Summary

Embedded C

1. C identifiers

Used for Identifying variables ex. int x or char x

Note : -

Lowercase for variables

Uppercase for constants

Each variables is given a type that defines what values it can hold and represent and what operations can be performed on it and how data can be stored.

C data types

Char : usually 8 bit (1 byte)

Int : usually the natural size for a machine or OS (e.g 16 , 32 , 64 )

-short int : at least 16 bits

-float : 32 bits

-long int : 32 bits

-double : 64 bits

-long double : 64 bits

Note : BIT or Boolean can hold two values (either 1 or 0 which means True or False ) .

It is used to carry out the result of Boolean logical expression or the binary status of a peripheral comparsion .

Data Types :

1: floating type : specified by a decimal point after a decimal.

Ex 3.14 f or 2.f float

7.L is Long double

It can also be written like that 1.65 e-2 which means .0165 .

Note : 3+7+9.2 is a simple calculation which is carried out by only the compiler in a time called compile-time so the processor is not included in the calculation but if the calculation is done during the runtime then the processor does the calculations .

2: Character constants :

Specified with single quotes like ‘7’ , ‘a’ ,’\n’.

Character are represented by ASCII character set which associates the integers 0 to 127 with specific characters .

3: string constants :

Specified with double quotes like “7” , “word”.

String constants would comprise the following sequence : this is a string \0 .

Note : ‘x’ is not like “x” since the first one is a character and the other is a word and the first one size is four and the second one size is two.

Type specifiers : -

1. Signed
2. Unsigned

They both an be applied to a char or int

1. Signed

It may represent negative values

MBS most significant bit is the sign bit

For 1 byte there’s 255 possible values   
if value is higher than 125 nearly we define it as unsigned .

Ex : 16 bit signed short can represent the numbers from – 32768 to 32767 like (-215 to 215-1) .

Hint : integer types are signed by default (writing short is equivalent to signed short int ).

1. Unsigned

A 16 bit unsigned short can represent the numbers from 0 to 65535 (2^16).

Unsigned is always non-negative and the MSB is part of numerical value doubling the maximum representable value compared to an equivalent signed type .

Note: char type is unsigned by default.

Int is signed by default

Hint :

Signed : signed int

Short: signed short int

Int : signed

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Type qualifiers :

Volatile refers to variables whose value may change in a manner beyond the control of the program.

Useful for multi threaded programming or interfacing to hardware.

Ex.empty delay loops

Qualifiers : Constant and Volatile.

1: constant

It means that the variable that it refers to cannot be changed to another value.

Const int x = 5 ; >>>> it’s stored in ROM .

X=6 ; will not be changed.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Functions:

First it’s better to declare functions using prototypes .

The type of the return informs the compiler how much to allocate in RAM to hold that value .

If we use a function that doesn’t use a parameter we call it void .

We should provide full definition for the function before program calls it .

We use a prototype of the function to alert the compiler before you actually define it .

Prototype describes the interface of the function .

The definition includes the statements that are executed .

The compiler reserves enough program memory to hold the statements In the function and stores the address of the first statement with the function name.

It is better to avoid global variables if possible .

There is only one return of the function.

Passing by value :

Passing dating by value makes the compilation show and takes up more space in the memory.

Variables are stored in RAM .

Instructions are stored in ROM.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Loops :

Matching else and if

An else always matches with the nearest un matched if .

If (a)

B=1;

If(!a)

B = 2 ;

Else

B=3;

? operator :

Max=(a<b)?(a):(b) ;

Max=(a<b) is the condition

A is return value when condition is true

B is return value when condition is false

Switch case :

It is actually less flexible than if.

Switch may only test int types and char types.

Do-while :

It executes at least once because of do then the rest is like while.

Go-to loop :

The statement then goto loop(or the name of the loop)

For :

Variable inside condition must be defined

Ex. For(int i =0 ; i <9 ; i++)

It’s preferred to define the variables before for.

We use **break** statement to exit a loop.

And the execution passes to the statements following the loop

We use **continue** to ignore the rest of the statements inside the loop.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Logical operators :

AND && ,,,,,,,OR ||

They return result of True or False.

C uses short circuiting of logical expressions which means that the computer will only evaluate as much of logical expressions as is necessary to determine if the expression has the value 1 or 0

So it saves time of more processing.

BIT operators or bitwise operator :

1.logical operators

2.shift operators

Binary operators perform data promotion on operands to make sure both are of same size .

XOR (^)

Note: xoring with 0xff will toggle the status .

Note:

Set : reg | 1<< bit

Clear : reg & (~(1<<bit))

Shift operators:

1. Shift right :

x>>num of shifts

1>>5 shift 1 5 times to the right

1. Shift left

X<<num of shifts

4<<3 shift 4 3 times to the left

Note : ~ not is used to inverse the binary representation of a number .

Ex : ~ 1010 = 0101

Note :with SET we use OR |.

With CLEAR we use AND &.

RESET : Reg & 0 .

READ : 1 & (reg>>bit).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

C preprocessor :

Any command that starts with # .

It’s processed during the preprocessing time when the language is converted to assembly language.

It’s not included in compile time.

It doesn’t understand C language.

/ forward slash : is used to attach the content of the next line to the end of the current directive .

Common error : using semicolons , comment and whitespaces.

1. **#include** used with

1: < > with the standard library ex <stdio.h>.

2:”” with files produced by the manufacturer or the user.

2.**#Define**: it means remove and place ex #define led 5.

Used with macro operation.

Pure text replacement.

It will allocate memory each time it’s used

In function call the memory is allocated for only once and each time the function is called a pointer to the function is used.

#define c.area(r) (pi\*r\*r)

Num of allocations depends on num of calls .

Float c.area(int r)

{

Return 3.14\*r\*r ;

}

Num of allocations is always one no matter how many times it’s called.

That’s because when it’s called it jumps to the function and return with the result.

Constant qualifiers :

# define max 30

It’s constant.

We use it to limit the array size

Int array[max] ;

Const int max = 30 ; it’s stored in ROM not RAM because it’s constant.

Note: Const int can’t be used to define an array or switch case.

Compiler considers it as integer not constant.

Macros definition:

Macros like functions but they are pure text replacement

And they are preprocessor operations.

Why we use macros !?

1. They are faster than functions and there’s no stack calls or values push and pop .
2. It accepts any kind of value (because of compiler casting)

But it takes up more memory.

3-Macros accept any kind of values.

Int max (int , int) the function accepts only integers as parameters.

#define max (x,y) (x>y) ? (x):(y) here casting happens by the compiler so it accepts any kind.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Scope and Extent :**

The Scope : the part of the program within which the name can be used .

It describes the visibility of an identifier with the program local and external.

The Extent :it describes the lifetime of a variable ,when the memory is allocated to store it and when that memory is released.

It has two kinds : automatic and static.

Note : look up compiler table :- it shows the places available in the memory for being used or allocated.

Local operand auto extent:

A variable declared in a function that has local scope by default

{} these represent the code blocks.

Local variables with the same name defined but different scopes or blocks or functions are unrelated.

Local variable has automatic extent which means lifetime is from the point it’s defined until the end of the block.

At this point memory is allocated on the stack.

The memory is managed automatically by the compiler.

If the variable is not explicitly initialized then it will hold a garbage value or an undefined value.

It’s better to initialize the variable.

At the end of the block the variables are destroyed an the memory is recovered … the variable is said to go “out of scope “

Local variables are also called automatic variables.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* **Declaration and Definition** **:**

A Declaration of a function or variables declares that the variable or function exists somewhere .

A Definition of variable or function allocates memory for the variable or the function.

We can do declaration k=more than once unlike the definition which can be done only once .

**Extern keyword:**

We use it to share variables with another file.

Extern int x ; // Declaration

Note : if we don’t want to share a variable with another file we use “Static”.

Note: function is extern by default thus it can be used in another file.

Extern: scope // sharing the scope before using you must declare (variable or function).

We don’t actually write extern with functions because it’s extern by default (the compiler considers it as extern) .

Note: To make variables with the same name but not the same value we use static word in the other file.

**Static and Global variables :**

Local variables : int value garbage if not initialized and auto extent.

Global variables:

Note : for global variable initializing value = 0 .

Scope : all the program.

Problem of the global variables is there could be sharing between files.

Using static int will solve the problem since it makes the variable only used in the file.

**Static with global variables**:

Inside the function scope if we initialize a variable as static it will not be destroyed once the function is executed and it will exist till the end of the program .

Note :The static variable inside the function after it’s called again the compiler will the initializing value of the static variable so it has another value.