Computer Languages

&

A look at ‘C’

Objectives

- Write Simple Computer Program.
- Use Simple Input Output.
- Understand the fundamentals of Data types.
- To use arithmetic operators.
- To understand the precedence of arithmetic operators.
- To write simple decision making programs.

Problem Solving

- Real life problem needs to be translated into a form that a digital computer can understand and provide answers.
- Digital computers deal with Logic and arithmetic operations.
- Real life problems has to be translated into logic and arithmetic.

Logic

- AND
- OR
- NOT
- NAND
- NOR
- XOR
- < , > , =< ,>=

uses the above logic for decision making

All have their own truth table

Arithmetic

- Addition.
- Subtraction.
- Multiplication.
- Division.

Statements

- Computer languages provide various statements to support logical and arithmetic operations.
- Decision making is based on the outcome of the logical and arithmetic operations.
- Logical and arithmetic operations are used in a defined structure provided by the language.
- Each statement has a unique format.

Computer Languages

There are Three levels of Computer Languages:

1- The Lowest Level

This consist of Binary Code which Computer understand it (Difficult for Users to get to grips with).

Requires:

Bit patterns as instructions i.e. 01011110011111
Each bit patterns means something. The patterns get other bit patterns as a sequence to control the hardware.

2- Assembly Language

This consist of symbolic representation of Binary codes (Mnemonics) i.e. letters of alphabets are used to generate Operational code.

E.g. LDA {Load a machine register},
MOVE {move data from source to destination}

Requires:

- **An assembler**
- **Considerable computing Knowledge**
- **Understanding of the Computer System Architecture**

Gives:

- **The programmer complete control Over the system**
- **The Most efficient code can be generated.**

3- High Level Languages

Designed to counter the requirements and difficulties encountered in:

- Remembering machine code which is impossible
- Remembering or referring to user manual for Mnemonics (Operational codes).
- Having Considerable Knowledge of hardware

Provides

Usage of high level statements which is closer to natural languages. i.e. using words and statements such as write, Print, Read, While, Do,… …..etc

Popular languages,: VB, JAVA, C++, C#…etc

Requirement: Compiler

Disadvantages:

Machine code is provided by translation and is not efficient (many redundancies generated by Compiler)

It is slow Running, Occupies more space (Memory)

O.K for PC, or Mini Computer or Mainframe

But for Micro, or Embedded Architecture, It may pose a problem.

‘C’ Language

- ‘C’ seems not to fit into any of these neat categories mentioned, though it is a high level language.
- It lacks some features found in other languages
- It gives very nearly the control which assembler languages offer.

However,

- Tedious to write.
  (Say writing a stock control package in ‘C’.)

BUT

- It is a very handy language for producing:

  Operating systems.
  Compilers.
  Interpreters for other languages.
  Word processing packages.
  programming Embedded systems.

- ‘C’ Comes from people who were responsible for giving us **UNIX**

The Bell Laboratories in the USA.

- Stems originally from the British-Produced language BCPL via an intermediate language called ‘B’.

- At the time, controversy appeared to exist as to whether the next language in the family should be called P or D !

- Well, we ended up with:

C# and C++, two Object Oriented Languages.

First Impression

‘C’ is a Structured Language

But, it look absolutely awful.

‘C’ makes little concession to readability

Cryptic symbols are preferred to English words.

’ {‘ = Begin, ‘}’ =End &&= AND….etc

Antiquated PASCAL uses Begin and End

Style differences

- Major Style differences exist between ‘C’ and other languages.
- Some languages demands that programs start with functions and subroutines are defined first, then the main program.

  This is theoretically a neat way of writing a program.

Advantages:

Neater way to write a Program
Compilation process easier

Disadvantage :

Does not make the listing easier to read.

'C' imposes no restriction when and where the procedures, functions are decelerated
i.e. No order is observed.

But,

The main module must be identified by the word ' main () ' in case Arduino ' loop ()'

The 'main()' or 'loop()' can appear any where in the program

- There are no restriction on indentation.

The Structure of a “C” program

- Inclusion of **libraries**.
- Declaration of **constants**, variables, data structure, such as arrays……etc.
- Setups or initializations.
- Declaration of functions.
- The main program. main(), or loop()

Example of a Basic structure

```
#include <stdio.h>
Int i=0;
Main()
{
Printf(“I am learning C”\n);
Return 0;                    /*indicates program ended successfully*/
}
```

Arduino “C” Structure

Data structural deceleration

Function deceleration

```
void setup()
        {
    Statements
        }


void loop()
    {
     Statements
    }
```

Example BLINK

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  This example code is in the public domain.
 */

void setup()
 {
  // initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
 }

void loop()
{
  digitalWrite(13, HIGH);     // set the LED on
  delay(1000);                // wait for a second
  digitalWrite(13, LOW);    // set the LED off
  delay(1000);                // wait for a second
}
```

Arduino “C” program Structure

- Declarations

- void setup()
- {
  // initialize the hardware
  Statements
- }

- void loop()
- {
  The main program, function calls……Etc
- }

Declaration

- ‘C’ demands that all variables and their type are declared first before they can be used.
- char : Character
- int : Integer
- float : Single precision floating point
- double : double precision floating point

e.g.

```
int  i=0,j=5,k=7 ;    // defines i, j , k as integers
char  c;              // defines letter ‘c’ as character
```

- Arrays are defined as:

```
int matrix[10]      // Defines an array of 10 integers.
```

Strings

- 'C' Contains no provisions for holding strings (collections of characters) as complete units.
- Strings are defined and handled as arrays of characters.

This is awkward at times, but generally very useful for the types of applications for which 'C' is most suited.

e.g. char **one-word[6]= "Hello";**

Below are all the valid declaration of string of characters

```
char Text1[15]  // 15 character space is defined uninitialized
char Text2[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o'}; // Location 8 is Terminator
char Text3[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o', '\0'};// ' \0' is the terminator
char Text4[ ] = "Arduino";
char Text5[8] = "Arduino"; //Initialize the array with space for terminator
char Text6[15] = "Arduino";// Initialize the array leaving extra space for a larger  str
```

External & Internal variables

- External

These variables are declared outside of the ‘main’ or ‘loop’ (in Arduino) program and any functions and are re-declared within each functions which uses them.

e.g. funct (value);

This is handled within the function by:

```
funct(value)
int value;      //value is an integer
{
    Statement;
}
return(value) // value is a returned integer.
```

Operators

- C, Provides a useful range of operators.
- Usual ones are arithmetic operators (+,-,*,/)
- Unusual ones are increment and decrement.

e.g. X=X+1; Y=Y-1;

'C' allows you to say: ++X instead of X=X+1

--Y instead of Y=Y-1

++, or -- can be suffixes as well as prefixes

e.g. If a=6; X=a-- means a:=5 and x:=5

':=' means becomes i.e. x must be a variable.

Relational Operators

- ‘>’ Greater than ‘>=‘ Greater than or Equal
- ‘<‘ Less than, ‘<=‘ Less than or Equal
- ‘==‘ Test for equality
- ‘!=‘ Test for inequality

Logical Operators

- ‘&&’ ---------> Logical AND
- ‘||’ ----------> Logical OR
- ‘!’ ----------> Logical NOT
- ‘<<‘ ----------> Bit shift left

e.g. int a=5 ; // Binary 00000101

int b=a<< 3; Means b=00101000

- ‘>>’ ----------> Bit shift right