C for C++ Users

Some C++ Only Features

- Boolean type
- Classes & Objects
- namespaces and using statements
- exception handling (with `try, catch, throw`)
- using `new` and `delete` for dynamic memory management
- templates
- Function Overloading
- Pass by Reference
- Default Paramenters

The c style implementation of libraries look like
`# include <stdio.h>`
There are 4 diffrent styles of casting (static, dynamic, const, and reinterpreted casts)

Declerations

- While declaring arrays, the size of the array should be specified as `const` or a literal. The size of an array may not be variable

`int size =100;`
`int list[size];` - Would NOT work
`int list[100];` - Would work

> *Unlike c++, C requires variable declarations at the top of the functions*
> *This means that, ALL variables should be declared at the beginning of the function*

`for (int i=0; i< 10; i++){} // wrong`

`int i;`
`for(i = 0; i < 10; i++){} // correct`

Standard I/O streams

`stdin::` input stream
`stdout::` output stream
`stderr::` error stream

Streams are sequences of characters flowing from one place to another `stdio.h` - contains basic I/O functions

Formatied I/O - refers to the conversion of data to and from a stream of characters for printing

`scanf:` - Reads from standard input (stdin)
`printf:` - writes to standard input (stdout)

Output with printf

`printf (format_string, list_of_expressions);`

- format_string is the layout of whats being printed
- list_of_expressions is a list of comma seperated variables

Specifier	Purpose
%d	int (signed decimal int)
%u	unsigned decimal int
%f	floating point (fixed notation)
%e	floating point (exponential notation)
%s	string
%c	char
%x	Hexidecimal number

- We can specify the field width. Defaults to right-justification (use - for left), we use this syntax:

`printf(%10d)` - for 10 spaces

Using scanf basics

`scanf (format_string, list_of_variable_addresses)`

- format string is like that of `printf`
- But instad of expressions, we need space to store incoming data, hence the list of variable addresses.

`int month, day;`
`printf("Please enter birth month, followed by birthday");`
`scanf(%d %d, &month, &day);`

Conversion Specifiers

- Mostly the same for output, Some small diffrence
- Use %f for type float, but use %lf for types double and long double

C Strings
`char word1[20];`
`scanf("%s", word1`);`

- Similarly you can read a string into a char array with scanf.
- Charecters are read from the keyboard until the first white space, the null character is automatically placed at the end for termination

`char greeting[] = "Hello";`
`printf("%s", greeting); // Prints the word`
Output: "Hello"

- The format string used for a scanf command is essentially a regular expression
- The data from stdin comes in as a string. The conversion specifier converts the data into the correct "type".
- If we specifiy a particular charecter in the format string, scanf will ignore that character if it occurs at the same spot
- We can also instruct scanf to look for or ignore particular characters
- **Example**
  `scanf("%25[^*]*%s", str1, str2);`

> *C-Strings dont need the address because the name acts as a pointer to the array*

Structs and enums

- C & C++ definitions of a struct are the same
- In C++, declaring a struct or an enum automatically creates a new type
- In C, we need to remind the compiler about the user defined types everytime we use them

> *We can define a new type in C using* `typedef`

```
typedef struct student {
char name[20];
double gpa;
} Student; // now the new type name
struct student s1; // C declaration using "tag"
Student s2; // C declaration using new type name
```

Dynamic Memory Allocation

- C does not come with the `new` and `delete` operators. We need to use `malloc` and `free`.
- These functions are in `` - The syntax for an array of ints is: `line * arr = (line*) malloc

(size_of_array * sizeof(int));`

For an arrau of the line strict, which has been typedef'ed is:
`line * arr = (line*) malloc (size_of_array * sizeof(line));`

Index