Object Oriented Methodology

Week – *0*, Lecture – *1*

Revision of C

SAURABH SRIVASTAVA

VISITING FACULTY

IIIT LUCKNOW

C Data types

C provides some basic data types

- `int` – used to store integers in moderate ranges
- `long` – used to store integers in much larger ranges
- `float` – used to store floating point numbers, i.e. real numbers
- `double` – used to store real numbers with higher precision
- `char` – a special type of 1-byte integer that can be interpreted as an ASCII character

In addition, the ranges of integer types can be modified by creating an `unsigned` version

- An unsigned integer uses all its possible bit combinations to store positive integers only

Based on the platform, different data types may take different amount of space

C Operators

The most commonly used operator is the assignment operator (=)

- It is used to assign the value of a variable or an expression on the right to a variable on the left…
- e.g. `i = j + 5;`

Other common operators include relational operators, which compare two values pr expressions

- The equality and inequality can be checked by == and != operators
- For inequality, there are operators <, >, <= and >=
- They all return `1` on comparison being successful and `0` otherwise

There are Arithmetic operators to perform computations

- `/`, `*`, + and − being used for division, multiplication addition and subtraction respectively
- Parentheses can be used to enforce precedence during computations

There are other specific operators too, such as the `&` operator and the −> operator…

- Revise all the C operators again, because they are also a part of C++

C Conditional Statements

Conditional statements are used to change the course of evaluation in a program

The most common way to do so is using an `if` statement

- Syntax:
  `if(condition) { statements to execute when the condition is true}`
- You can also add an `else` statement
- Syntax:
  `if(condition) { statements to execute when the condition is true}`
  `else { statements to execute when the condition is false}`
- The `else` can be immediately followed by another `if` statement, to make it look like `if-else-if`

Another option at your disposal is to use a `switch-case` construct

- Syntax:
  `switch(s) { case c:` $\texttt{statements}_c$ `case d:` $\texttt{statements}_d$ … `default:` $\texttt{statements}_{def}$`}`
- Here, `s` must be an integer type, and `c`, `d` etc. are its possible values
- Remember, without a `break` statement, the control *falls through* the `case` statements from top to bottom

C Iterative Statements

Iterative statements are used execute some logic repeatedly in a controlled fashion

There are three iterative statements that you may use

- The `do-while()` loop is an *exit controlled* loop, i.e. the loop condition is evaluated after the loop body
  Syntax:
  ```
  do { loop body } while(loop condition);
  ```
- The `while()` loop is an *entry controlled* loop, i.e. the loop condition is evaluated before the loop body
  Syntax:
  ```
  while(loop condition) { loop body }
  ```
- The `for()` loop is also an *entry controlled* loop, with slots for *initialisation* and *update* of variables as well
  Syntax:
  ```
  for(initialisation; loop condition; update) { loop body }
  ```

In addition, the `break` statement can take the control out of the innermost loop or switch

Pointers in C

Pointer variables in C are a special type of integer variables which store memory addresses

In general, the type of pointer variables is the same as the the type of the addresses they store

- For example
  ```
  double *ptr; // stores the address of a double variable
  ```

The `&` operator provides the address of a variable

- For example
  ```
  ptr = &d; // d is a double variable
  ```

The `*` operator returns the value stored at an address

- For example
  ```
  d2 = *ptr; // d2 is also double variable
  ```

Pointers, with the `malloc()` and `free()` library functions, allow usage of dynamic memory blocks

Custom Functions in C

A custom function in C can be declared using a prototype

- Syntax
  `<return type> <function name>(<0 or more parameter types>);`

A custom function in C can be defined using a function header followed by a code block

- Header Syntax
  `<return type> <function name>(0 or more <<type> <variable>>)`

A `return` statement can return a value of suitable type from the function

- The type should match the return type in prototype or header

All parameters are *passed by value*, i.e. the values are copied from calling to called function

To emulate *passing by reference*, Pointer variables are used…

- … which result in copying of an address rather than a value

Arrays and Structures in C

Arrays are collection of variables of the *same type*

- The length of an array remains fixed after its creation and cannot be modified
- Syntax
  `<data type> <array name>[<array capacity>];`
- A `char` array, when terminated by a special character (`'\0'`), is treated as a string

Structures are collection of variables of *different types*

- The variables in a structure variable remain fixed after its creation and cannot be modified
- Prior to creating structure variables, a template providing details of its member variables must be provided
- Syntax for defining the template
  `struct <structure name> { member variables };`
- Syntax for creating structure variables
  `struct <structure name> <structure variable name>;`

Revise everything if you feel “rusty”

Go through the material for the *Introduction to Programming* course if required

I will assume you know all the elements discussed today, as well as many more…

- … thus, I will not discuss them again for C++

We will start with C++ from next week !!