MTE201 C Programming Basics

COURSE SYNOPSES

C Programming: Introductory concepts, C fundamentals, operators and expression, data input and output, preparing and running a complete C program, control statements, functions, program structure, arrays, pointers, structures and unions, data files and low level programming. Advanced C Programming: Control statements, functions, program structure, arrays, pointers, structures and unions, data files and low level programming.

1.0. History of *C*

The C programming language was pioneered by Dennis Ritchie at AT&T Bell Laboratories in the early 1972. The language was formalized in 1988 by the American National Standard Institute (ANSI). It was not until the late 1970s, houser ver, that this programming language began to gain widespread popularity and support. This was because until that time C compilers user re not readily available for commercial use outside of Bell Laboratories. C is a general-purpose programming language, and is used for writing programs in many different domains, such as operating systems, numerical computing, graphical applications, etc. It is a small language, with just 32 keywords. It provides "high-level" structured programming constructs such as statement grouping, decision making, and looping, as user ll as "low level" capabilities such as the ability to manipulate bytes and addresses.

1.1. Programming

Computers are really very dumb machines indeed because they do as instruct by the user. To solve a problem using a computer, you must express the solution to the problem in terms of the instructions of the particular computer. A computer program is just a collection of the instructions necessary to solve a specific problem. The approach or method that is used to solve the problem is known as an algorithm. Normally, to develop a program to solve a particular problem, you first express the solution to the problem in terms of an algorithm and then develop a program that implements that algorithm. So, the algorithm for solving the even/odd problem might be expressed as follows: First, divide the number by two. If the remainder of the division is zero, the number is even; otherwise, the number is odd. With the algorithm in hand, you can then proceed to write the instructions necessary to implement the algorithm on a particular computer system. These instructions would be expressed in the statements of a particular computer language, such as Visual Basic, Java, C++, or C.

1.2. Operating Systems

An operating system is a program that controls the entire operation of a computer system. All input and output (that is, I/O) operations that are performed on a computer system are channeled through the operating system. One of the most popular operating systems today is the Unix operating system, which was developed at Bell Laboratories. Unix is a rather unique operating system in that it can be found on many different types of computer systems, and in different "flavors," such as Linux or Mac OS X.

1.3. The C Compiler

A compiler analyzes a program developed in a particular computer language and then translates it into a form that is suitable for execution on your particular computer system. The source code written in source file is the human readable source for your program. It needs to be "compiled" into machine language so that your CPU can actually execute the program as per the instructions given. The compiler compiles the source codes into final executable programs. The most frequently used and free available compiler is the GNU C/C++ compiler, otherwise you can have compilers either from HP or Solaris if you have the respective operating systems.

1.4. Integrated Development Environments (IDE)

This process of editing, compiling, running, and debugging programs is often managed by a single integrated application known as an Integrated Development Environment, or IDE for short. An IDE is a windows-based program that allows you to easily manage large software programs, edit files in windows, and compile, link, run, and debug your programs.

1.5. Touring the Code::Blocks workspace

Figure 1 illustrates the Code::Blocks workspace, which is the official name of the massive mosaic of windows you see on the screen. The details in Figure 1 are rather small, but what you need to find are the main areas, which are called out in the figure:

1. Toolbars: These messy strips, adorned with various command buttons, cling to the top of the Code::Blocks window. There are eight toolbars, which you can rearrange, show, or hide. Don’t mess with them until you get comfy with the interface.
2. Management: The window on the left side of the workspace features four tabs, though you may not see all four at one time. The window provides a handy oversight of your programming endeavors.
3. Status bar: At the bottom of the screen, you see information about the project and editor and about other activities that take place in Code::Blocks .
4. Editor: The big window in the center-right area of the screen is where you type code.

5. Logs: The bottom of the screen features a window with many, many tabs. Each tab displays information about your programming projects. The tab you use most often is named Build Log.

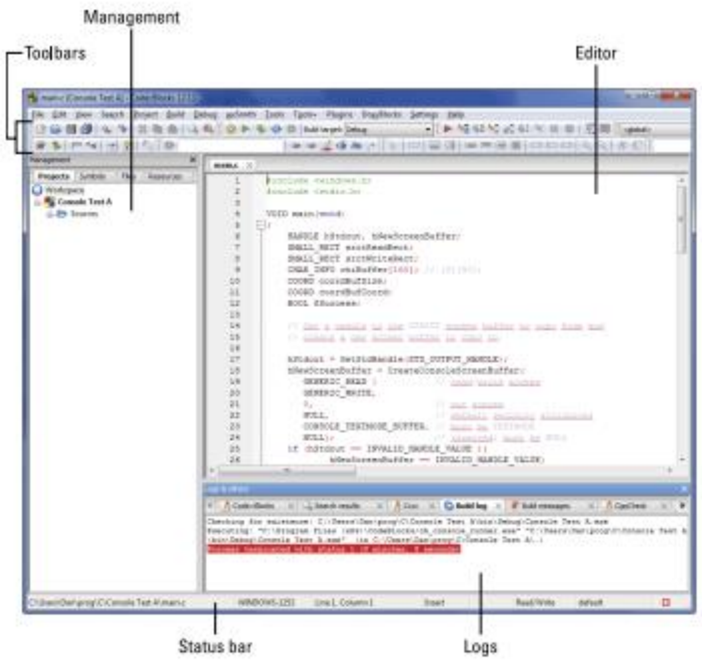
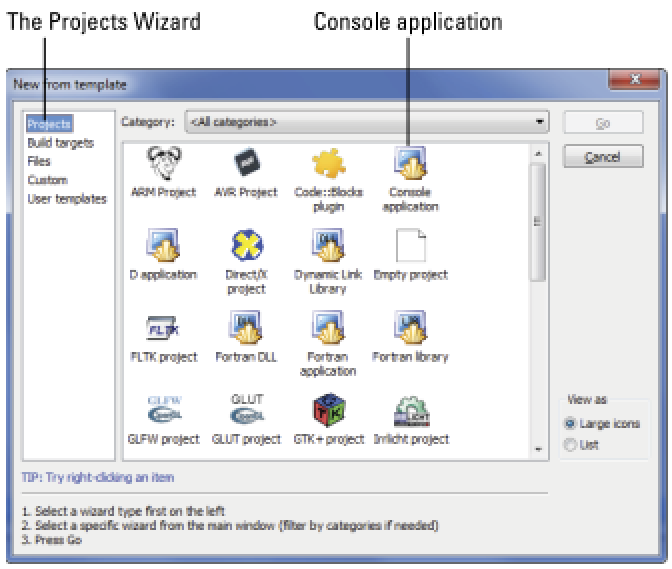


Figure 1 The Code::Blocks workspace.

1.6. Creating a new project

1. Start Code::Blocks . You see the Start Here screen, which displays the Code::Blocks logo and a few links. If you don't see the Start Here screen, choose File⇨Close workspace.
2. Click the Create a New Project link. The New from Template dialog box appears, as shown in Figure 2.
3. Choose Console Application and then click the Go button. The Console Application Wizard appears. You can place a check mark by the item Skip This Page Next Time to skip over the wizard's first screen.
4. Click the Next button.
5. Choose C as the language you want to use, and then click the Next button. C is quite different from C++ — you can do things in one language that aren't allouser d in the other.



TIP: Try right-clicking an item

1. Select a wizard type first on the left
2. Select a specific wizard from the main window (filter by categories if needed)
3. Press Go

Figure 2 New projects

6. Type MTE201_01 as the project title. When you set the project title, the project's filename is automatically filled in.
7. Click the … (Browse) button to the right of the text box titled Folder to Create Project In. I recommend that you create and use a special folder for all projects in this book.
8. Use the Make New Folder button in the Browse for Folder dialog box to create a project folder.
9. Click the OK button to select the folder and close the dialog box.
10. Click the Next button. The next screen (the last one) allows you to select a compiler and choose whether to create Debug or Release versions of your code, or both. The compiler selection is fine; the GNU GCC Compiler (or whatever is shown in the window) is the one you want.
11. Remove the check mark by Create Debug Configuration. You create this configuration only when you need to debug, or fix, a programming predicament that puzzles you.
12. Click the Finish button.

1.7. Features of C language

1. High level language: it is written in user understandable language making it user friendly and easy to comprehend.
2. Structured language: it improves clarity, quality and it reduces the develop time for designing a programming software

3. Rich library: it has its own library which includes most of the arithmetic and logic operation which are predefined. The user only includes the needed library in the code and their functionality can be executed without having to code them separately.
4. Extensibility: programs written in C language are highly extensible.
5. Recursion: this prevent the writing of the same function multiply times. Instead whenever the user needs the function user just have to call it. This help to reduce the time involved in the development cycle and also improves the code functionality.
6. Pointers: using the pointers user can directly interact with the physical memory of the computer system.
7. Faster execution: program execution is faster in C language than its predecessors.
8. Memory management: C language offers many functions where user can dynamically and directly interact with the memory of the computer system.

1.7.1. C-Tokens

Figure 3 presents the C-tokens in C programming language.

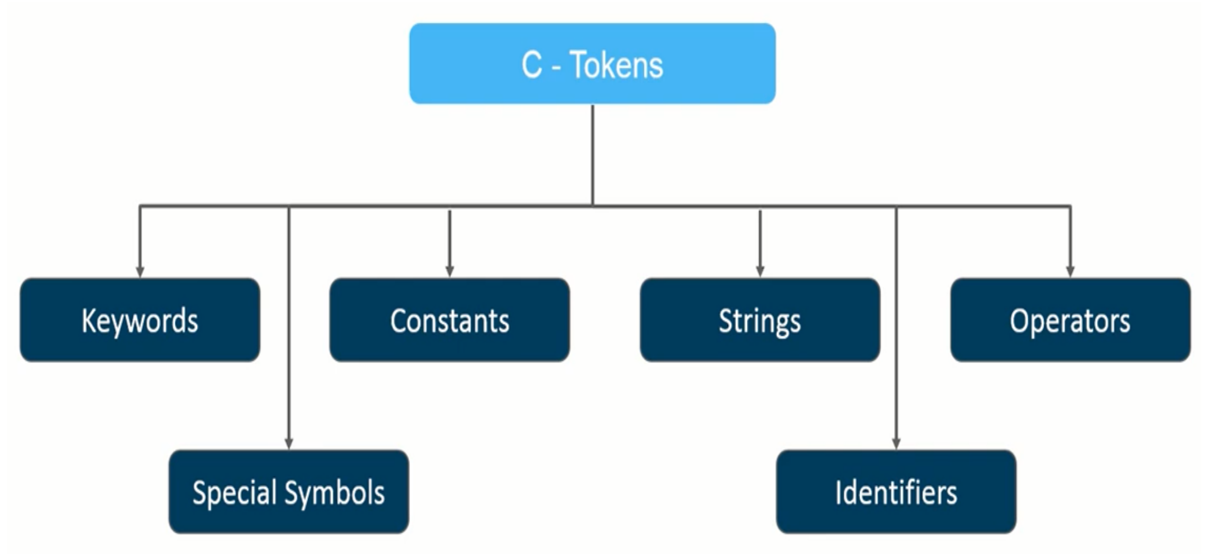
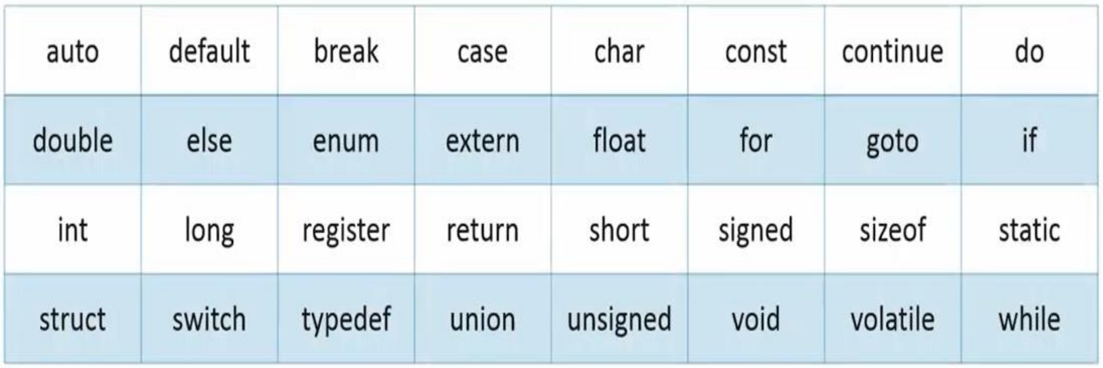


Figure 3 C-Tokens

Keywords: variables having specified meaning and are predefined in C library. Such as main, for, if, else etc. Keywords cannot be renamed or reprogramed. Figure 4 shows the 32 keywords in C language.



auto	default	break	case	char	const	continue	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

Figure 4 The 32 keywords in C language

Constants: sometimes refer to as literals are like variables, but unlike variables once they are declared their values cannot be changed. The syntax is:

const data_type variable_name;

Or

const data_type*variable_name;

Types of Constants in C language

1. Integer constants
2. Floating point constants
3. Character constants
4. String constants
5. Octal and Hexadecimal constants

Strings: these are collection of characters defined in form of an array and end with null character which describes the end of the string to the compiler. The syntax is:

char string_name[Length_of_the_string]

In strings we have alphabetical data meaning (A - Z) which is stored in form of arrays.

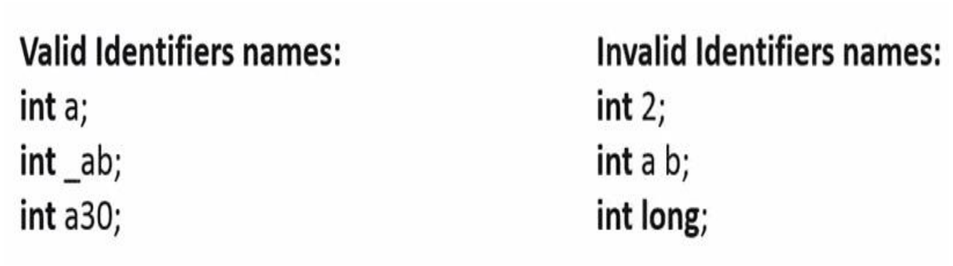
Identifiers: this are names declared in the program in order to name a value, variable function, array etc. The syntax is:

```
int x = 10;
```

In the above, x is the identifier and the value stored is 10. The keyword is int; this is the data type specified for the identifier x. meaning the value is an integer.

Rules for declaring an identifier

1. First character should be an alphabet or underscore.
2. Succeeding character can be digits or letters.
3. Special characters are not allowed except underscore.
4. Identifiers should not be keyword.

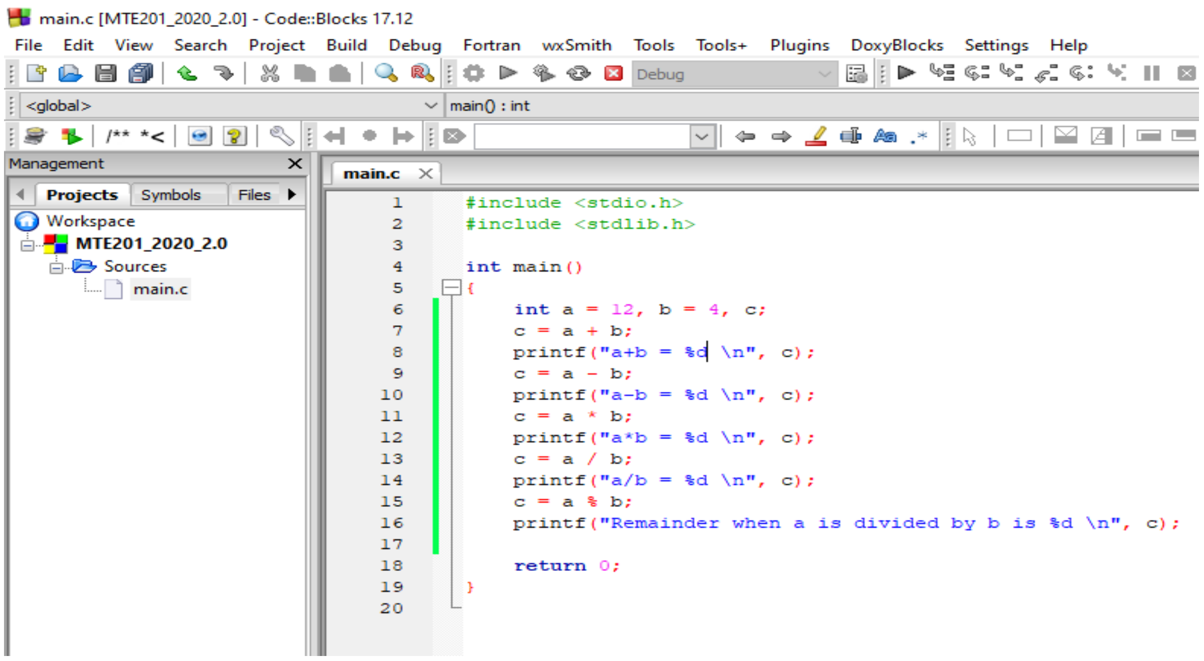


Valid Identifiers names:	**Invalid Identifiers names:**
int a;	**int** 2;
int _ab;	**int** a b;
int a30;	**int long**;

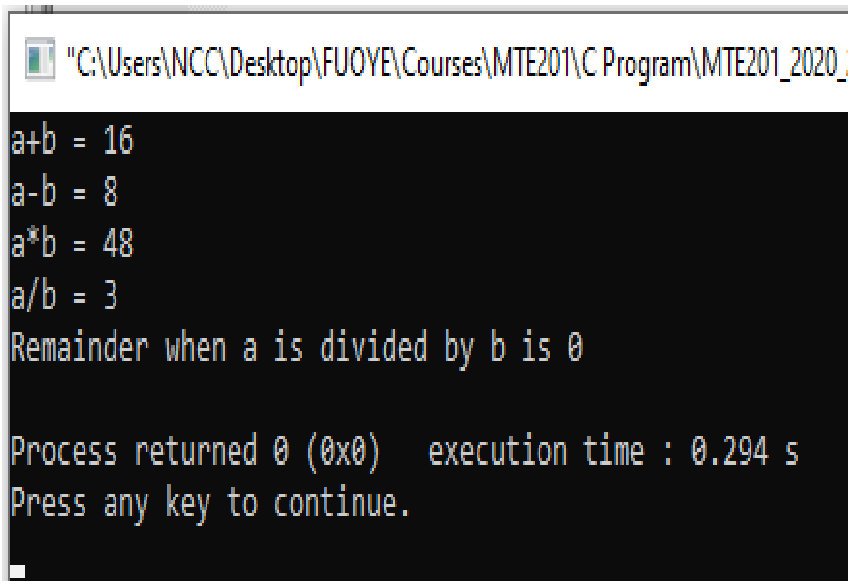
Special Symbols or Characters: this can be single character or sequence of characters having a special built-in meaning in language and typically cannot be used as identifiers. Such as: &, %, # etc. These special characters have meanings which are predefined in C library when designing the language, these is why they are used in particular segment of the code. For instance, the & is only used in printif and scanner statement, the % is used to specify the data type (integer data type %d, spring data type %S)

Operators: The following are the operators in C language

1. Arithmetic operators: these are used to perform mathematical operations. Such as addition, subtraction, multiplication and modulus.


main.c [MTE201_2020_2.0] - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Debug
<global>
main() : int
Management
Projects Symbols Files
Workspace
MTE201_2020_2.0
Sources
main.c
main.c

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int a = 12, b = 4, c;
    c = a + b;
    printf("a+b = %d \n", c);
    c = a - b;
    printf("a-b = %d \n", c);
    c = a * b;
    printf("a*b = %d \n", c);
    c = a / b;
    printf("a/b = %d \n", c);
    c = a % b;
    printf("Remainder when a is divided by b is %d \n", c);

    return 0;
}
```


"C:\Users\NCC\Desktop\FUOYE\Courses\MTE201\C Program\MTE201_2020_

```
a+b = 16
a-b = 8
a*b = 48
a/b = 3
Remainder when a is divided by b is 0

Process returned 0 (0x0)   execution time : 0.294 s
Press any key to continue.
```

2. Increment/decrement operators: These operators are used when loops are included in the program. Increment operators are use to increase the value of a variable by a specific number. Decrement operators are use to decrease the value of a variable by a specific number.

Example:
i++; // increment
i--; // decrement

3. Assignment operators: These operators are used to assign values to variable in C language.
   Example
   =, ==
   int b = 4;
4. Bitwise operators: These operators are used to perform bit operations. Decimal values are converted into binary values which are the sequence of bits and bit wise operators work on these bits.
   Bit wise operators in C language are:
   & (bitwise AND)
   | (bitwise OR)
   ~ (bitwise NOT)
   ^ (XOR)
   << (left shift)
   >> (right shift)

5. Relational operators: These operators are used to test or define relation between two entities or variables.
   Example
   <, >, =, !=,

   If (a=<b);
6. Logical operators: These operators are used to perform logical operation on a given expressions. There are three logical operators in C language:
   Logic AND (&&)
   Logic OR (||)
   Logic NOT (!)

1.8. Datatype and Variable Used in C Programming Language

1.8.1. Datatype

There are four datatypes in C language:

1. Basic datatype
2. Derived datatype

3. Enumeration datatype
4. Void datatype

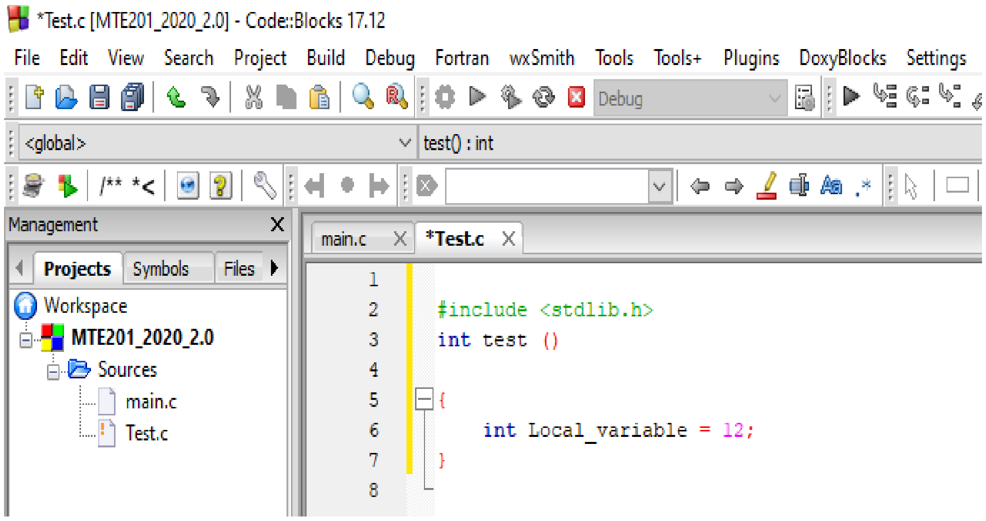
Table 1 presents these datatype examples

Type	Datatype
Basic Datatype	int, char, float, double
Derived Datatype	array, pointer, structure, union
Enumeration Datatype	enum
Void Datatype	void

1.8.2. Variable

Variables are defined as the reserved memory space which stores value of a definite datatype. The value of the variable is not constant, so it can be changed. The types of variables in C language are:

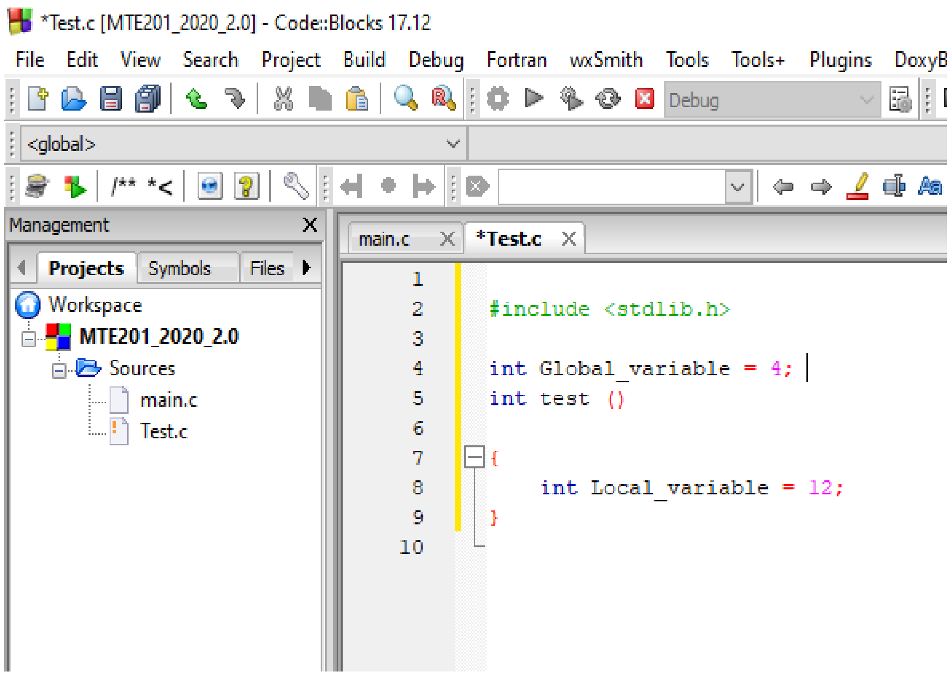
1. Local variable: these variables are declared inside a code block or a function and has it scope limited to that particular block of code or function.


*Test.c [MTE201_2020_2.0] - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings
Debug
<global>
test() : int
Management
Projects Symbols Files
Workspace
MTE201_2020_2.0
Sources
main.c
Test.c
main.c
*Test.c

```

#include <stdlib.h>
int test ()

{
    int Local_variable = 12;
}

```

In the figure above, the function test is defined and a local variable with int datatype is declare within the function.

2. Global variable: these variables are declared outside a code block or a function and has it scope across the entire program and allow any function to change it value.

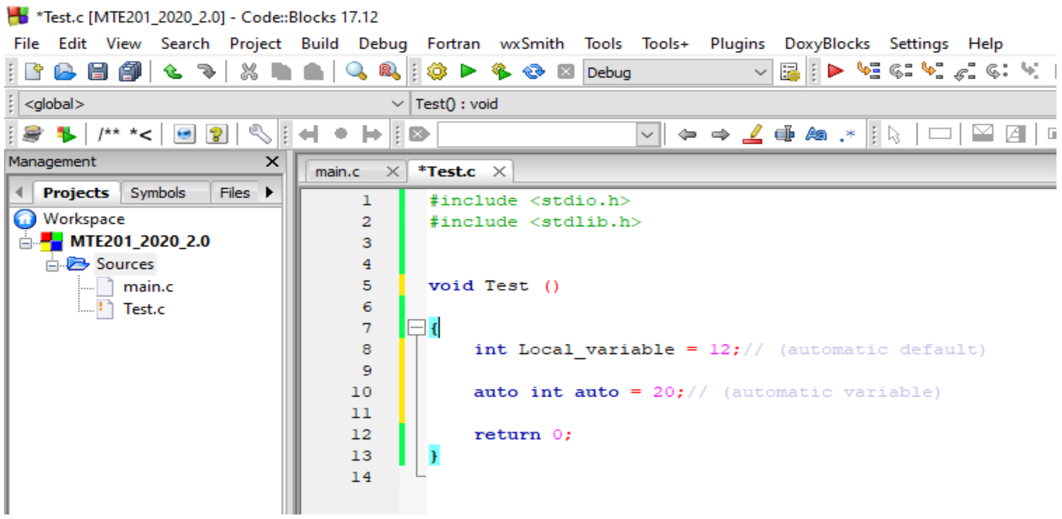

*Test.c [MTE201_2020_2.0] - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyB
Debug
<global>
Management
Projects Symbols Files
Workspace
MTE201_2020_2.0
Sources
main.c
Test.c
main.c
*Test.c

```c

#include <stdlib.h>

int Global_variable = 4;
int test ()

{
    int Local_variable = 12;
}

```

In the figure above, the function test is defined and a global variable with int datatype is declare outside the function.

3. Static variable: any variable declared with the keyword static is known as static variable. These variables retain their declared value throughout the entire execution of the program and will not be changed between multiple function calls.
4. External variable: These variables are declared by using the keyword extern. A variable can be share between multiple C source file by using external variable.
   Example

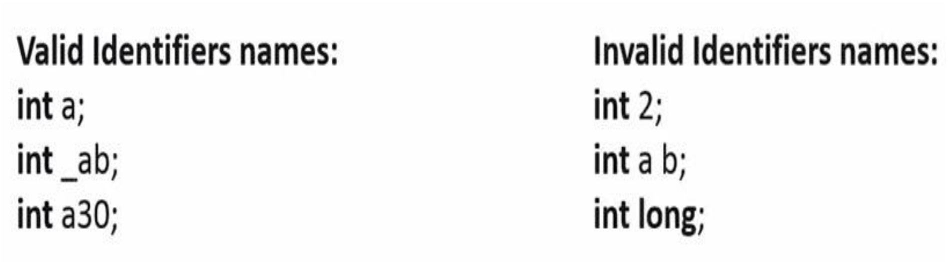
   extern int extern = 10;//(External Variable)
5. Automatic variable: These variables can be declared by using the keyword auto. By default, all the variables define in C language are Automatic Variable.


*Test.c [MTE201_2020_2.0] - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Debug
<global>
Test() : void
Management
Projects Symbols Files
Workspace
MTE201_2020_2.0
Sources
main.c
Test.c
main.c
*Test.c

```c
#include <stdio.h>
#include <stdlib.h>


void Test  ()

{

    int Local_variable = 12;// (automatic default)

    auto int auto = 20;// (automatic variable)

    return 0;
}
```

Rules for declaring a Variable

1. A variable can have alphabet, digit and underscore.
2. A variable name can start with alphabet and underscore only.
3. No spacing is allowed within the variable name.
4. A variable name must not be any reserve word or keyword.

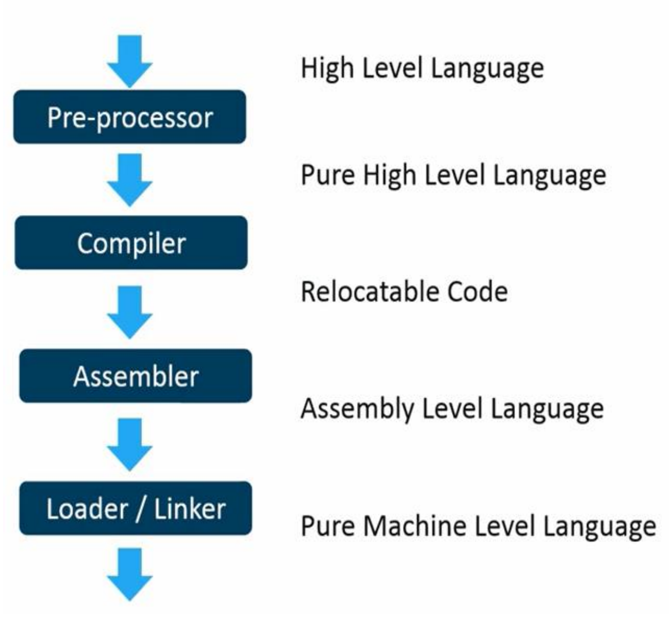


Valid Identifiers names:	Invalid Identifiers names:
int a;	int 2;
int _ab;	int a b;
int a30;	int long;

2.0. PREPROCESSOR DIRECTIVES

These are lines included in a program that begin with the character #, which distinguish them from a typical source code text. They are invoked by the complier to process some programs before compilation. It is a macro processor used automatically by the C compiler to transform user program before actual compilation. For instance, stdio.h, means standard input/output; this preprocessor directive activates the input and output unit to perform a C programming operation.

The operating sequence of the preprocessor directive is shown below:



Types of Preprocessor Directives in C Language

1. #include: this used to paste code of given file into current file. It is used include system defined and user defined header files. There are two variants to use the #include directive:
   #include <filename>
   #include “filename”
2. #define: this is used to define constant or macro substitution. It can use any basic datatype. Syntax is #define token value.
3. #undef: is used to undefine the constant or macro defined by #define. The syntax is #undef token
4. #ifdef
5. #ifndef
6. #else
7. #error

8. #pragma

2.1. Control statements

These statements enable user to specify the flow of program control. They specify the order in which the instruction in a program must be executed. They make it possible to make decisions, to perform tasks repeatedly or jump from one section of the code to another.

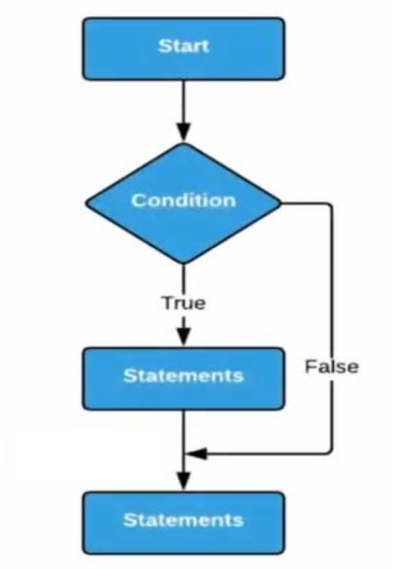
The variants of the control statements in C language are:

1. If statement
2. If-else statement
3. If-else ladder
4. Nested if
5. Switch
6. Ternary
7. Break

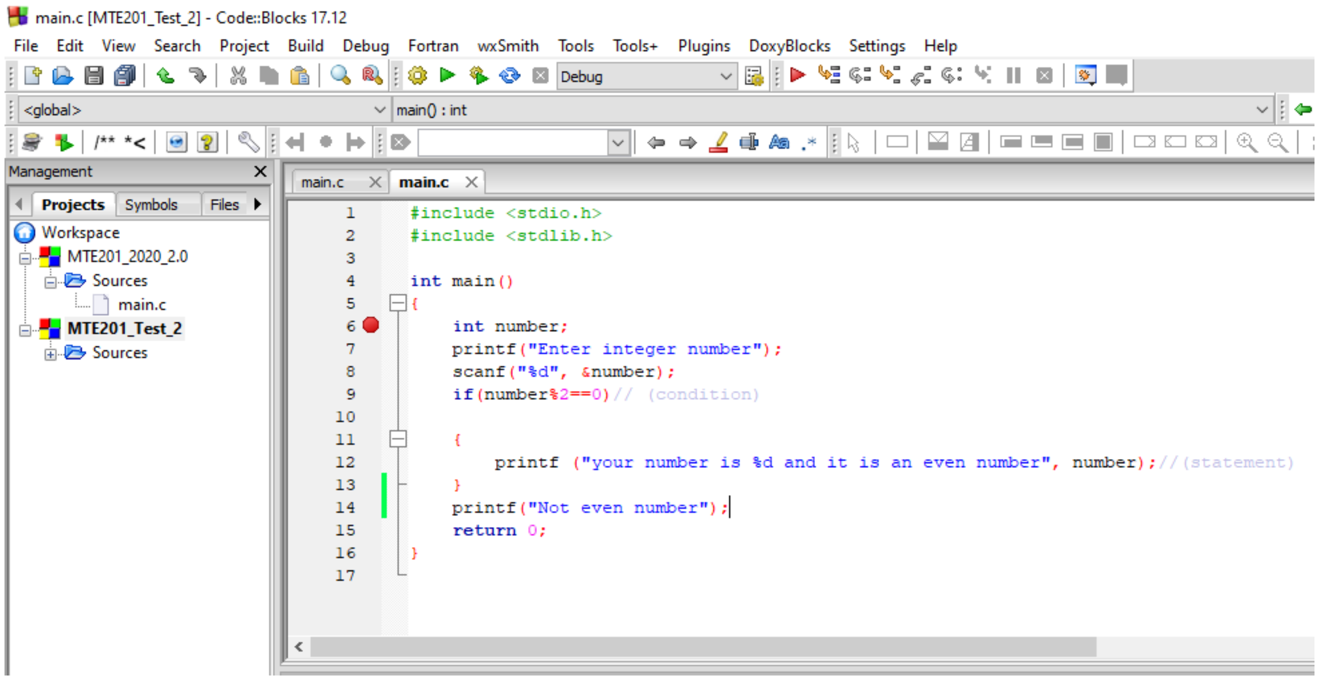
2.1.1. If control statement

The if statement in C language is defined as a programming conditional statement that, if proved true, it performs an operation or displays information inside the statement block.

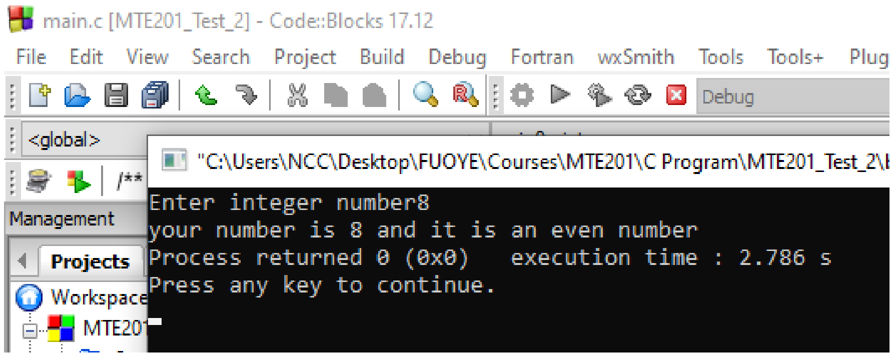
This can be explained using the flow chart



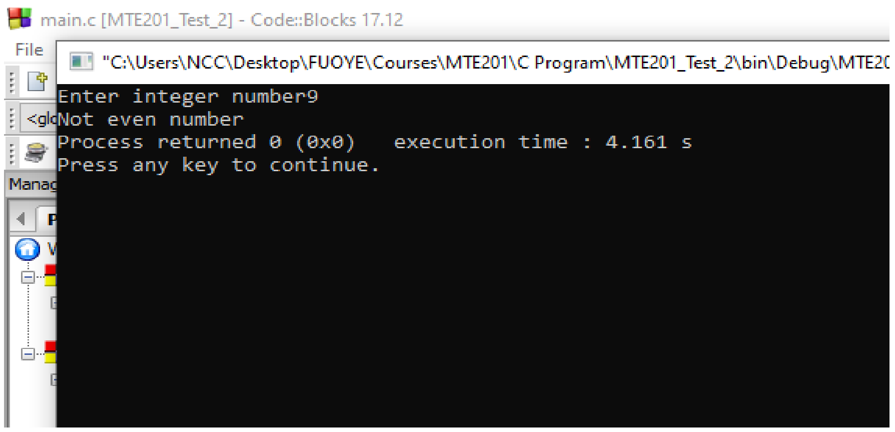
For better understanding let write the code:



```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int number;
    printf("Enter integer number");
    scanf("%d", &number);
    if(number%2==0)//  (condition)

    {
        printf ("your number is %d and it is an even number", number);//(statement)
    }
    printf("Not even number");
    return 0;
}
```



```
"C:\Users\NCC\Desktop\FUOYE\Courses\MTE201\C Program\MTE201_Test_2\
Enter integer number8
your number is 8 and it is an even number
Process returned 0 (0x0)   execution time : 2.786 s
Press any key to continue.
```

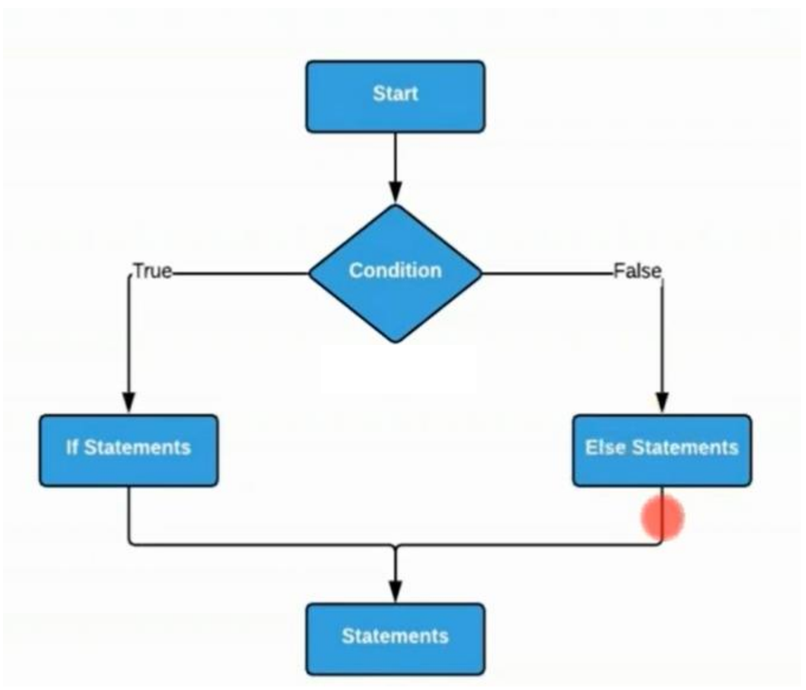


```
"C:\Users\NCC\Desktop\FUOYE\Courses\MTE201\C Program\MTE201_Test_2\bin\Debug\MTE20
Enter integer number9
Not even number
Process returned 0 (0x0)   execution time : 4.161 s
Press any key to continue.
```

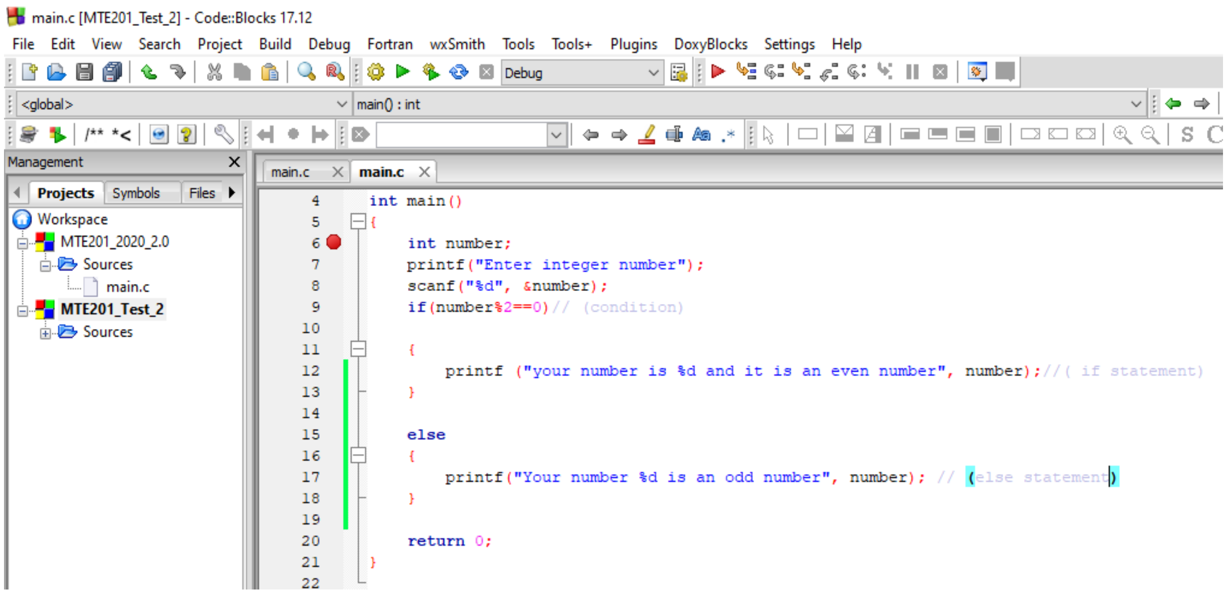
2.1.2. If-else control statement

This control statement defines a programming conditional statement that has two statement blocks over a condition. If proved true, then the **if** block is executed and if false, then the **else** block is executed.

This can be explained using the flow chart

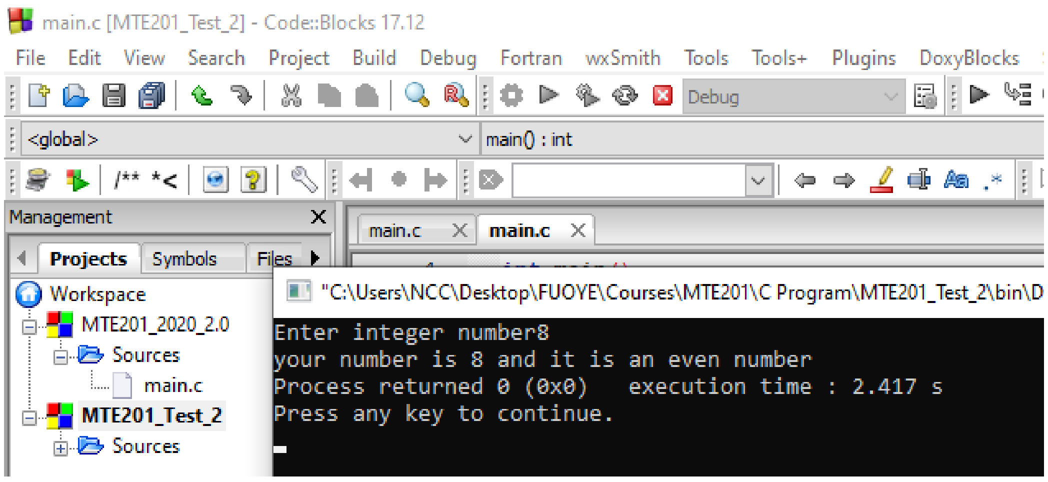


For better understanding let write the code:

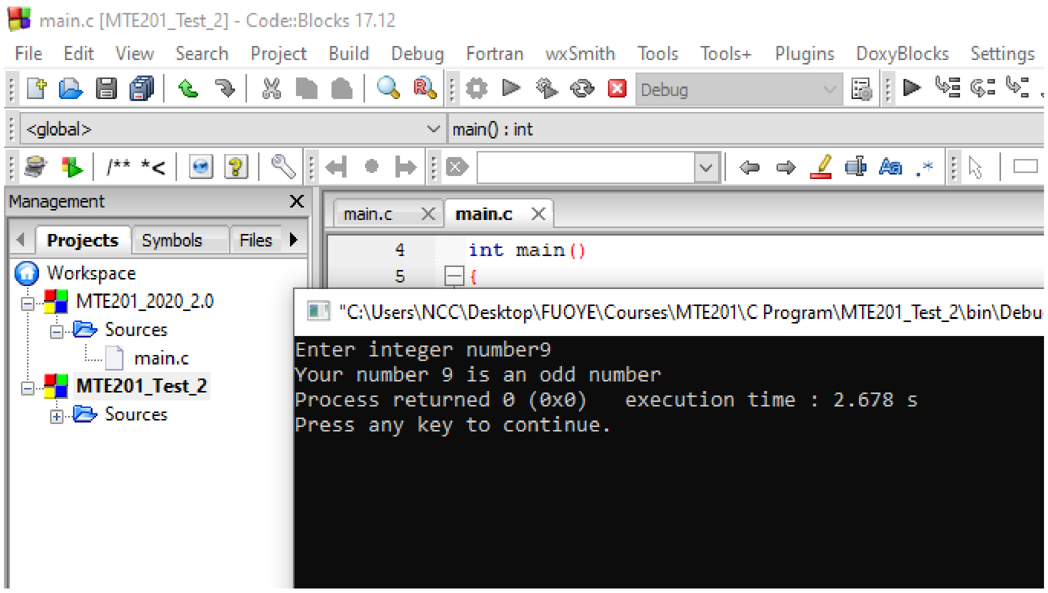


main.c [MTE201_Test_2] - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Debug
<global>
main() : int
Management
Projects Symbols Files
Workspace
MTE201_2020_2.0
Sources
main.c
MTE201_Test_2
Sources
main.c
main.c

```c
int main()
{
    int number;
    printf("Enter integer number");
    scanf("%d", &number);
    if(number%2==0)//  (condition)

    {
        printf ("your number is %d and it is an even number", number);//( if statement)
    }

    else
    {
        printf("Your number %d is an odd number", number); // (else statement)
    }

    return 0;
}
```


main.c [MTE201_Test_2] - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks
Debug
<global>
main() : int
Management
Projects Symbols Files
Workspace
MTE201_2020_2.0
Sources
main.c
MTE201_Test_2
Sources
main.c main.c
"C:\Users\NCC\Desktop\FUOYE\Courses\MTE201\C Program\MTE201_Test_2\bin\D

```
Enter integer number8
your number is 8 and it is an even number
Process returned 0 (0x0)   execution time : 2.417 s
Press any key to continue.
```


main.c [MTE201_Test_2] - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings
Debug
<global>
main() : int
Management
Projects Symbols Files
Workspace
MTE201_2020_2.0
Sources
main.c
MTE201_Test_2
Sources
main.c main.c
"C:\Users\NCC\Desktop\FUOYE\Courses\MTE201\C Program\MTE201_Test_2\bin\Debu

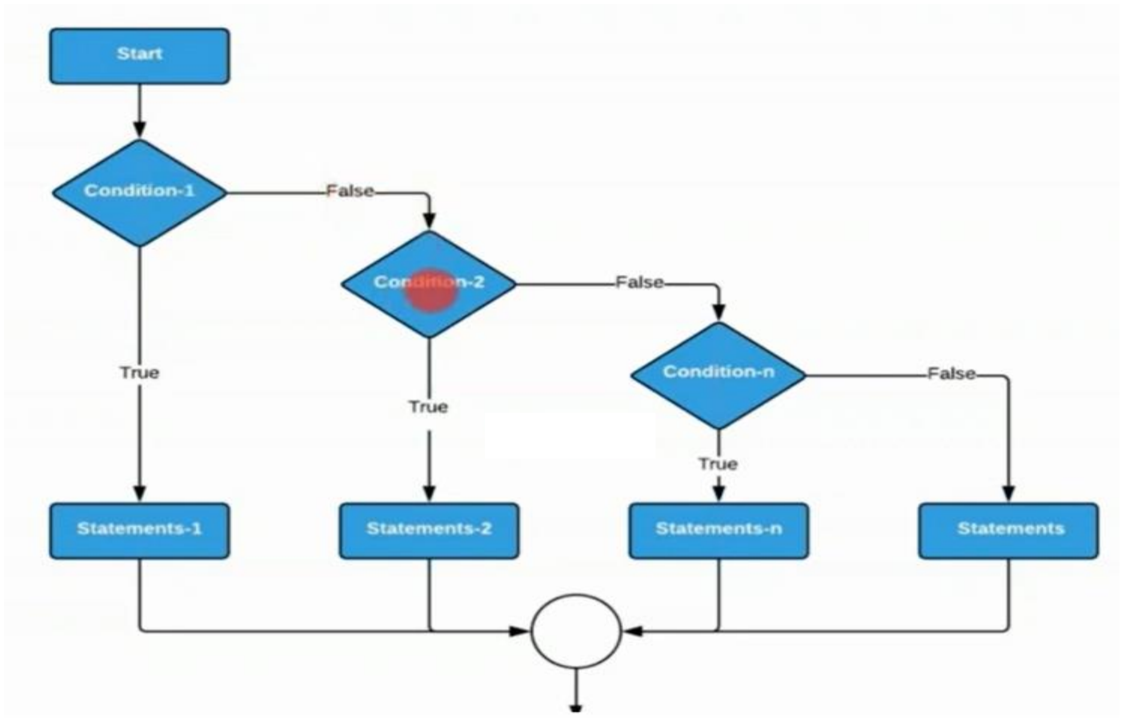
```
int main()
{
```

```
Enter integer number9
Your number 9 is an odd number
Process returned 0 (0x0)   execution time : 2.678 s
Press any key to continue.
```

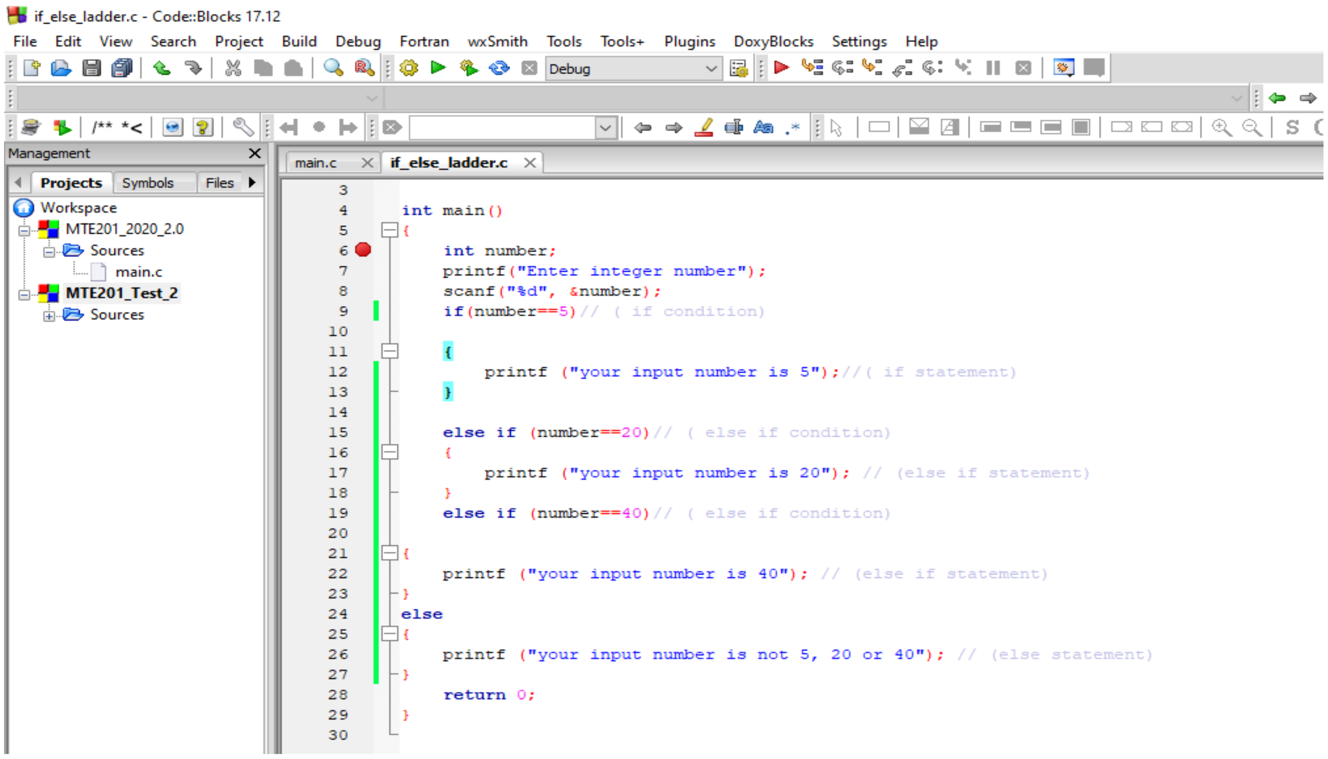
2.1.3. If-else ladder Control Statement

This control statement defines a programming conditional statement that has multiple else-if statement blocks. If any of the condition is true, then the control will exit the else-if ladder and execute the next set of statements.

This can be explained using the flow chart



For better understanding let write the code:


if_else_ladder.c - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Debug
Management
Projects Symbols Files
Workspace
MTE201_2020_2.0
Sources
main.c
MTE201_Test_2
Sources
main.c
if_else_ladder.c

```c

int main()
{
    int number;
    printf("Enter integer number");
    scanf("%d", &number);
    if(number==5)// ( if condition)

    {
        printf ("your input number is 5");//( if statement)
    }

    else if (number==20)// ( else if condition)
    {
        printf ("your input number is 20"); // (else if statement)
    }
    else if (number==40)// ( else if condition)

{

    printf ("your input number is 40"); // (else if statement)
}
else
{
    printf ("your input number is not 5, 20 or 40"); // (else statement)
}
    return 0;
}

```