C Concepts - I/O

Lecture 37
COP 3014 Spring 2017

April 20, 2017

C vs. C++: Some important differences

- C has been around since around 1970 (or before)
- C++ was based on the C language
- While C is not actually a strict subset of C++, most C code can be handled by a C++ compiler
- The older C-style way of including libraries looks like this:

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <ctype.h>
```

- The 4 different styles of casting are a C++ feature. C casts look like this

```
double val = 5.783;
int integer = (int) val;
```

C vs. C++: Some important differences

Some of the C++-Only features are:

- The Boolean type
- Classes and 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 parameters

structs and enums

- In both C and C++, the struct definition is 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 type every time we use it.
- We can define a new type in C using the `typedef` keyword.

```
typedef struct student // structure "tag"
{
      char name[20];
double gpa;
} Student; // this is now the new type name

struct student s1; // C declaration using "tag"
Student s2; // C declaration using new type name
```

Basics of Formatted Input/Output in C

- I/O is essentially done one character (or byte) at a time
- **stream** – a sequence of characters flowing from one place to another
  - *input stream*: data flows from input device (keyboard, file, etc) into memory
  - *output stream*: data flows from memory to output device (monitor, file, printer, etc)
- **Standard I/O streams** (with built-in meaning)
  - `stdin`: standard input stream (default is keyboard)
  - `stdout`: standard output stream (defaults to monitor)
  - `stderr`: standard error stream
- *Formatted I/O* – refers to the conversion of data to and from a stream of characters, for printing (or reading) in plain text format
  - All text I/O we do is considered formatted I/O
  - The other option is reading/writing direct binary information (common with file I/O, for example)

C I/O

`stdio.h` – contains basic I/O functions

- `scanf`: reads from standard input (stdin)
- `printf`: writes to standard output (stdout)
- There are other functions similar to printf and scanf that write to and read from other streams
- How to include, for C or C++ compiler

```
#include <stdio.h> // for a C compiler
#include <cstdio> // for a C++ compiler
```

Output with printf

- The basic format of a printf function call is:

  ```
  printf (format_string, list_of_expressions);
  ```

  where:
  - *format_string* is the layout of what's being printed
  - *list_of_expressions* is a comma-separated list of variables or expressions yielding results to be inserted into the output
- To output string literals, just use one parameter on printf, the string itself

  ```
  printf("Hello, world!\n");
  printf("Greetings, Earthling\n\n");
  ```

Conversion Specifiers

A conversion specifier is a symbol that is used as a placeholder in a formatting string. For integer output (for example), %d is the specifier that holds the place for integers.

Here are some commonly used conversion specifiers (not a comprehensive list):

Specifier	**Purpose**
%d	int (signed decimal integer)
%u	unsigned decimal integer
%f	floating point values (fixed notation) - float, double
%e	floating point values (exponential notation)
%s	string
%c	character

Printing Integers

- To output an integer, use %d in the format string, and an integer expression in the *list_of_expressions*.

```
int numStudents = 35123;
printf("FSU has %d students", numStudents);
// Output:
// FSU has 35123 students
```

- We can specify the field wicth (i.e. how many 'spaces' the item prints in). Defaults to right-justification. Place a number between the % and the d. In this example, field width is 10:

```
printf("FSU has %10d students", numStudents);
// Output:
// FSU has       35123 students
```

Printing Integers

- To left justify, use a negative number in the field width

```
printf("FSU has %-10d students", numStudents);
// Output:
// FSU has 35123       students
```

- If the field width is too small or left unspecified, it defaults to the minimum number of characters required to print the item:

```
printf("FSU has %2d students", numStudents);
// Output:
// FSU has 35123 students
```

- Specifying the field width is most useful when printing multiple lines of output that are meant to line up in a table format

Printing Floating-point numbers

- Use the %f modifer to print floating point values in fixed notation:

```
double cost = 123.45; printf("Your total is $%f
today\n", cost);
// Output:
// Your total is $123.450000 today
```

- Use %e for exponential notation:

```
printf("Your total is $%e today\n", cost);
// Output:
// Your total is $1.234500e+02 today
```

Note that the e+02 means “times 10 to the 2nd power”

Printing Floating-point numbers

- You can also control the decimal precision, which is the number of places after the decimal. Output will round to the appropriate number of decimal places, if necessary:

```
printf("Your total is $%.2f today\n", cost);
// Output:
// Your total is $123.45 today
```

- Field width can also be controlled, as with integers:

```
printf("Your total is $%9.2f today\n", cost);
// Output:
// Your total is $      123.45 today
```

- n the conversion specifier, the number before the decimal is field width, and the number after is the precision. (In this example, 9 and 2).

Printing characters and strings

- Use the formatting specifier %c for characters. Default field size is 1 character:

```
 char letter = 'Q'; printf("%c%c%c\n", '*',
letter, '*');
// Output is:  *Q*
```

- Use %s for printing strings. Field widths work just like with integers:

```
printf("%s%10s%-10sEND\n", "Hello", "Alice",
"Bob");
// Output:
// Hello      AliceBob      END
```

scanf basics

- To read data in from standard input (keyboard), we call the scanf function. The basic form of a call to scanf is:

  ```
  scanf(format_string, list_of_variable_addresses);
  ```

  - The format string is like that of `printf`
  - But instead of expressions, we need space to store incoming data, hence the list of variable addresses
- If x is a variable, then the expression &x means "address of x"
- scanf example:

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

scanf Basics

- Conversion Specifiers
  - Mostly the same as for output. Some small differences
  - Use %f for type float, but use %lf for types double and long double
- The data type read, the conversion specifier, and the variable used need to match in type
- White space is skipped by default in consecutive numeric reads. But it is not skipped for character/string inputs.

printf/scanf with C-strings

- An entire C-style string can be easily printed, by using the %s formatting symbol, along with the name of the char array storing the string (as the argument filling in that position):

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

- Be careful to only use this on char arrays that are being used as C-style strings. (This means, only if the null character is present as a terminator).

printf/scanf with C-strings

- Similarly, you can read a string into a char array with scanf. The following call allows the entry of a word (up to 19 characters and a terminating null character) from the keyboard, which is stored in the array word1:

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

- Characters are read from the keyboard until the first "white space" (space, tab, newline) character is encountered. The input is stored in the character array and the null character is automatically appended.
- Note also that the & was not needed in the scanf call (word1 was used, instead of &word1). This is because the name of the array by itself (with no index) actually IS a variable that stores an address (a pointer).