3 Functions, pointers and structures

3.1 Functions

The entry point into all our programs is called main() and this is a **function**, or a **piece of code** that does something, usually returning some value. We structure programs into functions to stop them become long unreadable blocks of code than cannot be seen in one screen or page and also to ensure that we do not have repeated identical chunks of code all over the place. We can call **library functions** like printf or strtok which are part of the C language and we can call our own or other peoples functions and libraries of functions. We have to ensure that the appropriate header file exists and can be read by the preprocessor and that the source code or compiled library exists too and is accessible.

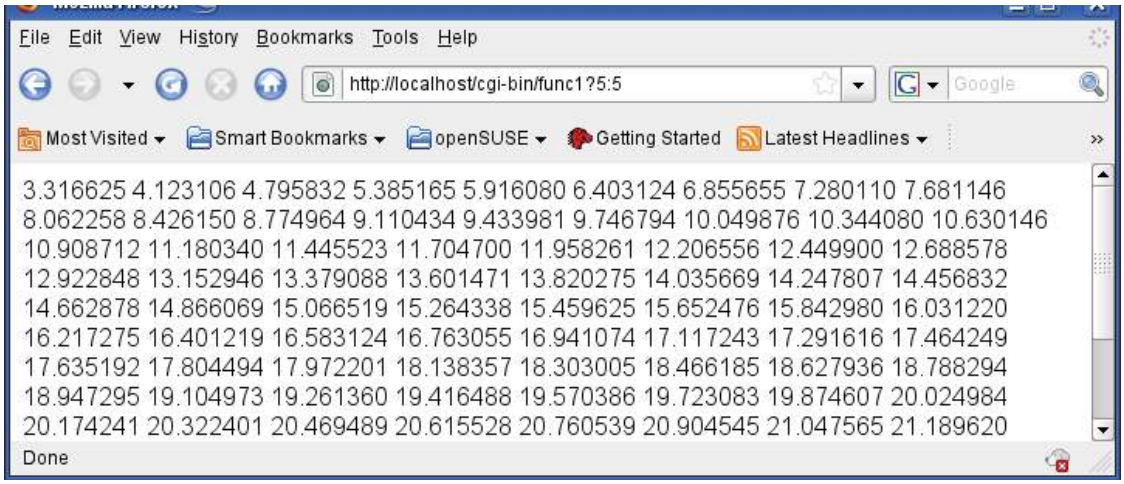
As we learned before, the **scope** of data is restricted to the **function** in which is was declared, so we use **pointers** to data and blocks of data to pass to functions that we wish to do some work on our data. We have seen already that strings are handled as pointers to arrays of single characters terminated with a NULL character.

```
/*****************************************************************
* C Programming in Linux (c) David Haskins 2008
* chapter3_1.c                                  *
*****************************************************************/
#include <stdio.h>
#include <string.h>
#include <math.h>

double doit(int number1, int number2)
{
        return sqrt((double)(number1 + number2) );
}

int main(int argc, char *argv[], char *env[])
{
        int n1 = 0, n2 = 0, i=0;
        n1 = atoi((char *) strtok(argv[1],":"));
        n2 = atoi((char *) strtok(NULL,":"));
        printf("Content-type:text/html\n\n<html><body>\n");
        for(i=1;i<=100;i++)     printf("%f ",doit(n1+i,n2*i));
        printf("</body></html>\n");
        return 0;
}
```

In this example we can repeatedly call the function “doit” that takes two integer arguments and reurns the result of some mathematical calculation.

Compile: gcc -o func1 chapter3_1.c -lm
Copy to cgi-bin: cp func1 /home/david/public_html/cgi-bin/func1

(You should be using the Makefile supplied or be maintaining a Makefile as you progress, adding targets to compile examples as you go.)



The result in a browser looks like this called with “func1?5:5”.

In this case the arguments to our function are sent as **copies** and are not modified in the function but used.

If we want to actual modify a variable we would have to send its pointer to a function.

```
/*****************************************************************
* C Programming in Linux (c) David Haskins 2008
* chapter3_2.c                                  *
*****************************************************************/
#include <stdio.h>
#include <string.h>
#include <math.h>
void doit(int number1, int number2, double *result)
{
        *result =  sqrt((double)(number1 + number2) );
}
int main(int argc, char *argv[], char *env[])
{
        int n1 = 0, n2 = 0, i=0;
        double result = 0;
        n1 = atoi((char *) strtok(argv[1],":"));
        n2 = atoi((char *) strtok(NULL,":"));
        printf("Content-type:text/html\n\n<html><body>\n");
        for(i=1;i<=100;i++)
        {
                doit(n1+i,n2*i,&result);
                printf("%f ", result);
        }
        printf("</body></html>\n");
   return 0;
}
```

We send the address of the variable ‘result’ with **&result**, and in the function doit we ***de-reference*** the pointer with ***result** to get at the float and change its value, outside its scope inside main. This gives identical output to chapter3_1.c.

3.2 Library Functions

C contains a number of built-in functions for doing commonly used tasks. So far we have used **atoi**, **printf**, **sizeof**, **strtok**, and **sqrt.** To get full details of any built-in library function all we have to do is type for example:

man 3 atoi

and we will see all this:

Which pretty-well tells you everything you need to know about this function and how to use it and variants of it. Most importantly it tells you which **header file** to include.

3.3 A short library function reference

Full details of all the functions available can be found at:
http://www.gnu.org/software/libc/manual/

```
Common Library Functions by Header File:

        math.h
                Trigonometric Functions e.g.:
                        cos sin tan
                Exponential, Logarithmic, and Power Functions e.g.:
                        exp log pow sqrt
                Other Math Functions e.g.:
                        ceil fabs floor  fmod

        stdio.h
                File Functions e.g.:
                        fclose  feof  fgetpos  fopen fread fseek
                Formatted I/O Functions e.g.:
                        printf scanf Functions
                Character I/O Functions e.g.:
                        fgetc fgets  fputc fputs getc getchar gets
                        putc putchar puts

        stdlib.h
                String Functions e.g.:
                        atof atoi atol
                Memory Functions e.g.:
                        calloc free malloc
                Environment Functions e.g.:
                        abort exit  getenv system
                Math Functions  e.g.:
                        abs div rand

        string.h
                String Functions e.g.:
                        strcat strchr strcmp strncmp  strcpy
                        strncpy strcspn strlen strstr strtok

        time.h
                Time Functions e.g.:
                        asctime clock difftime time
```

There is no point in learning about library functions until you find you need to do something which then leads you to look for a function or a library of functions that has been written for this purpose. You will need to understand the function signature – or what the argument list means and how to use it and what will be returned by the function or done to variables passed as pointers to functions.

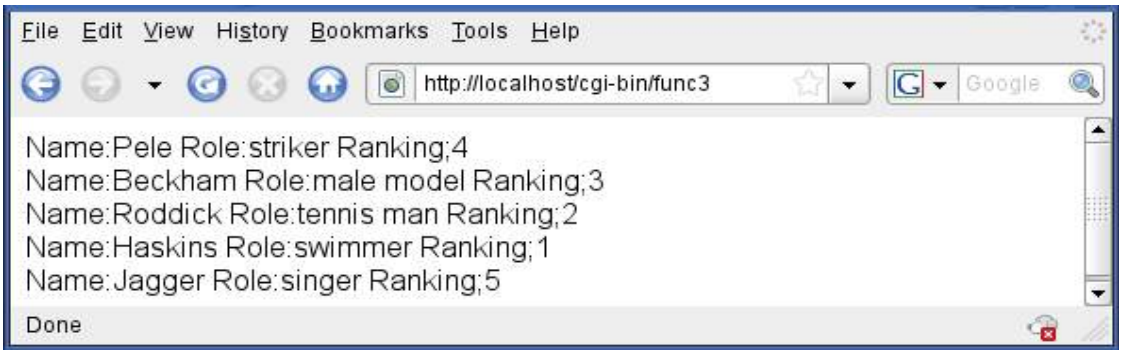
3.4 Data Structures

Sometimes we wish to manage a set of variable as a group, perhaps taking all the values from a database record and passing the whole record around our program to process it. To do this we can group data into structures.

This program uses a struct to define a set of properties for something called a player. The main function contains a declaration and instantiation of an array of 5 players. We pass a pointer to each array member in turn to a function to rank each one. This uses a switch statement to examine the first letter of each player name to make an arbitrary ranking. Then we pass a pointer to each array member in turn to a function that prints out the details.

```
/*****************************************************************
* C Programming in Linux (c) David Haskins 2008
* chapter3_3.c                                  *
*****************************************************************/
#include <stdio.h>
#include <string.h>
struct player
{
        char name[255];
        char role[255];
        int ranking;
};
void rank(struct player *p)
{
    switch(p->name[0])
    {
        case 'P': p->ranking = 4;break;
        case 'H': p->ranking = 1;break;
        case 'R': p->ranking = 2;break;
        case 'J': p->ranking = 5;break;
        case 'B': p->ranking = 3;break;
    }
}
void show(struct player p)
{
    printf("Name:%s Role:%s Ranking;%d<br>\n",
        p.name,p.role,p.ranking);
}
int main(int argc, char *argv[], char *env[])
{
    struct player myteam[5] = { { "Pele","striker",0 },

                { "Beckham","male model",0 },

                { "Roddick","tennis man",0 },

                { "Haskins","swimmer",0 },

                { "Jagger","singer",0 } };
    int i=0;
    printf("Content-type:text/html\n\n");
    for(i=0;i<5;i++)  rank ( &myteam[i] );
    for(i=0;i<5;i++) show  ( myteam[i] );
    return 0;
}
```

The results are shown here, as usual in a browser:



This is a very powerful technique that is quite advanced but you will need to be aware of it. The idea of **structure**s leads directly to the idea of **classes** and **objects**.

We can see that using a struct greatly simplifies the business task of passing the data elements around the program to have different work done. If we make a change to the definition of the struct it will still work and we simply have to add code to handle new properties rather than having to change the argument lists or signatures of the functions doing the work.

The definition of the structure does not actually create any data, but just sets out the formal shape of what we can instantiate. In the main function we can express this instantiation in the form shown creating a list of sequences of data elements that conform to the definition we have made.

You can probably see that a struct with additional functions or methods is essentially what a class is in Java, and this is also the case in C++. **Object Oriented languages** start here and in fact many early systems described as "object oriented" were in fact just built using C language structs.

If you take a look for example, at the Apache server development header files you will see a lot of structs for example in this fragment of httpd.h:

```
struct server_addr_rec {
    /** The next server in the list */
    server_addr_rec *next;
    /** The bound address, for this server */
    apr_sockaddr_t *host_addr;
    /** The bound port, for this server */
    apr_port_t host_port;
    /** The name given in "<VirtualHost>" */
    char *virthost;
};
```

Dont worry about what this all means – just notice that this is a very common and very powerful technique, and the design of data structures, just like the design of database tables to which it is closely related are the core, key, vital task for you to understand as a programmer.

You make the philosophical decisions that the **world is like this** and can be modelled in this way. A heavy responsibility – in philosophy this work is called **ontology** (what exists?) and **epistemology** (how we can know about it?). I bet you never thought that this was what you were doing!

3.5 Functions, pointers and structures – conclusion

We have used some simple data types to represent some information and transmit input to a program and to organise and display some visual output.

We have used HTML embedded in output strings to make output visible in a web browser.

We have learned about creating libraries of functions for reuse.

We have learning about data structures and the use of pointers to pass them around a program.

Exercise:

Using C library functions create a program that:

- opens a file in write mode,
- writes a command line argument to the file
- closes the file
- opens the file in read mode
- reads the contents
- closes the file
- prints this to the screen

This will give you experience with finding things out, looking for suitable library functions, and finding examples on-line or from a book.