Lecture 7: Arrays, strings, and functions

Arrays

Arrays are declared to store a certain number of variables of a specified type either on the **stack**, which is the local static memory that is allocated to run your program when it executes, or in the **heap**, which is memory that can be dynamically allocated and deallocated as your program runs. To declare an array called `values` that contains 5 floating point elements, you would use

```
float values[5];
```

This declaration requires that your program statically allocate a total of 20 bytes of memory (5 floats $\times$ 4 bytes/float = 20 bytes) at runtime. You can also initialize arrays to contain given values, such as

```
float values[] = {1,2,3,4,5};
```

which will allocate enough space to store the 5 numbers shown as floating point numbers. You can access these values and print them, for example, in a loop, as follows

```
for(i=0;i<5;i++)
  printf(``%f\n'',values[i]);
```

To declare multidimensional arrays on the stack, you would use

```
float values[5][3];
```

and you can access these values in a similar manner with

```
for(i=0;i<5;i++)
  for(j=0;j<3;j++)
    printf(``%f\n'',values[i][j]);
```

Strings

Strings are character arrays that contain the null character `'\0'` as the final character. They are **NOT** simply character arrays. It is important to understand the difference between a simple array of characters and a string. An array of 5 characters is declared with

```
char string[5];
```

this is **NOT** a string until the last character is the null character, or unless

```
string[4]='\0';
```

If you attempted to print an array of characters that is not terminated with the null character, then you get nonsense. Strings can be declared in a similar manner to arrays, as in

```
char string[] = ``Hello!'';
```

This is known as a declaration of a string constant. The string "Hello!" has a total of 6 characters, but the actual storage includes 7 characters, because the compiler adds on the null character `'\0'`. This is equivalent to the commands

```
char string[7];
string[0]='H';
string[1]='e';
string[2]='l';
string[3]='l';
string[4]='o';
string[6]='!';
string[7]='\0';
```

Note that since each element of a string is a character, if you wanted to determine whether or not a particular string contained a given character, you would use a loop such as

```
i=0;
while(string[i++]!='l') ;
printf(``Found c at location %d\n'',i-1);
```

Functions for use with strings

The standard library `string.h` contains a host of commands that are very useful when dealing with strings. To obtain the length of a string, you use

```
strlen(string);
```

This is useful when performing an operation on each individual character in a string, such as

```
for(i=0;i<strlen(string);i++)
  printf(``%c\n'',string[i]);
```

To copy the contents of one string into another, equating them **does not work!**. For example, the following code will **not work**:

```
char string1[10], string2[10];
string2 = string1;
```

Instead, each character from `string2` must be copied into `string1` with

```
for(i=0;i<strlen(string1)+1;i++)
  string2[i]=string1[i];
```

Note that the `for` loop ends with the condition `i<strlen(string1)+1` rather than with `i<strlen(string1)` so that we can be sure to copy the null character. The standard library `string.h` provides the useful function that does this for us as

```
strcpy(string2,string1);
```

which copies `string1` into `string2`.

The other useful function (among many others) is `strcmp`, which enables us to compare the equality of strings. This function is used as

```
if(strcmp(string1,string2)>0)
  printf(``string1 > string2\n'');
else if(strcmp(string1,string2)==0)
  printf(``string1 == string2\n'');
else if(strcmp(string1,string2)<0)
  printf(``string1 < string2\n'');
```

The equality of `string1` and `string2` implies that they are identical, while if `string1` is less than `string2`, then `string1` comes before `string2` alphabetically, and vice-versa when `string1` is greater than `string2`.

Functions

Functions in C must be declared at the beginning of your source code. The format of a declaration is given by

```
returntype FunctionName(argtype arg1, argtype arg2,...,etc...);
```

The actual function definition is declared in an identical manner with braces used to define the function, as in

```
returntype FunctionName(argtype arg1, argtype arg2,...,etc...) {

  returntype returnvariable;

  code for function...

  return returnvariable;
}
```

As an example, let's say we would like to read in a string from the standard input and print out the first location of a specified character in that string. The functions in the header file

```
/home/cos315/include/cos315.h
```

contain some useful operations and string types that we can use for these purposes. When we discussed the declaration of strings before, the memory allocated for these strings was all allocated on the stack. The functions in the `cos315.h` header file can be used to allocate space for strings dynamically in the heap. To declare a string using this header file and allocate space for it in the heap, we would use

```
string line = NewString();
```

This uses the new type called `string` in the header file. We will discuss the details of this declaration in more detail when we cover pointers in the next lecture. We need to be sure, however, that every time we allocate something in dynamic memory that we free up the memory associated with that string with

```
FreeString(line);
```

Once we allocate space for a string, we can read in a string from the command line with

```
GetLine(line,stdin);
```

which places the next line read in from `stdin` into the string `line`. The following code uses these functions to create the `find` function which finds the location of the occurence of a particular character in a given string.

```
#include<stdio.h>
#include "cos315.h"

#define TRUE 1

/*
 * Function declarations.
 *
 */
int find(string str, char c);

/*
 * Main function.
 *
 */
int main(void) {
  int location;
  char c = 'l';
  string line = NewString();

  printf("Type quit to exit.\n");
  while(TRUE) {
    printf("String: ");
    GetLine(line,stdin);

    if(!strcmp(line,"quit\n"))
      exit(0);

    location = find(line,c);
    if(location == -1)
      printf("Character %c not found in string %s\n",c,line);
    else
      printf("The character %c is located at index %d in string %s\n",
             c,location,line);
  }
  FreeString(line);
}
```

```
/*
 * Function: find
 * Usage: loc=find(string,'c');
 * ----------------------------
 * This function returns the first index at which the specified
 * character exists in the given string.  If it is not found
 * then a -1 is returned.
 *
 */
int find(string str, char c) {
  int i;
  for(i=0;i<strlen(str);i++)
    if(str[i]==c)
      return i;
  return -1;
}
```

This code is available on the class server as the file

`/home/cos315/assignments/assignment2/find.c`

To compile this code, you need to link it with the precompiled object file on the class server and you also need to tell it where to find the header file with the `-I` option, as in

```
$ gcc find.c /home/cos315/include/cos315.o -I/home/cos315/include
```