What is C?

C is a general-purpose programming language created by Dennis Ritchie at the Bell Laboratories in 1972.

It is a very popular language, despite being old.

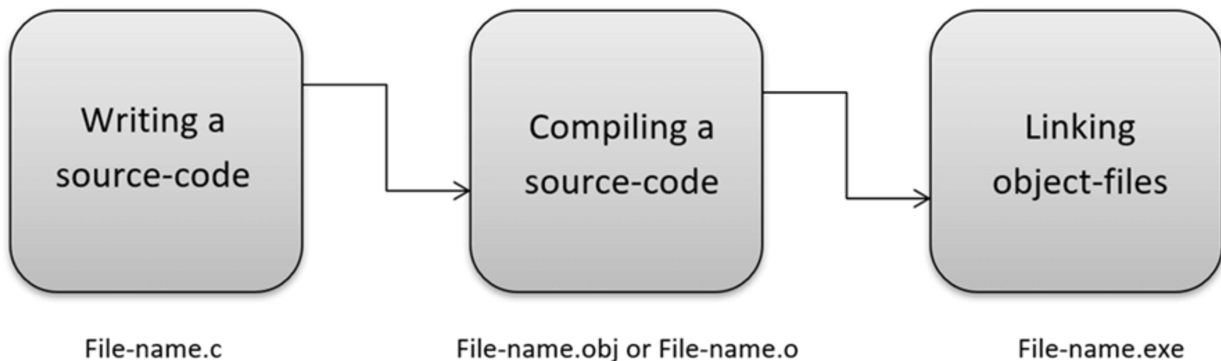
C is strongly associated with UNIX, as it was developed to write the UNIX operating system.

Why Learn C?

- It is one of the most popular programming language in the world
- If you know C, you will have no problem learning other popular programming languages such as Java, Python, C++, C#, etc, as the syntax is similar
- C is very fast, compared to other programming languages, like Java and Python
- C is very versatile; it can be used in both applications and technologies

How C Programming Language Works?

C is a compiled language. A compiler is a special tool that compiles the program and converts it into the object file which is machine readable. After the compilation process, the linker will combine different object files and creates a single executable file to run the program. The following diagram shows the execution of a 'C' program



Following is the list of popular compilers available online:

- Clang compiler
- MinGW compiler (Minimalist GNU for Windows)
- Portable 'C' compiler
- Turbo C

Environment setup :-

- **To install Dev C++ software, you need to follow the following steps.**
- **There are packages for different Operating Systems.**
- **Under package Dev-C++ 5.0 (4.9.9.2) with Mingw/GCC 3.4.2 compiler and GDB 5.2.1 debugger (9.0 MB) Click on the link "Download from SourceForge".**
- **This package will download C++ .exe file for Windows that can be used to install on Windows 7/8/XP/Vista/10.**
- **You will direct to SourceForge website, and your C++ download will start automatically.**

C Hello World! Example: Your First Program

```
#include<stdio.h> //Pre-processor
directive
```

```
void main()     //main function declaration
{
printf("Hello World");  //to output the string on a display


}
```

Output (Print Text)

```
#include <stdio.h>


int main() {
  printf("Hello World!");
  return 0;
}
```

New Lines

To insert a new line, you can use the `\n` character:

```
#include <stdio.h>



int main() {

  printf("Hello World!\n");

  printf("I am learning C.");

  return 0;

}
```

Pre-processor directive

#include is a pre-processor directive in ‘C.’
#include <stdio.h>, stdio is the library where the function **printf** is defined**.** printf is used for generating output. Before using this function, we have to first include the required file, also known as a header file (.h).

C Input Output (I/O)

In C programming, `printf()` is one of the main output function.

Integer Output

```
#include <stdio.h>
int main()
{
    int testInteger = 5;
    printf("Number = %d", testInteger);
    return 0;
}
```

float and double Output

Print Characters

```
#include <stdio.h>
int main()
{
    char chr = 'a';
    printf("character = %c", chr);
    return 0;
}
```

C Input

In C programming, `scanf()` is one of the commonly used function to take input from the user. The `scanf()` function reads formatted input from the standard input such as keyboards.

Integer Input/Output

```
}
```

Float and Double Input/Output

```
#include <stdio.h>

int main()
```

C Character I/O

```
#include <stdio.h>

int main()

{
```

```
    int a;

    float b;


    printf("Enter integer and then a float: ");


    // Taking multiple inputs

    scanf("%d%f", &a, &b);


    printf("You entered %d and %f", a, b);

    return 0;

}
```

- `%d` **for** `int`
- `%f` **for** `float`
- `%lf` **for** `double`
- `%c` **for** `char`

What Is Comment In C Language?

A **comment** is an explanation or description of the source code of the program. It helps a developer explain logic of the code and improves program readability. At run-time, a comment is ignored by the compiler.
There are two types of comments in C:
1) A comment that starts with a slash asterisk /* and finishes with an asterisk slash */ and you can place it anywhere in your code, on the same line or several lines.

2) Single-line Comments which uses a double slash // dedicated to comment single lines

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

int main() {

/* in main function

I can write my principal code

And this in several comments line */

int x = 42; //x is a integer variable

printf("%d", x);

return 0;}
```

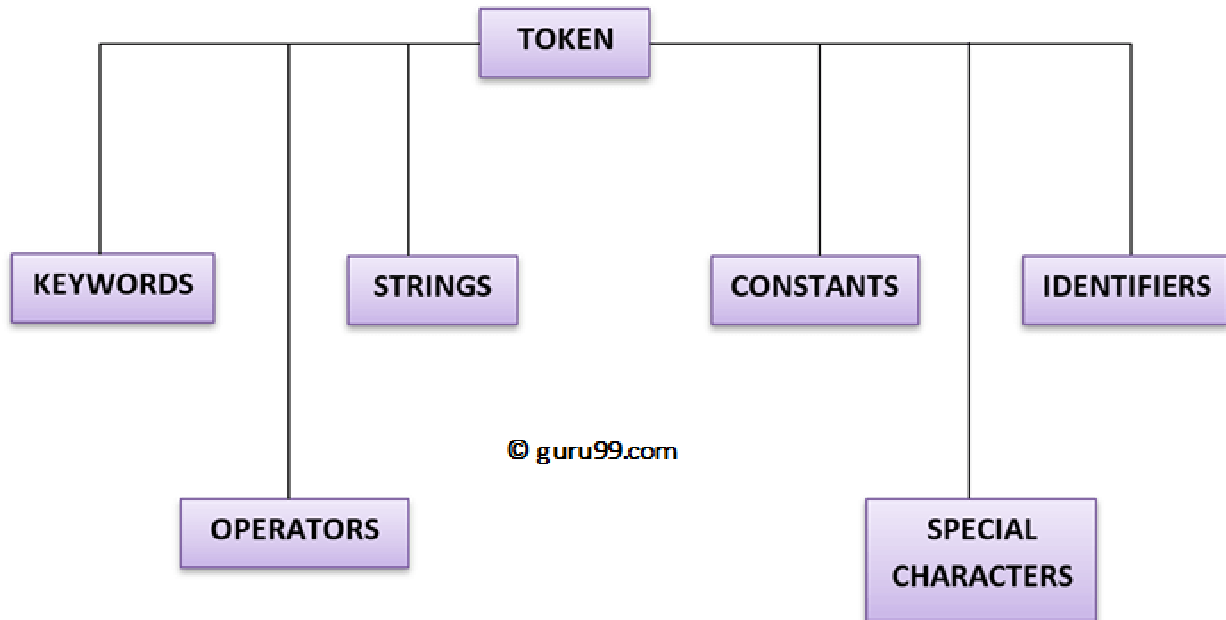
NEW LINE

To insert a new line, you can use the \n character:

Tip: Two\n \n characters after each other will create a blank line:

What is Token in C?

TOKEN is the smallest unit in a 'C' program. It is each and every word and punctuation that you come across in your C program. The compiler breaks a program into the smallest possible units (Tokens) and proceeds to the various stages of the compilation. C Token is divided into six different types, viz, Keywords, Operators, Strings, Constants, Special Characters, and Identifiers.



Keywords and Identifiers

In ‘C’ every word can be either a keyword or an identifier.
Keywords have fixed meanings, and the meaning cannot be changed. They act as a building block of a ‘C’ program. There are a total of 32 keywords in ‘C’. Keywords are written in lowercase letters.
Following table represents the keywords in ‘C’-

Keywords in C Programming Language			
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef

char	extern	return	union
const	short	float	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

An identifier is nothing but a name assigned to an element in a program. Example, name of a variable, function, etc.

What is a Variable?

A variable is an identifier which is used to store some value. Constants can never change at the time of execution. Variables can change during the execution of a program and update the value stored inside it.
Following are the rules that must be followed while creating a variable:

1. A variable name should consist of only characters, digits and an underscore.
2. A variable name should not begin with a number.
3. A variable name should not consist of whitespace.
4. A variable name should not consist of a keyword.
5. ‘C’ is a case sensitive language that means a variable named ‘age’ and ‘AGE’ are different.

```
E.g.

int my_variable = 48;
```

Format specifiers in C

The format specifier is used during input and output. It is a way to tell the compiler what type of data is in a variable during taking input using scanf() or printing using printf(). Some examples are %c, %d, %f, etc.

The format specifier in printf() and scanf() are mostly the same but there is some difference which we will see.

Character format specifier : %c

```
#include <stdio.h>

int main()

{

    char ch = 'A';

    printf("%c\n", ch);

    return 0;

}
```

For Signed Integer format specifier : %d, %i

```
#include <stdio.h>

int main()

{

    int x = 45, y = 90;

    printf("%d\n", x);
```

```
    printf("%i\n", x);

    return 0;

}
```

Floating-point format specifier : %f, %e or %E

```
#include <stdio.h>

int main()

{

    float a = 12.67;

    printf("%f\n", a);

    printf("%e\n", a);

    return 0;

}
```

String printing: %s

```
#include <stdio.h>

int main()

{

    char a[] = "geeksforgeeks";// string

    printf("%s\n", a);
```

```
    return 0;
}
```

Double floating-point number : %lf

```
#include <stdio.h>
int main()
{
    double a = 0.0;
    scanf("%lf", &a); // input is 45.65
    printf("%lf\n", a);
    return 0;
}
```

Data types

‘C’ provides various data types to make it easy for a programmer to select a suitable data type as per the requirements of an application. Following are the three data types:

1. Primitive data types
2. Derived data types
3. User-defined data types

There are five primary fundamental data types,

1. int for integer data

```
int age;
```

2. char for character data

```
Char letter;
```

3. float for floating point numbers

```
float division;
double BankBalance;
```

4. double for double precision floating point numbers

```
double BankBalance;
```

5. void

   A void data type doesn’t contain or return any value. It is mostly used for defining functions in ‘C’.

   ```
   void displayData()
   ```

Data type	Size in bytes	Range
Char or signed char	1	-128 to 127
Unsigned char	1	0 to 255
int or signed int	2	-32768 to 32767
Unsigned int	2	0 to 65535
Short int or Unsigned short int	2	0 to 255
Signed short int	2	-128 to 127
Long int or Signed long int	4	-2147483648 to 2147483647

Unsigned long int	4	0 to 4294967295
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

```
e.g.

int main() {

int x, y;

float salary = 13.48;

char letter = 'K';

x = 25;

y = 34;

int z = x+y;

printf("%d \n", z);

printf("%f \n", salary);

printf("%c \n", letter);

return 0;}
```

format specifiers in **printf** output function float (%f) and char (%c) and int (%d).

sizeof() operato:

We can use the sizeof() operator to check the size of a variable. See the following C program for the usage of the various data types:

```
// C Program to print size of

// different data type in C

#include <stdio.h>


int main()

{

      int size_of_int=sizeof(int);

     int size_of_char= sizeof(char);

      int size_of_float=sizeof(float);

    int size_of_double=sizeof(double);


    printf("The size of int data type :
%d\n",size_of_int );

    printf("The size of char data type :
%d\n",size_of_char);

    printf("The size of float data type :
%d\n",size_of_float);

    printf("The size of double data type :
%d",size_of_double);


  return 0;
```

```
}
```

Constants

Constants are the fixed values that never change during the execution of a program. Following are the various types of constants:

Integer constants

```
Example, 111, 1234
```

Character constants

```
Example, 'A', '9'
```

String constants

```
Example, "Hello", "Programming"
```

Real Constants

Like integer constants that always contains an integer value. ‘C’ also provides real constants that contain a decimal point or a fraction value. The real constants are also called as floating point constants. The real constant contains a decimal point and a fractional value.

```
Example, 202.15, 300.00
```

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

2.int main(){

3.const float PI=3.14;

4.PI=4.5;

5.printf(“Th value of PI is:%f”,PI);

6.return 0;
```

```
}
```

Literals in c:

A literal is a value that is expressed as itself. For example, the number 25 or the string "Hello World" are both literals.

- **Basically, constants are variables whose value cannot change.**
- **Literals are notations that represent fixed values. These values can be Strings numbers etc**
- **Literals can be assigned to variables**

```
var a = 10;

 var name = "Simba";

 const pi = 3.14;
```

Here a and name are variables. pi is a constant. (Constants are those variables whose value doesn't change.)
Here 10, "Simba" and 3.14 are literals.

- ## Storage Classes:

- A storage class defines the scope (visibility) and life-time of variables and/or functions within a C Program. They precede the type that they modify.

Every variable in C programming has two properties: type and storage class.

Type refers to the data type of a variable. And, storage class determines the scope, visibility and lifetime of a variable.

There are 4 types of storage class:

1. automatic
2. external
3. static
4. register

Local Variable

The variables declared inside a block are automatic or local variables. The local variables exist only inside the block in which it is declared.

```
int main() {

    int n1; // n1 is a local variable to main()

}



void func() {

   int n2;  // n2 is a local variable to func()

}
```

Global Variable

Variables that are declared outside of all functions are known as external or global variables. They are accessible from any function inside the program.

```
#include <stdio.h>
void display();

int n = 5;  // global variable

int main()
{
    ++n;
    display();
    return 0;
}

void display()
{
    ++n;
    printf("n = %d", n);
}
```

Static Variable

A static variable is declared by using the `static` keyword. For example;

```
static int i;

#include <stdio.h>
void display();


int main()
{
    display();
    display();
}
void display()
{
    static int c = 1;
    c += 5;
    printf("%d  ",c);
}
```

Register Variable

The `register` keyword is used to declare register variables. Register variables were supposed to be faster than local variables.

However, modern compilers are very good at code optimization, and there is a rare chance that using register variables will make your program faster.

Unless you are working on embedded systems where you know how to optimize code for the given application, there is no use of register variables.

Type Casting in C

Typecasting allows us to convert one data type into other. In C language, we use cast operator for typecasting which is denoted by (type).

(type)value;

Without Type Casting:

```
1. int f= 9/4;
2. printf("f : %d\n", f );//Output: 2
```

With Type Casting:

```
1. float f=(float) 9/4;
2. printf("f : %f\n", f );//Output: 2.250000
```

- ## C Programming Operators

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Bitwise operators
-

C Arithmetic Operators

An arithmetic operator performs mathematical operations such as addition, subtraction, multiplication, division etc on numerical values (constants and variables).

Operator	Meaning of Operator
+	addition or unary plus
-	subtraction or unary minus
*	multiplication
/	division
%	remainder after division (modulo division)

```
#include <stdio.h>

int main()

{

int a = 12, b = 6, c;

c = a + b;

printf("a+b = %d \n", c);

c = a - b;

printf("a-b = %d \n", c);

c = a *b;

printf("a*b = %d \n", c);

c = a / b;

printf("a/b = %d \n", c);
```

```
c = a % b;

printf("Remainder when a divided by b = %d \n", c);

return 0;

}
```

}C Increment and Decrement Operators

Increment ++ increases the value by 1 whereas decrement -- decreases the value by 1. These two operators are unary operators, meaning they only operate on a single operand.

```
// Working of increment and decrement operators

#include <stdio.h>

int main()

{

    int a = 10, b = 100;

    float c = 10.5, d = 100.5;



    printf("++a = %d \n", ++a);

    printf("--b = %d \n", --b);

    printf("++c = %f \n", ++c);
```

```
    printf("--d = %f \n", --d);

    return 0;
}
```

C Assignment Operators

An assignment operator is used for assigning a value to a variable. The most common assignment operator is `=`

Operator	Example	Same as
=	a = b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b

%=	a %= b	a = a%b

```c
// Working of assignment operators
#include <stdio.h>
int main()
{
    int a = 5, c;

    c = a;          // c is 5
    printf("c = %d\n", c);
    c += a;         // c is 10
    printf("c = %d\n", c);
    c -= a;         // c is 5
    printf("c = %d\n", c);
    c *= a;         // c is 25
    printf("c = %d\n", c);
    c /= a;         // c is 5
    printf("c = %d\n", c);
    c %= a;         // c = 0
    printf("c = %d\n", c);

    return 0;
}
```

C Relational Operators

A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.

Relational operators are used in decision making and loops.

Operator	Meaning of Operator	Example
==	Equal to	`5 == 3` is evaluated to 0
>	Greater than	`5 > 3` is evaluated to 1
<	Less than	`5 < 3` is evaluated to 0
!=	Not equal to	`5 != 3` is evaluated to 1
>=	Greater than or equal to	`5 >= 3` is evaluated to 1
<=	Less than or equal to	`5 <= 3` is evaluated to 0

```
// Working of relational operators
#include <stdio.h>
int main()
{
    int a = 5, b = 5, c = 10;

    printf("%d == %d is %d \n", a, b, a == b);
    printf("%d == %d is %d \n", a, c, a == c);
    printf("%d > %d is %d \n", a, b, a > b);
    printf("%d > %d is %d \n", a, c, a > c);
    printf("%d < %d is %d \n", a, b, a < b);
    printf("%d < %d is %d \n", a, c, a < c);
    printf("%d != %d is %d \n", a, b, a != b);
```

```
    printf("%d != %d is %d \n", a, c, a != c);
    printf("%d >= %d is %d \n", a, b, a >= b);
    printf("%d >= %d is %d \n", a, c, a >= c);
    printf("%d <= %d is %d \n", a, b, a <= b);
    printf("%d <= %d is %d \n", a, c, a <= c);

    return 0;
}
```

C Logical Operators

An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false. Logical operators are commonly used in decision making in C programming.

Operator	Meaning	Example
&&	Logical AND. True only if all operands are true	If c = 5 and d = 2 then, expression `((c==5) && (d>5))` equals to 0.
||	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression `((c==5) || (d>5))` equals to 1.
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression `!(c==5)` equals to 0.

```
// Working of logical operators

#include <stdio.h>
int main()
```

```
{
    int a = 5, b = 5, c = 10, result;

    result = (a == b) && (c > b);
    printf("(a == b) && (c > b) is %d \n", result);

    result = (a == b) && (c < b);
    printf("(a == b) && (c < b) is %d \n", result);

    result = (a == b) || (c < b);
    printf("(a == b) || (c < b) is %d \n", result);

    result = (a != b) || (c < b);
    printf("(a != b) || (c < b) is %d \n", result);

    result = !(a != b);
    printf("!(a != b) is %d \n", result);

    result = !(a == b);
    printf("!(a == b) is %d \n", result);

    return 0;
}
```

C Bitwise Operators

During computation, mathematical operations like: addition, subtraction, multiplication, division, etc are converted to bit-level which makes processing faster and saves power.

Bitwise operators are used in C programming to perform bit-level operations.

Operators	Meaning of operators

&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

Conditional Operator in C

The conditional operator is also known as a **ternary operator**. The conditional statements are the decision-making statements which depends upon the output of the expression. It is represented by two symbols, i.e., '?' and ':'.

Expression1? expression2: expression3;

```
1. #include <stdio.h>
2. int main()
3. {
4.     int age;  // variable declaration
5.     printf("Enter your age");
6.     scanf("%d",&age);   // taking user input for age variable
```

7. (age>=18)? (printf("eligible for voting")) : (printf("not eligible for voting")); // conditional operator

8. **return** 0;

9. }

Special operator:

Comma Operator

```
int a, c = 5, d;
```

Example 6: sizeof Operator

- **Conditional Branching control statements:**

1. if Statement:

```
if (condition)
{
    // code
}
```

- **How if statement works?**
- **If the test expression is evaluated to true, statements inside the body of `if` are executed.**
- **If the test expression is evaluated to false, statements inside the body of `if` are not executed.**

Expression is true.

```
int test = 5;

if (test < 10)
{
   // codes
}

// codes after if
```

Expression is false.

```
int test = 5;

if (test > 10)
{
   // codes
}

// codes after if
```

- Example 1: if statement

- // Program to display a number if it is negative

```
#include <stdio.h>
int main() {
    int number;

    printf("Enter an integer: ");
    scanf("%d", &number);

    // true if number is less than 0
    if (number < 0) {
        printf("You entered %d.\n", number);
    }

    printf("The if statement is easy.");

    return 0;
}
```

2. if...else Statement:

```
if (condition)
{
    // run code if test expression is true
}
else {
    // run code if test expression is false
}
```

How if...else statement works?

Expression is true.

```
int test = 5;

if (test < 10)
{
   // body of if
}
else
{
   // body of else
}
```

Expression is false.

```
int test = 5;

if (test > 10)
{
   // body of if
}
else
{
   // body of else
}
```

Example 2: if...else statement

```
// Check whether an integer is odd or even

#include <stdio.h>
int main() {
    int number;
    printf("Enter an integer: ");
    scanf("%d", &number);

    // True if the remainder is 0
    if  (number%2 == 0) {
        printf("%d is an even integer.",number);
    }
    else {
        printf("%d is an odd integer.",number);
    }

    return 0;
}
```

3. if...else Ladder:

The if...else ladder allows you to check between multiple test expressions and execute different statements.

```
if (test expression1) {
   // statement(s)
}
else if(test expression2) {
   // statement(s)
}
else if (test expression3) {
   // statement(s)
}
.
.
else {
   // statement(s)
}
```

Example : C if...else Ladder

```
// Program to relate two integers using =, > or < symbol

#include <stdio.h>
int main() {
    int number1, number2;
    printf("Enter two integers: ");
    scanf("%d %d", &number1, &number2);

    //checks if the two integers are equal.
    if(number1 == number2) {
        printf("Result: %d = %d",number1,number2);
    }

    //checks if number1 is greater than number2.
    else if (number1 > number2) {
        printf("Result: %d > %d", number1, number2);
    }
```

```
    //checks if both test expressions are false
    else {
        printf("Result: %d < %d",number1, number2);
    }

    return 0;
}
```

4. Nested if...else:

```
#include <stdio.h>
int main() {
    int number1, number2;
    printf("Enter two integers: ");
    scanf("%d %d", &number1, &number2);

    if (number1 >= number2) {
      if (number1 == number2) {
        printf("Result: %d = %d",number1,number2);
      }
      else {
        printf("Result: %d > %d", number1, number2);
      }
    }
    else {
        printf("Result: %d < %d",number1, number2);
    }

    return 0;
}
```

- **Switch-case:**

Instead of writing many `if..else` statements, you can use the `switch` statement.

The `switch` statement selects one of many code blocks to be executed:

- **Syntax of switch...case**

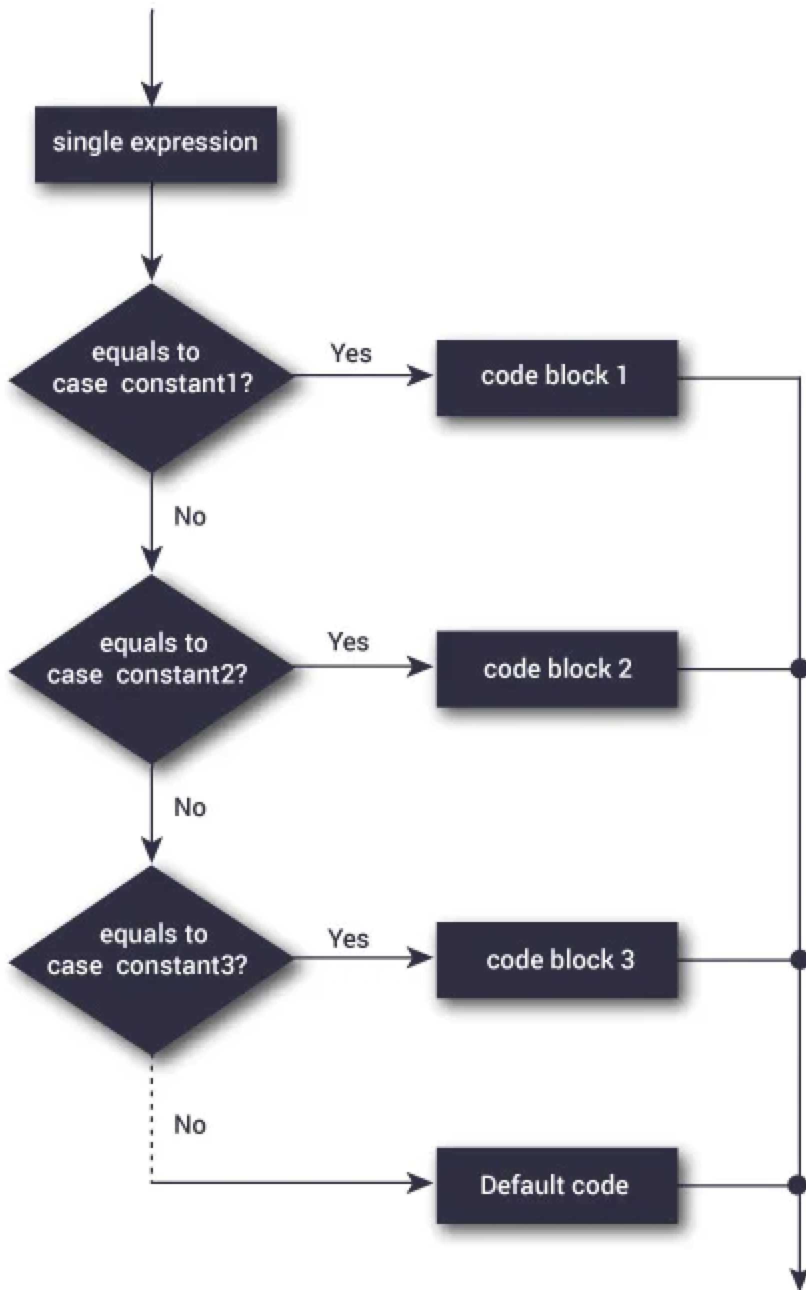
```
switch (expression)

{

    case constant1:

      // statements

      break;



    case constant2:

      // statements

      Break;

}
```

The `expression` is evaluated once and compared with the values of each `case` label.

- If there is a match, the corresponding statements after the matching label are executed. For example, if the value of the expression is equal to `constant2`, statements after `case constant2:` are executed until `break` is encountered.
- If there is no match, the default statements are executed.

Notes:

- If we do not use the `break` statement, all statements after the matching label are also executed.
- The `default` clause inside the `switch` statement is optional.




```
// Program to create a simple calculator
#include <stdio.h>

int main() {
```

```
r operation;
double n1, n2;cha

printf("Enter an operator (+, -, *, /): ");
scanf("%c", &operation);
printf("Enter two operands: ");
scanf("%lf %lf",&n1, &n2);

switch(operation)
{
    case k:
        printf("%.1lf + %.1lf = %.1lf",n1, n2, n1+n2);
        break;

    case '-':
        printf("%.1lf - %.1lf = %.1lf",n1, n2, n1-n2);
        break;

    case '*':
        printf("%.1lf * %.1lf = %.1lf",n1, n2, n1*n2);
        break;

    case '/':
        printf("%.1lf / %.1lf = %.1lf",n1, n2, n1/n2);
        break;

    // operator doesn't match any case constant +, -, *, /
    default:
        printf("Error! operator is not correct");
```

```
  }v

  return 0;
}
```

The break Keyword

When C reaches a `break` keyword, it breaks out of the switch block.

This will stop the execution of more code and case testing inside the block.

When a match is found, and the job is done, it's time for a break. There is no need for more testing.

- **Conditional looping control statements:**
- ## What is Loop in C?
- Looping Statements in C execute the sequence of statements many times until the stated condition becomes false.
- **While loop**
- **Do-while loop**
- **For while loop**
- **While loop:**

```
while (condition)
{
  Statements;
incriment/decriment oprator
}
e.g.#include<conio.h>
```

```
int main()
{
    int num=1;   //initializing the variable
    while(num<=10)   //while loop with condition
    {
        printf("%d\n",num);
        num++;       //incrementing operation
    }
    return 0;
}
```

Do-While :

Syntax:

```
do
{
  statements
} while (expression);
```

E.g.

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

```
        num++;      //incrementing operation
    }while(num<=10);
    return 0;
}
```

For loop :

Syntax

```
for (initial value; condition; incrementation or
decrementation )
{
  statements;
}
e.g.#include<stdio.h>
int main()
{
    int number;
    for(number=1;number<=10;number++)  //for loop
to print 1-10 numbers
    {
        printf("%d\n",number);     //to print the
number
    }
```

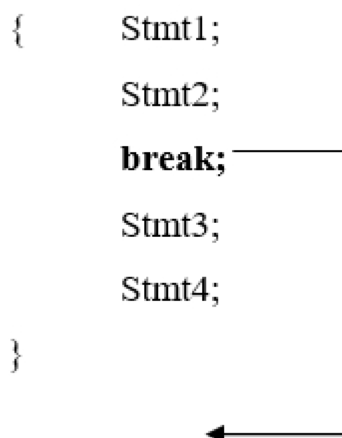
```
    return 0;
}
```

- **Unconditional control statements:**
    1. **Break**
    2. **Continue**
    3. **Goto**

***functions**

break

- It is a keyword which is used to terminate the loop (or) exit from the block.
- The control jumps to next statement after the loop (or) block.
- break is used with for, while, do-while and switch statement.
- When break is used in nested loops then, only the innermost loop is terminated.

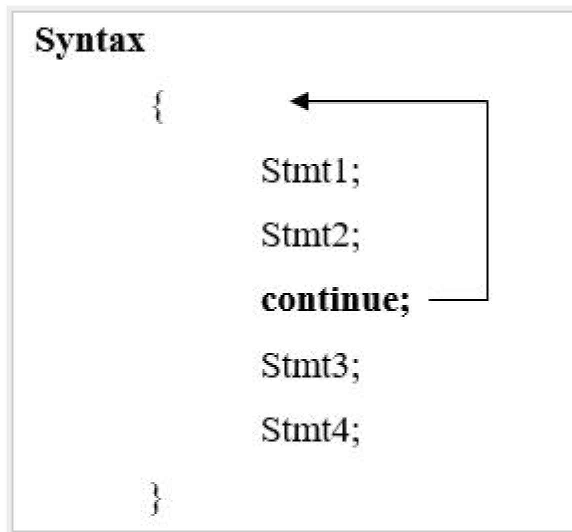


```
#include<stdio.h>
main( ){
   int i;
   for (i=1; i<=5; i++){
      printf ("%d", i);
      if (i==3)
      break;
   }
}
```

Continue

The `continue` statement skips the current iteration of the loop and continues with the next iteration. Its syntax is:

The syntax for the continue statement is as follows −



```
#include<stdio.h>
void main ()
{
   int i = 0;
   while(i!=10)
   {
      printf("%d", i);
      continue;
      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 statment can be used to repeat some part of the code for a particular condition

The syntax for the goto statement is as follows –

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

```
// Program to calculate the sum and average of positive numbers
// If the user enters a negative number, the sum and average are displayed.

#include <stdio.h>
```

```
int main() {

   const int maxInput = 100;
   int i;
   double number, average, sum = 0.0;

   for (i = 1; i <= maxInput; ++i) {
      printf("%d. Enter a number: ", i);
      scanf("%lf", &number);

      // go to jump if the user enters a negative number
      if (number < 0.0) {
         goto jump;
      }
      sum += number;
   }

jump:
   average = sum / (i - 1);
   printf("Sum = %.2f\n", sum);
   printf("Average = %.2f", average);

   return 0;
```

```
}o/p
1. Enter a number: 3
2. Enter a number: 4.3
3. Enter a number: 9.3
4. Enter a number: -2.9
Sum = 16.60
Average = 5.53
```

- # Functions:

- The function contains the set of programming statements enclosed by {}. A function can be called multiple times to provide reusability and modularity to the C program. In other words, we can say that the collection of functions creates a program. The function is also known as *procedure* or *subroutine* in other programming languages.
- **There are three aspects of a C function.:**
    1. **Function declaration**
    2. **Function definition**
    3. **Function call**

Syntax:

```
return_type function_name(data_type parameter...)
{
/code to be executed
}
```

• Types of Functions

- **Library Functions**: are the functions which are declared in the C header files such as scanf(), printf(), gets(), clrscr(), getchl(), floor() etc.
- **User-defined functions**: are the functions which are created by the C programmer, so that he/she can use it many times. It reduces the complexity of a big program and optimizes the code. As add(),sub(),multi(),div().
- A C function may or may not **return a value** from the function. If you don't have to return any value from the function, use void for the return type.

```
void hello()
{
printf("hello c");
}
```

- **If you want to return any value from the function, you need to use any data type such as int, long, char, etc. The return type depends on the value to be returned from the function.**

```
int get()
{
return 10;
}
```

Example for Function without argument

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

```
    printName();
}
void printName()
{
    printf("Javatpoint");
}
```

5

```
#include<stdio.h>
void sum(int, int);
void main()
{
    int a,b,result;
    printf("\nGoing to calculate the sum of two numbers:");
    printf("\nEnter two numbers:");
    scanf("%d %d",&a,&b);
    sum(a,b);
}
void sum(int a, int b)
{
    printf("\nThe sum is %d",a+b);
}
```

*

Recursion program:

```
#include<stdio.h>

int table(int no)

{

printf("%d",no);

No++;

table(no);

}

}

Int main()

{

table(1);

}
```

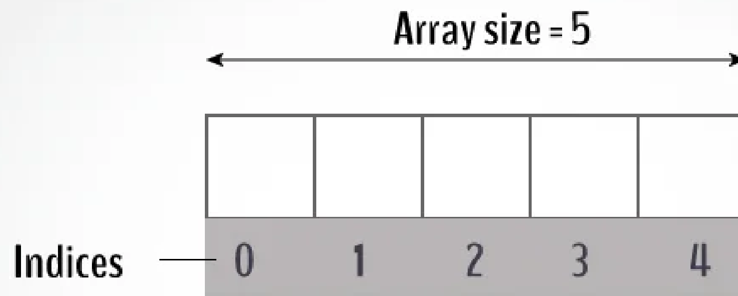
Pgm 1:program to calculate the area of the square

Pgm2

Pgm 3:Program to check whether a number is even or odd

- ## Arrays:

- An array is a variable that can store multiple values. For example, if you want to store 100 integers, you can create an array for it.



How to declare an array?

```
dataType arrayName[arraySize];
```

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

Access Array Elements

You can access elements of an array by indices.

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?

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

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

```
mark[0] is equal to 19

mark[1] is equal to 10

mark[2] is equal to 8

mark[3] is equal to 17

mark[4] is equal to 9
```

Change Value of Array elements

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

// make the value of the third element to -1

mark[2] = -1;

// make the value of the fifth element to 0

mark[4] = 0;
```

Input and Output Array Elements

```
// take input and store it in the 3rd element

scanf("%d", &mark[2]);

// take input and store it in the ith element

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

Here's how you can print an individual element of an array.

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

```c
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:

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

// Print the elements stored in the array

#include <stdio.h>

int main() {
```

```
  int values[5];

  printf("Enter 5 integers: ");

  // taking input and storing it in an array
  for(int i = 0; i < 5; ++i) {
      scanf("%d", &values[i]);
  }

  printf("Displaying integers: ");

  // printing elements of an array
  for(int i = 0; i < 5; ++i) {
```

```
        printf("%d\n", values[i]);
    }
    return 0;
}
```

Example 2: Calculate Average

```
// Program to find the average of n numbers using arrays

#include <stdio.h>

int main() {

  int marks[10], i, n, sum = 0;
  double average;

  printf("Enter number of elements: ");
```

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

for(i=0; i < n; ++i) {
  printf("Enter number%d: ",i+1);
  scanf("%d", &marks[i]);

  // adding integers entered by the user to the sum variable
  sum += marks[i];
}

// explicitly convert sum to double
// then calculate average
average = (double) sum / n;

printf("Average = %.2lf", average);
```

```
  return 0;

}
```

Run Code

Output

```
Enter number of elements: 5

Enter number1: 45

Enter number2: 35

Enter number3: 38

Enter number4: 31

Enter number5: 49

Average = 39.60
```

C Multidimensional Arrays

In C programming, you can create an array of arrays. These arrays are known as multidimensional arrays. For example,

	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]

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

Initializing a multidimensional array

// Different ways to initialize two-dimensional array

int c[2][3] = {{1, 3, 0}, {-1, 5, 9}};

```
int c[][3] = {{1, 3, 0}, {-1, 5, 9}};
```

```
int c[2][3] = {1, 3, 0, -1, 5, 9};
```

Initialization of a 3d array

```
int test[2][3][4] = {

   {{3, 4, 2, 3}, {0, -3, 9, 11}, {23, 12, 23, 2}},

   {{13, 4, 56, 3}, {5, 9, 3, 5}, {3, 1, 4, 9}}};
```

Example 2: Sum of two matrices

```
// Taking input using nested for loop

  printf("Enter elements of 1st matrix\n");

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

    for (int j = 0; j < 2; ++j)

    {
```

```
    printf("Enter a%d%d: ", i + 1, j + 1);

    scanf("%f", &a[i][j]);

  }



// Taking input using nested for loop

printf("Enter elements of 2nd matrix\n");

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

  for (int j = 0; j < 2; ++j)

  {

    printf("Enter b%d%d: ", i + 1, j + 1);

    scanf("%f", &b[i][j]);

  }
```

```
// adding corresponding elements of two arrays

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

  for (int j = 0; j < 2; ++j)

  {

    result[i][j] = a[i][j] + b[i][j];

  }



// Displaying the sum

printf("\nSum Of Matrix:");



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

  for (int j = 0; j < 2; ++j)

  {
```

```
        printf("%.1f\t", result[i][j]);

        if (j == 1)
          printf("\n");
      }
    return 0;
  }
```

C Programming Strings

In C programming, a string is a sequence of characters terminated with a null character `\0`. For example:

```
char greetings[] = "Hello World!";
printf("%s", greetings);
```

c		s	t	r	i	n	g	\0

How to declare a string?

```
char s[5];
```

How to initialize strings?

```
char c[] = "abcd";
```

```
char c[50] = "abcd";
```

```
char c[] = {'a', 'b', 'c', 'd', '\0'};
```

```
char c[5] = {'a', 'b', 'c', 'd', '\0'};
```

Differences

```
char greetings[] = {'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l',
'd', '!', '\0'};
char greetings2[] = "Hello World!";

printf("%lu\n", sizeof(greetings));   // Outputs 13
```

```
printf("%lu\n", sizeof(greetings2));  // Outputs 13
```

Example 1: scanf() to read a string

```
#include <stdio.h>

int main()

{

    char name[20];

    printf("Enter name: ");

    scanf("%s", name);

    printf("Your name is %s.", name);



 return 0;

}
```

Access Strings

```
char greetings[] = "Hello World!";

printf("%c", greetings[0]);
```

Modify Strings

To change the value of a specific character in a string, refer to the index number, and use single quotes:

```
char greetings[] = "Hello World!";

greetings[0] = 'J';

printf("%s", greetings);

// Outputs Jello World! instead of Hello World!
```

Loop Through a String

```
char carName[] = "Volvo";

int i;



for (i = 0; i < 5; ++i) {

  printf("%c\n", carName[i]);

}
```

- **Pointers:**

Pointers in C are used to store the address of variables or a memory location. This variable can be of any data type i.e, int, char, function, array, or any other pointer. The size of the pointer depends on the architecture.

Syntax:

```
datatype  *var_name;

int *myage;

e.g.

int myAge = 43; // an int variable


printf("%d", myAge);  // Outputs the value of myAge (43)

printf("%p", &myAge); /
```

In the example above, `&myAge` is also known as a pointer.

A pointer is a variable that stores the memory address of another variable as its value.

```
int myAge = 43;     // An int variable

int* ptr = &myAge;  // A pointer variable, with the name ptr, that stores the address of myAge


// Output the value of myAge (43)

printf("%d\n", myAge);


// Output the memory address of myAge (0x7ffe5367e044)

printf("%p\n", &myAge);


// Output the memory address of myAge with the pointer (0x7ffe5367e044)

printf("%p\n", ptr)
```

Example explained

Create a pointer variable with the name `ptr`, that points to an `int` variable (`myAge`). Note that the type of the pointer has to match the type of the variable you're working with.

Use the `&` operator to store the memory address of the `myAge` variable, and assign it to the pointer.

Now, `ptr` holds the value of `myAge`'s memory address.

```
int myAge = 43;     // Variable declaration
int* ptr = &myAge;  // Pointer declaration

// Reference: Output the memory address of myAge with the pointer
(0x7ffe5367e044)
printf("%p\n", ptr);

// Dereference: Output the value of myAge with the pointer (43)
printf("%d\n", *ptr);
```

• Call by Value and Call by Reference:

Call By Value:

In this parameter passing method, values of actual parameters are copied to function's formal parameters and the two types of parameters are stored in different memory locations. So any changes made inside functions are not reflected in actual parameters of the caller.
While calling a function, we pass values of variables to it. Such functions are known as "Call By Values".
// C program to illustrate
// call by value

```
1. #include <stdio.h>
2. void swap(int , int); //prototype of the function
3. int main()
4. {
5.    int a = 10;
6.    int b = 20;
```

```
7.     printf("Before swapping the values in main a = %d, b = %d\n",a,b); // printing
       the value of a and b in main
8.     swap(a,b);
9.     printf("After swapping values in main a = %d, b = %d\n",a,b); // The value of
       actual parameters do not change by changing the formal parameters in call by
       value, a = 10, b = 20
10.}
11.void swap (int a, int b)
12.{
13.    int temp;
14.    temp = a;
15.    a=b;
16.    b=temp;
17.    printf("After swapping values in function a = %d, b = %d\n",a,b); // Formal
       parameters, a = 20, b = 10
18.}
```

Call by Reference

While calling a function, instead of passing the values of variables, we pass address of variables(location of variables) to the function known as "Call By References.
// C program to illustrate
// Call by Reference

```
1.  #include <stdio.h>
2.  void swap(int *, int *); //prototype of the function
3.  int main()
4.  {
5.     int a = 10;
```

```
6.    int b = 20;
7.    printf("Before swapping the values in main a = %d, b = %d\n",a,b); // printing
   the value of a and b in main
8.    swap(&a,&b);
9.    printf("After swapping values in main a = %d, b = %d\n",a,b); // The values of
   actual parameters do change in call by reference, a = 10, b = 20
10.}
11.void swap (int *a, int *b)
12.{
13.   int temp;
14.   temp = *a;
15.   *a=*b;
16.   *b=temp;
17.   printf("After swapping values in function a = %d, b = %d\n",*a,*b); // Formal
   parameters, a = 20, b = 10
18.}
```

- **Compound Data Types:**

1. **Struct**
2. **Unoin**
3. **Enum**
4. **Typedef**

Structures (structs):

Structures (also called structs) are a way to group several related variables into one place. Each variable in the structure is known as a member of the structure.

Unlike an array, a structure can contain many different data types (int, float, char, etc.).

Create a Structure

```
struct MyStructure {    // Structure declaration
  int myNum;            // Member (int variable)
  char myLetter;        // Member (char variable)
}; //
```

Create a struct variable with the name "s1":

```
struct myStructure {
  int myNum;
  char myLetter;
};

int main() {
  struct myStructure s1;
  return 0;
}
```

Access Structure Members

To access members of a structure, use the dot syntax (.):
E.g.

```
// Create a structure called myStructure
struct myStructure {
  int myNum;
  char myLetter;
};

int main() {
  // Create a structure variable of myStructure called s1
  struct myStructure s1;

  // Assign values to members of s1
  s1.myNum = 13;
  s1.myLetter = 'B';
```

```
  // Print values
  printf("My number: %d\n", s1.myNum);
  printf("My letter: %c\n", s1.myLetter);

  return 0;
}
```

Now you can easily create multiple structure variables with different values, using just one structure:

```
#include <stdio.h>

struct myStructure {
  int myNum;
  char myLetter;
};

int main() {
  // Create different struct variables
  struct myStructure s1;
  struct myStructure s2;

  // Assign values to different struct variables
  s1.myNum = 13;
  s1.myLetter = 'B';

  s2.myNum = 20;
  s2.myLetter = 'C';

  // Print values
  printf("s1 number: %d\n", s1.myNum);
  printf("s1 letter: %c\n", s1.myLetter);

  printf("s2 number: %d\n", s2.myNum);
  printf("s2 letter: %c\n", s2.myLetter);

  return 0;
```

What About Strings in Structures?

Remember that strings in C are actually an array of characters, and unfortunately, you can't assign a value to an array like this:

```
#include <stdio.h>

struct myStructure {
  int myNum;
  char myLetter;
```

```
  char myString[30];  // String
};

int main() {
  struct myStructure s1;

  // Trying to assign a value to the string
  s1.myString = "Some text";

  // Trying to print the value
  printf("My string: %s", s1.myString);

  return 0;
}
error
```

However, there is a solution for this! You can use the `strcpy()` function and assign the value to `s1.myString`, like this:

```
struct myStructure {
  int myNum;
  char myLetter;
  char myString[30]; // String
};

int main() {
  struct myStructure s1;

  // Assign a value to the string using the strcpy function
  strcpy(s1.myString, "Some text");

  // Print the value
  printf("My string: %s", s1.myString);

  return 0;
}
```

Simpler Syntax

You can also assign values to members of a structure variable at declaration time, in a single line.

Just insert the values in a comma-separated list inside curly braces `{}`. Note that you don't have to use the `strcpy()` function for string values with this technique:

```
// Create a structure
```

```
struct myStructure {
  int myNum;
  char myLetter;
  char myString[30];
};

int main() {
  // Create a structure variable and assign values to it
  struct myStructure s1 = {13, 'B', "Some text"};

  // Print values
  printf("%d %c %s", s1.myNum, s1.myLetter, s1.myString);

  return 0;
}
```

Copy Structures

```
struct myStructure {
 int myNum;
 char myLetter;
};

int main() {
 // Create a structure variable of myStructure called s1
 struct myStructure s1;
 struct myStructure s2;
 // Assign values to members of s1
 s1.myNum = 13;
 s1.myLetter = 'B';

 // Print values
 printf("My number: %d\n", s1.myNum);
 printf("My letter: %c\n", s1.myLetter);

s2=s1;
 printf("My number: %d\n", s2.myNum);
 printf("My letter: %c\n", s2.myLetter);

 return 0;
}
```

Modify Values

If you want to change/modify a value, you can use the dot syntax (`.`).

And to modify a string value, the `strcpy()` function is useful again:

```
struct myStructure {
  int myNum;
  char myLetter;
  char myString[30];
};

int main() {
  // Create a structure variable and assign values to it
  struct myStructure s1 = {13, 'B', "Some text"};

  // Modify values
  s1.myNum = 30;
  s1.myLetter = 'C';
  strcpy(s1.myString, "Something else");

  // Print values
  printf("%d %c %s", s1.myNum, s1.myLetter, s1.myString);

  return 0;
```

```
}
```

Real Life Example

```
struct Car {
  char brand[50];
  char model[50];
  int year;
};

int main() {
  struct Car car1 = {"BMW", "X5", 1999};
  struct Car car2 = {"Ford", "Mustang", 1969};
  struct Car car3 = {"Toyota", "Corolla", 2011};

  printf("%s %s %d\n", car1.brand, car1.model, car1.year);
  printf("%s %s %d\n", car2.brand, car2.model, car2.year);
  printf("%s %s %d\n", car3.brand, car3.model, car3.year);

  return 0;
}
```

Union in C

- Union is like struct, except it uses less memory.
- The keyword union is used to declare the union in C.
- Variables inside the union are called **members of the union.**

Syntax:

```
union unionName
{
   //member definitions
};
```

E.g

```
#include <stdio.h>

union item
{
    int x;
    float y;
    char ch;
};

int main( )
```

```
{
    union item it;
    it.x = 12;
    it.y = 20.2;
    it.ch = 'a';

    printf("%d\n", it.x);
    printf("%f\n", it.y);
    printf("%c\n", it.ch);

    return 0;
}
```

o/p
1101109601
20.199892
A
In the above program, you can see that the values of x and y gets corrupted. Only variable ch prints the expected result. It is because, in union, the memory location is shared among all member data types.

KEY DIFFERENCES:

- **Every member within structure is assigned a unique memory location while in union a memory location is shared by all the data members.**
- **Changing the value of one data member will not affect other data members in structure whereas changing the value of one data member will change the value of other data members in union.**
- **Structure is mainly used for storing various data types while union is mainly used for storing one of the many data types.**
- **In structure, you can retrieve any member at a time on the other hand in union, you can access one member at a time.**
- **Structure supports flexible array while union does not support a flexible array.**

C Enums

An enum is a special type that represents a group of constants (unchangeable values).

To create an enum, use the `enum` keyword, followed by the name of the enum, and separate the enum items with a comma:

```
enum Level {
  LOW,
  MEDIUM,
  HIGH
};
```

Note that the last item does not need a comma.

It is not required to use uppercase, but often considered as good practice.

Enum is short for "enumerations", which means "specifically listed".

```
enum Level myVar;
enum Level myVar = MEDIUM;
```

- `int main() {`

```
  // Create an enum variable and assign a value to it
  enum Level myVar = MEDIUM;

  // Print the enum variable
  printf("%d", myVar);

  return 0;
}
```

- `enum Level {`

```
  LOW = 25,
  MEDIUM = 50,
  HIGH = 75
};
printf("%d", myVar);
```

Note that if you assign a value to one specific item, the next items will update their numbers accordingly:

- `enum Level {`

```
  LOW = 5,
  MEDIUM, // Now 6
  HIGH // Now 7
};
```

- Enum in a Switch Statement

```
#include <stdio.h>

enum Level {
  LOW = 1,
  MEDIUM,
  HIGH
};

int main() {
  enum Level myVar = MEDIUM;

  switch (myVar) {
    case 1:
      printf("Low Level");
      break;
    case 2:
      printf("Medium level");
      break;
    case 3:
      printf("High level");
      break;
  }

  return 0;
}
```

typedef in C

The typedef is a keyword that is used in C programming to provide existing data types with a new name. typedef keyword is used to redefine the name already the existing name.
Syntax:

```
typedef <existing_name> <alias_name>
```

Example:

```
typedef long long ll
```

E.g.
#include<stdio.h>
#include<string.h>
#include <stdio.h>

typedef int number;

```
// Driver code
int main()
{
  number var = 20;
  printf("%d", var);

  return 0;
}
```

Using typedef with an Array

```
typedef int arr[20]
```

```
// C program to implement
// typedef with array
#include <stdio.h>

typedef int Arr[4];

// Driver code
int main()
{
  Arr temp = {10, 20, 30, 40};
  printf("typedef using an array\n");

  for (int i = 0; i < 4; i++)
  {
    printf("%d ", temp[i]);
  }
  return 0;
}
```

```
typedef using an array
10 20 30 40
```

I/O function:

- **printf():**

printf() function is used in a C program to display any value like float, integer, character, string, etc on the console screen. It is a pre-defined function that is already declared in the stdio.h(header file).

Syntax 1:

To display any variable value.

printf("Format Specifier", var1, var2, …., varn);

- ***scanf():***

scanf() function is used in the C program for reading or taking any value from the keyboard by the user, these values can be of any data type like integer, float, character, string, and many more. This function is declared in stdio.h(header file), that's why it is also a pre-defined function. In scanf() function we use &(address-of operator) which is used to store the variable value on the memory location of that variable.

Syntax:

scanf("Format Specifier", &var1, &var2, …., &varn);

```
// C program to implement
// scanf() function
```

```
#include <stdio.h>

// Driver code
int main()
{
    int num1;

    // Printing a message on
    // the output screen
    printf("Enter a integer number: ");

    // Taking an integer value
    // from keyboard
    scanf("%d", &num1);

    // Displaying the entered value
    printf("You have entered %d", num1);

    return 0;
```

C Preprocessor Directives

The C preprocessor is a micro processor that is used by compiler to transform your code before compilation. It is called micro preprocessor because it allows us to add macros.

All preprocessor directives starts with hash # symbol.

Let's see a list of preprocessor directives.

- **#include**
- **#define**
- **#undef**
- **#ifdef**

- #ifndef
- #if
- #else
- #elif
- #endif
- #error
- #pragma

C #define

The #define preprocessor directive is used to define constant or micro substitution. It can use any basic data type.

#define token value

```
1. #include <stdio.h>
2. #define PI 3.14
3. main() {
4.    printf("%f",PI);
5. }
```

#undef Directive

#undef directive tells the preprocessor to remove all definitions for the specified macro. A macro can be redefined after it has been removed by the #undef directive.

```
/* Example using #undef directive by TechOnTheNet.com */

#include <stdio.h>

#define YEARS_OLD 12

#undef YEARS_OLD
```

```
int main()
{
   #ifdef YEARS_OLD
   printf("TechOnTheNet is over %d years old.\n", YEARS_OLD);
   #endif

   printf("TechOnTheNet is a great resource.\n");

   return 0;
}
```

#if Directive

In the C Programming Language, the #if directive allows for conditional compilation. The preprocessor evaluates an expression provided with the #if directive to determine if the subsequent code should be included in the compilation process

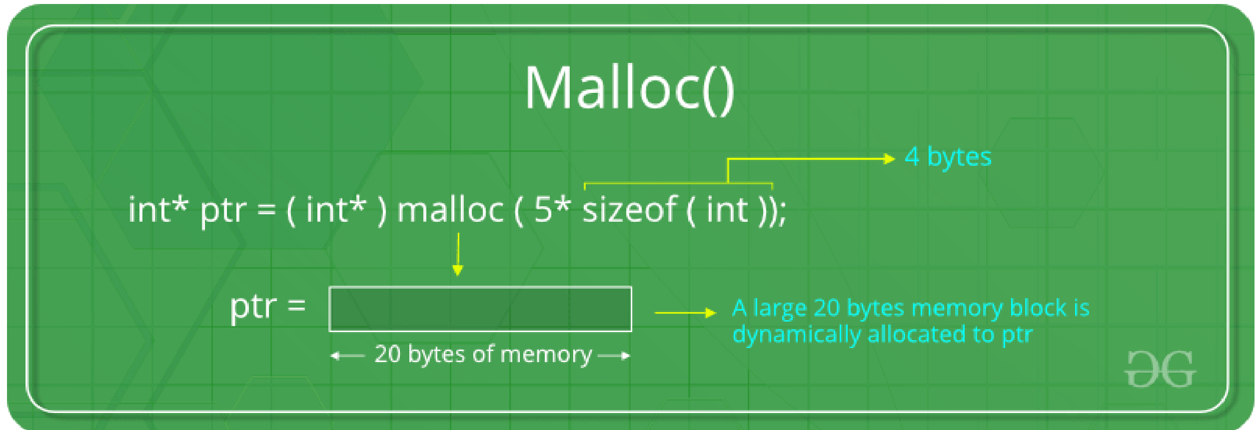
```
/* Example using #if directive by TechOnTheNet.com */

#include <stdio.h>

#define WINDOWS 1

int main()
{
   printf("TechOnTheNet is a great ");

   #if WINDOWS
   printf("Windows ");
   #endif

   printf("resource.\n");

   return 0;
}lj
```

Dynamic Memory Allocation

1. **malloc():Syntax:**

```
ptr = (cast-type*) malloc(byte-size)
For Example:
```

ptr = (int*) malloc(100 * sizeof(int));

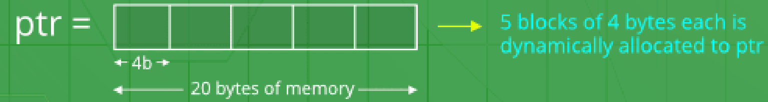


C calloc() method

1. **"calloc" or "contiguous allocation" method in C is used to dynamically allocate the specified number of blocks of memory of the specified type. it is very much similar to malloc() but has two different points and these are:**
2. **It initializes each block with a default value '0'.**
3. **It has two parameters or arguments as compare to malloc().**

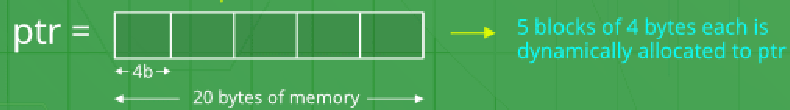
```
ptr = (cast-type*)calloc(n, element-size);
```

```
here, n is the no. of elements and element-size is the size of
each element.
```



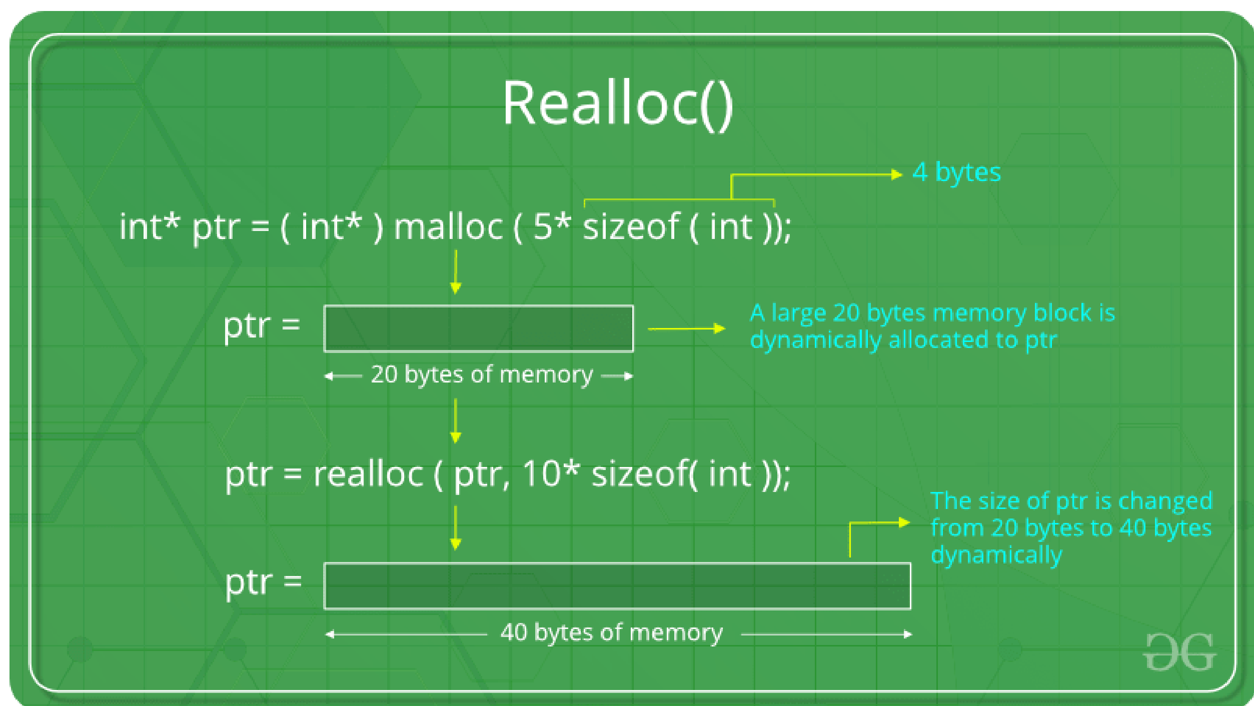
C free() method

“free” method in C is used to dynamically de-allocate the memory.



C realloc() method

“realloc” or “re-allocation” method in C is used to dynamically change the memory allocation of a previously allocated memory. In other words, if the memory previously allocated with the help of malloc or calloc is insufficient, realloc can be used to dynamically re-allocate memory. re-allocation of memory maintains the already present value and new blocks will be initialized with the default garbage value.



Error Handling in C

- **C language does not provide any direct support for error handling. However a few methods and variables defined in error.h header file can be used to point out using the return statement in a <u>function</u>.**

- **Whenever a function call is made in C language, a variable named `errno` is associated with it.**

errno, perror(). and strerror()

The C programming language provides perror() and strerror() functions which can be used to display the text message associated with errno.

- The perror() function displays the string you pass to it, followed by a colon, a space, and then the textual representation of the current errno value.
- The strerror() function, which returns a pointer to the textual representation of the current errno value.

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

main() {

  int dividend = 20;
  int divisor = 0;
  int quotient;

  if( divisor == 0){
    fprintf(stderr, "Division by zero! Exiting...\n");
    exit(-1);
  }

  quotient = dividend / divisor;
  fprintf(stderr, "Value of quotient : %d\n", quotient );

  exit(0);
}
```

- Program Exit Status

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

```
#include <stdlib.h>

main() {

  int dividend = 20;
  int divisor = 5;
  int quotient;

  if( divisor == 0) {
    fprintf(stderr, "Division by zero! Exiting...\n");
    exit(EXIT_FAILURE);
  }

  quotient = dividend / divisor;
  fprintf(stderr, "Value of quotient : %d\n", quotient );

  exit(EXIT_SUCCESS);
}
```

What is GCC?

The GNU Compiler Collection, commonly known as GCC, is a set of compilers and development tools available for Linux, Windows, various BSDs, and a wide assortment of other operating systems. It includes support primarily for C and C++ and includes Objective-C, Ada, Go, Fortran, and D. The Free Software Foundation (FSF) wrote GCC and released it as completely free (as in libre) software.

Source Code Editors

Although, you may choose any basic text editor such as notepad for writing and editing source code of C, we recommend choosing one of the editors below.

1. Notepad++ (Only for Windows)
2. Microsoft Visual Studio Code (For Windows and Linux)
3. ATOM (For Windows and Linux)
4. IDEs (Integrated Development Environment) such as Eclipse or Netbeans may be used but if you are a beginner in C programming, prefer using text editors mentioned above.

- **Multiple File compilation:**

```
#include <stdio.h>
#define B
#ifdef A
 int main()
 {
         printf("hello world");

 }
 #endif
 #ifdef B
 int main()
 {
         printf("hello India");

}
            #endif
```