Welcome to CSCI 4061
Announcement

- Project 1 will be online by Sept. 19
  - Form groups using Canvas
    - Groups of 3
    - Empty groups created with self sign-up enabled
  - Would be nice to have balanced groups
  - People without partners can come forward at the end

- Use ‘Project 1 Forum’ for discussions about Project - 1
  - Easy to moderate and manage
  - Reply to top level forum post for a new thread
Today

• C Programming Review
  – Arrays
  – Pointers
  – Memory Allocation
  – Structures
  – Linked List

• Compilation & Make
C Programming Review

• C programming will be used heavily throughout the course
• If you haven’t programmed in C before, **Right Now is the time to learn!!**
  • [https://www.lynda.com/C-training-tutorials](https://www.lynda.com/C-training-tutorials)
  • [https://www.tutorialspoint.com/cprogramming/](https://www.tutorialspoint.com/cprogramming/)
  • [https://www.geeksforgeeks.org/c-programming-language/](https://www.geeksforgeeks.org/c-programming-language/)
  • [https://learnxinyminutes.com/docs/c/](https://learnxinyminutes.com/docs/c/)
  • Google it !!
• We will review some C basics to refresh our minds.
Here's a simple C program:

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    printf("Hello World!\n");
    int i;
    for (int i = 0 ; i < argc ; i++){
        printf("argv[%d] : %s\n", i, argv[i]);
    }
    return 0;
}
```

- The value `argc` is the number of the arguments including the command, and `argv` is an array of strings containing each argument.
  - Minimum value of `argc` is 1 with executable name as first array element
  - Optional but good coding practice to write as arguments
Arrays in C

- Ordered Collection of Data
- Arrays are used to group data consecutively in memory and to provide an easy access to them
  - Elements access using integer index

```c
int array[10]; // Array of length 10 with garbage values
int my_ints[] = {4,5,10,27};  /* Array of length 4 with pre-defined initial values */

int i;
for (i=0 ; i<10 ; i++)  // Prints random values
    printf(“array[%d] is %d”, i, array[i]);

for (i=0 ; i<4 ; i++)  // Prints 4 5 10 27
    printf(“my_ints[%d] is %d”, i, my_ints[i]);
```

→ Run array.c
Pointers in C

• Pointers are used to reference variables by their address instead of by name
  • Stores Virtual Address of a variable

• The ‘ * ’ operator is used to dereference a pointer, and the ‘ & ’ (address-of) operator is used to give the address of a variable
• ‘*’ operator is used to dereference a pointer (Get value stored at the address)
• ‘&’ (address-of) operator is used to give the address of a variable

```c
int m = 7;
int *p;  // declare a pointer variable to an integer
printf("%d", m);  => What will be the output ?
printf("%d", p);  => What will be the output ?
printf("%d", *p);  => What will be the output ?
p = &m;  // p now points to m
printf("%d", m);  => What will be the output ?
printf("%d", p);  => What will be the output ?
printf("%d", *p);  => What will be the output ?
*p = *p + 3;
printf("%d", m);  => What will be the output ?
printf("%d", p);  => What will be the output ?
printf("%d", *p);  => What will be the output ?
p = p + 3;
printf("%d", m);  => What will be the output ?
printf("%d", p);  => What will be the output ?
printf("%d", *p);  => What will be the output ?
```
Pointers in C

• Don’t get confused about where the * is used! What’s happening in this example?
  • `int *p;`  //declaring a pointer variable to an int
  • `*p = 1;`  //the value which `p` points to is now 1
Arrays & Pointers in C

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

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

What will be the output?
Arrays & Pointers in C

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

\[
a[i]  <->  *(a + i)
&amp;i  <->  a+i    //  * and & cancel out each other
pa[i]  <->  *(pa + i)
pa+i   <->  &a[i]
\]

```c
a++  =>  What will be the output ?
pa++  =>  What will be the output ?
```
Arrays & Pointers in C

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

int i;
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);
```

→ Run array_pointer.c
Dynamic Memory Allocation

• Sometimes we don’t know how much memory we need to allocate beforehand, so we must allocate it on the fly. To do this, we use the `malloc` function.
• `malloc` returns a void-pointer which you must cast to the type of pointer you need. That pointer points to the newly allocated space in memory for your array.

```c
int *a = (int *)malloc(sizeof(int)*8);
```
Dynamic Memory Allocation

• It’s important to remember how much memory you allocated so you don’t run off the end of the array.
• Running off the end of a dynamically-allocated array could corrupt data in other parts of your program
  • extremely hard to debug!
• You must always keep a pointer that references your newly allocated array so that you can dispose of it when you are done.
  • free(ptr)
  • If pointer is lost → memory leak occurs
Dynamic Memory Allocation

• To free the allocated memory, use the \texttt{free()} system call. It takes one argument: a pointer to the allocated memory.
  \begin{verbatim}
  - free(a);
  \end{verbatim}
• It’s possible to run out of memory. \textit{It is a really really good idea to check malloc’s return value every time after allocation to see if its NULL.}

\begin{verbatim}
if(a==NULL){printf(“out of memory.”);}
\end{verbatim}

$\rightarrow$ Run \texttt{malloc.c}
More on Pointers

a) `char * arr = malloc(sizeof(char)*n);`
   Allocating n number of char, each arr[i] is char of size 1, where 0<=i<n

b) `char * arr = malloc(sizeof(int)*n);`
   Allocating n number of int, and each arr[i] is int of size 4, where 0<=i<n

c) `char **arr; // Pointer to a pointer to a char`
   arr = malloc (m * sizeof(char *)); // step 1
   for(int j=0;j<m;j++)
      arr[j] = malloc( n * sizeof(char)); // Step 2

In Step 1, we are allocating m pointers and each arr[i] is a pointer to char
In Step 2, we are allocating n number of chars, and each arr[i][j] is char of size 1, where 0<=j<n and 0<=i<=m
Run double_pointer.c  Total memory used?

Step 1

arr

1
2
3
.  .  .  m

1 2  ...  ...  ...  n

.  .  .

1 2  ...  ...  ...  n

Step 2
Structures

- Structures allow the bindings of several datatypes
- Structures are User defined data types

```c
struct complex_num {
    float real;
    float imaginary;
};

struct complex_num name; //Don’t forget struct!
name.real = 8.888; //Elements accessed using . operator
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)

    node->id is same as (*node).id
Compilation & Make Tool

• Purpose of Make tool: help a developer with compilation.
• When working on bigger projects it can take a lot of time to recompile all files…
• In most cases only a few files are actually changed by the developer.
  • The make tool keeps track of which files have been changed and recompiles only those files.
• The developer does not have to enter long compiler commands each time → makes compiling easier!
• The make tool accepts also other types of instructions that can help in automating tasks related to building of programs.
Creating a simple Makefile

- Create a file with the name of “makefile” in the directory where the source files are located.
- A simple make file might look as follows:

```
# This is how a comment looks like in a makefile
all:
    gcc helloWorld.c -o helloWorld

clean:
    rm helloWorld
```
- You will find a makefile like this in the test files in MakeLab folder!
  - `makefile.1`
Creating a simple Makefile

- all and clean are called targets
- Go into the directory where the makefile is located and enter “make” → the commands listed under “all” are executed.
- Enter “make clean” → the commands listed under the clean target are executed

- Go to MakeLab folder and try it!
  - Rename and run makefile.1
    - Run programs generated
    - Clean afterwards
Make - Variables

• We can use variables to remove redundancy in our rules. Take a look at this example:

CC = gcc
CFLAGS = -g -Wall
LDFLAGS = -lm (Note: this links the math library)
OBJJS = main.o apple.o

myprog: ${OBJJS}
${CC} ${LDFLAGS} ${OBJJS} -o myprog

main.o: main.c apple.h
${CC} ${CFLAGS} -c main.c
apple.o: apple.c apple.h
${CC} ${CFLAGS} -c apple.c

• Rename and run makefile.2
• Run programs generated
• Clean afterwards
Make - Shortcuts

• If we follow naming conventions, we can do the following:

CC = gcc
CFLAGS = -g -Wall -std=gnu11
LDFLAGS = -lm
PROGS = main
all: ${PROGS}
main: main.o apple.o
main.o: main.c apple.h
apple.o: apple.c apple.h

• Rename and run makefile.
  • Run programs generated
  • Clean afterwards

• The Make tool uses defaults to automatically compile your program using CC, CFLAGS, and LDFLAGS variables. (Naming conventions must be used for this to work properly - .o, .c, targets.)
Next Class

- Project 1
- fork / exec
- Debugging Tools
  - GDB
  - Valgrind
  - Splint
Questions?

Group Formations

Come forward and talk