A Crash Course in C

Steven Reeves

This class will rely heavily on C and C++. As a result this section will help students who are not familiar with C or who need a refresher. By the end of this section you will be able to write a simple C program to do scalar multiplication/array addition.

1 Fundamentals of C

Example Program:

```
int main()      //Declaring the main program
{
        int x;          /*declaration of an integer x*/
        x = 5;          /* assignment */
        printf("%d\n", x); /* output of value to the terminal %d\n designates int and new line*/
}
```

In this example program, we see the basic mechanics of C. Each line is terminated by the semicolon character. Code segments are contained in curly braces: that is all code contained in the main program is deliminated by the curly braces.

1.1 Data Types

1.1.1 Basic Data Types

The standard data types in C are:

- int for 32-bit integer
  - unsigned int for 0 to 4,294,967,295
  - int for 2,147,483,648 to 2,147,483,647
- float for 32-bit floating point number
- double for 64-bit floating point number
- char 8-bit variable, letters or integer from -128 to 127

1.1.2 Derived Data Types

- Array
  - basictype[number of entries]
  - gives number of entries of basictype
  - double[10] = double0, ..., double10
- Pointer
- Structure

1.1.3 Pointers

A pointer is a variable whose value is the address of another vairable, ie the direct address of a memory location. Pointers are important, and are the only way to do dynamic memory allocation in C. Every variable is a memory location, and has an address associated with it, this address can be accessed by using the (&) operator.

To denote a pointer we use the code

```
type *var-name;
//i.e.
double *dp; /*pointer to double*/
int *ip; /*pointer to integer*/
float *fp; /*pointer to float*/
char *ch; /*pointer to character*/
```

We can write this code to illustrate the use of pointers.

```
#include <stdio.h>

int main () {

   int  var = 20;   /* actual variable declaration */
   int  *ip;        /* pointer variable declaration */

   ip = &var;  /* store address of var in pointer variable*/

   printf("Address of var variable: %x\n", &var  );

   /* address stored in pointer variable */
   printf("Address stored in ip variable: %x\n", ip );

   /* access the value using the pointer */
   printf("Value of *ip variable: %d\n", *ip );

   return 0;
}
```

2 Headers

Notice in the previous code we had the segment `#include <stdio.h>`. A header file is a sfile with the extension .h, and contains C function declarations adn macro definitions to be shared between several source files. There are two types of headers, one that the programmer writes and the headers that come with the compiler.

You request to use a header file in your program by including it with the C preprocessing directive `#include`, like you have seen inclusion of stdio.h header file, which comes along with your compiler.

Including a header file is equal to copying the content of the header file but we do not do it because it will be error-prone and it is not a good idea to copy the content of a header file in the source files, especially if we have multiple source files in a program.

A simple practice in C or C++ programs is that we keep all the constants, macros, system wide global variables, and function prototypes in the header files and include that header file wherever it is required.

To use a header we use the compiler directive `#include <filename.h>` for system header files, and `#include "filename.h"` for programmer written headers.

If a header file happens to be included twice, the compiler will throw an error, in larger codes this may be challenging. Using the compiler directives can alleviate this error. We use `#ifndef` within the header file. For example:

```
#ifndef HEADER_FILE
#define HEADER_FILE

the entire header file file

#endif
```

The directive `#ifndef` is "if not defined", similarly theres a directive for "if defined", `#ifdef`, which is useful for conditional building. Both must be terminated by `#endif`.

3 Loops, and conditionals

There are two main types of loops in C, the `for` and `while` loops.

The for loop is the most straight forward, in which it iterates until it hits a certain number. That is:

```
for(int i = 0; i < iend; i++)
{
        //CODE
}
```

To initiate the loop, we type in `for` then we give the iterative conditions. Within the for loop we give an integer i, and that each iteration, i is increased by 1, until it is equal to iend. The ++ operator increases the variable by 1.

Before we talk about while, we will discuss the if conditional. If is a way to swap between code given a certain conditional statment. A general use of if:

```
if(statment){
        //CODE
}
else if(other statement)
{
        //Different CODE
}
else
{
        //Other different CODE
}
```

The else if and else statements are optional. Here are some example conditional statments:

```
if(k < 10) // strict inequality
if(k <= 10) // soft inequality
if(k != 10) // ! is not
if(k == 10) // use == for conditionals as = is for assignment
```

The while loop is like a combitation of for and if. In this case, we loop over a conditional. For example:

```
while(t<tmax)
{
        //CODE to execute
        t += dt;
}

while(t != tio)
{
        //CODE
        t += dt;
}
```

These types of while loops are common in simulation codes.

4 Functions

Like most programming languages C supports functions. A key difference from Fortran, is that there is no SUBROUTINE. So any functional type is used as a function in C. When writing a function you must declare it using the same data type as the output you wish to recieve. For example:

```
int myfunction(const int a, const int b)
{
        int c;
        c = a + b;
        return c;
}
```

Here the const delimiter tells the compiler that the data stored in this variable will not change and is only used for another variable. This works with all basic data types. Suppose we want the output to be an array. Then we use the void data type and pass the output as a parameter.

```
void myfunction( const int a[10], const int b[10], int c[10])
{
        for(int i = 0; i < 10; i++)
                c[i] = a[i] + b[i];
}
```

5 IO

Suppose we wish to write out the result of a function or an array into a file for later use. Or perhaps we want to read in data from a file. We can use the `fopen()` function to create a new file or to open an existing file in the directory that we are working in. The prototype of this function call is as follows —

```
FILE *fopen( const char * filename, const char * mode);
```

Here, filename is a string literal, which you will use to name your file, and access mode can have various values:

- r, opens an existing text file for reading
- w, opens a text file for writing, if it does not exist then a new file is created. Your program will write at the beginning of the file.
- r+, combines the functionality of r and w.
- w+, much like r+ only it truncates the file to 0 length if it exists.

There are other modes but these are the most important for this class. If you wish to use binary files then use b after each standard letter, e.g. rb, and rb+.

To close a file, use the `fclose()` function. The prototype for this function is

```
int fclose (FILE *fp);
```

make sure to include the `<stdio.h>` header when performing IO. To read values in the file, we use the function `fscanf()`. To print to another file use `fprintf()`. For example suppose we are reading values from a file containing scores from an online test (file = ifp) and outputing them to another file (ofp), then the code would be:

```
FILE *ifp, *ofp;
char *mode = "r";

ifp = fopen("score.txt", mode);

ofp = fopen("output.txt", "w");
while(fscaf(ifp, "%s %d", username, &score) != EOF)
        {
                fprintf(ofp, "%s %d", username, score+10);
        }
```

EOF is a parameter defined in stdio.h and represents the end of the file. The specifiers are called by useing the % sign. Here's a representative sample of them:

- c — a single character,
- d — a decimal integer: Number optionally preceded by a + or -
- e, E, f, g, G, Floating point: Decimal number containing a decimal point, optionally preceded by a + or - sign and optionally followed by the e or E character and a decimal number.
- s, a string of characters.

So, in the above sample code, we were reading a string and an integer.

6 An example program

We are going to use all the above information to creat a single precision ax + y, where x and y are arrays that are dynamically allocated. We will read x and y from a file and then output the answer to another file.

```
#include <stdio.h>
#include <stdlib.h> //header contains malloc and free
#define N 1024 // This will be the length of our vectors

// in this function we are passing dynamically allocated pointers
void saxpy(const float *x, const float *y, const float a, float *z)
```

```
{
        for (int i = 0; i < N; i++)
                z[i] = a*x[i] + b[i];

}

//main function
int main()
{
        float* x;
        float* y;
        float* z; //pointers to the float
        float a = 2.0f; //use f to denote single precision, sometimes compilers default to double.

        size_t size = N*sizeof(float); //size_t is an unsigned integer of at least 16 bits used for
            dynamic alloc
        x = (float*)malloc(size);
        y = (float*)malloc(size);
        z = (float*)malloc(size);
        //malloc is the dynamic allocator function

        FILE *if1, *if2, *of; //files
        if1 = fopen("xdat.txt", "r");
        if2 = fopen("ydat.txt", "r");
        of =  fopen("zdat.txt", "w");

        for(int i = 0; i < N; i++) //Read in data!
        {
                fscanf(if1, "%f",&x[i]);
                fscanf(if2, "%f",&y[i]);
        }
        //close the input files
        fclose(if1);
        fclose(if2);

        saxpy(x, y, a, z); //Call the saxpy function!

        for(int i = 0; i < N; i++)// write data!
        {
                fprintf(of, "%f", z[i]);
                fprintf ( of, "%s","\n"); // to make a new line!
        }

        //close the output file
        fclose(of);

        //deallocate dynamic data
        free(x);
        free(y);
        free(z);
}
```

For a more detailed tutorial into C programming visit this website.