While loop

Unlike the `for` loop which executes a block of code a specific number of times, the `while` executes a block of code until a condition is false. To count to ten, as in the last example with a `for` loop, the code would be:

```
temp=1;
while (temp<=10) {
        printf("%d\n",temp);
        temp=temp+1;
}
```

A while loop may never be executed if the condition is false.

Arrays

Arrays organize data. They can create strings and hold lists of numbers. Use brackets (`[]`) to create arrays. In between the brackets goes a number specifying the number of elements in the array. The following shows how to create a string:

```
char str[80];
```

Arrays can be indexed into. For example `String[0]` is of type `char` and is the first element in the array `String`. `String[80]` does not exist because C starts counting at 0. The index in the brackets can be a mathematical function or a return value for a function To access the last element in a string:

```
char my_char;

my_char = str[strlen(String)-1];
```

The function `strlen()` return the length of a string.

Strings

C defines strings by using a character array and whose contents is ASCII characters terminated by the `NULL` character. This does not mean that the 80th element in the above declaration must be `NULL`. It means that the last character of the string that you create needs to be `NULL` and can be shorter than the array. `NULL` is also used as a pointer so watch out! C has a few functions to manipulate strings with, for example `strcpy()`, `strcat()`. `strcpy`() takes two arguments and copies one string into another.

```
Char str2[20];

strcpy(str2,str1);
```

copies `str1` to `str2`. `strcat()` tacks on the string of the second argument to the first.

Conditional statements

The `if` statement allows for program flow control. The `if` statement will execute the instructions in the block after the `if` statement if the condition is true. For example:

```
if (test==1) {
        printf("Test is true\n");
}
```

In the condition statement (always contained in parenthesis) uses a double equal sign. The single equal sign is used for assignment and the double equal sign is use to test a condition. Watch out for these pitfalls:

```
if (temp=1) {  // Assigns 1 to temp
}
```

This snippet of code above does not test `temp`, it assigns it the value `1`.

```
if (temp==1); {        // Tests temp and then continues to the braced block.
}
```

This code snippet above will always execute the code in the block after the `if` statement because of the semicolon after the condition.

Using a while loop, create a function that reverses the string "Hello World!" Print the resulting reversed string in `main()`. The resulting reversed string should be contained in `String` in the function `string_reverse()` when it `returns()`. You will need to use the `if` statement. Create a function that changes a string to all uppercase. Mail me the source code and print out of the results. Here's a little help:

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

int string_reverse(char * String);
int string_upper(char * String);

// A program to show string reversal and capilatization
int main(int argc, char * argv[])
{
        char str[80];

        // copy the string to our storage
        strcpy(str,"Hello World!");
        // print the string
        printf("%s\n",str);
        // Reverse the string
        string_reverse(str);
        printf("%s\n",str);
        // Print it again
        string_upper(str);
        printf("%s\n",str);
}

// Reverse a string
int string_reverse(char * String)
{
        // Reverse a string

        // your code here

        // if you printf("%s\n",String) it should be reversed now - for testing
        return(0);
}

// Capitalize a string
int string_upper(char * String)
{
        // Upper case a string

        // your code here

        // if you printf("%s\n",String) it should be uppercase now - for testing
        return(0);
}
```

Next time we'll look at pointers and why `string_reverse()` has a `char * String` for an argument.

Getting Help

All ANSI C functions are documented in linux. Just type `man <function>` to get more help than you ever wanted. The man page will tell which header file to include, which library to link, describe the arguments and return value as well as describing the function itself.

```
# man strlen
```