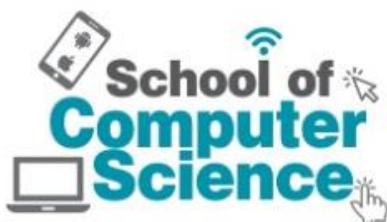


LECTURE NOTES  
ON  
**Computer Programming Using C**  
**21CS01101**

B.TECH. 1<sup>ST</sup> SEMESTER  
(COMPUTER SCIENCE AND ENGINEERING)

PREPARED BY  
DIXITA B. KAGATHARA  
ASSISTANT PROFESSOR  
[dixita.kagathara@darshan.ac.in](mailto:dixita.kagathara@darshan.ac.in)



## Table of Content

Unit-1 .....	4
Overview of C .....	4
1.1. Getting started with C.....	4
1.2. C Character set.....	4
1.3. C Tokens .....	5
1.4. Keywords.....	5
1.5. Identifiers.....	5
1.6. Constants.....	6
1.7. Variables.....	7
1.8. Data types .....	7
1.9. Operators.....	9
1.10. Input/Output Operations.....	13
1.11. Basic structure of C program.....	14
1.12. Features of C language .....	15
1.13. Arithmetic expression and evaluation.....	16
1.14. Type conversion .....	16
1.15. Operator precedence and Associativity.....	18
1.16. Maths function .....	20
1.17. Algorithm and Flowchart .....	20
.....	22
Unit-2 .....	25
Decision Making & Looping .....	25
2.1 Introduction.....	25
2.2 Decision making with if statement .....	25
2.3 Simple if statement .....	25
2.4 if-else statement .....	27
2.5 Nested if-else.....	28
2.6 if-else if ladder .....	30
2.7 Switch case statement.....	31
2.8 Conditional operator .....	33
2.9 Differentiate switch and if-else.....	34

2.10	Differentiate conditional operator and if-else.....	34
2.11	Loop and its types .....	34
2.12	while loop.....	35
2.13	do-while loop.....	36
2.14	for loop .....	37
2.15	Nested loop.....	37
2.16	Jump in to the loop .....	38

## Unit-1

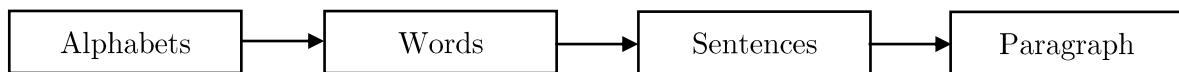
### Overview of C

---

#### 1.1. Getting started with C

- English language and C language learning are closely related.

*Steps in learning English language:*



*Steps in learning C language:*

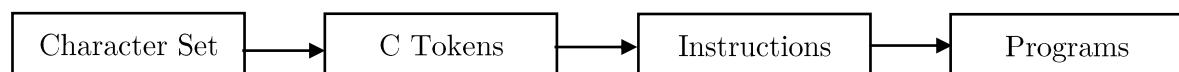


Figure 1.1.1 Steps in learning a language

- Like English, C language too has a set of rules (syntax rules) that one must follow while writing C programs.

#### 1.2. C Character set

- The characters can be used to form words, numbers and expressions.
- The characters in C are grouped into the following categories:
  - Alphabets (Letters)
  - Digits
  - Special characters
  - White spaces

Table 1.2.1 C character set

Categories of Character	
Alphabets	A, B,.....,Z a, b,.....,z
Digits	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Special symbols	~ ! # \$ ? % ^ & * ( ) _ +   \ ` - = { } [ ] : " ; < > ? , . /
White spaces	Blank space Horizontal tab Carriage return New line Form feed

### 1.3. C Tokens

- The smallest part of C program is called Token.
- C has mainly six types of tokens as shown below:

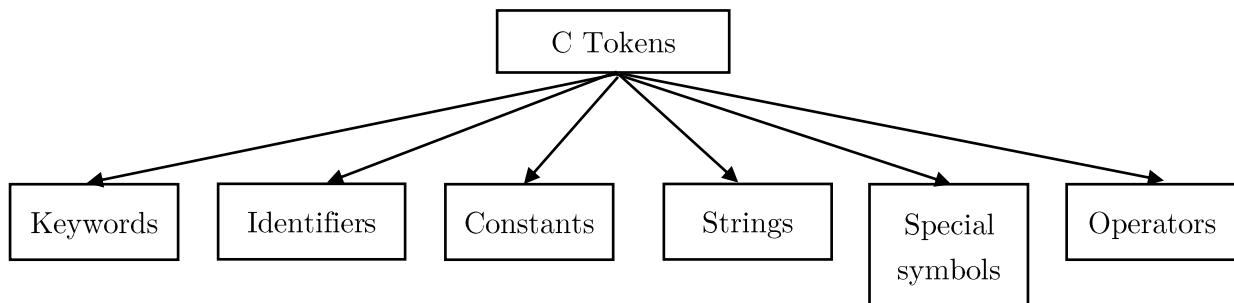


Figure 1.3.1 Tokens

### 1.4. Keywords

- Keywords are reserved words.
- Meaning of these words cannot be changed.
- Keywords serve as a basic building block for program statement.
- Keywords cannot be used as an identifier.
- List of keywords used in C are given below:

Table 1.4.1 Keywords

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

### 1.5. Identifiers

- The identifiers are user defined names used in a program for providing names to variables, arrays and functions.
- The identifiers consist of letters (uppercase, lowercase), digits and underscore.
- Rules to define identifiers:
  - First character must be an alphabet or an underscore.
  - It must consist of only alphabets (a to z & A to Z), digits (0 to 9) & underscore (\_).
  - Only first 31 characters are significant.
  - Cannot use C keyword.

- Must not contain white space.

## 1.6. Constants

- Constants in C refer to fixed values that do not change during the execution of a program.
- C supports following types of constants:

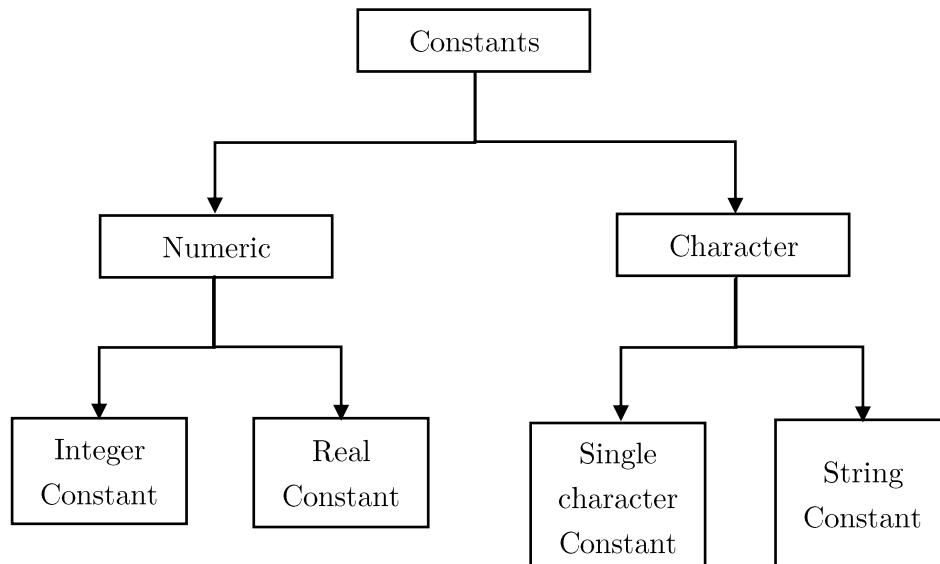


Figure 1.6.1 Types of Constants

### Integer constant:

- Integer constant is a number without decimal point and fractional part.
- There are three types of integers constant.

#### Decimal integer

- Decimal integer consists of a set of digits, 0 to 9 having optional – or + sign. No other characters are allowed like space, commas, and non-digit characters.
- Example: 123, -321, 0, +78

#### Octal integer

- Octal integer consists of any combination of digits from the set 0 to 7. Octal numbers are always preceded by 0.
- Example: 037, 0, 0551

#### Hexadecimal integer

- Hexadecimal integer consists of any combination of digits from the set 0 to 9 and A to F alphabets. It always starts with 0x or 0X. A represents 10, B represents 11... F represents 15.
- Example: 0X2A, 0x95, 0xA47C.

### Real constant:

- The number containing the fractional part is called real number. Ex: 0.0083, -0.75, +247.0, -0.75.
- A real number may also be expressed in exponential notation.
- The general form is: ***mantissa e exponent***

- Example: 215.65 can be written as 2.1565e2.

#### Single character constant:

- It contains single character enclosed within a pair of single quote mark.
- Example: '5', 'A', ';' , ' '

#### String constant:

- A string constant is a sequence of characters enclosed in double quotes.
- The characters may be letters, numbers, special characters and blank space.
- Example: "Darshan University", "Hello world", "5+3", etc...
- A character constant 'X' and string constant "X" is not equivalent.

## 1.7. Variables

- Variable is a symbolic name given to some value which can be changed.
- A variable may take different values at different times during execution.
- A variable name may consist of letters, digits and the underscore (\_) character.

Table 1.7.1 Variables

Variable name	Valid?	Remarks
First_tag	Valid	
int	Invalid	int is a keyword
Price\$	Invalid	\$ sign is not permitted
CSE Branch	Invalid	Blank space is not permitted
Int_type	Valid	Keyword may be part of variable name

## 1.8. Data types

- C language is rich in its data types.
- It determines the type and size of data associated with variables.
- ANSI C supports following data types:

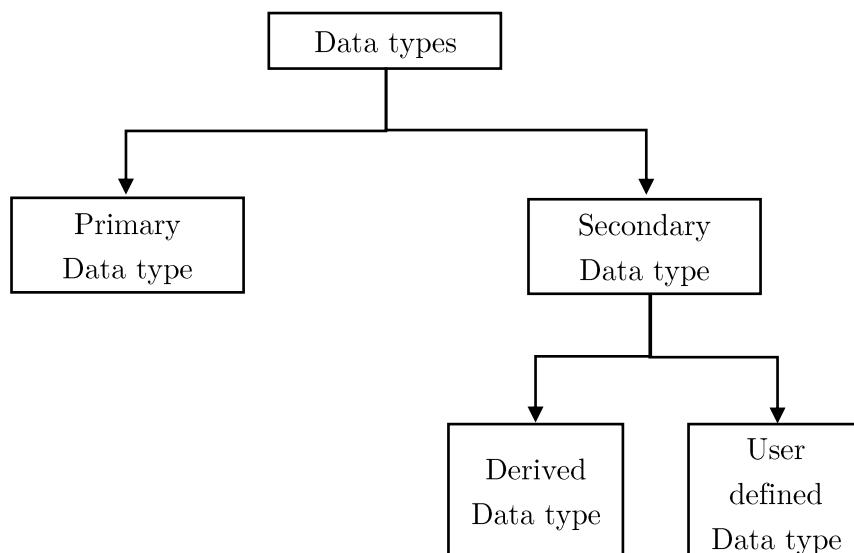


Figure 1.8.1 Data types

## Primary data types

- Primary data types are built in data types which are directly supported by machine.
- They are also known as fundamental data types.
- The fundamental data types are:
  - int
  - float
  - char
  - void

### Integer Types:

- int datatype can store integer number which is whole number without fraction part such as 10, 20, 30, etc.
- C language has three classes of integer storage namely ***short int, int and long int***.
- All of these data types have signed and unsigned forms.

### Floating point Types:

- float data type can store floating point number which represents a real number with decimal point and fractional part with 6 digits of precision such as 10.50, 155.25 etc.
- floating point numbers are defined in C by the keyword **float**.
- When the accuracy of the floating-point number is insufficient, we can use the double to define the number. The double is same as float but with longer precision up to 14 digits.

### Character Types:

- Char data type can store single character of alphabet or digit or special symbol such as 'a', '5', '\$' etc.
- Each character is assigned some integer value which is known as ASCII values.

### Void Types:

- The void type has no value therefore we cannot declare it as variable as we did in case of int or float or char.
- The void data type is used to indicate that function is not returning anything.

Table 1.8.1 Data types and it's range

Data Type	Storage Space	Value Range	Example
Char	1 bytes	-128 to 127	'a', '\$', '1', etc...
Int	2 bytes	-32768 to 32767	1, 2, 5, 9, 3, 0, etc..
Float	4 bytes	3.4E-38 to 3.4E+38	10.20, 20.30, 40.30, etc...
Double	8 bytes	3.4E-4932 to 1.1E+4932	10, 10.50, 100058, etc..

## Secondary data type

- Secondary data types are not directly supported by the machine.
- It is combination of primary data types to handle real life data in more convenient way.
- There are two types of secondary data types:
  1. Derived data types

## 2. User defined data types

**Derived data types:** Derived data type is extension of primary data type. It is built-in system and its structure cannot be changed. Examples: Array and Pointer.

- **Array:** An array is a fixed-size sequenced collection of elements of the same data type.
  - **Pointer:** Pointer is a special variable which contains memory address of another variable.
- User defined data types:** User defined data type can be created by programmer using combination of primary data type and/or derived data type. Examples: Structure, Union, Enum.
- **Structure:** Structure is a collection of logically related data items of different data types grouped together under a single name.
  - **Union:** Union is like a structure, except that each element shares the common memory.
  - **Enum:** Enum is used to assign names to integral constants, the names make a program easy to read and maintain.

## 1.9. Operators

- C supports a rich set of built-in operators.
- An operator is a symbol that tells the computer to perform certain mathematical or logical operation.
- C operators can be classified in following categories:
  - Arithmetic operator
  - Relational operator
  - Logical operator
  - Assignment operator
  - Increment/Decrement operator
  - Conditional operator
  - Bitwise operator
  - Special operator

### Arithmetic operators (+, -, \*, /, %)

- Arithmetic operators are used for mathematical calculation.
- It can operate on any built-in data type allowed in C.
- A modulo division operator (%) cannot be used on floating data type.
- C does not have an operator for exponentiation.

Table 1.9.1 Arithmetic operators

Operator	Meaning	Written as	Description	Integer Arithmetic (Ex: a=14,b=4)
+	Addition	a + b	Addition of a and b	$a + b = 18$
-	Subtraction	a - b	Subtraction of b from a	$a - b = 10$

*	Multiplication	a * b	Multiplication of a and b	a * b = 56
/	Division	a / b	Division of a by b (Decimal part truncated)	a / b = 3
%	Modulo division (remainder)	a % b	Modulo of a by b	a % b = 2

- If X,Y, and Z are floats, then:

$$X = 6.0/7.0 = 0.857143$$

$$Y = 1.0/3.0 = 0.333333$$

$$Z = -2.0/3.0 = -0.666667$$

### Relational operators (<, <=, >, >=, ==, !=)

- Relational operators are used to check relationship between two operands.
- If the relation is true, it returns 1; if the relation is false, it returns value 0.
- An expression containing a relational operator is termed as a relational expression.
- Relational expressions are used in decision statements such as if, for, while, etc...

Table 1.9.2 Relational operators

Operator	Meaning	Written as	Description	Example
<	is less than	a < b	a is less than b	6 < 4 is evaluated to 0 (false)
<=	is less than or equal to	a <= b	a is less than or equal to b	6 <= 4 is evaluated to 0 (false)
>	is greater than	a > b	a is greater than b	6 > 4 is evaluated to 1 (true)
>=	is greater than or equal to	a >= b	a is greater than or equal to b	6 >= 4 is evaluated to 1 (true)
==	is equal to	a == b	a is equal to b	6 == 4 is evaluated to 0 (false)
!=	is not equal to	a!=b	a is not equal to b	6!=4 is evaluated to 1 (true)

### Logical operators (&&, ||, !)

- An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false.

Table 1.9.3 Truth table

a	b	a&&b	a  b
0	0	0	0

0	1	0	1
1	0	0	1
1	1	1	1

- Logical operators are used to test more than one condition and make decisions.

*Table 1.9.4 Logical operators*

Operator	Meaning	Description	Example
<b>&amp;&amp;</b>	logical AND	Both non zero then true, either is zero then false	<code>a&gt;b &amp;&amp; x==10</code>
<b>  </b>	logical OR	Both zero then false, either is non zero then true	<code>Number&lt;0    number&gt;100</code>
<b>!</b>	Logical NOT	True only if the operand is 0	<code>!(a)</code>

### Assignment operators (`+=`, `-=`, `*=`, `/=`)

- Assignment operators (`=`) is used to assign the result of an expression to a variable.
- Assignment operator stores a value in memory.
- C also supports shorthand assignment operators which simplify operation with assignment.

*Table 1.9.5 Assignment operators*

Operator	Meaning	Same as
<code>=</code>	Assigns value of right side to left side	
<code>+=</code>	<code>a += 1</code>	<code>a = a + 1</code>
<code>-=</code>	<code>a -= 1</code>	<code>a = a - 1</code>
<code>*=</code>	<code>a *= 1</code>	<code>a = a * 1</code>
<code>/=</code>	<code>a /= 1</code>	<code>a = a / 1</code>
<code>%=</code>	<code>a %= 1</code>	<code>a = a % 1</code>

- The advantages of using shorthand operators are:

- What appears on the left-hand side need not be repeated and therefore it becomes easier to write.
- The statement is more concise.
- The statement is more efficient.

### Increment and decrement operators (`++`, `--`)

- Increment (`++`) operator used to increase the value of the variable by one.
- Decrement (`--`) operator used to decrease the value of the variable by one.
- These two operators are unary operators, meaning they only operate on a single operand.

*Table 1.9.6 Increment/Decrement operators*

Operator	Example	Explanation	Output

Pre increment operator (++x)	x=10; p=++x;	First increment value of x by one then assign.	x will be 11 p will be 11
Post increment operator (x++)	x=10; p=x++;	First assign value of x then increment value.	x will be 11 p will be 10

### Conditional operators (?:)

- A conditional operator is known as ternary operator.

**Syntax:** exp1 ? exp2 : exp3

- where exp1, exp2 and exp3 are expressions.

#### Working of the ?: Operator

- exp1 is evaluated first.
- if exp1 is true (nonzero), then
  - exp2 is evaluated and its value becomes the value of the expression
- If exp1 is false (zero), then
  - exp3 is evaluated and its value becomes the value of the expression
- Example:**

```
m=2, n=3;
1.      r=(m>n) ? m : n;    Value of r will be 3
2.      r=(m<n) ? m : n;    Value of r will be 2
```

### Bitwise operators (&, |, ^, <<, >>)

- Bitwise operators are used to perform operation bit by bit.
- Bitwise operators may not be applied to float or double.

*Table 1.9.7 Bitwise operators*

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	int a=8, b=6, c; c = a & b; printf("Output = %d", c); Output: 0
	Binary OR Operator copies a bit if it exists in either operand.	int a=8, b=6, c; c = a   b; printf("Output = %d", c); Output: 14
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	int a=8,b=6,c; c=a^b; printf("%d",c);

		Output: 14
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	<pre>int a=8, b; b = a &lt;&lt; 1; printf("Output = %d", b); Output: 16 (multiplying a by a power of two)</pre>
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	<pre>int a=8, b; b = a &gt;&gt; 1; printf("Output = %d", b); Output: 4 (dividing a by a power of two)</pre>

## Special operators

Table 1.9.8 Special operators

Operator	Description
&	Address operator, it is used to determine address of the variable.
*	Pointer operator, it is used to declare pointer variable and to get value from it.
,	Comma operator. It is used to link the related expressions together.
sizeof	It returns the number of bytes the operand occupies.
.	Member selection operator, used in structure.
→	Member selection operator, used in pointer to structure.

## 1.10. Input/Output Operations

- The **stdio.h** header file provides built in functions for reading (`scanf()`) data from input devices (keyboard) and writing (`printf()`) formatted data to output devices (monitor).

### **scanf():**

- `scanf()` is a library function that reads data with specified format from standard input (keyboard).
- Syntax: `scanf("control string", &variable1, &variable2...);`
- Example: `scanf("%d", &number);`
- First argument is specification of format and other arguments are pointer variables to store data.
- The function returns the total number of items successfully read.
- The value is assigned to variable name.

### **printf():**

- `printf()` is a library function that prints formatted data to standard output, generally monitor.
- Syntax: `printf("control string", variable1, variable2....);`
- Example: `printf("Your marks are %d", mark);`
- The control string indicated how many arguments follow and what their types are.

## 1.11. Basic structure of C program

Table 1.11.1 Basic structure of C program

Basic structure of C Program	Example
Documentation Section	// This program is to find area of circle
Link Section	#include<stdio.h>
Definition Section	#define PI 3.14
Global Declaration Section	int i; float areaofcircle(float);
main() { Declaration Part Executable Part }	void main() { //Declaration Part float r,area;  //Executable Part scanf("%f",&r); area=areaofcircle(r); printf("Area of Circle is :- %f",area); }
Subprogram Section (User Defined Section) Function1() Function2()	float areaofcircle(float r) { return PI * r * r; }

### 1.11.1. Comments

- Comments makes code more readable and also used to explain the code.
- Comments are not part of the program.
- There are two types of comments:
  - **Single line Comments:** it starts with two forward slashes (//).
  - Any line between // and the end of the line is considered as comment and thus ignored by the compiler (will not be executed).
  - Example: // This is a comment

- **Multi-line Comments:** it starts with /\* and ends with \*/.
- Any line between /\* and \*/ is considered as multi-line comment and thus ignored by the compiler.
- Example: /\* This is  
a multi-line comment \*/

### 1.11.2. Header files

- Header files contain the set of predefined standard library functions.
- The “#include” pre-processing directive is used to include the header files with “.h” extension in the program.
- Whenever we require the definition of a function, then we simply include that header file in which function is declared.
- There are two types of header files defined in a program:
  1. **System defined header file:**
    - The header file which is predefined is known as a system defined header file.
  2. **User-defined header file:**
    - The header file which is defined by the user is known as a user-defined header file.
- Some system headers files are given below:

*Table 1.11.2.1 Header files*

Name	Description
stdio.h	Input/Output Functions
conio.h	Console input/output
ctype.h	Character Handling Functions
math.h	Mathematics Functions
stdlib.h	General Utility Functions
string.h	String Functions
time.h	Date and Time Functions
errno.h	Testing error codes

## 1.12. Features of C language

- Modularity
- Extensibility
- Elegant syntax
- Case sensitive
- Less memory required
- The standard library concepts
- The portability of the compiler
- A powerful and varied range of operators

- Ready access to the hardware when needed

### 1.13. Arithmetic expression and evaluation

- An arithmetic expression is a combination of variables, constants and operators arranged as per the syntax of the language.
- C can handle any complex mathematical expression.

*Table 1.13.1 Expressions*

Arithmetic expression	C expression
$a \times b - c$	$a*b-c$
$(m+n)(x+y)$	$(m+n)*(x+y)$
$\frac{ab}{c}$	$(a*b)/c$
$3x^2+2x+1$	$(3*x*x)+(2*x)+1$

#### Evaluation of expression

- Expressions are evaluated using an assignment statement of the form:

*Variable=expression;*

- The expression is evaluated first and result then assign to left hand side variable.

#### Examples:

```
X=a*b-c;
Y=b/c*a;
Z=a-b/c+d;
```

### 1.14. Type conversion

- C permits mixing of constants and variables of different types in an expression.
- C automatically converts any intermediate values to the proper type so that the expression can be evaluated without losing any significance.
- This automatic conversion is known as ***implicit type conversion***.
- The lower type is automatically converted to the higher type before the operation proceeds.  
The result is higher type.

#### Example: Implicit Type Conversion

```
int a = 20;
double b = 20.5;
printf("%lf", a + b);
Output: 40.500000
```

- C uses the rule that, in all expressions except assignments, any implicit type conversions are made from lower size type to a higher size type as shown below:

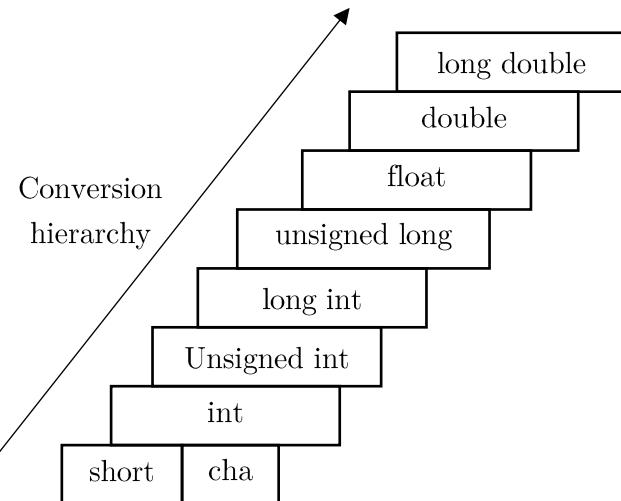


Figure 1.14.1 Conversion Hierarchy

- The final result of an expression is converted to the type of variable on the left of the assignment sign before assigning value to it.
- Following changes are introduced during final assignment.
  - float to int causes truncation of fractional part.
  - double to float causes rounding of digits.
  - long int to int causes dropping of the excess higher order bits.

### Explicit Conversion

- Sometimes we want to force a type conversion in a way that is different from automatic conversion.
- The process of such a local conversion or casting is known as ***explicit casting***.
- The general form of cast is: **(type-name) expression**
- Where type-name is one of the standard C data types. The expression may be constant, variable, or an expression.
- Some examples of casts and their actions shown below:

Table 1.14.1 Use of casts

Example	Action
X=(int)7.5	7.5 is converted to integer.
A=(int)21.3/(int)4.5	Evaluated as 21/5 and result would be 5.
B=(double)sum/n	Division is done in floating point mode.
Y=(int)(a+b)	The result of a+b is converted to integer.
Z=(int)a+b	a is converted to integer and then added to b.
P=cos((double)x)	Converts x to double before using it.

### Example: Explicit Type Conversion

```

double a = 4.5, b = 4.6, c = 4.9;
int result = (int)a + (int)b + (int)c;
printf("result = %d", result);
Output: 12
  
```

## 1.15. Operator precedence and Associativity

### Precedence

- Precedence of an operator is its priority in an expression for evaluation.
- There is distinct level of precedence.
- The operator at the higher level of precedence is evaluated first.
- Precedence rule decides the order in which different operators are applied.
- $5 + 3 * 2$  is calculated as  $5 + (3 * 2)$ , giving 11, and not as  $(5 + 3) * 2$ , giving 16.
- We say that the multiplication operator (\*) has higher "precedence" or "priority" than the addition operator (+), so the multiplication must be performed first.

### Associativity

- The operator at the same precedence level is evaluated either from 'left' to 'right' or 'right' to 'left' depending on the associativity of an operator.
- Associativity rule decides the order in which multiple occurrences of the same level operators are applied.
- Operator associativity is why the expression  $8 - 3 - 2$  is calculated as  $(8 - 3) - 2$ , giving 3, and not as  $8 - (3 - 2)$ , giving 7.
- We say that the subtraction operator (-) is "left associative", so the left subtraction must be performed first. When we can't decide by operator precedence alone in which order to calculate an expression, we must use associativity.
- Following table provides a complete list of operators, their precedence level, and their rule of association.
- Rank 1 indicates highest precedence level and 15 is the lowest.

Table 1.15.1 Summary of C Operators

Operator	Description	Associativity	Rank
()	Function Call	Left to Right	1
[]	Array Element Reference		
+, -	Unary Plus, Unary minus	Right to Left	2
++, --	Increment, Decrement		
!	Logical Negation		
~	Ones Complement		
*	Pointer Reference (Indirection)		
&	Address		
sizeof	Size of an object		
*	multiplication	Left to Right	3
/	Division		
%	Modulus		
+	Addition	Left to Right	4
-	Subtraction		

<<	Left Shift	Left to Right	5
>>	Right Shift		
<	Less than	Left to Right	6
<=	Less than or equal to		
>	Greater than		
>=	Greater than or equal to		
==	Equality	Left to Right	7
!=	Inequality		
&	Bitwise AND	Left to Right	8
^	Bitwise XOR	Left to Right	9
	Bitwise OR	Left to Right	10
&&	Logical AND	Left to Right	11
	Logical OR	Left to Right	12
?:	Conditional Expression	Right to Left	13
==, *=, /=,	Assignment Operators	Right to Left	14
%=, +=, -=,			
&=, ^=,  =,			
<<=, >>=			
,	Comma Operator	Left to Right	15

### Precedence of Arithmetic operators

- An arithmetic expression without parenthesis will be evaluated from left to right using the rules of precedence of operator.

**Example:  $x=a-b/3+c*2-1$**

When  $a=9$ ,  $b=12$  and  $c=3$  the statement becomes,

$$x=9-12/3+3*2-1$$

#### First Pass

$$\text{Step-1: } x=9-12/3+3*2-1$$

$$\text{Step-2: } x=9-4+3*2-1$$

$$\text{Step-3: } x=9-4+6-1$$

#### Second Pass

$$\text{Step-4: } x=5+6-1$$

$$\text{Step-5: } x=11-1$$

$$\text{Step-6: } x=10$$

## 1.16. Maths function

- Most of the C compiler supports basic math functions.
- To use math functions in a program, we should include **math.h** header file in the beginning of the program.
- List of some standard math functions given below:

*Table 1.16.1 Math functions*

Function	Meaning
<b>Trigonometric</b>	
acos(x)	Arc cosine of x
asin(x)	Arc sine of x
atan(x)	Arc tangent of x
atan2(x,y)	Arc tangent of x/y
cos(x)	Cosine of x
sin(x)	Sine of x
tan(x)	Tangent of x
<b>Hyperbolic</b>	
cosh(x)	Hyperbolic cosine of x
sinh(x)	Hyperbolic sine of x
tanh(x)	Hyperbolic tangent of x
<b>Other functions</b>	
ceil(x)	x rounded up to the nearest integer
exp(x)	e to the x power ( $e^x$ )
fabs(x)	Absolute value of x
floor(x)	x rounded down to the nearest integer
fmod(x,y)	Reminder of x/y
log(x)	Natural log of x, $x>0$
log10(x)	Base 10 log of x, $x>0$
pow(x,y)	x to the power of y ( $x^y$ )
sqrt(x)	Square root of x, $x\geq 0$

## 1.17. Algorithm and Flowchart

### Algorithm

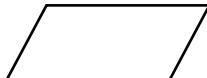
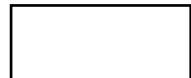
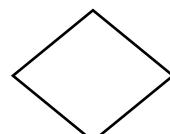
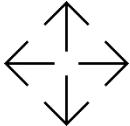
- Algorithm is a finite sequence of well-defined steps for solving a mathematical or computational problem.
- An algorithm takes some inputs, execute some finite number of steps and gives an output.
- It is written in the natural language like English.

- It is difficult to debug and to show branching and looping.
- Example:** write an algorithm to check whether number is Positive or Negative.  
 Step 1: Read no.  
 Step 2: If no is greater than equal zero, go to step 4.  
 Step 3: Print no is a negative number, go to step 5.  
 Step 4: Print no is a positive number.  
 Step 5: Stop.

### Flowchart

- Flowchart is a pictorial or graphical representation of a problem.
- It is drawn using various symbols given below:

*Table 1.12.1 Flowchart symbols*

Notation	Description
	Start / Stop
	Input / Output (Read / Print)
	Process
	Decision making
	Subroutine
	Flow / Directions

- It is easy to understand and to show branching and looping.
- The flowchart symbols are linked together with arrows showing the process flow direction.

- **Example:** Draw a flowchart to check whether number is Positive or Negative.

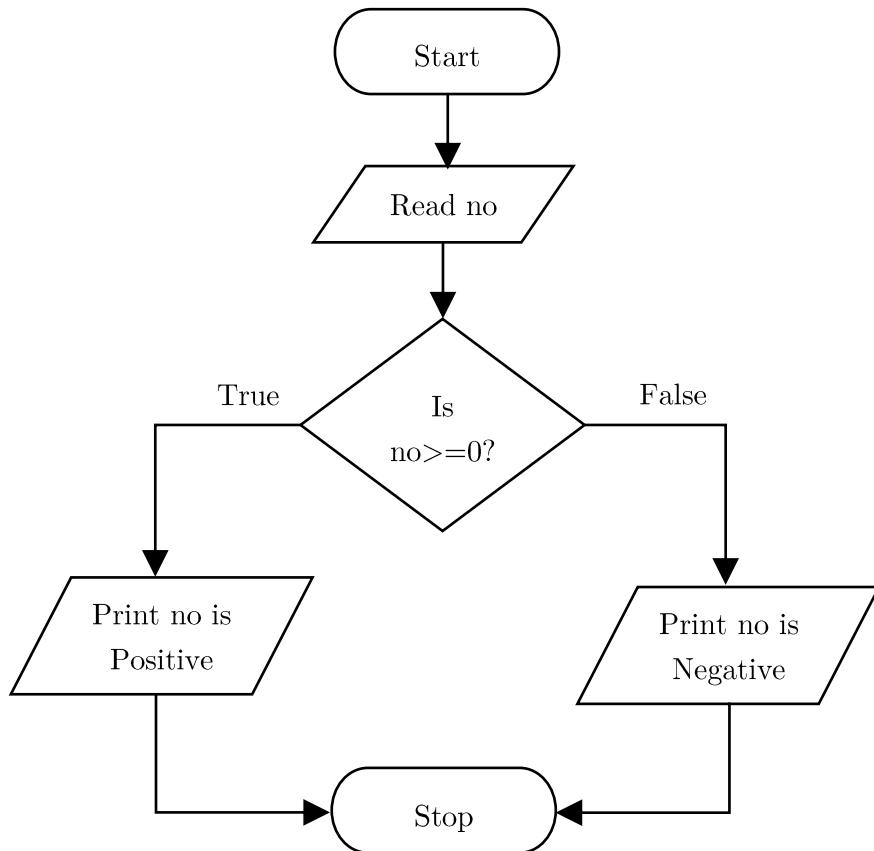


Figure 1.17.1 flowchart to check positive or negative number

### Difference between Flowchart and Algorithm

Table 1.13.2 Difference between Flowchart & Algorithm

Flowchart	Algorithm
Flowchart is a pictorial or graphical representation of a program.	Algorithm is a finite sequence of well-defined steps for solving a problem.
It is drawn using various symbols.	It is written in the natural language like English.
Easy to understand.	Difficult to understand.
Easy to show branching and looping.	Difficult to show branching and looping.
Flowchart for big problem is impractical.	Algorithm can be written for any problem.

Write an Algorithm and draw a Flowchart to find largest number from given 3 no.

Flowchart

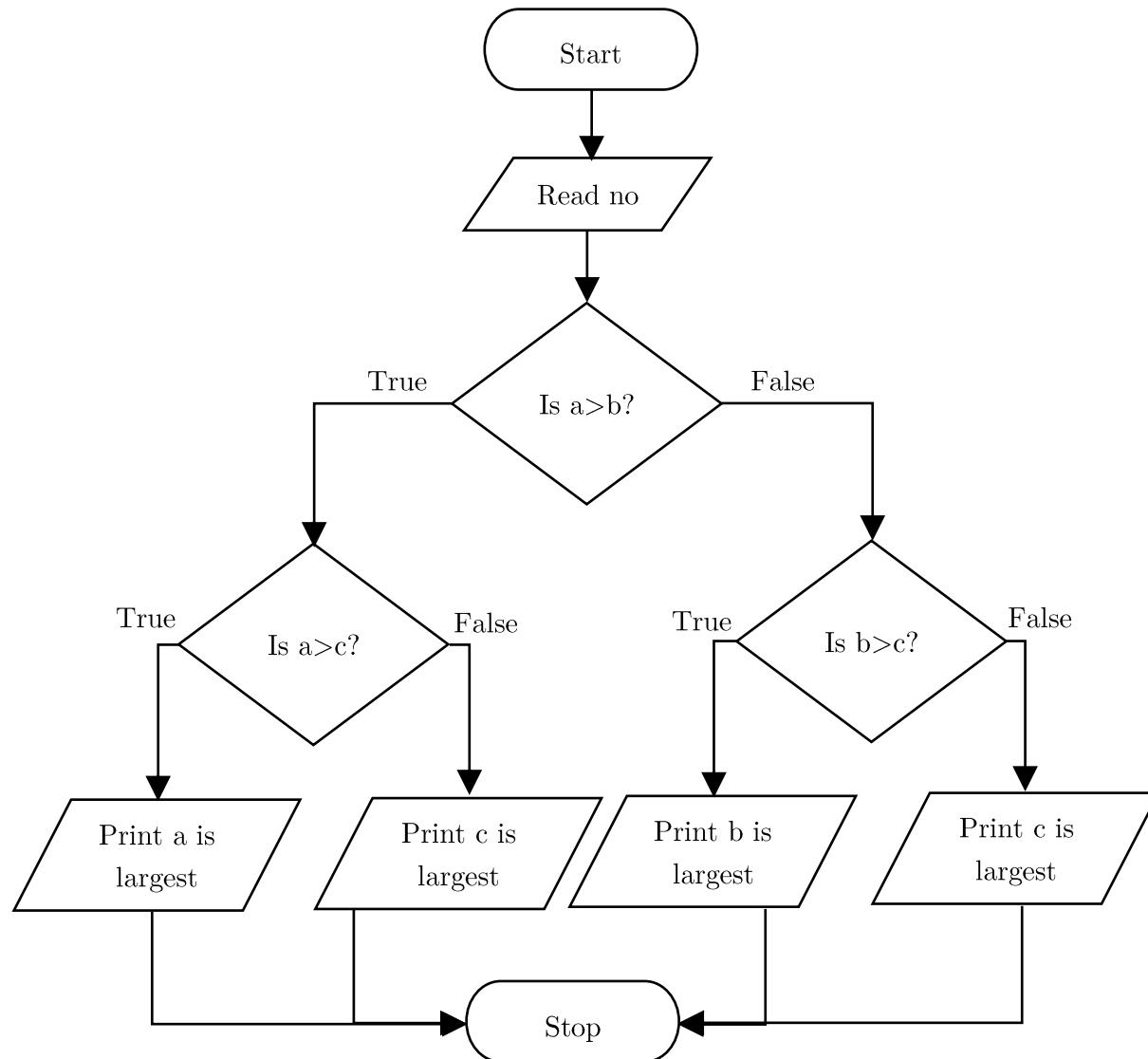


Figure 1.17.2 flowchart to find largest number from three number

### Algorithm

- Step 1: Read a, b, c.
- Step 2: If a>b, go to step 5.
- Step 3: If b>c, go to step 8.
- Step 4: Print c is largest number, go to step 9.
- Step 5: If a>c, go to step 7.
- Step 6: Print c is largest number, go to step 9.
- Step 7: Print a is largest number, go to step 9.
- Step 8: Print b is largest number.

Step 9: Stop.

**Write an Algorithm and draw a flowchart to print 1 to 10.**

**Flowchart**

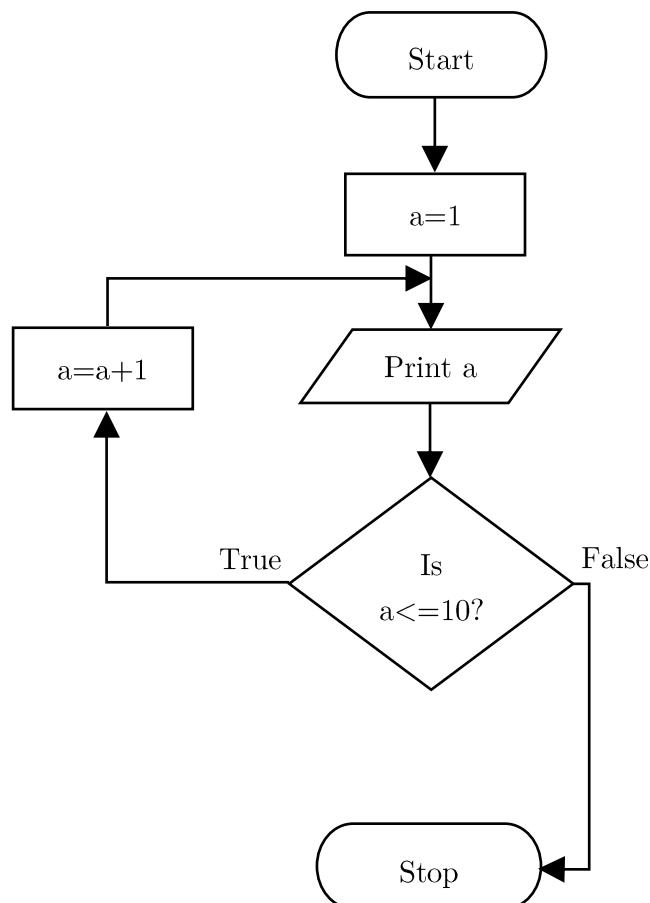


Figure 1.17.3 flowchart to print 1 to 10

### Algorithm

Step 1: Initialize a to 1.

Step 2: Print a.

Step 3: Repeat step 2 until  $a \leq 10$ .

    Step 3.1:  $a = a + 1$ .

Step 4: Stop.

## Unit-2

# Decision Making & Looping

---

### 2.1 Introduction

- C program is a set of statements which are normally executed sequentially.
- However, in practice, we have a number of situations where we have to change the order of execution of statement based on certain conditions.
- Decision Making statements (control statements) are used to control the flow of program.
- It evaluates condition or logical expression first and based on its result (either true or false), the control is transferred to particular statement.
- If result is true then it takes one path otherwise it takes another path.
- Decision making statements in C language are:
  1. If statement
  2. Switch statement
  3. Conditional operator statement
  4. goto statement

### 2.2 Decision making with if statement

- The if statement is powerful decision-making statement and is used to control the flow of execution of statements.
- It is two-way decision-making statement.

Syntax:

*if(test expression)*

- It allows computer to evaluate first and then, depending on whether the value of the expression (relation or condition) is ‘true’ or ‘false’, it transfers the control to a particular statement.
- It has two paths, one for true condition and other for the false condition.
- The if statement may be implemented in following different forms:
  1. Simple if statement
  2. if...else statement
  3. Nested if...else statement
  4. else if ladder statement

### 2.3 Simple if statement

- The general form of simple if statement is:

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

- It evaluates test expression first and based on its result, the control is transferred to the particular statement.
- If the test expression is true, the statement-block will be executed; otherwise, the statement-block will be skipped and the execution will jump to the statement-x.
- When the test expression is true, both the statement-block and the statement-x executed in sequence.

**Flowchart:**

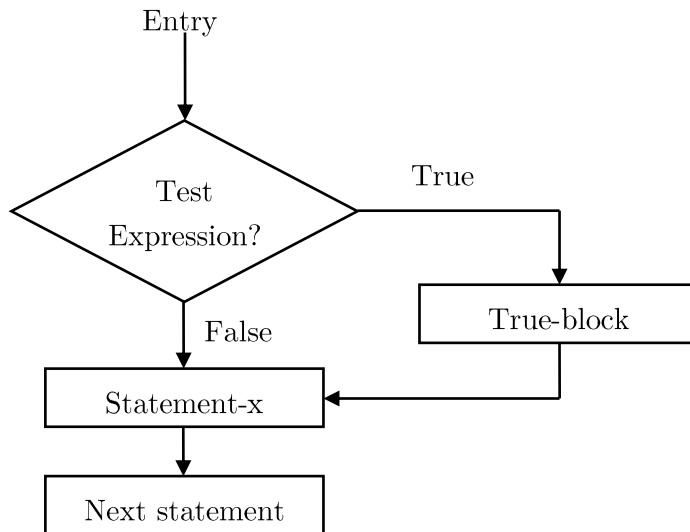


Figure 2.3.1 simple if statement flowchart

**Example:** Check whether given number is positive or negative using simple if statement.

```

#include<stdio.h>
void main()
{
    int a;
    printf("Enter Number:");
    scanf("%d",&a);
    if(a >= 0)
    {
        printf("Positive Number");
    }
    if(a < 0)
    {
        printf("Negative Number");
    }
}
  
```

## 2.4 if-else statement

- If supports statements only for true part, while if..else support statements for true part as well false part.
- if...else is two branch decision making statement.
- The general form of if..else statement is:

```

if (test expression)
{
    True - block statement(s);
}
else
{
    False - block statement(s);
}
statement-x;

```

- If the test expression is true, then the True-block statements(s) are executed; otherwise, false-block statement(s) are executed.
- Either true-block or false-block will be executed, not both.

**Flowchart:**

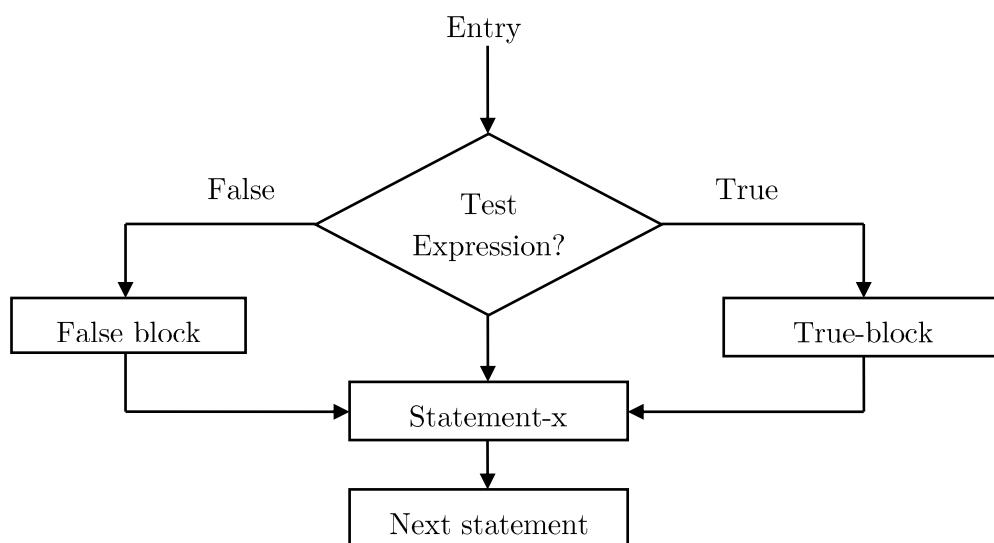


Figure 2.4.1 if-else statement flowchart

**Example:** Check whether given number is positive or negative using if-else statement.

```

#include<stdio.h>
void main()
{
    int a;
    printf("Enter Number:");
    scanf("%d",&a);
    if(a >= 0)

```

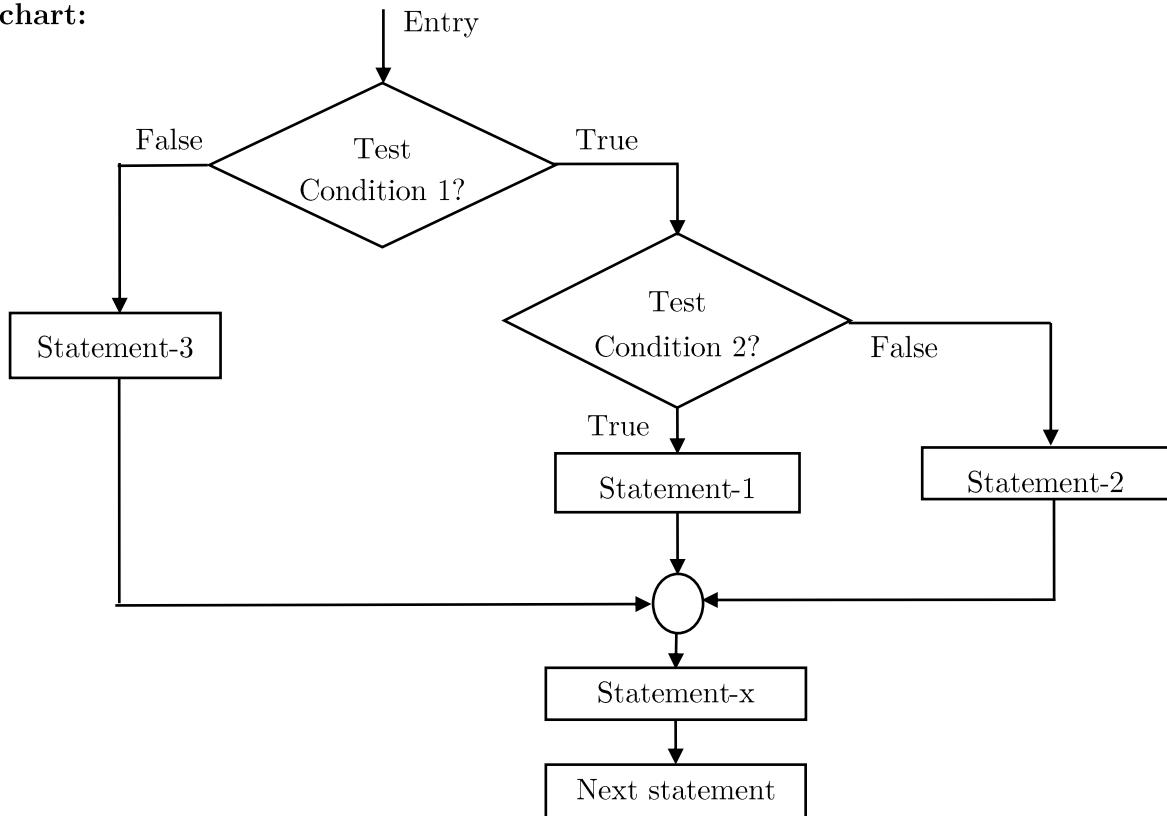
```
{
    printf("Positive Number");
}
else
{
    printf("Negative Number");
}
}
```

## 2.5 Nested if-else

- When a series of decisions are involved, we may have to use more than one if...else statement in nested form.

```
if(test-condition-1)
{
    if(test-condition-2)
    {
        Statement-1;
    }
    else
    {
        Statement-2;
    }
}
else
{
    Statement-3;
}
Statement-x;
```

- If test-condition-1 is false then Statement-3 will be executed and then control will be transferred to statement-x.
- If test-condition-1 is true then test-condition-2 is evaluated. If it is true then Statement-1 will be executed, if it is false then Statement-2 will be executed and then control will be transferred to statement-x.

**Flowchart:**

*Figure 2.5.1. Nested if-else statement flowchart*

**Example:** Find maximum number from given three numbers.

```

#include<stdio.h>
void main()
{
    int a,b,c;
    printf("Enter value of a, b, c:");
    scanf("%d%d%d", &a, &b, &c);
    if(a>b)
    {
        if(a>c)
            printf("a is max");
        else
            printf("c is max");
    }
    else
    {
        if(b>c)
            printf("b is max");
        else
    }
  
```

```

        printf("c is max");
    }
}

```

## 2.6 if-else if ladder

- It's another way of putting ifs together when multipath decisions are involved.
- It takes following general form:

```

if(condition-1)
    statement-1;
else if(condition-2)
    statement-2;
else if(condition-N)
    statement-N;
else
    default-statement;
statement-x;

```

- This construct is known as the else if ladder.
- The conditions are evaluated from top to down.
- As soon as a true condition is found, the statement associated with it is executed and control is transferred to the statement-x (skipping the rest of the ladder).
- When all the n condition become false, the final else containing the default- statement will be executed.

**Flowchart:**

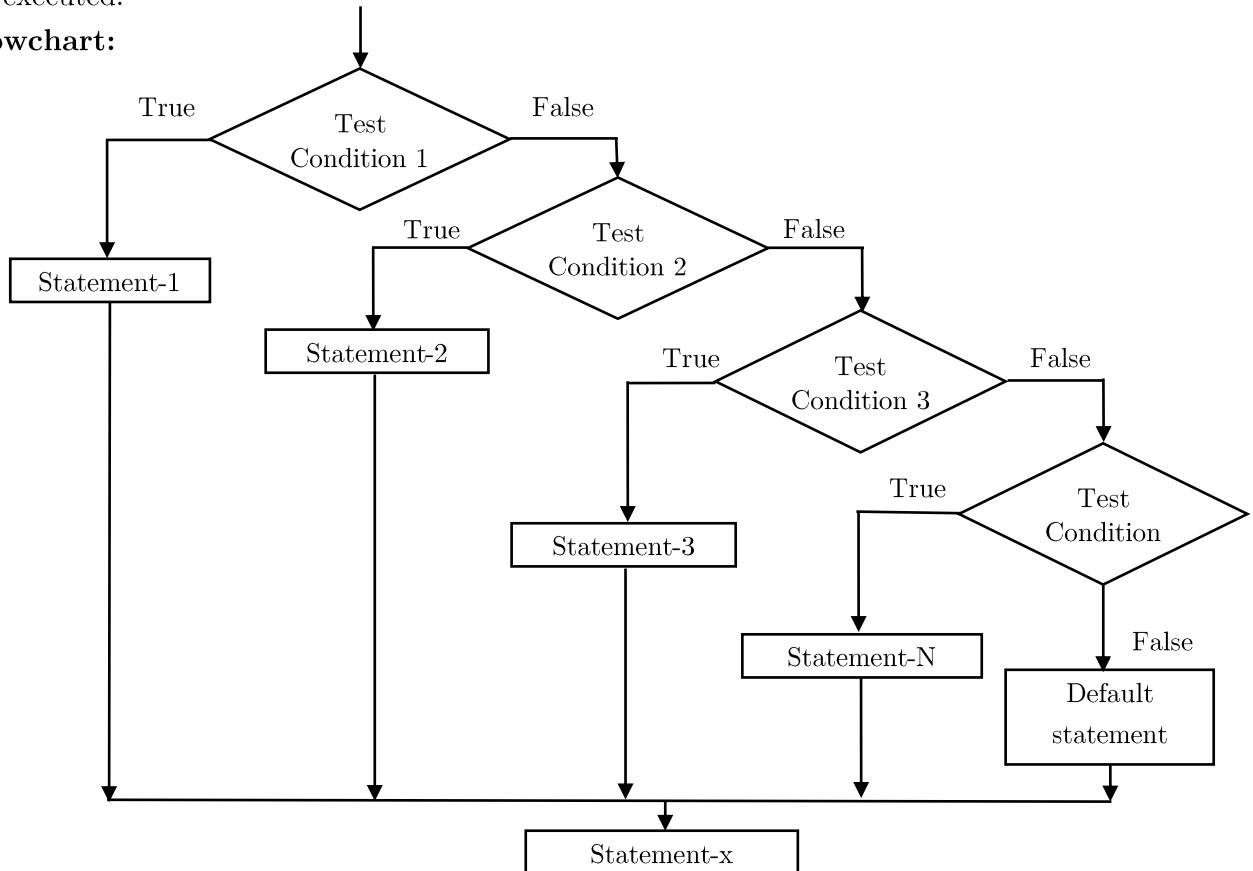


Figure 2.6.1 else-if ladder flowchart

**Example:** Check whether given number is positive, negative or zero using else if ladder.

```
#include<stdio.h>
void main()
{
    int a;
    printf("Enter Number:");
    scanf("%d",&a);
    if(a > 0)
        printf("Positive Number");
    else if(a==0)
        printf("Zero");
    else
        printf("Negative Number");
}
```

## 2.7 Switch case statement

- The switch statement allows to execute one code block among many alternatives.
- Switch is built-in multi-way decision statement.
- It works similar to if...else..if ladder.
- The expression is evaluated once and compared with the values of each case.
- If there is a match, the corresponding statements after the matching case are executed.
- If there is no match, the default statements are executed.
- If we do not use break, all statements after the matching label are executed.
- The default clause inside the switch statement is optional.

```
switch (expression)
{
    case constant1:
        // statements
        break;
    case constant2:
        // statements
        break;
    .
    .
    .
    default:
        // default statements
}
```

**Flowchart:**

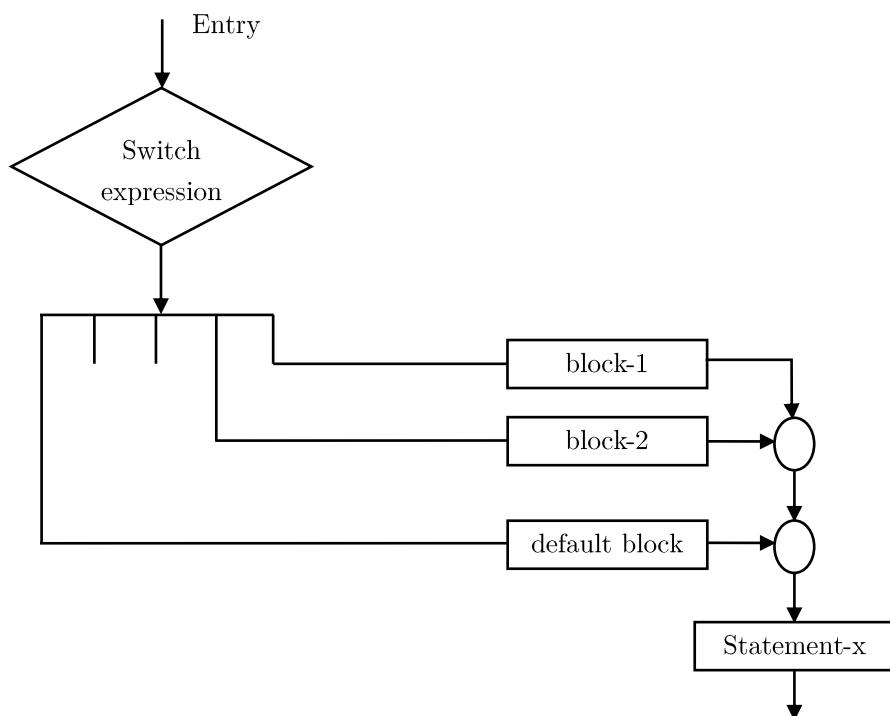


Figure 2.7.1 Switch-case statement flowchart

**Example:** Print Day name based on day number.

```

void main()
{
    int day;
    printf("Enter day number(1-7):");
    scanf("%d",&day);
    switch(day)
    {
        case 1:
            printf("Sunday");
            break;
        case 2:
            printf("Monday");
            break;
        case 3:
            printf("Tuesday");
            break;
        case 4:
            printf("Wednesday");
            break;
        case 5:
            printf("Thursday");
    }
  
```

```

    break;
  case 6:
    printf("Friday");
    break;
  case 7:
    printf("Saturday");
    break;
  default:
    printf("Invalid input");
    break;
}
}
  
```

### Rules for switch statement

- The switch expression must be integral type. Float or other data types are not allowed.
- Case labels must be constant or constant expression.
- Case labels must be unique. No two labels can have the same value.
- Case labels must end with colons.
- The break statement is optional.
- The default case statement is optional. There can be at most one default label.
- It is permitted to nest switch statement.

## 2.8 Conditional operator

- A conditional operator is known as ternary operator.
- The conditional operator works similar to the if-else.

**Syntax:**  $exp1 ? exp2 : exp3$

- where  $exp1$ ,  $exp2$  and  $exp3$  are expressions.

### Working of the ?: Operator

- $exp1$  is evaluated first
- if  $exp1$  is true (nonzero), then
  - $exp2$  is evaluated and its value becomes the value of the expression
- If  $exp1$  is false (zero), then
  - $exp3$  is evaluated and its value becomes the value of the expression

### Example:

consider  $m=2$ ,  $n=3$ ;

1.  $r=(m>n) ? m : n$ ; Value of  $r$  will be 3
2.  $r=(m<n) ? m : n$ ; Value of  $r$  will be 2

## 2.9 Differentiate switch and if-else

Table 2.9.1 Difference between if-else & switch-case

if-else	Switch-case
If statement is used to select among two alternatives.	The switch statement is used to select among multiple alternatives.
If can have values based on constraints.	Switch can have values based on user choice.
If implements Linear search.	Switch implements Binary search.
Float, double, char, int and other data types can be used in if condition.	Only int and char data types can be used in switch block.

## 2.10 Differentiate conditional operator and if-else

Table 2.10.1 Difference between conditional operator & if-else

Conditional operator	if-else
It is a programming statement.	It is a programming block.
It is not suitable for executing a function or a several statements	In that case, if else is more appropriate.
Nested ternary operator is not readable and cannot be debugged easily.	If else is more readable and easier to debug in case of issue.
Example: Max=(a>b)?a:b;	if(a>b) Max=a; else Max=b;

## 2.11 Loop and its types

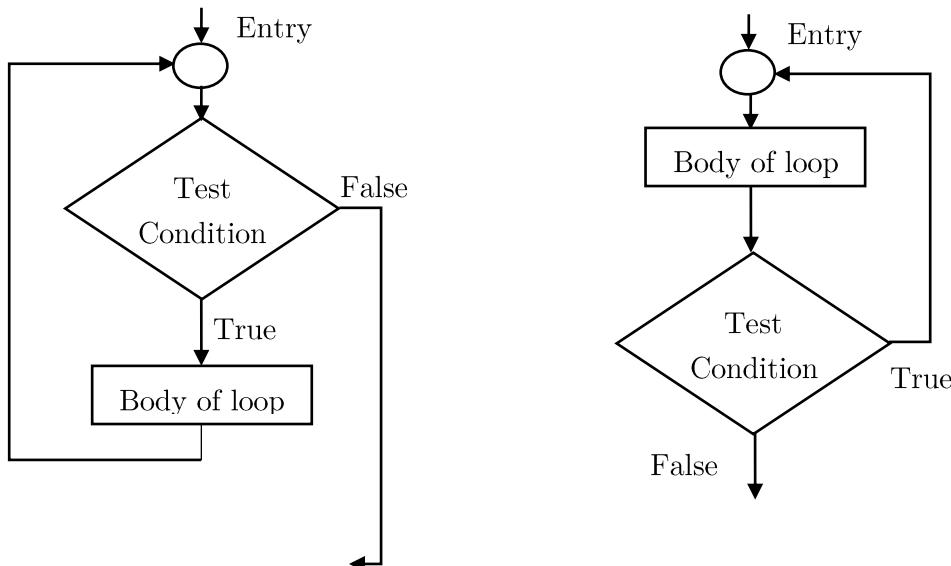
- Loop is used to execute the block of code several times according to the condition given in the loop.
- A loop mainly consists of two parts:
  - Body of the loop
  - Control statement
- The control statements test certain conditions and then directs the repeated execution of the statements.
- Depending on the position of the control statement in the body of loop, it is classified as the entry control loop or as an exit control loop.

Table 2.11.1. Difference between Entry & Exit control loop

Entry control loop	Exit control loop
Entry control loop checks condition before start of the loop execution.	Exit control loop first executes body of the loop and checks condition at last.

Body of loop may or may not be executed  
at all.

for, while are example of entry control loop.  
do...while is example of exit control loop.



- A looping process, consist of a following steps:
  - Initialization
  - Test condition
  - Execution of the body of loop
  - Incrementing/ updating the condition variable

## 2.12 while loop

- The simplest of all looping structure is while statement.
- The general format of the while statement is:

```

initialization;
while (test condition)
{
    body of the loop;
    increment or decrement;
}
  
```

- While loop is also known as entry control loop because first control-statement is executed and if it is true then only body of the loop will be executed.
- After the execution of the body, the test condition is once again evaluated and if it is true, the body is executed once again.
- This process is repeated till the test condition is true. When it becomes false, the control is transferred out of the loop.

- On exit, the program continues with the statements immediately after the body of the loop.
- The body of loop may have one or more statements.

**Example:** To print first 10 positive integer numbers

```
void main()
{
  int i;
  i = 1;           // initialization of i
  while(i <= 10)    // condition checking
  {
    printf("\t%d",i); // statement execution
    i++;            // increment of control variable
  }
}
```

## 2.13 do-while loop

- Sometime, it might be necessary to execute the body of loop before the test is performed.
- Such situation can be handled with the help of the do statement.
- The general format of the do...while statement is:

```
initialization;
do
{
  body of the loop;
  increment or decrement;
}
while(test-condition);
```

- On reaching the do statement, the program evaluates the body of loop first time.
- At the end of the loop, the test condition in the while statement evaluated.
- If the condition is true, the program continues to evaluate the body of loop once again.
- This process continues as long as the condition is true.
- Since the test condition evaluated at the bottom of the loop, the do...while construct provides the exit-control loop and therefore the body of loop is always executed at least once.

**Example:** To print first 10 positive integer numbers

```
void main()
{
  int i;
  i = 1;           // initialization of i
  do
  {
    printf("\t%d",i); // statement execution
    i++;            // increment of control variable
  } while(i <= 10); // condition checking
```

}

## 2.14 for loop

- for loop is another entry-controlled loop that provides a more concise loop control structure.
- The general form of the for loop is:

```

for (initialization; test condition; increment/decrement)
{
    body of the loop;
}
```

- When the control enters for loop, the variables used in for loop is initialized with the starting value such as i=0, count=0. Initialization part will be executed exactly one time.
- Then it is checked with the given test condition. The test condition is relational expression. If the given condition is satisfied, the control enters into the body of the loop. If condition is false then it will exit from the loop.
- After the completion of the execution of the loop, the control is transferred back to the increment part of the loop. The control variable is incremented using an assignment statement such as i++.
- If new value of the control variable satisfies the loop condition, then the body of the loop is again executed. This process goes on till the control variable fails to satisfy the condition.

**Example:** find the sum of the first 10 positive integer numbers

```

void main()
{
    int i, sum=0;                                //declare variable int sum=0;
    for(i=1; i <= 10; i++)                      // for loop
    {
        sum = sum + i ;                         // add the value of i and store it to sum
    }
    printf("%d", sum);
}
```

## 2.15 Nested loop

- Nesting of loops allows the looping of statements inside another loop.
- The nesting may continue up to any desired level.
- It takes the following forms:

```

Outer loop
{
    Inner loop
{
    // inner loop statements.
}
// outer loop statements.
```

}

- Outer loop and inner loop can be for loop, while loop or do..while loop.

**Example:**

```
#include <stdio.h>
void main()
{
    int i, j, rows;
    printf("Enter the number of rows: ");
    scanf("%d", &rows);
    for (i = 1; i <= rows; ++i)
    {
        for (j = 1; j <= i; ++j)
        {
            printf("* ");
        }
        printf("\n");
    }
}
```

Output:

```
Enter the number of rows: 3
*
*
*
*
```

## 2.16 Jump in to the loop

- Sometimes, when executing a loop, it becomes desirable to skip a part of the loop or to leave the loop as soon as a certain condition occurs.
- C permits a jump from one statement to another statement within a loop as well as a jump out of a loop.

### Jumping out of the loop

- An early exit from a loop can be achieved using:
  - break
  - goto

#### 2.16.1 break

- When, a break statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement immediately following the loop.
- When the loop is nested, the break would only exit from the loop containing it. The break will exit only in a single loop.

**Example:** Read and sum numbers till -1 is entered

```
void main()
{
```

```

int i, num=0;
float sum=0,average;
printf("Input the marks, -1 to end\n");
while(1)
{
    scanf("%d",&i);
    if(i===-1)
        break;
    sum+=i;
}
printf("%d", sum);
}
  
```

### 2.16.2 goto

- A goto statement can transfer the control to any place in the program.
- It can be used to jump within the loop as well as exiting from the loop.
- The goto statement is marked by label statement. Label statement can be used anywhere in the function above or below the goto statement.
- Generally, goto should be avoided because its usage results in less efficient code, complicated logic and difficult to debug.

**Example:** Following program prints 10,9,8,7,6,5,4,3,2,1

```

void main ()
{
    int n=10;
    loop:
        printf("%d,",n);
        n--;
        if (n>0)
            goto loop;
}
  
```

### 2.16.3 Skipping a part of the loop

- The continue statement can be used to skip the rest of the body of an iterative loop.
- The continue statement tells the compiler, “SKIP THE FOLLOWING STATEMENTS AND CONTINUE WITH THE NEXT ITERATION”.
- Continue statement causes the control to go directly to the test condition and then continue to iteration process.

**Example:** Find sum of 5 positive integers. If a negative number is entered then skip it.

```

void main()
{
    int i=1, num, sum=0;
    for (i = 0; i < 5; i++)
    {
        ...
}
  
```

```
scanf("%d", &num);
if(num < 0)
    continue; // starts with the beginning of the loop
    sum+=num;
}
}
```