Introduction to C

Session 4

Functions in C

- We have seen earlier how we can create a function.
- Means our programs in C can become ‘modular’

```
<return type> FunctionIdentifier (arg1, arg2, …){
       **function body**
}
```

```c
int addTwoNumbers(int x, int y){
    return x+y;
}
```

```c
void addTwoNumbers(int x, int y){
    printf("%d", x+y);
}
```

Functions in C

- Need to be careful of how we define and use the function though
- Either:
  - Define the function before using it (i.e. before the ‘`main`’ function)
    - If not, compiler infers argument and return types – can get unpredictable behaviour
  - Prototype the function before it is used
    - Can define it anywhere then
  - Prototype it or define in completely in a header file
    - Remember to include the header file

Functions in C

1. Define a function before using it

Defining the function after ‘main’, can result in unpredictable behaviour.

```c
int add(int p, int q)
{
    return p+q;
}

int main(int argc, char **argv)
{
    printf("%d", add(2, 3));
    return 0;
}
```

The function ‘add’ is defined before it is used in ‘main’

Functions in C

2. Prototype the function before it is used

The prototype defines the function name, return type and argument datatypes.

```
//function prototype
int add(int, int);

int main(int argc, char **argv)
{
    printf("%d", add(2, 3));
    return 0;
}
int add(int p, int q)
{
    return p+q;
}
```

The function ‘add’ is prototyped before it is used in ‘main’. It is defined after.

Functions in C

3. Prototype it or define in completely in a header file

test.h

```
//Function Prototypes
int add(int, int);
```

test.c

```
#include "test.h"
int main(int argc, char **argv)
{
    printf("%d", add(2, 3));
    return 0;
}
int add(int p, int q)
{
    return p+q;
}
```

Must #include the .h file where the prototype is

Functions in C

3. Prototype it or define in completely in a header file

test.h

```
int add(int p, int q)
{
    return p+q;
}
```

We can define the whole function here too

test.c

```
#include "test.h"
int main(int argc, char **argv)
{
    printf("%d", add(2, 3));
    return 0;
}
```

Must #include the .h file where the prototype is

Functions in C

$5^2 = 25$

$1+3+5+7+9 = 25$

- Lets try an example:
  - The square of a positive number, *n*, is equal to the sum of *n* odd integers starting with 1… (Yes, that's right! Try it …)
  - Write a function which demonstrates this…

Try for yourself ….

Functions in C

- There are many ways to do this – this is just one:

<u>Pseudocode:</u>

```
loop=0
sum=0
Get the number

Add one to sum

Increment loop by one

If loop = number
        Stop (return sum)
Else add two to loop
```

```c
int altSquare(int n)
{
    int sum = 0;
    int num = 0;
    int i = 1;
    while(true)
    {
            sum+=i;
            num++;
                if(num == n)
                break;
            i+=2;
    }
    return sum;
}
```

Functions in C

- We could have the function ‘altSquare’ in a header file (test.h)
  - Keep the main program file neat, make the program as modular as we can
  - Make sure we include the ‘test.h’ file in our main file.

These are ‘conditional defines’. Good practice, means a header file cannot be included twice and get multiple definition errors etc.

```
#ifndef TEST_H_
#define TEST_H_

int altSquare(int n)
{
    int sum = 0;

    int num = 0;

    int i = 1;

    while(true)
    {

            sum+=i;

            num++;

                if(num == n)

                break;

            i+=2;
    }

    return sum;
}

#endif
```

Input/Output (I/O) in C

- Sometimes hear of
  - ‘input streams’
  - ‘output streams’

These are ‘abstractions’ – they are not really ‘streams’, but interfaces that C provides different methods to interact with

Input/Output (I/O) in C

- For example, getting data from the keyboard – user typed input
  - This is a ‘process’ provided by the OS and provided to the program
    - We, as programmers, don’t have to set it up
- This process is called the ‘Standard Input Stream’ – ‘`stdin`’
- In C, we can scan the `stdin` by using the function ‘`scanf`’ or ‘`fgets`’ for example
  - `scanf/fgets` know where `stdin` is automatically

Other streams are ‘`stdout`’ and ‘`stderr`’

Input/Output (I/O) in C

- C provides functions which allow you to ‘plug in’ or ‘interface’ with these streams

`printf()`	Write data to stdout
`putchar()`	Writes one character to stdout
`putc()`	Similar to putchar(), specify stream
`puts()`	Writes a string of characters to stdout
fputs()	Writes string of characters to file
…etc…	

`scanf()`	Reads data from stdin
`gets()`	Reads string from stdin
`fgets()`	Similar to gets(), specify stream
…etc…	

Input/Output (I/O) in C

- Lets go back to our ‘altSquare’ function
- This time, get input from the user for the number to be squared.

Number up to 9 digits long, plus the \n character

```
char buffer[10];
int val;

printf("Please enter a number\n");

fgets(buffer, sizeof(buffer), stdin);
val = atoi(buffer);

printf("%d", altSquare(val));
```

Read the input into the buffer, from the stdin stream

Output the contents of the buffer

Input/Output (I/O) in C

- Lets go back to our ‘altSquare’ function
- This time, get input from the user for the number to be squared.

```c
#include <stdio.h>
#include <stdlib.h>
#include "test.h"
int main(int argc, char **argv)
{

    int val;

    printf("Please enter a number\n");

    scanf("%d", &val);

    printf("The number entered is: %d", altSquare(val));
    return 0;
}
```

This time, we use ‘scanf’
The ‘&’ means the user entry is written to ‘val’

Input/Output (I/O) in C

- We can format the data outputted to stdout (some examples – there are more..)

Data type		
d, i	Integer	printf(“%d”, 5); /* outputs 5 */
c	Character	printf(“%c”, q); /* outputs q */
s	String	printf(“%s”, “test”); /* outputs test */
f	Float	printf(“%f”, 1.23); /* outputs 1.23 */
d	Double	printf(“%f”, 1.23); (float is promoted to a double anyway)

Input/Output (I/O) in C

- More format of the data outputted to stdout

f	float or double	printf((“%.3f”, 1.23456); /*1.234 */
s	string	printf(“%5s”,”Hi”); /*prints 3 spaces then the two chars of ‘Hi’*/
s	string	printf(“%-5s”, “Hi”); /*Prints two chars of ‘Hi’, then 3 spaces */
s	string	printf(“%.2s”, “Test”); /*prints ‘Te’ */

Means 3 places after the dp will be printed.

File I/O

- Reading/writing to a file is similar to reading and writing from/to stdin/stdout
- Except!
  - We need to tell which ‘stream’ (file) to read from/to

```
FILE *f;
char *str[100];
f =  fopen("test.txt", "r");
while(fscanf(f, "%s", &str)==!NULL){
    printf("%s ", str);
}
fclose(f);
```

Create a file stream pointer

Open file for read and assign to file pointer

Read a string, assign to str

Print str to stdout

Close the file (can happen automatically when program closes)

More File I/O

int getc(FILE f)

```
f =  fopen("test.txt", "r");
printf("%c", getc(f));
fclose(f);
```

*Reads a character from the stream and prints it to stdout

char[] fgets(char str[], int len, stream)

```
char str[100];
fgets(str, 50, stdin);
printf("%s", str);
```

*Can get a string of characters (up to ‘len’ characters) from the stream (can be a file, or a standard stream)

More File I/O

int putc(int c, FILE *f)

```
FILE *f;
f = fopen("test.txt", "a");
printf("%c", putc('q', f));
```

*Appends the char ‘q’ to the end of the stream ‘f’ - i.e. the end of the file `test.txt`

char[] fputs(char str[], FILE)

```
FILE *f;
f = fopen("test.txt", "a");
char str[] = "hello";

if(fputs(str, f)==0){
    printf("Success");
}else{
    printf("Oh Dear");
}
```

`fputs` returns ‘0’ if it is successful

In conclusion …

- In this session, we have covered:
  - Functions
    - Prototyping
    - Header files
  - Input / Output
    - Streams
    - Writing to, and getting input from streams
    - File input/output