Welcome to CSCI 4061

Introductions

- Your TAs:
  - Mitch Terrell
    - terre101@umn.edu
    - Section 014 → Monday's @ 3:35 PM
    - Office Hours:
      - Tuesday 4:30 – 5:45 (remote)
      - Or by request
  - Sumanth Kaushik
    - kaush047@umn.edu
    - Section 012,013 → Monday's @ 1:25 & 2:30 PM
    - Office Hours:
      - Wednesday 3:00 – 5:00 PM (remote)
  - Andrew Bao
    - bao00065@umn.edu
    - Section 011 → Monday's @ 12:20 PM
    - Office Hours:
      - Friday 3:00 – 5:00 PM (remote)
- Come to *any* convenient office hours
- **First contact:** Piazza

Today

- Lab expectations
- Basic Unix commands
- C programming review
  - Arrays
  - Pointers
  - Structures

Lab Expectations

- Labs can be attended in person or remotely via Zoom
- Students can work in groups of up to 2
  - If you are attending via Zoom TA's can arrange students into break out rooms if requested

Lab Expectations

- Labs will have a small review section at the beginning
- The remainder of the time is for work on the lab exercise

Lab Exercises

- Each lab has an associated exercise
- It is due by midnight the day of the lab (Monday)
- You may work in groups of 2 on the exercise but each student must submit their own solution

Lab + Project Help

- Ask questions in Lab + Office Hours + on Piazza
- Utilize the “Lab Wiki” found on Canvas, the TA’s will keep this up to date as questions come in

Test Files

- We will be covering several commands today. Try them out on the machines in front of you.
- Retrieve Lab Data:
  - You can manually copy the lab code for each lab my navigating to Module # → Exercise #
    - And downloading the .tar file for each lab
  - Alternatively, the TA's are maintaining a git repository for each lab
    - In your desired folder on the CSE Lab machine call

```
$ git clone https://github.umn.edu/csci-4061-Spring-22/PostedLabs.git
```

Extract Tar Files

```
$ git clone https://github.umn.edu/csci-4061-Spring-22/PostedLabs.git
```

```
$ cd PostedLabs/01
```

```
$ tar -xzvf lab_01_files.tar.gz
```

Basic Unix Commands: ls

- ls: lists files in current directory. Use –l for more information
- Example:

```
$ ls -l
```

-rw------- 1 username grad 13 Sep 13 10:37 test.txt

permissons owner group File size Date created File name

Basic Unix Commands: chmod

- To execute a program in Unix, you need the **execute** permission.
- Change permissions using the chmod command (chmod users + - permissions) :

R W X	R W X	R W X
User (u)	Group (g)	Others (o)

- Example 1: give execute permissions to the user:

```
$ chmod +x helloWorld.c
```

- Example 2: remove read permissions from group and others:

```
$ chmod go-r helloWorld.c
```

Try it

- Go to the directory where you extracted the lab code
- List the directory contents by using ls -l.
- Observe the file permissions.
- Use chmod to remove read and write access for others for all testfiles.

Basic commands

- **`$ cd <path>`** – change directory
- **`$ rm <file>`** – deletes file
- **`$ cp <file1> <file 2>`** - copies file 1 to file 2
- **`$ mv <file1> <file2>`** - moves file1 to file2
- **`$ mkdir / rmdir <dirName>`** - creates / removes directory
- **`$ pwd`** – current working directory

Useful commands

- **`$ tar czvf compressed.tar.gz dirName`** – compresses “dirName” into archive
- **`$ tar xzvf compressed.tar.gz`** – decompresses the archive and creates directory “dirName”
- **`$ whereis programName`** – shows you the location of the program
- **`$ uname`** – operating system used
- **`$ man commandName`** **– Manual Pages. Documentation.** ***EXTREMELY USEFUL IN THIS CLASS!***

Man pages

- Instead of memorizing all Unix commands, use documentation.
- The man pages are divided into sections. They contain documentation on Unix commands, C functions and other things.
- If we want to see how to use the command mkdir, we would type: man mkdir
- If we want to see how to use the system call mkdir (in C) we would type: man 2 mkdir (in Linux), man -s 2 mkdir (in SunOS)
- 2 refers to section 2 of the man pages which contains system calls. For a complete listing of sections look up: **man man**

Shells

- A shell is a command interpreter (the window where you type commands in Unix)
- It can be used to execute programs
- There are different types of shells; commonly:
  - bash
  - tcsh
- The default shell on cselabs systems is tcsh. The underlying functionalities are very similar.
- Use echo $SHELL to see what youre using

Environment variables (1/2)

- Environment variables are variables that are visible to all programs running within that environment
- Examples: JAVA_HOME, PATH
- You can get a listing of all environment variables by running the command

```
$ env
```

Environment variables (2/2)

- In tcsh (not bash!) you can set an environment variable as follows:

```
$ setenv JAVA_HOME /home/mydir/java
```

- You can display the value of an environment variable using

```
$ echo $JAVA_HOME
```

- (note: dollar sign is needed)
- One environment variable you should know about: **PATH**.

Program execution

- Go to the directory where you extracted the test files.
- There you will find a program C file called “helloWorld.c”.
- The first thing you have to do to run a C program is to compile it
- Compile it using GCC (Gnu Compiler Collection) with the following command:

```
$ gcc helloWorld.c –o helloWorld
```

```
Translation: “Compile my c file into an executable file called helloWorld”
```

Program execution

- Alternatively, we have included a make file to compile hello world automatically… You will learn more about makefiles in class
- Try this command (remove helloWorld.out if you already made it)

```
$ make
```

```
$ ls -l
```

```
$ cat makefile
```

```
Translation: "Call make in my makefile which will compile my C
file, then print out what is in the makefile so I can observe it"
```

Program execution

- Now execute your helloWorld program:

```
$ ./helloWorld
```

```
Translation: "Please run helloWorld" [notice helloWorld is printed]
```

```
$ ls -l
```

```
Translation: Notice that helloWorld has execution permissions
```

C Programming Review

- C programming will be used heavily throughout this course.
- If you haven't programmed in C before, **Right Now is the time to learn!!** There are tutorials on the course website.
- We will review some C basics to refresh our minds.

C Programming Review

Here's a simple C program:

```
#include <stdio.h>  (When do we use "" ?)
Int main(int argc, char *argv[]) {
     printf("Hello World!\n");
     return 0;
}
```

The value argc is the number of the arguments including the command, and argv is an array of strings containing each argument.

Arrays in C

- Arrays are used to group data consecutively in memory and to provide an easy access to them.

```
int array[10];
int my_ints[] = {4,5,10,27};
int i;
for (i=0;i<4;i++)
     printf("my_ints[%d] is %d", i,
  my_ints[i]);
```

Pointers in C

- Pointers are used to reference variables by their address instead of by name.
- The *operator is used to dereference a pointer, and the &(address-of) operator is used to give the address of a variable

```
int m=7;
int *p;//declare a pointer variable
p=&m;//p now points to m
*p=*p+3;//m is now 10
```

Pointers in C

- Don’t get confused about where the * is used! What’s happening in this example?

```
int *p;
*p = 1;
```

- Why can this example go wrong?

Pointers in C

- Don’t get confused about where the * is used! What’s happening in this example?

  - `int *p; //declaring a pointer variable`
  - `*p = 1; //the value which p points to is now`

- P has to point to a valid memory location before assignment

Arrays & Pointers in C

- All arrays in C can be treated as pointer. This allows us to do pointer arithmetic

```
int a[6] = {1,2,3,4,5,6};
int i;
for(i=0;i<6;i++)
        *(a+i) = *(a+i)+1;
```

Arrays & Pointers in C

```
int a[8], x;
int *pa;
pa = &a[0]; //pa points to address of a[0], pa = a
x = *pa; //x = contents of pa(a[0] in this case)

a[i]  <-> *(a + i)
&a[i] <-> a+i
pa[i] <-> *(pa + i)
pa+i  <-> &a[i]
```

Arrays & Pointers in C

```
int a[8] = {1,2,3,4,5,6,7,8};
char c[8] = {'a','b','c','d','e','f','g','h'};
int *pa
char *pc;
pa = &a[0];
pc = &c[0];

for(i=0;i<8;i++)
 printf("%d. pa_add: %p val : %d, pc_add : %p val :
  %c\n", i, pa+i, *pa+i, pc+i, *pc+i);
```

Structures

- Structures allow the bindings of several datatypes.

```
struct complex_num {
float real;
float imaginary; };
struct complex_num name;//Don't forget struct!
name.real = 8.888;
name.imaginary = 6.666;
struct complex_num name[8]; //a structure array
   with 8 elements
name[6].real = 8.888;
name[6].imaginary = 6.666;
```

Linked List

```
typedef struct node_t {
int id;
struct node_t * next;
} node;
node *name;
name = (node *) malloc(sizeof(node));
if(name) {
   name->id = 8;// use -> node is a pointer
   name->next = NULL;}
```

How to insert and delete nodes? (pointer operations)

Linked List

- Go through linked list example provided with the test files
- Compile as:
  - gcc linked_list.c –o linked_list

Exercise

(1) Extract exercise_code.tar.gz

(2) Implement address.c and name.c based on description in the files

(3) Compile using *$ make* command

(4) Explore the Makefile and output of the above command to find the executable created

(5) Run your executable and test your code

(6) Follow these steps for submission:

1. *$ make clean* command
2. $ cd ..
3. $ tar czvf lab_1_<x500>.tar.gz exercise_code/
4. Submit your tar file on canvas
    1. Need help copying onto local machine? See Lab Wiki on Canvas

Questions?