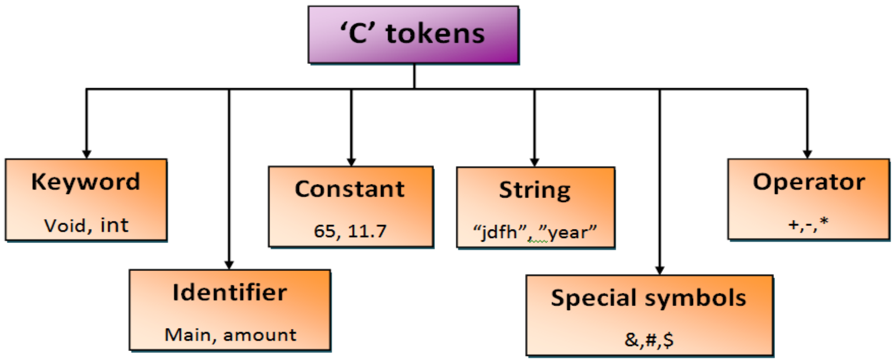
UNIT-II

Programming Style: Tokens of C, Keywords, Variables, Constants and rules to form variables and constants, Data Types, Declaration of Variables and initialization, Operators, Operator precedence and associativity. Type conversions.
Flow of Control: Selection: Two way selection, multi-way selection
Repetition and Unconditional Control Statements: concept of loop ,pre test and post test loops, initialization and updating loops ,while statement, do-while statement, for statements, nested loops, break ,continue, goto.

Tokens of C

Tokens in 'C' is the most important element to be used in creating a program in C. We can define the token as the smallest individual element in C. For `example, we cannot create a sentence without using words; similarly, we cannot create a program in C without using tokens in C. Therefore, we can say that token's in C is the building block or the basic component for creating a program in C language.



C tokens are of six types. They are:

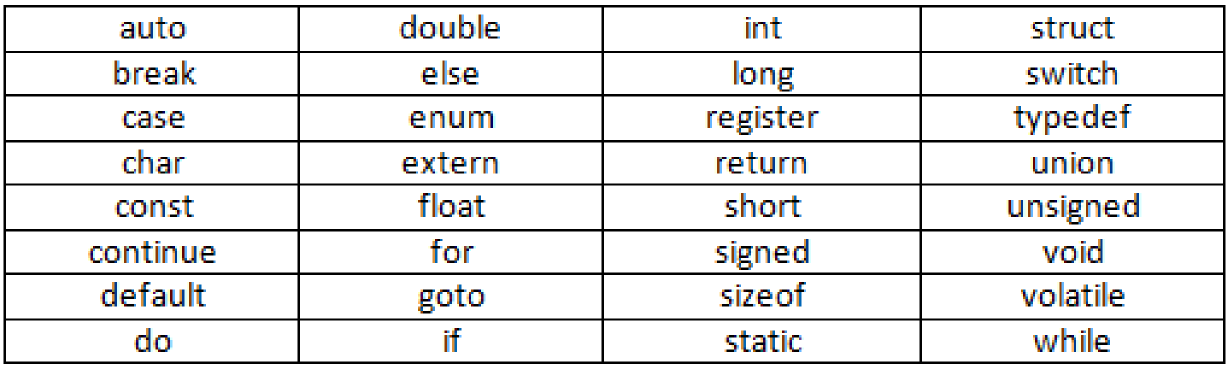
• Keywords	(eg: int, while),
• Identifiers	(eg: main, total),
• Constants	(eg: 10, 20),
• Strings	(eg: "total", "hello"),
• Special symbols	(eg: (), {}),
• Operators	(eg: +, /,-,*)

C tokens example program:

```
Void main()                //"void" is a keyword
{
        int a=10, b=20, c;      // "int" keyword, a, b, c (identifiers), (, ;)special symbol
        c = a + b;              // (+)  operator
        printf("the sum of two number=%d", c);
        getch();
}
```

(1) Keywords in C language:

Keywords in C can be defined as the **pre-defined** or the **reserved words** having its own importance, and each keyword has its own functionality. Since keywords are the pre-defined words used by the compiler, so they cannot be used as the variable names. If the keywords are used as the variable names, it means that we are assigning a different meaning to the keyword, which is not allowed. C language supports 32.



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

(2) Identifiers in C language:

Identifiers in C are used for naming variables, functions, arrays, structures, etc. Identifiers in C are the user-defined words. It can be composed of uppercase letters, lowercase letters, underscore, or digits, but the starting letter should be either an underscore or an alphabet. Identifiers cannot be used as keywords.

Rules for constructing identifiers in C are given below:

- The first character of an identifier should be either an alphabet or an underscore but not digit.
- In identifiers, both uppercase and lowercase letters are distinct. Therefore, we can say that identifiers are case sensitive.
- Commas or blank spaces cannot be specified within an identifier.
- Keywords cannot be represented as an identifier.
- The length of the identifiers should not be more than 31 characters.
- Identifiers should be written in such a way that it is meaningful, short, and easy to read.

Example of valid identifiers

```
float total;
int sum;
int _m _;
float sum_1;
```

Example of valid identifiers

```
float 2sum;
int int;
float m+n;
float char;
```

Differences between Keywords and Identifiers:

Keyword	**Identifier**
Keyword is a pre-defined word.	The identifier is a user-defined word
It must be written in a lowercase letter.	It can be written in both lowercase and uppercase letters.
Its meaning is pre-defined in the c compiler.	Its meaning is not defined in the c compiler.
It is a combination of alphabetical characters.	It is a combination of alphanumeric characters.
It does not contain the underscore character.	It can contain the underscore character.

(3) Constants:

A constant is a value assigned to the variable which will remain the same throughout the program, i.e., the constant value cannot be changed and also called literals.
There are two ways of declaring constant:

- Using const keyword
- Using #define pre-processor

Syntax:

const data_type variable_name = value; (or) #define variable_name value

Example:

Const int a=10; (or) #define PI 3.142

Types of C constant:

1. Integer constants
2. Real or Floating point constants
3. Octal & Hexadecimal constants
4. Character constants
5. String constants
6. Backslash character constants

Constant	Example
Integer constant	10, 11, 34, etc.
Floating-point constant	45.6, 67.8, 11.2, etc.
Octal constant	011, 088, 022, etc.
Hexadecimal constant	0x1a, 0x4b, 0x6b, etc.
Character constant	'a', 'b', 'c', etc.
String constant	"java", "c++", ".net", etc.

Rules for constructing C constant:

1. Integer Constants in C:

- An integer constant must have at least one digit.
- It must not have a decimal point.
- It can either be positive or negative.
- No commas or blanks are allowed within an integer constant.
- If no sign precedes an integer constant, it is assumed to be positive.
- The allowable range for integer constants is -32768 to 32767.

2. Real constants in C:

- A real constant must have at least one digit
- It must have a decimal point
- It could be either positive or negative
- If no sign precedes an integer constant, it is assumed to be positive.
- No commas or blanks are allowed within a real constant.

3. Character and string constants in C:

- A character constant is a single alphabet, a single digit or a single special symbol enclosed withinsingle quotes.
- The maximum length of a character constant is 1 character.
- String constants are enclosed within double quotes.

4. Backslash Character Constants in C:

- There are some characters which have special meaning in C language.
- They should be preceded by backslash symbol to make use of special function of them.
- Given below is the list of special characters and their purpose.

Backslash character	**Meaning**
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\”	Double quote
\’	Single quote
\\	Backslash
\v	Vertical tab
\a	Alert or bell
\?	Question mark

Example: program on const and #define constants:

```
#include<stdio.h>
#include<conio.h>
#define letter 'A'
#define backslash_char '\?'
void main()
{
      const int height = 100;
      const float number = 3.14;
      printf("value of height :%d \n", height );
      printf("value of number : %f \n", number );
      printf("value of letter : %c \n", letter );
      printf("value of backslash_char : %c \n", backslash_char);
      getch();
}
```

Output :

```
value of height : 100
value of number : 3.140000
value of letter :A
value of backslash_char : ?
```

(4) Strings:

Strings in C are always represented as an array of characters having null character '\0' at the end of the string. This null character denotes the end of the string. Strings in C are enclosed within double quotes, while characters are enclosed within single characters. The size of a string is a number of characters that the string contains.

Now, we describe the strings in different ways:

char a[10] = "welcome"; // The compiler allocates the 10 bytes to the 'a' array.
char a[] = "welcome"; // The compiler allocates the memory at the run time.
char a[10] = {'w','e','l','c','o','m','e','\0'}; // String is represented in the form of characters.

(5) Special symbols:

Special symbols are used in C and they have special meaning which cannot be used for another purpose.

- **Square brackets [ ]:** represents the single and multidimensional subscripts.
- **Simple brackets ( ):** used in function declaration and function calling.
- **Curly braces { }:** used in the opening and closing of the code as well as in loops.
- **Comma (,):** used for separating for more than one statement.
- **Pre-processor (#):** basically denotes that we are using the header file.

- **Asterisk (*):** used to represent pointers and operator for multiplication
- **Period (.):** used to access a member of a structure or a union.

(6) Operators:

Operators in C is a special symbol used to perform the functions. The data items on which the operators are applied are known as operands. Operators are applied between the operands. Depending on the number of operands, operators are classified as follows:

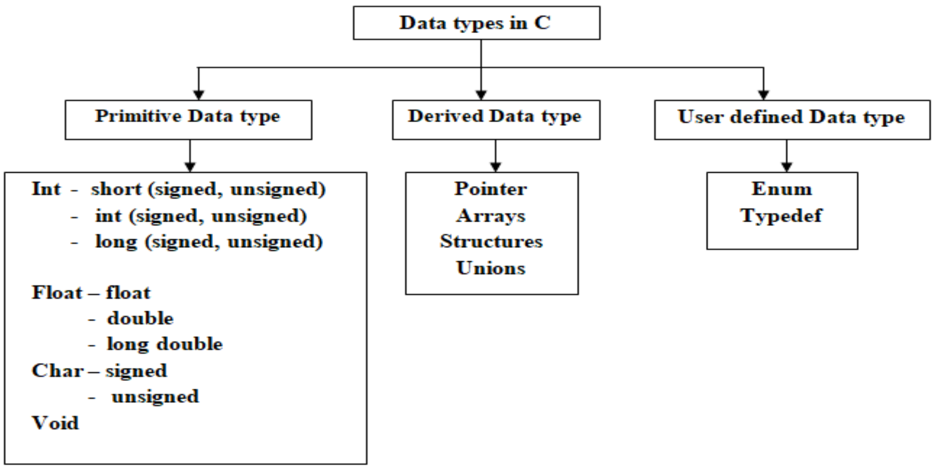
1) Arithmetic Operators 2) Relational Operators 3) Logical Operators
4) Bitwise Operators 5) Conditional Operators 6) Assignment Operator
7) Special Operator.

DATA TYPES

Definition:

"Data type" is a representation of a data. Clearly we can say that how much memory is required to allocate and what type of data is allowed to store in it. C language has some predefined set of data types to handle various kinds of data that we can use in our program. These data types have different storage capacities. Data types are classified into three types

- Primitive data type
- Derived data type
- User defined data type

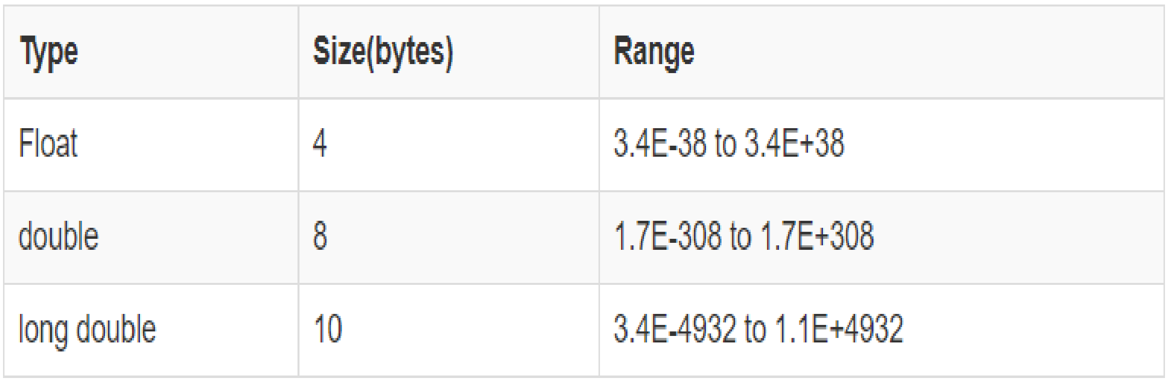


Primitive data types:

- **Integers**: Integers are used to store whole numbers

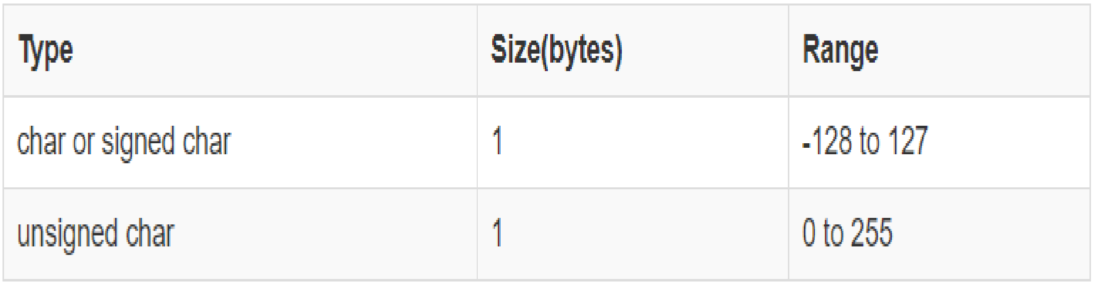
Type	Size(bytes)	Range
int or signed int	2 or 4 bytes	-32,768 to 32,767 (or) -2,147,483,648 to 2,147,483,647
Unsigned int	2 or 4 bytes	0 to 65,535 (or) 0 to 4,294,967,295
Signed short	2 bytes	-32,768 to 32,767
Unsigned short	2 bytes	0 to 65,535
Signed long	4 bytes	-2,147,483,648 to 2,147,483,647
Unsigned long	4 bytes	0 to 4,294,967,295

- **Float data type**: Floating types are used to store real numbers.



Type	Size(bytes)	Range
Float	4	3.4E-38 to 3.4E+38
double	8	1.7E-308 to 1.7E+308
long double	10	3.4E-4932 to 1.1E+4932

- **Char data type**: Character types are used to store characters value.



Type	Size(bytes)	Range
char or signed char	1	-128 to 127
unsigned char	1	0 to 255

- **Void type**: void type means no value. This is usually used to specify the type of functions which returns nothing

C – Variable

- A **variable** is a name of the memory location which is used to store data given by the user.
- Variable values can be changed, and it can be reused many times in the program.
- Before going to use a variable in the program we must need to declare or define a variable with specified type.
- C variable might be belonging to any of the data type like int, float, char etc.

Rules for naming C variable:

1. Variable name must begin with letter or underscore but not with digit.
2. Keywords are not allowed to use as a variable name because C programming is a case sensitive.
3. Variables can be constructed with combination of alphanumeric.
4. No special symbols are allowed other than underscore.
5. Commas, blank spaces are not allowed in between the variables.

Valid cases for variables:

```
int a_b_c;
float average3;
double _sum_;
```

Declaration and initialization of C variable:

- Variables should be declared in the C program before to use.
- Memory space is not allocated for a variable while declaration. It happens only on variable definition.
- Variable initialization means assigning a value to the variable.

Syntax for variable declaration:

```
Data_type variable_name;
```

Example:

int a;

Syntax for variable initialization or definition:

Data_type variable_name = value;

Example:

int a=10;

There are three types of variables in C program They are,

1. Local variable
2. Global variable
3. Static variable

1. **local variable in C:**

- The scope of local variables will be within the function only.
- These variables are declared within the function and can't be accessed outside the function.
- In the below example, m and n variables are having scope within the main function only. These are notvisible to test function.
- Likewise, a and b variables are having scope within the test function only. These are not visible to mainfunction.

Example:

```
#include<stdio.h>
#include<stdio.h>
void fun( )
{
        int a=10,b=20,c; //local varialbes
        c=a+b;
        printf("sum of the two variable=%d", c);
}
void main( )
{
        fun();
        printf("hello world");
        getch();
}
```

Output:

```
Sum of two variable= 30
Hello world
```

2. **Global variable in C:**

- The scope and life time of global variables will be throughout the program. These variables can be accessed fromanywhere in the program.
- This variable is defined outside the main function. So that, this variable is visible to main function andall other sub functions.

Example:

```
#include<stdio.h>
int a=10,b=20,c;
void fun( )
{
        int d=10;  //local variable
        c=a + b;
        printf("sum of the two variable=%d", c);
        printf("the value of d is=%d", d);
}
void main( )
```

```
{
        fun();
        printf(“the values of a and b are=%d,%d”,a,b);
        getch();
}
```

Output:

```
Sum of the two variable=30
The value of d is = 10
The values of a and b are=10,20
```

3. Static variables in C:

- A variable that is declared with the static keyword is called static variable. It retains its value between multiple function calls. The values of the static variables are not initializing again when the function called.

Example :

```
void fun()
{
         int x=10;
         static int y=10;
         ++x;
         ++y;
         printf("\n%d,\t %d",x,y);
}
void main()
{
        fun();
        fun();
        fun();
}
```

Output:

```
    11,     11
    11,     12
    11,     13
```

OPERATORS

Operators in C Language:

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

Precedence of Operators in C:

The precedence of operator specifies that which operator will be evaluated first and next. The associativity specifies the operator direction to be evaluated; it may be left to right or right to left. ( ), ^, /, *, +, -, = ...

Example : **int** value=10+20*10; //210

C operators can be classified into following types,

- Arithmetic operators
- Relational operators
- Logical operators
- Bitwise operators
- Assignment operators
- Conditional operators (ternary operator)
- Special operators (sizeof (), comma)

Arithmetic operators:

.

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

Example:

```
#include <stdio.h>              //stdio.h is a header file used for input.output purpose.
void main()
{
    //set a and b both equal to 5.
    int a=5, b=5, a1=6, a2=7;
    int c=a+b;        //addition
    int d=a-b;      // subtraction
    int e=a/b;      //division
    int f=a*b;      //multiplication
    printf("the addition is=%d",c);
    printf("\n the subtraction is=%d",d);
    printf("\n the division is=%d",e);
    printf("\n the multiplication is=%d",f);
    printf("\n the decrement operation is= %d %d",a--,--b);   // decrement operator
    printf("\n the increment operation is = %d %d",a1++,++a2);     // increment operator
}
```

Output:

```
The  addition is =10
The subtraction is = 0
The division = 1
The multiplication = 25
The decrement operation is = 5    4
 The increment operation is= 6    8
```

Relation operators

Relation operators are used to specify the relation between two operands such as greater than, less than, equalto and etc.If the relation between two operands is true then the result is one (1) otherwise zero(0).

Assume variable **A** holds 5 and variable **B** holds 10 then

Operator	Description	Example	Output
= =	Check if two operand are equal	A= =B	Is not true gives 0 as output
!=	Check if two operand are not equal.	A!=B	Is true gives 1 as output
>	Check if operand on the left is greater than operand on the right	A>B	Is not true gives 0 as output
<	Check operand on the left is smaller than right operand	A<B	Is true gives 1 as output
>=	check left operand is greater than or equal to right operand	A>=B	Is not true gives 0 as output
<=	Check if operand on left is smaller than or equal to right operand	A<=B	Is true gives 1 as output

Example:

```
#include <stdio.h>            //stdio.h is a header file used for input.output purpose.
void main()
{
   int A=5, B=10;
   int ans1= (A==B);                          // check equality
   printf("the result=%d", ans1);
   int ans2= (A!=B);                          // check inequality
```

```
    printf("\n the result=%d", ans2);
    int ans3= (A>B);                                     // check greater number
    printf("\n the result=%d", ans3);
    int ans4= (A<B);                                     // check lesser number
    printf("\n the result=%d", ans4);
    int ans5= (A>=B);                       // check greater equal
    printf("\n the result=%d", ans5);
    int ans6= (A<=B);                       // check lesser equal
    printf("\n the result=%d", ans6);
}
```

Output:

```
the result=0
 the result=1
 the result=0
 the result=1
 the result=0
 the result=1
```

Logical operators

Logical operators are used to perform operations on more than one expression. C language supports following 3 logical operators. Suppose a=1 and b=0,

Operator	Description	Example
&&	**Logical AND**: Called Logical AND operator. If both the operands are true, then the condition becomes true.	(a && b) is false
||	**Logical OR:** Called Logical OR Operator. If any of the two operands is true, then the condition becomes true.	(a || b) is true
!	**Logical NOT:** Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	(!a) is false

Example:

```
#include <stdio.h>
void main()
{
    int a=1, b=0;
    int ans1= (a && b);               // logical AND (&&)
    printf("the result=%d", ans1);
    int ans2= (a||b);                              // logical OR(||)
    printf("\n the result=%d", ans2);
    int ans3= (!a);                                // logical NOT (!)
    printf("\n the result=%d", ans3);
}
```

Output:

```
the result=0
 the result=1
 the result=0
```

Bitwise operators:

Bitwise operators perform manipulations of data at bit level. These operators also perform shifting of bits from right to left. Bitwise operators are not applied to float or double. Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ is as follows –

Assume variable 'A' holds 60(0011 1100) and variable 'B' holds 13(0000 1101), then –

Operator	**Description**	**Example**
&	**Bitwise AND:**	(A & B) = 12, i.e., 0000 1100
|	**Bitwise OR:**	(A | B) = 61, i.e.,0011 1101
^	**Bitwise exclusive OR**	(A ^ B) = 49, i.e., 0011 0001
~	**Binary Ones Complement**	(~A) = -61, i.e,.1100 0011 in 2's complement form
<<	**Binary Left Shift:**.	A << 2 = 240 i.e., 1111 0000
>>	**Binary Right Shift:**.	A >> 2 = 15 i.e., 0000 1111

Now let's see truth table for bitwise &, | and ^

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

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

Example:

Assume A = 60 and B = 13 in binary format, they will be as follows −

A = 0011 1100

B = 0000 1101

A&B = 0000 1100

A | B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

Example:

```
#include<stdio.h>
#include<conio.h>
Void main()
{
int a,b,c,d,e,f,g;
printf("enter a,b:");
scanf("%d%d",&a,&b);
c=a&b;
d=a|b;
e=a^b;
f=a<<4;
g=b>>3;
printf("result for bitwise and is %d",c);
printf("result for bitwise or is %d",d);
printf("result for bitwise not is %d",e);
printf("result for left shift is %d",f);
printf("result for right shift is %d",g);
}
```

Assignment Operators

Assignment operators are used to assign a value or perform an operation on right hand side and that will store on the left hand side variable.

Example: int a=20; //here 20 is a value that stored in variable a
int sum+=30//sum=sum+30//here 30 is added to the sum variable and stored in the same sum variable

Assignment operators supported by C language are as follows.

Operator	Description	Example

=	assigns values from right side operands to left side operand	a=b,
		c=a+b

+=	adds right operand to the left operand and assign the result to left	a+=b is same as a=a+b
-=	subtracts right operand from the left operand and assign the result to left operand	a-=b is same as a=a-b
*=	mutiply left operand with the right operand and assign the result to left operand	a*=b is same as a=a*b
/=	divides left operand with the right operand and assign the result to left operand	a/=b is same as a=a/b
%=	calculate modulus using two operands and assign the result to left operand	a%=b is same as a=a%b
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2

^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

Example:

```
#include<stdio.h>
#include<conio.h>
Void main()
{
      int a,b,c,d,e,f,g;
      printf("enter a,b:");
      scanf("%d%d",&a,&b);
      c+=b;
      d - =a;
      e*=b;
      a/=2;
      printf("result for assignment operator is %d",c);
      printf("result for assignment operator is %d",d);
      printf("result for assignment operator is %d",e);
      printf("result for assignment operator is %d",a);
}
```

Special operators:

Operator	Description	Example
sizeof	Returns the size of an variable	**sizeof(x)** return size of the variable **x**
&	Returns the address of an variable	**&x ;** return address of the variable **x**
*****	Pointer to a variable	***x ;** will be pointer to a variable **x**
Comma	Separate variables, expressions	**Int a, b, c;**

Example :

```
#include <stdio.h>
void main() {
int a = 16;
   printf("Size of variable a : %ld\n",sizeof(a));
   printf("Size of int data type : %ld\n",sizeof(int));
   printf("Size of char data type : %ld\n",sizeof(char));
   printf("Size of float data type : %ld\n",sizeof(float));
   printf("Size of double data type : %ld\n",sizeof(double));

}
```

Conditional operator:

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

1. **Ternary Operator**
2. **? : Operator**

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

The syntax of a conditional operator is :

(condition) ? expression 1: expression 2;

Explanation:

- The question mark **"?"** in the syntax represents the **if** part.
- The condition part is generally returns either true or false, based on which it is decided whether (expression 1) will be executed or (expression 2)
- If (condition) returns true then the expression on the left side of **" : "** i.e (expression 1) is executed.
- If (condition) returns false then the expression on the right side of **" : "** i.e (expression 3) is executed.

Example:

```
#include <stdio.h>
int main()
{
   int a=10,b=40,max1;    // assigning value 10 to variable a
   max1= (a>b) ? a : b
   printf("the value is :=%d",max1);
}
```

Output:

```
the value is : = 40
```

Increment Operator in C Programming

1. Increment operator is used to increment the current value of variable by adding integer 1.
2. Increment operator can be applied to only variables.
3. Increment operator is denoted by ++.

Different Types of Increment Operation

In C Programming we have two types of increment operator i.e Pre-Increment and Post-Increment Operator.

A. Pre Increment Operator

Pre-increment operator is used to increment the value of variable before using in the expression. In the Pre-Increment value is first incremented and then used inside the expression.

```
b = ++y;
```

In this example suppose the value of variable 'y' is 5 then value of variable 'b' will be 6 because the value of 'y'gets modified before using it in a expression.

Sample program

```
#include<stdio.h>
void  main()
{
int a,x=10;
a = ++x;
printf("Value of a : %d",a);
printf("Value of x : %d",x);

}
```

Output :

```
Value of a : 11
Value of x: 11
```

B. Post Increment Operator

Post-increment operator is used to increment the value of variable as soon as after executing expression completely in which post increment is used. In the Post-Increment value is first used in a expression and then incremented.

```
b = x++;
```

In this example suppose the value of variable 'x' is 5 then value of variable 'b' will be 5 because old value of 'x'is used.

Sample program

```
#include<stdio.h>

void main()
{
inta,b,x=10;
a = x++;
printf("Value of a : %d",a);
printf("Value of a : %d",x);

}
```

Output :

```
Value of a : 10
Value of x : 11
```

Decrement Operator in C Programming :

1. Decrement operator is used to decrease the current value of variable by subtracting integer 1.
2. Like Increment operator, decrement operator can be applied to only variables.
3. Decrement operator is denoted by –.

Different Types of Decrement Operation :

When decrement operator used in C Programming then it can be used as pre-decrement or post-decrement operator.

A. Pre Decrement Operator

Pre-decrement operator is used to decrement the value of variable before using in the expression. In the Pre-decrement value is first decremented and then used inside the expression.

```
b = --var;
```

Suppose the value of variable var is 10 then we can say that value of variable 'var' is firstly decremented thenupdated value will be used in the expression.

C Program

```
#include<stdio.h>

void main()
{
intb,y=10;
b = --y;
printf("Value of b : %d",b);
printf("Value of y : %d",y);
}
```

Output :

```
Value of a : 9
Value of y : 9
```

B. Post Decrement Operator

Post-decrement operator is used to decrement the value of variable immediatly after executing expression completely in which post decrement is used. In the Post-decrement old value is first used in a expression andthen old value will be decrement by 1.

```
b = var--;
```

Value of variable 'var' is 5. Same value will be used in expression and after execution of expression new valuewill be 4.

C Program

```
#include<stdio.h>

void main()
{
inta,x=10;
a = x--;
printf("Value of a : %d",a);
printf("Value of x : %d",x);

}
```

Output :

```
Value of a : 10
Value of x : 9
```

Operator precedence and associativity:

Symbol	Type of Operation	Associativity
[ ] ( ) . –> postfix ++ and postfix —	**Expression**	**Left to right**
prefix ++ and prefix — sizeof & * + – ~ !	**Unary**	**Right to left**
typecasts	**Unary**	**Right to left**
*** / %**	**Multiplicative**	**Left to right**
+ –	**Additive**	**Left to right**
<< >>	**Bitwise shift**	**Left to right**
< > <= >=	**Relational**	**Left to right**
== **!=**	**Equality**	**Left to right**
&	**Bitwise-AND**	**Left to right**
^	**Bitwise-exclusive-OR**	**Left to right**
|	**Bitwise-inclusive-OR**	**Left to right**
&&	**Logical-AND**	**Left to right**
||	**Logical-OR**	**Left to right**
? :	**Conditional-expression**	**Right to left**
= ***=** /= **%=** += –= <<= >>= **&=** ^= |=	**Simple and compound assignment**	**Right to left**
,	**Sequential evaluation**	**Left to right**

TYPE CASTING

Typecasting is converting one data type into another one. It is also called as data conversion or type conversion. For example, if you want to store a long value into a simple integer then you can type cast long to int. You can convert values from one type to another explicitly using the cast operator. New data type should be mentioned before the variable name or value in brackets which to be typecast.
'C' programming provides two types of type casting operations:

- Implicit type casting
- Explicit type casting

1. Implicit type casting:

- Implicit type casting means conversion of data types without losing its original meaning.
- This type of typecasting is essential when you want to change data types **without** changing the significance of the values stored inside the variable.
- During conversion, strict rules for type conversion are applied. If the operands are of two different data types, then an operand having lower data type is automatically converted into a higher data type.

Important Points about Implicit Casting :

- Implicit type of type conversion is also called as standard type conversion. We do not require any keyword or special statements in implicit type casting.
- Converting from smaller data type into larger data type is also called as **type promotion**.
- The implicit type conversion always happens with the compatible data types.

Example:

```
#include<stdio.h>
void main()
{
 short a=10;
 int b;
 b=a;
 printf("%d\n",a);
 printf("%d\n",b);
}
```

```
Output:   10
           10
```

2. Explicit conversion:

- This conversion is done by user. This is also known as typecasting. Data type is converted into another data type forcefully by the user.

Syntax:

```
(data_type) expression;
```

Here,

- The type name is the standard 'C' language data type.
- An expression can be a constant, a variable or an actual expression.

Example:

```
#include<stdio.h>
void main()
{
  double a = 1.2;
  //int b  = a; //Compiler will throw an error for this
  float b = (float)a + 1;
  printf("Value of a is %lf\n", a);
  printf("Value of b is %f\n", b);
}
```

Output:

```
Value of a is 1.200000
Value of b is  2.200000
```

FLOW OF CONTROL

Control Flow-Relational Expressions - Logical Operators

Control Structures are those statements that decide the order in which individual statements or instructions of a program are executed or evaluated.
Control Structures are broadly classified into:

1. **Conditional statements**: Specifies a condition to execute a group of statements in a program.The control structures, if, if-else, nested if, if else ladder and switch, belongs to this category.
2. **Iterative statements:** Executes one or more statements repeatedly until a condition is met.The control structures: while, do...while, and for loops, belong to this category.
3. **Jump statements:** Transfer of control from one part to another within the program. Ex. goto, break, continue and return.

Selection statements

Selection statements allow a program to test several *conditions*, and execute instructions based on which condition is true. That is why selection statements are also referred to as conditional statements.
There are two types of selection statements:

1. Two-way selection statements
2. Multi-way selection statements

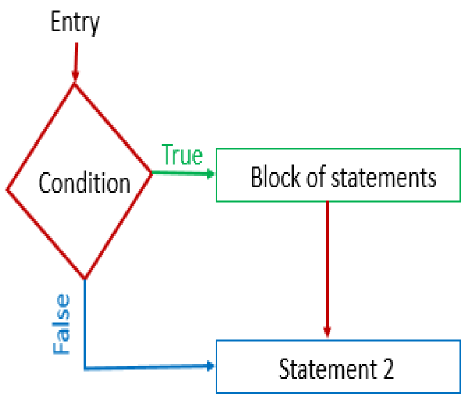
Two-way selection statements:

- The two-way selection is the basic decision statement for computers. The decision is based on resolving a binary expression, and then executing a set of commands depending on whether the response was true or false.
- Two way selection statements are:
  - Simple if statement
  - If – else statement
  - Nested if-else statement

Simple if / if statement:

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

```
if(expression)
{
        //statements
}
```



In the preceding syntaxes, a simple or compound statement (block of statements) is executed when the condition expression is true. Otherwise, the program control passes to next statement. If the condition expression is false then the C compiler does not do anything.
Note that the condition expression given in parenthesis, must be evaluated as true (non-zero value) or false (zero values). In addition, a compound statement must be provided by opening and closing braces.

Examples:

```
if(7) // a non zero value returns true
 if(0) //zero value returns false
if(i==0) //True
if i=0 other wise False
if(i=0) //false because value of the expression is zero
```

Example 1:

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

Output:

```
Enter a number: 4
4 is even number
This is out of statements

Enter a number: 5
This is out of statements
```

Example 2: **Program to find the largest number of the three.**

```
#include <stdio.h>
Void main()
{
   int a, b, c;
    printf("Enter three numbers?");
   scanf("%d %d %d",&a,&b,&c);
   if(a>b && a>c)
   {
      printf("%d is largest",a);
   }
   if(b>a  && b > c)
   {
      printf("%d is largest",b);
   }
   if(c>a && c>b)
   {
      printf("%d is largest",c);
   }
   if(a == b && a == c)
   {
      printf("All are equal");
   }
}
```

Output

```
Enter three numbers?
12 23 34
34 is largest
```

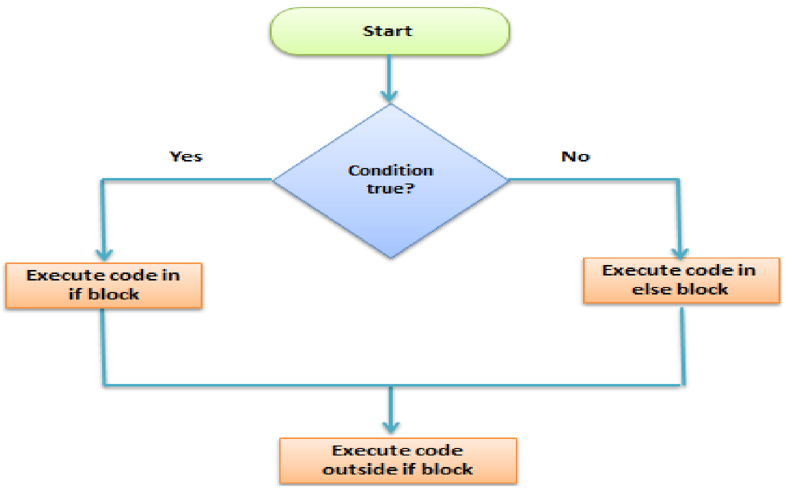
if-else statement:

- The if-else statement is used to perform two operations for a single condition.
- The if-else statement is an extension to the if statement using which, we can perform two different operations, i.e., one is for the correctness of that condition, and the other is for the incorrectness of the condition.

Syntax is:

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

The flow chart for the if-else condition:



Example 1: write a C code to check whether the given number is even or odd

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

Example 2: write a c code to find biggest of two numbers using if else statement

```
#include<stdio.h>
Void main()
{
        int a,b;
        printf("enter a and b value");
        scanf("%d%d",&a,&b);
        if(a>b)
        {
                Printf("a is biggest number");
        }
        Else
        {
                Printf(" b is biggest number");
        }
}
```

Null else, nested if, examples

Null Else: if else statement does not have any executable statements then it is consider as null else statement. So else part will contain only (;) because it doesn't have any executable statements.

The syntax is as follows:

```
if(test-condition)
{
        true-block statements;
}
else;
```

<u>Example:</u>

```
#include<stdio.h>
Void main()
{
int sal;
printf("enter a salary");
scanf("%d",&sal);
if(sal>10000)
{
    sal=sal+1000;
}
else;
}
```

Nested if –else statement:

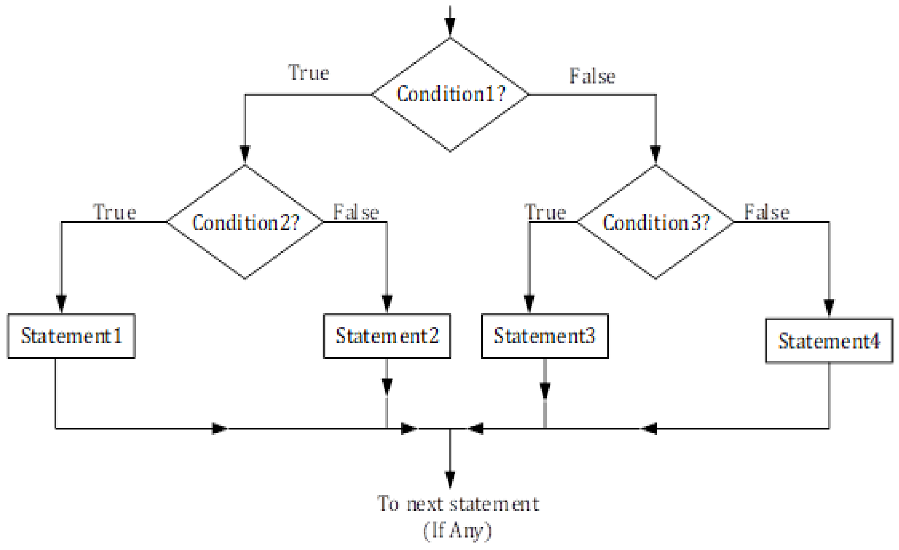
When an if-else is included within an if-else, it is known as a nested if statement. The control of a program moves into the inner if statements when the outer if statement is evaluated to be true. When a series of decision is required, nested if-else is used.

Syntax of nested if statement:

```
if(condition)
{
        //Nested if else inside the body of "if"
     if(condition2)
    {
        //Statements inside the body of nested "if"
     }
    else
    {
         //Statements inside the body of nested "else"
     }
}
else
```

```
        {
                //Statements inside the body of "else"
        }
```

Flowchart:



Example1: write C code to find the biggest of three numbers

```
#include<stdio.h>
Void main()
{
       int a,b;
       printf("enter three values");
       scanf("%d%d%d",&a,&b,&c);
             if(a>b)
             {
                if(a>c)
                    printf("Biggest Number=%d",a);
               else
                    printf("Biggest number=%d",c);
             }
             else
             {
                if(b>c)
                   printf("Biggest Number=%d",b);
                else
                   printf("Biggest number=%d",c);
             }
 }
```

Example 2:

```
       #include<stdio.h>
       Void main()
       {
                int password;
                char username;
                printf("enter user name");
                scanf("%c",&username);
                printf("enter password");
                scanf("%d",&password);
                if(username=='a')
                {
                                If(password==123)
                                                printf("login sucessfull");

                                else
```

```
                    printf("invalid password");
            }
             else
             {
                    printf("invalid username");

             }

        }
```

Example 3:

```
#include<stdio.h>
Void main()
{
        int var1, var2;
        printf("enter var1 and var2");
        scanf("%d%d",&var1,&var2);
        if(var1!=var2)
        {
                if(var1>var2)
                {
                        Printf("var1 is greater than var2");
                }
                Else
                {
                        Printf("var2 is greater than var1");
                }
        else
        {
                Printf("var1 is equal to var2");
        }
}
```

Multi-way selection statements

- A multi-way selection statement is used to execute at most ONE of the choices of a set of statements presented.
- There are two multi-way selection statements:
    - else if ladder
    - Switch case

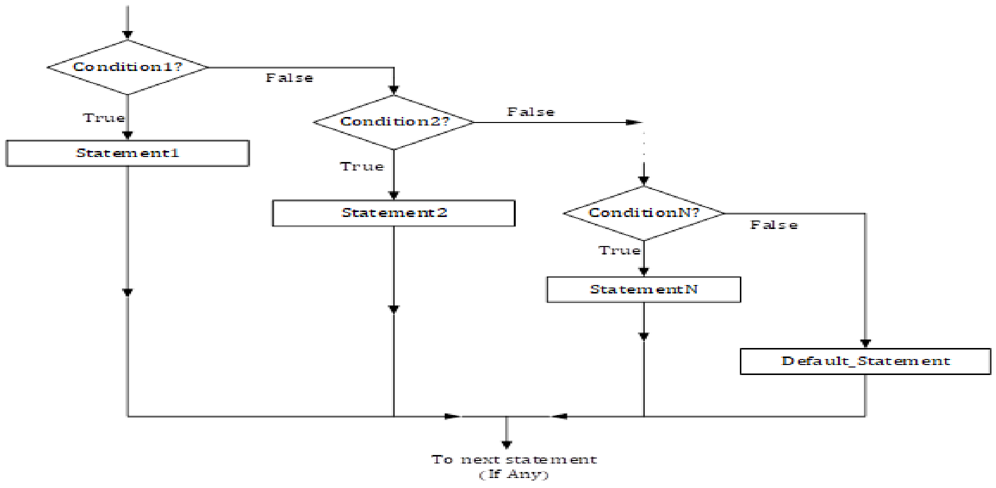
Else if ladder:

- It is used in the scenario where there are multiple cases to be performed for different conditions.
- In if-else-if ladder statement, if a condition is true then the statements defined in the if block will be executed, otherwise if some other condition is true then the statements defined in the else-if block will be executed, at the last if none of the condition is true then the statements defined in the else block will be executed.

Else if ladder (syntax):

```
if(condition1){

        //code to be executed if condition1 is true

}else if(condition2){

        //code to be executed if condition2 is true

}

else if(condition3){

        //code to be executed if condition3 is true

}
```

```
...
else{
        //code to be executed if all the conditions are false
}
```



Example 1:

```
#include <stdio.h>
void main ()
{
    int a = 100;
    if( a == 10 )
   {
        printf("Value of a is 10\n" );
   }
   else if( a == 20 )
   {
        printf("Value of a is 20\n" );
   }
  else if( a == 30 )
  {
      printf("Value of a is 30\n" );
  }
  else
  {
      printf("None of the values is matching\n" );
  }
 printf("Exact value of a is: %d\n", a );

}
```

Example 2:

```
 #include<stdio.h>
void main()
{
        int marks;
        printf("enter your marks");
        scanf("%d",&marks);
        if(marks>=90)
        {
                printf("YOUR GRADE :A\n");
        }
         else if(marks>=70 && marks<90)
        {
                printf(“YOUR GRADE: B\n”);
        }
         else if(marks>=50 && marks<70)
        {
                printf(“YOUR GRADE:C\n”);

        }
        Else
        {
                    printf(“YOUR GRADE: FAILED\n”);
        }
}
```

Switch case:

The switch statement in C is an alternate to if-else-if ladder statement which allows us to execute multiple operations for the different possibilities values of a single variable called switch variable. A switch statement is a conditional statement used in C programming to check the value of a variable and compare it with all the cases. If the value is matched with any case, then its corresponding statements will be executed. Each case has some name or number known as the identifier. The value entered by the user will be compared with all the cases until the case is found. If the value entered by the user is not matched with any case, then the default statement will be executed.

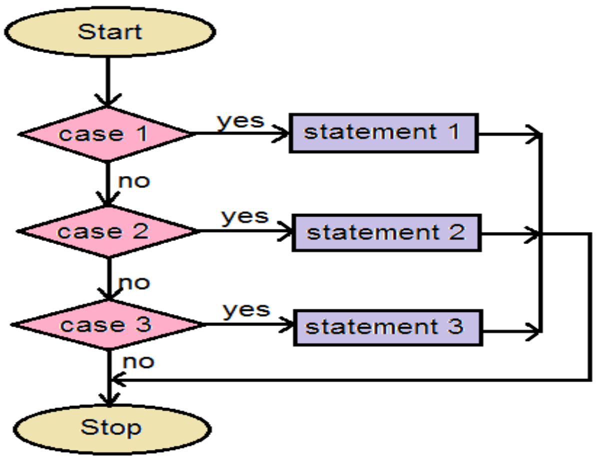
Syntax:

```
switch(expression){
case value1:
      //code to be executed;
      break;  //optional
case value2:
      //code to be executed;
      break;  //optional
 default:
               code to be executed if all cases are not matched;
}
```

The following rules apply to a **switch** statement −

- The **expression** used in a **switch** statement must have an integral or enumerated type, or be of a class type in which the class has a single conversion function to an integral or enumerated type.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The **constant-expression** for a case must be the same data type as the variable in the switch, and itmust be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a **break** statement is reached.
- When a **break** statement is reached, the switch terminates, and the flow of control jumps to the nextline following the switch statement.
- Not every case needs to contain a **break**. If no **break** appears, the flow of control will *fall through* to subsequent cases until a break is reached.
- A **switch** statement can have an optional **default** case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No **break** is needed in the default case.

Flow chart:



Programs on switch caseExample

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
        Int  a,b,c,ch;
        clrscr();
        printf("\nenter value of a and b:");
        scanf("%d %d",&a,&b);
        printf("\nlist:\n1.Addition\n2.Subtraction\n3.Multiplication\n4.Modulus");
        printf("\nEnteryour choice:");
        scanf("%d",&ch);

        switch(ch)
        {
        case 1: c=a+b;
                break;
        case2: c=a-b;
                break;
        case 3: c=a*b;
                break;
        case 4: c=a/b;
                break;
        default: printf("wrong option");
                break;
        }
        printf("\nResult of %d and %d is %d",a,b,c);
        getch();
        }

   printf(“YOUR GRADE: FAILED\n”);
}
```

Example:

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

Repetition and Unconditional Control Statements

What is a loop?

- A **Loop** executes the sequence of statements many times until the stated condition becomes false.
- A loop consists of two parts, a body of a loop and a control statement.
- The control statement is a combination of some conditions that direct the body of the loop to execute until the specified condition becomes false.
- The purpose of the loop is to repeat the same code a number of times.

Types of loops:

- Depending upon the position of a control statement in a program, looping in C is classified into two types:
  1. Entry controlled loop (or) pre-tested loop
  2. Exit controlled loop (or) post- tested loop
- In an **entry controlled loop,** a condition is checked before executing the body of a loop. It is also called as a pre-checking loop.
  Example: while loop, for loop
- In an **exit controlled loop**, a condition is checked after executing the body of a loop. It is also called as a post-checking loop.
  Example: do while loop

Pre- tested loops:

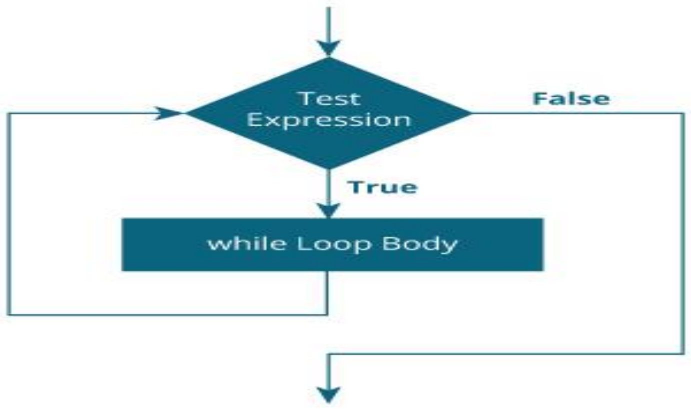
(i) **While loop:** A while loop in C programming repeatedly executes a target statement until the condition becomes false.

Syntax:

```
While (condition)
{
        //statements;
        Increment/decrement;
}
```

Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any nonzero value. The loop iterates while the condition is true. When the condition becomes false, the program control passes to the line immediately following the loop.

Flow chart:



Here, the key point to note is that a while loop might not execute at all. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

Example 1:

```
#include<stdio.h>
Void main()
{
        int i=1;
        while(i<=7)
        {
                printf("%d\t", i);
                i++;
        }
}
```

Output: **1 2 3 4 5 6 7**

Example 2:

```
#include<stdio.h>
Void main()
{
        Int num,i=1;
        Printf("enter a number");
        Scanf("%d", &num);
        While(i<=num)
        {
                Printf("the value is=%d", num);
                i++;
        }
}
```

(ii) **For loop:** A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

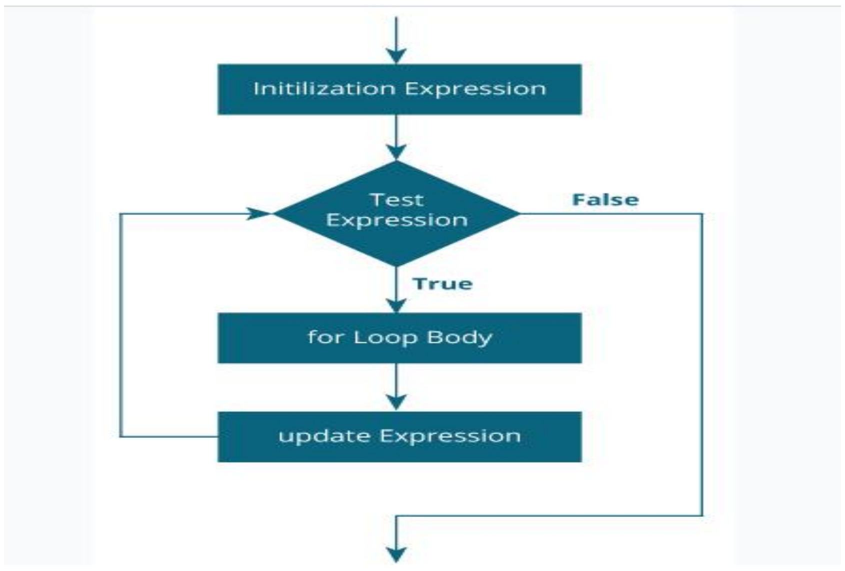
Syntax:

```
for(initialization; condition; increment/decrement)
{
        //statements;
}
```

Here is the flow of control in a 'for' loop −

- The **init** step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
- Next, the **condition** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and the flow of control jumps to the next statement just after the 'for' loop.
- After the body of the 'for' loop executes, the flow of control jumps back up to the **increment** statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.
- The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the 'for' loop terminates.

Flow chart:



Example1:

```
#include<stdio.h>
void main()
{
   int i, num=6;
   for(i=1;i<=5;i++)
   {
      printf("%d*%d=%d\n", num, i,num*i);
   }
}
```

Output:

```
6 * 1 = 6
6 * 2= 12
6 * 3 = 18
6 * 4 = 24
6 * 5= 30
```

Example 2:

```
#include<stdio.h>
Void main()
{
        int i;
        for( i=1;i<=5;i++)
         {
              Printf("%d", i);
         }
}
```

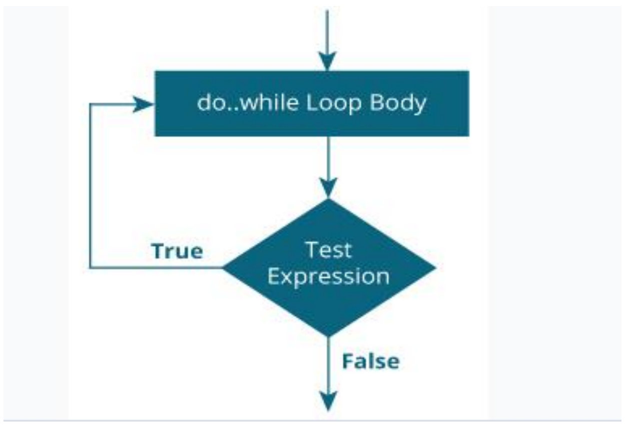
Output:

```
1
2
3
4
5
```

(iii) **do while loop**: In do-while, the body of a loop is always executed at least once. After the body is executed, then it checks the condition. If the condition is true, then it will again execute the body of a loop otherwise control is transferred out of the loop**.**

The syntax of do while loop is:

```
do
{
        //statements;
        increment/decrement;
}while(condition);
```



Example 1:

```
#include<stdio.h>
void main()
{
    int i=1;
    do
    {
        printf("%d\n",i);
        i++;
    }while(i<=5);
}
```

Output:

```
1
2
3
4
5
```

Example 2:

```
// 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: 1.5
Enter a number: 2.4
Enter a number: -3.4
Enter a number: 4.2
Enter a number: 0
Sum = 4.70
```

Infinite loop:

- An infinite loop is a looping construct that does not terminate the loop and executes the loop forever. It is also called an indefinite loop or an endless loop. It either produces a continuous output or no output.
- We can create an infinite loop through various loop structures.
    - for loop
    - while loop
    - do while loop

Infinite For loop:

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

Infinite while loop:

```
#include <stdio.h>
void main()  {
     int i=1;
   while(1)
   {
       i++;
     printf("%d", i);
   }
}
```

Do while loop:

```
#include <stdio.h>
void main()  {
     int i=1;
   do
   {
       i++;
     printf("%d", i);
   }while(1);
}
```

Nested loops

- The loop consists another loop is called nested loops. Any number of loops can be defined inside another loop, i.e., there is no restriction for defining any number of loops.
- It has possibility to define any type of loop inside another loop; for example, we can define '**while**' loop inside a '**for**' loop

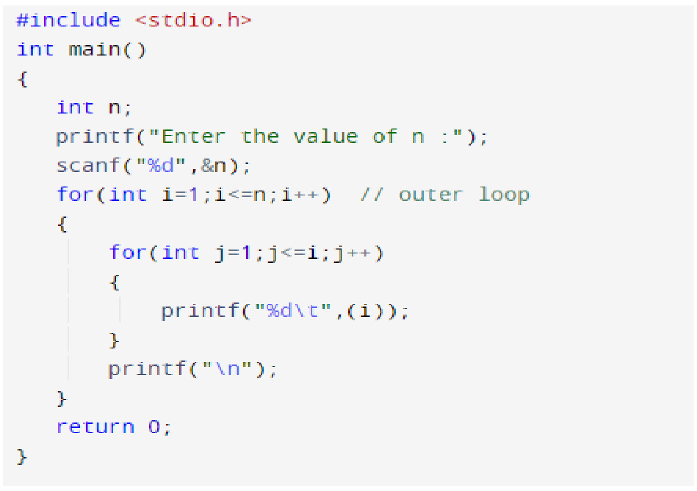
Syntax for nested loops:

```
Outer_loop{
                inner_loop{
                //inner loop statements
                }
                //outer loop statements
}
```

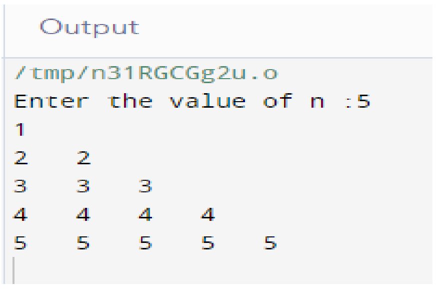
Nested for loop:

Syntax for nested for loop:

```
for (initialization; condition; increment/decrement)
{
   for(initialization; condition; increment/decrement)
   {
        // inner loop statements.
   }
   // outer loop statements.
}
```



```
#include <stdio.h>
int main()
{
    int n;
    printf("Enter the value of n :");
    scanf("%d",&n);
    for(int i=1;i<=n;i++)  // outer loop
    {
        for(int j=1;j<=i;j++)
        {
            printf("%d\t",(i));
        }
        printf("\n");
    }
    return 0;
}
```



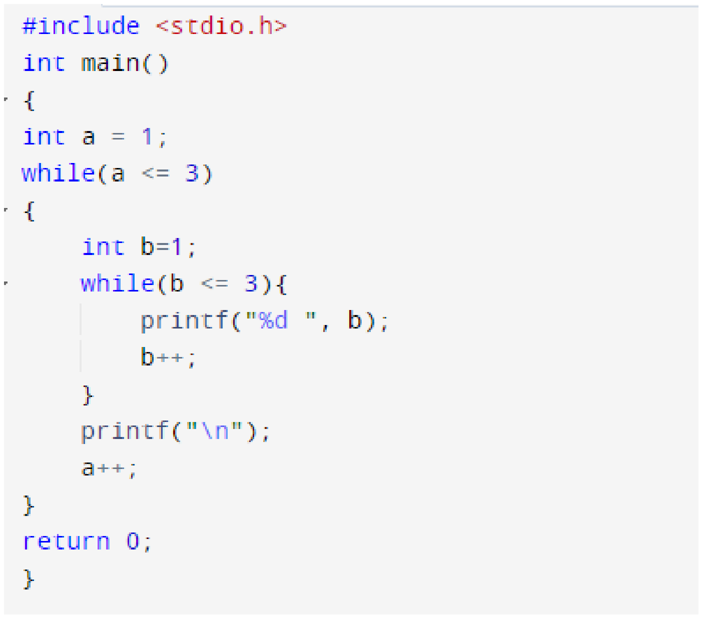
Output

```
/tmp/n31RGCGg2u.o
Enter the value of n :5
1
2    2
3    3    3
4    4    4    4
5    5    5    5    5
```

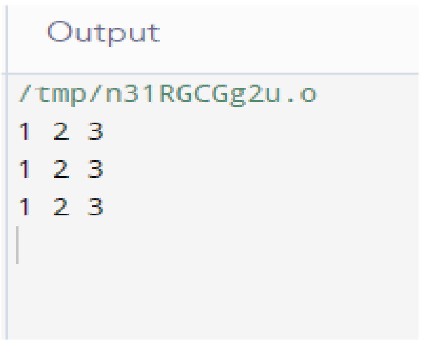
Nested while loop:

Syntax for nested while loop:

```
while(condition)
{
   while(condition)
   {
      // inner loop statements.
   }
// outer loop statements.
}
```



```
#include <stdio.h>
int main()
{
int a = 1;
while(a <= 3)
{
    int b=1;
    while(b <= 3){
        printf("%d ", b);
        b++;
    }
    printf("\n");
    a++;
}
return 0;
}
```



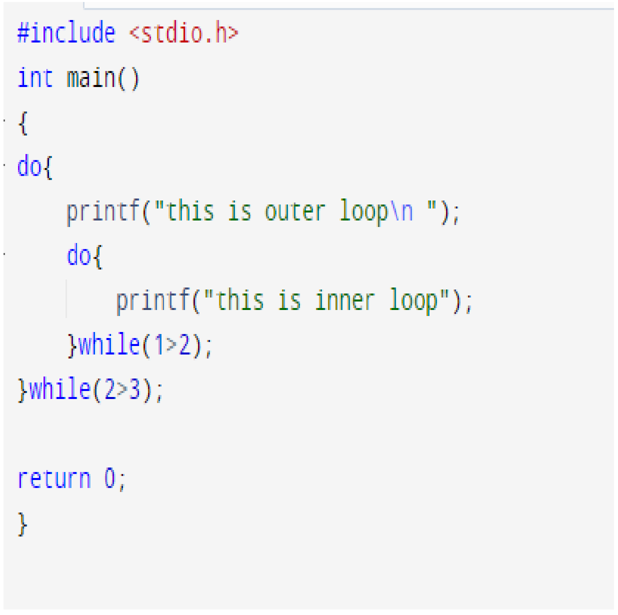
Output

```
/tmp/n31RGCGg2u.o
1 2 3
1 2 3
1 2 3
```

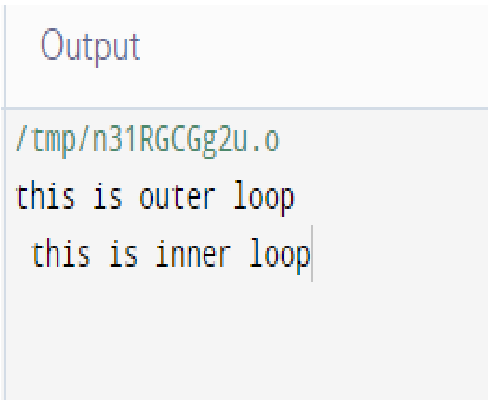
Nested do while:

Syntax for do while loop:

```
do
{
  do
 {
    // inner loop statements.
  }while(condition);
// outer loop statements.
}while(condition);
```



```
#include <stdio.h>
int main()
{
do{
    printf("this is outer loop\n ");
    do{
        printf("this is inner loop");
    }while(1>2);
}while(2>3);

return 0;
}
```



Output

```
/tmp/n31RGCGg2u.o
this is outer loop
 this is inner loop
```

JUMPING STATEMENTS

Jump Statement makes the control jump to another section of the program unconditionally when encountered. It is usually used to terminate the **loop** or **switch-case** instantly. It is also used to escape the execution of a section of the program and in infinite loops. The following are the jumping statements:

1. Break
2. continue
3. goto

1. Break:

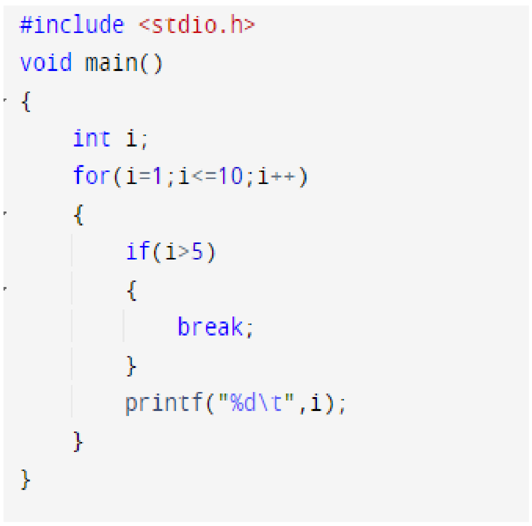
The break is a keyword in C which is used to bring the program control out of the loop. The break statement is used inside loops or switch statement. The break statement breaks the loop one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops. The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

1. With switch case
2. With loop

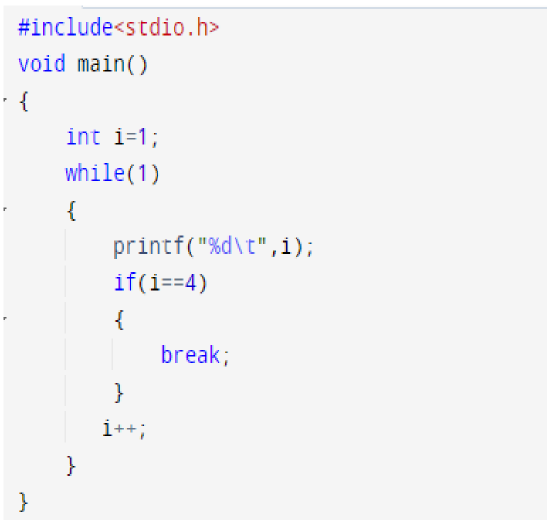
Syntax:

```
//loop or switch case
break;
```

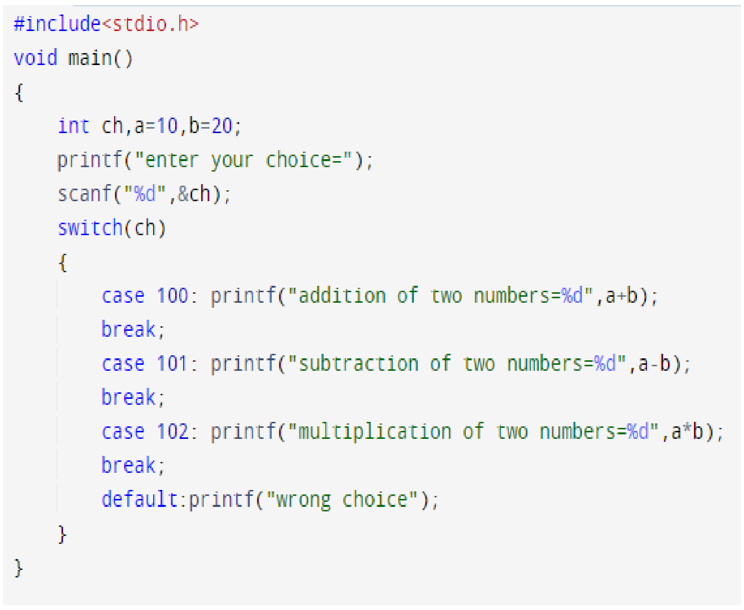
Example:



```c
#include <stdio.h>
void main()
{
    int i;
    for(i=1;i<=10;i++)
    {
        if(i>5)
        {
            break;
        }
        printf("%d\t",i);
    }
}
```



```c
#include<stdio.h>
void main()
{
    int i=1;
    while(1)
    {
        printf("%d\t",i);
        if(i==4)
        {
            break;
        }
       i++;
    }
}
```



```c
#include<stdio.h>
void main()
{
    int ch,a=10,b=20;
    printf("enter your choice=");
    scanf("%d",&ch);
    switch(ch)
    {
        case 100: printf("addition of two numbers=%d",a+b);
        break;
        case 101: printf("subtraction of two numbers=%d",a-b);
        break;
        case 102: printf("multiplication of two numbers=%d",a*b);
        break;
        default:printf("wrong choice");
    }
}
```

2. Continue:

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

Syntax:

```
//loop or switch case
continue;
```

Example:

```
#include<stdio.h>
void main()
{
    int i;
    for(i=1;i<=10;i++)
    {
        if(i==5)
        {
            continue;
        }
        printf("%d\t",i);
    }
}
```

```
#include <stdio.h>
void main()
{
    int i=1;
    while (i<=10)
    {
     if (i==7)
     {
          i++;
          continue;
     }
     printf("%d ", i);
     i++;
    }
}
```

3. goto:

The goto statement is known as jump statement in C. As the name suggests, goto is used to transfer the program control to a predefined label. The goto statement can be used to repeat some part of the code for a particular condition. It can also be used to break the multiple loops which can't be done by using a single break statement.

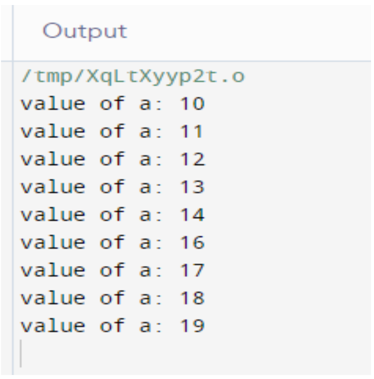
Syntax:

```
label:
    //some part of the code;
goto label;
```

Note: The label is an identifier. When the goto statement is encountered, the control of the program jumps to label: and starts executing the code

Examples:

```
#include <stdio.h>
void main () {
   int a = 10;
   LOOP:do {
      if( a == 15) {
         a = a + 1;
         goto LOOP;
      }
      printf("value of a: %d\n", a);
      a++;
   }while( a < 20 );
}
```



Output

```
/tmp/XqLtXyyp2t.o
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```