UNIT-1

1. PROGRAMMING PROCESS
2. OVERVIEW OF C
3. DATA INPUT/OUTPUT

THE PROGRAMMING PROCESS

- All programming involves creating something that solves a problem. The problems can range from something of great scientific or national importance.
- This section describes one approach to solve such problems - think of it as a rough guide to the things you should do when entering the land of programming.
- In broad terms, those things are:
- Identify the Problem
- Design a Solution
- Write the Program
- Check the Solution
- Of these, only the third step is usually called "programming“.

PROBLEM DEFINITION

- To solve a problem, the first step is to identify and define the problem.
- The problem must be stated clearly, accurately and precisely.
- E-x Find largest of three numbers

PROBLEM ANALYSIS

- The problem analysis helps in designing and coding for that particular problem.
- 1. Input specifications The number of inputs and what forms the input are available
- 2.Output specifications The number of outputs and what forms the output should be displayed.
- E-x input – a,b,c output - c

DESIGNING A PROGRAM

- Algorithms
- Flowcharts
- Algorithm - step by step procedure of solving a problem
- Flowcharts – It is the graphical representation of the algorithm.
- Coding- Writing instructions in a particular language to solve a problem.
- Testing- After writing a program, programmer needs to test the program for completeness, correctness, reliability and maintainability.
- Maintaining the program- It means periodic review of the programs and modifications based on user requirements.

ALGORITHM DEVELOPMENT

- An algorithm is a step by step procedure to solve a given problem in finite number of steps. The characteristics of an algorithm are
- (i) Algorithm must have finite number of steps.
- (ii) No instructions should be repeated.
- (iii) An algorithm should be simple.
- (iv) An algorithm must take atleast one or more input values.
- (v) An algorithm must provide atleast one or more output values.

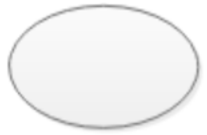
EXAMPLE-1

- Problem definition : To find simple interest
- Problem Analysis :
- inputs– p, r, t
- Output– simple interest
- Algorithm Step 1:Start
- Step 2:input p,r,t
- Step 3: calculate si=p*r*t/100
- Step 4: output si
- Step 5:stop

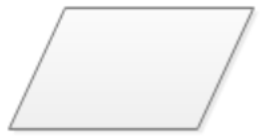
FLOWCHART

A flowchart is the graphical or pictorial representation of an algorithm with the help of different symbols, shapes and arrows in order to demonstrate a process or a program. With algorithms, we can easily understand a program. The main purpose of a flowchart is to analyze different processes. Several standard graphics are applied in a flowchart:

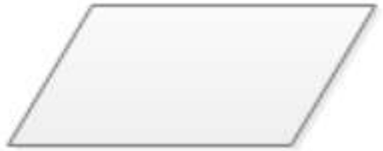
- Terminal Box - Start / End



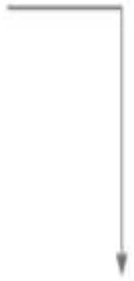
- Input / Output



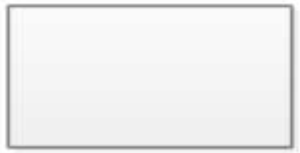
- Input / Output



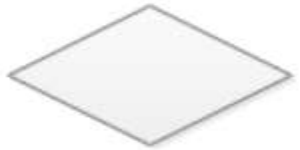
- Connector / Arrow



- Process / Instruction

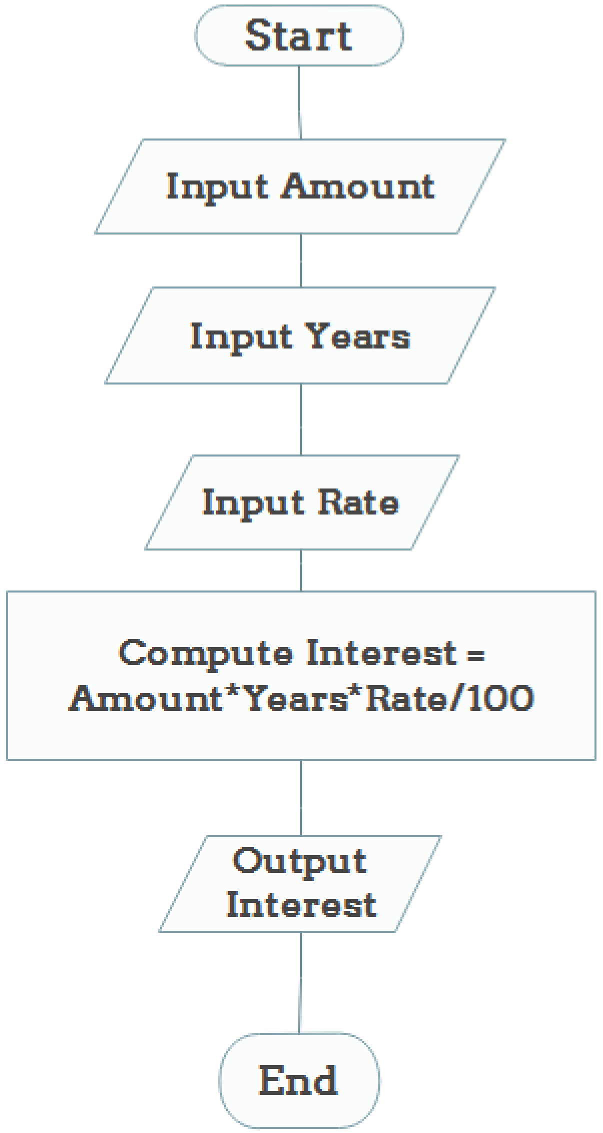


- Decision



The graphics above represent different part of a flowchart. The process in a flowchart can be expressed through boxes and arrows with different sizes and colors. In a flowchart, we can easily highlight a certain element and the relationships between each part.

Example 1: Calculate the Interest of a Bank Deposit
Algorithm:
•Step 1: Read amount,
•Step 2: Read years,
•Step 3: Read rate,
•Step 4: Calculate the interest with formula "Interest=Amount*Years*Rate/100
•Step 5: Print interest,
Flowchart:



Example 2: Convert Temperature from Fahrenheit (°F) to Celsius (℃)

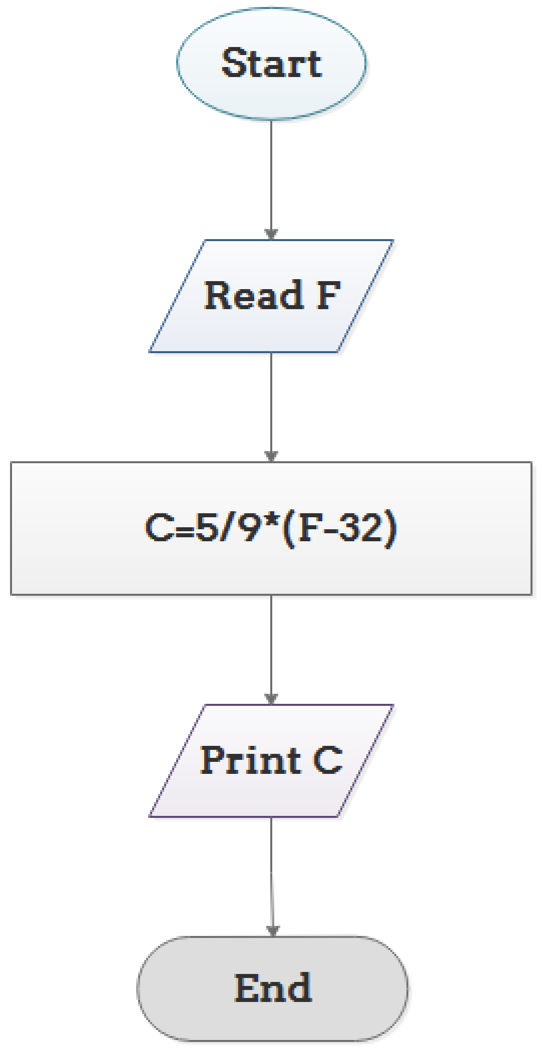
Algorithm:

Step 1: Read temperature in Fahrenheit,

Step 2: Calculate temperature with formula C=5/9*(F-32),

Step 3: Print C,

Flowchart:



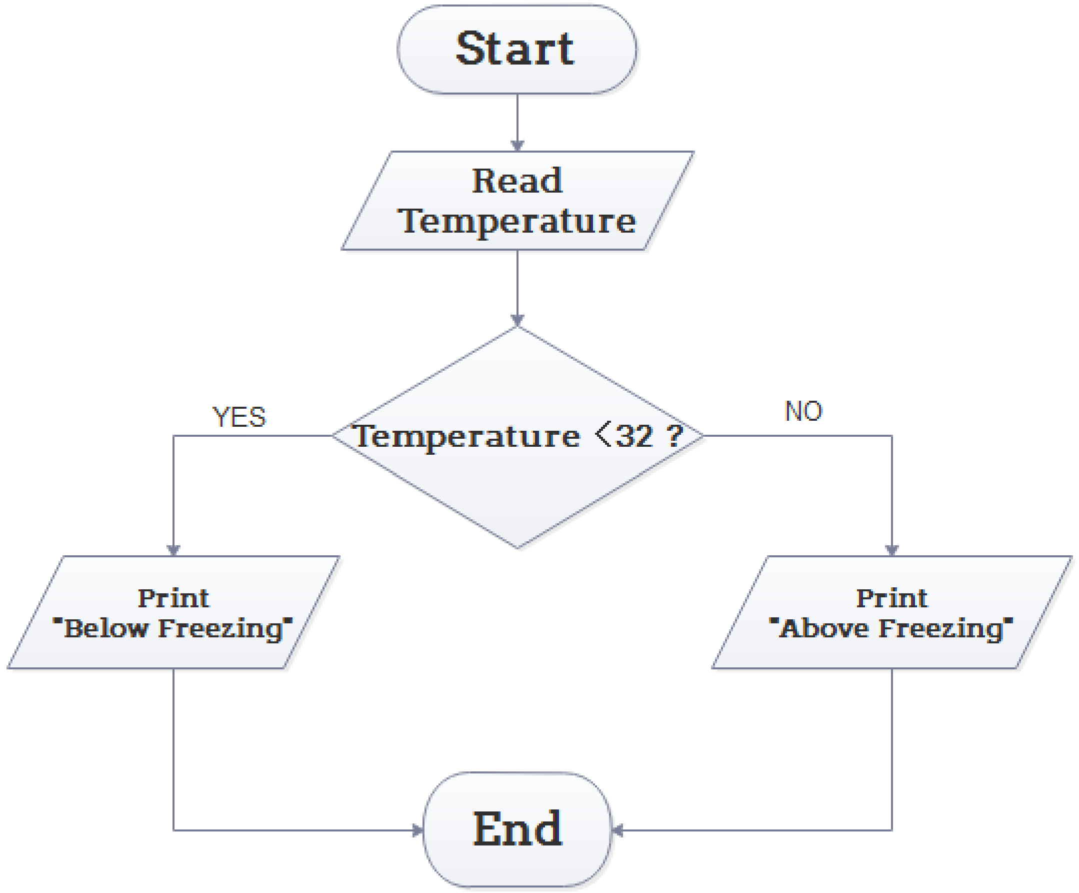
Example 3: Determine Whether a Temperature is Below or Above the Freezing Point

Algorithm:

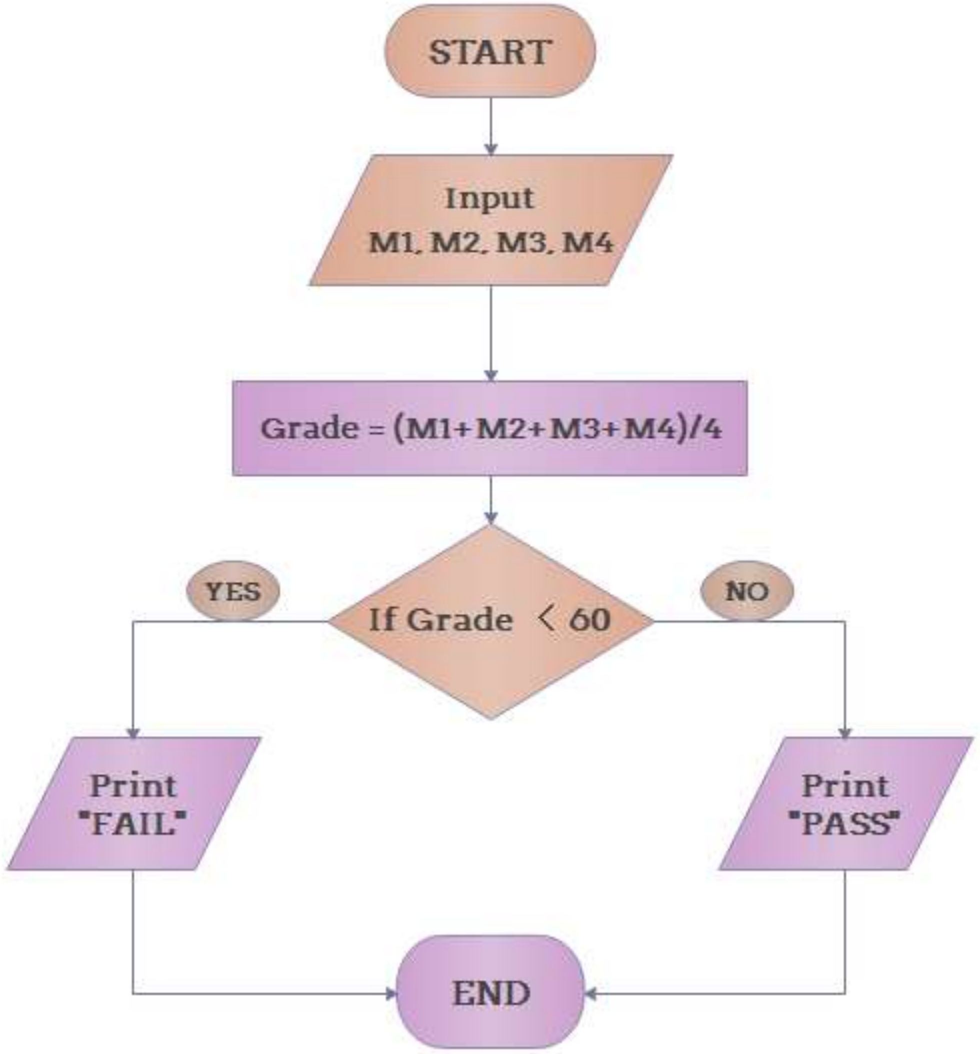
•Step 1: Input temperature,

•Step 2: If it is less than 32, then print "below freezing point", otherwise print "above freezing point"

Flowchart:



Example 4: Determine Whether A Student Passed the Exam or Not:
Algorithm:
•Step 1: Input grades of 4 courses M1, M2, M3 and M4,
•Step 2: Calculate the average grade with formula "Grade=(M1+M2+M3+M4)/4"
•Step 3: If the average grade is less than 60, print "FAIL", else print "PASS".



PROGRAM

- Programming is then the task of describing your design to the computer: teaching it your way of solving the problem.
- There are usually three stages to writing a program:
- Coding
- Compiling
- Debugging

CODING

- Coding is the act of translating the design into an actual program, written in some form of programming language. This is the step where you actually have to sit down at the computer and type!
- Coding is a little bit like writing an essay (but don't let that put you off). In most cases you write your program using something a bit like a word processor. And, like essays, there are certain things that you always need to to include in your program (a bit like titles, contents pages, introductions, references etc.). But we'll come on to them later.
- When you've finished translating your design into a program (usually filling in lots of details in the process) you need to submit it to the computer to see what it makes of it.

COMPILATION

Compilation is actually the process of turning the program written in some programming language into the instructions made up of 0's and 1's that the computer can actually follow. This is necessary because the chip that makes your computer work only understands binary machine code - something that most humans would have a great deal of trouble using since it looks something like:

01110110 01101101 10101111 00110000 00010101

Early programmers actually used to write their programs in that sort of a style - but luckily they soon learnt how to create programs that could take something written in a more understandable language and translate it into this gobbledy gook. These programs are called compilers and you can think of them simply as translators that can read a programming language, translate it and write out the corresponding machine code.

Compilers are notoriously pedantic though - if you don't write very correct programs, they will complain. Think of them as the strictest sort of English teacher, who picks you up on every single missing comma, misplaced apostrophe and grammatical error.

DEBUGGING

This is where debugging makes it first appearance, since once the compiler has looked at your program it is likely to come back to you with a list of mistakes as long as your arm. Don't worry though, as this is perfectly normal - even the most experienced programmers make blunders.

Debugging is simply the task of looking at the original program, identifying the mistakes, correcting the code and recompiling it. This cycle of code -> compile -> debug will often be repeated many many times before the compiler is happy with it. Luckily, the compiler never ever gets cross during this process - the programmer on the other hand...

It should also be said at this point that it isn't actually necessary to write the entire program before you start to compile and debug it. In most cases it is better to write a small section of the code first, get that to work, and then move on to the next stage. This reduces the amount of code that needs to be debugged each time and generally creates a good feeling of "getting there" as each section is completed.

Finally though, the compiler will present you with a program that the computer can run: hopefully, your solution!

OVERVIEW OF C

C is a computer programming language developed in 1972 by **Dennis M. Ritchie** at the Bell Telephone Laboratories to develop the UNIX Operating System. C is a simple and **structure oriented** programming language.
C is also called **mother Language** of all programming Language. It is the most widely use computer programming language, This language is used for develop system software and Operating System. All other programming languages were derived directly or indirectly from C programming concepts.

In the year 1988 'C' programming language standardized by ANSI (American national standard institute), that version is called **ANSI-C**. In the year of 2000 'C' programming Language standardized by 'ISO' that version is called C-99

Where we use C Language

C Language is mainly used for;

•Design Operating system

•Design Language Compiler

•Design Database

•Language Interpreters

•Utilities

•Network Drivers

•Assemblers

NEED OF PROGRAMMING

- Programming language is important because it defines **the relationship, semantics and grammar which allows the programmers to effectively communicate** with the machines that they program.
- A programming language serves several purposes:
- You can instruct the computer what to do in a **human-readable form**
- Allows the programmer to **structure the instructions into functions, procedures**, etc.
- This also allows the program to be **broken into "chunks"** which can be developed by a group of developers
- Provides **portability** - the low-level instructions of one computer will be different from that of another computer.
- Computer programmers have a full understanding of **the how and why of computer systems, including system limitations,** and can set realistic expectations and work around those limitations to fully maximize the use of the equipment and its accessories.
- Computer programming principles implemented today will likely influence how technologies **such as voice-recognition, artificial intelligence** and other sophisticated technologies will change in the future and how they will be applied to our day-to-day lives.
- Programming is important for **speeding up the input and output processes** in a machine. Programming is important to automate, collect, manage, calculate, analyze the processing of data and information accurately.

IMPORTANCE OF C

- C is robust language and has rich set of built-in functions, data types and operators which can be used to write any complex program.
- Program written in C are efficient due to availability of several data types and operators.
- C has the capabilities of an assembly language with the feature of high level language so it is well suited for writing both system software and application software.
- C is highly portable language i.e. code written in one machine can be moved to other which is very important and powerful feature.
- C supports low level features like bit level programming and direct access to memory using pointer which is very useful for managing resource efficiently.
- C has high level constructs and it is more user friendly as its syntaxes approaches to English like language.

- The flexibility of its use for memory management.
- Programmers have opportunities to control **how, when, and where to allocate and deallocate memory.**

Memory is allocated statically, automatically, or dynamically in C programming with the help of malloc and calloc functions.

- Another strong reason of using C programming language is that it **sits close to operating system. This feature makes it an efficient language because system level resources**, such as memory, can be accessed easily.
- C programming language is not limited to but used widely **in operating systems, language compilers, network drivers, language interpreters, and system utilities areas of development.**

STRUCTURE OF C PROGRAM

Every C program can be written with the following syntax.

Syntax

```
#include<headerfilename.h> --> include section
Returntype function_name(list of parameters or no parameter) --> user defined function
{
Set of statements
.........
}
Returntype main() --> main block or main function
{
.........
.........
}
```

Include section
# include is a pre-processor directive can be used to include all the predefined functions of given header files into current C program before compilation.

Syntax
#include<headerfile.h>

C library is collection of header files, header files is a container which is collection of related predefined functions.

User defined function section
If any function is defined by the user is known as user defined function. Function is collection of statement used to perform a specific Operation.

Syntax
Return_type function_Name() { .... // called function .... }
In the above syntax function name can be any user defined name, return type represents which type of value it can return to its calling function.

Syntax
functionName(); // calling function
Note: User defined function are Optional in a C program.

Main function

This is starting executable block of any program (it is always executed by processor and OS). One C program can have maximum one main() the entire statements of given program can be executed through main(). Without main() function no C program will be executed.

Syntax

Returntype main() { .... }

If return type is void that function can not return any value to the operating system. So that void can be treated as no return type.

Example

```
#include<stdio.h>
 #include<conio.h>
 void main()
{
printf("Hello main");
}
```

Output

```
Hello main
```

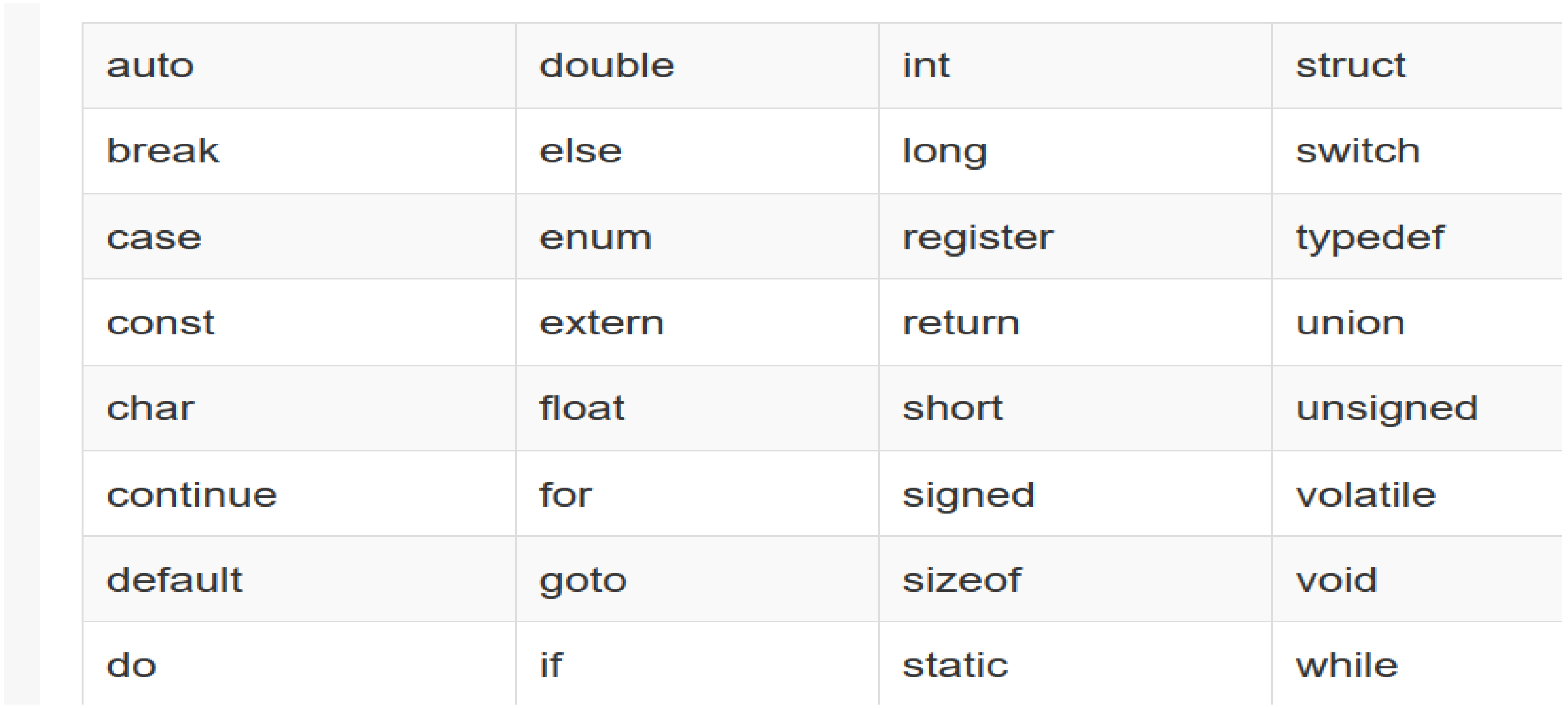
- **IO statements in C language**

IO represents input output statements and input statement can be used to read the input value from the standard input device (keyboard), output statement can be used to display the output in standard output device (Monitor) respectively.

In C language IO statement can be achieve by using scanf() and printf().

What are Keywords in C?

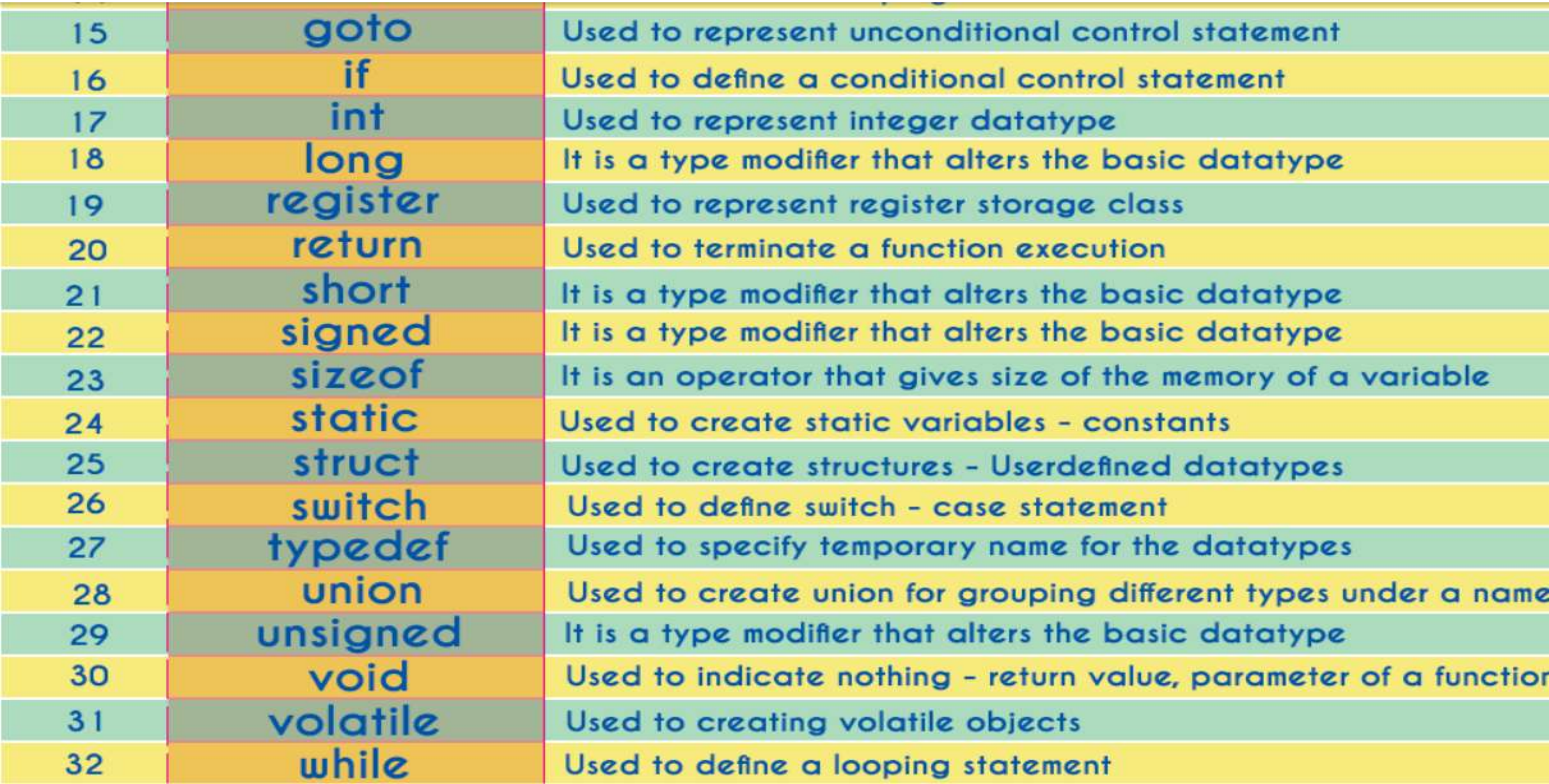
Keywords are preserved words that have special meaning in C language. The meaning of C language keywords has already been described to the C compiler. These meaning cannot be changed. Thus, keywords cannot be used as variable names because that would try to change the existing meaning of the keyword, which is not allowed.(Don't worry if you do not know what variables are, you will soon understand.) There are total 32 keywords in C language.



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

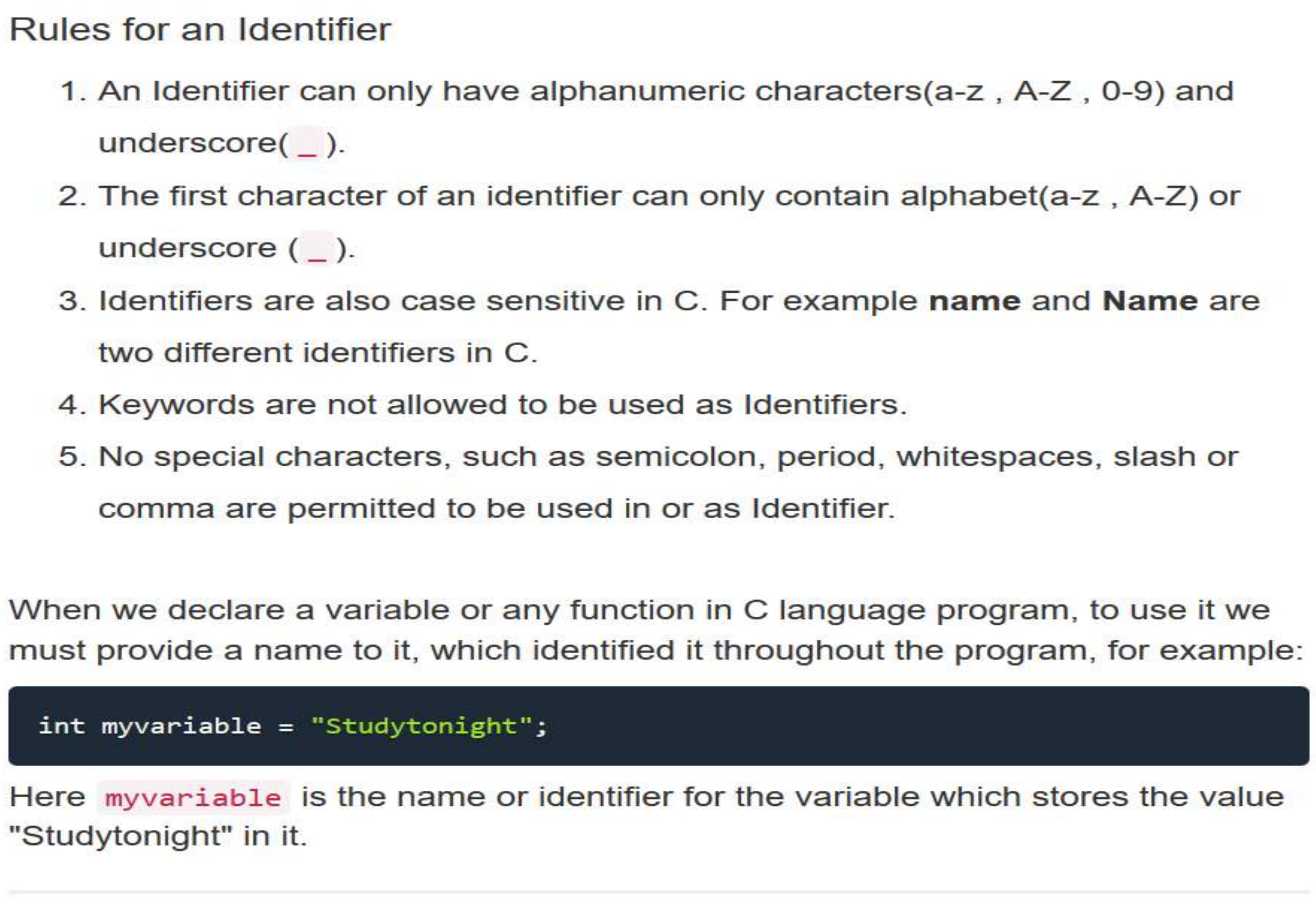
32 Keywords in C Programming Language with their Meaning

S.No	Keyword	Meaning
1	auto	Used to represent automatic storage class
2	break	Unconditional control statement used to terminate swicth & looping statements
3	case	Used to represent a case (option) in switch statement
4	char	Used to represent character data type
5	const	Used to define a constant
6	continue	Unconditional control statement used to pass the control to the begining of looping statements
7	default	Used to represent a default case (option) in switch statement
8	do	Used to define do block in do-while statement
9	double	Used to present double datatype
10	else	Used to define FALSE block of if statement
11	enum	Used to define enumarated datatypes
12	extern	Used to represent external storage class
13	float	Used to represent floating point datatype
14	for	Used to define a looping statement



15	goto	Used to represent unconditional control statement
16	if	Used to define a conditional control statement
17	int	Used to represent integer datatype
18	long	It is a type modifier that alters the basic datatype
19	register	Used to represent register storage class
20	return	Used to terminate a function execution
21	short	It is a type modifier that alters the basic datatype
22	signed	It is a type modifier that alters the basic datatype
23	sizeof	It is an operator that gives size of the memory of a variable
24	static	Used to create static variables - constants
25	struct	Used to create structures - Userdefined datatypes
26	switch	Used to define switch - case statement
27	typedef	Used to specify temporary name for the datatypes
28	union	Used to create union for grouping different types under a name
29	unsigned	It is a type modifier that alters the basic datatype
30	void	Used to indicate nothing - return value, parameter of a functio
31	volatile	Used to creating volatile objects
32	while	Used to define a looping statement

What are Identifiers?
In C language identifiers are the names given to variables, constants, functions and user-define data. These identifier are defined against a set of rules.



Rules for an Identifier

1. An Identifier can only have alphanumeric characters(a-z , A-Z , 0-9) and underscore(`_`).
2. The first character of an identifier can only contain alphabet(a-z , A-Z) or underscore (`_`).
3. Identifiers are also case sensitive in C. For example **name** and **Name** are two different identifiers in C.
4. Keywords are not allowed to be used as Identifiers.
5. No special characters, such as semicolon, period, whitespaces, slash or comma are permitted to be used in or as Identifier.

When we declare a variable or any function in C language program, to use it we must provide a name to it, which identified it throughout the program, for example:

```
int myvariable = "Studytonight";
```

Here `myvariable` is the name or identifier for the variable which stores the value "Studytonight" in it.

Character set

In C language characters are grouped into the following catagories,

1. Letters(all alphabets a to z & A to Z).
2. Digits (all digits 0 to 9).
3. Special characters, (such as colon `:` , semicolon `;` , period `.` , underscore `_` , ampersand `&` etc).
4. White spaces.

Data Types in C

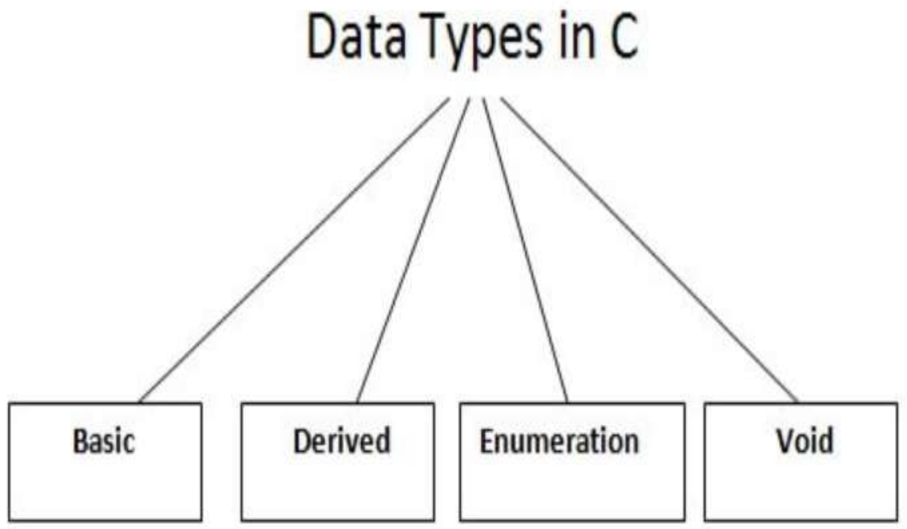
A data type specifies the type of data that a variable can store such as integer, floating, character, etc.

Basic Data Types

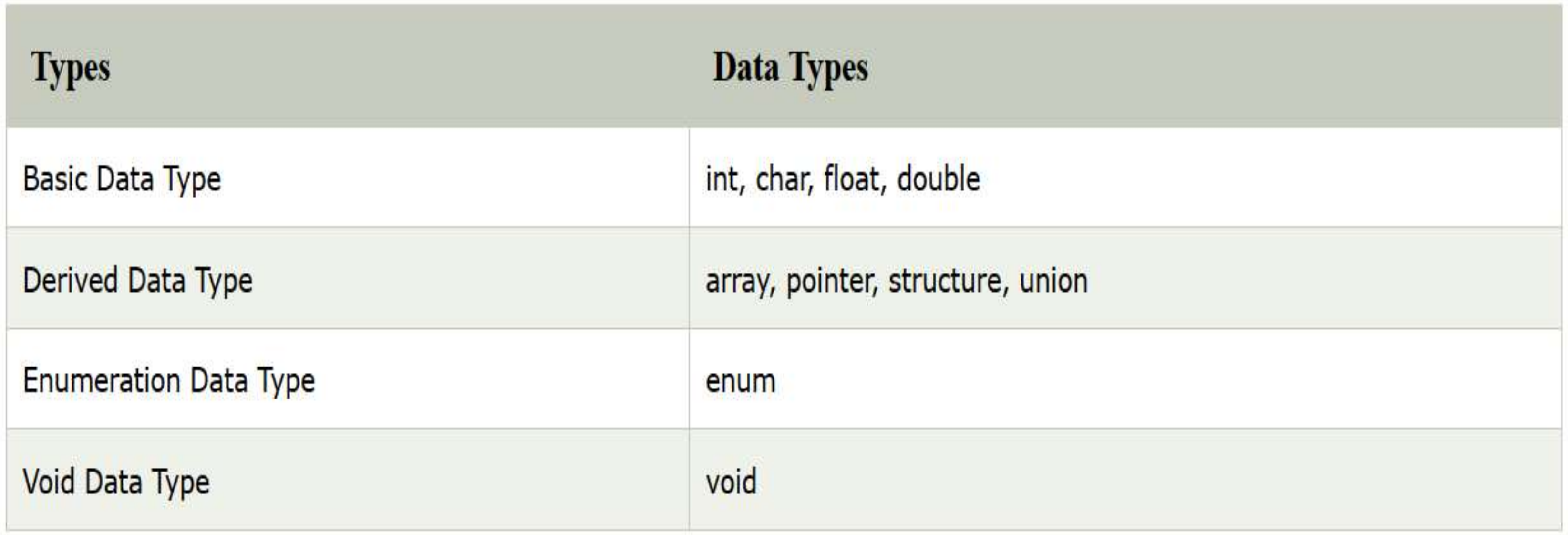
The basic data types are integer-based and floating-point based. C language supports both signed and unsigned literals.

The memory size of the basic data types may change according to 32 or 64-bit operating system.

Let's see the basic data types. Its size is given **according to 32-bit architecture**.



There are the following data types in C language.

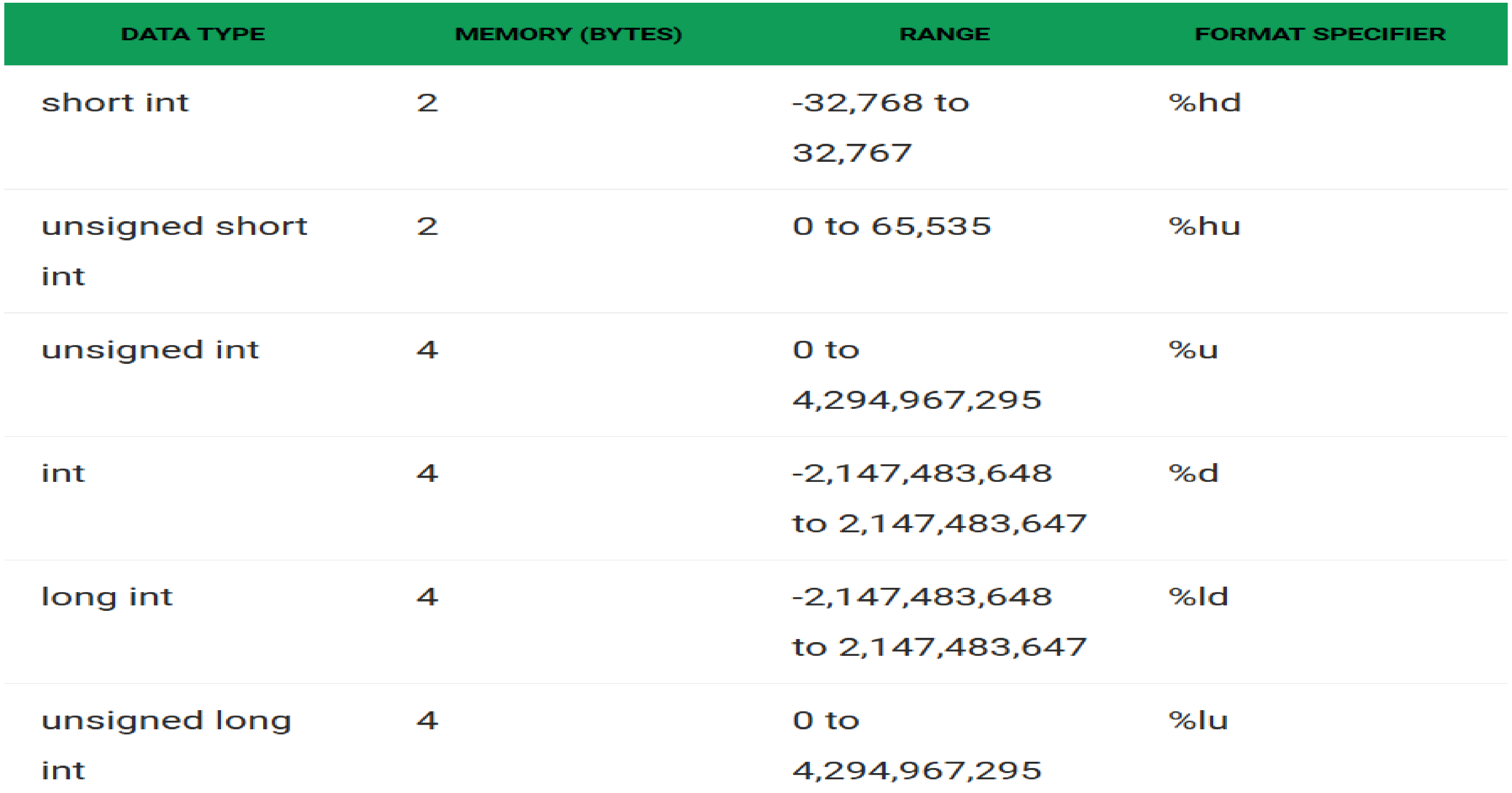


Types	Data Types
Basic Data Type	int, char, float, double
Derived Data Type	array, pointer, structure, union
Enumeration Data Type	enum
Void Data Type	void

Data Types	Memory Size	Range
char	1 byte	−128 to 127
signed char	1 byte	−128 to 127
unsigned char	1 byte	0 to 255
short	2 byte	−32,768 to 32,767
signed short	2 byte	−32,768 to 32,767
unsigned short	2 byte	0 to 65,535
int	2 byte	−32,768 to 32,767
signed int	2 byte	−32,768 to 32,767
unsigned int	2 byte	0 to 65,535
short int	2 byte	−32,768 to 32,767
signed short int	2 byte	−32,768 to 32,767
unsigned short int	2 byte	0 to 65,535

long int	4 byte	-2,147,483,648 to 2,147,483,647
signed long int	4 byte	-2,147,483,648 to 2,147,483,647
unsigned long int	4 byte	0 to 4,294,967,295
float	4 byte	
double	8 byte	
long double	10 byte	

FORMAT SPECIFIERS(ACCORDING TO GCC COMPILER)



DATA TYPE	MEMORY (BYTES)	RANGE	FORMAT SPECIFIER
short int	2	-32,768 to 32,767	%hd
unsigned short int	2	0 to 65,535	%hu
unsigned int	4	0 to 4,294,967,295	%u
int	4	-2,147,483,648 to 2,147,483,647	%d
long int	4	-2,147,483,648 to 2,147,483,647	%ld
unsigned long int	4	0 to 4,294,967,295	%lu

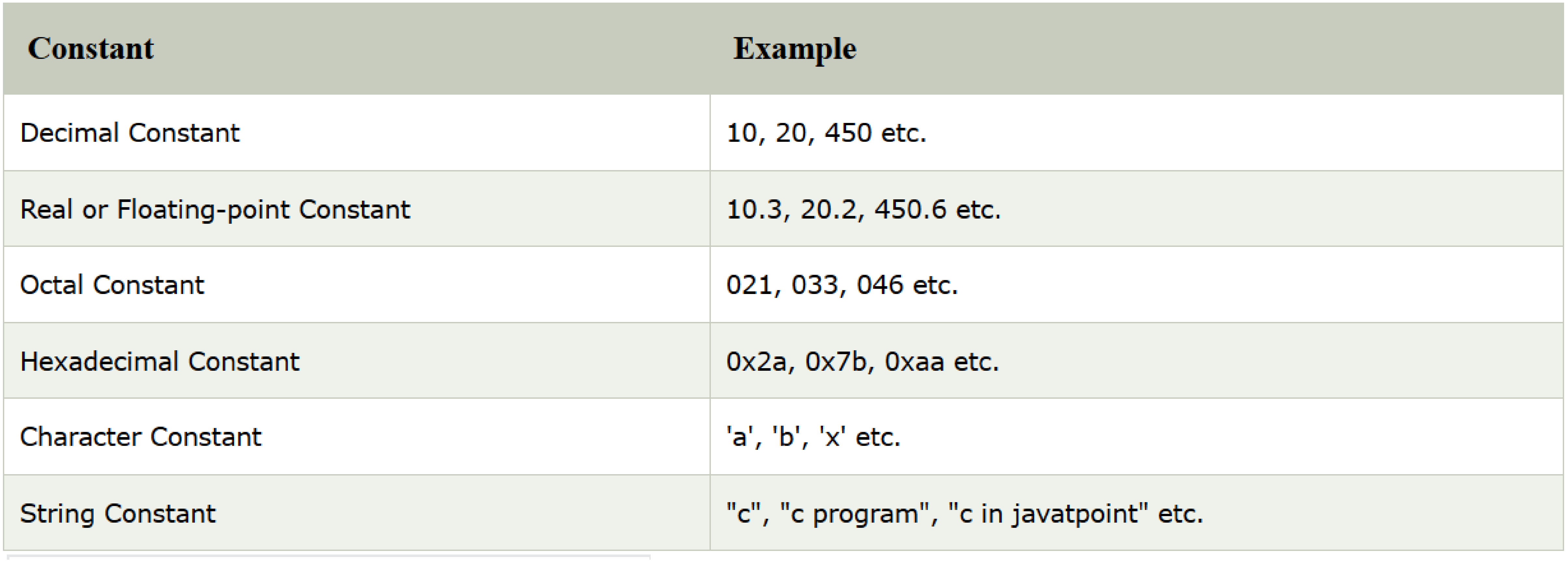
long long int	8	-(2^63) to (2^63)-1	%lld
unsigned long long int	8	0 to 18,446,744,073,709,551,615	%llu
signed char	1	-128 to 127	%c
unsigned char	1	0 to 255	%c
float	4		%f
double	8		%lf
long double	12		%Lf

Constants in C

A constant is a value or variable that can't be changed in the program, for example: 10, 20, 'a', 3.4, "c programming" etc.

There are different types of constants in C programming.

List of Constants in C



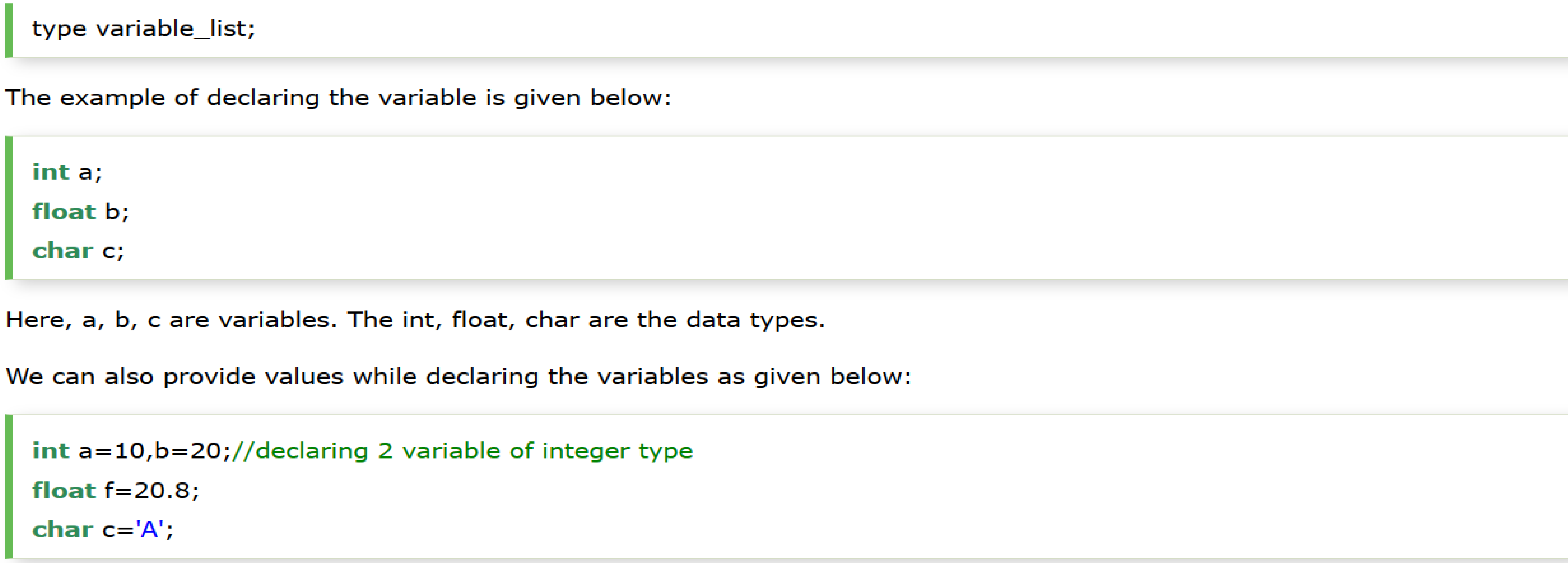
Constant	Example
Decimal Constant	10, 20, 450 etc.
Real or Floating-point Constant	10.3, 20.2, 450.6 etc.
Octal Constant	021, 033, 046 etc.
Hexadecimal Constant	0x2a, 0x7b, 0xaa etc.
Character Constant	'a', 'b', 'x' etc.
String Constant	"c", "c program", "c in javatpoint" etc.

Variables in C

A **variable** is a name of the memory location. It is used to store data. Its value can be changed, and it can be reused many times.

It is a way to represent memory location through symbol so that it can be easily identified.

Let's see the syntax to declare a variable:



```
type variable_list;
```

The example of declaring the variable is given below:

```c
int a;
float b;
char c;
```

Here, a, b, c are variables. The int, float, char are the data types.

We can also provide values while declaring the variables as given below:

```c
int a=10,b=20;//declaring 2 variable of integer type
float f=20.8;
char c='A';
```

Assignment Statement

An assignment is a statement in computer programming that is used to set a value to a variable name. The operator used to do assignment is denoted with an equal sign (=). This operand works by assigning the value on the right-hand side of the operand to the operand on the left-hand side.

It is possible for the same variable to hold different values at different instants of time.

A simple assignment statement could be:

int x = 5

This means that x can only take integer values, and currently a value of 5 is assigned to the variable x.

A single variable can hold various values at different times based on its life span and scope. When a variable which is already assigned a value is assigned another value, the new assignment value overwrites the previously assigned value. The assignment statement also varies from programming language to programming language.

Symbolic Constants in C

Symbolic Constant is a name that substitutes for a sequence of characters or a numeric constant, a character constant or a string constant.

When program is compiled each occurrence of a *symbolic constant* is replaced by its corresponding character sequence.

The syntax of Symbolic Constants in C

#define name text

where **name** implies symbolic name in caps.

text implies value or the text.

For example,

```
#define printf print
#define MAX 50
#define TRUE 1
#define FALSE 0
#define SIZE 15
```

The # character is used for **preprocessor** commands.

A preprocessor is a system program, which comes into action prior to Compiler, and it replaces the replacement text by the actual text.

Advantages of using Symbolic Constants

- They can be used to assign names to values.
- Replacement of value has to be done at one place and wherever the name appears in the text it gets the value by execution of the preprocessor.
- This saves time. if the Symbolic Constant appears 20 times in the program; it needs to be changed at one place only.

Input & Output Statements in C

The process of giving something to the computer is known as input. Input is mostly given through keyboard. The process of getting something from the computer is known as output. Output is mostly displayed on monitors.

Types of I/O Statements

1. Formatted I/O
   - scanf()
   - printf()
2. Unformatted I/O
   - gets(), puts()
   - getch(),getche(),putch()
   - getchar(),putchar()

The functions used for input and output are stored in the header file stdio.h. If programmer use any of the above function it is necessary to include header file.

1. Formatted Input & Output

(i) scanf() function

The function used to get input from the user during execution of program and stored in a variable of specified form is called scanf() function.

Syntax:

scanf(“format string”,& variable name);

format string	format specifier is written
&	address operator

Types:

- Single input e.g scanf(“%d”,&a);
- Multiple input scanf(“%d %c %d”, &a,&op,&b);|

Example

```
#include<stdio.h>
#include<conio.h>
void main ()
{
     int a;
     clrscr();
     printf("Enter integer value:");
     scanf("%d",&a);
     printf("The value you enter = %d",a);
```

```
      getch();
}
```

Output

```
Enter integer value:5
The value you enter =5
```

(ii) printf ()Function

This function is used to display text, constant or value of variable on screen in specified format.

Syntax:

```
printf("format string", argument list);
```

Types:

- printf(“hello world”); // Printf()with no argument list
- printf(“the value of integar=%d”,a); //Printf() with one argument
- printf(“Sum of %d+%d=%d”,a,b,a+b); //Printf()with more than one argument

Example:

```
#include<stdio.h>
#include<conio.h>
void main ()
{
printf(“HELLO WORLD!”);
getch();
}
```

Output

```
HELLO WORLD!
```

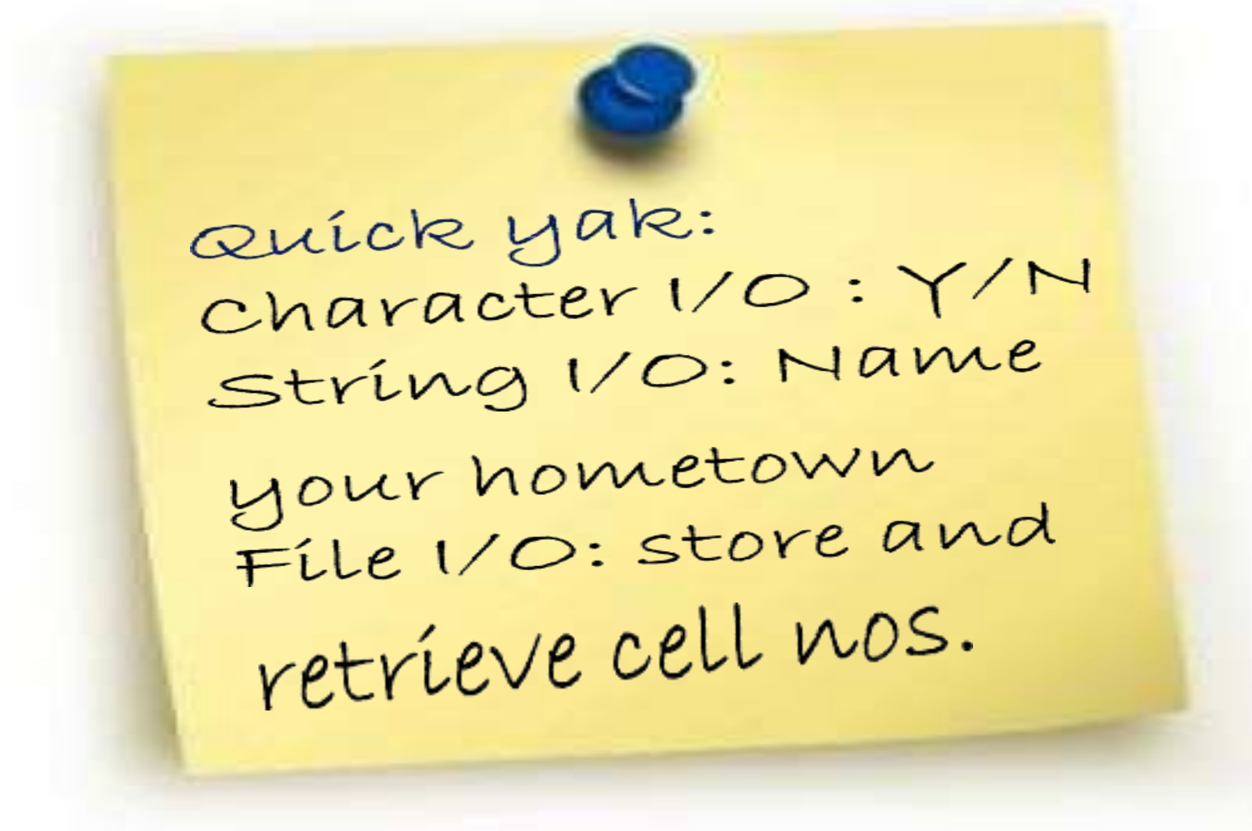
C Unformatted Functions

Unformatted input and output functions are only work with character data type. Unformatted input and output functions do not require any format specifiers. Because they only work with character data type.

–getchar()
–putchar()
–getch()
–putch()
–gets()
–puts()
- getche()

UNFORMATTED FUNCTIONS

- C has three types of I/O functions:
  1. Character I/O
  2. String I/O
  3. File I/O



Quick yak:
Character I/O : Y/N
String I/O: Name
your hometown
File I/O: store and
retrieve cell nos.

GETCHAR()

- This function reads a character-type data from standard input.
- It reads one character at a time till the user presses the enter key.

Syntax

```
Variable-name = getchar();
```

Example:

```
char c;
c = getchar();
```

```
#include<stdio.h>
 void main()
{
 char c;
 printf("enter a character");
 c=getchar();
 printf("c = %c ",c);
 }
```

```
Enter a character k
c = k
```

PUTCHAR()

- The putchar function like getchar is a part of the standard C input/output library. It transmits a single character to the standard output device (the computer screen).

Syntax

```
putchar( variable name);
```

Example: char c= ‘c’;

putchar (c);

```
#include<stdio.h>
 void main()
{
char ch;
printf("enter a character: ");
scanf("%c", ch);
putchar(ch);
}
```

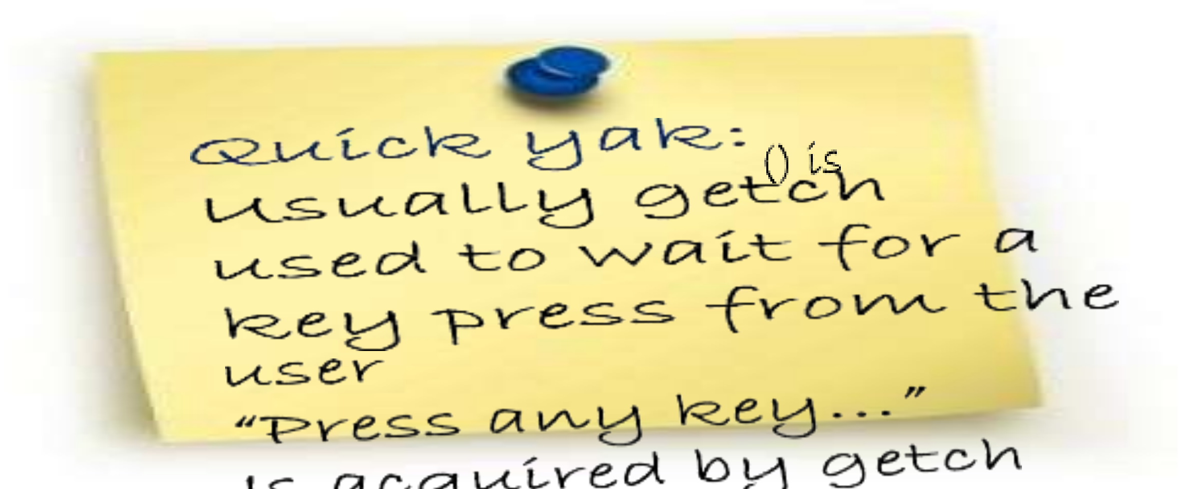
```
enter a character: r
r
```

GETCH() & GETCHE()

- These functions read any alphanumeric character from the standard input device
- The character entered is not displayed by the getch () function until enter is pressed
- The **getche()** accepts and displays the character.
- The **getch()** accepts but does not display the character.

Syntax

```
getche();
```


Quick yak:
usually getch() is
used to wait for a
key press from the
user
"Press any key..."
is acquired by getch

```
#include<stdio.h>
 void main()
{
    printf("Enter two alphabets:");
    getche();
    getch();
}
```

```
Enter two alphabets a
```

PUTCH()

Putch() displays any alphanumeric characters to the standard output device. It displays only one character at a time.

```
#include<stdio.h>
 void main()
{
      char ch;
      printf("Press any key to continue");
      ch = getch();
      printf(" you pressed:");
      putch(ch);
}
```

```
Press any key to continue
You pressed : e
```

GETS()

- String I/O

- This function is used for accepting any string until enter key is pressed (string will be covered later)

Syntax

```
char str[length of string in number];
gets(str);
```

```
#include<stdio.h>
 void main()
     {
     char ch[30];
     printf("Enter the string:");
     gets(ch);
     printf("Entered string: %s", ch);
     }
```

```
Enter the string: Use of data!
Entered string: Use of data!
```

PUTS()

- This function prints the string or character array. It is opposite to gets()

Syntax

```
char str[length of string in number];
gets(str);
puts(str);
```

```
#include<stdio.h>
void main()
      {
      char ch[30];
      printf("Enter the string:");
      gets(ch);
      puts("Entered string:");
      puts(ch);
      }
```

```
Enter the string: puts is in use
Entered string: puts is in use
```

Operators in C Language

C language supports a rich set of built-in operators. An operator is a symbol that tells the compiler to perform a certain mathematical or logical manipulation. Operators are used in programs to manipulate data and variables.

C operators can be classified into following types:

•Arithmetic operators
•Relational operators
•Logical operators
•Bitwise operators
•Assignment operators
•Conditional operators
•Special operators

Arithmetic operators

C supports all the basic arithmetic operators. The following table shows all the basic arithmetic operators.

Operator	Description
+	adds two operands
-	subtract second operands from first
*	multiply two operand
/	divide numerator by denominator
%	remainder of division
++	Increment operator - increases integer value by one
--	Decrement operator - decreases integer value by one

Relational operators

The following table shows all relation operators supported by C.

Operator	Description
==	Check if two operand are equal
!=	Check if two operand are not equal.
>	Check if operand on the left is greater than operand on the right
<	Check operand on the left is smaller than right operand
>=	check left operand is greater than or equal to right operand
<=	Check if operand on left is smaller than or equal to right operand

Logical operators

C language supports following 3 logical operators. Suppose a = 1 and b = 0,

Operator	Description	Example
&&	Logical AND	(a && b) is false
||	Logical OR	(a || b) is true
!	Logical NOT	(!a) is false

Bitwise operators

Bitwise operators perform manipulations of data at **bit level**. These operators also perform **shifting of bits** from right to left. Bitwise operators are not applied to float or double(These are datatypes, we will learn about them in the next tutorial).

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	left shift
>>	right shift

Now lets see truth table for bitwise &, | and ^

a	b	a & b	a | b	a ^ b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

The bitwise **shift** operator, shifts the bit value. The left operand specifies the value to be shifted and the right operand specifies the number of positions that the bits in the value have to be shifted. Both operands have the same precedence.

Example :
a = 0001000 b = 2
a << b = 0100000
a >> b = 0000010

Assignment Operators

Assignment operators supported by C language are as follows.

Operator	Description	Example
=	assigns values from right side operands to left side operand	a=b
+=	adds right operand to the left operand and assign the result to left	a+=b is same as a=a+b
-=	subtracts right operand from the left operand and assign the result to left operand	a-=b is same as a=a-b
*=	mutiply left operand with the right operand and assign the result to left operand	a*=b is same as a=a*b
/=	divides left operand with the right operand and assign the result to left operand	a/=b is same as a=a/b
%=	calculate modulus using two operands and assign the result to left operand	a%=b is same as a=a%b

Conditional operator

The conditional operators in C language are known by two more names

1.Ternary Operator

2.? : Operator

It is actually the if condition that we use in C language decision making, but using conditional operator, we turn the if condition statement into a short and simple operator.

The syntax of a conditional operator is :

expression 1 ? expression 2: expression 3

Explanation:

•The question mark **"?"** in the syntax represents the **if** part.

•The first expression (expression 1) generally returns either true or false, based on which it is decided whether (expression 2) will be executed or (expression 3)

•If (expression 1) returns true then the expression on the left side of **" : "** i.e (expression 2) is executed.

•If (expression 1) returns false then the expression on the right side of **" : "** i.e (expression 3) is executed.

Special operator

Operator	Description	Example
sizeof	Returns the size of an variable	**sizeof(x)** return size of the variable **x**
&	Returns the address of an variable	**&x ;** return address of the variable **x**
*	Pointer to a variable	***x ;** will be pointer to a variable **x**

Unary operators in C/C++

Unary operator: are operators that act upon a single operand to produce a new value.

Types of unary operators:

1.unary minus(-)
2.increment(++)
3.decrement(- -)
4.NOT(!)
5.Addressof operator(&)
6.sizeof()

1. unary minus

The minus operator changes the sign of its argument. A positive number becomes negative, and a negative number becomes positive.

```
int a = 10;
int b = -a; // b = -10
```

•unary minus is different from subtraction operator, as subtraction requires two operands.

2.increment

It is used to increment the value of the variable by 1. The increment can be done in two ways:

1.prefix increment

In this method, the operator preceeds the operand (e.g., ++a). The value of operand will be altered *before* it is used.

```
int a = 1;
int b = ++a; // b = 2
```

2.postfix increment

In this method, the operator follows the operand (e.g., a++). The value operand will be altered *after* it is used.

```
int a = 1;
int b = a++; // b = 1 int c = a; // c = 2
```

3. decrement

It is used to decrement the value of the variable by 1. The increment can be done in two ways:

1.prefix decrement

In this method, the operator preceeds the operand (e.g., – -a). The value of operand will be altered *before* it is used.

int a = 1;

int b = --a; // b = 0

2.posfix decrement

In this method, the operator follows the operand (e.g., a- -). The value of operand will be altered *after* it is used.

int a = 1;

int b = a--; // b = 1 int c = a; // c = 0

4. NOT(!): It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false. If x is true, then !x is false If x is false, then !x is true

5. Addressof operator(&): It gives an address of a variable. It is used to return the memory address of a variable. These addresses returned by the address-of operator are known as pointers because they “point” to the variable in memory. & gives an address on variable n int a; int *ptr; ptr = &a; // address of a is copied to the location ptr.

6. sizeof(): This operator returns the size of its operand, in bytes. The *sizeof* operator always precedes its operand.The operand is an expression, or it may be a cast.

```
#include <iostream>
using namespace std;

int main()
{
   float n = 0;
   cout << "size of n: " << sizeof(n);
   return 1;
}
```

Output:

```
size of n: 4
```

Arithmetic Expressions and their evaluation in C

Arithmetic Expressions consist of numeric literals, arithmetic operators, and numeric variables. They simplify to a single value, when evaluated. Here is an example of an arithmetic expression with no variables:

3.14*10*10

This expression evaluates to 314, the approximate area of a circle with radius 10. Similarly, the expression

3.14*radius*radius

would also evaluate to 314, if the variable radius stored the value 10.

Here are a couple expressions for you all to evaluate that use these operators:

Expression	Value
3 + 7 – 12	
6*4/8	
10*(12 - 4)	

Notice the parentheses in the last expression helps dictate which order to evaluate the expression. For the first two expressions, you simply evaluate the expressions from left to right.

Consider the following expression:
3 + 4*5

If evaluated from left to right, this would equal (3+4)*5 = 35

BUT, multiplication and division have a higher order of precedence than addition and subtraction. What this means is that in an arithmetic expression, you should first run through it left to right, only performing the multiplications and divisions. After doing this, process the expression again from left to right, doing all the additions and subtractions. So, 3+4*5 first evaluates to 3+20 which then evaluates to 23.

Consider this expression:

3 + 4*5 - 6/3*4/8 + 2*6 - 4*3*2

First go through and do all the multiplications and divisions:

3 + 20 - 1 + 12 - 24

Now, do all the additions and subtractions, left to right:

10

If you do NOT want an expression to be evaluated in this manner, you can simply add parentheses (which have the highest precedence) to signify which computations should be done first. (This is how we compute the subtraction first in 10*(12 - 4).)

our precedence chart has three levels: parentheses first, followed by multiplication and division, followed by addition and subtraction.

Integer Division

The one operation that may not work exactly as you might imagine in C is division. When two integers are divided, the C compiler will always make the answer evaluate to another integer. In particular, if the division has a leftover remainder or fraction, this is simply discarded. For example:

13/4 evaluates to 3
19/3 evaluates to 6 but

Similarly if you have an expression with integer variables part of a division, this evaluates to an integer as well. For example, in this segment of code, y gets set to 2.

```
int x = 8;
int y = x/3;
```

However, if we did the following,

```
double x = 8;
double y = x/3;
```

y would equal 2.66666666 (approximately).

The way C decides whether it will do an integer division (as in the first example), or a real number division (as in the second example), is based on the TYPE of the operands. If both operands are ints, an integer division is done. If either operand is a float or a double, then a real division is done. The compiler will treat constants without the decimal point as integers, and constants with the decimal point as a float. Thus, the expressions 13/4 and 13/4.0 will evaluate to 3 and 3.25 respectively.

The mod operator (%)

The one new operator (for those of you who have never programmed), is the mod operator, which is denoted by the percent sign(%). This operator is ONLY defined for integer operands. It is defined as follows:

a%b evaluates to the remainder of a divided by b. For example,

12%5 = 2
19%6 = 1
14%7 = 0
19%200 = 19

The precendence of the mod operator is the same as the precedence of multiplication and division.

For practice, try evaluating these expressions:

Expression	Value
3 + 10*(16%7) + 2/4	
3.0/6 + 18/(15%4+2)	
24/(1 + 2%3 + 4/5 + 6 + 31%8)	

(Note: The use of the % sign here is different than when it is used to denote a code for a printf statement, such as %d.)

What is Typecasting in C?

Typecasting is converting one data type into another one. It is also called as data conversion or type conversion. It is one of the important concepts introduced in 'C' programming.

'C' programming provides two types of type casting operations:

1. Implicit type casting
2. Explicit type casting

Implicit type casting

Implicit type casting means conversion of data types without losing its original meaning. This type of typecasting is essential when you want to change data types **without** changing the significance of the values stored inside the variable.

Implicit type conversion happens automatically when a value is copied to its compatible data type. During conversion, strict rules for type conversion are applied. If the operands are of two different data types, then an operand having lower data type is automatically converted into a higher data type. This type of type conversion can be seen in the following example.

```
#include <stdio.h>
 main()
{
 int num = 13;
char c = 'k'; /* ASCII value is 107 */
 float sum;
sum = num + c;
 printf("sum = %f\n", sum );}
```

Output:

```
sum = 120.000000
```

First of all, the c variable gets converted to integer, THEN the compiler converts **num** and **c** into "float" and adds them to produce a 'float' result.

Converting Character to Int

Consider the example of adding a character decoded in ASCII with an integer:

```
#include <stdio.h>
 main()
 {
 int number = 1;
char character = 'k'; /*ASCII value is 107 */
 int sum; sum = number + character;
printf("Value of sum : %d\n", sum ); }
```

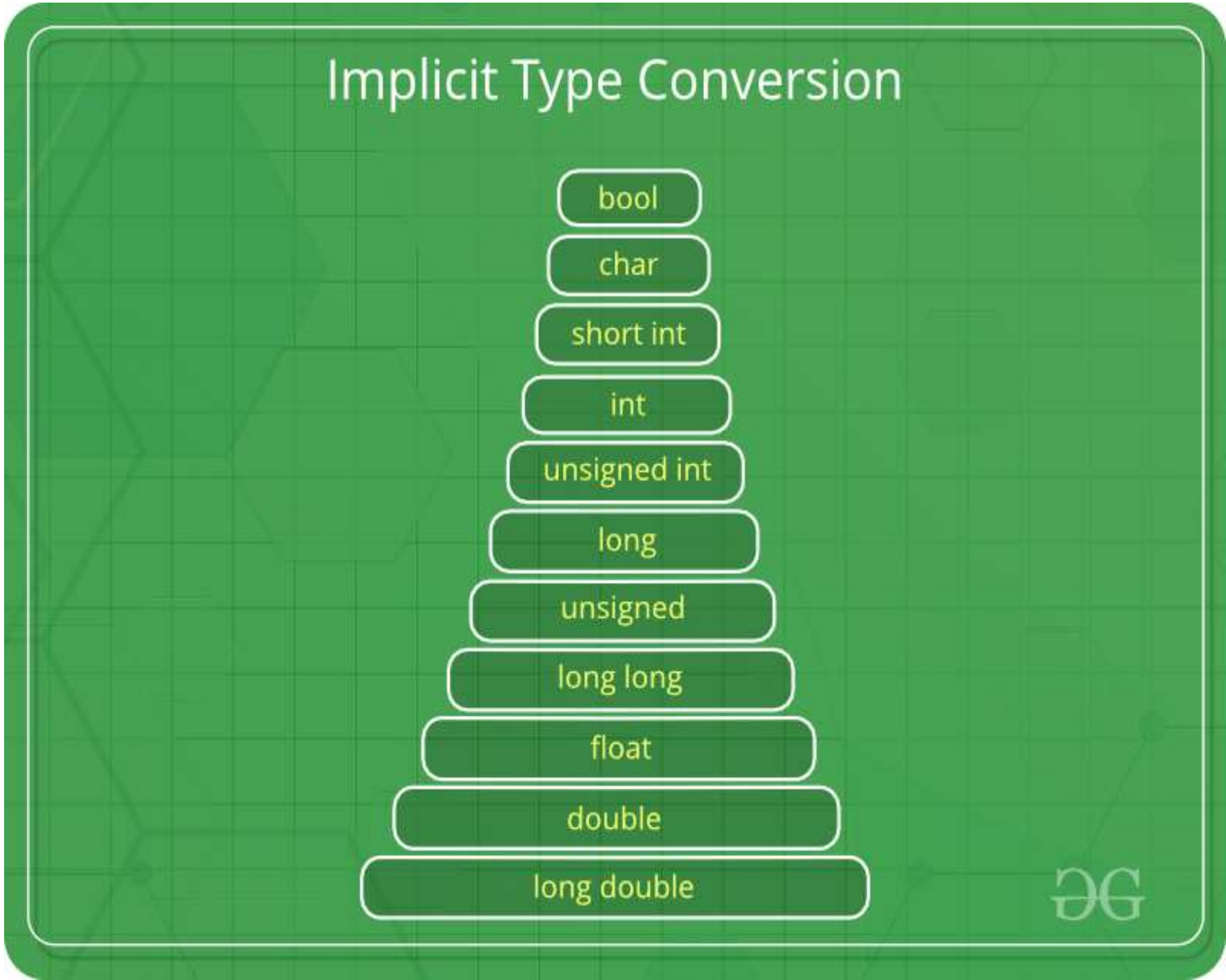
Output:

```
Value of sum : 108
```

Here, compiler has done an integer promotion by converting the value of 'k' to ASCII before performing the actual addition operation.

Implicit type conversion

The compiler first proceeds with promoting a character to an integer. If the operands still have different data types, then they are converted to the highest data type that appears in the following hierarchy chart:



Important Points about Implicit Conversions

•Implicit type of type conversion is also called as standard type conversion. We do not require any keyword or special statements in implicit type casting.

•Converting from smaller data type into larger data type is also called as **type promotion**.

•The implicit type conversion always happens with the compatible data types.

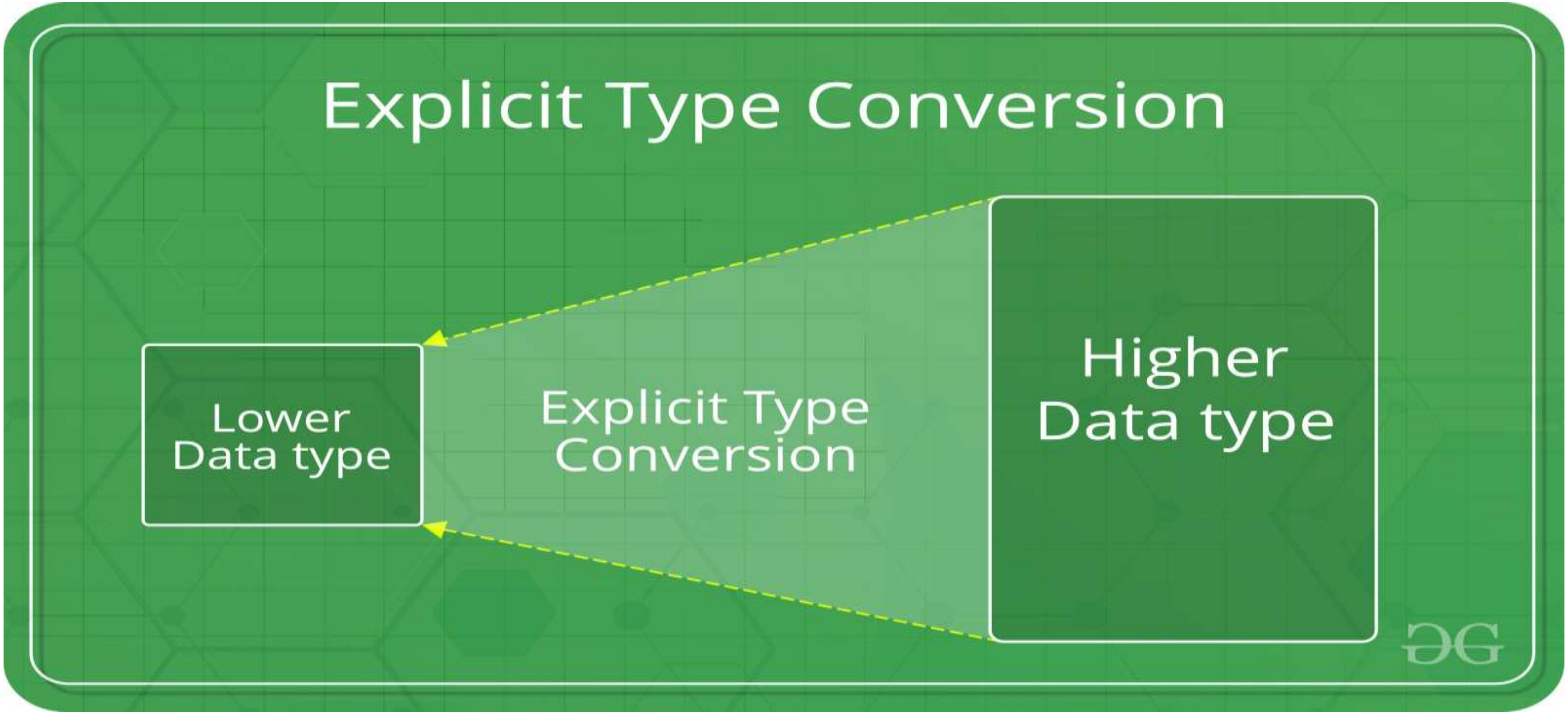
We cannot perform implicit type casting on the data types which are not compatible with each other such as:

1. Converting float to an int will truncate the fraction part hence losing the meaning of the value.
2. Converting double to float will round up the digits.
3. Converting long int to int will cause dropping of excess high order bits.

In all the above cases, when we convert the data types, the value will lose its meaning. Generally, the loss of meaning of the value is warned by the compiler.

'C' programming provides another way of typecasting which is explicit type casting.

Explicit Type Conversion–



Explicit type casting

In implicit type conversion, the data type is converted automatically. There are some scenarios in which we may have to force type conversion. Suppose we have a variable div that stores the division of two operands which are declared as an int data type.

```
int result, var1=10, var2=3;
result=var1/var2;
```

In this case, after the division performed on variables var1 and var2 the result stored in the variable "result" will be in an integer format. Whenever this happens, the value stored in the variable "result" loses its meaning because it does not consider the fraction part which is normally obtained in the division of two numbers.
To force the type conversion in such situations, we use explicit type casting.
It requires a type casting operator. The general syntax for type casting operations is as follows:

```
(type-name) expression
```

E.g.

```
int main()
{
    double x = 1.2;

    // Explicit conversion from double to int
    int sum = (int)x + 1;

    printf("sum = %d", sum);

    return 0;
}
```

Output:

```
sum = 2
```

Arithmetic operations using type casting

```
#include <stdio.h>
    int main()
    {
        int a = 9,b = 4, c;
        float d;
        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);
d = (float)a/b;
        printf("a/b = %f \n",d);
        c = a%b;
        printf("Remainder when a divided by b = 
%d \n",c);

        return 0;
    }
```

Summary

•Typecasting is also called as type conversion

•It means converting one data type into another.

•Converting smaller data type into a larger one is also called as type promotion.

•'C' provides an implicit and explicit way of type conversion.

•Implicit type conversion operates automatically when the compatible data type is found.

•Explicit type conversion requires a type casting operator.

Keep in mind the following rules for programming practice when dealing with different data type to prevent from data loss :

•Integers types should be converted to float.

•Float types should be converted to double.

•Character types should be converted to integer.

Operators hierarchy and associativity in C

The precedence of operator species that which operator will be evaluated first and next. The associativity specifies the operator direction to be evaluated; it may be left to right or right to left.

Let's understand the precedence by the example given below:
1.int value=10+20*10;
The value variable will contain **210** because * (multiplicative operator) is evaluated before + (additive operator).
The precedence and associativity of C operators is given below:

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right

Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Control statements

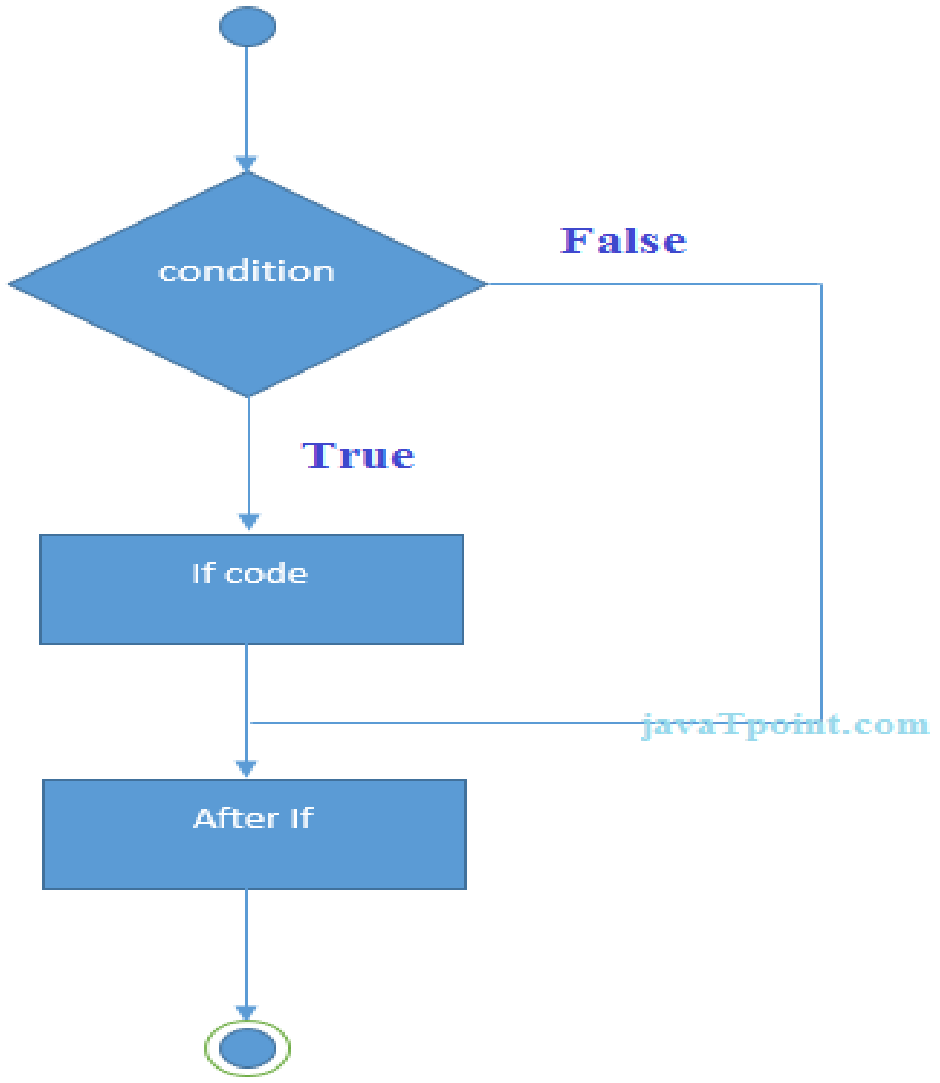
The if-else statement in C is used to perform the operations based on some specific condition. The operations specified in if block are executed if and only if the given condition is true.
There are the following variants of if statement in C language.

- If statement
- If-else statement
- If else-if ladder
- Nested if

If Statement

The if statement is used to check some given condition and perform some operations depending upon the correctness of that condition. It is mostly used in the scenario where we need to perform the different operations for the different conditions. The syntax of the if statement is given below.

```
if(expression){
//code to be executed
}
```


condition
False
True
If code
javaTpoint.com
After If

Let's see a simple example of C language if statement.

```
#include<stdio.h>
int main(){
int number=0;
printf("Enter a number:");
scanf("%d",&number);
if(number%2==0){
printf("%d is even number",number);
}
return 0;
}
```

OUTPUT:

```
Enter a number:4
4 is even number
enter a number:5
```

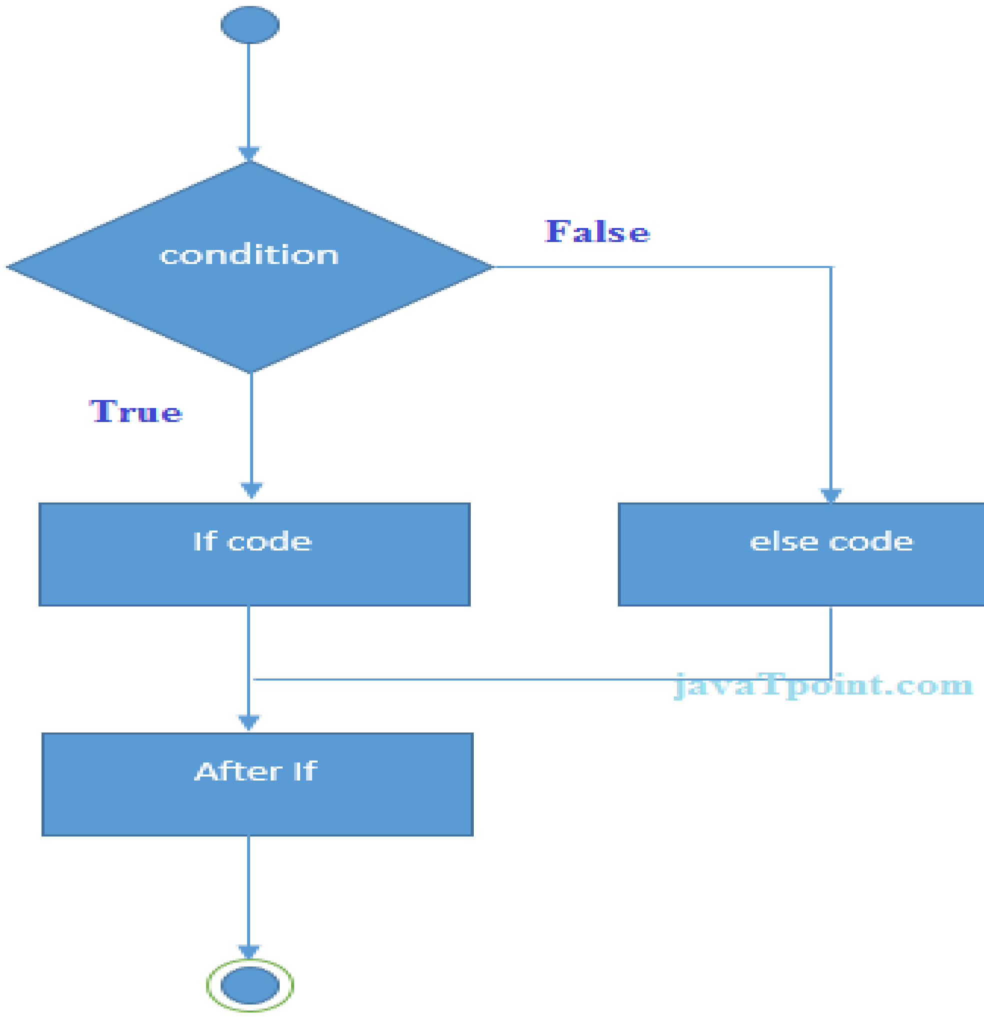
If-else Statement

The if-else statement is used to perform two operations for a single condition. The if-else statement is an extension to the if statement using which, we can perform two different operations, i.e., one is for the correctness of that condition, and the other is for the incorrectness of the condition. Here, we must notice that if and else block cannot be executed simulteneously. Using if-else statement is always preferable since it always invokes an otherwise case with every if condition. The syntax of the if-else statement is given below.

Syntax:

```
if(expression){
//code to be executed if condition is true
}else{
//code to be executed if condition is false
}
```

Flowchart of the if-else statement in C



Let's see the simple example to check whether a number is even or odd using if-else statement in C language.

```
#include<stdio.h>
int main(){
int number=0;
printf("enter a number:");
scanf("%d",&number);
if(number%2==0){
printf("%d is even number",number);
}
else{
printf("%d is odd number",number);
}
return 0;
}
```

Output

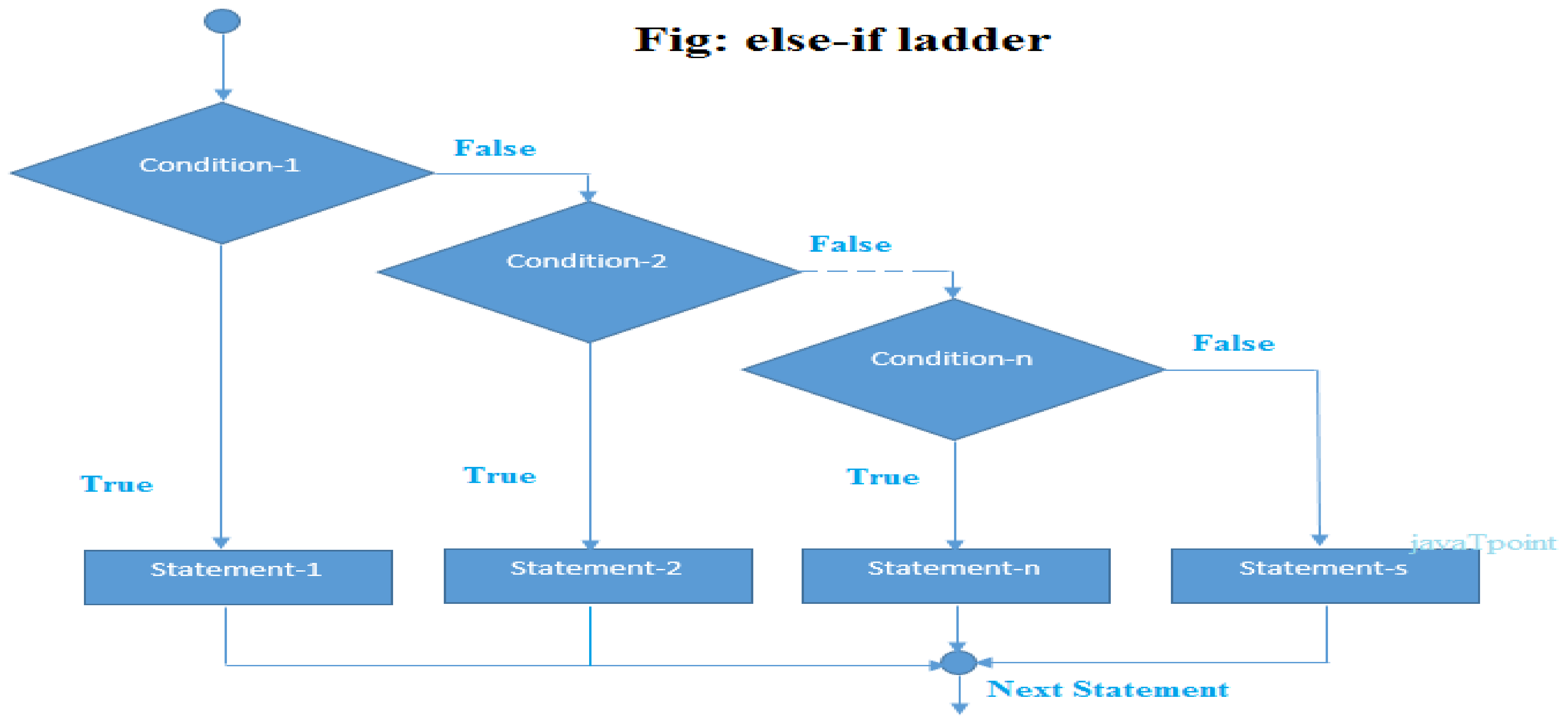
```
enter a number:4
 4 is even number
enter a number:5
 5 is odd number
```

If else-if ladder Statement

The if-else-if ladder statement is an extension to the if-else statement. It is used in the scenario where there are multiple cases to be performed for different conditions. In if-else-if ladder statement, if a condition is true then the statements defined in the if block will be executed, otherwise if some other condition is true then the statements defined in the else-if block will be executed, at the last if none of the condition is true then the statements defined in the else block will be executed. There are multiple else-if blocks possible. It is similar to the switch case statement where the default is executed instead of else block if none of the cases is matched.

```
if(condition1){
//code to be executed if condition1 is true
}else if(condition2){
//code to be executed if condition2 is true
}
else if(condition3){
//code to be executed if condition3 is true
}
...
else{
//code to be executed if all the conditions are false
}
```

Flowchart of else-if ladder statement in C



The example of an if-else-if statement in C language is given below.

```
#include<stdio.h>
int main(){
int number=0;
printf("enter a number:");
scanf("%d",&number);
if(number==10){
printf("number is equals to 10");
}
else if(number==50){
printf("number is equal to 50");
}
else if(number==100){
printf("number is equal to 100");
}
else{
printf("number is not equal to 10, 50 or 100");

}
return 0;
}
```

Output

```
enter a number:4
number is not equal to 10, 50 or
100
enter a number:50
number is equal to 50
```

C Switch Statement

The switch statement in C is an alternate to if-else-if ladder statement which allows us to execute multiple operations for the different possibles values of a single variable called switch variable. Here, We can define various statements in the multiple cases for the different values of a single variable.

The syntax of switch statement in c language is given below:

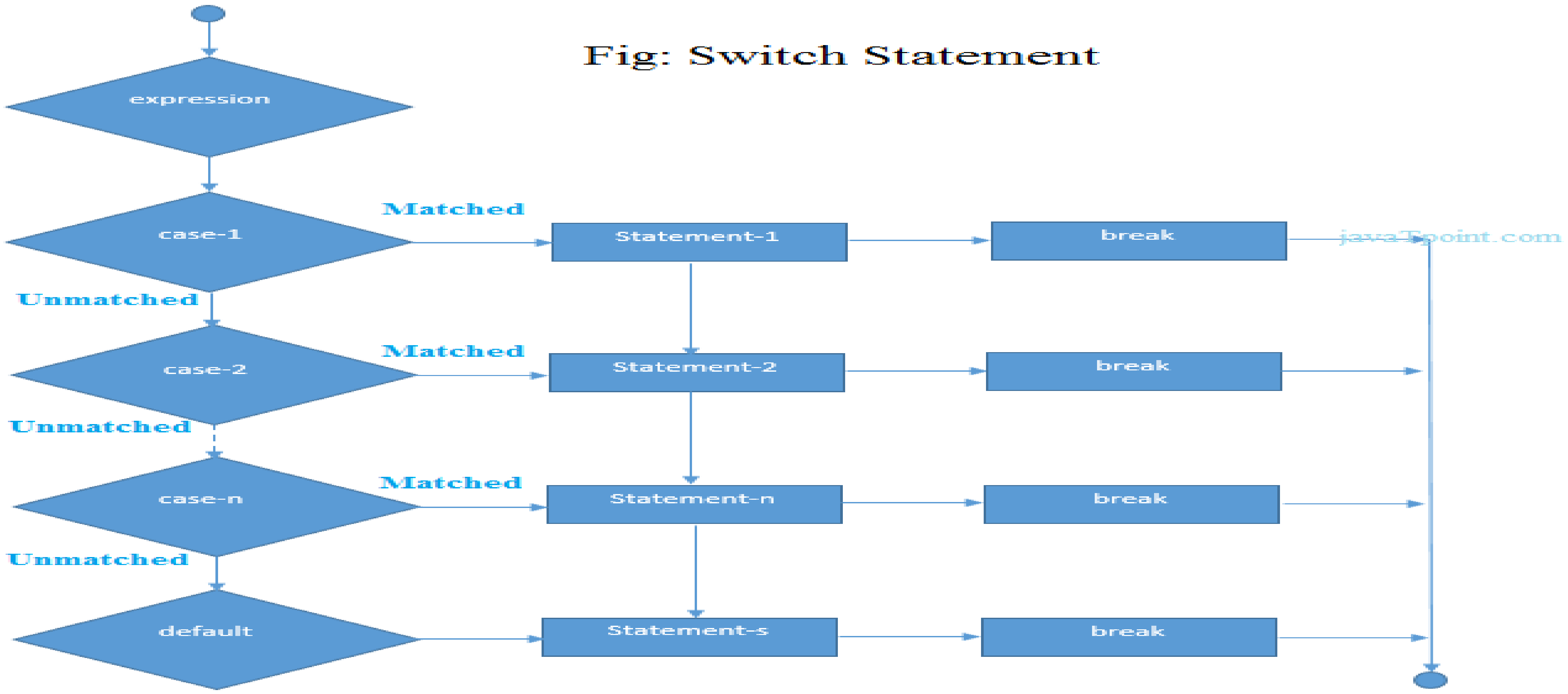
```
switch(expression){
case value1:
 //code to be executed;
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
......

default:
 code to be executed if all cases are not matched;

}
```

Rules for switch statement in C language

1) The *switch expression* must be of an integer or character type.

2) The *case value* must be an integer or character constant.
3) The *case value* can be used only inside the switch statement.
4) The *break statement* in switch case is not must. It is optional. If there is no break statement found in the case, all the cases will be executed present after the matched case. It is known as *fall through* the state of C switch statement.

Flowchart of switch statement in C



Let's see a simple example of c language switch statement.

```
#include<stdio.h>
int main(){
int number=0;
printf("enter a number:");
scanf("%d",&number);
switch(number){
case 10:
printf("number is equals to 10");
break;
case 50:
printf("number is equal to 50");
break;
case 100:
printf("number is equal to 100");
break;
default:
printf("number is not equal to 10, 50 or 100");
}
return 0;
}
```

Output

```
enter a number:4
number is not equal to 10, 50 or 100
enter a number:50
number is equal to 50
```

Loop Statements

In programming, loops are used to repeat a block of code until a specified condition is met.

C programming has three types of loops:

1. for loop
2. while loop
3. do...while loop

for Loop

The syntax of the for loop is:

```
1.for (initializationStatement; testExpression; updateStatement)
2.{
3.// statements inside the body of loop
4.}
```

How for loop works?

•The initialization statement is executed only once.

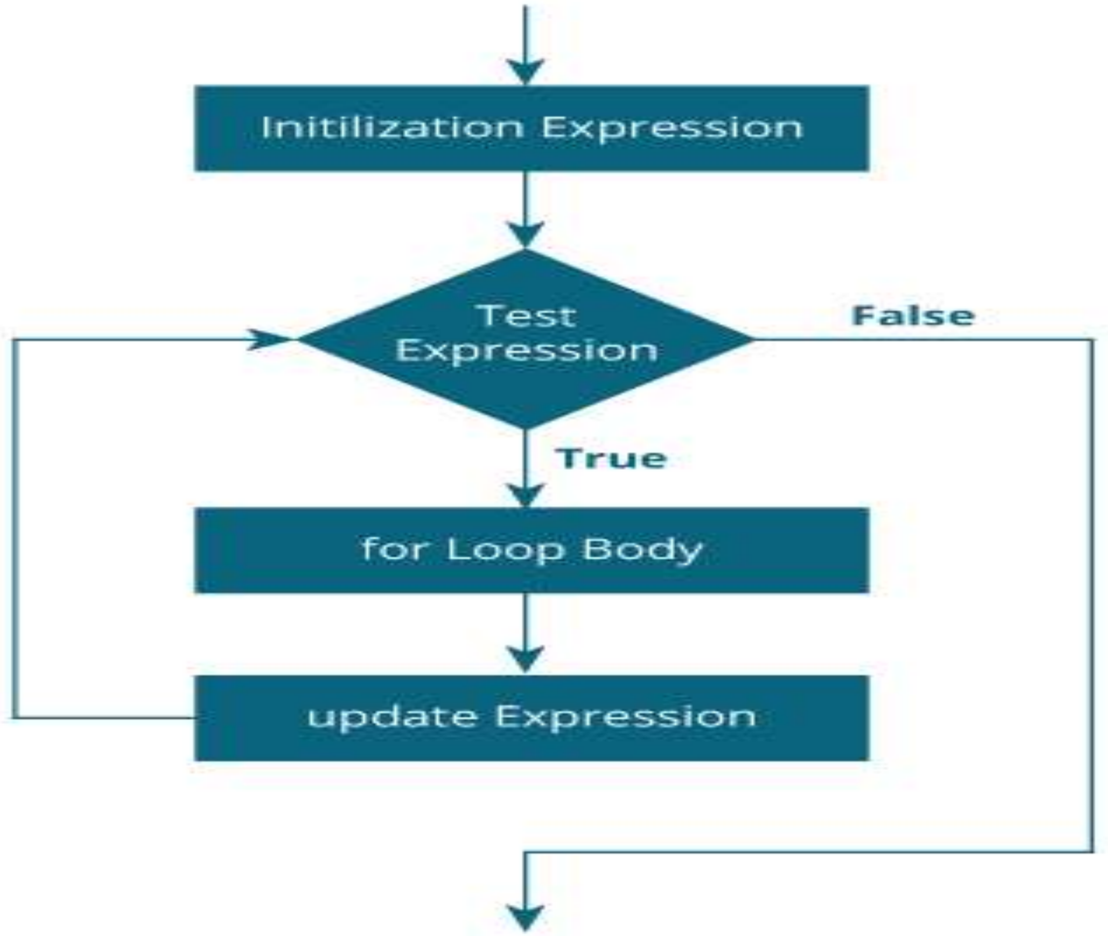
•Then, the test expression is evaluated. If the test expression is evaluated to false, the for loop is terminated.

•However, if the test expression is evaluated to true, statements inside the body of for loop are executed, and the update expression is updated.

•Again the test expression is evaluated.

This process goes on until the test expression is false. When the test expression is false, the loop terminates.

for loop Flowchart



Example 1: for loop

```
# Print numbers from 1 to 10
#include <stdio.h>
int main() {
int i;
for (i = 1; i < 11; ++i)
{
printf("%d ", i);
}
return 0;
}
```

Output

```
1 2 3 4 5 6 7 8 9 10
```

- *i* is initialized to 1.
- The test expression i < 11 is evaluated. Since 1 less than 11 is true, the body of for loop is executed. This will print the 1 (value of *i*) on the screen.
- The update statement ++i is executed. Now, the value of *i* will be 2. Again, the test expression is evaluated to true, and the body of for loop is executed. This will print 2 (value of *i*) on the screen.
- Again, the update statement ++i is executed and the test expression i < 11 is evaluated. This process goes on until *i* becomes 11.
- When *i* becomes 11, *i* < *11* will be false, and the for loop terminates

while loop

The syntax of the while loop is:

1.while (testExpression)
2.{
3.// statements inside the body of the loop
4.}

How while loop works?

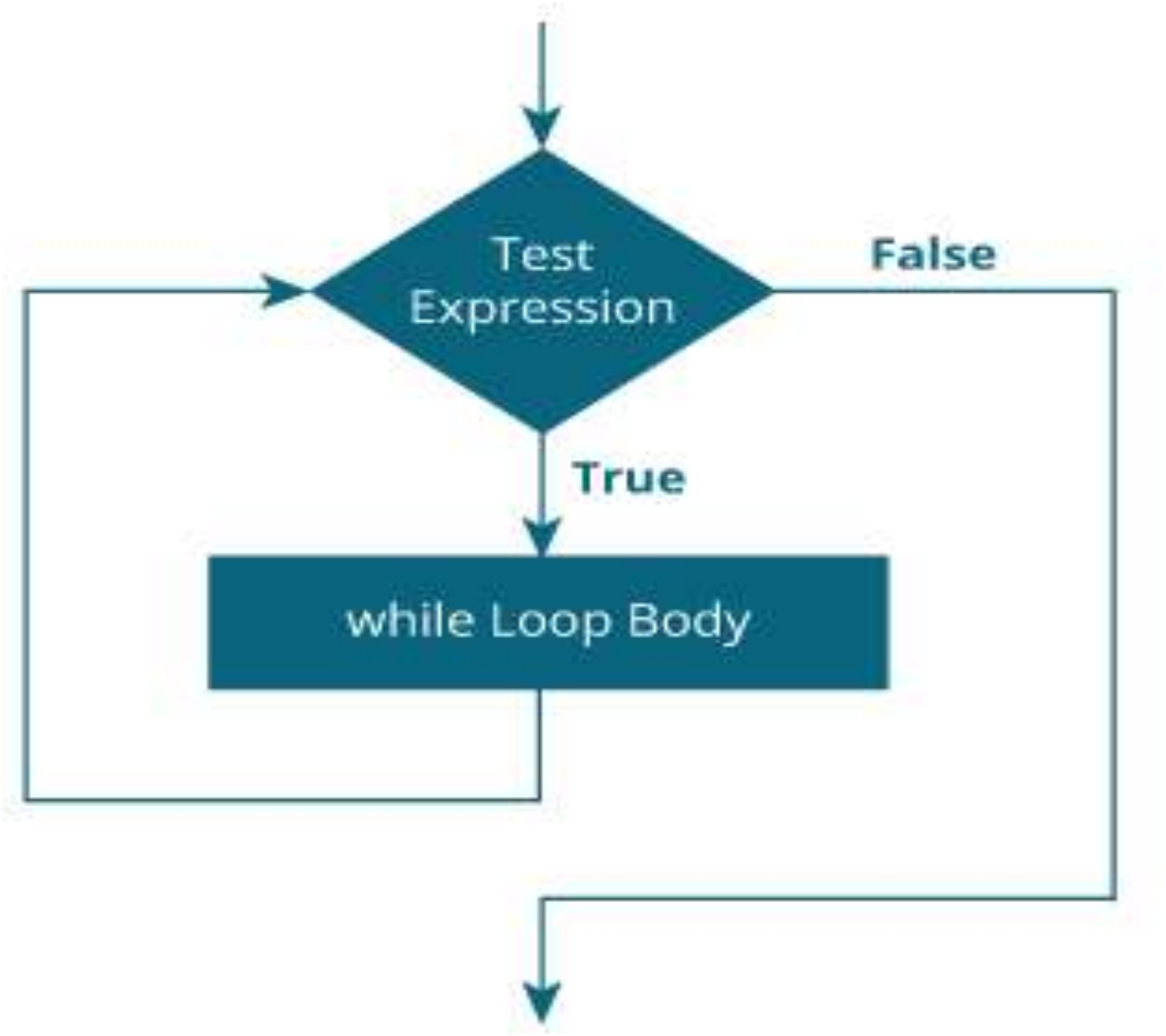
•The while loop evaluates the test expression inside the parenthesis ().

•If the test expression is true, statements inside the body of while loop are executed. Then, the test expression is evaluated again.

•The process goes on until the test expression is evaluated to false.

•If the test expression is false, the loop terminates (ends).

Flowchart of while loop



Example 1: while loop

Print numbers from 1 to 10

```
#include <stdio.h>
int main()
 {
int i=1;
While(i < 11)
{
printf("%d\n", i);
 i++;
}
return 0;
}
```

Output

```
1
2
3
4
5
6
7
8
9
10
```

Here, we have initialized *i* to 1.

1. When *i* is 1, the test expression i <= 5 is true. Hence, the body of the while loop is executed. This prints 1 on the screen and the value of *i* is increased to 2.
2. Now, *i* is 2, the test expression i <= 5 is again true. The body of the while loop is executed again. This prints 2 on the screen and the value of i is increased to 3.
3. This process goes on until *i* becomes 6. When *i* is 6, the test expression i <= 5 will be false and the loop terminates.

do...while loop

The do..while loop is similar to the while loop with one important difference. The body of do...while loop is executed at least once. Only then, the test expression is evaluated.

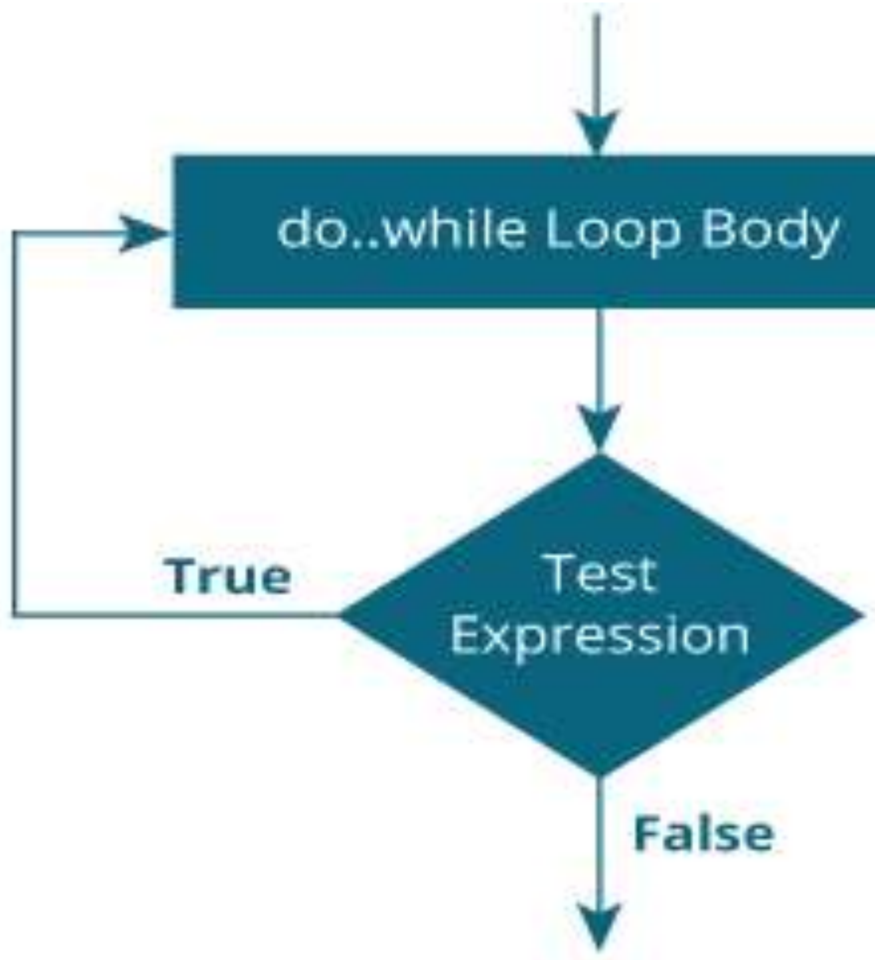
The syntax of the do...while loop is:

```
1.do
2.{
3.// statements inside the body of the loop
4.}
5.while (testExpression);
```

How do...while loop works?

- The body of do...while loop is executed once. Only then, the test expression is evaluated.
- If the test expression is true, the body of the loop is executed again and the test expression is evaluated.
- This process goes on until the test expression becomes false.
- If the test expression is false, the loop ends

Flowchart of do...while Loop



Example 2: do...while loop

Print numbers from 1 to 10

```
#include <stdio.h>
int main()
 {
int i=1;
do
{
printf("%d\n", i);
 i++;
} While(i < 11);
return 0;
}
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Jump Statements(break, continue,goto)

C break statement

The break is a keyword in C which is used to bring the program control out of the loop. The break statement is used inside loops or switch statement. The break statement breaks the loop one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops. The break statement in C can be used in the following two scenarios:

With switch case

With loop

Syntax:

//loop or switch case

break;

Flowchart of break in c

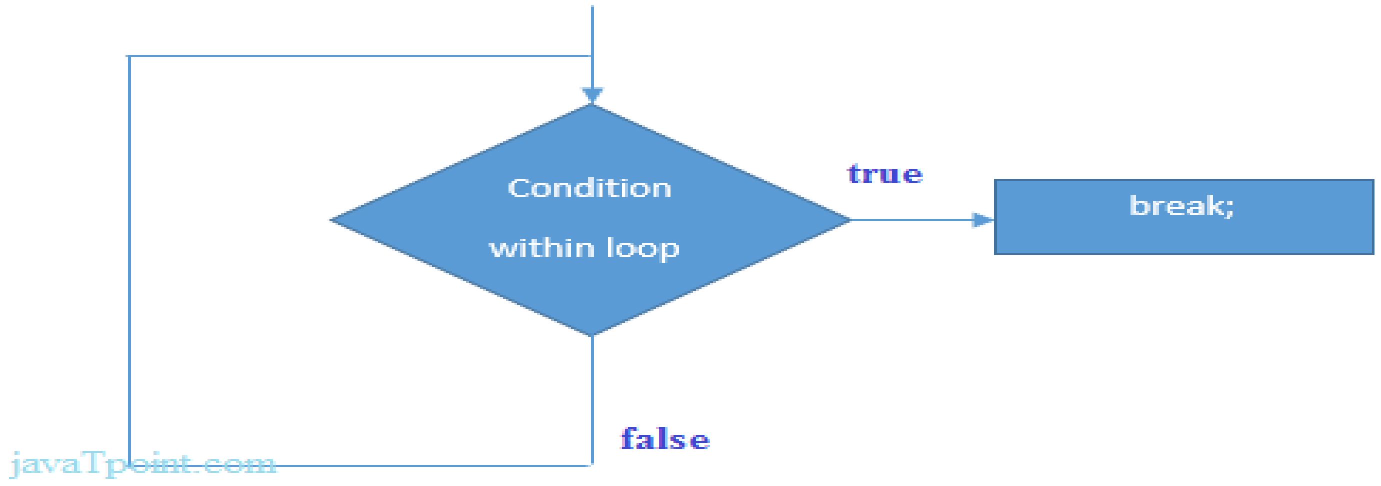


Figure: Flowchart of break statement

Example

```
#include<stdio.h>
#include<stdlib.h>
void main ()
{
    int i;
    for(i = 0; i<10; i++)
    {
        printf("%d ",i);
        if(i == 5)
          break;
     }
     printf("came outside of loop i = %d",i);

}
```

Output

```
0 1 2 3 4 5 came outside of loop i = 5
```

C continue statement

The **continue statement** in C language is used to bring the program control to the beginning of the loop. The continue statement skips some lines of code inside the loop and continues with the next iteration. It is mainly used for a condition so that we can skip some code for a particular condition.

Syntax:

```
1.//loop statements
2.continue;
3.//some lines of the code which is to be skipped
```

Continue statement example

```c
#include<stdio.h>
int main(){
int i=1;//initializing a local variable
//starting a loop from 1 to 10
for(i=1;i<=10;i++){
if(i==5){//if value of i is equal to 5, it will continue the loop
continue;
}
printf("%d \n",i);
}//end of for loop
return 0;
```

Output

```
1
2
3
4
6
7
8
9
10
```

As you can see, 5 is not printed on the console because loop is continued at i==5.

C goto statement

The goto statement is known as jump statement in C. As the name suggests, goto is used to transfer the program control to a predefined label. The goto statment can be used to repeat some part of the code for a particular condition. It can also be used to break the multiple loops which can't be done by using a single break statement. However, using goto is avoided these days since it makes the program less readable and complicated.

Syntax:

1.label:

2.//some part of the code;

3.goto label;

goto example

Let's see a simple example to use goto statement in C language.

```c
#include <stdio.h>
int main()
{
int num,i=1;
printf("Enter the number whose table you want to print?");
scanf("%d",&num);
table:
printf("%d x %d = %d\n",num,i,num*i);
i++;
if(i<=10)
goto table;
}
```

Output:

```
Enter the number
whose table you want
to print?10
10 x 1 = 10
10 x 2 = 20
10 x 3 = 30
10 x 4 = 40
10 x 5 = 50
10 x 6 = 60
10 x 7 = 70
10 x 8 = 80
10 x 9 = 90
10 x 10 = 100
```

NESTED CONTROL STRUCTURES

C programming allows to use one loop inside another loop. Loops,like *if-else* statements,can be nested,on within another.the inner and outer loops need not be generated by the same type of control structure.It is essential,however,that one loop be completely embedded within on other-there can be no overlap. Each loop must be controlled by a different index.

Syntax

The syntax for a **nested for loop** statement in C is as follows −

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

The syntax for a **nested do...while loop** statement in C programming language is as follows −

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

The syntax for a **nested while loop** statement in C programming language is as follows −

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

Example:

```
#include<stdio.h>
main()
{
    int i, j;
    for(i=0;i<4;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("for i=%d, j=%d\n",i,j);
        }
        printf("------------\n");
    }
}
```

Output:

```
for i=0, j=0
for i=0, j=1
for i=0, j=2
------------
for i=1, j=0
for i=1, j=1
for i=1, j=2
------------
for i=2, j=0
for i=2, j=1
for i=2, j=2
------------
for i=3, j=0
for i=3, j=1
for i=3, j=2
------------
```

Decision Making and Branching

In programming the order of execution of instructions may have to be changed depending on certain conditions. This involves a kind of decision making to see whether a particular condition has occurred or not and then direct the computer to execute certain instructions accordingly.

Decision making with if statement:

If Statement : The if statement is powerful decision making statement and is used to control the flow of execution of statements The If statement may be complexity of conditions to be tested

(a) Simple if statement
(b) If else statement
(c) Nested If-else statement
(d) Else –If ladder

Simple If Statement : The general form of simple if statement is

```
                If(test expression)
{     statement block;
                }     statement-x ;
```

Ex :

```
 If(category = sports)
{   marks = marks + bonus marks;
} printf("%d",marks);
```

If the student belongs to the sports category then additional bonus marks are added to his marks before they are printed. For other bonus marks are not added .

If –Else Statement : The If statement is an extension of the simple If statement the general form is

```
                If (test expression)
                {
                   true-block statements;
                }
else
                {
                  false-block statements;
                }
  statement – x;
```

If the test expression is true then block statement are executed, otherwise the false –block statement are executed. In both cases either true-block or false-block will be executed not both.

```
Ex :     If (code == 1)
boy = boy + 1;
         else
         girl = girl + 1;
         st-x;
```

Here if the code is equal to ‘1’ the statement *boy=boy+1;* Is executed and the control is transfered to the *statement st-n*, after skipping the else part. If code is not equal to ‘1’ the statement *boy =boy+1;* is skipped and the statement in the else part *girl =girl+1;* is executed before the control reaches the statement st-n.

Nested If –else statement : When a series of decisions are involved we may have to use more than one if-else statement in nested form of follows .

```
                If(test expression)
{ if(test expression)
{    st –1;
}
else
{   st – 2;
}else
{
   st – 3;
}
}st – x;
```

If the condition is false the st-3 will be executed otherwise it continues to perform the nested If – else structure (inner part). If the condition 2 is true the st-1 will be executed otherwise the st-2 will be evaluated and then the control is transferred to the st-x

Some other forms of nesting If-else

```
If ( test condition1)
{  if (test condition2)
st –1 ;
} else
if (condition 3)
{ if (condition 4)
st – 2;
}st – x;
```

Else-If ladder : A multi path decision is charm of its in which the statement associated with each else is an If. It takes the following general form.

```
            If (condition1)
St –1;
Else  If (condition2)
St –2;
Else if (condition 3)
St –3;
 |
Else
Default – st;
St –x;
```

```
Ex :            If (code = = 1)            Color = "red";
Else if ( code = = 2)    Color = "green"
Else if (code = = 3)     Color = "white";
Else     Color = "yellow";
```

If code number is other than 1,2 and then color is yellow.

Switch Statement : Instead of else –if ladder, 'C' has a built-in multi-way decision statement known as a switch. The general form of the switch statement is as follows.

```
                    Switch (expression)
{
case value1   : block1;
                    break;
case value 2  : block 2;
                    break;


|
default         :  default block;
                    break;
}
st – x;
```

Ex :

```
                    switch (number)
{
case 1 : printf("Monday");
            break;
case 2 : printf("Tuesday");
            break;
|case 3 : printf("Wednesday");
            break;
case 4 : printf("Thursday");
            break;
case 5 : printf("Friday");
            break;
          default : printf("Saturday");
            break;
}
```

Goto Statement : The goto statement is used to transfer the control of the program from one point to another. It is something reffered to as unconditionally branching. The goto is used in the form

Goto label;

Label statement : The label is a valid 'C' identifier followed by a colon. we can precode any statement by a label in the form

Label : statement ;

This statement immediately transfers execution to the statement labeled with the label identifier

Ex :

```
i = 1;
bc : if(1>5)
goto ab;
printf("%d",i);
++i;
goto bc;
ab : {
printf("%d",i); }
```

Output :

```
1
2
3
4
5
```

Decision making & looping

The ‘C’ language provides three loop constructs for performing loop operations they are

1. The while statement
2. Do-while statement
3. The for statement

The While Statement : This type of loop is also called an entry controlled, is executed and if is true then the body of the loop is executed this process repeated until the boolean expression becomes false. Ones it becomes false the control is a transferred out the loop. The general form of the while statement is

```
            While (boolean expression)
{
                 body of the loop;
}
```

Ex :
```
                    i = 1;
                    While(I<=5)
{   printf("%d",i);
i++; }
```

In the above example the loop with be executed until the condition is false

Do-While Statement : This type of loop is also called an exist controlled loop statement.

i.e.., The boolean expression evaluated at the bottom of the loop and if it is true then body of the loop executed again and again until the boolean expression becomes false. Ones it becomes false the control is do-while statement is

```
Do
{
        body of the loop ;
}
while ( boolean expression)
```

Ex :
```
i = 1;
Do
{
printf("%d",i);
i++;
}
While(i<=5)
```

The For Statement : The for loop is another entry control led loop that provides a more concise loop control structure the general form of the for loop is

```
For ( initialisation; test condition; increment)
{
        body of the loop;
}
```

Where initialization is used to initialize some parameter that controls the looping action, 'test condition' represents if that condition is true the body of the loop is executed, otherwise the loop is terminated After evaluating information and the new value of the control variable is again tested the loop condition. If the condition is satisfied the body of the loop is again executed it this process continues until the value of the control variable false to satisfy the condition.

Ex :

```
for (I=1; I<=5; I++)
{
printf("%d",i);
        }
```

Output : 1 2 3 4 5

Jumping From The Loops :

Break Statement : The break statement can be accomplish by using to exist the loop. when break is encountered inside a loop, the loop is immediately exited and the program continues with the statement which is followed by the loop. If nested loops then the break statement inside one loop transfers the control to the next outer loop.

Ex :

```
for (I=1; I<5; I++)
{
if ( I == 4)
break;
printf("%d",i);
}
```

Output : 1 2 3

Continue Statement : The continue statement which is like break statement. Its work is to skip the present statement and continues with the next iteration of the loop .

Ex :

```
for (I=1;I<5;I++)
{
if ( I == 3)
continue;
printf("%d",i);
}
```

Output : 1 2 4 5