A. **Operators, Expressions, Precedence and Associativity, Evaluating Expressions, Type Conversion Statements, Simple Programs.**

B. **Selection & Making Decisions: Logical Data and Operators, Two Way Selection, Multiway Selection, More Standard Functions.**

C. **Repetition: Concept of Loop, Pre-test and Post-test Loops, Initialization and Updating, Event and Counter Controlled Loops, Loops in C, Other Statements Related to Looping.**

A. Operators

Operators, Expressions, Precedence, and Associativity, Evaluating Expressions, Type Conversion Statements, Simple Programs.

c

Special Operators (sizeof, pointer, address, typecasting, comma, dot, etc.)

The following are the special operators in the C programming language.

sizeof( ) Operator

This operator is used to find the size of the memory (in bytes) allocated for a variable.

Syntax: sizeof(variableName);

Example: sizeof(A); ⇒ the result is 2 bytes if A is an integer

```
//sizeof( ) Operator
#include <stdio.h>
int main()
{int a;
 char b;
 float c;
 double d;
 printf("Storage size of data type int is :%d \n",sizeof(a));
 printf("Storage size of data type char is :%d \n",sizeof(b));
 printf("Storage size of data type float is :%d \n",sizeof(c));
```

```
 printf("Storage size of data type double is :%d\n",sizeof(d));
 return 0;
}
```

Pointer operator (*)

This operator is used to define pointer variables in the C programming language.

Address operator (&)

This operator is used to get the memory address of a variable in the C programming language.

Example:

```
//Pointer and Address Operators
#include <stdio.h>
int main()
{int *ptr, q;
 q = 10;
 ptr = &q;
 printf("\n q value: %d", *ptr);
 printf("\n q address: %p", ptr);
 return 0;
}
/*Output
 q value: 10
 q address: 0061FF18
*/
```

Type Conversion in C

Converting value of one data type into another data type is called type conversion. There are 2 types of type conversions in C.

1. Implicit Conversion - The C compiler automatically converts the data type of a value into another data type AUTOMATICALLY.

   Example:

   ```
   int b;
   ```

```
b = 3.5;  //3.5 first converts to integer 3 and then assigns to b
```

Example:

```
#include<stdio.h>
int main() {
  // create a double variable
  double value = 7125.17;
  printf("Double Value: %.2lf\n", value);

  // convert double value to integer
  int number;
  number = value;
  printf("Integer Value: %d", number);
  return 0;
}
// Output: 7125
```

Here, the C compiler automatically converts the double value 7125.17 to integer value 7125.

Since the conversion is happening automatically, this type of conversion is called implicit type conversion.

2. **Explicit Conversion -** We have to MANUALLY convert one data type into another data type. We need to use **Typecasting operator ( )** for explicit conversion.

Typecasting operator: ( )

Typecasting in C is the process of converting one data type to another data type by the programmer using the casting operator ( ). This is also called **Explicit** conversion because we force the type conversion using this operator.

Syntax for Explicit conversion:

(type) variable or value

Example:

```
int x;
float y;
y = (float) x;
```

Example:

```
int main() {
  // create an integer variable
  int number = 97;
  printf("Integer Value: %d\n", number);

  // (char) converts number to character
  char alphabet = (char) number;
  printf("Character Value: %c", alphabet);

  return 0;
}
/* OUTPUT
Integer Value: 97
Character Value: a  */
```

Here,

- **(char)** - explicitly converts a **number into character**
- **number** - value that is to be converted to char type

Comma operator (,)

Comma operator is used for 2 purposes:

1. **Separator** - when we declare multiple variables, we use comma as separator.

   int a, b, c;

   int m=7, n=10;

2. **Operator** - when we assign multiple number of values to a variable, we use comma.

   x = 10, 20, 30, 40;

   y = (10, 20, 30, 40);

Example:

```
//Comma as Seperator and Operator
#include <stdio.h>
int main()
{
//comma as seperator
```

```
int x,y;

//comma as operator
x = 10,20,30;  //x=10 because = has higher priority than ,
y = (10,20,30);  //x=30 because () has higher priority than =

printf("x= %d, y= %d\n",x,y);
return 0;
}
//Output: x= 10, y= 30
/*Note: int a = 1,2,3;   //Wrong, compile time error to initialize with
comma separator while declaring a variable   */
```

Dot operator (.)

This operator is used to access members of a structure or union.

Example:

```
#include <stdio.h>
struct Pair {
    int first, second;
};

int main(void) {
    struct Pair p;
    p.first = 10;
    p.second = 20;
    printf("%d %d\n", p.first, p.second);
}
/* Output:     10 20      */
```

What are Expressions in C?

An expression is a sequence of operators (symbols) and operands (constants or variables) that represents a specific value. An Expression always reduces to a single value.

Operators are symbols that perform tasks such as arithmetic operations, logical operations, conditional operations, etc.

Operands are the constant or variable values on which the operators perform the task. The operand can be a direct value or variable or address of memory location.

- **Simple Expression** - contains only one operator.
  - 2 + 5
  - - a
- **Complex Expression** - contains more than one operator
  - 2 + 5 * 7 (we reduce it to a series of simple expressions)
  - First, we calculate the expression 5 * 7 to 35 and
  - Then, we calculate the expression 2 + 35 to 37 as a result.
- Expressions return values as a boolean, an integer, or any other data type of C.
- Expression may consist of other expressions; in this case, first inner expressions are calculated, then the overall expression will be evaluated.

6 Types of Simple Expressions in C

1. **Primary Expressions**
2. **Postfix Expressions**
3. **Prefix Expressions**
4. **Unary Expressions**
5. **Binary Expressions**
6. **Ternary Expressions**

1. Primary Expressions:

- Names - A name is any identifier for a variable, function, or any other object in the language.

  Ex: x y10 cost

- Constants - A constant is a piece of data whose values can't change during the execution of a program.

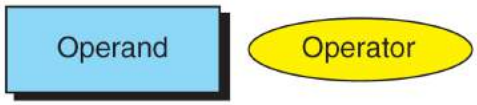
  Ex: 10 47.35 'A' "Hello"

- Parenthetical Expressions - Any value or expression enclosed in parentheses must be reduced to a single value.

  Ex: (5 * 7 + 2) (a = 45 + b +10)

2. Postfix Expressions:

- The operator comes after the operand.

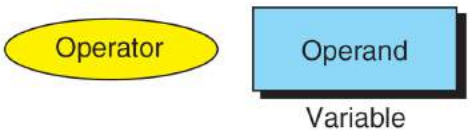


a++ (same as a = a + 1)

The operand in a postfix expression must be a variable.

3. Prefix Expressions:

- The operator comes before the operand.



++a (same as a = a + 1)

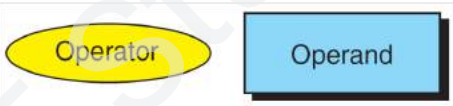
The operand of a prefix expression must be a variable.

Note:

- For **a++** : First, current value of **a** is used in the expression; Then, **a** will be incremented
- For **++a** : First, **a** will be incremented; Then, new value of **a** will be used in the expression

4. Unary Expressions:

- Consists of One Operand and One Operator



Expression	Contents of a Before *and* After Expression	Expression Value
+a	3	+3
-a	3	-3
+a	-5	-5
-a	-5	+5

Example:

```
 int a = 5;
 printf("\n Unary expression - : %d",-a);
//Output: Unary expression - : -5
```

Example:

```
int n;
printf("%d",sizeof(n));  //unary because sizeof operator needs 1 variable
```

5. Binary Expressions:

- Formed by Operand-Operator-Operand combination

Example:

```
 int m1 = 50, m2 = 60;
 printf("\n Total marks = %d", m1 + m2);
//Output: 110
```

6. Ternary Expressions:

- Ternary expression needs 1 Condition, 2 Operands, and 2 Operators
- Formed by **Condition-Operator-Operand-Operator-Operand**

Example:

```
int x = 55;
(x>=25) ? printf("Pass") : printf("Fail");
//Output: Pass
```

7. Assignment Expressions:

- Evaluates the Operand on the right-hand side of the Operator (=) and place the value in the variable on the left.

Example - Simple Assignment

a = 5 b = x + 1 i = i + 1

8. Compound Assignment:

- A compound assignment is a shorthand format for a simple assignment

Compound Expression	Equivalent Simple Expression
x *= expression	x = x * expression
x /= expression	x = x / expression
x %= expression	x = x % expression
x += expression	x = x + expression
x -= expression	x = x - expression

Example - Compound Assignment

x += 7	is same as	x = x + 7
x -= 3	is same as	x = x - 3
x *= 5	is same as	x = x * 5
x *= y + 3	is same as	x = x * (y + 3)
x /= 2	is same as	x = x / 5
x %= 3	is same as	x = x % 3

What are Precedence and Associativity?

Precedence (priority) is used to find the **order of different operators** to be evaluated in a single statement.

```
2 + 3 * 4        // * evaluates first
2 + 12           // + evaluates next
14
```

Associativity is used to find the **order of operators with same precedence** to be evaluated in a single statement.

//left to right associativity

```
3 * 8 / 4 * 5    //both * and / has same precedence or priority
 24  / 4 * 5     //left to right associativity
    6  * 5
      30
```

//right to left associativity

```
a = b = c = 0    // all = have same precedence or priority
```

For commonly used arithmetic calculations, you may use the simple **BODMAS** order of priority to easily remember the **precedence** or priority of evaluation.

B - **B**racket

O - **O**f Squareroot or **O**f Exponent

DM - **D**ivision or **M**ultiplication (same priority)
AS - **A**ddition or **S**ubtraction (same priority)

C has about 45 operators. The following table shows the Precedence and Associativity of all opetators.

Operator Precedence (Priority) and Associativity Chart

Precedence	Operator	Description	Associativity
1	( )	Parentheses (function call)	left-to-right
	[ ]	Brackets (array subscript)	left-to-right
	.	Member selection via object name	left-to-right
	->	Member selection via a pointer	left-to-right
	a ++ / a - -	Postfix increment/decrement (a is a variable)	left-to-right
2	++ a / - - a	Prefix increment/decrement (a is a variable)	right-to-left
	+ / -	Unary plus / minus	right-to-left
	! ~	Logical negation / bitwise complement	right-to-left
	(type)	Cast (convert value to temporary value of type)	right-to-left
	*	Dereference	right-to-left
	&	Address (of operand)	right-to-left
	sizeof	Determine size in bytes on this implementation	right-to-left
3	*, /, %	Multiplication/division/modulus	left-to-right
4	+ / -	Addition/subtraction	left-to-right
5	<< , >>	Bitwise shift left, Bitwise shift right	left-to-right
6	< , <=	Relational less than/less than or equal to	left-to-right
	> , >=	Relational greater than/greater than or equal to	left-to-right

7	== , !=	Relational is equal to/is not equal to	left-to-right
8	&	Bitwise AND	left-to-right
9	^	Bitwise exclusive OR	left-to-right
10	|	Bitwise inclusive OR	left-to-right
11	&&	Logical AND	left-to-right
12	||	Logical OR	left-to-right
13	? :	Ternary conditional	right-to-left
14	=	Assignment	right-to-left
	+= , -=	Addition/subtraction assignment	right-to-left
	*= , /=	Multiplication/division assignment	right-to-left
	%= , &=	Modulus/bitwise AND assignment	right-to-left
	^= , |=	Bitwise exclusive/inclusive OR assignment	right-to-left
	<> =	Bitwise shift left/right assignment	right-to-left
15	,	expression separator	left-to-right

Side Effects

A side effect is an action that results from the evaluation of an expression.

Expressions with Side Effects

```
x = 4           // x receives value 4
x = x + 4       // x receives value 7
y = ++x * 2     // y receives 16 and ALSO x value changes to 8
```

Expressions without Side Effects

```
a=4, b=4, c=5
result = a * 4 + b / 2 - c * b  //values of a, b, c, d do not change
```

B. Selection & Making Decisions

Logical Data and Operators, Two Way Selection, Multiway Selection, More Standard Functions.

Logical Data and Operators in making decisions

The conditions play an important role in the selection and decision-making process in C language. The results of conditions control the flow of execution of the programs. We use 2 types of operators to write conditions in C programs.

1. Relational Operators (== != < <= > >=)
2. Logical Operators (&& || !)

Both of these operators result in either TRUE or FALSE.

- TRUE is a non-zero value (ex: 1).
- FALSE is a 0 value. (Ex: 0)

The Relational and Logical Operators are discussed in detail with examples in our previous section. So, please refer to the previous section to review how these work.

Summary of relational operators:

operator	Meaning	examples
>	greater than	6 > 3 --> true 3 > 6 --> false
<	less than	2 < 5 --> true 5 < 2 --> false
>=	greater than or equal to	4 >= 4 --> true 5 >= 4 3 >= 4 --> false
<=	less than or equal to	2 <= 2 --> true 1 <=2 3 <= 2 --> false
==	equivalence, same as. Note the use of two equal signs, not 1. 1 equal sign is an assignment, 2 equal sign is equivalence	1 == 1 --> true 2 == 1 --> false
!=	not equivalent, different	4 != 3 --> true 3 != 3 --> false

Summary of Logical Operators:

Operator	Meaning	Example
&&	Logical AND - Returns TRUE if all conditions are TRUE otherwise returns FALSE	10 < 5 && 12 > 10 is FALSE
||	Logical OR - Returns FALSE if all conditions are FALSE otherwise returns TRUE	10 < 5 || 12 > 10 is TRUE
!	Logical NOT - Returns TRUE if condition is FALSE and returns FALSE if the condition is TRUE	!(10 < 5 && 12 > 10) is TRUE

Note1: We use Logical operators to Join 2 or more conditions with Relational operators

Note2: If the result of a condition is zero, the expression is FALSE. If the condition results in non-zero, then it is TRUE.

What is a Selection Statement in C?

Selection Statements in C are used to make decisions based on the results of conditions. The program statements normally execute sequentially. If you put some condition for a block of statements, the execution flow may change based on the result evaluated by the condition. This process is called decision-making in 'C.'

These **Selection Statements** are also called,

- **Conditional Statements**
- **Flow-Control Statements**
- **Decision Making Statements**

The conditional statements are possible with the help of the following 3 selection types:

Type	**Single Selection**	**Two-Way Selection**	**Multi-Way Selection**
Command	**if** statement	**if - else** statement	Nested **if - else** statements **else - if** Ladder statements **switch - case** statements

Single Selection in C ("if" statement)

The basic method to perform selection in C is to use the "if" statement. "if" statement is responsible for modifying the flow of execution of a program. "if" statement is always used with a condition. The condition is evaluated first before executing any statement inside the body of "if".

- The condition evaluates to either TRUE or FALSE.
- True is a non-zero value, and
- False is a zero value.

The if statement allows you to do something if a condition is TRUE, and do nothing if the condition is FALSE.

Syntax:

```
if (condition){
   //do the code in here if the condition is true
}
```

- "if" statement is always used with a condition.
- First, the condition is evaluated for TRUE or FALSE
- TRUE, then the "if" block statement is executed
- FALSE, then the "if" block skipped and continues the flow to the outside "if" block

➢ **One statement** within "if" block **does NOT need** to be enclosed in curly brackets **{ }.**

Example: (one statement in single selection "if" block)

```
#include<stdio.h>
int main()
{
    int num1=1;
    int num2=2;
    if(num1 < num2)      //condition check
        printf("num1 is smaller than num2");
    return 0;
}
```

Example: (one statement in single selection "if" block)

```
int x = 30;
if (x == 35)  // False
  x = x+ 1;  // this is skipped because the condition is false

printf("%d",x); // x still has the original value 30


/* Output
30
*/
```

Example: (one statement in single selection "if" block)

```
int measured = 1;
if (measured)
  printf("Done with measurement \n");

/* Output
Done with measurement  (because "if" condition is non-zero 1 [TRUE])
*/
```

Many statements in single selection "if" condition

An "if" condition may also have many statements.

➢ **Many statements** within the "if" block **need** to be enclosed in curly braces { **}.**

Example: (many statements in single selection "if" block)

```
int a=10, b=20, c;
if (a<b)
  { a = b + 5;
    c = a * 2;
  }
printf("%d", c);  //prints 60
    return 0;
}
```

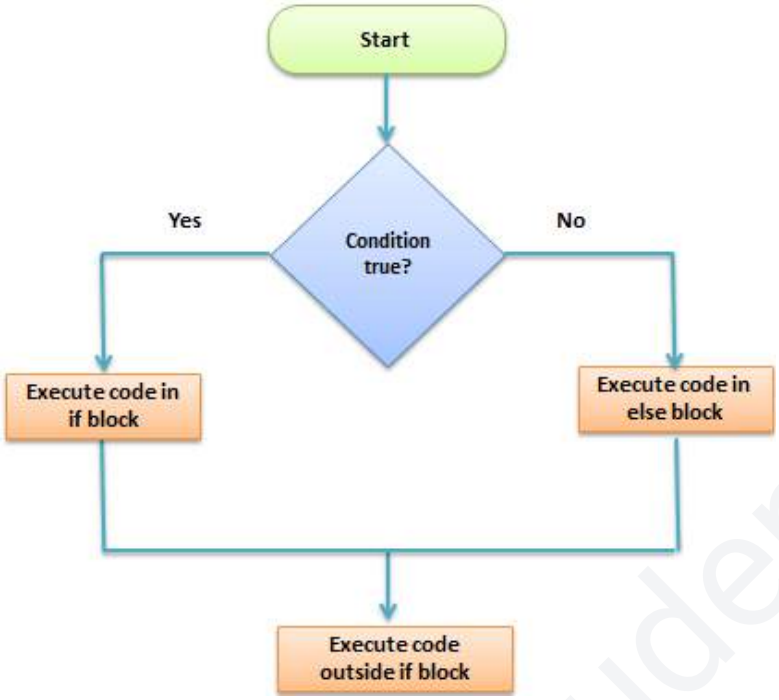
Two Way Selection (if-else statement)

Description: [if-else statement]

- If the condition is TRUE, then the true ("if") block of statements will be executed.
- If the condition is FALSE, then the false ("else") block of statements will be executed.

- After the execution of either "if" or "else" block statements, the control will be automatically transferred to the statements outside the **"if-else"** block.

The flow chart of the "if-else" block is as follows:



Syntax:

The " **if-else** " statement is an extended version of **" if "**. The syntax of "if-else" is as follows:

```
if (condition)
{
    True block of statements
    statement 1;
    statement 2;
    ...
    statement n;
}
else
{
    False block of statements
    statement 1;
    statement 2;
    ...
```

```
    statement n;
}
Statements outside if-else block
```

Example1: [if-else statement, two-way selection]

```
#include<stdio.h>
#include<conio.h>
void main()
{   int a=40, b=20;
    clrscr();
    if(a==b)
    {
        printf("a and b are same");
    }
    else
    {
        printf("a and b are not the same");
    }
getch();
}
// Output:        a and b are not the same
```

Example2: [if-else statement, two-way selection]

```
//Find Pass or Fail based on pass marks of 35
#include<stdio.h>
int main()
{   int total=70;

    if(total>35)
        printf("Passed");
    else
        printf("Failed");

    return 0;
}
/*Output   : Passed   */
```

Example3: [if-else statement, two-way selection]

```
/* Aim: Program to Check whether the given number is Even or Odd.
```

```
Even number: an integer exactly divisible by 2. Ex: 0, 8, -24
Odd number: an integer not exactly divisible by 2. Ex: 1, 7, -11, 15 */
#include <stdio.h>
int main() {
    int num;
    printf("Enter an integer: ");
    scanf("%d", &num);
    // true if num is perfectly divisible by 2
    if(num % 2 == 0)
        printf("%d is even.", num);
    else
        printf("%d is odd.", num);
     return 0;
}
/* Output
Enter an integer: 6
6 is even.

Enter an integer: 7
7 is odd.
*/
```

Multiway Selection

When a series of decisions is required, the multi-way selection statements are used.

There are 3 types of multi-way selection statements

1. **Nested " if-else " statements**
2. **" else-if " Ladder statements**
3. **" switch-case " statements**

1. **nested if-else statements**

Nesting means using one "**if-else"** construct within another "**if-else"** construct. Use nested "if-else" when you need to decide more within the parent "if" condition.

Example: (Nested if-else statements, multi-way selection)

```
#include<stdio.h>
int main()
{   int result=75;
    if(result >= 35)
    {
```

```
        if(result==35)
        {
            printf("Just Passed with %d \n",result);
        }
        else
        {
            printf("Passed with %d, which is more than 35",result);
        }
    }
    else
    {
        printf("Failed with %d, which is less than 35", result);
    }
    return 0;
}
/* Output
Passed with 75, which is more than 35     */
```

Note: The above program checks if a number is equal or greater than 35 and prints the result using nested if-else construct.

2. else-if ladder

"if-else-if" ladder is used when multiple paths of decisions are required. Use the "else-if" ladder when you need to decide a series of decisions after each of the previous "if" conditions.

Description:

- This is called **"else-if" ladder** or **nested "if-else-if"** statements
- The conditions are evaluated from top to bottom.
- If any condition is true, the statement associated with that condition is executed.
- When all the conditions are false, then the last default **"else"** statement is executed.

Syntax:

```
if (condition 1)
{
   statement1;
} else if (condition 2)
{
   Statement2;
```

```
} else if (condition n)
{
   Statement n;
} else
{
   Default statement;
}
Statement in the main program
```

Example1: ["else-if" ladder statements, multi-way selection]

```
#include<stdio.h>
int main()
{
    int marks=75;
    if(marks>=75){
        printf("First class");
    }
    else if(marks>=65){
        printf("Second class");
    }
    else if(marks>=55){
        printf("Third class");
    }
    else if(marks>=35){
        printf("Just pass");
    }
    else {
        printf("Failed");
    }
    return 0;
}
```

Example2: ["else-if" ladder statements, multi-way selection]

Aim: Find whether the given year is a Leap Year?

Conditions:

- The century year is a leap year only if it is perfectly divisible by 400.

- Century years (years ending with 00 or divisible by 100) that are not divisible by 400 are not leap years.
- A leap year is exactly divisible by 4 except for century years (years ending with 00).

For example,

1900 is not a leap year

1999 is not a leap year

2000 is the leap year

2004 is the leap year

2012 is the leap year

```c
#include <stdio.h>
int main() {
   int year;
   printf("Enter a year nnnn: ");
   scanf("%d", &year);

   // leap year if perfectly divisible by 400
   if (year % 400 == 0) {
      printf("%d is a leap year.", year);
   }

   // not a leap year if divisible by 100
   // but not divisible by 400
   else if (year % 100 == 0) {
      printf("%d is not a leap year.", year);
   }

   // leap year if not divisible by 100
   // but is divisible by 4
   else if (year % 4 == 0) {
      printf("%d is a leap year.", year);
   }

   // all other years are not leap years
   else {
```

```
        printf("%d is not a leap year.", year);
    }

    return 0;
}

/* Output1:
Enter a year nnnn: 1900
1900 is not a leap year.

    Output2:
Enter a year nnnn: 2012
2012 is a leap year.
*/
```


3. "switch-case" statement

Switch statement in C tests the value of an expression and compares it with multiple cases. Once a case value is matched, a block of statements associated with that particular case is executed.

The "switch" statement is one of the decision control statements of the C language. It is primarily used when the user has to make a decision among many alternatives or choices.

In a switch statement,

- the "case" value can be an integral constant ("char" and "int" type),
- the "case" value cannot be a "float" type or a string value,
- duplicate "case" values are not allowed.

Syntax:

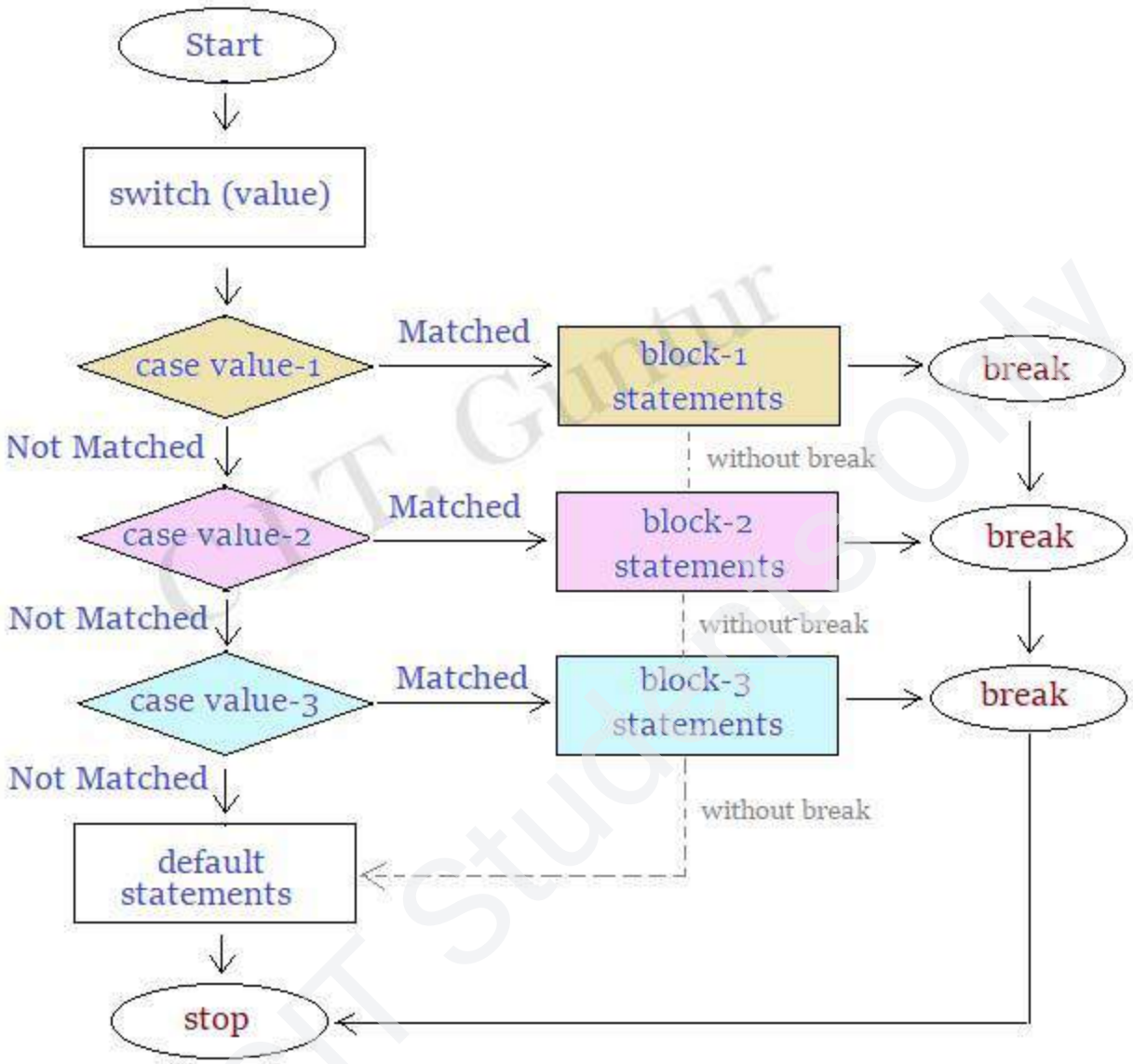
```
switch( expression )                // Expression evaluates to a single value
{
```

```
        case value-1:               //Case is picked when expression gives Value-1
                    Block-1;
                    Break;
        case value-2:               //Case is picked when expression gives Value-2
                    Block-2;
                    Break;
        case value-n:               //Case is picked when expression gives Value-n
                    Block-n;
                    Break;
        default:    // when value of expression didn't match with any case
                    Block-1;
}
Statement outside switch block
```

How does switch statement work?

1. First, the <expression> inside the switch clause is evaluated to an integral constant (int or char).
2. The result is then compared against the case value inside each case statement.
3. If a match is found, all the statements in the matched case are executed until a "break" or end of the switch is reached.
4. The "break" statement brings the control outside "switch" block.
5. *** If "break" is not present after the matching case statements are executed, it will continue to execute all the statements in the below cases including default, till the end of "switch" statement.
6. If no matched case, then control goes to "default" block if it exists (default is optional) and then comes out of the block.
7. Also, there must be only one "case" to be executed; if not, the "default" is to be executed.

Flow chart for Switch case:



Example1: [switch-case statements; multi-way selection]

Aim: Print the given ranks of 1, 2, or 3 in English words

```
#include<stdio.h>
int main()
{                    //opening switch block
int rank;
printf("\n Enter rank 1,2 or 3: ");
scanf("%d",&rank);
```

```
switch (rank) {  //opening switch block
 case 1: printf("\n First Rank");
         break;
 case 2: printf("\n Second Rank");
         break;
 case 3: printf("\n Third Rank");
         break;
 default:
         printf("\n You are not ranked");
} //closing switch block
return 0;
} //closing main function

/* Output
 Enter rank 1,2 or 3: 2
 Second Rank
 Enter rank 1,2 or 3: 4
 You are not ranked
*/
```

	Valid Expressions	Invalid Expressions	
Rule	int & char types are valid	float & string types are invalid	
	2 + 3, 9 * 16 % 2, 10 / 2 + 5, 'a' , 'a' + 1	3.5, 7.0 / 2.5, "Surya Kumar"	
int a=1 char c='B'	a, a – 4, a + c	p, p + 2.5, p * 10	float p=2.5

Example - Valid switch expression:

```
switch (2+5) {
 case 5:
      printf("2+5 makes 7");
      break;
 case 4:
      printf("2+5 is not 4");
      break;
```

```
}

/* Output
Output:
2+5 makes 7
*/
```

Example - Invalid switch expression:

```
switch (4.5) { //error: Switch quantity not an integer
 case 5:
        printf("I am 5");
        break;
 default:
        printf("I am default");
        break;
}
```

C. Repetition

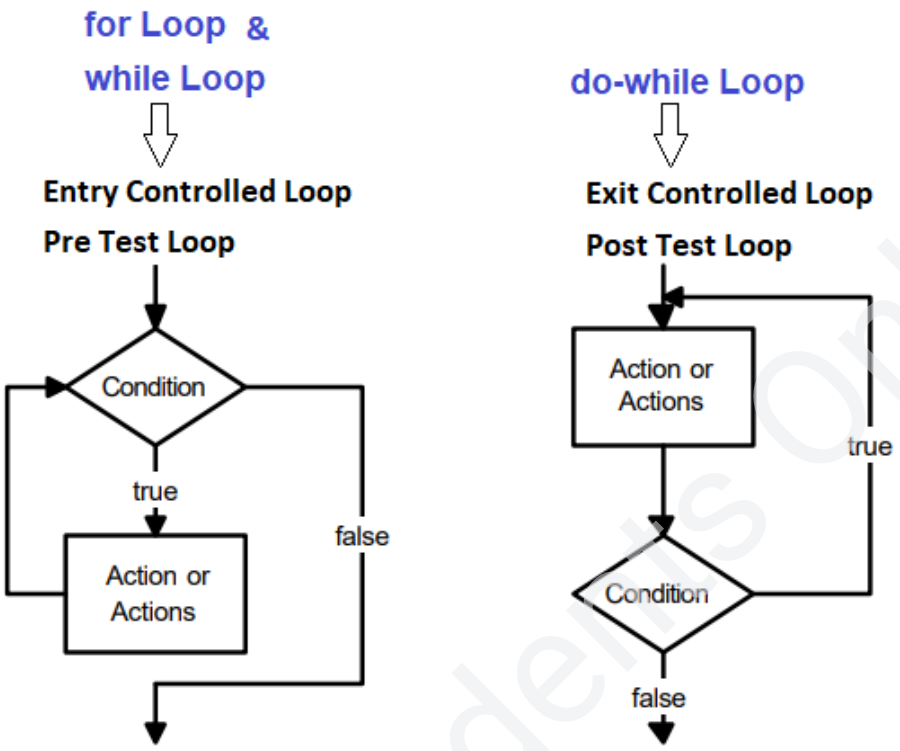
Concept of Loop, Pre-test and Post-test Loops, Initialization and Updating, Event and Counter Controlled Loops, Loops in C, Other Statements Related to Looping.

Iterative Statements or Looping Statements

The process of executing block statements many times repeatedly until the given condition is satisfied is called iterative statements or looping statements. The 3 types of iterative statements:

1. **'for' loop (also, Nested 'for' loop)**
2. **'while' loop**

3. 'do-while' loop



'for' loop:

A for loop is to run a block of statements many times until the given condition satisfied. We choose "for" loop when we KNOW how many times we need to repeatedly execute the block of statements.

"for" loop is an ENTRY-controlled loop or a PRE-TEST loop;
Meaning: it first checks the "condition" to decide if it needs to execute the block of statements.

"for" is a Counter-controlled loop because it is controlled by a counter variable, where the number of times the loop will execute is known ahead of time.

Examples:

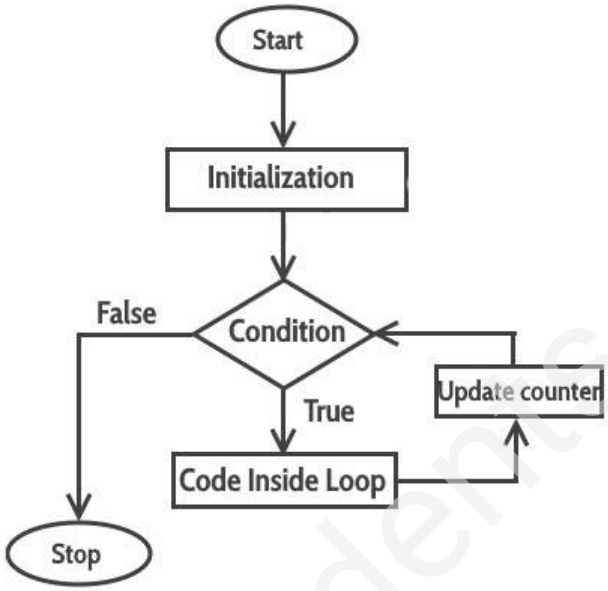
read 5 numbers
print 7 items
sort n items

Syntax:

```
for (Initialization; Condition Expression; Increment or Decrement)
{
```

```
 //Block of statements inside this loop
}
```

Flow Chart of "for" loop:



How does the "for" loop work?

1. First, the initialization statement is executed only once.
2. Second, the "condition" is evaluated.
    a. If the "condition" is FALSE, the control will exit the "for" loop
    b. If the "condition" is TRUE, the statements inside the body of the "for" loop are executed; then, the Update Counter will either Increment or Decrement.
3. Again the "condition" is tested.
4. This process of steps 2 and 3 will repeat until the "condition" is FALSE. When the test expression is false, the control exits the loop.

Example: [for loop]

```
// Print numbers from 1 to 5
#include <stdio.h>
int main() {
  int counter;
  for (counter = 1; counter < 11; ++counter)
  {
    printf("%d ", counter);
  }
  return 0;
```

```
}
```

Output:

```
1 2 3 4 5
```

Example2: [for loop]

```
// Program to find the sum of first n natural numbers
// Natural numbers: Positive integers 1,2,3...n
#include <stdio.h>
int main()
{
    int num, count, sum = 0;
    printf("Enter +ve integer (max of natural number 1-100: ");
    scanf("%d", &num);

    // for loop terminates when num is less than count
    for(count = 1; count <= num; ++count)
    {
        sum += count;
    }

    printf("Sum = %d", sum);
    return 0;
}
```

Output

Enter +ve integer (max of natural number 1-100): 10

Sum = 55

Example3: [for loops with if condition]

```
//Find GCD of 2 numbers
#include <stdio.h>
int main()
{ int num1, num2, i, gcd;
  printf("Enter two integers: ");
  //Storing user input into num1 and num2
  scanf("%d %d", &num1, &num2);

  for(i=1; i <= num1 && i <= num2; ++i)
```

```
  {
    // Checks if the current value of i is
    // factor of both the integers num1 & num2
    if(num1%i==0 && num2%i==0)
      gcd = i;
  }

  printf("GCD of input numbers %d and %d is: %d", num1, num2, gcd);
  return 0;
}

/* Output
Enter two integers: 9 27
GCD of input numbers 9 and 27 is: 9
*/
```

Example4: [FOR Loop]

```
/* C program to print all the characters of C character Set  */
#include<stdio.h>
int main() {
int i;
printf("ASCII ==> Character\n");
for(i = -128; i <= 127; i++)
 printf("%d ==> %c\n", i, i);
return 0;
}
```

Example5: [FOR Loop & Conditional]

```
/* C Program to print character type using ASCII table */
#include <stdio.h>
#include <ctype.h>
int main() {
   printf("| Character | ASCII | Type        |\n");
   printf("| :-------: | ----: | :---------- |\n");
   for (int i = 32; i < 128; i++) {
       printf("|  %3c      | %3d   | ", i, i);
       if (isalpha(i))
           printf("Alphabet    |\n");
```

```
        else if (isdigit(i))
            printf("Digit       |\n");
        else if (ispunct(i))
            printf("Punctuation |\n");
        else if (isspace(i))
            printf("Space       |\n");
        else if (iscntrl(i))
            printf("Control     |\n");
    }
    return 0;
}
```

Output:

Character	ASCII	Type
	32	Space
!	33	Punctuation
"	34	Punctuation
#	35	Punctuation
0	48	Digit
1	49	Digit
A	65	Alphabet
B	66	Alphabet

'while' loop:

A "while" loop is to run a block of statements many times until the given condition satisfied. A while loop has one specific condition and executes as long as the given condition is TRUE. We choose "while" loop when we DO NOT KNOW how many times we need to repeatedly execute the block of statements.

"**while" loop is an ENTRY controlled loop or a PRE-TEST loop;**
Meaning: it first checks the "condition" to decide if it needs to execute the block of statements.

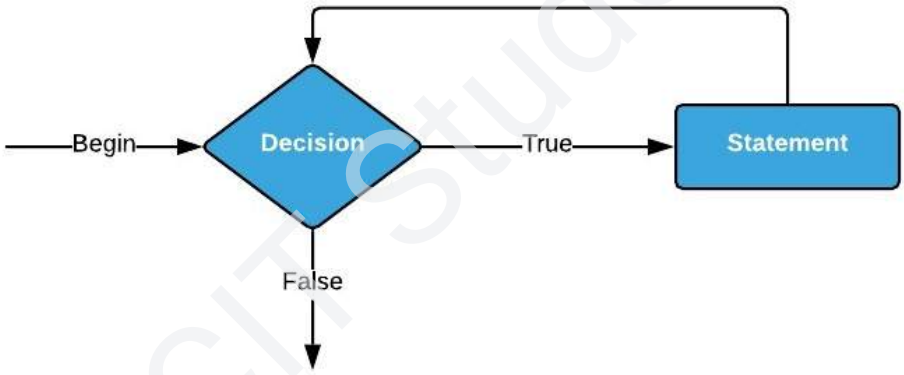
"while" and "do-while" statements can be Event-controlled loops because termination of

the loop depends on an event instead of executing a fixed number of times.
Examples: read until input ends
read until a number encountered
search through data until item found

Syntax of 'while" loop:

```
Initialization;
while (ConditionExpression)
{
  // code block of the loop
  Increment or Decrement (optional)
}
```

Flow Chart of 'while' loop



How "while" loop works?

1. First, while loop tests the "condition" inside the parentheses ().
2. If "condition" is TRUE, statements inside the body of "while" loop are executed.
   a. Counter will be incremented or decremented if it is there (Optional in while loop)
3. Then, "condition" is tested again.
4. The steps 2 and 3 will repeat until "condition" is FALSE.
5. When the "condition" is FALSE, the control will exit the loop.

Different ways to write 'while' loop structure:

`while(true)`	`while( i<5)`	`while( i<=n)`

{	{	{
}	}	}

Example1: [while loop]

```
// Print numbers from 0 to 5
#include <stdio.h>
int main() {
  int i = 0;
  while (i <= 5) {
    printf("%d\n", i);
    ++i;
  }
  return 0;
}
/* Output  : 0 1 2 3 4 5          */
```

Example2: [while loop]

```
#include<stdio.h>
int main ()
{  /* local variable Initialization */
   int i = 1,times=5;
   /* while loops execution */
   while( i <= times )
   {  printf("while loop#: %d\n", i);
      i++;
   }
   return 0;
}
```

Output:

```
while loop#:1
while loop#:2
while loop#:3
while loop#:4
while loop#:5
```

“do-while” loop

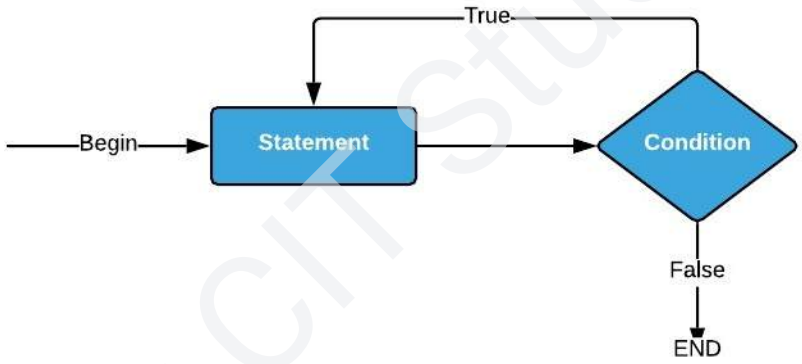
The “do..while” loop is similar to the “while” loop with one important difference.
The body of “do..while” loop is **Executed At Least Once** regardless of the “condition”.
Only then, the “condition” is checked.

“do-while” loop is an EXIT-controlled loop or a POST-TEST loop;
Meaning: it first executes the block of statements at least once and then checks the “condition”
It can also be an **EVENT-controlled** loop.

Syntax of the “do..while” loop:

```
Initialization;
do {
  // code block of the loop
  Increment or Decrement (optional)
}
while (ConditionExpression);
```

Flow Chart of ‘do-while’ loop:



How do-while works?

1. First, the **body of “do-while” loop is executed AT LEAST ONCE.**
2. Only then, the “condition” is checked.
   a. If TRUE, the block of statements in the body of the loop is executed
3. Then, the “condition” is checked once again.
4. Steps 2 and 3 will repeat until the “condition” becomes FALSE.
5. If the “condition” is FALSE, the control will exits the loop.

Example1: [do - while]

```c
#include <stdio.h>
int main() {
    // Initialization statement
    int i = 0;
    do {// loop body
        printf("%d ", i);
        i++;   // update expression : i = i + 1
    } while (i > 0); // condition
    return 0;
}
```

Output: 0

Example 2: [do-while]

```c
// Program to add numbers until the user enters zero
#include <stdio.h>
int main() {
  double number, sum = 0;
  // the body of the loop is executed at least once
  do {
    printf("Enter a number: ");
    scanf("%lf", &number);
    sum += number;
  }
  while(number != 0.0);

  printf("Sum = %.2lf",sum);
  return 0;
}
```

OUTPUT:

Enter a number: 10

Enter a number: 5.25

Enter a number: 0

Sum = 15.25

Here, we have used a do...while loop to prompt the user to enter a number. The loop works as long as the input number is not 0.

Unconditional Statements

The statements that do not need any condition to control the flow of execution of a program are called Unconditional statements. They are,

1. **break** statement
2. **continue** statement
3. **goto** statement

"break" statement

The "break" statement is used to

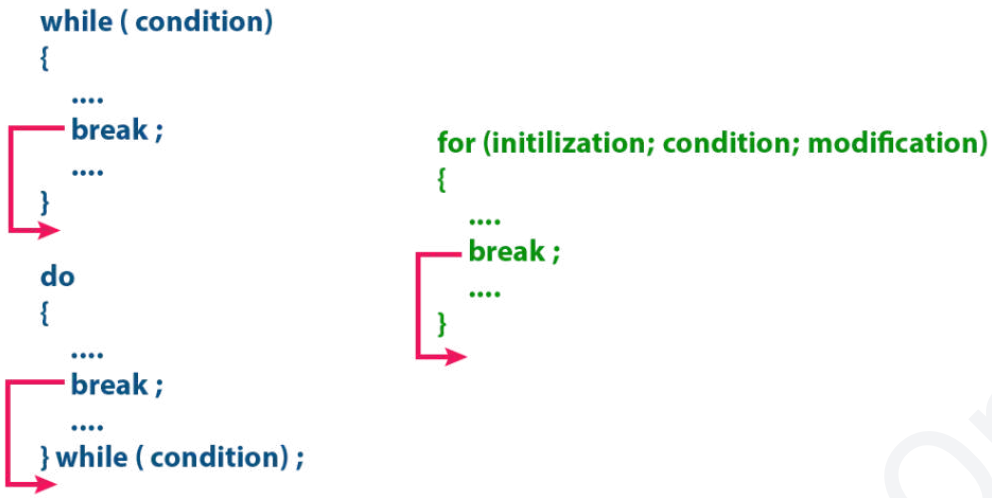
1. terminate a "switch-case" statement
2. terminate looping statements "for", "while" and "do-while"

Note: The "break" statement almost always needs "if" condition to work properly.

"break" Syntax:

```
break;
```

"break" Execution Flow:



Example1: [break in while]

```
//break statement in while loop
#include<stdio.h>
int main()
{    int i = 0;
     while (i<=5)
     {
          if(i==3)
              break;
          printf("%d ",i);
          i++;
     }
     return 0;
}

/* OUTPUT
0 1 2      */
```

"continue" statement

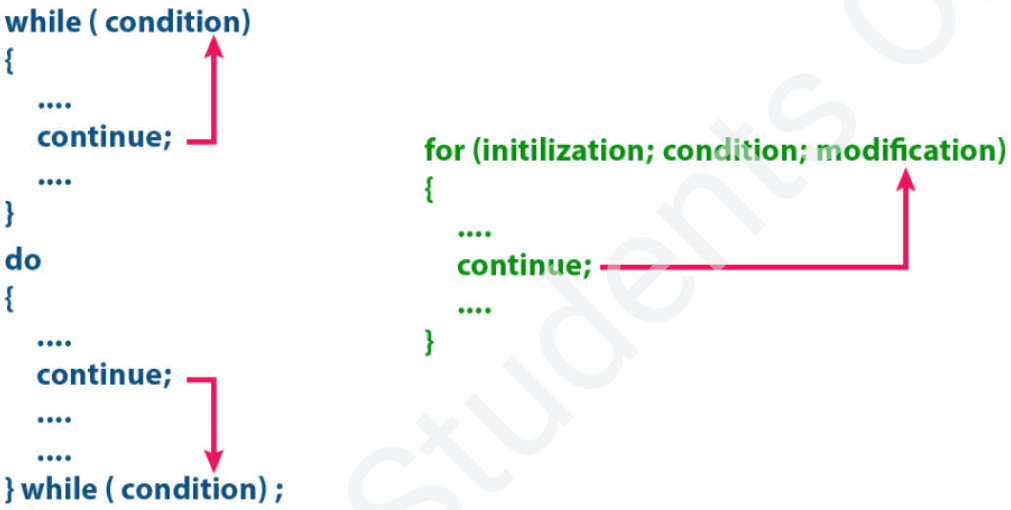
The "continue statement forces the control to skip the current iteration and go to the next iteration of the loop.

1. In "while" and "do-while" statements, the "continue" statement will directly jump the execution control to "condition",
2. In "for" statement, the "continue" statement will jump the execution control to increment/decrement part of the loop.

"continue" Syntax:

```
continue;
```

"continue" Execution Flow:



Example1: [continue statement in for loop]

```
//continue statement in for loop
#include<stdio.h>
int main()
{    int i;
     for(i=0; i<=5; i++)
     {
        if(i==4)
            continue;
        printf("%d ",i);
     }
     return 0;
}

/* OUTPUT */
0 1 2 3 5
```

"goto" statement

The "goto" statement is used to jump the execution control from one statement to another statement in the same function.

"goto" Syntax:

```
    //code
    LABEL:
    //code
    //code
    goto LABEL;
    //code
```

The "LABEL" can be any name defined by the user.
Note: The use of "goto" statement is dangerous as the control flow of the program is not easily manageable. Instead, prepare the logic using "break or continue' and avoid using 'goto' unless it is necessary.

Example1: [goto label jump statement]

```
//goto jump statement
#include<stdio.h>
int main()
{   int rank;
    ENTRY:
    printf("\n Enter rank [1-3] : ");
    scanf("%d",&rank);

    if ((rank<1)||(rank>3))
    {
        printf("\n Rank must be 1, 2 or 3. Please ReEnter!");
        goto ENTRY;
    }
    printf("\n The Rank is %d. Thank you.", rank);
    return 0;
}

/* OUTPUT
 Enter rank [1-3] : 0
 Rank must be 1, 2 or 3. Please ReEnter!

 Enter rank [1-3] : 1
 The Rank is 1. Thank you.       */
```

Quick Reference

Differences between "switch" and "if-else-if" ("else-if" ladder)

switch	if-else-if (else-if ladder)
The control directly jumps to the matched "case" of the value in "switch"	The control goes through every "else-if" until it finds TRUE
"switch" is considered more readable	"if-else-if" ladder is compact than Nested "if-else"
Use of "break" statement in "switch-case" is essential	Use of "break" statement in "else-if" is not required
Data type of the variable or value used in "switch" can be "int" or "char" only. The "float" or string are NOT allowed.	else-if ladder accepts any data type
Switch can only be used for a specific value; a range of values is not allowed.	If condition generally uses relational operators (< or >) to test values.
Switch-case statement works on Equality operator	If-else condition works on TRUE or FALSE boolean value

Syntax:
switch(expression) {
case value-1:
Block-1;
Break;
case value-2:
Block-2;
Break;
case value-n:
Block-n;
Break;
default:
Block-1;
}
Statement outside switch block
	Syntax:
	if (condition 1)
	{
	statement1;
	} else if (condition 2)
	{
	Statement2;
	} else if (condition n)
	{
	Statement n;
	} else
	{
	Default statement;
	}
	Statement in the main program

Comparison of Loops

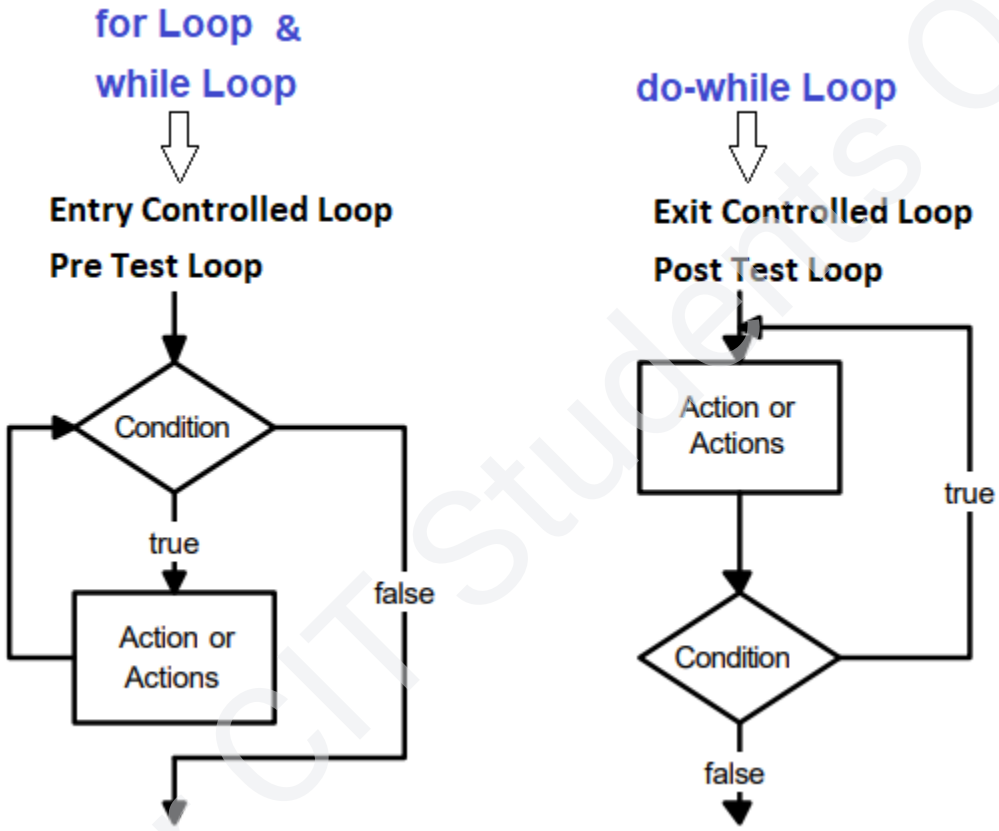
'for' loop	'while' loop	'do-while' loop
Entry controlled loop - CONDITION is specified at TOP	**Entry** controlled loop - CONDITION is specified at TOP	**Exit** controlled loop - CONDITION is specified at BOTTOM
COUNTER (Inc/Dec) controlled loop	**EVENT** (or Condition) controlled loop	**EVENT** (or Condition) controlled loop
Use it when you **know** how many times to iterate	Use it when you **don't know** how many times to iterate	Use it when you **don't know** how many times to iterate
If the condition is TRUE, the body of "for" will be executed.	If the condition is TRUE, the body of "while" will be executed.	The body of "do-while" be executed at least once before checking the condition.
Repeats a Preset number of times	Repeats until a condition is met	Block of statements is **executed at least once**; then Repeats until a condition is met
No BRACKETS { } for single statement	No BRACKETS { } for single statement	BRACKETS { } are compulsory even for a single statement

Syntax:
`for(initialization;`
`condition;`
`Inc/Dec)`
`{`
`// code block`
`}`
	Syntax:
	`Initialization;`
	`while(condition)`
	`{`
	`// code block`
	`Inc/Dec;(optional)`
	`}`
		Syntax:
		`Initialization;`
		`do`
		`{`
		`// code block`
		`Inc/Dec;(optional)`
		`}while(condition);`

Example: for
`#include <stdio.h>`
`int main() {`
`int i;`
`//for(init;cond;Inc/Dec)`
`for (i=0; i<=5; i++)`
`{`
`//loop body`
`printf("%d ", i);`
	Example: while
	`#include <stdio.h>`
	`int main() {`
	`int i = 0;`
	`//while condition`
	`while (i<=5){`
	`//loop body`
	`printf("%d ", i);`
	`//update expression`
	`i++;`
		Example: do-while
		`#include <stdio.h>`
		`int main() {`
		`int i = 0;`
		`do {`
		`//loop body`
		`printf("%d ", i);`
		`//update expression`
		`i++;`
		`} while (i > 0);`

}
return 0;
}
	}
	return 0;
	}
		//condition
		return 0;
		}

Output: 0 1 2 3 4 5	Output: 0 1 2 3 4 5	Output: 0



Comparison of "break" and "continue" statements

break	continue
Used to terminate the loop	Used to SKIP current iteration and go to NEXT iteration
Control passed to outside the loop	Control passed to the beginning of the loop
EXIT from control loop	Loop takes NEXT iteration
"break" may be used in both SWITCH and all LOOPS (for, while, do-while)	"continue" may only be used in LOOPS (for, while, do-while)

Syntax:
`for (init; condition; inc/dec)`
`{`
`//body of loop`
`  if(condition to break)`
`  break;`
`}`
	Syntax:
	`for (init; condition; inc/dec)`
	`{`
	`//body of loop`
	`  if(condition to break)`
	`  continue;`
	`}`

Example:
`#include<stdio.h>`
`int main()`
`{  int i;`
`  for(i=0;i<=5;i++)`
`  {`
`  if(i==3)`
`  break;`
`  printf("%d ",i);`
`  }`
`  return 0;`
`}`
	Example:
	`#include<stdio.h>`
	`int main()`
	`{  int i;`
	`  for(i=0;i<=5;i++)`
	`  {`
	`  if(i==3)`
	`  continue;`
	`  printf("%d ",i);`
	`  }`
	`  return 0;`
	`}`

Output: `0 1 2`	**Output:** `0 1 2 4 5`

break	exit (return value)
break is a keyword in C.	exit() is a standard C library function in stdlib.h
break causes an immediate termination from the switch or loop (for, while or do-while) and continues the remaining program.	exit() terminates whole program execution.
break is a reserved word in C; therefore it can't be used as a variable name.	exit() can be used as a variable name.
No header files need to be included to use break statement.	stdlib.h needs to be included to use exit().
break transfers the control to outside the switch or loop (for, while or do-while).	exit() returns the control to the operating system or another program that uses this one as a sub-process.

Example of break
// some code here before while loop
while(true)
{
...
if(condition)
break;
}
// some code hereafter while loop
	Example of exit()
	// some code here before while loop
	while(true)
	{
	...
	if(condition)
	exit(1);
	}
	// some code hereafter while loop

In the above code, break terminates the while loop and some code here after the while loop will be executed after breaking the loop.	In the above code, when if(condition) returns true, exit(1) will be executed and the program will get terminated. Upon call of exit(1).
Conclusion: break is a statement that terminates a switch or loops and continues the next program statements.	**Conclusion:** exit() is a library function that causes the immediate termination of the entire program.