UCLA CS35L

Week 5

Monday

Reminders

- Assignment 4 due this Friday (5/1)
- Assignment 5 due next Friday (5/8)

- Survey Posted by Computer Science Department under CCLE Week 4

- Anonymous feedback for Daniel
  - https://forms.gle/tZwuMbALe825DBVn8

Intro to C

What is C

- C without the plusses!
  - C++ was created to be C with extra features
- C was created in 1972 at Bell Labs by Dennis Ritchie (Ken Thompson also helped)
  - The development of C is closely tied to the development of Unix as an OS
  - Started as a new systems programming language B, but was improved and renamed C
  - Multiple versions since then
    - C89 was the first C standardized for any ANSI system
    - C99 is pretty common to work with too (Mostly C89 with added features. Like declaring variables in the for loop! And // comments)
- What is C useful for?

C vs C++

- So what did original C have?
  - Still manual memory management
  - Still a compiled language
  - Same basic data types (int, float, double, char, etc)
- What is missing that you are used to from C++?
  - Classes – C is procedural and NOT object-oriented
  - Standard Template Library (STL)
  - Exceptions
  - String objects – C uses char arrays or C-Strings
  - Stream operators like cin, cout, <<, >>
  - No bool – False is 0 and True is anything else
- Good place for syntax and tutorials
  - https://www.programiz.com/c-programming

Structs

- Structs are the closest things we have to a class in C.
  - Note – we can only pack variables in a struct. No methods

```
struct building
{
    char name[32];
    char location[64];
    int houseNum;
    struct Person owner;
}
```

typedef

- Preprocessor that lets you define custom types so you can call them easier.

```
typedef struct Point
{
    int x;
    int y;
} Point

Point p1 = {1, 2}         //declared with typedef
struct Point p1 = {3, 4}//declared with full type name
```

Pointers

- Same as C++

```
int x = 5;                  //declare int x and set to 5
int *ptrX = &x;             //declare pointer and set to address of x
*ptrX = 5*3;                //derefence pointer, and set it equal to 15
int **doublePtr = &ptrX //a pointer to a pointer
```

- You can even have pointers to functions!

Pointers

- NOTE – you can't pass by reference in C. So you need to pass by Pointer.

```
void increment(int *num)
{
    *num++;
}

x = 10;
increment(&x);
```

File I/O in C

Comes from <stdio.h>

File Pointers

- C uses File Pointers to access files in various modes
  - **r** for reading, **w** for writing, **a** for append
  - More options for more features
- Note reserved file descriptors – `stdin, stdout, stderr`

```
File* inFile = fopen("input", "r");
File* outFile = fopen("output", "w");

//do stuff

fclose(inFile);
fclose(outFile);
```

I/O in C - printf

- Most typical way to print characters/strings to stdout
- Use format specifiers to print variables passed as arguments
  - %d for int, %s for string, %c for chars, etc
  - https://www.tutorialspoint.com/c_standard_library/c_function_printf.htm

```
int year = 1992;
char *name = Daniel;

printf("I was born in %d, and my name is %s\n", year, name);
//Print - I was born in 1992, and my name is Daniel
```

I/O in C - fprintf

- Just like printf, but you can specify a FILE* as the target

```
char *name = Daniel;
FILE* fptr = fopen("someFile", "rw");

fprintf(stdout, "%s is printing to stdout", name);
fprintf(fptr, "%s is printing to file", name)
```

I/O in C – snprintf and sprintf

- These “print” to a string buffer variable
  - snprintf is more secure since you can specify the maximum size of your buffer

```c
#include <stdio.h>
char buffer1[50];
char buffer2[20];
char *s = "Daniel";

// Counting the character and storing in buffer using snprintf
int copy1 = snprintf(buffer1, 49, "%s\0", s);
int copy2 = snprintf(buffer2, 4, "%s\0", s)

// Print the string stored in buffer and character count
printf("first copy got %s in %d characters\n", buffer1, copy1);
printf("second copy got %s in %d characters\n", buffer2, copy2);

//first copy got Daniel in 7 characters
//second copy got Dan in 7 characters
```

I/O in C – getchar/putchar

- Read/Write a single character from stdin/stdout

```c
#include <stdio.h>
// implements cat: copies stdin to stdout
int main(void)
{
    char c;
    while ((c = getchar()) != EOF) // EOF char indicates end of file
        putchar(c);
}
```

Dynamic Memory in C

Note – comes from <stdlib.h>

Dynamic Memory

- We create a memory block on the heap of a specified size
- Typically used for arrays (can also be strings in C) where we don't know the size of the array until runtime
  - Like creating a user-chosen size Connect-4 board
- 4 functions to be aware of
  - `malloc()` – allocates memory, is like **new** in C++
  - `free()` – deallocates memory, is like **delete** in C++
  - `calloc()` – allocates memory, but initiliazes to 0
  - `realloc()` – reallocates previous block to a new size

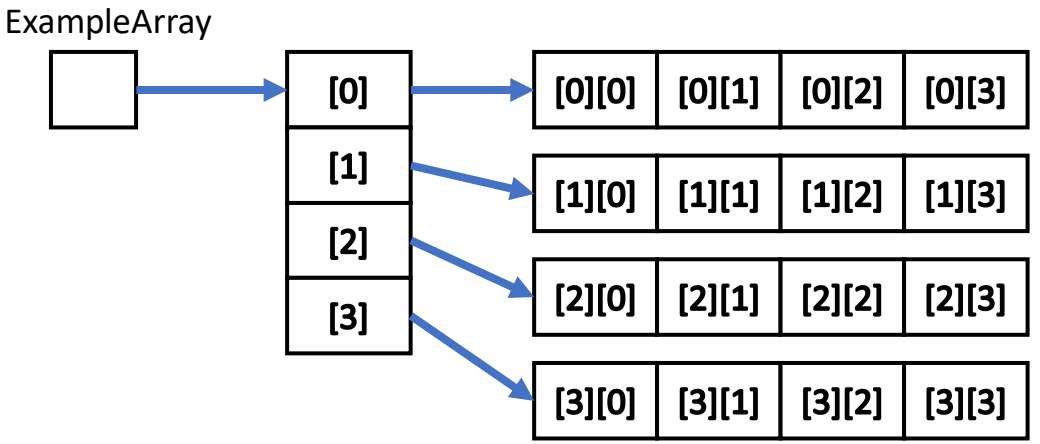
malloc

- Allocates a specific number of bytes on the heap, and returns the address pointing to that memory space.

```
//allocate a C-string
char *letters; // Pointer for memory block
int n = 5;     // Number of array elements
letters = malloc(n * sizeof(char));
// Check memory allocated correctly before you use it
if (letters == NULL)
    printf("Error allocating memory\n");

//allocate a Point struct
struct Point *p1 = malloc(sizeof(struct Point));
if (p1 == NULL)
    exit(1);
p1->x = 1;
p1->y = 2;
```

malloc 2D Array

```c
int rows = 4
int cols = 4

//create a 2D array
int** exampleArray = malloc(rows * sizeof(int *));
for (int i = 0; i < rows; i++)
{
    exampleArray[i] = malloc(cols * sizeof(int));
}

//Access elements
exampleArray[0][0] = 1;
```



free

- Deallocates memory created by malloc
- If you forget to free – you get memory leaks

```
struct Point *p1 = malloc(sizeof(struct Point));
free(p1);
```

realloc

- Change size of previously allocated memory
  - Move C-String from size 10 array to size 100
- Memory address may be different, but the contents are preserved

```
//Syntax: newPtr = realloc(oldPtr, newSize)
char* line = malloc(sizeof(char) * 10);
char* bigLine = realloc(line, sizeof(char) * 100);
```

Common Dynamic Memory Issues

- **Using Before Writing**
  - Malloc'd memory contents initialize to garbage, you need to write to it before you can try to use it.
- **Forgetting to free**
  - Causes memory leak
- **Double free** (freeing the same address twice)
  - Causes undefined behavior and maybe a seg-fault
- **Using after free** (dangling pointer)
  - causes undefined behavior
- **Reading/Writing past the sizes of your array**
  - causes undefined behavior

- NOTE – Dynamic Memory issues typically troubleshot with Valgrind

Function Pointers

Passing function as arguments

- Just like in languages like JavaScript, or Python, we can pass functions as arguments in C. We just need to use function pointers.

```c
int add(int x, int y)
{
    return x + y;
}
int main()
{
    int (*add_fn_ptr)(int, int) = &add;
    int sum = (*add_fn_ptr)(2, 3);
}
```

More useful example of a function pointer

```c
int compare(const void *a, const void *b)
{
    return (*(int *)a - *(int *)b);
}
int main()
{
    int values[] = {40, 10, 100, 90, 20, 25};
    qsort(values, 6, sizeof(int), compare);
    int i;
    for (i = 0; i < 6; ++i)
    {
        printf("%d\n", values[i]);
    }
    return 0;
}
```

- Note qsort code can be found:
  - https://pubs.opengroup.org/onlinepubs/009695399/functions/qsort.html

Compiling in C

General Overview

- Very similar compilation process between C and C++ (preprocess, convert to assembly code, convert to binary object file, link objects together into executable binary)
- Instead of g++, use gcc
- Example:
  - `gcc -g -Wall -std=c99 source.c -o outputExec`
  - `-Wall` -> Display compiler warnings
  - `-g` -> Debug with GDB
  - `-std=c99` -> use C99 Standard
  - `-o` -> specify output file name