C++ Course

C++ is a general purpose, case-sensitive, free-form programming language that supports object-oriented, procedural and generic programming.

C++ is a middle-level language, as it encapsulates both high and low level language features.

C++ developed by **Bjarne Stroustrup** in the 1980s.

Usage of C++

* Window application
* Client-Server application
* Device drivers
* Embedded firmware etc

Note: 1. The cout object is defined inside the std namespace

2. iostream is preprocessor means run first when program start.

3. cout is used to print the output

4. cin is used to take input e.g: cin >> a , a is variable of any type.

**Data Types :**

Primitive : int , char , Boolean, float, long, double, wchar\_t, void

User Defined: Class, Union, Structure, Enum , Typeof

Derived : Function, Arrays, Objects, Pointers, Reference

Hello world:

#include <iostream>

using namespace std;

int main( ){

cout << “ Hello C++” ;

return 0;

}

**Variables :** A variable is a name given to a memory location. It is the basic unit of storage in a program.

Declaring a single variable :

type variable\_name;

e.x: int a;

Types of variables :

* Local Variables
* Instance Variables
* Static Variables

**Conditional :**

* 1. If- else
  2. Switch
  3. If –else if- else

**Loops:**

|  |  |
| --- | --- |
| 1. | **while loop** – First checks the condition, then executes the body. |
| 2. | **for loop**– firstly initializes, then, condition check, execute body, update. |
| 3. | **do-while** – firstly, execute the body then condition check |

  E.g : if-else

int main(){

int age ;

cout << "Enter your age : ";

cin>>age;

if(age >= 18){

cout << "You are eligible to vote";

}else{

cout << "You can't vote";

}

return 0;

}

1. For loop

int main(){

for ( int i = 0; i <10; i++)

{

cout << i << endl;

}

return 0;

}

1. While loop

int main(){

int a = 13;

while (a < 18)

{

cout << a;

a++;

}

return 0;

}

1. Do-while

int main(){

int a = 5; // initialization

do

{

cout << "Hello Bhai \n";

a++; // updation

} while (a< 4); // test

return 0;

}

**Functions :**

1. Standard Library Functions: Predefined in C++

2. User-defined Function: Created by users

Syntax:

returnType functionName (parameter1, parameter2,...) {

// function body

}

e.g :

int add(int a, int b){

return a + b;

}

e.g:

#include <iostream>

using namespace std;

int add(int a, int b){

return a + b;

}

int main(){

int a , b;

cout << "Enter a " << endl;

cin >> a;

cout << "Enter b " << endl;

cin >> b;

cout << "The sum of a ,b is: " << add(a,b); }

e.g: void means that the function returns nothing

void greet( ){

cout << "Hello Deepak Good evening";

}

int main(){

greet();

return 0;

}

e.g:

void print(int num) {

cout << num <<endl;

return ;

}

int add(int n1, int n2){

print(n1);

print(n2);

int sum = n1+n2;

return sum;

}

int main(){

int n1 = 12;

int n2 = 8;

cout<< add(n1,n2) << endl;

return 0;

}

**Function Prototype:**

In C++, the code of function declaration should be before the function call. However, if we want to define a function after the function call, we need to use the function prototype.

Syntax:

void add(int , int) function prototype

int main(){

void(5,3) calling the function before declaration

return 0;

}

int add(int a, int b){ function definition

cout << a + b ;

}

**Arrays :**

An array can be defined as a group or collection of similar kinds of elements or data items that are stored together in contiguous memory spaces.

All the memory locations are adjacent to each other, and the number of elements in an array is the size of the array.

e.g: Elements

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 5 | 10 | 6 | 9 | 8 | 3 |

index 0 1 2 3 4 5 6

Initialization and Declaration of an Array :

Syntax:

datatype arr\_ name[size ];

e.g: int club[10] ;

e.g: int arr[6 ] = {2,5,6,9,10,15} or

e.g: int arr[ ] = {2,5,6,9,10,15}

Types of array :

1. One dimensional
2. Two dimensional
3. Multi dimensional
4. One dimensional : -

e.g: int arr[2] = {2,8}

2. Two dimensional :-

e.g: int arr[2][3] = { {1,2,5} , {2,4,9} }

3. Multi-dimensional :-

e.g: int arr[2][3][2]= { {1,2,5} , {2,4,9} , {5,0,7}}

**Strings** : String is a sequence of characters as an object of the class. This class is called std:: string. String class stores the characters as a sequence of bytes with the functionality of allowing access to the single-byte character.

Syntax:

string s = “ “;

string dk = “Deepak Kumar”;

Operation on string :

1. getline(cin , str) : This function is used to store a stream of characters as entered by the user in the object memory.
2. push\_back( str) : This function is used to input a character at the end of the string.
3. pop\_back ( ) : Introduced from C++11(for strings), this function is used to delete the last character from the string.

String methods:

1. append( ): append the string at last of string
2. strcpy (chat \*str1, char \*str2) : copy str2 into str1
3. strcat (chat \*str1, char \*str2) : concate str1 , str2

Note : It is important to note that the strcat() function only accepts character arrays as its arguments. We can not use string objects in the function.

1. length ( ) : used to find the lengt h of string
2. resize(10) : used to resize of the string
3. shrink\_to\_fit( ) : The shrink\_to\_fit() function is used to decrease the string’s capacity to make it equal to the string’s minimum capacity. This function helps us save memory if we are sure that no additional characters are required.
4. capacity(): used to find the capacity of string e.g (length)

Iterator methods:

1. begin(): This function returns an iterator to the beginning of the string.
2. end():This function returns an iterator to the end of the string.
3. rbegin():This function returns a reverse iterator pointing at the end of the string.
4. rend():This function returns a reverse iterator pointing at beginning of the string.

e.g:

#include <iostream>

#include<cstring>

#include<string> //// for string class

using namespace std;

int main(){

string str = "geeksforgeeks";

// Declaring iterator

string::iterator itr1;

// Declaring reverse iterator

string::reverse\_iterator itr2;

cout << "The string using forward iterators is : ";

for (itr1 = str.begin(); itr1 != str.end(); itr1++)

cout << \*itr1 <<en

cout << "The string using reverse iterators is : ";

for (itr2 = str.rbegin(); itr2 != str.rend(); itr2++)

cout << \*itr2 <<endl;

return 0; }

Manipulating Functions:

1. copy (target char arr, length to be copied, starting index) :

This function copies the substring in the target character array mentioned in its arguments. It takes 3 arguments, target char array, length to be copied, and starting position in the string to start copying.

1. swap ( ) : used to swap two string.

e.g:

int main(){

string str1 = "geeksforgeeks is a good platform";

string str2 = "geeksforgeeks rocks";

// Declaring a char arr

char arr[50];

str1.copy(arr, 13, 0);

cout<< "New Copied character array is : " << endl;

cout<<arr;

return 0;

}

e.g:

int main( ){

string str1 = "geeksforgeeks is a good platform";

string str2 = "geeksforgeeks rocks";

str1.swap(str2);

cout << "The 1st string after swapping is : ";

cout << str1 << endl;

cout << "The 2nd string after swapping is : ";

cout << str2 << endl;

return 0;

}

**Pointers :**

Pointer is a variable in C++ that holds the address of another variable. They have [data type](https://beginnersbook.com/2017/08/cpp-data-types/) just like variables, for example an integer type pointer can hold the address of an integer variable and an character type pointer can hold the address of char variable.

**e.g:**

int \*x; // a pointer to integer

double \*x; // a pointer to double

float \*x; // a pointer to float

char \*ch // a pointer to a character

Dereference operator (\*):

The dereference operator (\*) helps us get the value that has been stored in a memory address.

Syntax:

type \*var\_name;

e.g: int \*pointer;

int\* pointer; // preferred syntax

Address-of operator (&) : The address of a variable can be obtained by preceding the name of a variable with an ampersand sign (&), known as address-of operator.

e.g:

foo = &myvar;

e.g 1:

// address of pointer

int main(){

int a = 10;

int b = 23;

cout<< "Memory address of a :"<<&a<<endl;

cout<< "Memory address of b :"<<&b<<endl;

return 0;

}

e.g 2:

int main(){

int a = 10;

int b = 23;

// assigning address of a ,b into pointer ptr1, ptr2

int \*ptr1 = &a;

int \*ptr2 = &b;

cout<< "Memory address of a :"<<&a<<endl;

cout<< "Memory address of b :"<<&b<<endl;

cout<< "Pointer a :"<<\*ptr1<<endl;

cout<< "Pointer b :"<<\*ptr2<<endl;

return 0;

}

e.g 3:

void myPointer() {

int a = 10;

int \*ptr;

ptr = &a; // store the address of variable a

cout << "Adress of vaiable a :" << &a<<endl;

cout << "Pointer of a i.e. pointVar:" << ptr <<endl;

// print the content of the address ptr points to

cout << "Content of the address pointed to by ptr (\*ptr) = " << \* ptr << endl;

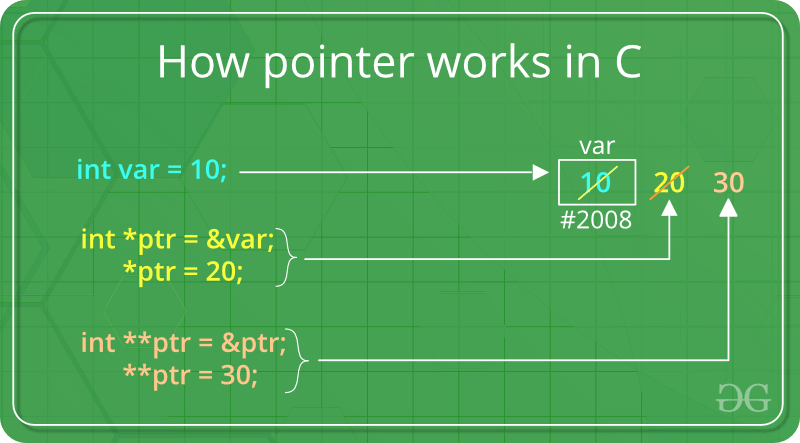
}

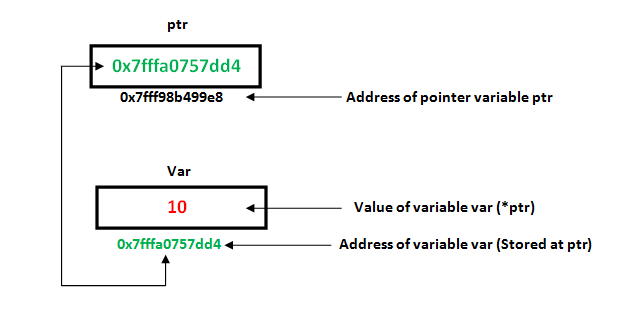
int main(){

myPointer();

return 0;

}





Note:

int var, \*varPoint;

// Wrong!

// varPoint is an address but var is not

varPoint = var;

// Wrong!

// &var is an address

// \*varPoint is the value stored in &var

\*varPoint = &var;

// Correct!

// varPoint is an address and so is &var

varPoint = &var;

// Correct!

// both \*varPoint and var are values

\*varPoint = var;

**Pointers Operations:**

1. ++a : value is first incremented and then used inside the expression.
2. a++ : In the Post-Increment, value is first used in an expression and then incremented.

e.g:

int main() {

int a = 5;

int \*ptr = &a;

cout <<"Memory addresss of a before increment : "<<ptr<<endl;

++ptr; // icreases by 4 beacuse of int datatype

cout <<"Memory addresss after increment of ptr : "<<ptr<<endl;

return 0;

}

1. -- a : In the Post-decrement, value is first used in an expression and then

decremented.

1. a-- : In the Post-decrement, value is first used in an expression and then

decremented.

e.g:

int main() {

int a = 5;

int \*ptr = &a;

cout <<"Memory addresss of a before decrement : "<<ptr<<endl;

--ptr; // decrease by 4 beacuse of int datatype

cout <<"Memory addresss after decrement of ptr : "<<ptr<<endl;

return 0;

}

Pointer to Pointer:

e.g:

int main() {

int a = 10;

int \*ptr = &a;

int \*\*q = &ptr;

cout<<\*q<<endl; // print address of ptr

cout<<\*\*q<<endl; // print value of a

return 0;

}