C Tutorial

This guide attempts to outline a few things that may help you get started with the C programming language, It is not meant to act as a complete tutorial, instead it is a guide for those who already know C++.

If you have never programmed in C before and this guide is not enough to help you, I strongly suggest that you buy a copy of “The C Programming Language” second edition, by Kernighan and Ritchie. A cheap copy can usually be found at www.barnesandnoble.com or www.half.com.

C Hello World

The first program that you always start with when learning a new language is hello world.

```
#include<stdio.h>

int main()
{
   printf("Hello World!\n");

   return 0;
}
```

You'll notice that C is very similar to C++. One of the most notable differences is the use of the printf function which is used to print information to the screen. C does not have the concept of a stream which means that cin and cout cannot be used.

The printf function can also be used to print variables as well. here is an example.

```
#include<stdio.h>

int main()
{
   int x = 5;

   printf("Hello World! x=%d\n", x);

   return 0;
}
```

The '%d' inside the string passed to printf gets replaced with the value of the variable passed as the second argument (in this example, x). If you need to print a character you place a '%c' where you want that character to appear. If you need to print a string you place a '%s' where you want that character to appear.

If you need to print more than one variable in a single printf call you simply include multiple '%' statements in your string and then give all of the needed arguments to printf, Here is an example.

```
#include<stdio.h>

int main()
{
   char* name = "Bob";
   int age = 25;
   char mi = 'c';

   printf("My name is %s, my age is %d and my middle initial is
            %c.\n", name, age, mi);

   return 0;
}
```

The printf function can take as many arguments as needed to print as many variables as you want but keep in mind that the order of those variables is very important. Our string contains a %s, %d, and %c in that particular order. So printf is expecting to find a string, an integer, and a character as its second, third and fourth arguments.

C strings

C does not have an official 'string' data type. Instead strings are implemented using an array of characters. Here is a typical string declaration and initialization.

```
char* name = “bob”;
```

Comparing C strings

So what if you want to check the value of a string to see if it is equal to “foo”? Can you use the following code?

```
char* x = “bar”;
if(x == “foo”)
```

Absolutely not! Remember, x is really a pointer to a character. If you wish to compare x to “foo” you will have to compare each and every character of x to “foo”. You can easily write a for loop to do this, or you can use the strcmp function.

```
#include<stdio.h>    /* printf */
#include<string.h>   /* strcmp */

int main()
{
   char* x = "bar";
   int retval = strcmp(x, "foo");

   if (retval == 0)
      printf("The two strings are equal.\n");
   else
      printf("The strings are NOT equal.\n");

   return 0;
}
```

If the strcmp function returns 0, then the two strings are equal. If it returns any number other than 0, the strings are not equal. For more information see the man page for strcmp.

Copying

Since strings are just arrays which are implemented using pointers, you cannot create a copy of a string with the = operator. This will just cause the pointer pointing to the string to be copied. If you wish to deep copy a string, you will need to either copy each individual character, or use the strcpy function.

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

int main()
{
   char* x = "hello";
   char* y = (char*)malloc( sizeof(char) * 10 );

   strcpy(y, x);

   int retval = strcmp(x, y);
   if (retval == 0)
      printf("The copy worked.\n");
   else
      printf("The copy did NOT worked.\n");

   free(y);
   return 0;
}
```

Ignore the malloc() line for now. it just creates a string of size 10.

Notice the unusual order of arguments for strcpy. It takes the destination string as the first argument and the source string as the second argument.

String length

Getting the length of a string as as easy as a call to strlen.

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

int main()
{
   char* x = "hello";

   int length = strlen(x);
   printf("The string is of length %d\n", length);

   return 0;
}
```

String concatenation

Concatenating two strings together requires a call to strcat.

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

int main()
{
   char* x = (char*)malloc( sizeof(char) * 10 );
   char* y = (char*)malloc( sizeof(char) * 10 );

   strcpy(x,"he");
   strcat(x,"llo");

   printf("%s\n",x);
   free(x);
   free(y);
   return 0;
}
```

Other string functions

index - Locate a character in a string.

Use 'man string' to get more information.

Command Line Arguments

Processing arguments in C is very similar to processing arguments in C++. Here is an example program that will print out all of it's arguments on separate lines and then terminate.

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

int main(int argc, char* argv[])
{
   printf("argv[0] (program name) = %s\n", argv[0]);

   int i;
   for(i=1; i < argc; i++)
   {
      printf("argv[%d] (argument %d) = %s\n", i ,i, argv[i]);
   }

   return 0;
}
```

The very first argument (argv[0]) will always be the name of the program (executable file) that is being executed. All of the other arguments will be put in argv[1], argv[2], argv[3], etc. The argc integer tells you how many arguments to expect.

Files

In c++, you used ifstream and ofstream to access files. In C, we will use other methods. You should include 'stdio.h' to work with the methods described below.

Opening files

The first thing you need to do to work with a file is to open it. If you want to open a file for reading, you will need a line of code similar to this.

```
FILE* in = fopen("myfile.txt","r");
```

This line of code creates a file pointer and then uses the fopen function to obtain a file pointer to “myfile.txt” for reading. The “r” represents the mode that we wish to use for opening the file. Mode “r” allows you to read the file and mode “w” allows you to write the file. Consult the 'fopen' man page for more information on file modes.

Reading files

Now that you've opened the file for reading, you can read from the file with a variety of different methods. To find out more about these methods check the 'gets' manpage. We will be using the fgetc method which allows you to read through a file one character at a time.

```
#include<stdio.h>

int main()
{
   FILE * in = fopen("myfile.txt","r");

   char currentCharacter;

   while(!feof(in))
   {
      currentCharacter = fgetc(in);
      printf("%c", currentCharacter);
   }

   return 0;
}
```

Writing files

writing files is similar to reading them. Instead of giving the “r” mode to fopen, you give it the “w” mode. It is important to note that the “w” mode will cause the contents of the specified file to be destroyed. If you wish to append data you should use the “a” mode.

Just like there is a collection of get functions for reading data from a file, there is also a collection of put functions for writing data to a file. Read the 'puts' man page for more information. Here we will use the fprintf function which allows you to write an entire string to a file.

```
#include<stdio.h>

int main(int argc, char* argv[])
{

   if (argc < 2)
   {

      printf("This program writes its arguments to args.txt\n");
      printf("You must specify at least one argument.\n");
   }

   FILE * out = fopen("args.txt","w");
```

```
        int i;
        for(i=1; i < argc; i++)
        {
           fprintf(out,"%s\n",argv[i]);
        }

        return 0;
     }
```

File Position Indicator

When you open a file the file position indicator is set to point to the first character/byte in the file. As you read through a file, the file position indication is updated to point to the next character. If you read the first 100 bytes, the position indicator will point to the 101st byte/character of the file.

There are methods available for manipulating the position indicator. the most important one is fseek which allows you to skip over a specified number of bytes in a file. For more information on fseek and other position indicator manipulation functions read the man page on fseek.

Note: The section titled 'Memory Management' was taken from the following location: http://www2.sis.pitt.edu/~ir/KS/Data/RMiles/c13.html *The text has been slightly modified in some places put the material largely remains the same.*

Memory Management

Sometimes you will come across situations where you do not know when the program runs exactly how much memory your program will need, for example you may want the user to tell you how many items he or she wants to store, and then get a chunk of memory just the right size. This is the most efficient way of writing your programs, it means that you do not use more resources than required. When a C program runs, a number of services are made available to it by the system it is running on. One of these is a memory manager. The memory manager looks after what memory is used by who. On a single user system, like the IBM PC, this is a very simple affair, either you have the memory or the system does. On a multi-user system, like UNIX, things are a lot more complicated. You however do not need to worry about this, you simply ask the memory manager for a chunk of memory and it will give it to you if there is enough left. When you have finished with the memory, it is nice if you give it back so that other programs can use it.

The include file 'alloc.h' contains all the memory allocation routines. You can find out how much memory is left, the size of the biggest block available and things like that. We are only going to use two, malloc and free.

malloc

This function will try to get a chunk of memory for you. It returns a pointer to the memory it found, if it could get some. If not it returns a NULL pointer. You must always check to make sure that malloc worked before you use the memory you think you got! malloc has one parameter, the size of the block of memory you want. You can use sizeof to find out how much memory you need, for example :

```
sizeof( int)*99
```

This would be enough space for 99 integers. Because malloc returns a pointer to an item of type void you must use the cast facility to make this compatible with the type you really want; i.e. if you want to store 99 integers you will have a pointer to integers which you want to make point at the block of memory which malloc is going to get for you. The cast will make sure that the compiler does not produce an error when it sees the assignment, for example :

```
int* buffer;
buffer = (int*)malloc(sizeof(int) * 99);
if ( buffer == NULL )
{
   printf("No memory!\n") ;
}
else
{
   /******
   do something with the memory...
   *******/
}
```

I would access an item in buffer in exactly the same way as I would access elements in any block of memory or array, for example buffer [20]. Note that the effects of going over the end of a block of memory which has been fetched in this way as just the same as any other - your program will do stupid things and then fall over! Note also that you have no idea what is in the memory when you get it! Do not assume that it has been emptied of silly values. There is a function, calloc, which you can use if you want all your memory to be cleared before you get it. This function is used differently from malloc, so you will have to look it up.

free

When you have finished with some memory you can give it back to the memory manager by the use of the free function, for example :

```
free(buffer);
```

This would give back the memory we requested above. Note that if we try to use this memory again, it may have been used for something else and so this would cause big problems. It is good housekeeping to always give back memory which you have allocated. If you do not you might find that the total amount of memory available will drop to a point where your program will not run any more.

Other Resources

If you need more information you should check the following resources.

man pages

You can use the 'man' command under Linux/Unix to obtain useful information about many C functions and libraries. Some of the following may be of interest to you.

strings	getcwd
strstr	fork
strlen	execvp
strcpy	wait
strchr	waitpid
strcmp	kill (use this command 'man 2 kill')
fopen	dup2
fclose	waitpid
fread	
fwrite	
fprintf	
puts	
gets	

Useful websites

Strings
http://cermics.enpc.fr/~ts/C/FUNCTIONS/function.ref.html
(look at string.h section)

Memory management
http://www2.sis.pitt.edu/~ir/KS/Data/RMiles/c13.html

files
http://www.cprogramming.com/tutorial/cfileio.html
vergil.chemistry.gatech.edu/resources/programming/c-tutorial/io.html

Command Line Arguments
http://www.eskimo.com/~scs/cclass/notes/sx13.html
http://juicystudio.com/tutorial/c/command.asp

Process Control
http://jan.netcomp.monash.edu.au/ssw/processes/unix.html
http://www.cs.uleth.ca/~holzmann/C/system/pipeforkexec.html
http://www.dgp.toronto.edu/~ajr/209/notes/proc.html

General
http://www2.sis.pitt.edu/~ir/KS/Data/RMiles/contents.html
http://www.physics.drexel.edu/courses/Comp_Phys/General/C_basics/
http://www.its.strath.ac.uk/courses/c/

Books

The C Programming Language second edition
Brian W Kernighan & Dennis M. Ritchie
Prentice Hall
http://search.barnesandnoble.com/booksearch/isbnInquiry.asp?isbn=0131103628