

**C Programming Language Tutorial**

***EASY PROGRAMMING***

**FEATURES OF C LANGUAGE:**

1. Simple
2. Fast speed
3. Rich Library
4. structured programming language
5. Mid-level programming language
6. Memory Management
7. Machine Independent or Portable
8. Pointers
9. Recursion
10. Extensible

**SIMPLE:** C is a simple language in the sense that it provides a **structured approach** (to break the problem into parts).

**FAST SPEED:** The compilation and execution time of C language is fast.

**RICH LIBRARY:** C **provides a lot of inbuilt functions** that make the development fast.

**STRUCTURED PROGRAMMING LANGUAGE:** C is a structured programming language in the sense that **we can break the program into parts using functions.**

**MID-LEVEL PROGRAMMING LANGUAGE:**  C is **intended to do low-level programming.** It**also supports the features of a high-level language**. That is why it is known as mid-level language.

**MEMORY MANAGEMENT:** It supports the feature of **dynamic memory allocation**. We can free the allocated memory at any time by calling the **free ()** function.

**MACHINE INDEPENDENT OR PORTABLE:** c programs **can be executed on different machines.**

**POINTERS:** The pointer in C language is a **variable which stores the address of another variable.**

**RECURSION:** Recursion is **expressing an entity in terms of itself.**

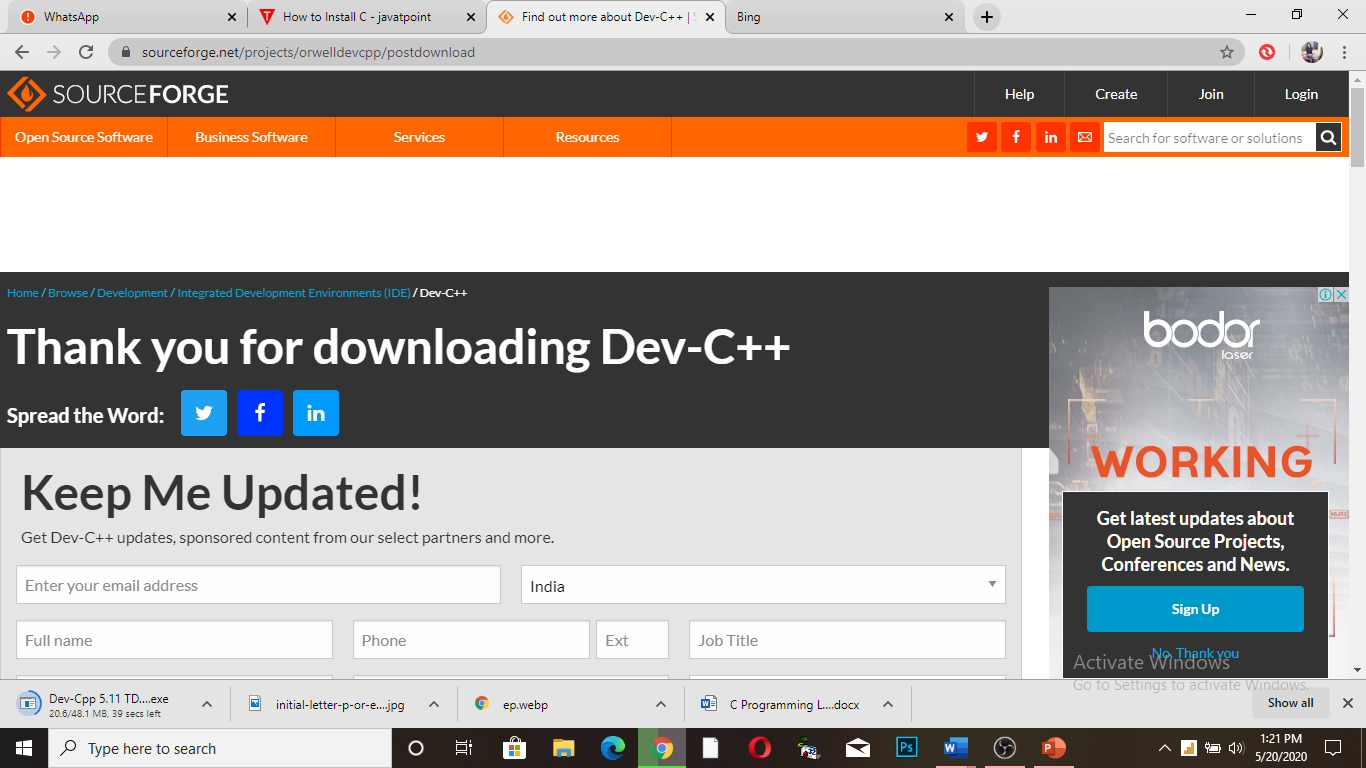
**EXTENSIBLE:** C language is extensible because it **can easily adopt new features.**

**HOW TO INSTALL C**

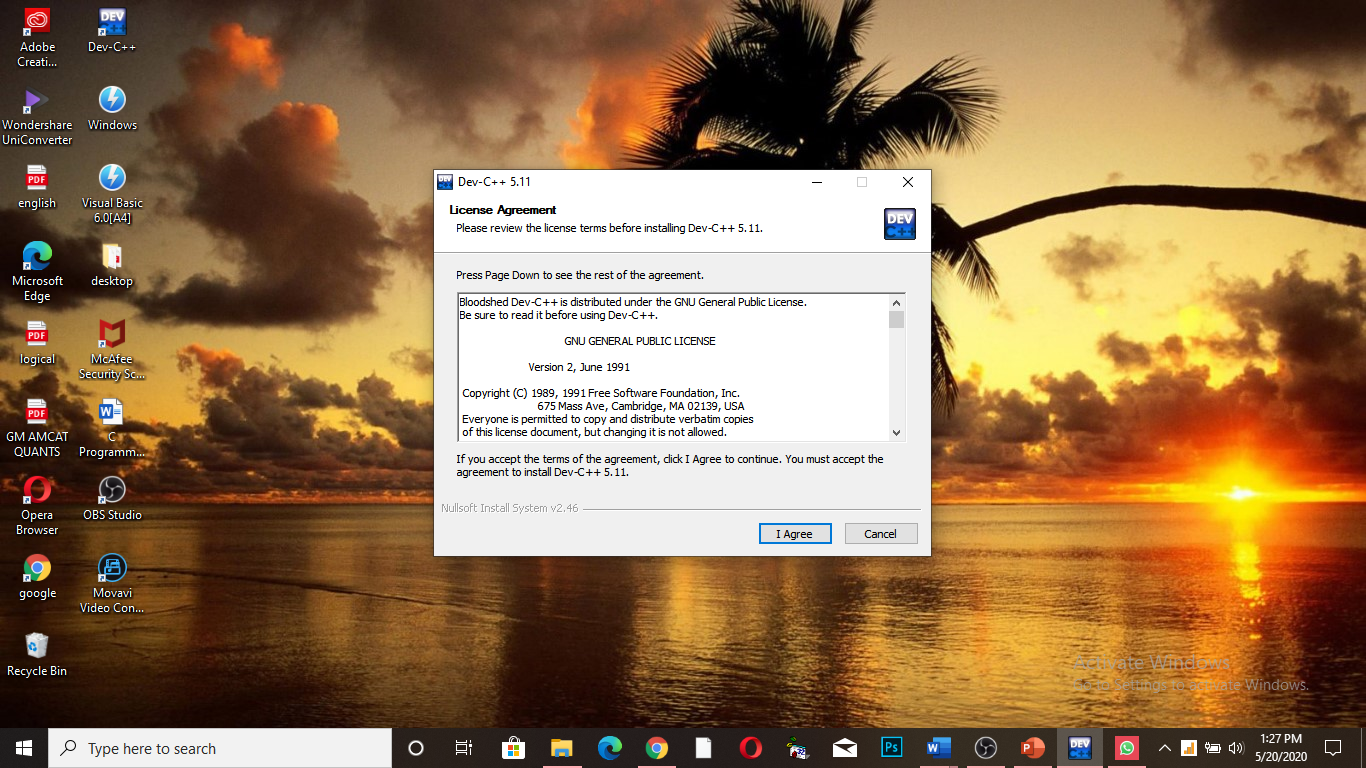
There are many compilers available for c language, you need to download any one and I am going to tell you about Dev C++ and Turbo C++ both. You can choose any one of the following, so let’s begin:

**Steps to download Dev C++**

**1.**Download Dev C++ by this link <https://sourceforge.net/projects/orwelldevcpp/>



2.open Dev C++ for setup



3.click next and then install the software and then finish.

**Steps to download Turbo C++**

## Go to <https://turbo-c.soft32.com/>

## Click to download now, When the download is complete, locate the zipped folder named Turbo.C.3.2, right click on the zipped folder (Turbo C.3.2.zip) then click on Extract.

## Find Turbo C++ setup file then double click on it to begin the installation process.

## Accept the License Agreement and click on next, follow the onscreen instruction to complete the installation process.

## 

## 

## 

## 

## 10.Open the software you will see interface like this

## 

** C PROGRAMMING LANGUAGE TUTORIAL**

**Variables in C**

A variable is nothing but a name given to a storage area that our program can manipulate. Each variable in c has a specific type, which determines the size and layout of the variable’s memory.

Let's see the syntax to declare a variable:

type variable\_list;

examples:

1. int a;
2. float b;
3. char c;

Here a, b, c are the variables. The int, float, char are the data types.

**Rules for defining variables**

1. A variable can have alphabets, digits, and underscore.
2. A variable name can start with the alphabet, and underscore only. It can't start with a digit.
3. No whitespace is allowed within the variable name.
4. A variable name must not be any reserved word or keyword, e.g. int, float, etc.

## Types of Variables in C

1. local variable
2. global variable
3. static variable
4. automatic variable
5. external variable

** C PROGRAMMING LANGUAGE TUTORIAL**

### **Local Variable**

A variable that is declared inside the function or block is called a local variable.

void function () {

int x=1;//**local variable**

}

### **Global Variable:** A variable that is declared outside the function or block is called a global variable.

int value=2;//**global variable**

void function () {

int x=1;//**local variable**

}

### **Static Variable:** A variable that is declared with the static keyword is called static variable. Static is used for both local and global variables. It creates once, when program’s execution enters in the function first time, destroys when program’s execution finishes.

void function () {

int x=1;//**local variable**

static int y=10;//**static variable**

x=x+1;

y=y+1;

printf ("%d, %d",x,y);

}

### **Automatic Variable:** We can explicitly declare an automatic variable using **auto keyword**, all variables in C that are declared inside the block, are automatic variables by default. Automatic variables create a new each time when program’s execution enters in the function and destroyed when leaves.

void main () {

int x=1;//local variable (also automatic)

auto int y=2;//automatic variable

}

**External variable:** An external variable is a**variable defined outside any function block.**

To declare an external variable, you need to use **extern keyword.**

extern int x=1;//external variable (also global)

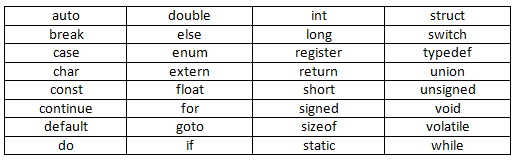
**NOTE:** Global variable is a variable that is available throughout the program. An extern variable is also available throughout the program but extern only declares the variables but it doesn’t allocate any memory for this variable.

**C PROGRAMMING LANGUAGE TUTORIAL**

**Keywords in c**

**Keywords** are specific reserved words in **C** each of which has a specific feature associated with it.

You cannot use it as a variable name, constant name, etc. There are only 32 reserved words (keywords) in the C language.



We will discuss each term in upcoming videos in detail.

****

** C PROGRAMMING LANGUAGE TUTORIAL**

**C Identifiers**

In **C** language **identifiers** are the names given to variables, constants, functions and user-define data. These **identifiers** are defined against a set of rules i.e.

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

**Note:** The identifier is only used to identify an entity uniquely in a program at the time of execution whereas, a variable is a name given to a memory location to hold a value. Variable is only a kind of identifiers.

 **C PROGRAMMING LANGUAGE TUTORIAL**

There are two types of identifiers:

* Internal identifier
* External identifier

**Internal identifier**

The internal identifiers can be local variables. If the identifier is not used in the external linkage, then it is known as an internal identifier.

**External identifier**

The external identifiers can be global variables. If the identifier is used in the external linkage, then it is known as an external identifier.

Differences between Keyword and Identifier

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



 **C PROGRAMMING LANGUAGE TUTORIAL**

# **C Operators**

An operator is simply a symbol that is used to perform operations. There are many types of operators in c language i.e.

* Arithmetic operators
* Relational operators
* Logical operators
* Bitwise operators
* Assignment operators
* Conditional operators
* Special operators

**Arithmetic operators**

The following table shows all the basic arithmetic operators.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| + | Adds two operands |
| - | Subtract second operands from first |
| \* | Multiply two operands |
| / | Divide numerator by denominator |
| % | Remainder of division |
| ++ | Increases integer value by one |
| -- | Decreases integer value by one |

### **Relational operators**

|  |  |
| --- | --- |
| Operator | description |
| > | Check if operand on the left is greater than operand on the right |
| < | Check operand on the left is smaller than right operand |
| >= | check left operand is greater than or equal to right operand |
| <= | Check if operand on left is smaller than or equal to right operand |
| == | Check if two operands are equal |
| != | Check if two operands are not equal. |

### **Logical operators**

C language supports 3 logical operators.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| && | Logical AND | (a&&b) is false |
| || | Logical OR | (a||b) is true |
| ! | Logical NOT | (! a) is false |

 **C PROGRAMMING LANGUAGE TUTORIAL**

### **Bitwise operators**

Bitwise operators perform manipulations of data at bit level. These operators also perform shifting of bits from right to left.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| & | Bitwise AND |
| | | Bitwise OR |
| ^ | Bitwise exclusive OR |
| << | Left shift |
| >> | Right shift |

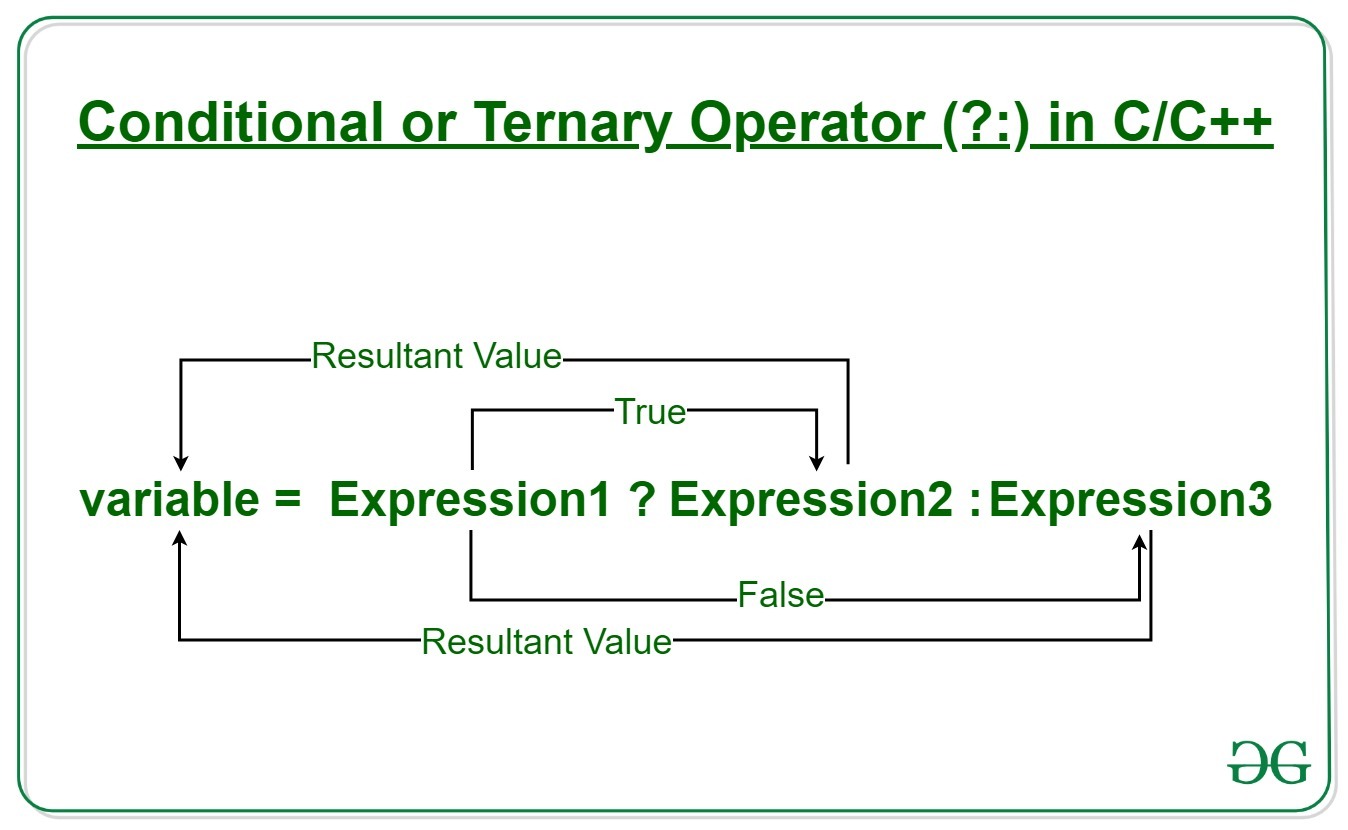
### **Assignment Operators**

|  |  |  |
| --- | --- | --- |
| **Operators** | **Description** | **Example** |
| = | assigns values from right side operands to left side operand | a=b |
| += | adds right operand to the left operand and assign the result to left | a+=b is same as a=a+b |
| -= | subtracts right operand from the left operand and assign the result to left operand | a-=b is same as a=a-b |
| \*= | multiply left operand with the right operand and assign the result to left operand | a\*=b is same as a=a\*b |
| /= | Divides left operand with right operand and assign the result to left operand | a/=b is same as a=a/b |
| %= | Calculate modulus using two operands and assign the result to left operand | a%=b is same as a=a%b |

### **Conditional operator**

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

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

****

Examples:

int x = 20, y = 10;

var result = x > y? "x is greater than y”: "x is less than or equal to y";

out: - x is greater than y

 **C PROGRAMMING LANGUAGE TUTORIAL**

### **Special operators**

|  |  |  |
| --- | --- | --- |
| **Operators** | **Description** | **Examples** |
| sizeof | Returns the size of an variable | sizeof(x) return size of the variable x |
| & | Returns the address of an variable | &x ; return address of the variable x |
| \* | Pointer to a variable | \*x ; will be pointer to a variable x |

# **C Format Specifier**

The format specifiers determine the format of the input and output. The format specifier always starts with a '%' character. 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 ().

| **SPECIFIER** | **USED FOR** |
| --- | --- |
| %c | a single character |
| %s | a string |
| %hi | short (signed) |
| %hu | short (unsigned) |
| %Lf | long double |
| %n | prints nothing |
| %d | a decimal integer (assumes base 10) |
| %i | a decimal integer (detects the base automatically) |
| %o | an octal (base 8) integer |
| %x | a hexadecimal (base 16) integer |
| %p | an address (or pointer) |
| %f | a floating-point number for floats |
| %u | int unsigned decimal |
| %e | a floating-point number in scientific notation |
| %E | a floating-point number in scientific notation |
| %% | the % symbol |

* **%c**

1. **int** main ()
2. {
3. **char** a='c';
4. printf ("Value of a is: %c", a);
5. **return** 0;
6. }

Out: Value of a is: c

* **%s**

1. **int** main ()
2. {
3. printf ("%s", "Easy programming");
4. **return** 0;
5. }

Out: Easy programming

%f and %lf

the %f is a float and the %lf is a long float ... float 3.14 vs long float 3.141662354652365237452374572355

**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;  } |

**Output:**

**45**

**45**

**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;  } |

**Output:**

**12.670000**

**1.2670000e+01**

**Unsigned Octal number for integer : %o**

|  |
| --- |
| #include <stdio.h>  int main()  {      int a = 67;      printf("%o\n", a);      return 0;  } |

**Output:**

**103**

**Unsigned Hexadecimal for integer : %x, %X**

|  |
| --- |
| #include <stdio.h>  int main()  {      int a = 15;      printf("%x\n", a);      return 0;  } |

**Output:**

**f**

**String printing : %s**

|  |
| --- |
| #include <stdio.h>  int main()  {      char a[] = "Easy programming";      printf("%s\n", a);      return 0;  } |

**Output:**

Easy programming

|  |
| --- |
| #include <stdio.h>  int main()  {      int a = 0;      scanf("%d", &a); // input is 45      printf("%d\n", a);      return 0;  } |

**Integer may be octal or in hexadecimal : %i**

|  |
| --- |
| #include <stdio.h>  int main()  {      int a = 0;      scanf("%i", &a); // input is 017 (octal of 15 )      printf("%d\n", a);      scanf("%i", &a); // input is 0xf (hexadecimal of 15 )      printf("%d\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;  } |

**Output:**

**45.650000**

**These are some basic examples of format specifiers that are mostly used in our program**

 **C PROGRAMMING LANGUAGE TUTORIAL**

### **Constants in c**

Constants refer to fixed values that the program may not alter during its execution. These fixed values are also called **literals**.

Constants can be of any of the basic data types like an integer constant, a floating constant, a character constant, or a string literal.

|  |  |
| --- | --- |
| **Constant** | **Example** |
| Decimal Constant | 10, 20, 450 etc. |
| Real or Floating-point Constant | 10.3, 20.2, 450.6 etc. |
| Octal Constant | 021, 033, 046 etc. |
| Hexadecimal Constant | 0x2a, 0x7b, 0xaa etc. |
| Character Constant | 'a', 'b', 'x' etc. |
| String Constant | "c", "c program", "c in easy programming" etc. |

There are two simple ways in C to define constants –

* **Using const keyword.**
* #include<stdio.h>
* int main(){
* const float PI=3.14;
* printf("The value of PI is: %f",PI);
* return 0;
* }

**Note:** If you try to change the value of PI, it will render compile time error.

* **Using #define preprocessor.**

The #define preprocessor directive is used to define constant or micro substitution.

1. #include <stdio.h>
2. #define MIN(a,b) ((a)<(b)?(a):(b))
3. void main() {
4. printf("Minimum value is: %d\n", MIN(10,20));
5. }

Output:

Minimum is:10

 **C PROGRAMMING LANGUAGE TUTORIAL**

# **Escape Sequence in C**

### An **escape sequence**in**C** language is a sequence of characters that doesn't represent itself when used inside string literal or character.

|  |  |
| --- | --- |
| **Sequence** | **Meaning** |
| \b | Backspace |
| \r | For carriage return |
| \n | Newline |
| \t | Horizontal Tab |
| \v | Vertical Tab |
| \\ | Backslash |
| \’ | Single Quote |
| \” | Double Quote |
| \? | Question Mark |
| \a | Beep sound |

#include<stdio.h>

int main(){

printf("hello\bworld");

printf("\n");

printf("priya\r");

printf("\n");

printf("babli\rAnj");

printf("\n");

printf("how\tare\tyou\n");

printf("\\hello\n");

printf("\'hello\n");

printf("\?hello\n");

printf("\''hello\n");

printf("\ahow are you");

}

 **C PROGRAMMING LANGUAGE TUTORIAL**

# **C if else Statement**

The if-else statement in C is used to perform the operations based on some specific condition. **If the Boolean expression evaluates to true,** then the if block will be executed, otherwise, the else block will be executed.

## If Statement:

## An if statement is a **programming conditional statement that, if proved true, performs a function or displays information**.

## Syntax

1. **if**(expression){
2. //code to be executed
3. }

**If else statement**

**Syntax:**

1. **if**(expression){
2. //code to be executed if condition is true
3. }**else**{
4. //code to be executed if condition is false
5. }

#include<stdio.h>

int main()

{

int a, b, big;

printf("Enter any two number: ");

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

if(a>b)

big=a;

else

big=b;

printf("\nBiggest of the two number is: %d", big);

}

## if else-if Statement

## if a condition is true then the statements defined in the if block will be executed, otherwise if some other condition is true then the statements defined in the else-if block will be executed, at the last if none of the condition is true then the statements defined in the else block will be executed

**Syntax:**

1. **if**(condition1){
2. //code to be executed if condition1 is true
3. }**else** **if**(condition2){
4. //code to be executed if condition2 is true
5. }
6. **else** **if**(condition3){
7. //code to be executed if condition3 is true
8. }
9. ...
10. **else**{
11. //code to be executed if all the conditions are false
12. }

 **C PROGRAMMING LANGUAGE TUTORIAL**

# **C Switch Statement**

The **switch** statement allows us to execute one code block among many alternatives.  Each value is called a **case**, and the variable being **switched**on is checked for each **switch case**.

The **syntax** of switch statement is:

1. **switch**(expression){
2. **case** value1:
3. //code to be executed;
4. **break**;  //optional
5. **case** value2:
6. //code to be executed;
7. **break**;  //optional
8. ......
10. **default**:
11. code to be executed **if** all cases are not matched;
12. }

**Note:** The break in C or C++ is a **loop control statement**which is used to **terminate the loop.**

### **Rules for switch statement**

 1)The case value must be an integer or character constant.

2) The case value can be used only inside the switch statement.

3) The switch expression must be of an integer or character type.

4)The break statement in switch case is optional. If there is no break statement found in the case, all the cases will be executed present after the matched case.

Example: switch case using break statement

#include <stdio.h>

int main()

{

int num;

printf("Enter value of num:");

scanf("%d",&num);

switch (num)

{

case 1:

printf("You have entered value 1\n");

break;

case 2:

printf("You have entered value 2\n");

break;

case 3:

printf("You have entered value 3\n");

break;

default:

printf("Input value is other than 1,2 & 3 ");

}

return 0;

}

Output

Enter value of num: 3

You have entered value 3

Example: switch case without using break statement

#include <stdio.h>

int main()

{

int num;

printf("Enter value of num:");

scanf("%d",&num);

switch (num)

{

case 1:

printf("You have entered value 1\n");

case 2:

printf("You have entered value 2\n");

case 3:

printf("You have entered value 3\n");

default:

printf("Input value is other than 1,2 & 3 ");

}

return 0;

}

Output:

Enter value of num:2

You have entered value 2

You have entered value 3

Input value is other than 1,2 & 3

 **C PROGRAMMING LANGUAGE TUTORIAL**

**Difference Between if else and switch Case**

|  |  |
| --- | --- |
| **if else** | **switch Case** |
| if statement evaluates integer, character, pointer or floating-point type or Boolean type. | switch statement evaluates only character or integer value. |
| If the expression in the “if” block is true, the statements inside the “if” block will be executed. If not, the else block will execute. | Executes the statements in the matched case until a break statement. |
| if-else statement test for equality as well as for logical expression. | switch statement test only for equality. |
| If a statement is used to select among two alternatives | The switch statement is used to select among multiple alternatives. |
| It is difficult to edit the if-else statement if the nested if-else statement is used. | It is easy to edit switch cases as they are recognized easily. |
| If there are multiple choices implemented through 'if-else', then the speed of the execution will be slow. | If we have multiple choices then the switch statement is the best option as the speed of the execution will be much higher than 'if-else'. |

 **C PROGRAMMING LANGUAGE TUTORIAL**

# **C Loops**

A **loop**statement allows us to execute a statement or group of statements multiple times. A **loop** in a computer program is an instruction that repeats until a specified condition is reached.

1)It provides code reusability.

2) Using loops, we do not need to write the same code again and again.

## Types of C Loops

There are three types of loops in [C language](https://www.javatpoint.com/c-programming-language-tutorial)

1. while
2. do while
3. For

**While loop**

It is an entry-controlled loop. In while loop, a condition is evaluated before processing a body of the loop. If a condition is true then and only then the body of a loop is executed.

**Syntax:**

1. **while**(condition){
2. //code to be executed
3. }

**Example:**

#include<stdio.h>

int main(){

int i=1;

while(i<=10){

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

i++;

}

return 0;

}

**2nd example:**

#include<stdio.h>

int main(){

int i=1;

while(i>10){

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

i++;

}

return 0;

}

### **do-while loop in C**

A do-while loop is similar to the while loop except that the condition is always executed after the body of a loop.

**NOTE:** In a while loop, the body is executed if and only if the condition is true. In some cases, we have to execute a body of the loop at least once even if the condition is false. This type of operation can be achieved by using a do-while loop.

**Syntax:**

1. **do**{
2. //code to be executed
3. }**while**(condition);

**Example:**

#include<stdio.h>

int main(){

int i=1;

do{

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

i++;

}while(i<=10);

return 0;

}

**2nd Example:**

#include<stdio.h>

int main(){

int i=1;

do{

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

i++;

}while(i>10);

return 0;

}

**For loop**

contains three parts: the initialization, the condition, and afterthought or Iterator-based loops.

* The initial value of the for loop is performed only once.
* The condition is a Boolean expression that tests and compares the counter to a fixed value after each iteration, stopping the for loop when false is returned.
* The incrementation/decrementation increases (or decreases) the counter by a set value.

**Syntax:**

1. **for**(initialization;condition;incr/decr){
2. //code to be executed
3. }

**Example:**

1. #include<stdio.h>
2. int main(){
3. int i=0;
4. for(i=1;i<=10;i++){
5. printf("%d \n",i);
6. }
7. return 0;
8. }

 **C PROGRAMMING LANGUAGE TUTORIAL**

# **C continue statement**

The **continue** statement in **C** programming works somewhat like the break statement. Instead of forcing termination, it forces the next iteration of the loop to take place, skipping any code in between.

### **Syntax:**

1. //loop statements
2. continue;
3. //some lines of the code which is to be skipped

**Example:**

1. #include<stdio.h>
2. int main(){
3. int i=1;//initializing a local variable
4. //starting a loop
5. for(i=1;i<=10;i++){
6. if(i==5){
7. continue;
8. }
9. printf("%d \n",i);
10. }return 0;
11. }

 **C PROGRAMMING LANGUAGE TUTORIAL**

**Goto statement**

The Goto statement in C Programming is used to **alter the flow of a program.** When the compiler reaches the goto statement then it will jump unconditionally to the location specified in the goto statement .

**Syntax:**

1. label:
2. //some part of the code;
3. **goto** label;

**Example:**

#include <stdio.h>

int main()

{

int Totalmarks;

printf(" \n Please Enter your Subject Marks \n ");

scanf("%d", & Totalmarks);

if(Totalmarks >= 50)

{

goto Pass;

}

else

goto Fail;

Pass:

printf(" \n Congratulation! You made it \n");

Fail:

printf(" \n Better Luck Next Time \n");

return 0;

}

  **C PROGRAMMING LANGUAGE TUTORIAL**

# **C Functions**

 A **function** is a set of statements that take inputs, do some specific computation and produces output.  Suppose you are building an application in C language and in one of your program, you need to perform a same task more than once. In such case you have two options –

a) Use the same set of statements every time you want to perform the task  
b) Create a function to perform that task, and just call it every time you need to perform that task.

## Function Aspects

* **Function declaration** A function must be declared globally in a c program to tell the compiler about the function name, function parameters, and return type.
* **Function call** Function can be called from anywhere in the program. The parameter list must not differ in function calling and function declaration. We must pass the same number of functions as it is declared in the function declaration.
* **Function definition** It contains the actual statements which are to be executed. It is the most important aspect to which the control comes when the function is called. Here, we must notice that only one value can be returned from the function.

|  |  |
| --- | --- |
| Function declaration | return\_type function\_name (argument list); |
| Function call | function\_name (argument\_list) |
| Function definition | return\_type function\_name (argument list) {function body;} |

**Types of function in c:**

1. **Library Functions**: are the functions which are declared in the C header files such as scanf (), printf(), gets(), puts(), ceil(), floor() etc.
2. **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.

## Different aspects of function calling

* function without arguments and without return value
* function without arguments and with return value
* function with arguments and without return value
* function with arguments and with return value

### **Example for Function without argument and without return value**

**Example 1**

1. #include<stdio.h>
2. **void** Easyprogramming();  //function declaration
3. **void** main ()
4. {
5. printf("Hello ");
6. Easyprogramming ();  //function calling
7. }
8. **void** Easyprogramming ()  //function defination
9. {
10. printf("welcome to our channel");
11. }

### **Example for Function without argument and with return value**

1. #include<stdio.h>
2. **int** sum();  // declaration
3. **void** main()
4. {
5. **int** result;
6. printf("\nGoing to calculate the sum of two numbers:");
7. result = sum();  //call
8. printf("%d",result);
9. }
10. **int** sum()  //defination
11. {
12. **int** a,b;
13. printf("\nEnter two numbers");
14. scanf("%d %d",&a,&b);
15. **return** a+b;
16. }

### **Example for Function with argument and without return value**

1. #include<stdio.h>
2. **void** sum(**int**, **int**);
3. **void** main()
4. {
5. **int** a,b,result;
6. printf("\nGoing to calculate the sum of two numbers:");
7. printf("\nEnter two numbers:");
8. scanf("%d %d",&a,&b);
9. sum(a,b);
10. }
11. **void** sum(**int** a, **int** b)
12. {
13. printf("\nThe sum is %d",a+b);
14. }

### **Example for Function with argument and with return value**

1. #include<stdio.h>
2. **int** sum(**int**, **int**);
3. **void** main()
4. {
5. **int** a,b,result;
6. printf("\nGoing to calculate the sum of two numbers:");
7. printf("\nEnter two numbers:");
8. scanf("%d %d",&a,&b);
9. result = sum(a,b);
10. printf("\nThe sum is : %d",result);
11. }
12. **int** sum(**int** a, **int** b)
13. {
14. **return** a+b;
15. }

 **C PROGRAMMING LANGUAGE TUTORIAL**

# **Recursion in C**

**Recursion** is the process of repeating items in a self-similar way i.e. A function that calls itself is known as a **recursive** function. And, this technique is known as **recursion.**

**Example:**

#include<stdio.h>

long int multiply(int n);

int main() {

int n;

printf("Enter a positive integer: ");

scanf("%d",&n);

printf("Factorial of %d = %ld", n, multiply(n));

return 0;

}

long int multiply(int n) {

if (n>=1)

return n\*multiply(n-1);

else

return 1;

}

 **C PROGRAMMING LANGUAGE TUTORIAL**

# **C Array**

An array is defined as the collection of similar type of data items stored at contiguous memory locations. Arrays are the derived data type in C programming language which can store the primitive type of data such as int, char, double, float, etc. It also has the capability to store the collection of derived data types, such as pointers, structure, etc.

**Syntax:**

data\_type array\_name[array\_size];

**Declaration**

**int** number[4];

**Initialization**

number[0]=60

number[1]=50

number[2]=70

number[3]=30

|  |  |  |  |
| --- | --- | --- | --- |
| **60** | **50** | **70** | **30** |

**number[0] number[1] number[2] number[3]**

1. #include<stdio.h>
2. int main(){
3. int i=0;
4. int marks[4];//declaration of array
5. marks[0]=60;//initialization of array
6. marks[1]=50;
7. marks[2]=70;
8. marks[3]=30;
9. //traversal of array
10. for(i=0;i<4;i++){
11. printf("%d \n",marks[i]);
12. }//end of for loop
13. return 0;
14. }

## Declaration with Initialization

## int number[4]={60,50,70,30};

## Example:

1. #include<stdio.h>
2. int main(){
3. int i=0;
4. int marks[4]={60,50,70,30 };//declaration and initialization of array
5. //traversal of array
6. for(i=0;i<4;i++){
7. printf("%d \n",marks[i]);
8. }
9. return 0;
10. }

  **C PROGRAMMING LANGUAGE TUTORIAL**

# **Two Dimensional Array**

## The two-dimensional array can be defined as an array of arrays. The 2D array is organized as matrices which can be represented as the collection of rows and columns.

## Syntax:

data\_type array\_name[rows][columns];

**Declaration:**

int twodimen[4][3];

## Initialization

## int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};

### **Two-dimensional array example**

1. #include<stdio.h>
2. int main(){
3. int i=0,j=0;
4. int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};
5. //traversing 2D array
6. for(i=0;i<4;i++){
7. for(j=0;j<3;j++){
8. printf("arr[%d] [%d] = %d \n",i,j,arr[i][j]);
9. }//end of j
10. }//end of i
11. return 0;
12. }

Advantages

* It is better and convenient way of storing the data of same datatype with same size.
* It allows us to store known number of elements in it.
* It allocates memory in contiguous memory locations for its elements. It does not allocate any extra space/ memory for its elements. Hence there is no memory overflow or shortage of memory in arrays.
* Iterating the arrays using their index is faster compared to any other methods like linked list etc.
* It allows to store the elements in any dimensional array – supports multidimensional array.

Disadvantages

* It allows us to enter only fixed number of elements into it. We cannot alter the size of the array once array is declared. Hence if we need to insert more number of records than declared then it is not possible. We should know array size at the compile time itself.
* Inserting and deleting the records from the array would be costly since we add / delete the elements from the array, we need to manage memory space too.

  **C PROGRAMMING LANGUAGE TUTORIAL**

# **C Pointers**

The pointer in C language is a variable which stores the address of another variable. This variable can be of type int, char, array, function, or any other pointer.

1. **int** n = 5;
2. **int**\* p = &n;

**Note**

* Variable p of type pointer is pointing to the address of the variable n of type integer.
* The pointer in c language can be declared using \* (asterisk symbol).

**Example:**

1. #include<stdio.h>
2. **int** main(){
3. **int** num=5;
4. **int** \*p;
5. p=&num;//stores the address of number variable
6. printf("Address of p variable is %x \n",p);
7. printf("Value of p variable is %d \n",\*p);
8. **return** 0;
9. }

### **Pointer to array**

1. **int** arr[10];
2. **int** \*p=&arr[10];

### **Pointer to a function**

1. **void** show (**int**);
2. **void**(\*p)(**int**) = &display;

### **Pointer to structure**

1. **struct** st {
2. **int** i;
3. **float** f;
4. }ref;
5. **struct** st \*p = &ref;

  **C PROGRAMMING LANGUAGE TUTORIAL**

# **Dynamic memory allocation**

The concept of **dynamic memory allocation in c language**enables the C programmer to allocate memory at runtime. To allocate memory dynamically, library functions are malloc (), calloc (), realloc () and free () are used. These functions are defined in the <stdlib.h> header file.

## malloc() function :

The malloc() function allocates single block of requested memory.

It doesn't initialize memory at execution time, so it has garbage value initially.

It returns NULL if memory is not sufficient.

**The syntax of malloc() function is :**

ptr=(cast-type\*)malloc(byte-size)

 ptr=(int\*)malloc(n\*sizeof(int));

## calloc() function in C

The calloc() function allocates multiple block of requested memory.

It initially initialize all bytes to zero.

It returns NULL if memory is not sufficient.

**The syntax of calloc() function is:**

ptr=(cast-type\*)calloc(number, byte-size)

ptr=(int\*)calloc(n,sizeof(int));

## realloc() function in C

If memory is not sufficient for malloc() or calloc(), you can reallocate the memory by realloc() function. In short, it changes the memory size.

**syntax of realloc() function:**

ptr=realloc(ptr, new-size)

## free() function in C

The memory occupied by malloc() or calloc() functions must be released by calling free() function. Otherwise, it will consume memory until program exit.

**syntax of free() function:**

free(ptr)

  **C PROGRAMMING LANGUAGE TUTORIAL**

# **C Strings**

A string in C is a sequence of zero or more characters followed by a NULL '0' character.

There are two ways to declare a string in c language.

1. By char array
2. By string literal

**example of declaring** **string by char array:**

char ch[11]={'e', 'a', 's', 'y', 'p', 'r', 'o', 'g', 'r', 'a', 'm',  '\0'};

**example of declaring**  **string by the string literal:**

char ch[]="easyprogram";

**Note:** In such case, '\0' will be appended at the end of the string by the compiler.

**String Example**

The '%s' is used as a format specifier for the string in c language.

#include<stdio.h>

#include <string.h>

int main(){

  char ch[11]={'e', 'a', 's', 'y', 'p', 'r', 'o', 'g', 'r', 'a', 'm',  '\0'};

   char ch2[11]="easyprogram";

   printf("Char Array Value is: %s\n", ch);

   printf("String Literal Value is: %s\n", ch2);

 return 0;

}

**C PROGRAMMING LANGUAGE TUTORIAL**

# **C gets() and puts() functions**

The gets() and puts() are declared in the header file stdio.h. Both the functions are involved in the input/output operations of the strings.

The **gets()** function enables the user to enter some characters followed by the enter key. All the characters entered by the user get stored in a character array. It returns the string entered by the user.

The **puts ()** function is very much similar to printf () function. The puts() function is used to print the string on the console which is previously read by using gets() or scanf() function.

The puts() function returns an integer value representing the number of characters being printed on the console. Since, it prints an additional newline character with the string, which moves the cursor to the new line on the console.

**Example:**

#include<stdio.h>

#include <string.h>

**int** main(){

**char** name[20];

printf("Enter your name: ");

gets(name); //reads string from user

printf("Your name is: ");

puts(name);  //displays string

**return** 0;

}

**C PROGRAMMING LANGUAGE TUTORIAL**

# **strlen() function**

The strlen() function returns the length of the given string. It doesn't count null character '\0'.

#include<stdio.h>

#include <string.h>

**int** main(){

**char** ch[20]=“easyprogramming” ;

   printf("Length of string is: %d",strlen(ch));

**return** 0;

}

# **strcpy()**

The strcpy(destination, source) function copies the source string in destination.

#include<stdio.h>

#include <string.h>

**int** main(){

**char** ch[20]=”easyprogramming”;

**char** ch2[20];

   strcpy(ch2,ch);

   printf("Value of second string is: %s",ch2);

**return** 0;

}

# **strcat()**

The strcat(first\_string, second\_string) function concatenates two strings and result is returned to first\_string.

#include<stdio.h>

#include <string.h>

**int** main(){

**char** ch[10]=”hello”;

**char** ch2[10]=”programmer”;

   strcat(ch,ch2);

   printf("Value of first string is: %s",ch);

**return** 0;

}

# **strcmp()**

The strcmp(first\_string, second\_string) function compares two string and returns 0 if both strings are equal.

#include<stdio.h>

#include <string.h>

**int** main(){

**char** str1[20],str2[20];

  printf("Enter 1st string: ");

  gets(str1);//reads string from console

  printf("Enter 2nd string: ");

  gets(str2);

**if**(strcmp(str1,str2)==0)

      printf("Strings are equal");

**else**

      printf("Strings are not equal");

**return** 0;

}

# **strrev()**

The strrev(string) function returns reverse of the given string.

#include<stdio.h>

#include <string.h>

**int** main(){

**char** str[20];

  printf("Enter string: ");

  gets(str);//reads string from console

  printf("String is: %s",str);

  printf("\nReverse String is: %s",strrev(str));

**return** 0;

}    **strlwr()**

The strlwr(string) function returns string characters in lowercase.

#include<stdio.h>

#include <string.h>

**int** main(){

**char** str[20];

  printf("Enter string: ");

  gets(str);//reads string from console

  printf("String is: %s",str);

  printf("\nLower String is: %s",strlwr(str));

**return** 0;

}

# **strupr()**

The strupr(string) function returns string characters in uppercase.

#include<stdio.h>

#include <string.h>

**int** main(){

**char** str[20];

  printf("Enter string: ");

  gets(str);//reads string from console

  printf("String is: %s",str);

  printf("\nUpper String is: %s",strupr(str));

**return** 0;

}

**C PROGRAMMING LANGUAGE TUTORIAL**

# **C Structure**

Structure in c is a user-defined data type that enable

s us to store the collection of different data types. Each element of a structure is called a member.

**struct** structure\_name

{

    data\_type member1;

    data\_type member2;

    .

    .

    data\_type memeberN;

};

**example to define a structure**

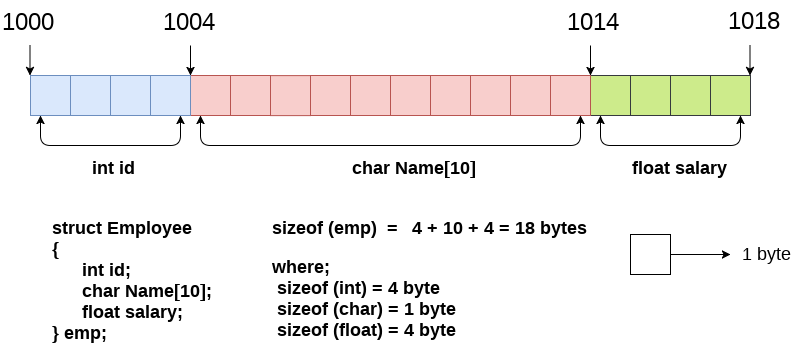
**struct** employee

{   **int** id;

**char** name[10];

**float** salary;

};



Example:

#include<stdio.h>

#include <string.h>

**struct** employee

{   **int** id;

**char** name[50];

**float** salary;

}e1,e2;  //declaring e1 and e2 variables for structure

**int** main( )

{

   //store first employee information

   e1.id=101;

   strcpy(e1.name, "Sonoo Jaiswal");//copying string into char array

   e1.salary=56000;

  //store second employee information

   e2.id=102;

   strcpy(e2.name, "James Bond");

   e2.salary=126000;

   //printing first employee information

   printf( "employee 1 id : %d\n", e1.id);

   printf( "employee 1 name : %s\n", e1.name);

   printf( "employee 1 salary : %f\n", e1.salary);

   //printing second employee information

   printf( "employee 2 id : %d\n", e2.id);

   printf( "employee 2 name : %s\n", e2.name);

   printf( "employee 2 salary : %f\n", e2.salary);

**return** 0;

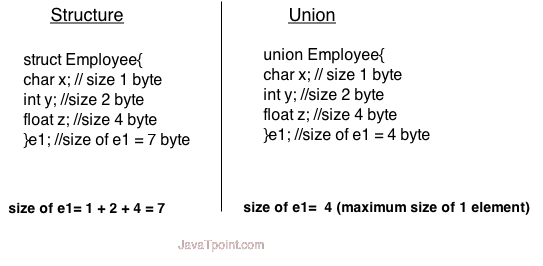
}

**C PROGRAMMING LANGUAGE TUTORIAL**

# **C Union**

**Union in c language** is *a user-defined data type* that is used to store the different type of elements.

At once, only one member of the union can occupy the memory. In other words, we can say that the size of the union in any instance is equal to the size of its largest element.



**union** union\_name

{

    data\_type member1;

    data\_type member2;

    .

    .

    data\_type memeberN;

};

Let's see the example to define union for an employee in c.

**union** employee

{   **int** id;

**char** name[50];

**float** salary;

};

**C Union example**

#include <stdio.h>

#include <string.h>

**union** employee

{   **int** id;

**char** name[50];

}e1;  //declaring e1 variable for union

**int** main( )

{

   //store first employee information

   e1.id=101;

   strcpy(e1.name, "Sonoo Jaiswal");//copying string into char array

   //printing first employee information

   printf( "employee 1 id : %d\n", e1.id);

   printf( "employee 1 name : %s\n", e1.name);

**return** 0;

}

**C PROGRAMMING LANGUAGE TUTORIAL**

**Most ask programming questions**

**fibonacci series**

In case of **fibonacci series**, next number is the sum of previous two numbers for example 0, 1, 1, 2, 3, 5, 8, 13, 21 etc.

#include<stdio.h>

int fibonacci(int);

int main(){

int n,sum;

printf("enter the number of fibonnaci series");

scanf("%d",&n);

fibonacci(n);

}

int fibonacci(int n){

int a=0,b=1,sum,i;

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

if(i<=1){

sum=i;

}

else{

sum=a+b;

a=b;

b=sum;

}

printf("%d",sum);

}

}

**prime number**

prime numbers can't be divided by other numbers than itself or 1. For example 2, 3, 5, 7, 11, 13, 17, 19, 23.... are the prime numbers.

#include <stdio.h>

int main()

{

int number, i;

printf("Enter a positive integer: ");

scanf("%d",&number);

printf("Factors of %d are: ", number);

for(i=1; i <= number; ++i)

{

if (number%i == 0)

{

printf("%d ",i);

}

}

return 0;

}

**Swapping**

Swapping means **interchanging**. If the program has two variables a and b where a = 4 and b = 5, after swapping them, a = 5, b = 4.

#include<stdio.h>

int main(){

int a,b,temp;

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

temp=a;

a=b;

b=temp;

printf("%d %d",a,b);

}

## Factorial

The factorial of a positive integer n, denoted by n!, is the product of all positive integers less than or equal to n: n!=n×(n-1)×(n-2)×(n-3)×⋯×3×2×1. For example, 5!=5×4×3×2×1=120.

#include<stdio.h>

int factorial(int);

int main(){

int n;

printf("enter the value ");

scanf("%d",&n);

factorial(n);

}

int factorial(int n){

int i,fact=1;

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

fact=fact\*i;

}

printf("%d",fact);

}

**palindrome number**

A palindrome number is a number **that** is **equal** to **its reverse.**

#include<stdio.h>

int palindrome(int);

int main(){

int n,o;

printf("enter the number ");

scanf("%d",&n);

palindrome(n);

}

int palindrome(int n){

int remainder,reverse=0,o;

o=n;

while(n!=0){

remainder=n%10;

reverse=reverse\*10+remainder;

n=n/10;

}

if(o==reverse){

printf("palindrome");

}

else{

printf("not palidrome");

}

}

**Armstrong number**

An armstrong number is**any number of n digits which is equal to the sum of nth power of digits in the number.**

#include<stdio.h>

#include<math.h>

int armstrong(int);

int main(){

int n;

printf("enter the number");

scanf("%d",&n);

armstrong(n);

}

int armstrong(int n){

int cube=0,sum=0,o;

n=o;

while(n!=0){

n=n%10;

sum+=n\*n\*n;

n=n/10;

}

if(sum==o){

printf("armstrong");

}

else

{

printf("not");

}

}

**Sum of digits**

#include<stdio.h>

int sumdigit(int);

int main(){

int n;

printf("enter the value ");

scanf("%d",&n);

sumdigit(n);

}

int sumdigit(int n){

int sum=0,digit;

while(n!=0){

digit=n%10;

sum+=digit;

n=n/10;

}

printf("%d",sum);

}

**Maximum value of an array**

#include<stdio.h>

int main(){

int i,j,n,arr1[20],arr2[20],max1,result;

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

printf("\n enter the first elements of array");

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

scanf("%d",&arr1[i]);

}

int max = arr1[0];

for(i=1;i<5;i++){

if (arr1[i] > max) {

max = arr1[i];

}

}

printf("the max value is %d",max);

printf("\n enter the second elements of array");

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

scanf("%d",&arr2[i]);

}

max1 = arr2[0];

for(i=1;i<5;i++){

if (arr2[i] > max1) {

max1 = arr2[i];

}

}

printf("the max value is %d",max1);

}