Oregon State University
School of Electrical Engineering and Computer Science

CS 261 – Recitation 1

Spring 2011
Outline

• Using Secure Shell Clients
• The GCC
• Some Examples
• Intro to C
Get your terminals

• Windows people, get ssh
  http://www.colorado.edu/its/docs/authenticate/printouts/win_ssh.html
  – It already has a build-in file transfer client

• Mac people, you have ssh built-in and you can use a file transfer client such as Cyberduck.
Using Secure Shell Clients

- Open ‘Terminal’ & type `<username>@<host name>` (for Mac)
Using Secure Shell Clients

List of available servers:

- flip.engr.oregonstate.edu
- flop.engr.oregonstate.edu – off-campus

And more (see http://eecs.oregonstate.edu/it/)
Basic commands

- **pwd**
  - Present working directory
- **ls**
  - list files and directories in current directory
  - `% ls -la`: ‘el’ denotes long listing
    ‘a’ including all hidden files
- **cd**
  - Change directory
- **mkdir**
  - make new directory
- **cp**
  - copy `<srcFileName> <desFileName>`
- **mv**
  - Moves / renames `<srcFileName> <desFileName>`
- **rm**
  - remove file
- **cat**
  - show file content
- **exit**
  - exit the session
Basic commands continued..

- . (dot)
  - Represents current directory
- .. (dot dot)
  - Represents parent directory

More information on how to use gcc and vim editor, please refer to 2010 Fall CS151
Text Editors

• Vim
  – To start: `vim <filename>`
  – 3 modes for text editing:
    • Insert (i) / Replace (r / R) /Browse (Esc)
  – To save - :w
  – To quit - :q
  – dd delete 1 line
  – ZZ save the file and quit vi

• Emacs
  – To start: `emacs <filename>`
  – Common commands:
    • Ctrl X Ctrl S: Save
    • Ctrl X Ctrl C: Exit

• You can use other GUI editors like Notepad++ which have syntax highlighting and is available on ENGR servers.
Useful vi Commands

- **arrow keys** and h,j,k,l move cursor
- **dd** delete 1 line
- **:w** save file
- **:q!** quit even if file is not saved
- **ZZ** save the file and quit vi
The GCC

• The GNU Compiler Collection (usually shortened to GCC) is a command line compiler system produced by the GNU Project, supporting various programming languages (includes C, C++).

• Compiling with GCC:
  
  gcc <list of options> sourcefile.c

  e.g.: gcc -o test test.c

• Output:
  – Compiling the code converts it into object files (*.o)
  – Linking the code uses the information from the object code to build executable.
Using the GCC compiler (cont.)

• Compile multiple files:
  1. To stop the process till compilation step:
     
     gcc –c  code1.c code2.c code3.c

  2. To link the individual ‘.o’ files to generate the executable:
     
     gcc –o executor code1.o code2.o code3.o

The same can be done in a single step:

     gcc –o executor code1.c code2.c code3.c
Examples

• Write a program to print “Hello World”.
• Compile it using “make”
• Contents of “makefile”

```
default: main
main: main.c
    gcc main.c -o main

clean:
    rm main main.o```


NOTE

You can use any IDE (Integrated Development Environment) to develop and test your C application before submitting. However, Linux is the environment in which the program will be graded. So make sure your program will compile and run without errors or warnings using GCC only on ‘flop.engr.oregonstate.edu’.
Intro 2 C

• Useful websites:
  – cplusplus.com
  – http://www.cprogramming.com/
Headers in C

• Essential header:

#include <stdio.h> : Includes the standard Input/output library. Without this statement the program will not be able to print/read data.

• Other headers & including any files : (Next class..)
Variable/function declaration

- All the variables/functions are required to be declared prior to its use in the program.

Eg.

```c
int add (int , int); // function declaration /prototyping

void main(){
    int var1 = 10;
    printf(“%d”, var1);
    int var2=20;
    printf(“%d”, var2);
    int result = add(var1, var2);
}

int add(int a, int b){
    return (a+b);
}
```

Should be placed before `void main(){ ..}`
Pointers

• A tutorial on Pointers & Arrays in C

• Example:
  int var1=10;
  int *pointertovar1;
  pointertovar1=&var1; // & is called as ‘ampersand’. It means ‘address of’
  *pointertovar1=20;   // * denotes the ‘thing pointed by’
  printf(“%d”, var1);  //Now the value of var1 becomes 20.
Memory allocation and structures

- Memory has to be managed manually due to absence of ‘Garbage collector’.
- Syntax:
  
  \[ \text{datatype} \ast \text{varname} = (\text{datatype} \ast) \text{alloc} \ (\text{sizeof}(\text{datatype})) \];

  eg. \text{struct record} \ast \text{Rec} = (\text{struct record} \ast)\text{malloc}(\text{sizeof}(\text{struct record}))

- Structures are similar to classes.
  - \text{struct record} {
    
    \text{char} \text{name}[20];
    
    \text{int} \text{id};
    
    \text{float} \text{GPA};

  }

- A structure stores only variables but no functions.
- Details of both these topics in the next session..
That’s all for today.

Please remember the earlier NOTE.