Overview

- This lecture will:
  - justify the use of C for teaching graphics;
  - introduce the C language (syntax);
  - show how to write simple C programs;
  - show how to compile C programs.
- This will be achieved using practical examples.

C, OpenGL and Computer Graphics

- C is the most commonly used low level graphics programming language.
- OpenGL is the most commonly used API.
- So in this module we will combine both.
- C is much like Java in syntax (almost identical).
- C structure is very different:
  - Java is OO – everything is a class;
  - C is procedural – everything is a function (Ada).
- C always has a main function which controls program executing.
- C is compiled rather than interpreted.

A typical C program

- The `hello.c` program is shown below:

```
/* C program: the hello world favourite, for manic depressives.
   20/12/01  (c) Dan Cornford 2001         */

/* We need to include standard IO library (printf) */
#include <stdio.h>

/* Main function. */
int main(void)
{
  /* Call the printf function passing in the string Goodbye
     world, with the \n to force a line feed */
  printf("Goodbye world!\n");

  return 0; /* ANSI C requires main to return an int. */
}
```

Variables and constant

- Variable names are made up of letters, numbers and the underscore character.
- The first character must be a letter. The number of significant characters is reasonably large, but not infinite.
- All variables should be declared (at the top of the functions) before use.

```
type variable = initial_value; /* Description */

int course_number = 215; /* Code of the course */
```

- Use symbolic constants for all but the most obvious values in a program. These are best defined at the head of the file (outside the function definitions) as in the following example:

```
#define GOLD 1.618034
```

Types

- There are four basic types in C:
  - `char` for single characters,
  - `int` for integers,
  - `float` for floating point numbers and
  - `double` for double precision floating point numbers
- Modifiers: `signed`, `unsigned`, `short` and `long`.
- Can declare our own types:

```
typedef type type_name;

typedef float Real;
```

- Created to describe the logical rather than physical type.

More on Types

- Enumerated types can be useful for storing fixed non-numeric data. The syntax is

```
enum tag {enum_list} var_list;
```

- The machine type of an enumerated type is an `int`. Example:

```
enum VarType {discrete, ordinal, continuous};
```

- Can have constant variables! (Compile time type checking)

```
const type variable = value;
```

So we can replace

```
#define GOLD 1.618034
```

by

```
const float GOLD = 1.618034;
```

Operators

- C has all the usual arithmetic operators: `+`, `-`, `*`, `/` and `%`.
- Comparison operators are `>`, `>=`, `<`, `<=`, `==`, and `!=`.
- We will not use bitwise operators, but these are one of the more powerful features of C.
- Assignment operator is `=`. Can combine with arithmetic operators:

```
i += 2;
```

is equivalent to

```
i = i + 2;
```

- Increment and decrement operators `++`, `--`.

```
i++;
```

Statements

- A C program consists of a sequence of statements.
- There are 10 different statements in C.
- Any sequence of statements surrounded by curly braces { and } is treated as a single (compound) statement.
- An expression can be made into a statement by adding a semi-colon at the end.
- Expression statements are usually one of the assignment expressions or a function call.

```
x = 3;
i += 4;
```

Conditional Statements

- Conditional statements in C use the `if` construct.
- The general form is:

```
if (expression)
    statement 1
else
    statement 2
```

- Be careful about compound statements in branches:

```
if (a > b)     |    if (a > b) {
    z = b;     |        foo = bar;
else           |        z = a;
    z = a;     |    }
               |    else
               |        z = b;
```

Switch Statement

- If we want to choose between a number of different conditions then a switch statement is more appropriate than lots of ifs.

```
VarType var;
....
switch (var) {
    case ordinal:
      /* Do ordinal type things */
    case discrete:
      /* Do things for ordinal AND discrete types */
      break;    /* Don't fall through to next case */
    case continuous:
      /* Do continuous type things */
      break;
    default: /* Treats all other cases */
      fprintf(stderr,"Unknown value in switch statement\n");
      break;
}
```

- Don't forget to specify a default behaviour – often used to process keyboard input in graphics.

Loops

- Two main types in C:

```
while (e)         |    for (e1; e2; e3)
  s;              |      statement;
```

- Example:

```
char c, s[];
int i, j;
for (i = 0, j = strlen(s) - 1; i < j; i++, j--) {
    c = s[i];
    s[i] = s[j];
    s[j] = c;
}
```

- Do while statement

```
int num;
do {
    scanf("%d", &num);
} while (num >= 0);
```

Function Syntax

- Functions are the building blocks of C programs.
- The general form is:

```
return_type function_name(arguments)
{
    declarations
    statement
}
```

- Arguments should be declared together with their type in a comma separated list.

```
int foo(int bar, double baz)
{
    /* function body */
}
```

- If a function returns no value, it has return value `void`.

Function Arguments and Local Variables

- Function arguments should be used to communicate values rather than have global variables.

```
int x = 6;
int y = 0;
y = foo(x);
/* x still has the value 6 */
....
int foo(int x)
{
    x *= 2;
    return x + 2;
}
```

Function Arguments and Local Variables

- Any variables declared in the function body are local to the function.

```
int bar = 0;
int baz = 0;
baz = foo(bar);
...
int foo(int x)
{
    int bar = 6;
    return x * bar;
}
```

- To modify and return multiple values we need to use pointers.

More on Functions

- When putting multiple functions in one file (as we shall do) they should either:
  - be in the order they are called, above the `main` program;
  - or have function headers (prototypes) at the top of the code.
- In most code you write you will need to use some standard C libraries:
  - `#include <stdio.h>`,
  - `#include <math.h>`,

  are the most common ones.
- We will come back to functions – also very important when using OpenGL.

General form of a C program:

```
/* Please don't forget to comment me */

#include <libraries>
#define CONSTANT 215
typedef double Real;
global variables

Real myfunc(int arg1, double arg2);
void myotherfunc(int arg);

int main(void)
{
  local variables
  statements;
  return 0;
}

Real myfunc(int arg1, double arg2)
{
  Real value;
  statements;
  return value;
}
```

```
void myotherfunc(int arg)
{
  statements;
  return;
}
```

Compilation

- We will use Solaris and emacs for the coding in the labs, with the `gcc` compiler.
- To compile code:

  `gcc -c program.c`
- To link code:

  `gcc -o program program.o`
- This gets a bit boring so later we will write a `Makefile` for the code – covered in the labs.
- Of course everything also works with MS Visual Studio, so you can also use this readily.

Summary

- Having finished this lecture you should:
  - understand why C was chosen for the course;
  - be able to use: variables and constants, types, operators, statements, conditional statements, switch statements, loops and functions in C;
  - be able to write simple C programs;
  - be able to compile C programs.