

20ES1103 Programming for Problem Solving

UNIT II

Introduction to the C Language

Background of C program

Brief History of C

- The C programming language was developed in the Bell Labs of AT&T by an employee called Dennis Ritchie between 1969 and 1973 while working on Unix operating system.
- He created this language using **ALGOL**, **BCPL**, and **B** the languages that were used before C was created.
- Ritchie added many powerful features to C and used it to further develop the UNIX operating system.
- **American National Standards Institute (ANSI)** in 1983, formed a committee to provide a comprehensive definition to the C language and thus came into existence the new ANSI C language with better features.

What is C Character set?

C Character set is a collection of characters supported in C programming language.

C Programming language has a rich set of characters which are used to construct c program instructions.

What does C Character Set contains?

Alphabets

- C Language supports all alphabets of English. It supports both UPPERCASE & lowercase letters

Digits

- C Language supports 10 digits to construct numbers. Those 10 digits are 0,1,2,3,4,5,6,7,8,9

Special Symbols

- C supports a rich set of special symbols that include symbols to perform mathematical operations, condition checking, white space, back space, etc...

Commonly used Special Symbols with ASCII Values

These are Control Characters

| ASCII Value | Character | Meaning |
|-------------|-----------|--------------------|
| 0 | NULL | null |
| 1 | SOH | Start of header |
| 2 | STX | start of text |
| 3 | ETX | end of text |
| 4 | EOT | end of transaction |
| 5 | ENQ | enquiry |
| 6 | ACK | acknowledgement |
| 7 | BEL | bell |
| 8 | BS | back Space |
| 9 | HT | Horizontal Tab |
| 10 | LF | Line Feed |
| 11 | VT | Vertical Tab |
| 12 | FF | Form Feed |
| 13 | CR | Carriage Return |
| 14 | SO | Shift Out |
| 15 | SI | Shift In |
| 16 | DLE | Data Link Escape |
| 17 | DC1 | Device Control 1 |
| 18 | DC2 | Device Control 2 |
| 19 | DC3 | Device Control 3 |

These are Printable Characters

| ASCII Value | Character |
|-------------|-----------|
| 32 | Space |
| 33 | ! |
| 34 | " |
| 35 | # |
| 36 | \$ |
| 37 | % |
| 38 | & |
| 39 | ' |
| 40 | (|
| 41 |) |
| 42 | * |
| 43 | + |
| 44 | , |
| 45 | - |
| 46 | . |
| 47 | / |
| 48 | 0 |
| 49 | 1 |
| 50 | 2 |
| 51 | 3 |

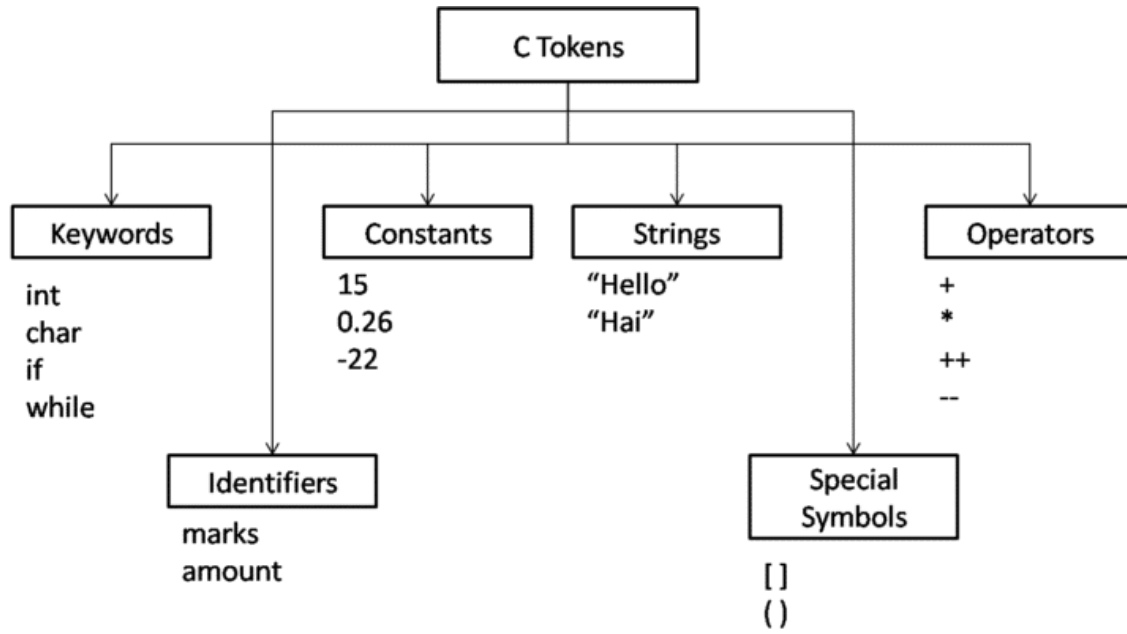
| ASCII Value | Character |
|-------------|-----------|
| 64 | @ |
| 65 | A |
| 66 | B |
| 67 | C |
| 68 | D |
| 69 | E |
| 70 | F |
| 71 | G |
| 72 | H |
| 73 | I |
| 74 | J |
| 75 | K |
| 76 | L |
| 77 | M |
| 78 | N |
| 79 | O |
| 80 | P |
| 81 | Q |
| 82 | R |
| 83 | S |

| ASCII Value | Character |
|-------------|-----------|
| 96 | ` |
| 97 | a |
| 98 | b |
| 99 | c |
| 100 | d |
| 101 | e |
| 102 | f |
| 103 | g |
| 104 | h |
| 105 | i |
| 106 | j |
| 107 | k |
| 108 | l |
| 109 | m |
| 110 | n |
| 111 | o |
| 112 | p |
| 113 | q |
| 114 | r |
| 115 | s |

C Tokens:

In C programs, each word and punctuation is referred to as a token.

C Tokens are the smallest building block or smallest unit of a C program.



Identifiers

C Identifiers

- Identifier refers to name given to entities such as variables, functions, structures etc.
- Identifiers must be unique. They are created to give a unique name to an entity to identify it during the execution of the program. For example:
 - ❑ `int money;`
 - ❑ `double accountBalance;`
- Here, money and accountBalance are identifiers.
- Also remember, identifier names must be different from keywords. You cannot use `int` as an identifier because `int` is a keyword.

- **Rules for naming identifiers**

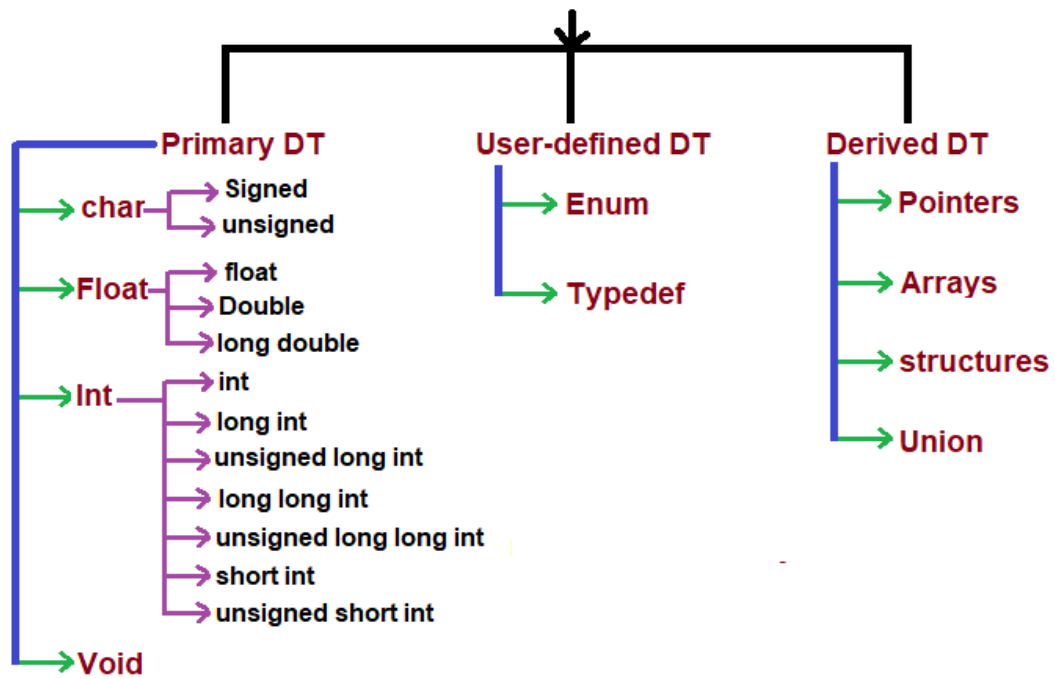
- o First character must be an alphabet (or underscore).
- o Must consist of only letters(**a-z,A-Z**), digits(**0-9**) or underscore.
- o Only first 31 characters are significant.
- o Cannot use a Keyword.
- o Must not contain white space.

Types

Datatypes:

DT - Data type

Data Types in C



| Type | Storage size | Value range |
|----------------|--------------|--|
| char | 1 byte | -128 to 127 or 0 to 255 |
| unsigned char | 1 byte | 0 to 255 |
| signed char | 1 byte | -128 to 127 |
| 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 |
| short | 2 bytes | -32,768 to 32,767 |
| unsigned short | 2 bytes | 0 to 65,535 |
| long | 4 bytes | -2,147,483,648 to 2,147,483,647 |
| unsigned long | 4 bytes | 0 to 4,294,967,295 |

| Variable Type | Keyword | Bytes Required | Range | Format |
|-----------------------|---------------|----------------|--------------------------------|--------|
| Character (signed) | Char | 1 | -128 to +127 | %c |
| Integer (signed) | Int | 2 | -32768 to +32767 | %d |
| Float (signed) | Float | 4 | -3.4e38 to +3.4e38 | %f |
| Double | Double | 8 | -1.7e308 to +1.7e308 | %lf |
| Long integer (signed) | Long | 4 | 2,147,483,648 to 2,147,438,647 | %ld |
| Character (unsigned) | Unsigned char | 1 | 0 to 255 | %c |
| Integer (unsigned) | Unsigned int | 2 | 0 to 65535 | %u |
| Unsigned long integer | unsigned long | 4 | 0 to 4,294,967,295 | %lu |
| Long double | Long double | 10 | -1.7e932 to +1.7e932 | %Lf |

Void Types:

The void type has no values. It is used to represent the type of function. The type of function is said to be void when it does not return any value to the calling function.

Variables

Variables in C

- A variable is nothing but a name given to a storage area that our programs can manipulate.
- A variable definition tells the compiler where and how much storage to create for the variable.
- A variable definition specifies a data type and contains a list of one or more variables of that type as follows –

Syntax:

type variable_list;

Ex:

```
int i, j, k;  
char c, ch;  
float f, salary;  
double d;
```

type variable_name = value;

```
int d = 3, f = 5;    // declaration and initializing d and f.  
char x = 'x';       // the variable x has the value 'x'.
```

Constants

Constants

- C Constants are also like normal variables. But, only difference is, their values can not be modified by the program once they are defined.
- Constants refer to fixed values. They are also called as literals
- Constants may be belonging to any of the data type.

Syntax:

const type constant_name;

```
#include<stdio.h>
main()
{
    const int SIDE = 10;
    int area;
    area = SIDE*SIDE;
    printf("The area of the square with side: %d is: %d sq. units"
        , SIDE, area);
}
```

- **TYPES OF C CONSTANT:**

- Integer constants
- Real or Floating point constants
- Octal & Hexadecimal constants
- Character constants
- String constants
- Backslash character constants

Input/Output

Input/Output

- **Input** means to provide the program with some data to be used in the program and **Output** means to display data on screen or write the data to a printer or a file.
- The standard input-output header file, named `stdio.h` contains the definition of the functions `printf()` and `scanf()`, which are used to display output on screen and to take input from user respectively.

printf()

- The printf() method, in C, **prints** the value passed as the parameter to it, on the console screen.

printf(“%X”, variableOfXType);

- where %X is the [format specifier in C](#). It is a way to tell the compiler what type of data is in a variable

scanf()

- The scanf() method, in C, reads the value from the console as per the type specified.

scanf("%X", &variableOfXType);

- where %X is the [format specifier in C](#).
- It is a way to tell the compiler what type of data is in a variable
- & is the address operator in C, which tells the compiler to change the real value of this variable, stored at this address in the memory.

Keywords

- Keywords are pre-defined words in a C compiler.
- Each keyword is meant to perform a specific function in a C program.
- Since keywords are referred names for compiler, they can't be used as variable name.
- C language supports 32 keywords which are given below.

Keywords in C Language

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

| 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 |
| \N | Octal constant (N is an octal constant) |
| \XN | Hexadecimal constant (N – hex.dcm1 cnst) |

Structure of a C Program

// Name of Program → Documentation section

#include<stdio.h> }
#include<conio.h> } → Preprocessor Directives

#define max 100 → Definition section

void add() ; }
int x=100; } → Global declaration section

int main() → main () Function section / Entry Point

{ int a=100; → Variable declaration

printf("Hello Main"); }
return 0; } → Body of Main function
}

void add(){
printf("Hello add"); } → Function Definition
}

Example Program

BASIC STRUCTURE OF A 'C' PROGRAM:

| |
|--|
| Documentation section [Used for Comments] |
| Link section |
| Definition section |
| Global declaration section [Variable used in more than one function] |
| main() { Declaration part Executable part } |
| Subprogram section [User-defined Function] Function1 Function 2 : : Function n |

Example:

→ //Sample Prog Created by:Bsource

→ #include<stdio.h>
→ #include<conio.h>

→ void fun();

→ int a=10;

→ void main()
→ {
→ clrscr();
→ printf("a value inside main(): %d",a);
→ fun();
→ }

→ void fun()
→ {
→ printf("\na value inside fun(): %d",a);
→ }

//Program to add two numbers:

```
#include <stdio.h>
void main()
{

    int number1, number2, sum;

    printf("Enter two integers: ");
    scanf("%d %d", &number1, &number2);

    // calculating sum
    sum = number1 + number2;

    printf("%d + %d = %d", number1, number2, sum);
    getch();
}
```


Logical Data and Operators

C – Operators

- The symbols which are used to perform logical and mathematical operations in a C program are called C operators.
- These C operators join individual constants and variables to form expressions.
- Operators, functions, constants and variables are combined together to form expressions.
- Consider the expression $A + B * 5$. where, $+$, $*$ are operators, A , B are variables, 5 is constant and $A + B * 5$ is an expression.

Different types of operators

- Arithmetic operators
- Assignment operators
- Relational operators
- Logical operators
- Bit wise operators
- Conditional operators (ternary operators)
- Increment/decrement operators
- Special operators

| Arithmetic Operators/Operation | Example |
|--------------------------------|---------|
| + (Addition) | $A+B$ |
| − (Subtraction) | $A-B$ |
| * (multiplication) | $A*B$ |
| / (Division) | A/B |
| % (Modulus) | $A\%B$ |

Assignment
Operators

| Operators | Example/Description |
|-----------|--|
| = | sum = 10; 10 is assigned to variable sum |
| += | sum += 10; This is same as sum = sum + 10 |
| -= | sum -= 10; This is same as sum = sum - 10 |
| *= | sum *= 10; This is same as sum = sum * 10 |
| /= | sum /= 10; This is same as sum = sum / 10 |
| %= | sum %= 10; This is same as sum = sum % 10 |
| &= | sum&=10; This is same as sum = sum & 10 |
| ^= | sum ^= 10; This is same as sum = sum ^ 10 |

Relational Operators

| Operators | Example/Description |
|-----------|--|
| > | $x > y$ (x is greater than y) |
| < | $x < y$ (x is less than y) |
| >= | $x \geq y$ (x is greater than or equal to y) |
| <= | $x \leq y$ (x is less than or equal to y) |
| == | $x == y$ (x is equal to y) |
| != | $x != y$ (x is not equal to y) |

Logical Operators

| Operators | Example/Description |
|------------------|---|
| && (logical AND) | <code>(x>5)&&(y<5)</code> It returns true when both conditions are true |
| (logical OR) | <code>(x>=10) (y>=10)</code> It returns true when at-least one of the condition is true |
| ! (logical NOT) | <code>!((x>5)&&(y<5))</code> It reverses the state of the operand “ <code>((x>5) && (y<5))</code> ” If “ <code>((x>5) && (y<5))</code> ” is true, logical NOT operator makes it false |

CONDITIONAL OR TERNARY OPERATORS IN C:

- Conditional operators return one value if condition is true and returns another value if condition is false.
- This operator is also called as ternary operator.

Syntax : (Condition? true_value: false_value);

Example : (A > 100 ? 0 : 1);

- In above example, if A is greater than 100, 0 is returned else 1 is returned. This is equal to if else conditional statements.

big = a > b ? (a > c ? a : c) : (b > c ? b : c) ;

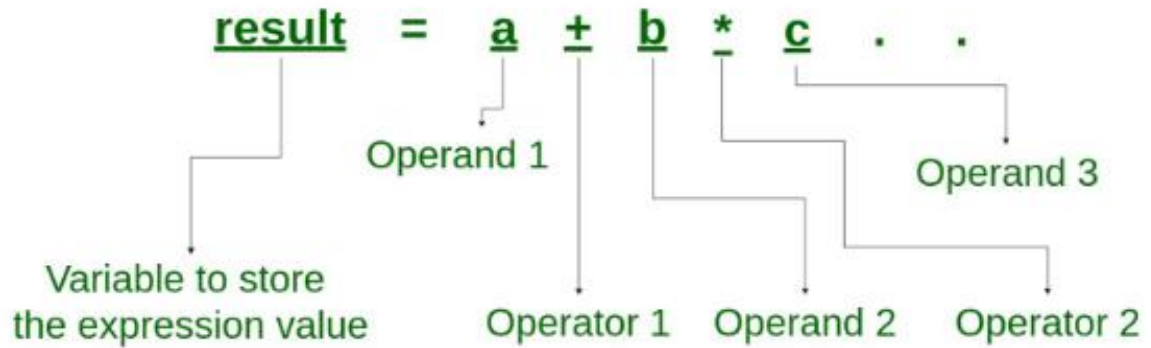
Increment/decrement operators

- Increment operators are used to increase the value of the variable by one and decrement operators are used to decrease the value of the variable by one in C programs.
- Syntax:
Increment operator: ++var_name; (or) var_name++;
Decrement operator: --var_name; (or) var_name --;
- Example:
Increment operator : ++i; i++;
Decrement operator : --i; i--;

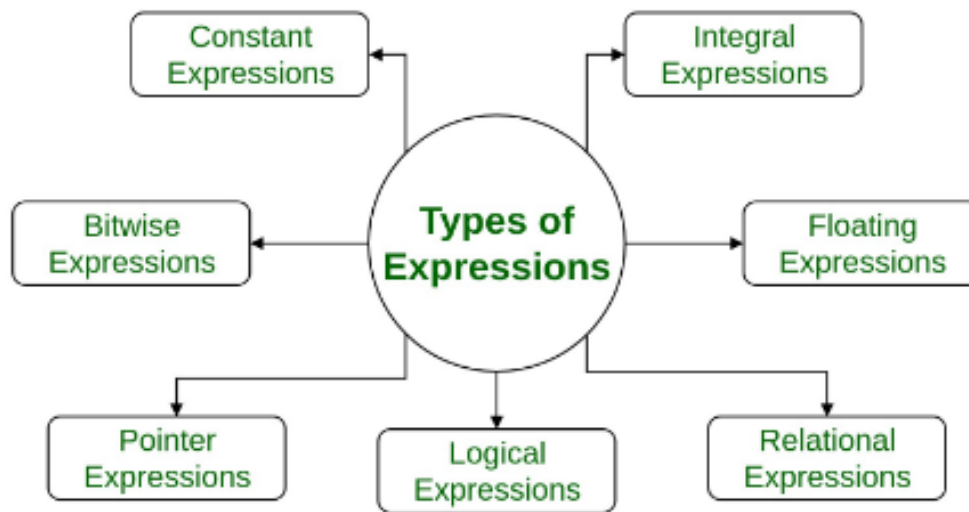
Expressions

Expressions

- An expression is a combination of operators, constants and variables. An expression may consist of one or more operands, and zero or more operators to produce a value.



Types of Expressions



Precedence and Associativity

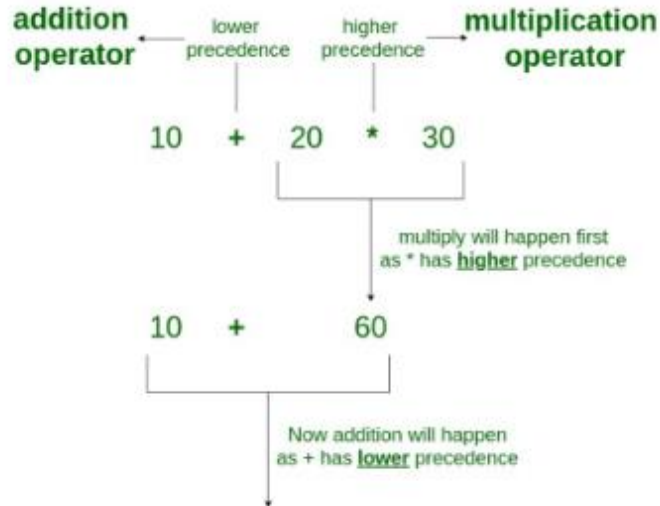
Precedence and Associativity

- The symbols which are used to perform logical and mathematical operations in a C program are called C operators.
- These C operators join individual constants and variables to form expressions.
- Operators, functions, constants and variables are combined together to form expressions.
- Consider the expression $A + B * 5$. where, $+$, $*$ are operators, A , B are variables, 5 is constant and $A + B * 5$ is an expression.

Operator Precedence in C

Operator precedence determines which operator is evaluated first when an expression has more than one operators.

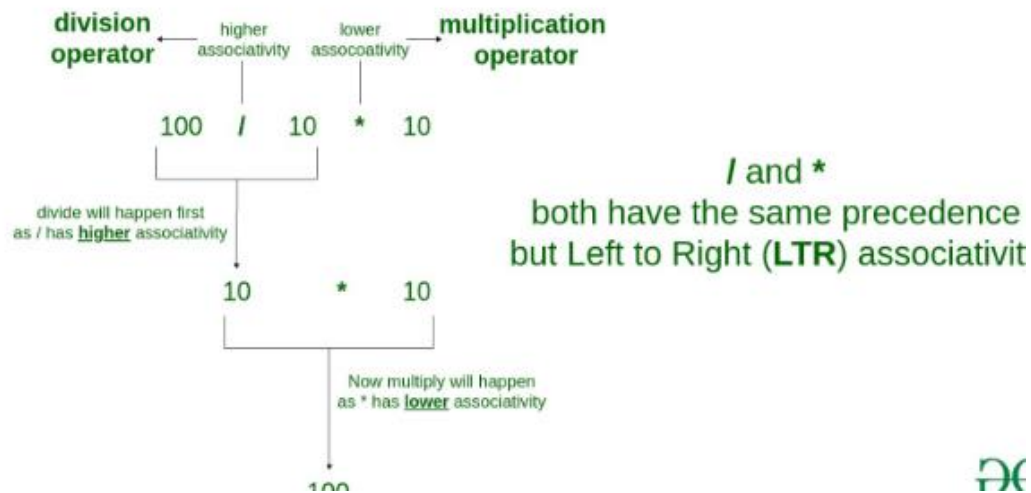
Operator Precedence



Operators Associativity

- **Operators Associativity** is used when two operators of same precedence appear in an expression. Associativity can be either **Left to Right** or **Right to Left**.
- For example multiplication and division arithmetic operators have same precedence, let's say we have an expression $5*2/10$, this expression would be evaluated as $(5*2)/10$ because the associativity is left to right for these operators.

Operator Associativity



| Operator | Priority | Associativity |
|-----------------------|----------|---------------|
| {}, (), [] | 1 | Left to right |
| ++, --, ! | 2 | Right to left |
| *, /, % | 3 | Left to right |
| +, - | 4 | Left to right |
| <, <=, >, >=, ==, != | 5 | Left to right |
| && | 6 | Left to right |
| | 7 | Left to right |
| ?: | 8 | Right to left |
| =, +=, -=, *=, /=, %= | 9 | Right to left |

Evaluating Expressions

Evaluating Expressions

- In the C programming language, an expression is evaluated based on the operator precedence and associativity.
- When there are multiple operators in an expression, they are evaluated according to their precedence and associativity.
- The operator with higher precedence is evaluated first and the operator with the least precedence is evaluated last.

$$10 + 4 * 3 / 2$$

$$4 * 3 ==> 12$$

$$12 / 2 ==> 6$$

$$10 + 6 ==> 16$$

The expression is evaluated to **16**.

Write a C Program to evaluate

```
a=1,b=2,c=3  
x= a-b/3+c*2-1;
```

```
y= a-b/(3+c)*(2-1);  
z=a-(b/(3+c)*2)-1;
```

Type Conversion

Type Conversion

- Type casting refers to changing a variable of one data type into another.
- Two Types
 - ✓ **Implicit Type Conversion**
 - ✓ **Explicit Type Conversion**

Advantages of Type Conversion

- This is done to take advantage of certain features of type hierarchies or type representations.
- It helps us to compute expressions containing variables of different data types.

Implicit Type Conversion

- When the type conversion is performed **automatically by the compiler without programmers** intervention, such type of conversion is known as **implicit type** conversion or type promotion.

```
int x;  
If(x>0)  
{  
printf("%c", x); /*Implicit casting from int to char %c*/  
}
```


Explicit Type Conversion

- The type conversion **performed by the programmer** by posing the data type of the expression of specific type is known as explicit type conversion. The explicit type conversion is also known as type casting.
- Type casting in c is done in the following form:

(data_type)expression;

where, data_type is any valid c data type, and expression may be constant, variable or expression.

```
int x;  
If(x>0)  
{  
    printf("%c", (char)x); /*Explicit casting from int to char*/  
}
```

Example of Typecasting

```
int x=7, y=5 ;
```

```
float z;
```

```
z=x/y;    /*Here the value of z is 1*/
```

If we want to get the exact value of 7/5 then we need explicit casting from int to float:

```
int x=7, y=5;
```

```
float z;
```

```
z = (float)x/(float)y;  /*Here the value of z is 1.4*/
```

```
#include<stdio.h>

void main()
{
    int a = 25, b = 13;
    float result;
    result = a/b;
    // display only 2 digits after decimal point
    printf("(Without typecasting) 25/13 = %.2f\n", result );
    result = (float)a/b;
    // display only 2 digits after decimal point
    printf("(With typecasting) 25/13 = %.2f\n", result );
    getch();
}
```

Statements in C

Statements in C

- The bodies of C functions (including the main function) are made up of statements.
- Two types
 - ✓ Simple statements
 - ✓ Compound Statements
- Simple statements that do not contain other statements
- Compound statements that have other statements inside them.
- Control structures are compound statements like **if/then/else**, **while**, **for**, and **do..while** that control how or whether their component statements are executed.

- Simple Statements
- The simplest kind of statement in C is an expression (followed by a semicolon, the terminator for all simple statements)
 1. `x = 2;` `/* an assignment statement */`
 2. `x = 2+3;` `/* another assignment statement */`
 3. `2+3;` `/* has no effect---will be discarded by smart compilers */`
 4. `puts("hi");` `/* a statement containing a function call */`
 5. `root2 = sqrt(2);` `/* an assignment statement with a function call */`

Compound statements

- Compound statements come in two varieties:

conditionals

loops

- These are compound statements that test some condition and execute one or another block depending on the outcome of the condition.

```
if(houseIsOnFire)
{
    /* ouch! */
    scream();
    runAway();
}
```

The body of the if statement is executed only if the expression in parentheses at the top evaluates to true

Storage Class

storage class

- A storage class represents the visibility and a location of a variable. It tells from what part of code we can access a variable. A storage class in C is used to describe the following things:
- The variable scope.
- The location where the variable will be stored.
- The initialized value of a variable.
- A lifetime of a variable.
- Who can access a variable?

Types Storage Classes

- **There are FOUR types of storage classes**
 - Automatic Storage class (auto)
 - Register Storage class (register)
 - Static Storage class (static)
 - External Storage class (extern)

Auto Storage Class

- The variables defined using auto storage class are called as local variables. Auto stands for automatic storage class.
- The auto storage class is the default storage class for all local variables.

```
{  
    int mount;  
    auto int month;  
}
```

- The example above defines two variables with in the same storage class. 'auto' can only be used within functions, i.e., local variables.

Extern Storage Class in C

- Extern stands for external storage class. Extern storage class is used when we have global functions or variables which are shared between two or more files.
- Keyword extern is used to declaring a global variable or function in another file to provide the reference of variable or function which have been already defined in the original file.
- Example, `extern void display();`

First File: main.c

```
#include <stdio.h>
extern i;
main() {
    printf("value of the external integer is = %d\n", i);
    return 0;}
```

Second File: original.c

```
#include <stdio.h>
i=48;
```

Static Storage Class in C

- The static variables are used within function/ file as local static variables. They can also be used as a global variable
- Static local variable is a local variable that retains and stores its value between function calls or block and remains visible only to the function or block in which it is defined.
- Static global variables are global variables visible only to the file in which it is declared.
- Example: `static int count = 10;`

- `#include <stdio.h>`
- `/* function declaration */`
- `void func(void);`
- `static int count = 5; /* global variable */`
- `main()`

Register storage class

- The register storage class is used to define local variables that should be stored in a register instead of RAM.

```
{  
    register int miles;  
}
```

Selection

Two-way selection

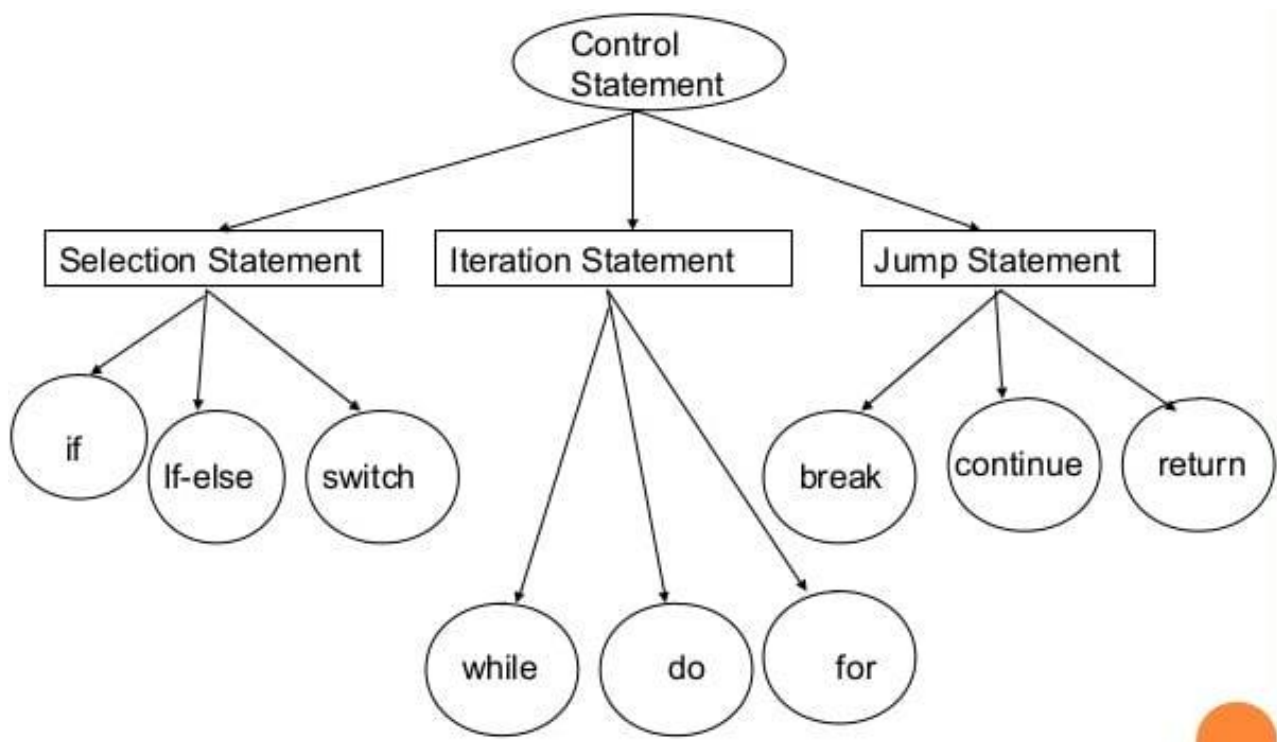
- The two-way selection is the basic decision statement for computers.
- The decision is based on resolving a binary expression, and then executing a set of commands depending on whether the response was true or false.
- C, like most contemporary programming languages, implements two-way selection with the if...else statement.
- An **if...else statement** is a paired statement used to selectively execute code based on **two alternatives**.

Conditional Statement

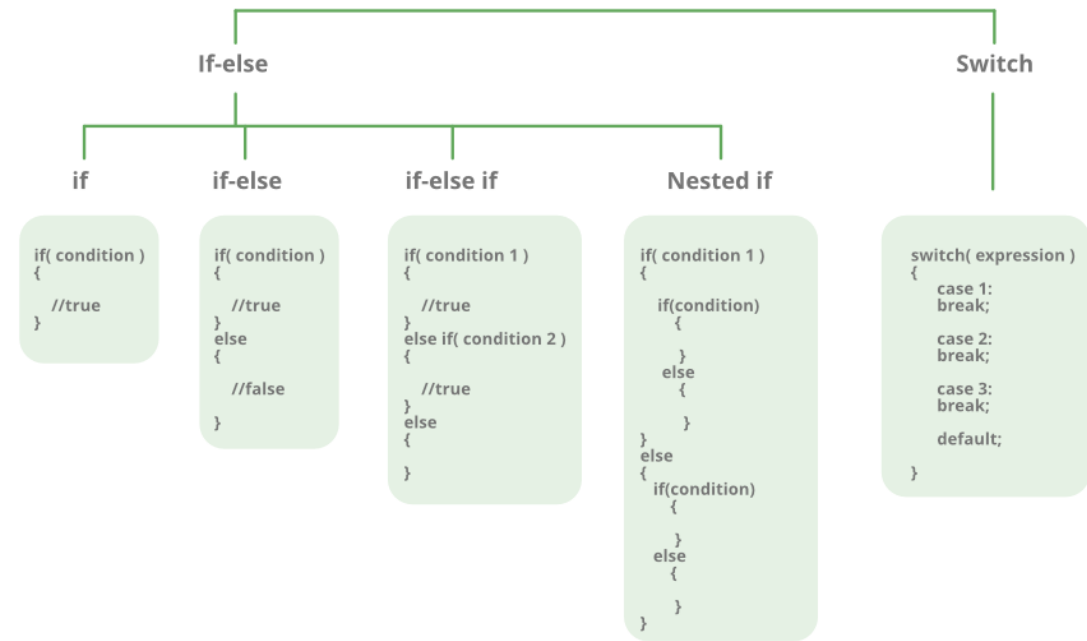


- All the statements written in a program are executed from top to bottom one by one. Conditional / Control statement are use to execute or transfer the control from one part to another depending on a condition.





Decision Making



Syntax: - if (condition)

```
{  
.....  
.....  
}
```

```
Include <stdio.h>  
Include <conio.h>  
Void main()  
{  
Int age;  
Clrscr();  
If(age>18)  
{  
Printf(he is eligible for voting);  
}  
getch();  
}
```

In Another Words, It Can Also Be Said As Single Blocked Conditional Statements In Which Only One Part Is Mentioned. That Is Known As TRUE Part.

Write a program to input two numbers and print the greatest number in C

```
Include <stdio.h>
Include<conio.h>
Void main()
{
Int a,b;
Clrscr();
If(a==b)
{
Printf("a is equal to b");
}
else
If(a>b)
{
Printf("a is greater");
}
else
{
Printf("b is greater");
}
getch();
}
```


Multiway Selection

Multi-way Selection: switch statement

- A multi-way selection statement is used to execute at most ONE of the choices of a set of statements presented.
- Syntax of the multi-way select statement:

switch(EXPRESSION)

{

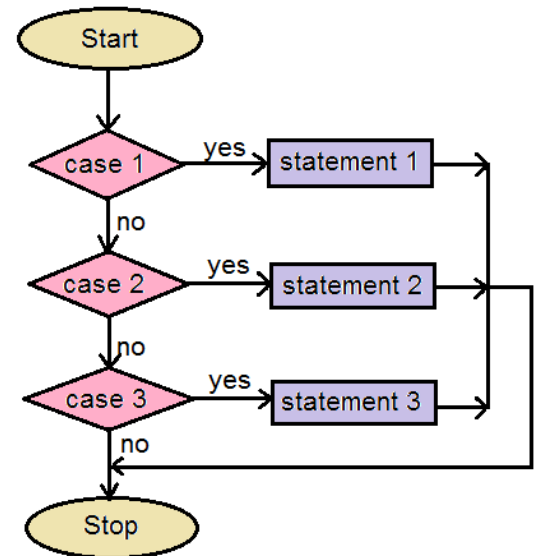
case 1: one or more statements; break;

case 2: one or more statements; break;

...

default: one or more statements;

}



```
switch (expression)
{
    case constant-1: statement
                    :
                    statement
    case constant-2: statement
                    :
                    statement
    case constant-n: statement
                    :
                    statement
    default         : statement
                    :
                    statement
} // end switch
```

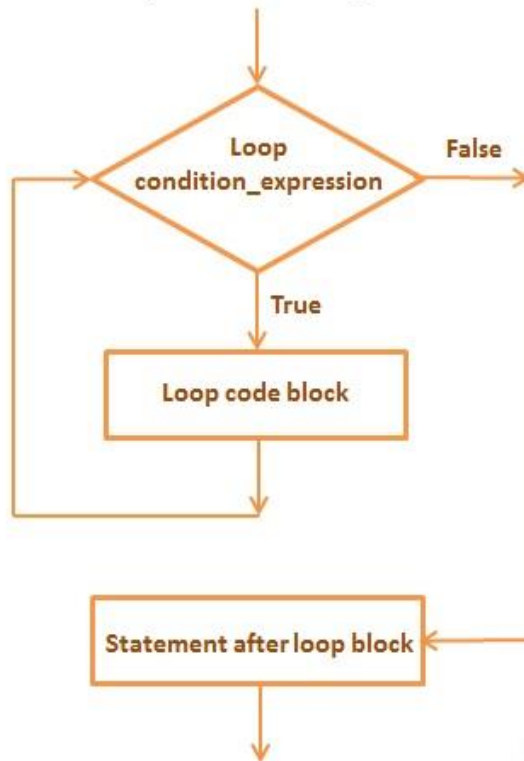
```
1 // Program fragment to demonstrate switch
2 switch (printFlag)
3 {
4     case 1: printf("This is case 1\n");
5
6     case 2: printf("This is case 2\n");
7
8     default: printf("This is default\n");
9 } // switch
```

Repetition: Concept of a Loop

Concept of a Loop

- In programming, a loop is used to repeat a block of code until the specified condition is met.
- 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.

Loop Flow Diagram



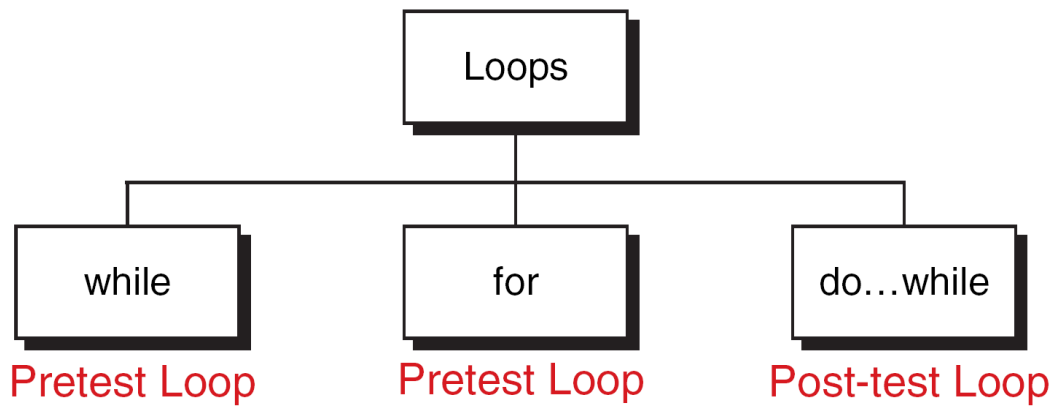
Types of Loops

- Depending upon the position of a control statement in a program, a loop is classified into two types:
 1. Entry controlled loop
 2. Exit controlled 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.
- 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.

- The control conditions must be well defined and specified otherwise the loop will execute an infinite number of times. The loop that does not stop executing and processes the statements number of times is called as an **infinite loop**.
- An infinite loop is also called as an "**Endless loop**." Following are some characteristics of an infinite loop:
 1. No termination condition is specified.
 2. The specified conditions never meet.
- The specified condition determines whether to execute the loop body or not.

- 'C' programming language provides us with three types of loop constructs:
 1. The while loop
 2. The for loop
 3. The do-while loop

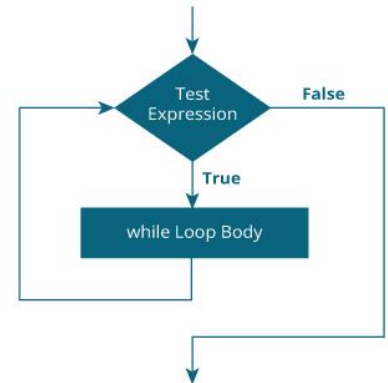
The first two are **pretest loops**, and the third is a **post-test loop**. We can use all of them for event-controlled and counter-controlled loops.



While Loop

- A while loop is the most straightforward looping structure. The basic format of while loop is as follows:

```
while (test Expression)
{
    statements inside the body of the loop
}
```



Example: Write a c program to print series of numbers from 1 to 10 using a while loop.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int num=1; //initializing the variable
    while(num<=10) //while loop with condition
    {
        printf("%d\n",num);
        num++; //incrementing operation
    }
    getch();
}
```

Write a C program to print the Reverse of a given number using while loop.

- For example, if the input is 123, the output will be 321. In the program, we use the modulus operator (%) to obtain digits of the number. To invert the number write its digits from right to left.

```
#include <stdio.h>
void main()
{
    int n, rev = 0, rem;
    printf("Enter an integer: ");
    scanf("%d", &n);
    while (n != 0)
    {
        rem = n % 10;
        rev = rev * 10 + rem;
        n = n / 10;
    }
    printf("Reversed number = %d", rev);
    getch();
}
```

Write a C program to generate Fibonacci series up to n number using while loop.

- The Fibonacci sequence is a sequence where the next term is the sum of the previous two terms. The first two terms of the Fibonacci sequence are 0 followed by 1.
- The Fibonacci sequence:
- 0, 1, 1, 2, 3, 5, 8, 13, 21

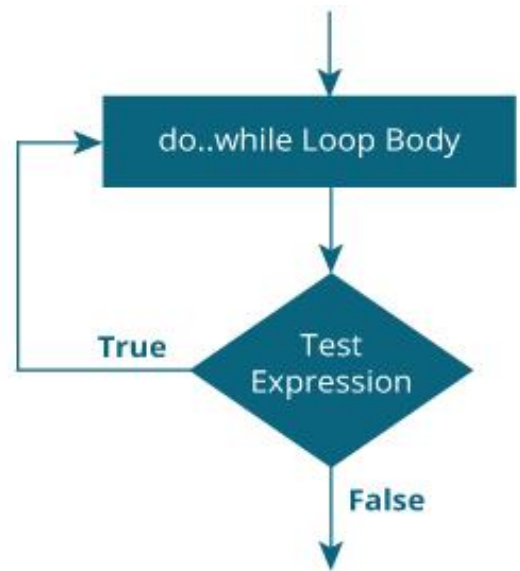

```
#include<stdio.h>
#include<conio.h>
main()
{
    int f1=0,f2=1,f3,i=3,len;
    printf("enter length of the fibonacci series:");
    scanf("%d",&len);
    printf("%d\t%d",f1,f2); // It prints the starting two values
    while(i<=len)          // checks the condition
    {
        f3=f1+f2; // performs add operation on previous two values
        printf("\t%d",f3); // It prints from third value to given length
        f1=f2;
        f2=f3;
        i=i+1; // incrementing the i value by 1
    }
    getch();
}
```

C program is used to display the multiplication table of a given number

```
#include <stdio.h>
void main()
{
    int num, i = 1;
    printf(" Enter any Number:");
    scanf("%d", &num);
    printf("Multiplication table of %d: ", num);
    while (i <= 10)
    {
        printf("%d * %d = %d", num, i, num * i);
        i++;
    }
    getch();
}
```

do...while

```
do  
{  
    // statements inside the body of the loop  
}  
while (testExpression);
```



Do—while Example Program: Program to print numbers

```
#include <stdio.h>
void main ()
{
    int a = 10;
    /* do loop execution */
    do
    {
        printf("value of a: %d\n", a);
        a = a + 1;
    }
    while( a < 20 );
    getch();
}
```

// Program to add numbers until the user enters zero

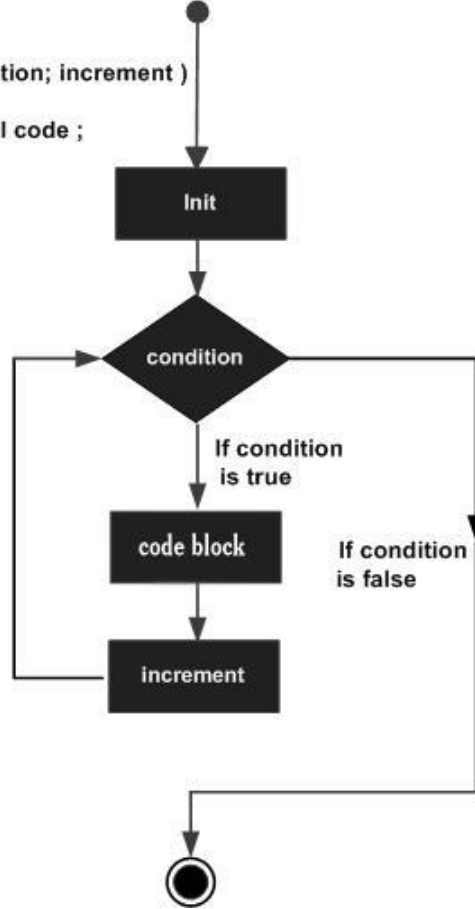
```
#include <stdio.h>
void main()
{
    int number, sum = 0;
    // the body of the loop is executed at least once
    do
    {
        printf("Enter a number: ");
        scanf("%d", &number);
        sum += number;
    }
    while(number != 0);
    printf("Sum = %d", sum);
    getch();
}
```

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

```
for( init; condition; increment )  
{  
    conditional code ;  
}
```



C program to print numbers from 1 to 10

```
#include <stdio.h>
void main()
{
    int i;
    for (i = 1; i < 11; ++i)
    {
        printf("%d ", i);
    }
    getch();
}
```


Program to print numbers

```
#include <stdio.h>
void main ()
{
    int a;
    /* for loop execution */
    for( a = 10; a < 20; a = a + 1 )
    {
        printf("value of a: %d\n", a);
    }
    getch();
}
```

Sum of Natural Numbers Using for Loop

```
#include <stdio.h>
void main()
{
    int n, i, sum = 0;
    printf("Enter a positive integer: ");
    scanf("%d", &n);
    for (i = 1; i <= n; ++i)
    {
        sum += i;
    }
    printf("Sum = %d", sum);
    return 0;
}
```

Unconditional Jump Statements

Jump statements interrupt the sequential execution of statements, so that execution continues at a different point in the program.

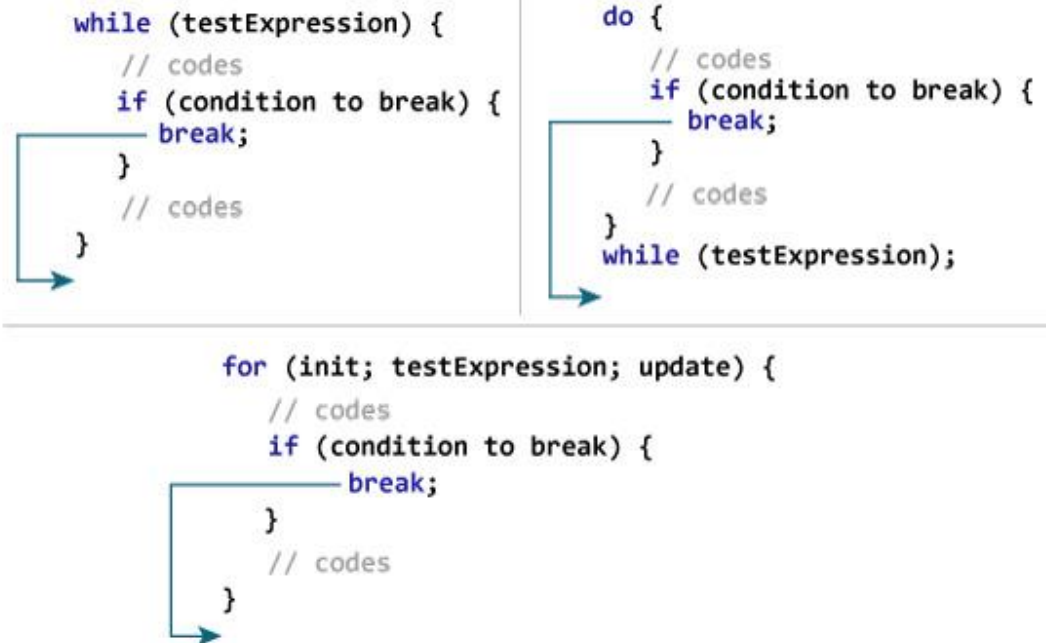
- **The break statement**
- **The continue statement**
- **The goto Statement**
- **the return statement**

Break Statement

- The break statement terminates execution of the immediately enclosing while , do, for, or switch statement.

syntax:

break;




```
# include <stdio.h>
void main()
{
    int i, number, sum = 0;
    for(i=1; i <= 10; ++i)
    {
        printf("Enter a n%d: ",i);
        scanf("%d", &number);
        // If the user enters a negative number, the loop ends
        if(number < 0)
        {
            break;
        }
        sum += number; // sum = sum + number;
    }
    printf("Sum = %d",sum);
    getch();
}
```

The Continue Statement


The continue statement skips the current iteration of the loop and continues with the next iteration.

Syntax:

continue;



```
while (testExpression) {  
    // codes  
    if (testExpression) {  
        continue;  
    }  
    // codes  
}
```



```
do {  
    // codes  
    if (testExpression) {  
        continue;  
    }  
    // codes  
} while (testExpression);
```



```
for (init; testExpression; update) {  
    // codes  
    if (testExpression) {  
        continue;  
    }  
    // codes  
}
```

Continue Statement

```
// C program to explain the use of continue statement
#include <stdio.h>
void main()
{
    int i=0;
    clrscr();
    while(i<=10)
    {
        if(i == 6)
            continue;
        printf("%d ", i);
        i++;
    }
    getch();
}
```

Multiplication table using Continue Statement

```
#include <stdio.h>
void main()
{
    int num, i = 1;
    printf(" Enter any Number:");
    scanf("%d", &num);
    printf("Multiplication table of %d: ", num);
    while (i <= 10)
    {
        i++;
        if(i==5)
            continue;
        printf("%d * %d = %d", num, i, num * i);
    }
    getch();
}
```


The return statement

- The return statement terminates execution of a function and returns control to the calling function, with or without a return value. A function may contain any number of return statements. The return statement has the following syntax:

Syntax:

```
return expression(opt);
```

Goto Statement

- The goto statement unconditionally transfers program control to a labeled statement

Use of goto statement

```
#include
#include
void main()
{
    int n;
    clrscr();
    printf("Enter a number:=");
    scanf("%d",&n);
    if(n%2==0)
        goto even;
    else
        goto odd;

even:
    printf("\nThe number is EVEN");
    exit(0);

odd:
    printf("The number is ODD");
    getch();
}
```

Sum of 1 to n Natural Numbers using goto

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i=1,n,sum=0;
    printf("Enter the value of n");
    scanf("%d",&n);
    x: if(i<=n)
    {
        sum=sum+i;
        i++;
        goto x;
    }
    printf("sum is %d",sum);
    getch();
}
```

Nested Loops in C

- **Nested loop** means a [loop statement](#) inside another loop statement. That is why nested loops are also called as “**loop inside loop**”.

Syntax: Nested for loop

```
for ( initialization; condition; increment )
{
    for ( initialization; condition; increment )
    {
        // statement of inside loop
    }
    // statement of outer loop
}
```

Nested While Loop:

```
while(condition)
{
    while(condition)
    {
        // statement of inside loop
    }
    // statement of outer loop
}
```

Nested do-While Loop:

```
do
{
    do{
        // statement of inside loop
    }while(condition);
    // statement of outer loop
}while(condition);
```

here is no rule that a loop must be nested inside its own type. In fact, there can be any type of loop nested inside any type and to any level.

```
do
{
  while(condition)
  {
    for ( initialization; condition; increment )
    {
      // statement of inside for loop
    }
    // statement of inside while loop
  }
  // statement of outer do-while loop
}while(condition);
```

// C program that uses nested for loop to print a square pattern

```
#include <stdio.h>
void main()
{
    int i, j, N;
    printf("Enter number of rows: ");
    scanf("%d", &N);
    /* Iterate through N rows */
    for(i=1; i<=N; i++)
    {
        /* Iterate over columns */
        for(j=1; j<=N; j++)
        {
            printf("*");
        }
        /* Move to the next line/row */
        printf("\n");
    }
    getch();
}
```



```
#include <stdio.h>
void main()
{
    for (int i=0; i<2; i++)
    {
        for (int j=0; j<4; j++)
        {
            printf("%d, %d\n",i ,j);
        }
    }
    getch();
}
```

write a C program to print multiplication table from 1 to 5.

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

Output Displays:

For i=1 it prints the product of i and j

1 2 3 4 5

Similarly for i=2 it prints

2 4 6 8 10

Recursion

Recursion

- Recursion is the process of repeating items in a self-similar way. In programming languages, if a program allows you to call a function inside the same function, then it is called a recursive call of the function.

```
void recursion()  
{  
    recursion(); /* function calls itself */  
}  
  
int main()  
{  
    recursion();  
}
```

Generate the Fibonacci series for a given number using a recursive function

```
#include <stdio.h>

int fibonacci(int i)
{
    if(i == 0)
    {
        return 0;
    }
    if(i == 1)
    {
        return 1;
    }
    return fibonacci(i-1) + fibonacci(i-2);
}
```

```
int main()
{
    int i;
    for (i = 0; i < 10; i++) {
        printf("%d\t", fibonacci(i));
    }
    return 0;
}
```

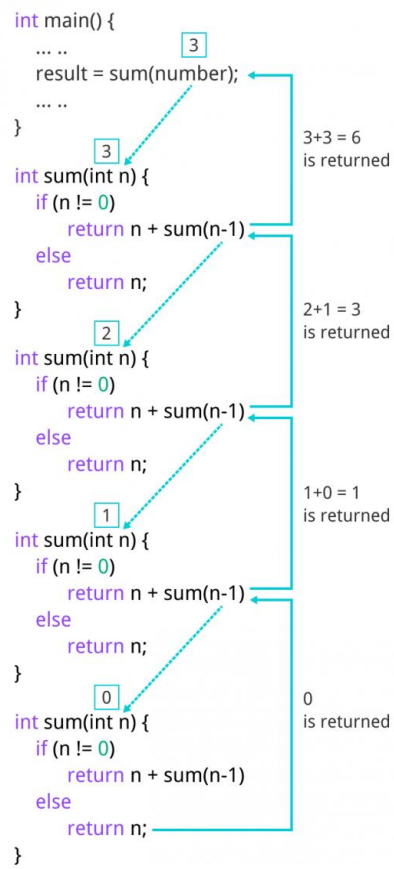
Sum of Natural Numbers Using Recursion

```
#include <stdio.h>

int sum(int n);

void main()
{
    int number, result;
    printf("Enter a positive integer: ");
    scanf("%d", &number);
    result = sum(number);
    printf("sum = %d", result);
    getch();
}

int sum(int n) {
    if (n != 0)
        // sum() function calls itself
        return n + sum(n-1);
    else
        return n;
}
```



The Calculator Program

/*C program to design calculator with basic operations using switch.*/

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

```
int main()
```

```
{
```

```
    int num1,num2;
```

```
    float result;
```

```
    char ch;  //to store operator choice
```

```
    printf("Enter first number: ");
```

```
    scanf("%d",&num1);
```

```
    printf("Enter second number: ");
```

```
    scanf("%d",&num2);
```

```
    printf("Choose operation to perform (+,-,*,/,%): ");
```

```
    scanf(" %c",&ch);
```

```
    result=0;
```

```
switch(ch)
```

```
{
```

```
    case '+':
```

```
        result=num1+num2;
```

```
        break;
```

```
    case '-':
```

```
        result=num1-num2;
```

```
        break;
```

```
    case '*':
```

```
        result=num1*num2;
```

```
        break;
```

```
    case '/':
```

```
        result=(float)num1/(float)num2;
```

```
        break;
```

```
    case '%':
```

```
        result=num1%num2;
```

```
        break;
```

```
    default:
```

```
        printf("Invalid operation.\n");
```

```
}
```

```
printf("Result: %d %c %d = %f\n",num1,ch,num2,result);
```

```
return 0;
```

```
}
```


Arrays

Arrays in c

An array is defined as an ordered set of similar data items. All the data items of an array are stored in consecutive memory locations in RAM. The elements of an array are of same data type and each item can be accessed using the same name.

- **Syntax:**

```
datatype arrayname[arraysize];
```

Example:

```
int data[100];
```

```
float mark[5];
```

It's important to note that the size and type of an array cannot be changed once it is declared.

Access Array Elements

- Suppose you declared an array mark as above. The first element is mark[0], the second element is mark[1] and so on.

| mark[0] | mark[1] | mark[2] | mark[3] | mark[4] |
|---------|---------|---------|---------|---------|
| | | | | |

How to initialize an array?

- It is possible to initialize an array during declaration. For example,

Ex:

```
int mark[5] = {19, 10, 8, 17, 9};
```

(or)

```
int mark[] = {19, 10, 8, 17, 9};
```

Here, we haven't specified the size. However, the compiler knows its size is 5 as we are initializing it with 5 elements.

```
// print the first element of the array
```

```
printf("%d", mark[0]);
```

```
// print the third element of the array
```

```
printf("%d", mark[2]);
```

```
// print ith element of the array
```

```
printf("%d", mark[i-1]);
```

Example 1: Array Input/Output

// Program to take 5 values from the user and store them in an array

// Print the elements stored in the array

```
#include <stdio.h>
void main()
{
    int values[5],i;
    printf("Enter 5 integers: ");
    // taking input and storing it in an
    array
    for(i = 0; i < 5; ++i)
    {
        scanf("%d", &values[i]);
    }

    printf("Displaying integers: ");
    // printing elements of an array
    for(i = 0; i < 5; ++i)
    {
        printf("%d\n", values[i]);
    }
    return 0;
}
```

Two dimensional (2D) arrays in C programming with example

- An array of arrays is known as 2D array. The two dimensional (2D) array in [C programming](#) is also known as matrix. A matrix can be represented as a table of rows and columns.

```
#include<stdio.h>

void main()
{
    int a[2][3], i, j;
    for(i=0; i<2; i++)
    {
        for(j=0;j<3;j++)
        {
            printf("Enter value for a[%d][%d]:", i, j);
            scanf("%d", &a[i][j]);
        }
    }
}
```

```
//Displaying array elements
printf("Two Dimensional array elements:\n");
for(i=0; i<2; i++)
{
    for(j=0;j<3;j++)
    {
        printf("%d ", a[i][j]);
    }
}
getch();
}
```


Multidimensional Arrays

- Here, you will learn to work with multidimensional arrays (two-dimensional and three-dimensional arrays) with the help of examples.
- In C programming, you can create an array of arrays. These arrays are known as multidimensional arrays.
- For example:

```
float x[3][4];
```

Here, x is a two-dimensional (2d) array. The array can hold 12 elements. You can think the array as a table with 3 rows and each row has 4 columns

| | Column 1 | Column 2 | Column 3 | Column 4 |
|-------|----------|----------|----------|----------|
| Row 1 | x[0][0] | x[0][1] | x[0][2] | x[0][3] |
| Row 2 | x[1][0] | x[1][1] | x[1][2] | x[1][3] |
| Row 3 | x[2][0] | x[2][1] | x[2][2] | x[2][3] |

- Similarly, you can declare a three-dimensional (3d) array.
- For example 1:

```
float y[2][4][3];
```

Here, the array **y** can hold **24** elements.

Example 2:

```
int Employees[2][4][3] = { { {10, 20, 30}, {15, 25, 35}, {22, 44, 66}, {33, 55, 77} }, { {1, 2, 3}, {5, 6, 7}, {2, 4, 6}, {3, 5, 7} } };
```

Here, We have 2 tables and the 1st table holds 4 Rows * 3 Columns, and the 2nd table also holds 4 Rows * 3 Columns

For a large number of rows and columns, we can access them using For Loop. Say, for example, to access array Employees[10][25][60]

```
int tables, rows, columns;
for (tables = 0; tables < 10; tables++)
{
    for (rows = 0; rows < 25; rows++)
    {
        for (columns = 0; columns < 60; columns++)
        {
            Printf("%d", Employees[tables][rows][columns]);
        }
    }
}
```

Multi Dimensional Array in C Example

// C Program to store and print 12 values entered by the user

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

```
void main()
```

```
{
```

```
    int i,j,k,threed[2][3][2];
```

```
    clrscr();
```

```
    printf("Enter 12 values: \n");
```

```
    for(i = 0; i < 2; i++)
```

```
    {
```

```
        for(j = 0; j < 3; j++)
```

```
        {
```

```
            for(k = 0; k < 2; k++)
```

```
            {
```

```
                scanf("%d", &threed[i][j][k]);
```

```
            }
```

```
        }
```

```
    }
```

// Printing values with proper index.

```
printf("\nDisplaying values:\n");
```

```
for(i = 0; i < 2; i++)
```

```
{
```

```
printf("\n");
```

```
    for(j = 0; j < 3; j++)
```

```
    {
```

```
        printf("\n");
```

```
        for(k = 0; k < 2; k++)
```

```
        {
```

```
            printf("%d\t",threed[i][j][k]);
```

```
        }
```

```
    }
```

```
}
```

```
getch();
```

```
}
```

Output:

Enter 12 values:

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

Displaying values:

| | |
|---|---|
| 1 | 2 |
| 3 | 4 |
| 5 | 6 |
| 7 | 8 |
| 9 | 1 |
| 2 | 4 |

—

Activate Windows
Go to PC settings to
activate Windows.

Example Programs

Using while, for, do-while

Decimal to Binary

Decimal to Binary

| | | | | |
|---|--|----|-------|---|
| 2 | | 47 | | |
| 2 | | 23 | _____ | 1 |
| 2 | | 11 | _____ | 1 |
| 2 | | 5 | _____ | 1 |
| 2 | | 2 | _____ | 1 |
| 2 | | 1 | _____ | 0 |
| | | 0 | _____ | 1 |
| | | | | R e m a i n d e r |

$$(47)_{10} = (101111)_2$$

C program to convert decimal to binary using while loop

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

```
void main()
```

```
{
```

```
    long long decimal, tempDecimal, binary;
```

```
    int rem, place = 1;
```

```
    binary = 0;
```

```
    /* Input decimal number from user */
```

```
    printf("Enter any decimal number: ");
```

```
    scanf("%lld", &decimal);
```

```
    tempDecimal = decimal;
```

```
    /* Decimal to binary conversion */
```

```
    while(tempDecimal > 0)
```

```
    {
```

```
        rem = tempDecimal % 2;
```

```
        binary = (rem * place) + binary;
```

```
        tempDecimal /= 2;
```

```
        place *= 10;
```

```
    }
```

```
    printf("Decimal number = %lld\n", decimal);
```

```
    printf("Binary number = %lld", binary);
```

```
    getch();
```

```
}
```


factorial of a Number

- The *factorial of a Number* n , denoted by $n!$, is the product of all positive integers less than or equal to n

For example,

$$6! = 6 * 5 * 4 * 3 * 2 * 1 = 720$$

C Program to find factorial of a given number using while

```
include<stdio.h>
#include<conio.h>
void main()
{
    int n,i,f;
    f=i=1;
    clrscr();
    printf("Enter a Number to Find Factorial: ");
    scanf("%d",&n);

    while(i<=n)
    {
        f=f*i;
        i++;
    }
    printf("The Factorial of %d is : %d",n,f);
    getch();
}
```

Write a c program to check whether a number is strong or not

- **Definition of strong number:**

- A number is called strong number if sum of the factorial of its digit is equal to number itself. For example: 145 since
- $1! + 4! + 5! = 1 + 24 + 120 = 145$

Program

```
#include<stdio.h>
void main()
{
    int num,i,f,r,sum=0,temp;
    printf("Enter a number: ");
    scanf("%d",&num);
    temp=num;
    while(num>0)
    {
        i=1;
        f=1;
        r=num%10;
```

```
while(i<=r)
{
    f=f*i;
    i++;
}
sum=sum+f;
num=num/10;
}
if(sum==temp)
    printf("%d is a strong number",temp);
else
    printf("%d is not a strong number",temp);

getch();
}
```

Armstrong

- A number is an Armstrong number when the **sum of nth power of each digit is equal to the number itself**. Here n is the number of digits in the given number.

For example

1634 (here n = 4)

$$= 1^4 + 6^4 + 3^4 + 4^4$$

$$= 1 + 1296 + 81 + 256$$

$$= 1634$$

AmstrongNumber

123 (here n = 3)

$$= 1^3 + 2^3 + 3^3$$

$$= 1 + 8 + 27$$

$$= 36$$

Not Amstrong Number

//amstrong number

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

```
#include<conio.h>
```

```
#include<math.h>
```

```
void main()
```

```
{
```

```
clrscr();
```

```
int n,t,n1,count=0,r,sum=0;
```

```
clrscr();
```

```
printf("enter n value");
```

```
scanf("%d",&n);
```

```
n1=n;
```

```
t=n;
```

```
while(n>0)
```

```
{
```

```
count=count+1;
```

```
n=n/10;
```

```
}
```

```
while(n1>0)
```

```
{
```

```
r=n1%10;
```

```
sum=sum+pow(r,count);
```

```
n1=n1/10;
```

```
}
```

```
if(sum==t)
```

```
{
```

```
printf("\n amstrong");
```

```
}
```

```
else
```

```
{
```

```
printf("not amstrong");
```

```
}
```

```
getch();
```

```
}
```

C Program to find a Prime Number

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

```
void main()
```

```
{
```

```
    int n, i, c = 0;
```

```
    printf("Enter any number n:");
```

```
    scanf("%d", &n);
```

```
    for (i = 1; i <= n; i++)
```

```
    {
```

```
        if (n % i == 0)
```

```
        {
```

```
            c++;
```

```
        }
```

```
    }
```

```
    if (c == 2)
```

```
    {
```

```
        printf("n is a Prime number");
```

```
    }
```

```
    else
```

```
{
```

```
    printf("n is not a Prime number");
```

```
    }
```

```
    getch();
```

```
}
```

Write a C Program to Find Maximum and Minimum Element in an Array

```
#include <stdio.h>
#define MAX_SIZE 100 // Maximum
array size
void main()
{
    int arr[MAX_SIZE];
    int i, max, min, size;
    /* Input size of the array */
    printf("Enter size of the array: ");
    scanf("%d", &size);
    /* Input array elements */
    printf("Enter elements in the array: ");
    for(i=0; i<size; i++)
    {
        scanf("%d", &arr[i]);
    }
```

```
/* Assume first element as maximum and minimum */
max = arr[0];
min = arr[0];
for(i=1; i<size; i++)
{
    /* If current element is greater than max */
    if(arr[i] > max)
    {
        max = arr[i];
    }
    /* If current element is smaller than min */
    if(arr[i] < min)
    {
        min = arr[i];
    }
}
/* Print maximum and minimum element */
printf("Maximum element = %d\n", max);
printf("Minimum element = %d", min);
getch();
}
```


Generate Fibonacci series upto n number using dowhile

```
#include<stdio.h>
void main()
{
    int i,n,f,f1,f2;
    printf("Enter Number of Fibonacci Values Needed : ");
    scanf("%d",&n);
    f=0;
    f1=1;
    f2=1;
    do
    {
        i++;
        printf("%d\n",f);
        f1=f2;
        f2=f;
        f=f1+f2;
    }
    while(i<=n+1);
}
```

C program for Palindrome number using For Loop

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i,n,r,s=0;
    clrscr();
    printf("\n Enter The Number:");
    scanf("%d",&n);
    //LOOP TO FIND REVERSE OF A NUMBER
    for(i=n;i>0; )
    {
        r=i%10;
        s=s*10+r;
        i=i/10;
    }
```

```
if(s==n)
{
    printf("\n %d is a Palindrome Number",n);
}
else
{
    printf("\n %d is not a Palindrome
Number",n);
}
getch();
}
```

C program to find Perfect Number or Not using For Loop

```
#include<stdio.h>
#include<conio.h>

void main()
{
    int n, i, sum=0;
    clrscr();
    printf(" Enter a number: ");
    scanf("%d", &n);

    for(i=1;i<n;i++)
    {
        if(n%i==0)
        {
            sum=sum+i;
        }
    }
    /* if-else condition to print Perfect Number or
    Not */
    if(sum==n)
        printf("\n %d is a Perfect Number.",n);
    else
        printf("\n %d is Not a Perfect Number.",n);
    getch();
}
```

Print * in equilateral triangle pattern in C using for loop

```
#include <stdio.h>

void main()
{
    int n,i,j;
    n = 5; // number of rows.
    for(i = 1; i <= n; i++)
    {
        for(j = 1; j <= n-i; j++)
        {
            printf(" ");
        }
        for(j = 1; j <= i; j++)
            printf("* ");
        printf("\n");
        getch();
    }
}
```

prime numbers between 1 to n using for loop

```
#include<stdio.h>
int main()
{
    int num,i,count,n;
    printf("Enter max range: ");
    scanf("%d",&n);
    for(num = 1;num<=n;num++){
        count = 0;
        for(i=2;i<=num/2;i++){
            if(num%i==0){
                count++;
                break;
            }
        }
        if(count==0 && num!= 1)
            printf("%d ",num);
        getch();
    }
}
```