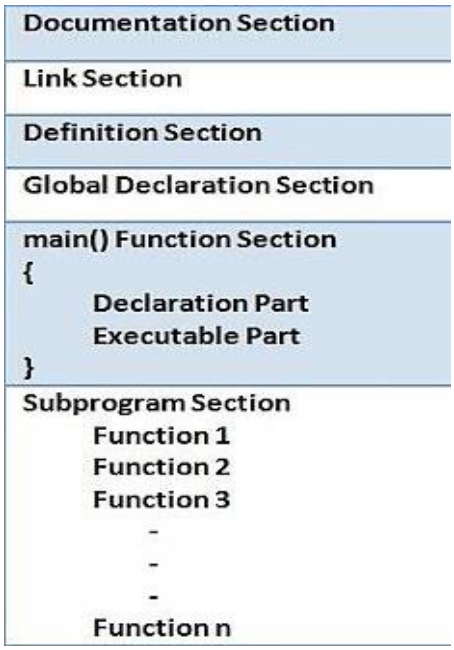
About C programming language:

C is a procedural programming language. It was initially developed by **Dennis Ritchie** in the year 1972. The main features of C language include low-level access to memory, a simple set of keywords, and clean style, these features make C language suitable for system programming like an operating system or compiler development.

Basic Structure of a C program:



Documentation Section: This section consists of comment lines which include the name of programmer, the author and other details like time and date of writing the program. Documentation section helps anyone to get an overview of the program.

Link Section: The link section consists of the header files of the functions that are used in the program. It provides instructions to the compiler to link functions from the system library. For example, stdio.h, conio.h etc.

Definition Section: All the symbolic constants (like macros), are written in definition section.

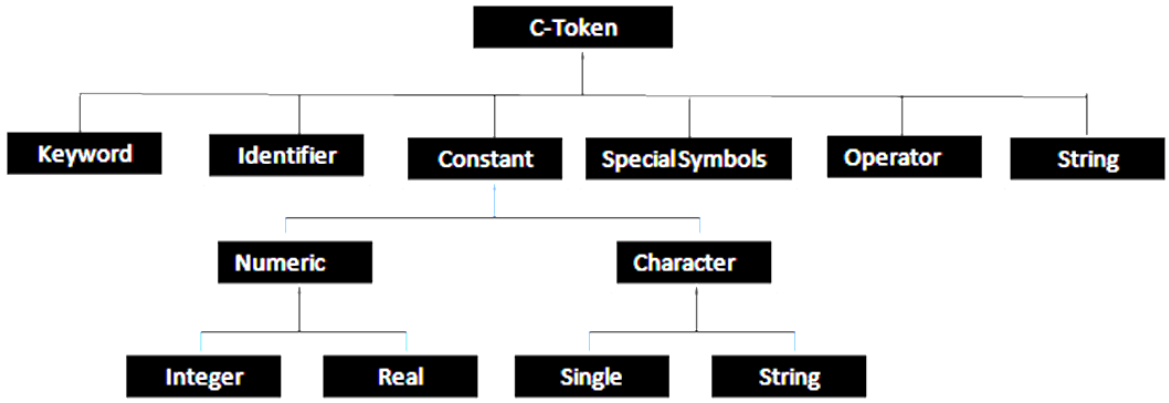
Global Declaration Section: The global variables that can be used anywhere in the program are declared in global declaration section.

main() Function Section: It is necessary have **one main()** function in every C program. It is the starting point of a C program and also known as heart of the program. This section contains two parts, declaration and executable part.
The declaration part declares all the variables that are used in executable part. Each statement in the declaration and executable part must end with a semicolon (;). The execution of program starts at opening braces and ends at closing braces.

Subprogram Section: The subprogram section contains all the user defined functions that are used to perform a specific task. These user defined functions are called in the main() function.

Tokens in C:

A token is a smallest unit in C-programing language. They are the building blocks of the program. Tokens are divided into six types as shown below.



Keyword: A keyword is a basic building block of any programming language. Their meanings are per-defined to the compiler and it cannot be change. C-language provides 32 keywords and all are written in lower case only. It cannot be used as a variable name.

Example: main, break, if, int, float, for, exit, etc.

Identifier: An identifier is a named memory area that is used to store value. It is user define name that are used to refer any kind of variable, array or function. Identifier can be defining using both lower case and upper case letters but normally lower case letters are used. Examples: name, salary, year, amount, etc.

Constant: A constant is a variable whose value does not change throughout the program. It can be further divided into the following categories.

- **Numeric Constant:** A numeric constant contain digits from 0 to 9.
  It has the following two types:
  - **Integer Constant:** An integer constant contains integer value; it does not contain fractional part. It may be negative or positive. Example: 100, -25, 50
    An integer constant may also represent in following three types.
    [a] Decimal [b] Octal [c] Hexadecimal
  - **Real Constant:** A real constant contains numerical value with fractional part. It may be positive or negative. Example: 10.5, -0.050, 25.075

- **Character Constant:** Here we mainly consider Single Character constant. It contains alphabets from a to z. Single character constant includes single character or digit enclosed by signal quote. Example: ‘a’, ‘10’, ‘-5’

String: String is a sequential series of characters enclosed within double quotes (“ ”).
Example: “Welcome to India”, “123”, “ABC”, “a”

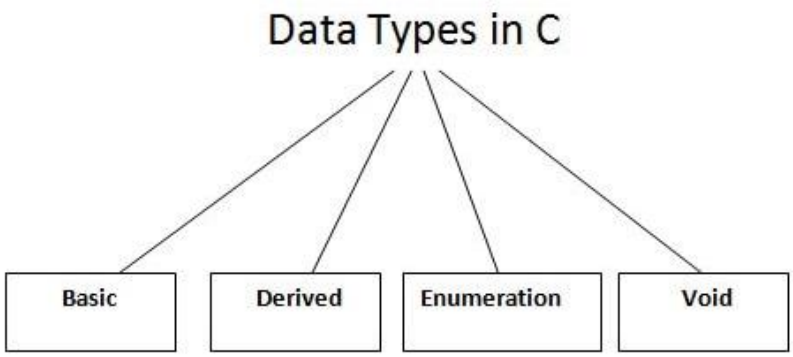
Special symbols: Some special symbols are used to specify the operation or to direct the compiler to perform the task. Special symbols includes &,!,(), [],{},*,^,<>

Operator: Operators are used to direct the compiler to perform some logical or arithmetic calculation. The following table shows different operators that can be used in C language.

Operator	Use	Symbol
Arithmetic	to perform arithmetic calculation	+,-,*,/
Logical	to perform logical operation	AND(&&), OR(||), NOT(!)
Relational	to perform relation comparison	>,<,=,=<,=>!
Modulo	to perform modulo operation	%
Increment/Decrement	to perform increment/decrement	++,--
Conditional	to evaluate expression	?:

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

Primitive data type: Primitive data types are predefined types of data, which are supported by the programming language. For example, integer, character, and string are all primitive data types.
Non-Primitive data type: Non-primitive data types are not defined by the programming language, but are created by the programmer itself. Eg: Array, Structure, Union etc.



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

char: The most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers.

int: As the name suggests, an int variable is used to store an integer.

float: It is used to store decimal numbers (numbers with floating point value) with single precision.

double: It is used to store decimal numbers (numbers with floating point value) with double precision.

What are Variables?

In programming, a variable is a container (storage area) to hold data. To indicate the storage area, each variable should be given a unique name (identifier). Variable names are just the symbolic representation of a memory location.

For example: int num = 24;

Here, **num** is a variable of **int** (integer) type and the variable is assigned an integer value 24.

Rules for naming a variable:

- A variable name can only have letters (both uppercase and lowercase letters), digits and underscore.
- The first letter of a variable should be either a letter or an underscore.
- There is no rule on how long a variable name (identifier) can be. However, it should be short for easy to use.

Writing our first C Program: Example 1

```
#include <stdio.h>
void main()
{
        // printf() displays the string inside quotation
        printf("Hello, World");
}
```

Output: Hello, World

Explanation:

- The **#include** is a pre-processor command that tells the compiler to include the contents of stdio.h (standard input and output header file) file in the program.
- The **stdio.h** file contains functions such as **scanf()** and **printf()** to take input and display output respectively.
- If you use the **printf()** function without writing **#include<stdio.h>**, the program will not compile.
- The execution of a C program starts from the **main()** function.
- **printf()** is a library function to send formatted output to the screen. In this program, **printf()** displays **Hello, World** text on the screen.

Example 2: C program to ADD two integers

```
#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);
}
```

Output: Enter two integers: 12 11
12 + 11 = 23

Explanation:

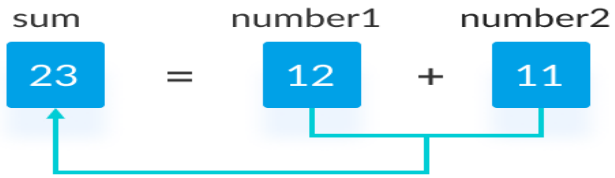
In this program, the user is asked to enter two integers. These two integers are stored in variables **number1** and **number2** respectively.

printf("Enter two integers: ");

scanf("%d %d", &number1, &number2);

Then, these two numbers are added using the **+** operator, and the result is stored in the **sum** variable.

sum = number1 + number2;



Finally, the **printf()** function is used to display the sum of numbers.

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

Conditional Operator in C (? :):

Conditional operators return one value if condition is true and returns another value is 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.

Example:

```
include <stdio.h>
void main()
{
        int x=1, y ;
        y = ( x ==1 ? 2 : 0 ) ;
        printf("x value is %d\n", x);
        printf("y value is %d", y);
}
```

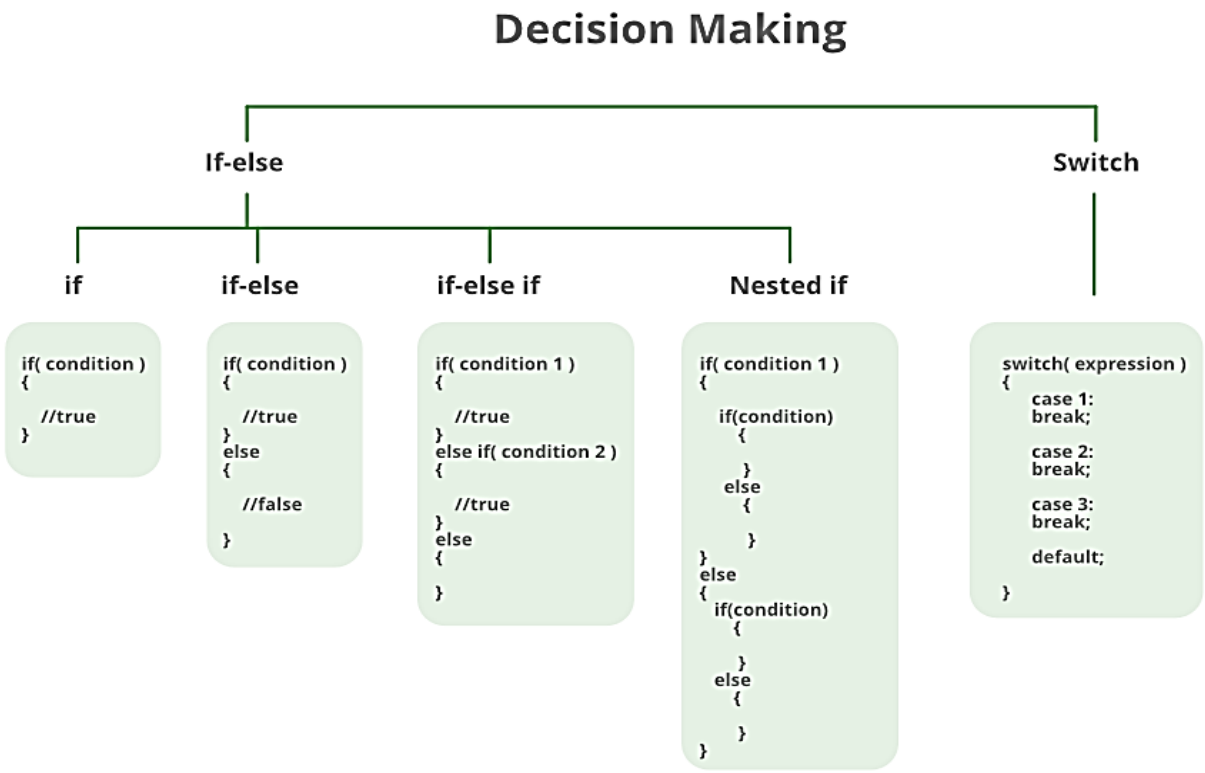
OUTPUT: x value is 1
y value is 2

DECISION CONTROL STATEMENTS:

The conditional statements help to jump from one part of a program to another depending if a particular condition is satisfied or not. That means we can control the flow of a program in such a way so that it executes certain statements based on the outcome of a condition (i.e. true or false).

There are various types of decision making control statements in C language. They are,

- if statement
- if else statement
- nested if statement
- switch case statement



if statement:

It is the most simple decision making statement and is used to decide whether a certain statement or block of statements will be executed or not, i.e if a certain condition is true then a block of statement is executed otherwise not.

Syntax:

```
if(condition)
{
        // Statements to execute if condition is true
}
```

If the condition is **true**, then the statements inside the block will be executed, otherwise not.

Example:

```
#include <stdio.h>
void main( )
{
        int x, y;
        x = 15;
        y = 13;
        if (x > y )
        {
                printf("x is greater than y");
        }
}
```

if-else statement:

Here if the expression is true, the first block is executed and if false then the second block is executed.

Syntax:

```
if (condition)
{
        // executes this block if condition is true
}
else
{
        // executes this block if condition is false
}
```

Example:

```
#include <stdio.h>
void main( )
{
        int x, y;
        x = 15;
        y = 13;
        if (x > y )
        {
                printf("x is greater than y");
        }
        else
        {
                printf("y is greater than x");
        }
}
```

Nested if statement:

Nested if statements means one or more **if** statement inside another **if** statement. The following example is shown to explain nested if.

//C program to find largest of three numbers

```
#include <stdio.h>
void main( )
{
        int a, b, c;
        printf("Enter 3 numbers...");
        scanf("%d%d%d",&a, &b, &c);
        if(a > b)
        {
                if(a > c)
                {
                        printf("a is the greatest");
                }
                else
                {
                        printf("c is the greatest");
                }
        }
        else
        {
                if(b > c)
                {
                        printf("b is the greatest");
                }
                else
                {
                        printf("c is the greatest");
                }
        }
}
```


switch-case statement:

switch statement tests the value of a variable and compares it with multiple cases. Once the **case** match is found, a block of statements associated with that particular **case** is executed.

Syntax:

```
switch (n)
{
        case 1:     // code to be executed if n = 1;
        break;
        case 2:     // code to be executed if n = 2;
        break;
        --------
        --------
        default:    // code to be executed if n doesn't match any cases
}
```

Example: Program to print Days according to input numbers

```
#include <stdio.h>
void main()
{
        int num;
        printf("\n Insert a Number between 1 to 7");
        scanf("%d", &num);
        switch (num)
        {
                case 1:         printf("MONDAY");
                                break;
                case 2:         printf("TUESDAY");
                                break;
                case 3:         printf("WEDNESDAY");
                                break;
                case 4:         printf("THURSDAY");
                                break;
                case 5:         printf("FRIDAY");
                                break;
                case 6:         printf("SATURDAY");
                                break;
                case 7:         printf("SUNDAY");
                                break;
                default:        printf("WRONG INPUT");
                                break;
        }
}
```

Loop Control Statements:

A **Loop** executes the sequence of statements many times until the given condition becomes false. A loop consists of two parts, a body of a loop and a control statement. The purpose of the loop is to repeat the same code a number of times.

In C, we have 3 types of Loops:

- **while Loop**

Syntax:

```
while (condition)
{
    //Statements;
}
```

Example:

```
#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()
}
```

- **do-while loop**

A do-while loop is similar to the **while** loop except that the condition is always executed after the body of a loop. It is also called an exit-controlled loop.

Syntax:

```
do
{
    //Statements
} while (condition);
```

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int num=1;      //initializing the variable
    do      //do-while loop
    {
        printf("%d\t",2*num);
        num++;                  //incrementing operation
```

```
        } while(num<=10);
        getch();
}
```

- **for loop**

Syntax:

```
for (initial value ; condition ; incrementation/decrementation)
{
        //Statements;
}
```

Example:

```
#include<stdio.h>
void main()
{
        int num;
        for(num=1; num<=10; num++)        //for loop to print 1-10 numbers
        {
                printf("%d\n", num);      //to print the number
        }
        getch();
}
```

Difference between while and do-while loop:

while loop	do-while loop

Syntax:
while (condition)
{
statements; //body of loop
}
	Syntax:
	do
	{
	statements; // body of loop.
	} while(condition);

Condition check appears at the start of the loop.	condition appears at the end of the loop
Do not executes if the initial condition is false	Executes at least once even if the condition is found false
Also called Entry-controlled loop	Also called Exit-controlled loop
No semicolon is used	Semicolon is used at the end of the loop

JUMP statements in C:

Jump statements are used to interrupt the normal flow of program. We have following jump statements in C:

- break
- continue
- goto
- return

break statement: The break statement is used inside loop or switch statement. When compiler finds the break statement inside a loop, compiler will abort the loop and continue to execute statements followed by loop.

Example:

```
for(i=0; i<10; i++)
{
    if(i==5)
    break;
}
```

here the control will leave the loop body when the value of 'i' becomes 5.

continue statement: The continue statement is also used inside loop. When compiler finds the **continue** statement inside a loop, it will skip all the following statements in the loop and start the next iteration.

Example:

```
for(i=0; i<10; i++)
{
        if(n==5)
        continue;
        printf("\n%d",n);
}
```

goto statement: The goto statement is a jump statement which jumps from one point to another point.

Example:

```
for(i=0; i<10; i++)
{
        if(i==5)
        goto label;
}
label
```

```
{
        printf("Now this line will be executed.");
}
```

return statement: The return statement terminates the execution of a function and returns control to the calling function. If it is used, it must be the last statement of a function. If no return statement appears in a function definition, control automatically returns to the calling function after the last statement of the called function is executed.

Syntax: **return** value;