will create a new file `p1.c` if no such file exists.
C - Program Structure

- Every C program has a main() function which is an entry point:
  ```c
  int main()
  {
      printf("Hello everybody\n");
  }
  ```
- Program is built out of blocks `{...}`
- Building blocks can also be functions, loops.
- Functions have:
  - arguments (optional)
  - return value (optional)
  - statements
  - variables (optional)
- Example:
  ```c
  int main()
  {
      int j = 1;
      j = 8+1;
  }
  ```
Basic Data Types

- Data types define how data is interpreted
- Examples:
  ```
  int    (for integers)
  float  (for floating point numbers)
  double (for double precision floating point numbers)
  char   (for characters)
  ```
- Note: the size of each data type is dependent on implementation. The command `sizeof()` returns the size of the type in bytes:
  ```
  sizeof(char) = 1  
  sizeof(int) = 4   
  sizeof(float) = 4 
  sizeof(double) = 8
  ```
- Examples for defining/declaring variables:
  ```
  int j;   /* j is an integer */
  float f; /* f is a float */
  char c1,c2; /* c1, c2 are characters */
  /* and this is just a comment.... */
  ```
Integers

- There are two types of integer types used:
  - signed integer (int i;)
  - unsigned integer (unsigned int j;)
signed integers use one bit for sign -
unsigned integers hold bigger values

- **Booleans**
  There are no boolean data types in C:
  we use 0 for FALSE,
  any other integer for TRUE (usually 1)

- **Operations on integers:**
  ```c
  int j = 1; /* initialization */
  j = j+1; /* or : */
  j++;
  j = j-1; /* or : */
  --j;
  j = j*6; /* or j *=6; */
  j = j/6; /* or j /= 6; */
  ```
Expressions, Statements

- Expression - consists of operands (variables or constants) and operators:
  - Relational expressions: \( x > y, 2 == y, x! = 5 \)
  - Arithmetic expressions: \( x+2, y--, x*y, 6/2 \)
  - The assignment expression: \( x = y, y = 4 \)
  Beware !!!
  Don’t use \( (x=y) \) to check equality !!!

- Statements:
  - simple: \( x=5; \)
  - compound: \( \{ x=5; y = z = 3; f *= 5.0; \} \)
  - loops
  - do while
  - multi-way if
Loops

- **“for” loop**
  
  ```
  for (initialization; test; increment/decrement) {
  .... /* statements */
  }
  ```

  ```
  for (j = 0; j < 10; j++) {
  printf(“shoop “);
  }
  ```

- **“while” loop**
  
  ```
  while (test) {
  .... /* statements */
  }
  ```

  ```
  j = 0;
  while (j < 10) {
  printf(“shoop “); j ++;
  }
  ```
Loops, If Statement

• **“do while” loop**
  ```c
  do
  {
      ...... /* statements */
  }
  ```
  ```c
  while (test);
  j=0;
  do {
      printf("shoop "); j++;
  } while (j <10)
  ```

• **If Statement**
  ```c
  if (test) <statement>
  were <statement> can be simple:
  if (x>1) x = 0; or compound:
  if (x>1) { x=0; printf("hey");}
  or another if statement:
  if (x>1)
      if (x>2)
          x=0;
  ```
If - Else Statement

- **if else Statement**
  if (test)
    <statement1>
  else
    <statement2>

- **Example:**
  int j=1;
  int k=1;
  if (j==k)
    printf("j==k\n");
  else
    printf("j!=k\n");

- **Example:**
  if (j==k)
    printf("j==k\n");
  else if (j<k)
    printf("j<k\n");
  else
    printf("j>k\n");
Functions

- **Function definition:**
  ```cpp
  int square(int a) {
      return(a*a);
  }
  ```
- **Function declaration:**
  ```cpp
  int square(int); /* prototype */
  ```
  A function must be **declared** before use.
- A function can be **declared** via a prototype before it is actually **defined**.
- **Prototype:**
  - the function name
  - type of return value
  - type of arguments
- **Example:**
  ```cpp
  int square(int);
  main() {
      int x,y=2;
      x = square(y); /* A function call */
      printf("square of %d is %d\n",y,x);
  }
  ```
Input/Output - Standard I/O

- Standard input - from keyboard
- Standard output - to screen
- Examples of standard library functions:
  - Output: printf, putchar, puts, fprintf
  - Input: scanf, getchar, gets, fscanf
- printf, scanf use special formats to read and write different variable types:
  ```c
  int j; float f; double d; char c;
  printf("%d %f %lf %c", j, f, d, c);
  scanf("%d %f %lf %c", &j, &f, &d, &c);
  ```
- & is the address operator. When reading a value into a variable we use &.
- main () {
    int j;
    for (j=1; j<10; j++) {
      printf("Hello world\n");
      printf("j = %d\n",j);
    }
}
Simple C Program

- `#include <stdio.h> /* to use I/O functions */`

```c
func1 ()
{
    int j;
    for (j=1; j<10; j++)
    {
        printf("Hello world\n");
        printf("j = %d\n",j);
    }
}
```

```c
main()
{
    func1(); /* function call.. */
}
```

- Write this in file `my_prog.c`, compile and run!
Summary of Lecture 1

• UNIX basics
• Basic Data Types
• Expressions / Statements
• loops: for, while, do-while
• if, if-else statements
• Standard I/O
• Simple C programs