The University Of Sheffield.

Introduction to C Programming

The University Of Sheffield.

Who we are

Research & Innovation Support in IT Services

Dr Michael Griffiths

Dr Norbert Gyenge

The University Of Sheffield.

Course Outline

1. Introduction to C and Structured Programming
2. Using functions to build programs, managing data and computer memory
3. Data Structures, Strings and File Input/Output

The University Of Sheffield.

Course Format

- Course slides
- Basic C-programming examples - highlighting techniques and syntax
- Demonstrate techniques with practical examples
- Frequent breaks for practice sessions

The University Of Sheffield.

Course pre-requisites

- Course material
  - `git clone --branch introtoc2022v1 https://github.com/rcgsheffield/introc`
  - `http://rcg.group.shef.ac.uk/courses/introc`
- A C-Compiler or IDE
  - Codeblocks - https://www.codeblocks.org/downloads/
  - MSys2 - https://www.msys2.org/
  - Eclipse IDE (more advanced) - https://www.eclipse.org/downloads/packages/release/2022-06/r/eclipse-ide-cc-developers

The University Of Sheffield.

Resources

- Cplusplus
  - https://cplusplus.com/
- W3 Schools
  - https://www.w3schools.com/c/

The University Of Sheffield.

Learning Outcomes

- Layout and Syntax of a program
- Compiling and Running a program
- Programming Structures
- Data Types and Variables

The University Of Sheffield.

Program Development Steps

- Edit
  - Create program and store on system
- Preprocessor
  - Manipulate code prior to compilation
- Compiler
  - Create object code and store on system
- Linker
  - Link object code with libraries, create executable output
- Loader
- Execution
  - CPU executes each instruction

The University Of Sheffield.

Program Structure

- Collection of Text files
  - Source files
  - header files
- Resource files
- Source file layout
  - Function layout
- Program starts with a function called main
- Pre-processor directives

The University Of Sheffield.

Layout and Syntax of a C Program:Hello World

```
/* This is a hello world program*/

#include <stdio.h>    /*pre-processor statement*/

/*The program starts with a main function*/
/*This program will return an integer result*/
int main()
{

        /*Blocks of program statements are enclosed by pairs of
         * curly braces at the beginning and end of the block*/

        /*Use the printf function to Display a welcome message on the users screen*/
        printf("Welcome to the C Language!\n);"


   /*The program finishes when the final statement in the main program block
    * is executed or a return statement is reached*/

        /*Return the integer result 0 as the output of the program*/
        return(0);
}
```

The University Of Sheffield.

Features of a simple C-Program

- A C-Program is just a text file you can edit with most text editor
- Lots of comments - Enclosed by /* */
- Program blocks enclosed by curly braces {}
- Statements terminated with a ;
- Preprocessor statement
  - #include <stdio.h>
  - Enables functions to call standard input ouput functions (e.g. printf, scanf)
  - Not terminated with a ;
- printf is a function call from the standard C-library which uses escape sequence characters e.g. \n newline

The University Of Sheffield.

Compiling the Program

gcc -o runcode [options] mycode.c

- runcode is the executable program
- mycode.c is the source code
- Options, for example compile for debugging, set optimisation flags, definitions etc..

The University Of Sheffield.

Compiler Options

Option	Action
-s	Remove any symbol and object relocation information from the program. Reduce the size of the program and runtime overhead
-c	Compile, do not link
-o exefile	Specify name for the resulting executable
-g	Produce debugging information (no optimisation)
-llibrary_name (lower case L)	Link the given library into the program e.g. include math library by using the option -lm
-Idirectory_name (upper case I)	Add directory to search path for include files
-O N	Set optimisation level to N
-Dmacro[=defn]	Define a macro

The University Of Sheffield.

Programming Structures

- Understand program structures by reviewing a practical example
- Review the root_bisection.c example understand how this follows the saem structure
- Develop understanding of root_bisection.c

```c
/*Bisection method for finding roots*/

/* here is an example use of the while statement
// which is used for finding the root of a polynomial
// which is known to lie within a certain interval.
// a is the lower value of the range
// b is the upper value of the range */
#include <stdio.h>
#include <math.h>
#include <float.h>

/*
  Note FLT_MIN, FLT_MAX and FLT_EPSILON
  defined in float.h
*/

float sign(float f){return(fabs(f)/f);}

int main(char **argv, int argc)
{
/*Main routines here see next slide*/
        return 0;
}
```

The University Of Sheffield.

Practical demonstration

- Overview of the course examples
- Compiling and running the welcome.c example in the basics folder
- Inspecting the root_bisection.c example in the exampleprograms folder

The University Of Sheffield.

Practice Session

1. Access the course examples from http://rcg.group.shef.ac.uk/courses/cic6006/
2. Compile and run the hello world program add another message which is displayed to the screen for example “Good Bye and Thank you”
3. Inspect the program in the cexamples folder “root_bisection.c” familiarise yourself with the general structure of the program
4. Compile and run root_bisection.c

The University Of Sheffield.

Program Features

- Values and variable types
- Arithmetic operations
- Control Structures
  - Conditional selection and branching
  - Repetition of operations
- Functions

The University Of Sheffield.

Class Question:

What is the syntax used for a comment in a C program, which option/options is/are correct?

A. Comment lines start with the # symbol
B. Comment lines start with the characters /*
C. Comment lines start with the characters //
D. Comment lines start with the characters rem
E. Comments are enclosed by the characters /* and */

The University Of Sheffield.

Values and Variables

A variable is a container for a value we can have different types

– Characters
– Integer Values
– Floating-point values
– Memory locations

The University Of Sheffield.

Variables

Variables of the same type are compared using the comparison operator ==

- Variable declaration using the assignment operator =

```
float myfloat;
float fanother=3.1415927;
```

- Other types using unsigned and long
  - long double, long int, short int, unsigned short int
- Occupy memory have a size

Precision and range machine dependent

The University Of Sheffield.

Variable Types

Type	Size (bytes)	Range
int	4	-2^{31} to $+2^{31}$
float	4	-3.2×10^{32} to $+3.2 \times 10^{32}$
double	8	-1.7×10^{302} to $+1.7 \times 10^{302}$
char	1	
short int	2	-32768 to 32767
unsigned short int	2	0 to 65536
unsigned char	1	0 to 255

The University Of Sheffield.

Operators

- Arithmetic operations

=, -, /, %, *

- Assignment operations

=, +=. -=, *=, %=, /=, !

- Increment and decrement (pre or post) operations

++, --

- Logical operations

||, &&, !

- Bitwise operations

|, &, ~

- Comparison

<, <=, >, >=, ==, !=

The University Of Sheffield.

Practical Demonstration

Locate and inspect the variable definition lines

- Inspect the program root_bisection.c
- Which lines define a variable?
- Which lines initialise a variable?
- Which lines are initialised using a constant header file?
- Inspect the header file in the cplusplus web site

```
int main(char **argv, int argc)
{

        float x,fx;
        float a = 0;
        float fa = -FLT_MIN;
        float b = 3;
        float fb = FLT_MAX;
```

The University Of Sheffield.

Practice Session

1. Inspect compile and run arith.c
2. After the display results section add two new lines as follows
   a. Use the ++ operator to increment the sum variable
   b. Use the -- operator to decrement the difference variable
3. Use the printf command to display new values for sum and difference
4. Compile, run and debug your modified arith.c program

```
  while( fabs(b-a)>(FLT_EPSILON*b))
  {
    x = (a+b)/2;
    /*The function whose root is to be determined*/
    fx = pow(x,3)-2*x-5;
    if(sign(fx)==sign(fa))
    {
      a = x;
      fa = fx;
printf("a=%f fa=%f\n",a,fa);
    }
    else
    {
      b = x;
      fb = fx;
printf("b=%f fb=%f\n",b,fb);
    }

  }
  printf(" The root is :%f\n",x);
```

The University Of Sheffield.

Conditional Statements

The if statement allows decision making functionality to be added to applications.

- General form of the if statement is:

```
if(condition)
        statement;
```

The University Of Sheffield.

Comparison Operators

Compare values using conditional operators.

- == equal to
- > greater than
- < less than
- >= greater than or equal to
- <= less than or equal to

The University Of Sheffield.

else

An alternative form of the if statement is

```
if(condition)
        statement;
else
        statement;
```

If the condition is true the first statement is executed if it is false the second statement is executed.

The University Of Sheffield.

Demonstration

- Build and run the example if1.c and if2.c
- Note the condition operator in round brackets
- Use of {} when multiple statements are used
- Test the program with different conditions
- Build and run if3.c
  - What is wrong with this code?
  - Was there are compiler warning?
- Build and run the programs ifelse.c and ifelseifelse.c

The University Of Sheffield.

Multiple selection Structures Using Switch

- Used for testing variable separately and selecting a different action

```
switch(file)
{
        case 'm': case 'M':
        ++nMaxima;
        break;
        case 't': case 'T':
        ++nTitania;
        break;
        default: /*Catch all other characters*/
        ++nOther;
        Break;
} /*End of file check switch */
```

The University Of Sheffield.

Repetition Using While

Execute commands until the conditions enclosed by the while statement return false.

```
while(conditions)
{
    Statements;
}
```

Good practice to always use {} in a do while loop

The University Of Sheffield.

do while

Good practice to always use {} in a do while loop

```
do
{
      Statements…;
      Statements…;
}
while(conditions)
```

The University Of Sheffield.

Demonstration

- Build and run the example while1.c
- Modify the loop so that it counts to 20
- Modify the loop so that it counts to 20 in steps of 2
- Build and run dowhile.c

The University Of Sheffield.

Class Question

If a is initially equal to ten, Which while loop is finite?

A. while(a>=10){a=10};
B. while(a<10){a=a+10};
C. while(a<10){b+=10};
D. while(a<=10){a-=10};

The University Of Sheffield.

Counter controlled repetition

- Components of a typical for loop structure

```
for(expression1; expression2; expression3)
    statement;
```

example

```
for(counter=1; counter<=10, counter++)
    statement;
```

The University Of Sheffield.

Demonstration

- Build and run the example for1.c
- Build and run nestedfor1.c

The University Of Sheffield.

Practice Session

Write a program that uses a for loop to display numbers for three different cases

a. Display the values from 1 to 20

b. Display the values from 2 to 20

c. Display the values from 10 to 1

(Hint: Use the example for1.c)

The University Of Sheffield.

Practical Examples

- Inspect, Compile and run the following
- Finding a root by method of bisection - does this now make sense?
  - -If statement, while statement
  - -And simple one line function!
- Finding a root using the Newton-Raphson method
  - -While statement

The University Of Sheffield.

Further Sessions

- Building Applications using Make
- From C to C++
- Boost your programming using the standard template libraries