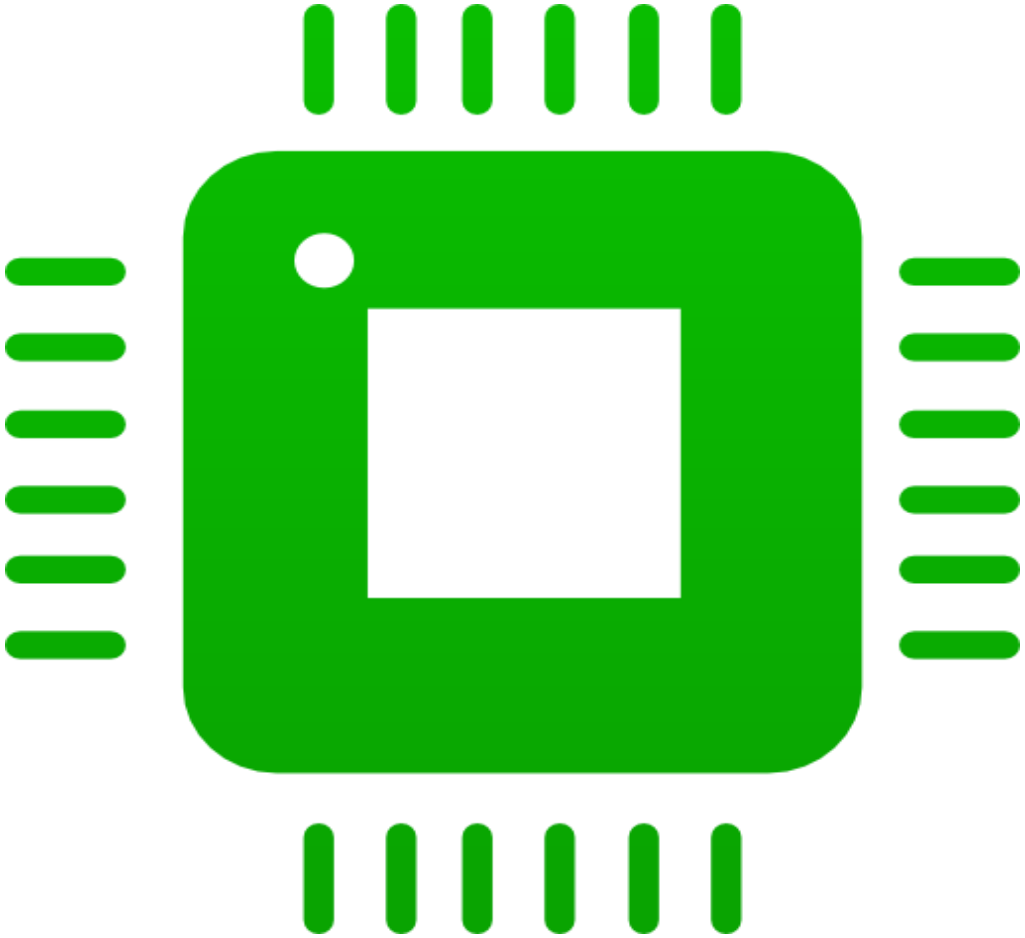
Fab Lab Ismailia represent :
Embedded Systems Workshop
by : Mohammed hemed



C for Embedded systems

Mind map for our session

- Main goal : Learn c language basics from the side view of embedded systems .
- Introduction to C language .
- C vs Embedded C
- What is a compiler
- Why we program AVR in C
- C data types
- Variables
- I/O programming in C
- Bitwise operations in C
- Flowcharts
- C Control Statements
- Functions
- Constants & preprocessors & diff bet executive and directive instructions
- Write your own configuration header file
- Write your own libraries

Introduction to C language

- C language was developed by Dennis Ritchie in 1969. It is a collection of one or more functions, and every function is a collection of statements performing a specific task.

- Middle-level language as it supports high-level applications and low-level applications .

- C language is a software designed with different keywords, data types, variables, constants, etc.

- Embedded C is a generic term given to a programming language written in C, which is associated with a particular hardware architecture.

- Embedded C is an extension to the C language with some additional header files. These header files may change from controller to controller.

C vs Embedded C

C programming	Embedded C programming
Possesses native development in nature.	Possesses cross development in nature.
Independent of hardware architecture.	Dependent on hardware architecture (microcontroller or other devices).
Used for Desktop applications, OS and PC memories.	Used for limited resources like RAM, ROM and I/O peripherals on embedded controller.

What is a compiler ?

- a computer program that translates a program written in a high-level language into another language, usually machine language .

- So let us look at this as Embedded developers :

When we run our code on microcontroller : the compiler produce hex file , then we download into the flash of the microcontroller , the size of hex files is one of the main concerns as we have limited on-chip flash .

- You should know that it's a trade-off :

If I write my program in assembly language It's very good in performance and code size , but it's very long in the time of code development .

So Why we program AVR in C ?

- Easier and take less time , easy to update .

- Code is portable which mean you can use same code for other microcontrollers with little or no modification .

- It have the advantage of high level language ease , and the force of control of hardware like talk to the place where store the variable

or the (pointer) , it also can allocate memory

to benefit from the memory as much as you can .

- Learning ANCI C give you the ability to deal to all uCs like : ARM

C data types

- We [illegible] ; we can , fc

- we [illegible] n't need (i [illegible] .

Table 7-1: Some Data Types Widely Used by C Compilers

Data Type	Size in Bits	Data Range/Usage
unsigned char	8-bit	0 to 255
char	8-bit	–128 to +127
unsigned int	16-bit	0 to 65,535
int	16-bit	–32,768 to +32,767
unsigned long	32-bit	0 to 4,294,967,295
long	32-bit	–2,147,483,648 to +2,147,483,648
float	32-bit	±1.175e-38 to ±3.402e38
double	32-bit	±1.175e-38 to ±3.402e38

Variables & Data types

- In the world of uC we have some constrains as most of 8-bit uCs have a limited size for RAM or ROM (some uC may have only 128 bytes) so we have an efficient way to write code that achieve the goal with little memory consumption.
- In contrast when we write code in PC which has a big processor and RAMs , we don't care the size of each variable , and the size of the whole code .
- Digital data : in Embedded systems we usually deal with digital data to express the values of the different variables like :

 - sensors readings (temperature - pressure - humidity - light , ...)

 - PORT value - time - motor speed , ...

- The numbers are considered the most common used data in ES world , which the most thing which consume the SRAM .

I/O programming in C

- To access a port register as a byte we use "PORTx" label .

- We choose our port as input or output via "DDRX' register .

* Where x = port name (A – B – C– D)

For example :

DDRD = 0b11111111 ; ------ > this line mean make all PORTD bits as output

DDRD = 0b00000000 ; ------ > this line mean make all PORTD bits as input

PORTD = 0 ; ------ > this line mean make all PORTD bits = zero logic level

PortD = 11111111 ; ------ > this line mean make all PORTD bits = one logic level

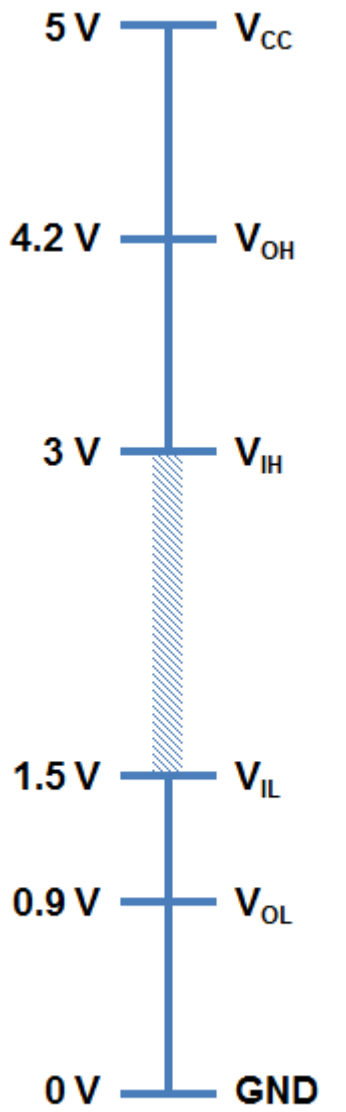
- Logic level in our case (atmega32) :

If Vcc = 5v our high logic level is in between (vcc : vcc *0.6) = 3v : 5V

our low logic level is in between (0v : vcc *0.3) = 0v : 1.5 v

- The area between high logic level and low logic level is called

Noise margin – invalid state – undefined state



ATMega328
DC Characteristics

Bitwise operation in C

- One of the most important and powerful features of the C language is its ability to perform bit manipulation .

- (And &)– (OR |) – (EX-Or ^) – (invert or not ~) .

Table 7-2: Bit-wise Logic Operators for C

		AND	OR	EX-OR	Inverter
A	B	A&B	A|B	A^B	Y=~B
0	0	0	0	0	1
0	1	0	1	1	0
1	0	0	1	1	
1	1	1	1	0	

The following shows some examples using the C bit-wise operators:

```
1.  0x35 & 0x0F = 0x05      /* ANDing */
2.  0x04 | 0x68 = 0x6C      /* ORing  */
3.  0x54 ^ 0x78 = 0x2C      /* XORing */
4.  ~0x55 = 0xAA            /* Inverting 55H */
```

Bitwise operation in C

```
C = A & B;          A   0 1 1 0 0 1 1 0
(AND)               B   1 0 1 1 0 0 1 1
                    C   0 0 1 0 0 0 1 0

C = A | B;          A   0 1 1 0 0 1 0 0
(OR)                B   0 0 0 1 0 0 0 0
                    C   0 1 1 1 0 1 0 0

C = A ^ B;          A   0 1 1 0 0 1 0 0
(XOR)               B   1 0 1 1 0 0 1 1
                    C   1 1 0 1 0 1 1 1

B = ~A;             A   0 1 1 0 0 1 0 0
(COMPLEMENT)        B   1 0 0 1 1 0 1 1
```

Bit set/reset/complement/test
Bit masking

- Use a "mask" to select bit(s) to be altered

```
C = A & 0xFE;     A    a b c d e f g h
                  0xFE 1 1 1 1 1 1 1 0   Clear selected bit of A
                  C    a b c d e f g 0

C = A & 0x01;     A    a b c d e f g h
                  0xFE 0 0 0 0 0 0 0 1   Clear all but the selected bit of A
                  C    0 0 0 0 0 0 0 h

C = A | 0x01;     A    a b c d e f g h
                  0x01 0 0 0 0 0 0 0 1   Set selected bit of A
                  C    a b c d e f g 1

C = A ^ 0x01;     A    a b c d e f g h
                  0x01 0 0 0 0 0 0 0 1   Complement selected bit of A
                  C    a b c d e f g h'
```

Bitwise shift operation in C

Table 7-4: Bit-wise Shift Operators for C

Operation	Symbol	Format of Shift Operation
Shift right	>>	data >> number of bits to be shifted right
Shift left	<<	data << number of bits to be shifted left

The following shows some examples of shift operators in C:

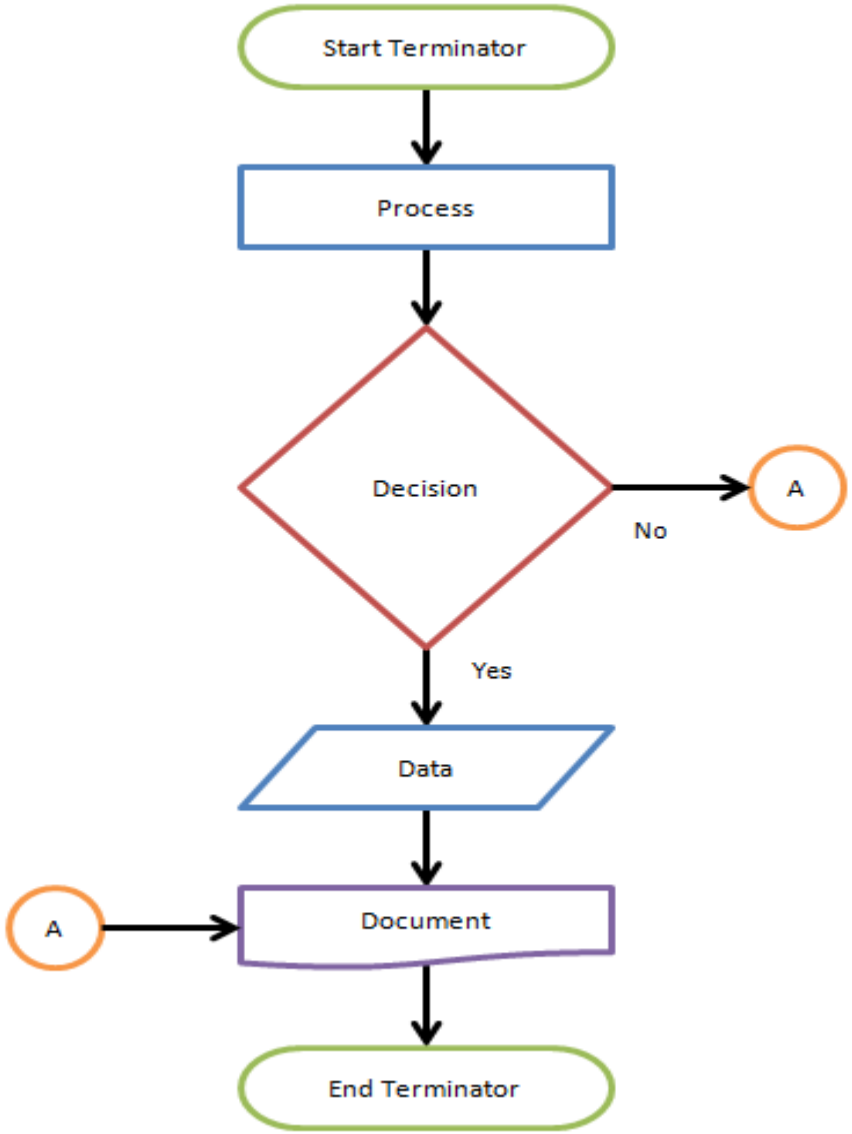
1. 0b00010000 >> 3 = 0b00000010 /* shifting right 3 times */
2. 0b00010000 << 3 = 0b10000000 /* shifting left 3 times */
3. 1 << 3 = 0b00001000 /* shifting left 3 times */

Flowcharts

• A flowchart is a diagram that describes a process, system or computer algorithm.

- They are widely used in multiple fields to document, study, plan, improve and communicate often complex processes in clear, easy-to-understand diagrams.

- As many developers we design our code using pseudo code or flowcharts to simplify the code , Then write the actual code .



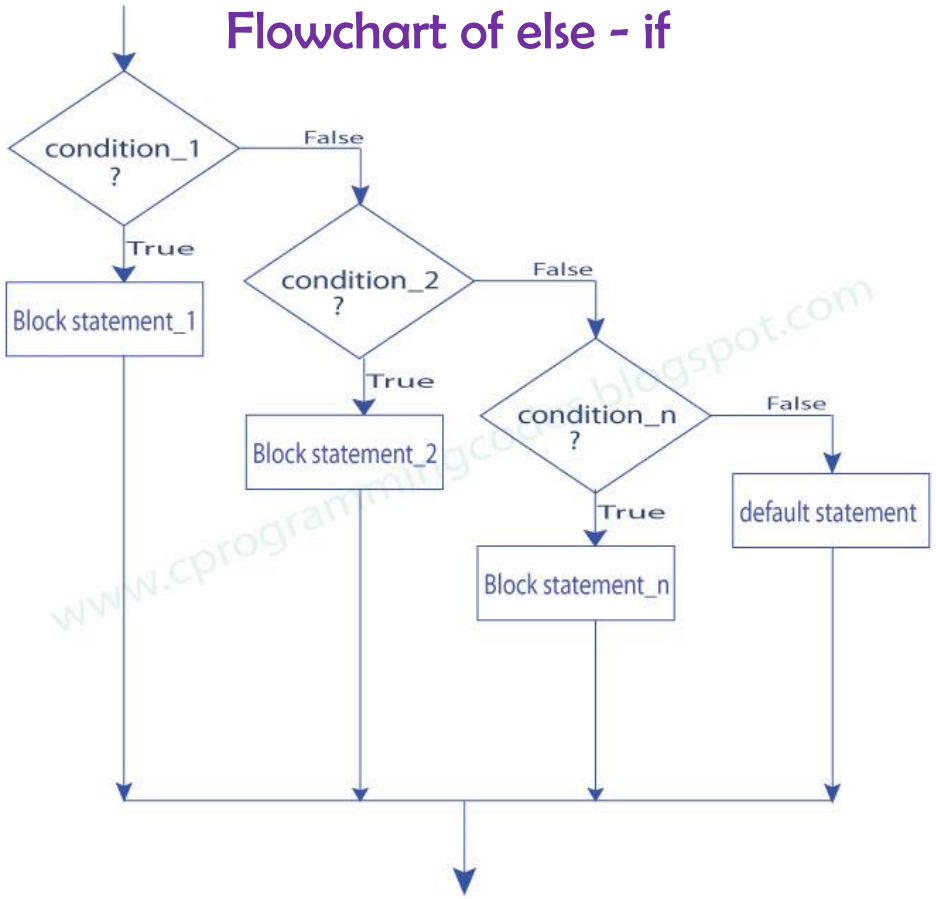
C language control Statements

- C provides two sytles of flow control:
- Branching : the program chooses to follow one branch or another
- If
- if-else
- else if
- switch
- break & continue :
- Looping : Loops provide a way to repeat commands and control how many times they are repeated.
- while
- do…while
- for

If statement

- IF statement : It is used to execute an instruction or sequence/block of instruction only if a condition is fulfilled.
- Different forms of implements if-statement are :

Flowchart of else - if



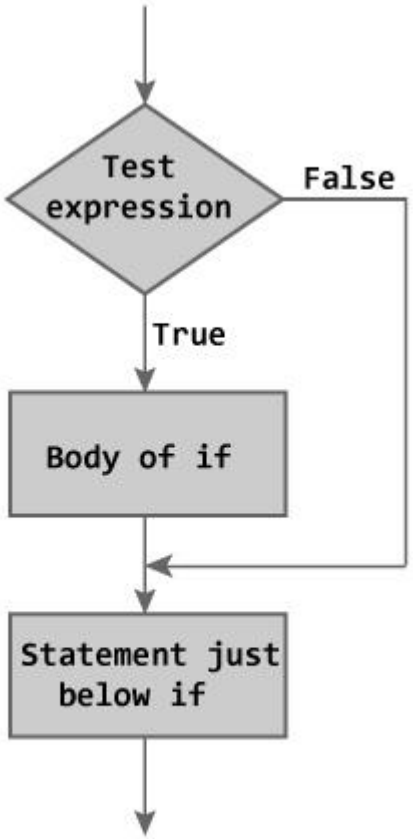


Figure: Flowchart of if Statement

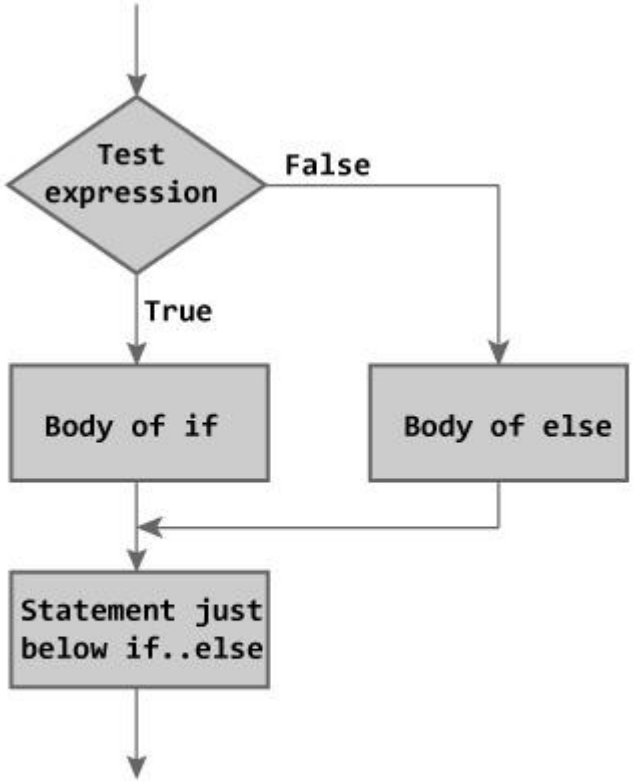


Figure: Flowchart of if...else Statement

Switch case

- A switch statement allows a variable to be

tested for equality against a list of values ,each value is called a case,

and the variable being switched on is checked for each switch case.

- break statement : break statement in C programming

has the following two usages :

- When a break statement is encountered inside a loop,

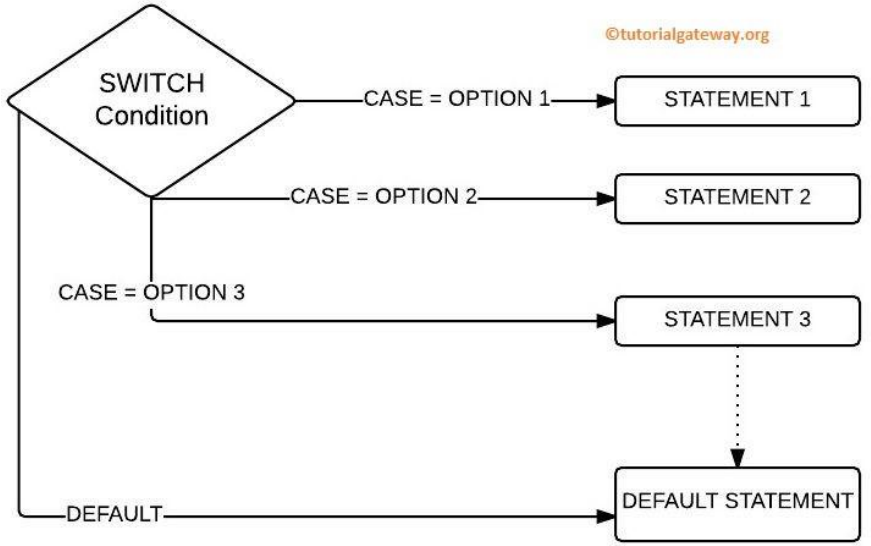
the loop is immediately terminated and the program

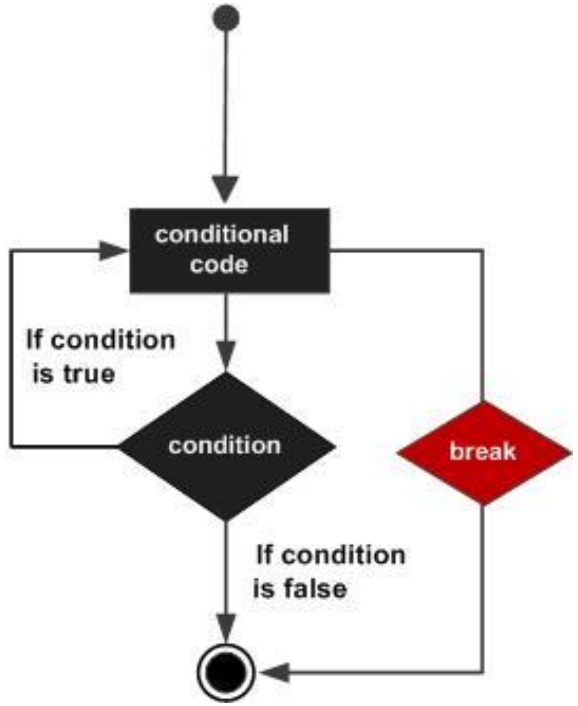
control resumes at the

next statement following the loop :

```
for (int a =0 ; a<8 ; a++)
{
PORTA = a ;
If (a == 5) break ;
}
```

- It can be used to terminate a case in the switch statement .





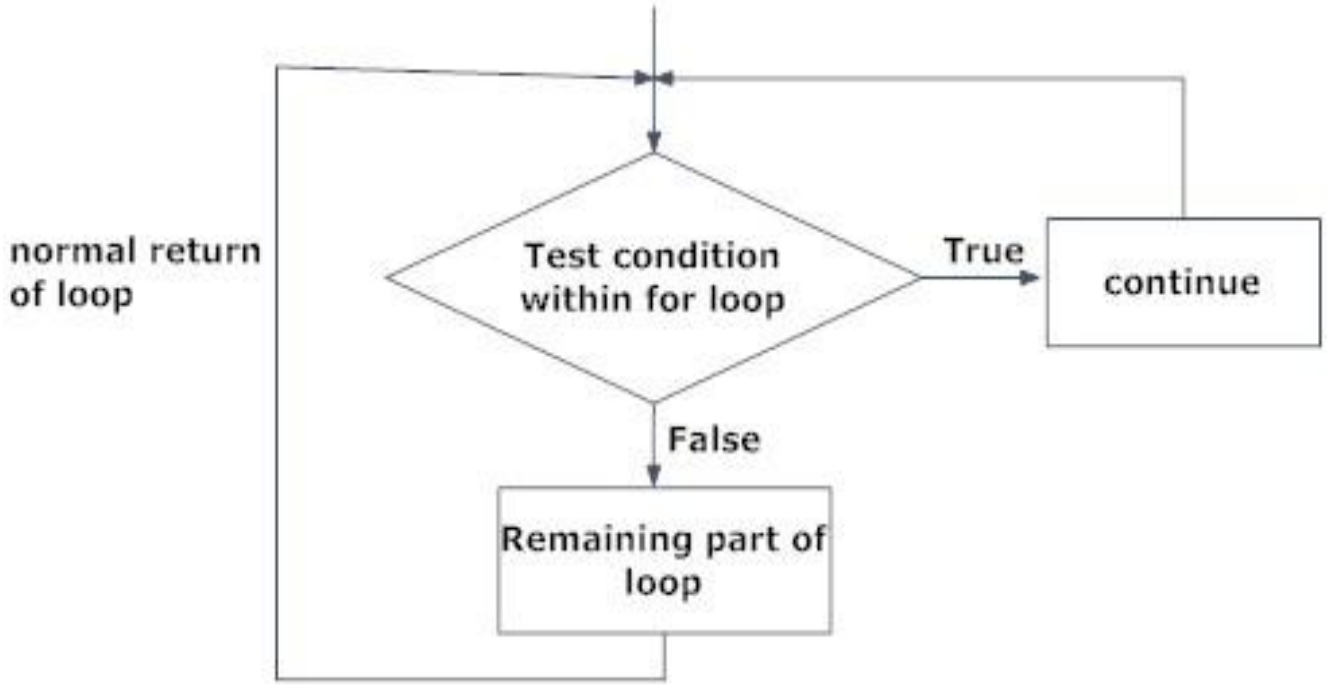
Switch case example with break

```c
#include <stdio.h>

int main () {

   /* local variable definition */
   char grade = 'B';

   switch(grade) {
      case 'A' :
         printf("Excellent!\n" );
         break;
      case 'B' :
      case 'C' :
         printf("Well done\n" );
         break;
      case 'D' :
         printf("You passed\n" );
         break;
      case 'F' :
         printf("Better try again\n" );
         break;
      default :
         printf("Invalid grade\n" );
   }

   printf("Your grade is  %c\n", grade );

   return 0;
}
```

continue Statement

- Continue : The continue statement in C programming works somewhat like the break statement. Instead of forcing termination, it forces the next iteration of the loop to take place, skipping any code in between.
- For the for loop, continue statement causes the conditional test and increment portions of the loop to execute. For the while and do...whileloops, continue statement causes the program control to pass to the conditional tests.

```
for (sint8 a =0 ; a<8 ; a++)
{
// if false the next statement will execute
If (a == 5) continue ;
// if true this statement will skip
PORTA = a ;
}
```



While loop

- In microcontroller applications ,we don't want our product to run for only one time(unless you're a bomb maker) so while building programs we put operating functions within the while loop.

```
While( Some Condition or 1(mostly in main code) )
{
Check or do task-1
Check or do task-2
Check or do task-3
}
```

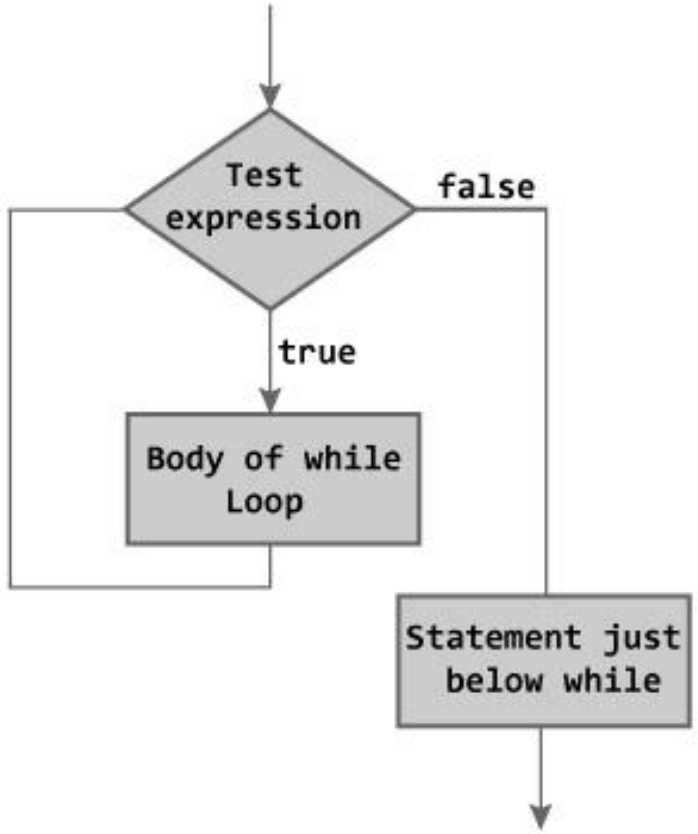


Figure: Flowchart of while Loop

For loop

- A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

/* for loop to set all bits of portA */

for (sint8 a = 0 ; a<=7 ; a++) PORTA |=(1<<a);

Syntax

The syntax of a **for** loop in C programming language is –

```
for ( init; condition; increment ) {
   statement(s);
}
```

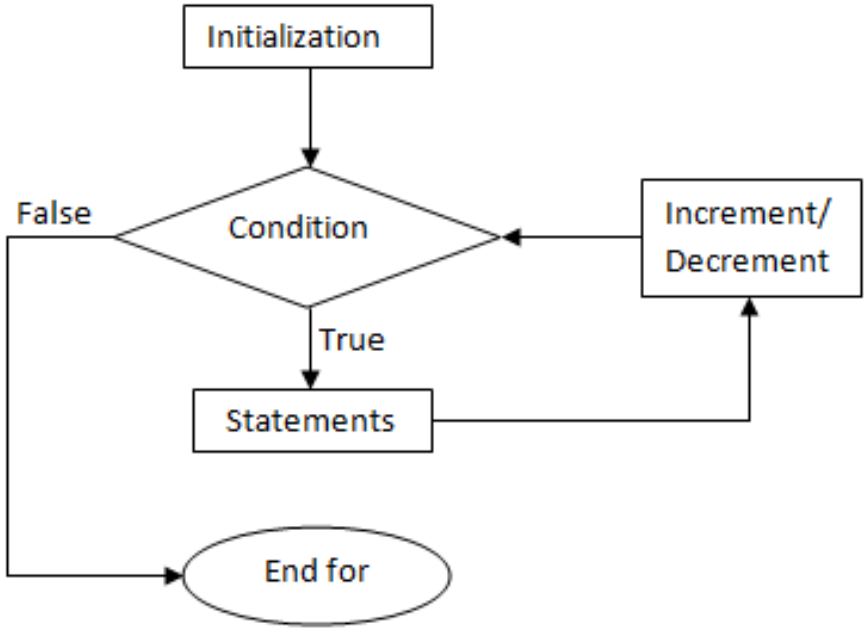


fig: Flowchart for for loop

do...while loop

- Unlike for and while loops, which test the loop condition at the top of the loop, the do...while loop in C programming checks its condition at the bottom of the loop.
- A do...while loop is similar to a while loop, except the fact that it is guaranteed to execute at least one time.

Syntax

The syntax of a **do...while** loop in C programming language is –

```
do {
   statement(s);
} while( condition );
```

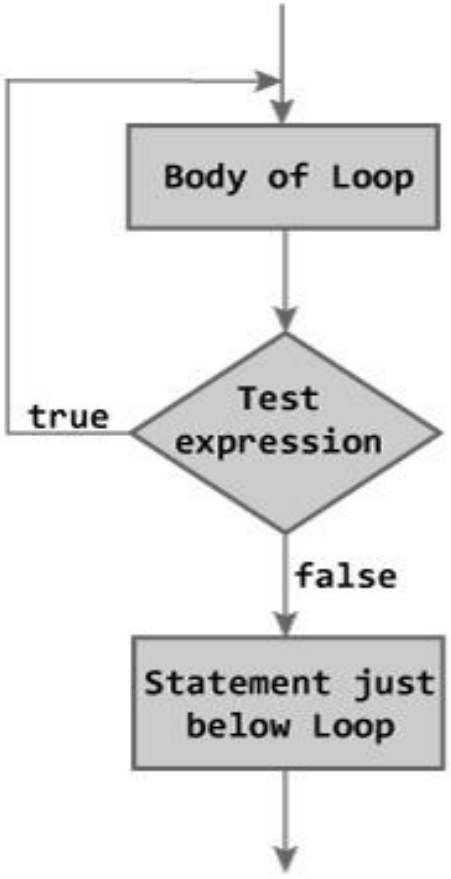


Figure: Flowchart of do...while Loop

Functions

- A function is a group of statements that together perform a task. Every C program has at least one function, which is **main()**, and all the most trivial programs can define additional functions
- You can divide up your code into separate functions.

How you divide up your code among different functions is up to you,

but logically the division is such that each function performs a specific task.

- A function **declaration** tells the compiler about a function's name, return type, and parameters. A function **definition** provides the actual body of the function

General form of function definition

```
return_type function_name( parameter list ) {
   body of the function
}
```

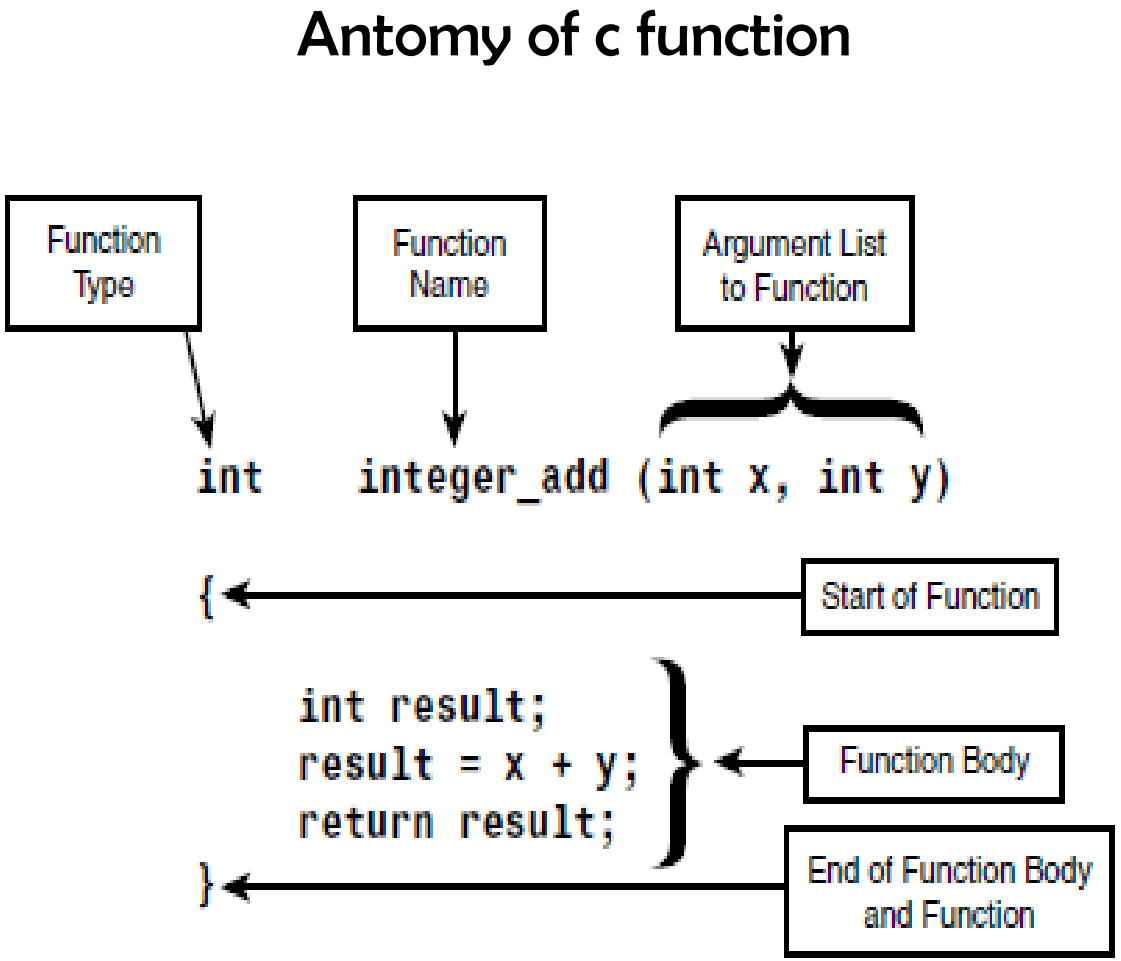
- **Return Type** – A function may return a value. The **return_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword **void**.
- **Function Name** – This is the actual name of the function. The function name and the parameter list together constitute the function signature.
- **Parameters** – A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
- **Function Body** – The function body contains a collection of statements that define what the function does.

Example

```
/* function returning the max between two numbers */
int max(int num1, int num2) {

   /* local variable declaration */
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```

Function Anatomy - Calling a function

```c
int main () {

   /* local variable definition */
   int a = 100;
   int b = 200;
   int ret;

   /* calling a function to get max value */
   ret = max(a, b);

   printf( "Max value is : %d\n", ret );

   return 0;
}

/* function returning the max between two numbers */
int max(int num1, int num2) {

   /* local variable declaration */
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```



Constants & pre-processor

- The **C Preprocessor** is not a part of the compiler, but is a separate step in the compilation process. In simple terms, a C Preprocessor is just a text substitution tool and it instructs the compiler to do required pre-processing before the actual compilation.
- If there is a constant appearing in several places in your program, it's a good idea to associate a symbolic name to the constant, and then use the symbolic name to replace the constant throughout the program. There are two advantages to doing so. First, your program will be more readable. Second, it's easier to maintain your program. For instance, if the value of the constant needs to be changed, you just find the statement that associates the constant with the symbolic name and replace the constant with the new one. Without using the symbolic name, you have to look everywhere
- #define PI 3.14

Directive & executable instructions

- Executable instructions in c : if - while - for , ...

- Arithmetic operation like : variable = variable + 1 ; Variable ++ ;

- Logic operation : (AND &) – (OR |) – (NOT ~) – (XOR ^)

- Directive instructions : Not get into the form of the code , but used to guide the compiler to do something , for example the GCC compiler knew all ANSI-C instructions , but don't know the delay_ms ( ) function , so we guide the compiler to the place which contain the delay functions which we will need :

#include <delay.h> . Any line start with # hash , what is following is preprocessor

- #include < avr/delay.h> when the file is within the paths of the compiler
- #include "avr_config.h" when the file is within the source file itself
- #define PORT_ON PORTD=0xff
- #define PORT_OFF PORTD=0x00

- We must know before the compilation process (converting the file to hex) , the c preprocessor replace all macros or preprocessors with its value .

How to make your own config header file ?

- To minimize the development time there are some libraries we always need to call in most project , some important preprocessor used in configuration we must write .
- so we make a header file called (avr_config.h) which contains all of these and more ,

  also you can add some features to fit your app

```
#ifndef LIBRARY_NAME_H
#define LIBRARY_NAME_H
// some important macros : processor speed (F_CPU  1000000)
//calling important libraries : #include <avr\io.h>
//typedef to write code faster and easier
#endif
```

- typedef : The C programming language provides a keyword called **typedef**, which you can use to give a type, a new name :

- typedef unsigned char uint8 ;
- typedef char sint8 ;
- typedef unsigned int uint16;
- typedef unsigned long int uint32;

Configuration Header file

avr_config.h

```
/*
   Avr Configuration Header file and important librarries
   needed in all codes
   by : Mohammed hemed
*/


#ifndef AVR_CONFIG_H_
#define AVR_CONFIG_H_

/* CPU frequency = 1MHz 1MIPS */
#ifndef F_CPU
#define F_CPU 1000000
#endif

/* important libraries embedded in avr compiler */
#include <avr\io.h>
#include <avr\interrupt.h>
#include <util\delay.h>

/* data types shortcuts */
typedef signed char     sint8 ;
typedef signed short    sint16;
typedef signed long     sint32;
typedef unsigned char   uint8 ;
typedef unsigned short  uint16 ;
typedef unsigned long   uint32;



#endif
```

How to make your own library ?

- C libraries consist of two type of files :

1- Definition file or header file and has (file.h) extention.

2- Implementation file : consist of the actual code and functions of the library.

```
/* Motors interfacing library */
/* this library work with
   - DC motors
   - SG90 9g micro servo
   by : Mohammed hemed
*/


/* library name */
#ifndef MOTORS_H_
#define MOTORS_H_

/* define constants of the library */

#define motorPort   PORTD
#define DcMotor1Pin1 PD0
#define DcMotor1Pin2 PD1
#define DcMotor2Pin1 PD2
#define DcMotor2Pin2 PD3
#define servoSignal PD2

/* declare function prototypes */

void DCrotateClkwise (sint8 motorNum);
void DCrotateAntiClkwise (sint8 motorNum);
void DCstop (sint8 motorNum);
void setServoPosition (uint8 position);




#endif
```

```
#ifndef _LIBRARYNAME_H_
#define _LIBRARYNAME_H_

        هنا تكتب جميع التعريفات للدوال والمتغيرات المختلفة


#endif
```

How to make your own library ?

References :

- Books :
- Simply AVR - > Abdallah Ali
- The AVR microcontroller & Embedded Systems using Assembly & c -- > Mazidi
- ATMEGA 32A Datasheet
- PIC microcontroller -- > Milan Verle
- Websites :
- https://www.sparkfun.com
- http://maxembedded.com
- https://www.tutorialspoint.com/cprogramming
- https://stackoverflow.com
- https://www.quora.com
- https://www.lucidchart.com

Any questions ?

- Instructor : Mohammed Hemed

- Embedded Systems developer at fab lab Ismailia

Repository link of Embedded workshop Material:

https://github.com/FabLab-Ismailia/Embedded-Systems-Workshop

Contact me :

- Gmail :

mohammedhemed23@gmail.com

- Linkedin :

https://www.linkedin.com/in/mohammedhemed

See you „„