**Pointers in C++ | C++ Tutorials for Beginners #12**

In this series of our C++ tutorials, we will visualize pointers in the C++ language in this lecture. In our last lesson, we discussed break statements and continue statements in C++.

**Pointers in C++**

A pointer is a data type which holds the address of other data type. The “**&**” operator is called “**address off**" operator, and the "**\***” operator is called “**value at**” dereference operator. An example program for pointers is shown in figure 1.

Graphical user interface, text, application

Description automatically generated

***Figure 1: Pointer Program***

As shown in figure 1, at 1st line an integer variable “**a**” is initialized with the value “**3**". At the 2nd line, the address of integer variable "**a**” is assigned to the integer pointer variable “**b**". At the 3rd line, the address of the integer pointer variable "**b**” is printed. The output of the following program is shown in figure 2.

Text

Description automatically generated

***Figure 2: Pointer Program Output***

As shown in figure 2, the address of the integer pointer variable "**b**” is printed. The main thing to note here is that the address printed by the variable “**b**" is the address of integer variable "**a**” because we had assigned the address of variable “**a**” to the integer pointer variable “**b**". To clarify, we will print both variable "a" and variable "b" addresses, which are shown in figure 3.

Text

Description automatically generated

***Figure 3: Pointer Program Example 2***

As shown in figure 3, now we printed both variable “**a**” and variable “**b**” addresses. The output for the following program is shown in figure 4.

A picture containing text, electronics, close

Description automatically generated

***Figure 4: Pointer Program Example 2 Output***

As shown in figure 4, both variables "**a**” and “**b**” have the same addresses, but in actual, this is the address of the variable "**a**”, the variable “**b**" is just pointing to the address of the variable "**a**”.

To see the value of variable “**a**" using a pointer variable, we can use the "**\***" dereference operator. An example of the dereference operator program is shown in figure 5.

A screenshot of a computer

Description automatically generated with medium confidence

***Figure 5: Dereference Operator example***

As shown in figure 5, the value at address “**b**” is printed. The main thing to note here is that the value printed by the pointer variable “**b**” will be the value of variable “**a**” because the pointer variable “**b**" is pointing to the address of the variable "**a**”. The output for the following program is shown in figure 6.

A screenshot of a computer

Description automatically generated with medium confidence

***Figure 6: Dereference Operator Example***

**Pointer to Pointer**

Pointer to Pointer is a simple concept, in which we store the address of one Pointer to another pointer. An example program for Pointer to Pointer is shown in figure 7.

Text

Description automatically generated

***Figure 7: Pointer to Pointer Example Program***

As shown in figure 7, at the 1st line, the address of the pointer variable "**b**” is assigned to the pointer variable “**c**”. At 2nd line, the address of the pointer variable "**b**” is printed. At the 3rd line, the address of the pointer variable "**c**” is printed. At line 4th, the value at the pointer variable "**c**” is printed. At line 5th, the pointer variable "**c**” will be dereferenced two times, and it will print the value at pointer variable "**b**”. The output of the following program is shown in figure 2. The output for the following program is shown in figure 8.

Text

Description automatically generated

***Figure 8: Pointer to Pointer Example Program Output***

**Code as described/written in the video**

#include<iostream>

using namespace std;

int main(){

// What is a pointer? ----> Data type which holds the address of other data types

int a=3;

int\* b;

b = &a;

// & ---> (Address of) Operator

cout<<"The address of a is "<<&a<<endl;

cout<<"The address of a is "<<b<<endl;

// \* ---> (value at) Dereference operator

cout<<"The value at address b is "<<\*b<<endl;

// Pointer to pointer

int\*\* c = &b;

cout<<"The address of b is "<<&b<<endl;

cout<<"The address of b is "<<c<<endl;

cout<<"The value at address c is "<<\*c<<endl;

cout<<"The value at address value\_at(value\_at(c)) is "<<\*\*c<<endl;

return 0;

}

# Arrays & Pointers Arithmetic in C++ | C++ Tutorials for Beginners #13

In this tutorial, we will discuss arrays and pointer arithmetic in C++

#### What are Arrays in C++

* An array is a collection of items which are of the similar type stored in contiguous memory locations.
* Sometimes, a simple variable is not enough to hold all the data.
* For example, let’s say we want to store the marks of 2500 students; initializing 2500 different variable for this task is not feasible.
* To solve this problem, we can define an array with size 2500 that can hold the marks of all students.
* For example **int marks[2500];**

An example program for an array is shown in code snippet below.

int marks[] = {23, 45, 56, 89};

cout<<marks[0]<<endl;

cout<<marks[1]<<endl;

cout<<marks[2]<<endl;

cout<<marks[3]<<endl;

Copy

***Code Snippet 1: Array Program 1***

As shown in the code snippet, we initialized an array of size 4 in which we have stored marks of 4 students and then printed them one by one. The main point to note here is that array store data in continuous block form in the memory, and array indexes start from 0. Output for the following program is shown in figure 1.

Graphical user interface, text

Description automatically generated

***Figure 1: Array Program 1 Output***

Another example program to declare an array is shown in code snippet 2.

int mathMarks[4];

mathMarks[0] = 2278;

mathMarks[1] = 738;

mathMarks[2] = 378;

mathMarks[3] = 578;

cout<<"These are math marks"<<endl;

cout<<mathMarks[0]<<endl;

cout<<mathMarks[1]<<endl;

cout<<mathMarks[2]<<endl;

cout<<mathMarks[3]<<endl;

Copy

***Code Snippet 2: Array Program 2***

As shown in code snippet 2, we have declared an array of size 4 and then assigned values one by one to each index of the array. Output for the following program is shown in figure 2.

Text

Description automatically generated with medium confidence

***Figure 2: Array Program 2 Output***

To change the value at the specific index of an array, we can simply assign the value to that index. For example: “**marks[2] = 333**” can place the value “**333**” at the index “**2**” of the array. We can use loops to print the values of an array, instead of printing them one by one. An example program to print the value of the array with "for" loop is shown in code snippet 3.

for (int i = 0; i < 4; i++)

{

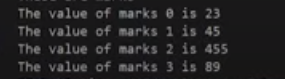
cout<<"The value of marks "<<i<<" is "<<marks[i]<<endl;

}

Copy

***Code Snippet 3: Array program with a loop***

As shown in code snippet 3, we initialized an integer variable “i" with the value 0 and set the running condition of the loop to the length of an array. In the loop body, each index number and the value at each number is being printed. Output for the following program is shown in figure 3.



***Figure 3: Array program with loop output***

#### Pointers and Arrays

Storing the address of an array into pointer is different than storing the address of a variable into the pointer because the name of the array is an address of the first index of an array. So to use ampersand "&" with the array name for assigning the address to a pointer is wrong.

* &Marks --> Wrong
* Marks --> address of the first block

An example program for storing the starting address of an array in the pointer is shown in code snippet 4.

int\* p = marks;

cout<<"The value of marks[0] is "<<\*p<<endl;

Copy

***Code Snippet 4: Pointer and Array Program***

As shown in code snippet 7, we have assigned the address of array “marks” to the pointer variable “\*p” and then printed the pointer “\*p”. The main thing to note here is that the value at the pointer “\*p” is the starting address of the array “marks”. The output for the following program is shown in figure 4.



***Figure 4: Pointer and Array Program Output***

As shown in figure 4, we have printed the value at pointer "\*p", and it has shown us the value of the first index of the array "marks" because the pointer was pointing at the first index of an array and the value at that index was "23". If we want to access the 2nd index of an array through the pointer, we can simply increment the pointer with 1. For example: "**\*(p+1)**" will give us the value of the 2nd index of an array. An example program to print the values of an array through the pointer is shown in code snippet 5.

int\* p = marks;

cout<<"The value of \*p is "<<\*p<<endl;

cout<<"The value of \*(p+1) is "<<\*(p+1)<<endl;

cout<<"The value of \*(p+2) is "<<\*(p+2)<<endl;

cout<<"The value of \*(p+3) is "<<\*(p+3)<<endl;

Copy

***Code Snippet 5: Pointer and Array Program 2***

As shown in code snippet 5, 1st we have printed the value at pointer “**\*p**”; 2nd we have printed the value at pointer “**\*(p+1)**”; 3rd we have printed the value at pointer “**\*(p+2)**”; 4th we have printed the value at pointer “**\*(p+3)**". This program will output the values at "0, 1, 2, 3" indices of an array "marks". The output of the following program is shown in figure 5.

Text

Description automatically generated

***Figure 5: Pointer and Array Program 2 Output***

#### Code as described/written in the video

#include<iostream>

using namespace std;

int main(){

// Array Example

int marks[] = {23, 45, 56, 89};

int mathMarks[4];

mathMarks[0] = 2278;

mathMarks[1] = 738;

mathMarks[2] = 378;

mathMarks[3] = 578;

cout<<"These are math marks"<<endl;

cout<<mathMarks[0]<<endl;

cout<<mathMarks[1]<<endl;

cout<<mathMarks[2]<<endl;

cout<<mathMarks[3]<<endl;

// You can change the value of an array

marks[2] = 455;

cout<<"These are marks"<<endl;

// cout<<marks[0]<<endl;

// cout<<marks[1]<<endl;

// cout<<marks[2]<<endl;

// cout<<marks[3]<<endl;

for (int i = 0; i < 4; i++)

{

cout<<"The value of marks "<<i<<" is "<<marks[i]<<endl;

}

// Quick quiz: do the same using while and do-while loops?

// Pointers and arrays

int\* p = marks;

cout<<\*(p++)<<endl;

cout<<\*(++p)<<endl;

// cout<<"The value of \*p is "<<\*p<<endl;

// cout<<"The value of \*(p+1) is "<<\*(p+1)<<endl;

// cout<<"The value of \*(p+2) is "<<\*(p+2)<<endl;

// cout<<"The value of \*(p+3) is "<<\*(p+3)<<endl;

return 0;

}

**Structures, Unions & Enums in C++ | C++ Tutorials for Beginners #14**

In this tutorial, we will discuss structures, unions & enums in C++

**Structures in C++**

The structure is a user-defined data type that is available in C++. Structures are used to combine different types of data types, just like an array is used to combine the same type of data types. An example program for creating a structure is shown in Code Snippet 1.

struct employee

{

/\* data \*/

int eId;

char favChar;

float salary;

};

Copy

***Code Snippet 1: Creating a Structure Program***

As shown in Code Snippet 1, we have created a structure with the name “employee”, in which we have declared three variables of different data types (eId, favchar, salary). As we have created a structure now we can create instances of our structure employee. An example program for creating instances of structure employees is shown in Code Snippet 2.

int main() {

struct employee harry;

harry.eId = 1;

harry.favChar = 'c';

harry.salary = 120000000;

cout<<"The value is "<<harry.eId<<endl;

cout<<"The value is "<<harry.favChar<<endl;

cout<<"The value is "<<harry.salary<<endl;

return 0;

}

Copy

***Code Snippet 2: Creating Structure instances***

As shown in Code Snippet 2, 1st we have created a structure variable “harry” of type “employee”, 2nd we have assigned values to (eId, favchar, salary) fields of the structure employee and at the end we have printed the value of “salary”.

Another way to create structure variables without using the keyword “struct” and the name of the struct is shown in Code Snippet 3.

typedef struct employee

{

/\* data \*/

int eId; //4

char favChar; //1

float salary; //4

} ep;

Copy

***Code Snippet 3: Creating Structure Program 2***

As shown in Code Snippet 3, we have used a keyword “**typedef**” before struct and after the closing bracket of structure, we have written “ep”. Now we can create structure variables without using the keyword “struct” and name of the struct. An example is shown in Code Snippet 4.

int main(){

ep harry;

struct employee shubham;

struct employee rohanDas;

harry.eId = 1;

harry.favChar = 'c';

harry.salary = 120000000;

cout<<"The value is "<<harry.eId<<endl;

cout<<"The value is "<<harry.favChar<<endl;

cout<<"The value is "<<harry.salary<<endl;

return 0;

}

Copy

***Code Snippet 4: Creating Structure instance 2***

As shown in Code Snippet 4, we have created a structure instance “harry” by just writing “ep” before it.

**Unions in C++**

Unions are similar to structures but they provide better memory management then structures.  Unions use shared memory so only 1 variable can be used at a time. An example program to create unions is shown in Code Snippet 5.

union money

{

/\* data \*/

int rice; //4

char car; //1

float pounds; //4

};

Copy

***Code Snippet 5: Creating Unions Program***

As shown in Code Snippet 5, we have created a union with the name “money” in which we have declared three variables of different data types (rice, car, pound). The main thing to note here is that:

* We can only use 1 variable at a time otherwise the compiler will give us a garbage value
* The compiler chooses the data type which has maximum memory for the allocation.

An example program for creating an instance of union money is shown in Code Snippet 6.

int main(){

union money m1;

m1.rice = 34;

cout<<m1.rice;

return 0;

}

Copy

***Code Snippet 6: Creating a Union Instance***

As shown in Code Snippet 6, 1st we have created a union variable “m1” of type “money”, 2nd we have assigned values to (rice) fields of the union money, and in the end, we have printed the value of “rice”. The main thing to note here is that once we have assigned a value to the union field “rice”, now we cannot use other fields of the union otherwise we will get garbage value. The output for the following program is shown in figure 1.



***Figure 1: Creating Union Instance Output***

**Enums in C++**

Enums are user-defined types which consist of named constants. Enums are used to make the program more readable. An example program for enums is shown in Code Snippet 8.

int main(){

enum Meal{ breakfast, lunch, dinner};

Meal m1 = lunch;

cout<<m1;

return 0;

}

Copy

***Code Snippet 7: Enums Program***

As shown in Code Snippet 7, 1st we have created an enum “Meal” in which we have stored three named constants (breakfast, lunch, dinner). 2nd we have assigned the value of “lunch” to the variable “m1” and at the end, we have printed “m1”. The main thing to note here is that (breakfast, lunch, dinner) are constants; the value for “breakfast” is “0”, the value for “lunch” is “1” and the value for “dinner” is “2”. The output for the following program is shown in figure 2.



***Figure 2: Enums Program Output***

**Code as described/written in the video**

#include<iostream>

using namespace std;

typedef struct employee

{

/\* data \*/

int eId; //4

char favChar; //1

float salary; //4

} ep;

union money

{

/\* data \*/

int rice; //4

char car; //1

float pounds; //4

};

int main(){

enum Meal{ breakfast, lunch, dinner};

Meal m1 = lunch;

cout<<(m1==2);

// cout<<breakfast;

// cout<<lunch;

// cout<<dinner;

// union money m1;

// m1.rice = 34;

// m1.car = 'c';

// cout<<m1.car;

// ep harry;

// struct employee shubham;

// struct employee rohanDas;

// harry.eId = 1;

// harry.favChar = 'c';

// harry.salary = 120000000;

// cout<<"The value is "<<harry.eId<<endl;

// cout<<"The value is "<<harry.favChar<<endl;

// cout<<"The value is "<<harry.salary<<endl;

return 0;

}

# Functions & Function Prototypes in C++ | C++ Tutorials for Beginners #15

In this tutorial, we will discuss functions and functions prototype in C++

#### Functions in C++

Functions are the main part of top-down structured programming. We break the code into small pieces and make functions of that code. Functions help us to reuse the code easily. An example program for the function is shown in Code Snippet 1.

int sum(int a, int b){

int c = a+b;

return c;

}

Copy

***Code Snippet 1: Function example***

As shown in Code Snippet 1, we created an integer function with the name of sum, which takes two parameters “int a” and “int b”. In the function, body addition is performed on the values of variable “a” and variable “b” and the result is stored in variable “c”. In the end, the value of variable “c” is returned to the function. We have seen how this function works now we will see how to pass values to the function parameters. An example program for passing the values to the function is shown in Code Snippet 2.

int main(){

int num1, num2;

cout<<"Enter first number"<<endl;

cin>>num1;

cout<<"Enter second number"<<endl;

cin>>num2;

cout<<"The sum is "<<sum(num1, num2);

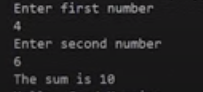
return 0;

}

Copy

***Code Snippet 2: Passing Value to Function Parameters***

As shown in Code Snippet 2, we have declared two integer variables “num1” and “num2”, we will take their input at run time. In the end, we called the “sum” function and passed both variables “num1” and “num2” into sum function. “sum” function will perform the addition and returns the value at the same location from where it was called. The output of the following program is shown in figure 1.



**Figure 1: Function Output**

#### Function Prototype in C++

The function prototype is the template of the function which tells the details of the function e.g(name, parameters) to the compiler. Function prototypes help us to define a function after the function call. An example of a function prototype is shown in Code Snippet 3.

// Function prototype

int sum(int a, int b);

Copy

**Code Snippet 3: Function Prototype**

As shown in Code Snippet 3, we have made a function prototype of the function “sum”, this function prototype will tell the compiler that the function “sum” is declared somewhere in the program which takes two integer parameters and returns an integer value. Some examples of acceptable and not acceptable prototypes are shown below:

* int sum(int a, int b); //Acceptable
* int sum(int a, b); // Not Acceptable
* int sum(int, int); //Acceptable

##### **Formal Parameters**

The variables which are declared in the function are called a formal parameter. For example, as shown in Code Snippet 1, the variables “a” and “b” are the formal parameters.

##### **Actual Parameters**

The values which are passed to the function are called actual parameters. For example, as shown in Code Snippet 2, the variables “num1” and “num2” are the actual parameters.

The function doesn't need to have parameters or it should return some value. An example of the void function is shown in Code Snippet 4.

void g(){

cout<<"\nHello, Good Morning";

}

Copy

**Code Snippet 4: Void Function**

As shown in Code Snippet 4, void as a return type means that this function will not return anything, and this function has no parameters. Whenever we will call this function it will print “Hello, Good Morning”

#### Code as described/written in the video

#include<iostream>

using namespace std;

// Function prototype

// type function-name (arguments);

// int sum(int a, int b); //--> Acceptable

// int sum(int a, b); //--> Not Acceptable

int sum(int, int); //--> Acceptable

// void g(void); //--> Acceptable

void g(); //--> Acceptable

int main(){

int num1, num2;

cout<<"Enter first number"<<endl;

cin>>num1;

cout<<"Enter second number"<<endl;

cin>>num2;

// num1 and num2 are actual parameters

cout<<"The sum is "<<sum(num1, num2);

g();

return 0;

}

int sum(int a, int b){

// Formal Parameters a and b will be taking values from actual parameters num1 and num2.

int c = a+b;

return c;

}

void g(){

cout<<"\nHello, Good Morning";

}

# Call by Value & Call by Reference in C++ | C++ Tutorials for Beginners #16

In this tutorial, we will discuss call by value and call by reference in C++

#### Call by Value in C++

Call by value is a method in C++ to pass the values to the function arguments. In case of call by value the copies of actual parameters are sent to the formal parameter, which means that if we change the values inside the function that will not affect the actual values. An example program for the call by value is shown in Code Snippet 1.

void swap(int a, int b){ //temp a b

int temp = a; //4 4 5

a = b; //4 5 5

b = temp; //4 5 4

}

Copy

**Code Snippet 1: Call by Value Swap Function**

As shown in Code Snippet 1, we created a swap function which is taking two parameters “int a” and “int b”. In function body values of the variable, “a” and “b” are swapped.  An example program is shown in Code Snippet 2, which calls the swap function and passes values to it.

int main(){

int x =4, y=5;

cout<<"The value of x is "<<x<<" and the value of y is "<<y<<endl;

swap(x, y);

cout<<"The value of x is "<<x<<" and the value of y is "<<y<<endl;

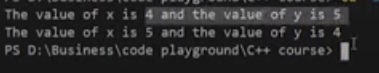
return 0;

}

Copy

**Code Snippet 2: Passing Values to Swap Function**

As shown in Code Snippet 2, we have initialized two integer variables “a” and “b” and printed their values. Then we called a “swap” function and passed values of variables “a” and “b” and again printed the values of variables “a” and “b”. The output for the following program is shown in figure 1.



**Figure 1: Call by Value Swap Function Output**

As shown in figure 3, the values of “a” and “b” are the same for both times they are printed. So the main point here is that when the call by value method is used it doesn’t change the actual values because copies of actual values are sent to the function.

#### Call by Pointer in C++

A call by the pointer is a method in C++ to pass the values to the function arguments. In the case of call by pointer, the address of actual parameters is sent to the formal parameter, which means that if we change the values inside the function that will affect the actual values. An example program for the call by reference is shown in Code Snippet 3.

// Call by reference using pointers

void swapPointer(int\* a, int\* b){ //temp a b

int temp = \*a; //4 4 5

\*a = \*b; //4 5 5

\*b = temp; //4 5 4

}

Copy

**Code Snippet 3: Call by Pointer Swap Function**

As shown in Code Snippet 3, we created a swap function which is taking two pointer parameters “int\* a” and “int\* b”. In function body values of pointer variables, “a” and “b” are swapped.  An example program is shown in Code Snippet 4, which calls the swap function and passes values to it.

int main(){

int x =4, y=5;

cout<<"The value of x is "<<x<<" and the value of y is "<<y<<endl;

swapPointer(&x, &y); //This will swap a and b using pointer reference

cout<<"The value of x is "<<x<<" and the value of y is "<<y<<endl;

return 0;

}

Copy

**Code Snippet 4: Passing Values to Call by Pointer Swap Function**

As shown in Code Snippet 4, we have initialized two integer variables “a” and “b” and printed their values. Then we called a “swap” function and passed addresses of variables “a” and “b” and again printed the values of variables “a” and “b”. The output for the following program is shown in figure 2.

Graphical user interface, text, application

Description automatically generated

**Figure 2: Call by Pointer Swap Function Output**

As shown in figure 2, the values of “a” and “b” are swapped when the swap function is called. So the main point here is that when the call by pointer method is used it changes the actual values because addresses of actual values are sent to the function.

#### Call by Reference in C++

Call by reference is a method in C++ to pass the values to the function arguments. In the case of call by reference, the reference of actual parameters is sent to the formal parameter, which means that if we change the values inside the function that will affect the actual values. An example program for a call by reference is shown in Code Snippet 5.

void swapReferenceVar(int &a, int &b){ //temp a b

int temp = a; //4 4 5

a = b; //4 5 5

b = temp; //4 5 4

}

Copy

**Code Snippet 5: Call by Reference Swap Function**

As shown in Code Snippet 5, we created a swap function that is taking reference of “int &a” and “int &b” as parameters. In function body values of variables, “a” and “b” are swapped.  An example program is shown in Code Snippet 6, which calls the swap function and passes values to it.

int main(){

int x =4, y=5;

cout<<"The value of x is "<<x<<" and the value of y is "<<y<<endl;

swapReferenceVar(x, y); //This will swap a and b using reference variables

cout<<"The value of x is "<<x<<" and the value of y is "<<y<<endl;

return 0;

}

Copy

**Code Snippet 6: Passing Values to Call by Reference Swap Function**

As shown in Code Snippet 6, we have initialized two integer variables “a” and “b” and printed their values. Then we called a “swap” function and passed variables “a” and “b” and again printed the values of variables “a” and “b”. The output for the following program is shown in figure 3.

A close up of a keyboard

Description automatically generated with medium confidence

**Figure 3: Call by Reference Swap Function Output**

As shown in figure 3, the values of “a” and “b” are swapped when the swap function is called. So the main point here is that when the call by reference method is used it changes the actual values because references of actual values are sent to the function.

#### Code as described/written in the video

#include<iostream>

using namespace std;

int sum(int a, int b){

int c = a + b;

return c;

}

// This will not swap a and b

void swap(int a, int b){ //temp a b

int temp = a; //4 4 5

a = b; //4 5 5

b = temp; //4 5 4

}

// Call by reference using pointers

void swapPointer(int\* a, int\* b){ //temp a b

int temp = \*a; //4 4 5

\*a = \*b; //4 5 5

\*b = temp; //4 5 4

}

// Call by reference using C++ reference Variables

// int &

void swapReferenceVar(int &a, int &b){ //temp a b

int temp = a; //4 4 5

a = b; //4 5 5

b = temp; //4 5 4

// return a;

}

int main(){

int x =4, y=5;

// cout<<"The sum of 4 and 5 is "<<sum(a, b);

cout<<"The value of x is "<<x<<" and the value of y is "<<y<<endl;

// swap(x, y); // This will not swap a and b

// swapPointer(&x, &y); //This will swap a and b using pointer reference

swapReferenceVar(x, y); //This will swap a and b using reference variables

// swapReferenceVar(x, y) = 766; //This will swap a and b using reference variables

cout<<"The value of x is "<<x<<" and the value of y is "<<y<<endl;

return 0;

}

# Inline Functions, Default Arguments & Constant Arguments in C++ | C++ Tutorials for Beginners #17

In this tutorial, we will discuss inline functions, default arguments, and constant arguments in C++

#### Inline Functions in C++

Inline functions are used to reduce the function call. When one function is being called multiply times in the program it increases the execution time, so inline function is used to reduce time and increase program efficiency. If the inline function is being used when the function is called, the inline function expands the whole function code at the point of a function call, instead of running the function. Inline functions are considered to be used when the function is small otherwise it will not perform well. Inline is not recommended when static variables are being used in the function. An example of an inline function is shown in Code Snippet 1.

inline int product(int a, int b){

return a\*b;

}

Copy

**Code Snippet 1: Inline function**

As shown in Code Snippet 1, 1st inline keyword is used to make the function inline. 2nd a product function is created which has two arguments and returns the product of them. Now we will call the product function multiple times in our main program which is shown in Code Snippet 2.

int main(){

int a, b;

cout<<"Enter the value of a and b"<<endl;

cin>>a>>b;

cout<<"The product of a and b is "<<product(a,b)<<endl;

cout<<"The product of a and b is "<<product(a,b)<<endl;

cout<<"The product of a and b is "<<product(a,b)<<endl;

cout<<"The product of a and b is "<<product(a,b)<<endl;

cout<<"The product of a and b is "<<product(a,b)<<endl;

cout<<"The product of a and b is "<<product(a,b)<<endl;

cout<<"The product of a and b is "<<product(a,b)<<endl;

cout<<"The product of a and b is "<<product(a,b)<<endl;

return 0;

}

Copy

**Code Snippet 2: Calling Inline Product Function**

As shown in Code Snippet 2, we called the product function multiple times. The main thing to note here is that the function will not run instead of it the function code will be copied at the place where the function is being called. This will increase the execution time of the program because the compiler doesn’t have to copy the values and get the return value again and again from the compiler. The output of the following program is shown in figure 1.

A picture containing text, electronics, keyboard, computer

Description automatically generated

**Figure 1: Inline Function Output**

#### Default Arguments in C++

Default arguments are those values which are used by the function if we don’t input our value. It is recommended to write default arguments after the other arguments. An example program for default arguments is shown in Code Snippet 3.

float moneyReceived(int currentMoney, float factor=1.04){

return currentMoney \* factor;

}

int main(){

int money = 100000;

cout<<"If you have "<<money<<" Rs in your bank account, you will recive "<<moneyReceived(money)<< "Rs after 1 year"<<endl;

cout<<"For VIP: If you have "<<money<<" Rs in your bank account, you will recive "<<moneyReceived(money, 1.1)<< " Rs after 1 year";

return 0;

}

Copy

**Code Snippet 3: Default Argument Example Program**

As shown in Code Snippet 3, we created a “moneyReceived” function which has two arguments “int currentMoney” and “float factor=1.04”. This function returns the product of “currentMoney” and “factor”. In our main function, we called “moneyReceived” function and passed one argument “money”. Again we called “moneyReceived” function and passed two arguments ”money” and “1.1”. The main thing to note here is that when we passed only one argument “money” to the function at that time the default value of the argument “factor” will be used. But when we passed both arguments then the default value will not be used. The output for the following program is shown in figure 2.



**Figure 2: Default Argument Example Program Output**

#### Constant Arguments in C++

Constant arguments are used when you don’t want your values to be changed or modified by the function. An example of constant arguments is shown in Code Snippet 4.

int strlen(const char \*p){

}

Copy

**Code Snippet 4: Constant Arguments Example**

As shown in Code Snippet 4, we created a “strlen” function which takes a constant argument “p”. As the argument is constant so its value won’t be modified.

#### Code as described/written in the video

#include<iostream>

using namespace std;

inline int product(int a, int b){

// Not recommended to use below lines with inline functions

// static int c=0; // This executes only once

// c = c + 1; // Next time this function is run, the value of c will be retained

return a\*b;

}

float moneyReceived(int currentMoney, float factor=1.04){

return currentMoney \* factor;

}

// int strlen(const char \*p){

// }

int main(){

int a, b;

// cout<<"Enter the value of a and b"<<endl;

// cin>>a>>b;

// cout<<"The product of a and b is "<<product(a,b)<<endl;

int money = 100000;

cout<<"If you have "<<money<<" Rs in your bank account, you will recive "<<moneyReceived(money)<< "Rs after 1 year"<<endl;

cout<<"For VIP: If you have "<<money<<" Rs in your bank account, you will recive "<<moneyReceived(money, 1.1)<< " Rs after 1 year";

return 0;

}

# Classes, Public and Private access modifiers in C++ | C++ Tutorials for Beginners #21

In this tutorial, we will discuss classes, public and private access modifiers in C++

#### Why use classes instead of structures

Classes and structures are somewhat the same but still, they have some differences. For example, we cannot hide data in structures which means that everything is public and can be accessed easily which is a major drawback of the structure because structures cannot be used where data security is a major concern. Another drawback of structures is that we cannot add functions in it.

#### Classes in C++

Classes are user-defined data-types and are a template for creating objects. Classes consist of variables and functions which are also called class members.

#### Public Access Modifier in C++

All the variables and functions declared under public access modifier will be available for everyone. They can be accessed both inside and outside the class. Dot (.) operator is used in the program to access public data members directly.

#### Private Access Modifier in C++

All the variables and functions declared under a private access modifier can only be used inside the class. They are not permissible to be used by any object or function outside the class.

An example program to demonstrate classes, public and private access modifiers are shown in Code Snippet 1.

class Employee

{

private:

int a, b, c;

public:

int d, e;

void setData(int a1, int b1, int c1); // Declaration

void getData(){

cout<<"The value of a is "<<a<<endl;

cout<<"The value of b is "<<b<<endl;

cout<<"The value of c is "<<c<<endl;

cout<<"The value of d is "<<d<<endl;

cout<<"The value of e is "<<e<<endl;

}

};

void Employee :: setData(int a1, int b1, int c1){

a = a1;

b = b1;

c = c1;

}

Copy

**Code Snippet 1: Class Program**

As shown in Code Snippet 1, 1st we created an “employee” class, 2nd three integer variables “int a”, “int b”, and “int c” were declared under the private access modifier, 3rd two integer variables “int d” and “int e” was declared under the public access modifiers, 4th “setData” function was declared, 5th “getData” function was defined and values of all the variables are printed. 6th “setData” function was defined outside the “employee” class by using a scope resolution operator; “setData” function is used to assign values to the private member of the class. An example to create the object of the class and use its class members is shown in Code Snippet 2.

int main(){

Employee harry;

harry.d = 34;

harry.e = 89;

harry.setData(1,2,4);

harry.getData();

return 0;

}

Copy

**Code Snippet 2: Creating Object Example**

As shown in Code Snippet 2, 1st we created an object “harry” of the class “employee”; 2nd we assigned values to “int d” and “int e” which are public class members. If we try to assign values to the private class member’s compiler will throw an error. 3rd we passed the values to the function “setData” and at the end, we called “getData” function which will print the values of all the variables. The output for the following program is shown in figure 1.

Graphical user interface, text

Description automatically generated

**Figure 1: Class Program Output**

As shown in figure 1, all the values of our data members are printed.

#### Code as described/written in the video

#include<iostream>

using namespace std;

class Employee

{

private:

int a, b, c;

public:

int d, e;

void setData(int a1, int b1, int c1); // Declaration

void getData(){

cout<<"The value of a is "<<a<<endl;

cout<<"The value of b is "<<b<<endl;

cout<<"The value of c is "<<c<<endl;

cout<<"The value of d is "<<d<<endl;

cout<<"The value of e is "<<e<<endl;

}

};

void Employee :: setData(int a1, int b1, int c1){

a = a1;

b = b1;

c = c1;

}

int main(){

Employee harry;

// harry.a = 134; -->This will throw error as a is private

harry.d = 34;

harry.e = 89;

harry.setData(1,2,4);

harry.getData();

return 0;

}

**OOPs Recap & Nesting of Member Functions in C++ | C++ Tutorials for Beginners #22**

In this tutorial, we will discuss the nesting of a member function in C++

**Object-Oriented programming Recap**

* Stroustrup initially named C++ language as C with classes because C++ language was almost the same as C language but they added a new concept of classes in it.
* Classes are the extension of structures in C language.
* Structures had limitations such as; members are public and no methods.
* Classes have some additional futures than structures such as; classes that can have methods and properties.
* Classes have a feature to make class members as public and private.
* In C++ objects can be declared along with class deceleration as shown in Code Snippet 1.

class Employee{

// Class definition

} harry, rohan, lovish;

Copy

***Code Snippet 1: Declaring Objects with Class Declaration***

**Nesting of Member Functions**

If one member function is called inside the other member function of the same class it is called nesting of a member function. A program to demonstrate the nesting of a member function is shown below.

class binary

{

private:

string s;

void chk\_bin(void);

public:

void read(void);

void ones\_compliment(void);

void display(void);

};

Copy

***Code Snippet 2: Binary Class***

As shown in Code Snippet 2, we created a binary class that has, “s” string variable and “chk\_bin” void function as private class members; and “read” void function, “ones\_compliment” void function, and “display” void function as public class members. The definitions of these functions are shown below.

void binary::read(void)

{

cout << "Enter a binary number" << endl;

cin >> s;

}

Copy

***Code Snippet 3: Read Function***

As shown in Code Snippet 3, we have created a “read” function. This function will take input from the user at runtime.

void binary::chk\_bin(void)

{

for (int i = 0; i < s.length(); i++)

{

if (s.at(i) != '0' && s.at(i) != '1')

{

cout << "Incorrect binary format" << endl;

exit(0);

}

}

}

Copy

***Code Snippet 4: Check Binary Function***

As shown in Code Snippet 4 we have created a “chk\_bin” function. This ”for” loop in the function will run till the length of the string and “if” condition in the body of the loop will check the whole string that if there are any values in the string other than “1” and “0”. If there are values other than “1” and “0” this function will output “Incorrect binary format”.

void binary::ones\_compliment(void)

{

chk\_bin();

for (int i = 0; i < s.length(); i++)

{

if (s.at(i) == '0')

{

s.at(i) = '1';

}

else

{

s.at(i) = '0';

}

}

}

Copy

***Code Snippet 5: One's Compliment***

As shown in Code Snippet 5, in the body of the “ones\_compliment” function; the “chk\_bin” function is called, and as we have discussed above that if one member function is called inside the other member function of the same class it is called **nesting of a member function**. The “for” loop inside the “ones\_compliment” functions runs till the length of the string and the “if” condition inside the loop replaces the number “0” with “1” and “1” with “0”.

void binary::display(void)

{

cout<<"Displaying your binary number"<<endl;

for (int i = 0; i < s.length(); i++)

{

cout << s.at(i);

}

cout<<endl;

}

Copy

***Code Snippet 6: Display Function***

As shown in Code Snippet 6, the “for” loop inside display function runs till the length of the string and prints each value of the sting.

int main()

{

binary b;

b.read();

// b.chk\_bin();

b.display();

b.ones\_compliment();

b.display();

return 0;

}

Copy

***Code Snippet 7: Main Function***

As shown in Code Snippet 7, we created an object “b” of the binary data type, and the functions “read”, “display”, “ones\_compliment”, and “display” are called. The main thing to note here is that the function ”chk\_bin” is the private access modifier of the class so we cannot access it directly by using the object, it can be only accessed inside the class or by the member function of the class.

**Code as described/written in the video**

// OOPs - Classes and objects

// C++ --> initially called --> C with classes by stroustroup

// class --> extension of structures (in C)

// structures had limitations

// - members are public

// - No methods

// classes --> structures + more

// classes --> can have methods and properties

// classes --> can make few members as private & few as public

// structures in C++ are typedefed

// you can declare objects along with the class declarion like this:

/\* class Employee{

// Class definition

} harry, rohan, lovish; \*/

// harry.salary = 8 makes no sense if salary is private

// Nesting of member functions

#include <iostream>

#include <string>

using namespace std;

class binary

{

private:

string s;

void chk\_bin(void);

public:

void read(void);

void ones\_compliment(void);

void display(void);

};

void binary::read(void)

{

cout << "Enter a binary number" << endl;

cin >> s;

}

void binary::chk\_bin(void)

{

for (int i = 0; i < s.length(); i++)

{

if (s.at(i) != '0' && s.at(i) != '1')

{

cout << "Incorrect binary format" << endl;

exit(0);

}

}

}

void binary::ones\_compliment(void)

{

chk\_bin();

for (int i = 0; i < s.length(); i++)

{

if (s.at(i) == '0')

{

s.at(i) = '1';

}

else

{

s.at(i) = '0';

}

}

}

void binary::display(void)

{

cout<<"Displaying your binary number"<<endl;

for (int i = 0; i < s.length(); i++)

{

cout << s.at(i);

}

cout<<endl;

}

int main()

{

binary b;

b.read();

// b.chk\_bin();

b.display();

b.ones\_compliment();

b.display();

return 0;

}

# C++ Objects Memory Allocation & using Arrays in Classes | C++ Tutorials for Beginners #23

In this tutorial, we will discuss objects memory allocation and using arrays in C++

#### Objects Memory Allocation in C++

The way memory is allocated to variables and functions of the class is different even though they both are from the same class.

The memory is only allocated to the variables of the class when the object is created. The memory is not allocated to the variables when the class is declared. At the same time, single variables can have different values for different objects, so every object has an individual copy of all the variables of the class. But the memory is allocated to the function only once when the class is declared. So the objects don’t have individual copies of functions only one copy is shared among each object.

#### Arrays in Classes

Arrays are used to store multiple values of the same type. An array is very helpful when multiple variables are required, instead of making multiple variables one array can be used which can store multiple values. Array stores data in sequential order. An example program to demonstrate the use of arrays in classes is shown below.

class Shop

{

int itemId[100];

int itemPrice[100];

int counter;

public:

void initCounter(void) { counter = 0; }

void setPrice(void);

void displayPrice(void);

};

Copy

**Code Snippet 1: Shop Class**

As shown in Code Snippet 1, we created a shop class which has, “itemId[100]” and “itemPrice” as integer array variable and “counter” variable as private class members; and “initCounter” void function, “setPrice” void function, and “displayPrice” void function as public class members. The definitions of these functions are shown below.

void Shop ::setPrice(void)

{

cout << "Enter Id of your item no " << counter + 1 << endl;

cin >> itemId[counter];

cout << "Enter Price of your item" << endl;

cin >> itemPrice[counter];

counter++;

}

Copy

**Code Snippet 2: Set Price Function**

As shown in Code Snippet 2, we have created a “setPrice” function. This function will take input for “itemId” and “ItemPrice” from the user at runtime. The value of the counter will be incremented by one every time this function will run.

void Shop ::displayPrice(void)

{

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

{

cout << "The Price of item with Id " << itemId[i] << " is " << itemPrice[i] << endl;

}

}

Copy

**Code Snippet 3: Display Price Function**

As shown in Code Snippet 3, the “for” loop inside the “displayPrice” function runs till the length of the counter and prints values of the array “itemId” and “ItemPrice”.

int main()

{

Shop dukaan;

dukaan.initCounter();

dukaan.setPrice();

dukaan.setPrice();

dukaan.setPrice();

dukaan.displayPrice();

return 0;

}

Copy

**Code Snippet 4: Main Function**

As shown in Code Snippet 4, we created an object “dukaan” of the shop data type, and the functions “initCounter” is called. The function “setPrice” is called three times. Loops can also be used to call the function multiple times. The “displayPrice” function is also called in the main function. The output of the following program is shown in figure 1.

Text

Description automatically generated with medium confidence

**Figure 1: Program Output**

As shown in figure 1, for the item 1 we entered the ID “1001” and price “12”; for the item 2 we entered the ID “1002” and price “23”; for the item 3 we entered the ID “1003” and price “34”. The Output of the program has displayed the ID and the price of each item.

#### Code as described/written in the video

#include <iostream>

using namespace std;

class Shop

{

int itemId[100];

int itemPrice[100];

int counter;

public:

void initCounter(void) { counter = 0; }

void setPrice(void);

void displayPrice(void);

};

void Shop ::setPrice(void)

{

cout << "Enter Id of your item no " << counter + 1 << endl;

cin >> itemId[counter];

cout << "Enter Price of your item" << endl;

cin >> itemPrice[counter];

counter++;

}

void Shop ::displayPrice(void)

{

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

{

cout << "The Price of item with Id " << itemId[i] << " is " << itemPrice[i] << endl;

}

}

int main()

{

Shop dukaan;

dukaan.initCounter();

dukaan.setPrice();

dukaan.setPrice();

dukaan.setPrice();

dukaan.displayPrice();

return 0;

}

**Static Data Members & Methods in C++ OOPS | C++ Tutorials for Beginners #24**

In this tutorial, we will discuss static data members and methods in C++

**Static Data Members in C++**

When a static data member is created, there is only a single copy of the data member which is shared between all the objects of the class. As we have discussed in our previous lecture that if the data members are not static then every object has an individual copy of the data member and it is not shared.

**Static Methods in C++**

When a static method is created, they become independent of any object and class. Static methods can only access static data members and static methods. Static methods can only be accessed using the scope resolution operator. An example program is shown below to demonstrate static data members and static methods in C++.

class Employee

{

int id;

static int count;

public:

void setData(void)

{

cout << "Enter the id" << endl;

cin >> id;

count++;

}

void getData(void)

{

cout << "The id of this employee is " << id << " and this is employee number " << count << endl;

}

static void getCount(void){

// cout<<id; // throws an error

cout<<"The value of count is "<<count<<endl;

}

};

Copy

***Code Snippet 1: Employee Class***

As shown in Code Snippet 1, we created an employee class that has integer “id” variable and “count” static integer variable as private class members; and “setData” void function, “getData” void function, and “getCount” static void function as public class members. These functions are explained below.

We have defined a “setData” function. This function will take input for “id” from the user at runtime and increment in the count. The value of the counter will be incremented by one every time this function will run.

We have defined a “getData” function. This function will print the values of the variables “id” and “count”.

We have defined a static “getCount” function. This function will print the value of the variable count”. The main thing to note here is that “getCount” function is static, so if we try to access any data members or member functions which are not static the compiler will throw an error.

// Count is the static data member of class Employee

int Employee::count; // Default value is 0

int main()

{

Employee harry, rohan, lovish;

// harry.id = 1;

// harry.count=1; // cannot do this as id and count are private

harry.setData();

harry.getData();

Employee::getCount();

rohan.setData();

rohan.getData();

Employee::getCount();

lovish.setData();

lovish.getData();

Employee::getCount();

return 0;

}

Copy

***Code Snippet 2: main Program***

As shown in Code Snippet 2:

* The count variable is declared whose default value is “0”.
* Then we created objects “harry”, “rohan”, and “lovish” of the employee data type
* The functions “setData”, “getData” are called by the object “harry”, the function “getCount” is called by using class name and scope resolution operator because it is a static method.
* The functions “setData”, “getData” are called by the object “rohan”, the function “getCount” is called by using class name and scope resolution operator because it is a static method.
* The functions “setData”, “getData” are called by the object “lovish”, the function “getCount” is called by using class name and scope resolution operator because it is a static method.

The output of the following program is shown in figures 1 and 2.

Graphical user interface, text

Description automatically generated

***Figure 1: Program Output 1***

***Text

Description automatically generated***

***Figure 2: Program Output 2***

As shown in figures 1 and 2, for the “harry” object we entered the ID “1”; for the “rohan” object we entered the ID “2”; and for the “lovish” object we entered the ID “3”. The Output of the program has displayed the ID and the count of each employee.

**Code as described/written in the video**

#include <iostream>

using namespace std;

class Employee

{

int id;

static int count;

public:

void setData(void)

{

cout << "Enter the id" << endl;

cin >> id;

count++;

}

void getData(void)

{

cout << "The id of this employee is " << id << " and this is employee number " << count << endl;

}

static void getCount(void){

// cout<<id; // throws an error

cout<<"The value of count is "<<count<<endl;

}

};

// Count is the static data member of class Employee

int Employee::count; // Default value is 0

int main()

{

Employee harry, rohan, lovish;

// harry.id = 1;

// harry.count=1; // cannot do this as id and count are private

harry.setData();

harry.getData();

Employee::getCount();

rohan.setData();

rohan.getData();

Employee::getCount();

lovish.setData();

lovish.getData();

Employee::getCount();

return 0;

}

**Array of Objects & Passing Objects as Function Arguments in C++ | C++ Tutorials for Beginners #25**

In this tutorial, we will discuss an array of objects and passing objects as a function arguments in C++

**An array of Objects in C++**

An array of objects is declared the same as any other data-type array.  An array of objects consists of class objects as its elements. If the array consists of class objects it is called an array of objects. An example program to demonstrate the concept of an array of objects is shown below.

class Employee

{

int id;

int salary;

public:

void setId(void)

{

salary = 122;

cout << "Enter the id of employee" << endl;

cin >> id;

}

void getId(void)

{

cout << "The id of this employee is " << id << endl;

}

};

Copy

***Code Snippet 1: Employee Class***

As shown in Code Snippet 1, we created an employee class that has integer “id” variable and “salary” integer variable as private class members; and “setId” void function, “getId” void function as public class members. These functions are explained below.

We have defined a “setId” function. In this function, the “salary” variable is assigned by the value “122” and the function will take input for “id” from the user at runtime. We have defined a “getId” function. This function will print the values of the variables “id”.

int main()

{

Employee fb[4];

for (int i = 0; i < 4; i++)

{

fb[i].setId();

fb[i].getId();

}

return 0;

}

Copy

***Code Snippet 2: main program***

As shown in Code Snippet 2, we created an array “fb” of size “4” which is of employee data-type. The “for” loop is used to run “setId” and “getId” functions till the size of an array. The main thing to note here is that the objects can also be created individually but it is more convenient to use an array if too many objects are to be created. The output of the following program is shown in figure 1.

Text

Description automatically generated

***Figure 1: Employee Program Output***

As shown in figure 1. As we input the Id for an employee it gives us the output of the employee Id.

**Passing Object as Function Argument**

Objects can be passed as function arguments. This is useful when we want to assign the values of a passed object to the current object. An example program to demonstrate the concept of passing an object as a function argument is shown below.

class complex{

int a;

int b;

public:

void setData(int v1, int v2){

a = v1;

b = v2;

}

void setDataBySum(complex o1, complex o2){

a = o1.a + o2.a;

b = o1.b + o2.b;

}

void printNumber(){

cout<<"Your complex number is "<<a<<" + "<<b<<"i"<<endl;

}

};

Copy

***Code Snippet 3: Complex Class***

As shown in Code Snippet 3, we created a complex class that has integer “a” variable and “b” integer variable as private class members; and “setData” void function, “setDataBySum” void function, and “printNumber” void function as public class members. These functions are explained below.

We have defined a “setData” function. In this function the values are assigned to the variables “a” and “b” because they are private data members of the class and values cannot be assigned directly. We have defined a “setDataBySum” function. In this function, the values of two objects are added and then assigned to the variables “a” and “b”. We have defined a “printNumber” function. In this function, the values of the variable “a” and “b” are being printed.

int main(){

complex c1, c2, c3;

c1.setData(1, 2);

c1.printNumber();

c2.setData(3, 4);

c2.printNumber();

c3.setDataBySum(c1, c2);

c3.printNumber();

return 0;

}

Copy

***Code Snippet 4: main program 2***

As shown in Code Snippet 4:

* We have created object “c1”, “c2”, and”c3” of complex data-type.
* The object “c1” calls the “setData” and “printNumber” functions.
* The object “c2” calls the “setData” and “printNumber” functions.
* The object “c3” calls the “setDataBySum” and “printNumber” functions.

The output of the following program is shown in figure 2.



***Figure 2: Complex Program Output***

**Code as described/written in the video**

#include <iostream>

using namespace std;

class Employee

{

int id;

int salary;

public:

void setId(void)

{

salary = 122;

cout << "Enter the id of employee" << endl;

cin >> id;

}

void getId(void)

{

cout << "The id of this employee is " << id << endl;

}

};

int main()

{

// Employee harry, rohan, lovish, shruti;

// harry.setId();

// harry.getId();

Employee fb[4];

for (int i = 0; i < 4; i++)

{

fb[i].setId();

fb[i].getId();

}

return 0;

}

Copy

**Code 25b as described/written in the video**

#include<iostream>

using namespace std;

class complex{

int a;

int b;

public:

void setData(int v1, int v2){

a = v1;

b = v2;

}

void setDataBySum(complex o1, complex o2){

a = o1.a + o2.a;

b = o1.b + o2.b;

}

void printNumber(){

cout<<"Your complex number is "<<a<<" + "<<b<<"i"<<endl;

}

};

int main(){

complex c1, c2, c3;

c1.setData(1, 2);

c1.printNumber();

c2.setData(3, 4);

c2.printNumber();

c3.setDataBySum(c1, c2);

c3.printNumber();

return 0;

}

**Friend Functions in C++ | C++ Tutorials for Beginners #26**

In this tutorial, we will discuss friend function in C++

**Friend Function in C++**

Friend functions are those functions that have the right to access the private data members of class even though they are not defined inside the class. It is necessary to write the prototype of the friend function. One main thing to note here is that if we have written the prototype for the friend function in the class it will not make that function a member of the class. An example program to demonstrate the concept of friend function is shown below.

class Complex{

int a, b;

friend Complex sumComplex(Complex o1, Complex o2);

public:

void setNumber(int n1, int n2){

a = n1;

b = n2;

}

// Below line means that non member - sumComplex funtion is allowed to do anything with my private parts (members)

void printNumber(){

cout<<"Your number is "<<a<<" + "<<b<<"i"<<endl;

}

};

Complex sumComplex(Complex o1, Complex o2){

Complex o3;

o3.setNumber((o1.a + o2.a), (o1.b+o2.b))

;

return o3;

}

Copy

***Code Snippet 1: Complex Class***

As shown in Code Snippet 1, we created a complex class that has integer “a” variable and “b” integer variable as private class members; and “setNumber” void function, “printNumber” void function as public class members. The “sumComplex” friend function prototype is written as well in the complex class. These functions are explained below.

We have defined a “setNumber” function. In this function the values are assigned to the variables “a” and “b” because they are private data members of the class and values cannot be assigned directly. We have defined a “printNumber” function. In this function, the values of the variable “a” and “b” are being printed. We have defined a “sumComplex” friend function. In this function, the object “o3” is created which calls the “setNumber” function and passes the values of two objects after performing addition on them.

int main(){

Complex c1, c2, sum;

c1.setNumber(1, 4);

c1.printNumber();

c2.setNumber(5, 8);

c2.printNumber();

sum = sumComplex(c1, c2);

sum.printNumber();

return 0;

}

Copy

***Code Snippet 2: main Program***

As shown in Code Snippet 2:

* We have created object “c1”, “c2”, and”sum” of complex data-type.
* The object “c1” calls the “setNumber” and “printNumber” functions.
* The object “c2” calls the “setNumber” and “printNumber” functions.
* The function “sumComplex” is called and the values are assigned to the “sum”.
* The object “sum” calls the “printNumber” functions.

The output of the following program is shown in figure 1.

Graphical user interface, text, application, chat or text message

Description automatically generated

***Figure 1: Complex Program Output***

As shown in figure 1, the output of the complex number program is printed.

**Properties of Friend Function**

* Not in the scope of the class
* Since it is not in the scope of the class, it cannot be called from the object of that class, for example, **sumComplex()**is invalid
* A friend function can be invoked without the help of any object
* Usually contain objects as arguments
* Can be declared under the public or private access modifier, it will not make any difference
* It cannot access the members directly by their names, it needs (object\_name.member\_name) to access any member.

**Code as described/written in the video**

#include<iostream>

using namespace std;

// 1 + 4i

// 5 + 8i

// -------

// 6 + 12i

class Complex{

int a, b;

friend Complex sumComplex(Complex o1, Complex o2);

public:

void setNumber(int n1, int n2){

a = n1;

b = n2;

}

// Below line means that non member - sumComplex funtion is allowed to do anything with my private parts (members)

void printNumber(){

cout<<"Your number is "<<a<<" + "<<b<<"i"<<endl;

}

};

Complex sumComplex(Complex o1, Complex o2){

Complex o3;

o3.setNumber((o1.a + o2.a), (o1.b+o2.b))

;

return o3;

}

int main(){

Complex c1, c2, sum;

c1.setNumber(1, 4);

c1.printNumber();

c2.setNumber(5, 8);

c2.printNumber();

sum = sumComplex(c1, c2);

sum.printNumber();

return 0;

}

/\* Properties of friend functions

1. Not in the scope of class

2. since it is not in the scope of the class, it cannot be called from the object of that class. c1.sumComplex() == Invalid

3. Can be invoked without the help of any object

4. Usually contans the objects as arguments

5. Can be declared inside public or private section of the class

6. It cannot access the members directly by their names and need object\_name.member\_name to access any member.

\*/

# Friend Classes & Member Friend Functions in C++ | C++ Tutorials for Beginners #27

In this tutorial, we will discuss friend classes and member friend functions in C++

#### Member Friend Functions in C++

Friend functions are those functions that have the access to private members of the class in which they are declared. The main thing to note here is that only that function can access the member function which is made a friend of the other class. An example of the friend function is shown below.

class Complex

{

int a, b;

// Individually declaring functions as friends

friend int Calculator ::sumRealComplex(Complex, Complex);

friend int Calculator ::sumCompComplex(Complex, Complex);

public:

void setNumber(int n1, int n2)

{

a = n1;

b = n2;a

}

void printNumber()

{

cout << "Your number is " << a << " + " << b << "i" << endl;

}

};

int Calculator ::sumRealComplex(Complex o1, Complex o2)

{

return (o1.a + o2.a);

}

int Calculator ::sumCompComplex(Complex o1, Complex o2)

{

return (o1.b + o2.b);

}

Copy

**Code Snippet 1: Friend function example**

As shown in a code snippet 1, a complex class is created which consists of two friend functions “sumRealComplex” and “sumCompComplex” of the calculator class. The main thing to note here is that “sumRealComplex” and “sumCompComplex” are the friend functions of complex class so they can access all the private members of the complex class.

#### Friend Classes in C++

Friend classes are those classes that have permission to access private members of the class in which they are declared. The main thing to note here is that if the class is made friend of another class then it can access all the private members of that class. An example program to demonstrate friend classes in C++ is shown below.

// Forward declaration

class Complex;

class Calculator

{

public:

int add(int a, int b)

{

return (a + b);

}

int sumRealComplex(Complex, Complex);

int sumCompComplex(Complex, Complex);

};

Copy

**Code Snippet 2: Calculator Class**

As shown in code snippet 2, a complex class is declared at the top which is known as forward declaration. Forward declaration hints to the compiler that this class is declared somewhere forward in the code. After that calculator class is defined this consists of three public member functions, “add”, “sumRealComplex”, and “sumCompComplex”. The “add” function will add the values of “a” and “b” and return the value. The “sumRealComplex” and “sumCompComplex” are taking two objects of the complex class. The code for the complex class is shown below.

class Complex

{

int a, b;

// Individually declaring functions as friends

// friend int Calculator ::sumRealComplex(Complex, Complex);

// friend int Calculator ::sumCompComplex(Complex, Complex);

// Aliter: Declaring the entire calculator class as friend

friend class Calculator;

public:

void setNumber(int n1, int n2)

{

a = n1;

b = n2;a

}

void printNumber()

{

cout << "Your number is " << a << " + " << b << "i" << endl;

}

};

int Calculator ::sumRealComplex(Complex o1, Complex o2)

{

return (o1.a + o2.a);

}

int Calculator ::sumCompComplex(Complex o1, Complex o2)

{

return (o1.b + o2.b);

}

Copy

**Code Snippet 3: Complex Class**

As shown in code snippet 3, a complex class is defined which consists of, two private data members “a” and “b”, and two public member functions “setNumber” and “printNumber”. The function “setNumber” will assign the values to the variables “a” and “b”. The function “printNumber” will print the values of the variables “a” and “b”. Two functions “sumRealComplex” and “sumCompComplex” are defined at the end. The function “sumRealComplex” will add the real values and the function “sumCompComplex” will add the complex value. The main program is shown below.

int main()

{

Complex o1, o2;

o1.setNumber(1, 4);

o2.setNumber(5, 7);

Calculator calc;

int res = calc.sumRealComplex(o1, o2);

cout << "The sum of real part of o1 and o2 is " << res << endl;

int resc = calc.sumCompComplex(o1, o2);

cout << "The sum of complex part of o1 and o2 is " << resc << endl;

return 0;

}

Copy

**Code snippet 4: Main Program**

As shown in code snippet 4, 1st two objects “o1” and “o2” of the “complex” data type are declared. 2nd “setNumber” function is called with the “o1” and “o2” objects and the values are passed. 3rd object “calc” of the calculator data type is declared. 4th “sumRealComplex” function is called by the “calc” object and the object “o1” and “o2” are passed to it. 5th “sumCompComplex” function is called by the “calc” object and the object “o1” and “o2” are passed to it. The output of the following program is shown in figure 1.

**More on C++ Friend Functions (Examples & Explanation) | C++ Tutorials for Beginners #28**

In this tutorial, we will discuss more on friend functions in C++ with examples

**Friend Functions in C++**

As we have already discussed in previous lectures friend functions are those functions that can access the private data members of the other class. An example program to demonstrate friend functions in C++ is shown below.

**Friend Function Example 1**

class Y;

class X{

int data;

public:

void setValue(int value){

data = value;

}

friend void add(X, Y);

};

class Y{

int num;

public:

void setValue(int value){

num = value;

}

friend void add(X, Y);

};

void add(X o1, Y o2){

cout<<"Summing data of X and Y objects gives me "<< o1.data + o2.num;

}

Copy

**Code Snippet 1: Friend Function Example 1**

As shown in a code snippet 1,

* 1st class “Y” is declared at the top which is known as forward declaration to let the compiler know that this class is defined somewhere in the program.
* 2nd class “X” is defined which consists of private data member “data” and public member function “setValue” which assigns the value to the private data member “data”. At the end friend function “add” is declared.
* 3rd class “Y” is defined which consists of private data member “num” and public member function “setValue” which assigns the value to the private data member “num”. At the end friend function “add” is declared.
* 4th function “add” is defined which add the value of the objects of class “X” and “Y” and print it.

The main program is shown in Code Snippet 2.

int main(){

X a1;

a1.setValue(3);

Y b1;

b1.setValue(15);

add(a1, b1);

return 0;

}

Copy

**Code Snippet 2: Main Program**

As shown in Code Snippet 2,

* 1st object “a1” of the data type “X” is declared
* 2nd function “setValue” is called by the object “a1” and the value “3” is passed
* 3rd object “b1” of the data type “Y” is declared
* 4th function “setValue” is called by the object “b1” and the value “15” is passed
* 5th function “add” is called and the objects “a1” and “b1” are passed to it. The function “add” will add the values of both objects and print them.

The output of the following program is shown in figure 1.



Figure 1: Program Output 1

As shown in figure 1, the sum of both values is shown which is “18”.

**Friend Function Example 2**

class c2;

class c1{

int val1;

friend void exchange(c1 & , c2 &);

public:

void indata(int a){

val1 = a;

}

void display(void){

cout<< val1 <<endl;

}

};

class c2{

int val2;

friend void exchange(c1 &, c2 &);

public:

void indata(int a){

val2 = a;

}

void display(void){

cout<< val2 <<endl;

}

};

void exchange(c1 &x, c2 &y){

int tmp = x.val1;

x.val1 = y.val2;

y.val2 = tmp;

}

Copy

**Code Snippet 3: Friend Function Example 2**

As shown in a code snippet 3,

* 1st class “c2” is declared at the top which is known as forward declaration to let the compiler know that this class is defined somewhere in the program.
* 2nd class “c1” is defined which consists of private data member “val1” and friend function “exchange” which takes reference variables “c1” and “c2” as parameters. The public member function “indata” is defined which assigns the value to the private data member “val1” and the function “display” prints the value of the data member “val1”.
* 3rd class “c2” is defined which consists of private data member “val2” and friend function “exchange” which takes reference variables “c1” and “c2” as parameters. The public member function “indata” is defined which assigns the value to the private data member “val2” and the function “display” prints the value of the data member “val2”.
* 4th function “exchange” is defined which swap the values.

The main program is shown in Code Snippet 4.

int main(){

c1 oc1;

c2 oc2;

oc1.indata(34);

oc2.indata(67);

exchange(oc1, oc2);

cout<<"The value of c1 after exchanging becomes: ";

oc1.display();

cout<<"The value of c2 after exchanging becomes: ";

oc2.display();

return 0;

}

Copy

**Code Snippet 4: Main program**

As shown in Code Snippet 4,

* 1st object “oc1” of the data type “c1” is declared
* 2nd object “oc2” of the data type “c2” is declared
* 3rd function “indata” is called by the object “oc1” and the value “34” is passed
* 4th function “indata” is called by the object “oc2” and the value “67” is passed
* 5th function “exchange” is called and the objects “oc1” and “oc2” are passed to it. The function “exchange” will swap both values and
* 6th function “display” is called by the objects “oc1” and “oc2” which will print their values.

The output of the following program is shown in figure 2.



**Figure 2:**Program Output 2

As shown in figure 2, the values are swapped.

**Constructors In C++ | C++ Tutorials for Beginners #29**

In this tutorial, we will discuss constructors in C++

**Constructors in C++**

A constructor is a special member function with the same name as the class. The constructor doesn’t have a return type. Constructors are used to initialize the objects of its class. Constructors are automatically invoked whenever an object is created.

**Important Characteristics of Constructors in C++**

* A constructor should be declared in the public section of the class
* They are automatically invoked whenever the object is created
* They cannot return values and do not have return types
* It can have default arguments
* We cannot refer to their address

An example program to demonstrate the concept of the constructor is shown below.

#include <iostream>

using namespace std;

class Complex

{

int a, b;

public:

// Creating a Constructor

// Constructor is a special member function with the same name as of the class.

//It is used to initialize the objects of its class

//It is automatically invoked whenever an object is created

Complex(void); // Constructor declaration

void printNumber()

{

cout << "Your number is " << a << " + " << b << "i" << endl;

}

};

Complex ::Complex(void) // ----> This is a default constructor as it takes no parameters

{

a = 10;

b = 0;

// cout<<"Hello world";

}

Copy

**Code Snippet 1: Constructor Example Program**

As shown in a code snippet 1,

* 1st “complex” class is defined which consists of private data members “a” and “b”.
* 2nd default constructor of the “complex” class is declared.
* 3rd function “printNumber” is defined which will print the values of the data members “a” and “b”.
* 4th default constructor is defined which will assign the values to the data members “a” and “b”. The main things to note here are that whenever a new object will be created this constructor will run and if the parameters are not passed to the constructor it is called a default constructor.

The main program is shown in code snippet 2.

int main()

{

Complex c1, c2, c3;

c1.printNumber();

c2.printNumber();

c3.printNumber();

return 0;

}

Copy

**Code Snippet 2: Main Program**

As shown in Code Snippet 2,

* 1st objects “c1”, “c2”, and “c3” of the complex data type are created. The main thing to note here is that when we are creating objects the constructor will run for each object and will assign the values.
* 2nd function “printNumber” is called by the objects “c1”, “c2”, and “c3”.

The output for the following program is shown in figure 1.

Graphical user interface, text

Description automatically generated

**Figure 1:** Program Output

As shown in figure 1, whenever a “printNumber” function is called it prints the values which are being assigned through the constructor.

**Parameterized and Default Constructors In C++ | C++ Tutorials for Beginners #30**

In this tutorial, we will discuss parameterized and default constructors in C++

**Parameterized and Default Constructors in C++**

Parameterized constructors are those constructors that take one or more parameters. Default constructors are those constructors that take no parameters. The main things to note here are that constructors are written in the public section of the class and the constructors don’t have a return type. An example program to demonstrate the concept of the constructor is shown below.

**Parameterized Constructors Example Program 1**

#include<iostream>

using namespace std;

class Complex

{

int a, b;

public:

Complex(int, int); // Constructor declaration

void printNumber()

{

cout << "Your number is " << a << " + " << b << "i" << endl;

}

};

Complex ::Complex(int x, int y) // ----> This is a parameterized constructor as it takes 2 parameters

{

a = x;

b = y;

// cout<<"Hello world";

}

Copy

**Code Snippet 1:  Parameterized Constructor Example Program 1**

As shown in a code snippet 1,

* 1st “complex” class is defined which consists of private data members “a” and “b”.
* 2nd parameterized constructor of the “complex” class is declared which takes two parameters.
* 3rd function “printNumber” is defined which will print the values of the data members “a” and “b”.
* 4th parameterized constructor is defined which takes two parameters and assigns the values to the data members “a” and “b”. The main things to note here are that whenever a new object will be created this constructor will run.

The main program is shown in code snippet 2.

int main(){

// Implicit call

Complex a(4, 6);

a.printNumber();

// Explicit call

Complex b = Complex(5, 7);

b.printNumber();

return 0;

}

Copy

**Code Snippet 2: Main Program**

As shown in Code Snippet 2,

* 1st parameterized constructor is called implicitly with the object “a” and the values “4” and “6” are passed
* 2nd function “printNumber” is called which will print the values of data members
* 3rd parameterized constructor is called explicitly with the object “b” and the values “5” and “7” are passed
* 4th function “printNumber” is called again which will print the values of data members

The output for the following program is shown in figure 1.

Text

Description automatically generated with medium confidence

**Figure 1:** Program Output 1

**Parameterized Constructors Example Program 2**

#include<iostream>

using namespace std;

class Point{

int x, y;

public:

Point(int a, int b){

x = a;

y = b;

}

void displayPoint(){

cout<<"The point is ("<<x<<", "<<y<<")"<<endl;

}

};

Copy

**Code Snippet 3: Parameterized Constructor Example Program 2**

As shown in Code Snippet 3,

* 1st “point” class is defined which consists of private data members “x” and “y”.
* 2nd parameterized constructor of the “point” class is defined which takes two parameters and assigns the values to the private data members of the class.
* 3rd function “displayPoint” is defined which will print the values of the data members “x” and “y”.

The main program is shown in code snippet 4.

int main(){

Point p(1, 1);

p.displayPoint();

Point q(4, 6);

q.displayPoint();

return 0;

}

Copy

**Code Snippet 4: Main Program**

As shown in Code Snippet 4,

* 1st parameterized constructor is called implicitly with the object “p” and the values “1” and “1” are passed
* 2nd function “displayPoint” is called which will print the values of data members
* 3rd parameterized constructor is called implicitly with the object “q” and the values “4” and “6” are passed
* 4th function “displayPoint” is called which will print the values of data members

The output for the following program is shown in figure 2.



**Figure 2:** Program Output 2

**Constructor Overloading In C++ | C++ Tutorials for Beginners #31**

In this tutorial, we will discuss constructor overloading in C++

**Constructor Overloading in C++**

Constructor overloading is a concept in which one class can have multiple constructors with different parameters. The main thing to note here is that the constructors will run according to the arguments for example if a program consists of 3 constructors with 0, 1, and 2 arguments, so if we pass 1 argument to the constructor the compiler will automatically run the constructor which is taking 1 argument. An example program to demonstrate the concept of Constructor overloading in C++ is shown below.

#include <iostream>

using namespace std;

class Complex

{

int a, b;

public:

Complex(){

a = 0;

b =0;

}

Complex(int x, int y)

{

a = x;

b = y;

}

Complex(int x){

a = x;

b = 0;

}

void printNumber()

{

cout << "Your number is " << a << " + " << b << "i" << endl;

}

};

Copy

**Code Snippet 1: Constructor Overloading Program Example**

As shown in Code Snippet 1,

* 1st we created a “complex” class which consists of private data members “a” and “b”.
* 2nd default constructor of the “complex” class is declared which has no parameters and assigns “0” to the data members “a” and “b”.
* 3rd parameterized constructor of the “complex” class is declared which takes two parameters and assigns values to the data members “a” and “b”.
* 4th parameterized constructor of the “complex” class is declared which takes one parameter and assigns values to the data members “a” and “b”.
* 5th function “printNumber” is defined which will print the values of the data members “a” and “b”.

The main program is shown in code snippet 2.

int main()

{

Complex c1(4, 6);

c1.printNumber();

Complex c2(5);

c2.printNumber();

Complex c3;

c3.printNumber();

return 0;

}

Copy

**Code Snippet 2: Main Program**

As shown in Code Snippet 2,

* 1st parameterized constructor is called with the object “c1” and the values “4” and “6” are passed. The main thing to note here is that this will run the constructor with two parameters.
* 2nd function “printNumber” is called which will print the values of data members
* 3rd parameterized constructor is called with the object “c2” and the value “5” is passed. The main thing to note here is that this will run the constructor with one parameter.
* 4th function “printNumber” is called which will print the values of data members
* 5th default constructor is called with the object “c3”. The main thing to note here is that this will run the constructor with no parameters.
* 6th function “printNumber” is called which will print the values of data members

The output for the following program is shown in figure 1.

Text

Description automatically generated

**Figure 1:** Program Output

As shown in figure 1, all the values which were passed and assigned through parameterized constructors and the values which were assigned through the default constructor are printed.

**Constructors With Default Arguments In C++ | C++ Tutorials for Beginners #32**

In this tutorial, we will discuss constructors with default arguments in C++

**Constructors with Default Arguments in C++**

Default arguments of the constructor are those which are provided in the constructor declaration. If the values are not provided when calling the constructor the constructor uses the default arguments automatically. An example program to demonstrate the concept default arguments in C++ is shown below.

#include<iostream>

using namespace std;

class Simple{

int data1;

int data2;

int data3;

public:

Simple(int a, int b=9, int c=8){

data1 = a;

data2 = b;

data3 = c;

}

void printData();

};

void Simple :: printData(){

cout<<"The value of data1, data2 and data3 is "<<data1<<", "<< data2<<" and "<< data3<<endl;

}

Copy

**Code Snippet 1: Constructor with Default Arguments Program Example**

As shown in a code snippet 1,

* 1st we created a “simple” class which consists of private data members “data1”, “data2” and “data3”.
* 2nd parameterized constructor of the “simple” class is defined which takes three parameters and assigns values to the data members “a” and “b”. The main thing to note here is that the value “9” and “8” are the default values for the variables “b” and “c”.
* 3rd function “printData” is defined which prints the values of the data members “data1”, “data2”, and “data3”.

The main program is shown in code snippet 2.

int main(){

Simple s(12, 13);

s.printData();

return 0;

}

Copy

**Code Snippet 2: Main Program**

As shown in code snippet 2,

* 1st parameterized constructor is called with the object “s” of the data type “simple” and the values “12” and “13” are passed. The main thing to note here is that the value of the parameter “c” will be automatically set by the default value.
* 2nd function “printData” is called which will print the values of data members.

The output for the following program is shown in figure 1.



**Figure 1:**Program Output

As shown in figure 1, the value “12”, “13”, and “8” are printed. The constructor assigned the values “12” and “13” to the variables “a” and “b” but the value for the variable “c” was not passed that’s why constructors set the value “8” which was the default value for the variable “c”.

**Dynamic Initialization of Objects Using Constructors | C++ Tutorials for Beginners #33**

In this tutorial, we will discuss the dynamic initialization of objects using constructors in C++

**Dynamic Initialization of Objects Using Constructors**

The dynamic initialization of the object means that the object is initialized at the runtime.  Dynamic initialization of the object using a constructor is beneficial when the data is of different formats. An example program is shown below to demonstrate the concept of dynamic initialization of objects using constructors.

#include<iostream>

using namespace std;

class BankDeposit{

int principal;

int years;

float interestRate;

float returnValue;

public:

BankDeposit(){}

BankDeposit(int p, int y, float r); // r can be a value like 0.04

BankDeposit(int p, int y, int r); // r can be a value like 14

void show();

};

Copy

**Code Snippet 1: Dynamic Initialization of Objects using Constructor Example**

As shown in Code Snippet 1,

* 1st we created a “BankDeposit” class which consists of private data members “principal”, “years”, “interestRate”, and “returnValue”.
* 2nd default constructor of the “BankDeposit” class is declared.
* 3rd parameterized constructor of the “BankDeposit” class is declared which takes three parameters “p”, “y”, and “r”. The main thing to note here is that the parameter “r” is of a float data type.
* 4th parameterized constructor of the “BankDeposit” class is declared which takes three parameters “p”, “y”, and “r”. The main thing to note here is that the parameter “r” is of an integer data type.
* 5th function “show” is declared.

The definition of constructors and function is shown below.

BankDeposit :: BankDeposit(int p, int y, float r)

{

principal = p;

years = y;

interestRate = r;

returnValue = principal;

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

{

returnValue = returnValue \* (1+interestRate);

}

}

BankDeposit :: BankDeposit(int p, int y, int r)

{

principal = p;

years = y;

interestRate = float(r)/100;

returnValue = principal;

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

{

returnValue = returnValue \* (1+interestRate);

}

}

void BankDeposit :: show(){

cout<<endl<<"Principal amount was "<<principal

<< ". Return value after "<<years

<< " years is "<<returnValue<<endl;

}

Copy

**Code Snippet 2: Definition of Constructors and Function**

As shown in Code snippet 2,

* 1st the constructor “BankDeposit” is defined in which the value of the parameter “p” is assigned to the data member “principal”; the value of the parameter “y” is assigned to the data member “year”; the value of the parameter “r” is assigned to the data member “interestRate”. At the end “for” loop is defined which will run till the length of the variable “y” and add “1” in the “interestRate”; then multiply the value with the “returnValue”. The main thing to note here is that in this constructor the data type of the parameter “r” is float.
* 2nd another constructor “BankDeposit” is defined in which the value of the parameter “p” is assigned to the data member “principal”; the value of the parameter “y” is assigned to the data member “year”; the value of the parameter “r” is converted to “float” and divided by “100” then assigned to the data member “interestRate”. At the end “for” loop is defined which will run till the length of the variable “y” and add “1” in the “interestRate”; then multiply the value with the “returnValue”. The main thing to note here is that in this constructor the data type of the parameter “r” is float.
* 3rd the function “show” is defined which will print the values of the data members “principal”, “year”, and “returnValue”.

The main program is shown in code snippet 3.

int main(){

BankDeposit bd1, bd2, bd3;

int p, y;

float r;

int R;

cout<<"Enter the value of p y and r"<<endl;

cin>>p>>y>>r;

bd1 = BankDeposit(p, y, r);

bd1.show();

cout<<"Enter the value of p y and R"<<endl;

cin>>p>>y>>R;

bd2 = BankDeposit(p, y, R);

bd2.show();

return 0;

}

Copy

**Code Snippet 3: Main Program**

As shown in a code snippet 3,

* 1st the object “bd1”, “bd2”, and “bd3” of the data type “BankDeposit” are created.
* 2nd the integer variables “p” and “y” are declared; the float variable “r” is declared, and the integer variable “R” is declared.
* 3rd the values for the variables “p”, “y”, and”r” are taken from the user on the runtime.
* 4th parameterized constructor “BankDeposit” is called with the object “bd1” and the variables “p”, “y”, and “r” are passed. The main thing to note here is that this will run the constructor with float parameters “r”.
* 5th function “show” is called which will print the values of data members
* 6th the values for the variables “p”, “y”, and ”R” are taken from the user on the runtime.
* 7th parameterized constructor “BankDeposit” is called with the object “bd2” and the variables “p”, “y”, and “R” are passed. The main thing to note here is that this will run the constructor with integer parameters “R”.
* 8th function “show” is called which will print the values of data members.

The output for the following program is shown in figure 1.

Text

Description automatically generated

**Figure 1:** Program Output

As shown in figure 1, the first time the values “100”, “1”, and “0.05” are entered and it gives us the return value of “105”. The second time the values “100”, “1”, and “5” are entered and it gives us the return value of “105”. So the main thing to note here is that the compiler figures out the run time by seeing the data type and runs the relevant constructor.

**Copy Constructor in C++ | C++ Tutorials for Beginners #34**

In this tutorial, we will discuss copy constructor in C++

**Copy Constructor in C++**

A copy constructor is a type of constructor that creates a copy of another object. If we want one object to resemble another object we can use a copy constructor. If no copy constructor is written in the program compiler will supply its own copy constructor. An example program to demonstrate the concept of a Copy constructor in C++ is shown below.

#include<iostream>

using namespace std;

class Number{

int a;

public:

Number(){

a = 0;

}

Number(int num){

a = num;

}

// When no copy constructor is found, compiler supplies its own copy constructor

Number(Number &obj){

cout<<"Copy constructor called!!!"<<endl;

a = obj.a;

}

void display(){

cout<<"The number for this object is "<< a <<endl;

}

};

Copy

**Code Snippet 1: Copy Constructor Example Program**

As shown in Code Snippet 1,

* 1st we created a “number” class which consists of private data member “a”.
* 2nd default constructor of the “number” class is defined which has no parameters and assign “0” to the data members “a”.
* 3rd parameterized constructor of the “number” class is defined which takes one parameter and assigns values to the data members “a”.
* 4th copy constructor of the “number” class is defined which takes its own reference object as a parameter and assigns values to the data members “a”.
* 5th function “display” is defined which will print the values of the data members “a”.

The main program is shown in code snippet 2.

int main(){

Number x, y, z(45), z2;

x.display();

y.display();

z.display();

Number z1(z); // Copy constructor invoked

z1.display();

z2 = z; // Copy constructor not called

z2.display();

Number z3 = z; // Copy constructor invoked

z3.display();

// z1 should exactly resemble z or x or y

return 0;

}

Copy

**Code Snippet 2: Main Program**

As shown in Code Snippet 2,

* 1st objects “x”, “y”, “z”, and “z1” are created of the “number” data type. The main thing to note here is that the object “z” has a value “45”.
* 2nd function “display” is called by the objects “x”, “y”, and “z”.
* 3rd copy constructor is invoked and the object “z” is passed to “z1”
* 4th function “display” is called by the object “z1”
* 5th the value of “z” is assigned to “z1”. The main thing to note here is that it will not invoke a copy constructor because the object “z” is already created.
* 6th function “display” is called by the object “z2”
* 7th the value of “z” is assigned to “z3”. The main thing to note here is that it will invoke a copy constructor because the object “z3” is being created.
* 8th function “display” is called by the object “z3”

The output for the following program is shown in figure 1.

Text

Description automatically generated

**Figure 1:**Program Output

As shown in figure 1, all the values which were passed and assigned through copy constructors are printed.

**Destructor in C++ in Hindi | C++ Tutorials for Beginners #35**

In this tutorial, we will discuss Destructor in C++

**Destructor in C++**

A destructor is a type of function which is called when the object is destroyed. Destructor never takes an argument nor does it return any value. An example program to demonstrate the concept of destructors in C++ is shown below.

#include<iostream>

using namespace std;

// Destructor never takes an argument nor does it return any value

int count=0;

class num{

public:

num(){

count++;

cout<<"This is the time when constructor is called for object number"<<count<<endl;

}

~num(){

cout<<"This is the time when my destructor is called for object number"<<count<<endl;

count--;

}

};

Copy

**Code Snippet 1: Destructor Example Program**

As shown in Code Snippet 1,

* 1st global variable “count” is initialized.
* 2nd we created a “num” class.
* 3rd default constructor of the “num” class is defined which has no parameters and does increment in the variable “count” and prints its value. The main thing to note here is that every time the new object will be created this constructor will run.
* 4th destructor of the “num” class is defined. The destructor prints the value of the variable “count” and decrement in the value of “count”. The main thing to note here is that every time the object has been destroyed this destructor will run.

The main program is shown in code snippet 2.

int main(){

cout<<"We are inside our main function"<<endl;

cout<<"Creating first object n1"<<endl;

num n1;

{

cout<<"Entering this block"<<endl;

cout<<"Creating two more objects"<<endl;

num n2, n3;

cout<<"Exiting this block"<<endl;

}

cout<<"Back to main"<<endl;

return 0;

}

Copy

**Code Snippet 2: Main Program**

As shown in Code Snippet 2,

* 1st object “n1” is created of the “num” data type. The main thing to note here is that when the object “n1” is created the constructor will run.
* 2nd inside the block two objects “n2” and “n3” are created of the “num” data type. The main things to note here are that when the objects “n2” and “n3” are created the constructor will run for both objects and when the block ends the destructor will run for both objects “n2” and “n3”.
* 3rd when the program ends the destructor for the object “n1” will run.

The output for the following program is shown in figure 1.

Text

Description automatically generated

**Figure 1:**Program Output

As shown in figure 1, first the constructor for the object “n1” was called; second the constructor for the objects “n2” and “n3” was called; third the destructor was called for the objects “n2” and “n3”; at the end destructor for the object “n1” was called.

**Inheritance Syntax & Visibility Mode in C++ | C++ Tutorials for Beginners #37**

In this tutorial, we will discuss inheritance syntax and visibility mode in C++

**Inheritance Syntax and Visibility mode in C++**

Inheritance is a process of inheriting attributes of the base class by a derived class. The syntax of the derived class is shown below.

// Derived Class syntax

class {{derived-class-name}} : {{visibility-mode}} {{base-class-name}}

{

class members/methods/etc...

}

Copy

**Code Snippet 1: Derived Class syntax**

As shown in a code snippet 1,

* After writing the class keyword we have to write the derived class name and then put a “:” sign.
* After “:” sign we have to write the visibility mode and then write the base class name.

Note:

* Default visibility mode is private
* Public Visibility Mode: Public members of the base class becomes Public members of the derived class
* Private Visibility Mode: Public members of the base class become private members of the derived class
* Private members are never inherited

An example program is shown below to demonstrate the concept of inheritance.

#include <iostream>

using namespace std;

// Base Class

class Employee

{

public:

int id;

float salary;

Employee(int inpId)

{

id = inpId;

salary = 34.0;

}

Employee() {}

};

// Creating a Programmer class derived from Employee Base class

class Programmer : public Employee

{

public:

int languageCode;

Programmer(int inpId)

{

id = inpId;

languageCode = 9;

}

void getData(){

cout<<id<<endl;

}

};

Copy

**Code Snippet 2: Inheritance Example Program**

As shown in Code snippet 2,

* 1st we created an “employee” class which consists of public data member’s integer “id” and float “salary”.
* 2nd the “employee” class consists of a parameterized constructor that takes an integer “inpid” parameter and assigns its value to the data member “id”. The value of variable “salary” is set to “34”.
* 3rd the “employee” class also consists of default constructor.
* 4th we created a “programmer” class that is inheriting “employee” class. The main thing to note here is that the “visibility-mode” is “public”.
* 5ththe “programmer” class consists of public data member’s integer “languageCode”.
* 6th the “programmer” class consists of a parameterized constructor that takes an integer “inpid” parameter and assigns its value to the data member “id”. The value of variable “languageCode” is set to “9”.
* 7th “programmer” class consists of a function “getData” which will print the value of the variable “id”.

The main program is shown in code snippet 3.

int main()

{

Employee harry(1), rohan(2);

cout << harry.salary << endl;

cout << rohan.salary << endl;

Programmer skillF(10);

cout << skillF.languageCode<<endl;

cout << skillF.id<<endl;

skillF.getData();

return 0;

}

Copy

**Code Snippet 3: Main Program**

As shown in a code snippet 3,

* 1st objects “harry” and “rohan” is created of the “employee” data type. Object “harry” is passed with the value “1” and the object “rohan” is passed with the value “2”.
* 2nd the “salary” of both objects “rohan” and “harry” are printed.
* 3rd object “skillF” is created of the “programmer” data type. Object “skillF” is passed with the value “10”.
* 4th the “languageCode” and “id” of both object “skillF” is printed.
* 5th the function “getData” is called by the “skillF” object. This will print the “id”.

The output for the following program is shown in figure 1.

Text

Description automatically generated

**Figure 1:**Program Output

**Single Inheritance Deep Dive: Examples + Code | C++ Tutorials for Beginners #38**

In this tutorial, we will discuss single inheritance in C++

**Single Inheritance in C++**

Single inheritance is a type of inheritance in which a derived class is inherited with only one base class. For example, we have two classes “employee” and “programmer”. If the “programmer” class is inherited from the “employee” class which means that the “programmer” class can now implement the functionalities of the “employee” class.

An example program to demonstrate the concept of single inheritance in C++ is shown below.

class Base

{

int data1; // private by default and is not inheritable

public:

int data2;

void setData();

int getData1();

int getData2();

};

void Base ::setData(void)

{

data1 = 10;

data2 = 20;

}

int Base::getData1()

{

return data1;

}

int Base::getData2()

{

return data2;

}

Copy

**Code Snippet 1: Base Class**

As shown in a code snippet 1,

* 1st we created a “base” class which consists of private data member’s integer “data1” and public data member integer “data2”.
* 2nd the “base” class consists of three member functions “setData”, “getData1”, and “getData2”.
* 3rd the function “setData” will assign the values “10” and “20” to the data members “data1” and “data2”.
* 4th the function “getData1” will return the value of the data member “data1”.
* 5th the function “getData2” will return the value of the data member “data2”.

The derived class will inherit the base class which is shown below.

class Derived : public Base

{ // Class is being derived publically

int data3;

public:

void process();

void display();

};

void Derived ::process()

{

data3 = data2 \* getData1();

}

void Derived ::display()

{

cout << "Value of data 1 is " << getData1() << endl;

cout << "Value of data 2 is " << data2 << endl;

cout << "Value of data 3 is " << data3 << endl;

}

Copy

**Code Snippet 2: Derived Class**

As shown in Code snippet 2,

* 1st we created a “derived” class which is inheriting the base class publically. The “derived” class consists of private data member’s integer “data3”.
* 2nd the “derived” class consists of two public member functions “process” and “display”.
* 3rd the function “process” will multiply the values “data2” and “data1”; and store the values in the variable “data3”.
* 4th the function “display” will print the values of the data member “data1”, “data2”, and “data3”.

The main program is shown in code snippet 3.

int main()

{

Derived der;

der.setData();

der.process();

der.display();

return 0;

}

Copy

**Code Snippet 3: Main Program**

As shown in a code snippet 3,

* 1st object “der” is created of the “derived” data type.
* 2nd the function “setData” is called by the object “der”. This function will set the values of the data members “data1” and “data2”
* 3rd the function “process” is called by the object “der”. This function will multiply the values “data2” and “data1”; and store their value in the variable “data3”.
* 4th the function “display” is called by the object “der”. This function will print the values of the data member “data1”, “data2”, and “data3”.

The output for the following program is shown in figure 1.

A picture containing text, electronics, close, keyboard

Description automatically generated

**Figure 1:**Program Output

**Protected Access Modifier in C++ | C++ Tutorials for Beginners #39**

In this tutorial, we will discuss protected access modifiers in C++

**Protected Access Modifiers in C++**

Protected access modifiers are similar to the private access modifiers but protected access modifiers can be accessed in the derived class whereas private access modifiers cannot be accessed in the derived class. A table is shown below which shows the behavior of access modifiers when they are derived “public”, “private”, and “protected”.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Public Derivation** | **Private Derivation** | **Protected Derivation** |
| **Private members** | Not Inherited | Not Inherited | Not Inherited |
| **Protected members** | Protected | Private | Protected |
| **Public members** | Public | Private | Protected |

As shown in the table,

1. If the class is inherited in public mode then its private members cannot be inherited in child class.
2. If the class is inherited in public mode then its protected members are protected and can be accessed in child class.
3. If the class is inherited in public mode then its public members are public and can be accessed inside child class and outside the class.
4. If the class is inherited in private mode then its private members cannot be inherited in child class.
5. If the class is inherited in private mode then its protected members are private and cannot be accessed in child class.
6. If the class is inherited in private mode then its public members are private and cannot be accessed in child class.
7. If the class is inherited in protected mode then its private members cannot be inherited in child class.
8. If the class is inherited in protected mode then its protected members are protected and can be accessed in child class.
9. If the class is inherited in protected mode then its public members are protected and can be accessed in child class.

An example program to demonstrate the concept of protected access modifiers is shown below.

#include<iostream>

using namespace std;

class Base{

protected:

int a;

private:

int b;

};

class Derived: protected Base{

};

int main(){

Base b;

Derived d;

// cout<<d.a; // Will not work since a is protected in both base as well as derived class

return 0;

}

Copy

**Code Snippet 1: Protected Access Modifier Example Program**

As shown in a code snippet 1,

* 1st we created a “Base” class which consists of protected data member integer “a” and private data member integer “b”.
* 2nd we created a “Derived” class which is inheriting the “Base” class in protected mode.
* 3rd the object “b” of the data type “Base” is created.
* 4th the object “d” of the data type “Derived” is created.
* 5th if we try to print the value of the data member “a” by using the object “d”; the program will throw an error because the data member “a” is protected and the derived class is inherited in the protected mode. So the data member “a” can only be accessed in the “derived” but not outside the class.

**Multilevel Inheritance Deep Dive with Code Example in C++ | C++ Tutorials for Beginners #40**

In this tutorial, we will discuss multilevel inheritance in C++

**Multilevel Inheritance in C++**

Multilevel inheritance is a type of inheritance in which one derived class is inherited from another derived class. For example, we have three classes “animal”, “mammal” and “cow”. If the “mammal” class is inherited from the “animal” class and “cow” class is inherited from “mammal” which means that the “mammal” class can now implement the functionalities of “animal” and “cow” class can now implement the functionalities of “mammal” class.

An example program is shown below to demonstrate the concept of multilevel inheritance in C++.

#include <iostream>

using namespace std;

class Student

{

protected:

int roll\_number;

public:

void set\_roll\_number(int);

void get\_roll\_number(void);

};

void Student ::set\_roll\_number(int r)

{

roll\_number = r;

}

void Student ::get\_roll\_number()

{

cout << "The roll number is " << roll\_number << endl;

}

Copy

**Code Snippet 1: Student Class**

As shown in a code snippet 1,

* 1st we created a “student” class which consists of protected data member integer “roll\_number”.
* 2nd the “student” class consists of a public function “set\_roll\_number” and “get\_roll\_number”
* 3rd the function “set\_roll\_number” will set the value of the data member “roll\_number”.
* 4th the function “get\_roll\_number” will print the value of the data member “roll\_number”.

The code for the “exam” class is shown below which is inheriting the “student” class

class Exam : public Student

{

protected:

float maths;

float physics;

public:

void set\_marks(float, float);

void get\_marks(void);

};

void Exam ::set\_marks(float m1, float m2)

{

maths = m1;

physics = m2;

}

void Exam ::get\_marks()

{

cout << "The marks obtained in maths are: " << maths << endl;

cout << "The marks obtained in physics are: " << physics << endl;

}

Copy

**Code Snippet 2: Exam Class**

As shown in Code snippet 2,

* 1st we created an “exam” class that is inheriting “student” class in public mode.
* 2nd the “exam” class consists of protected data members float “math” and float “physics”.
* 3rd the “exam” class consists of public member functions “set\_marks” and “get\_marks”.
* 4th the function “set\_marks” will set the value of the data members “math” and “physics”.
* 5th the function “get\_marks” will print the value of the data members “math” and “physics”.

The code for the “result” class is shown below which is inheriting the “exam” class

class Result : public Exam

{

float percentage;

public:

void display\_results()

{

get\_roll\_number();

get\_marks();

cout << "Your result is " << (maths + physics) / 2 << "%" << endl;

}

};

Copy

**Code Snippet 3: Result Class**

As shown in a code snippet 3,

* 1st we created a “Result” class which is inheriting the “Exam” class in public mode.
* 2nd the “Result” class consists of private data member’s float “percentage”.
* 3rd the “exam” class consists of the public member function “display\_results”.
* 4th the function “display\_results” will call the “get\_roll\_number” and “get\_marks” functions, and add the values of “math” and “physics” variables then divide that value with “2” to get a percentage and prints it.

It can be clearly seen that the class “Exam” is inheriting class “student” and class “Results” is inheriting class “Exam”; which is an example of multilevel inheritance. The code main program is shown below.

int main()

{

Result harry;

harry.set\_roll\_number(420);

harry.set\_marks(94.0, 90.0);

harry.display\_results();

return 0;

}

Copy

**Code Snippet 4: Main Program**

As shown in Code snippet 4,

* 1st object “harry” is created of the “Result” data type.
* 2nd the function “set\_roll\_number” is called by the object “harry” and the value “420” is passed.
* 3rd the function “set\_marks” is called by the object “harry” and the values “94.0” and “90.0” are passed.
* 4th the function “display\_results” is called by the object “harry”.

The output for the following program is shown in figure 1.

Text

Description automatically generated

**Multiple Inheritance Deep Dive with Code Example in C++ | C++ Tutorials for Beginners #41**

In this tutorial, we will discuss multiple inheritances in C++

**Multiple Inheritances in C++**

Multiple inheritances are a type of inheritance in which one derived class is inherited with more than one base class. For example, we have three classes “employee”, “assistant” and “programmer”. If the “programmer” class is inherited from the “employee” and “assistant” class which means that the “programmer” class can now implement the functionalities of the “employee” and “assistant” class. The syntax of inheriting multiple inheritances is shown below.

// class DerivedC: visibility-mode base1, visibility-mode base2

// {

// Class body of class "DerivedC"

// };

Copy

**Code Snippet 1: Multiple inheritances syntax**

As shown in a code snippet 1,

* After writing the class keyword we have to write the derived class name and then put a “:” sign.
* After “:” sign we have to write the visibility mode and then write the base class name and again we have to write the visibility mode and write another base class name.

An example program is shown below to demonstrate the concept of multiple inheritances in C++.

class Base1{

protected:

int base1int;

public:

void set\_base1int(int a)

{

base1int = a;

}

};

class Base2{

protected:

int base2int;

public:

void set\_base2int(int a)

{

base2int = a;

}

};

class Base3{

protected:

int base3int;

public:

void set\_base3int(int a)

{

base3int = a;

}

};

Copy

**Code Snippet 2: Base Classes**

As shown in Code snippet 2,

* 1st we created a “Base1” class which consists of protected data member integer “base1int”.
* 2nd the “Base1” class consists of a public function “set\_base1int”. This function will set the value of the data member “base1int”.
* 3rd we created a “Base2” class which consists of protected data member integer “base2int”.
* 4th the “Base2” class consists of a public function “set\_base2int”. This function will set the value of the data member “base2int”.
* 5th we created a “Base3” class which consists of protected data member integer “base3int”.
* 2nd the “Base3” class consists of a public function “set\_base3int”. This function will set the value of the data member “base3int”.

The code for the “Derived” class is shown below. “Derived” class will inherit all the base classes.

class Derived : public Base1, public Base2, public Base3

{

public:

void show(){

cout << "The value of Base1 is " << base1int<<endl;

cout << "The value of Base2 is " << base2int<<endl;

cout << "The value of Base3 is " << base3int<<endl;

cout << "The sum of these values is " << base1int + base2int + base3int << endl;

}

};

Copy

**Code Snippet 3: Derived Class**

As shown in a code snippet 3,

* 1st we created a “Derived” class which is inheriting “Base1”, “Base2”, and “Base3” classes in public mode.
* 2nd the “Derived” class consists of the public member function “show”.
* 4th the function “show” will first print the values of “base1int”, “base2int”, and “base3int” individually and then print the sum of all three values.

It can be clearly seen that the class “Derived” is inheriting class “Base1”, “Base2”, and “Base3”. This is an example of multiple inheritances. The code main program is shown below.

int main()

{

Derived harry;

harry.set\_base1int(25);

harry.set\_base2int(5);

harry.set\_base3int(15);

harry.show();

return 0;

}

Copy

**Code Snippet 4: Main Program**

As shown in Code snippet 4,

* 1st object “harry” is created of the “Derived” data type.
* 2nd the function “set\_base1int” is called by the object “harry” and the value “25” is passed.
* 3rd the function “set\_base2int” is called by the object “harry” and the value “5” is passed.
* 4th the function “set\_base3int” is called by the object “harry” and the value “15” is passed.
* 4th the function “show” is called by the object “harry”.

The output for the following program is shown in figure 1.

Text

Description automatically generated

**Figure 1:** Program Output

**Ambiguity Resolution in Inheritance in C++ | C++ Tutorials for Beginners #43**

In this tutorial, we will discuss ambiguity resolution in inheritance in C++

**Ambiguity Resolution in Inheritance**

Ambiguity in inheritance can be defined as when one class is derived for two or more base classes then there are chances that the base classes have functions with the same name. So it will confuse derived class to choose from similar name functions. To solve this ambiguity scope resolution operator is used “::”. An example program is shown below to demonstrate the concept of ambiguity resolution in inheritance.

class Base1{

public:

void greet(){

cout<<"How are you?"<<endl;

}

};

class Base2{

public:

void greet()

{

cout << "Kaise ho?" << endl;

}

};

class Derived : public Base1, public Base2{

int a;

public:

void greet(){

Base2 :: greet();

}

};

Copy

**Code Snippet 1:** Ambiguity Resolution in Inheritance Example Program 1

As shown in a code snippet 1,

1. We have created a “Base1” class which consists of public member function “greet”. The function “greet” will print “how are you?”
2. We have created a “Base2” class which consists of public member function “greet”. The function “greet” will print “kaise ho?”
3. We have created a “Derived” class which is inheriting “Base1” and “Base2” classes. The “Derived” class consists of public member function “greet”. The function “greet” will run the “greet” function of the “Base2” class because we have used a scope resolution operator to let the compiler know which function should it run otherwise it will cause ambiguity.

The code of the main function is shown below

int main(){

// Ambibuity 1

Base1 base1obj;

Base2 base2obj;

base1obj.greet();

base2obj.greet();

Derived d;

d.greet();

return 0;

}

Copy

**Code Snippet 2:** Main program 1

As shown in code snippet 2,

1. Object “base1obj” is created of the “Base1” data type.
2. Object “base3obj” is created of the “Base2” data type.
3. The function “greet” is called by the object “base1obj”.
4. The function “greet” is called by the object “base2obj”.
5. Object “d” is created of the “Derived” data type.
6. The function “greet” is called by the object “d”.

The main thing to note here is that when the function “greet” is called by the object “d” it will run the “greet” function of the “Base2” class because we had specified it using scope resolution operator “::” to get rid ambiguity. The output for the following program is shown in figure 1.

Text

Description automatically generated

**Figure 1: Output**

Another example of ambiguity resolution in inheritance is shown below.

class B{

public:

void say(){

cout<<"Hello world"<<endl;

}

};

class D: public B{

int a;

// D's new say() method will override base class's say() method

public:

void say()

{

cout << "Hello my beautiful people" << endl;

}

};

Copy

**Code Snippet 3:**  Ambiguity Resolution in Inheritance Example Program 2

As shown in a code snippet 3,

1. We have created a “B” class which consists of public member function “say”. The function “say” will print “hello world”
2. We have created a “D” class that is inheriting the “B” class. The “D” class consists of the public member function “say”. The function “say” will print “Hello my beautiful people”

The main thing to note here is that both “B” and “D” classes have the same function “say”, So if the class “D” will call the function “say” it will override the base class “say” method because compiler by default run the method which is already written in its own body. But if the function “say” was not present in the class “D” then the compiler will run the method of the class “B”.

The code of the main function is shown below,

int main(){

// Ambibuity 2

B b;

b.say();

D d;

d.say();

return 0;

}

Copy

**Code Snippet 4:** Main Program 2

As shown in code snippet 4,

1. Object “b” is created of the “B” data type.
2. The function “say” is called by the object “b”.
3. Object “d” is created of the “D” data type.
4. The function “say” is called by the object “d”.

The output for the following program is shown in figure 2.



**Figure 2: Output**

**Virtual Base Class in C++ | C++ Tutorials for Beginners #44**

In this tutorial, we will discuss virtual base class in C++

**Virtual Base Class in C++**

The virtual base class is a concept used in multiple inheritances to prevent ambiguity between multiple instances. For example: suppose we created a class “A” and two classes “B” and “C”, are being derived from class “A”. But once we create a class “D” which is being derived from class “B” and “C” as shown in figure 1.

A picture containing diagram

Description automatically generated

**Figure 1:** Virtual Base Class Example Diagram

As shown in figure 1,

1. Class “A” is a parent class of two classes “B” and “C”
2. And both “B” and “C” classes are the parent of class “D”

The main thing to note here is that the data members and member functions of class “A” will be inherited twice in class “D” because class “B” and “C” are the parent classes of class “D” and they both are being derived from class “A”.

So when the class “D” will try to access the data member or member function of class “A” it will cause ambiguity for the compiler and the compiler will throw an error. To solve this ambiguity we will make class “A” as a virtual base class. To make a virtual base class “virtual” keyword is used.

When one class is made virtual then only one copy of its data member and member function is passed to the classes inheriting it. So in our example when we will make class “A” a virtual class then only one copy of the data member and member function will be passed to the classes “B” and “C” which will be shared between all classes. This will help to solve the ambiguity.

The syntax of the virtual base class is shown in the code snippet below,

#include <iostream>

using namespace std;

class A {

public:

void say()

{

cout << "Hello world"<<endl;

}

};

class B : public virtual A {

};

class C : public virtual A {

};

class D : public B, public C {

};

Copy

**Code Snippet 1:** Virtual Base Class Syntax Example Code

**Code Example Demonstrating Virtual Base Class in C++ | C++ Tutorials for Beginners #45**

In this tutorial, we will discuss demonstrating of virtual base class in C++

**Virtual Base Class in C++**

The virtual base class is a concept used in multiple inheritances to prevent ambiguity between multiple instances. For example: suppose we created a class “Student” and two classes “Test” and “Sports”, are being derived from class “Student”. But once we create a class “Result” which is being derived from class “Test” and “Sports” as shown in figure 1.

A picture containing diagram

Description automatically generated

**Figure 1:** Virtual Base Class Example Diagram

As shown in figure 1,

1. Class “Student” is a parent class of two classes “Test” and “Sports”
2. And both “Test” and “Sports” classes are the parent of class “Result”

The main thing to note here is that the data members and member functions of class “Student” will be inherited twice in class “Result” because class “Test” and “Sports” are the parent classes of class “Result” and they both are being derived from class “Student”.

So when the class “Result” will try to access the data member or member function of class “Student” it will cause ambiguity for the compiler and the compiler will throw an error. To solve this ambiguity we will make class “Student” as a virtual base class. To make a virtual base class “virtual” keyword is used.

When one class is made virtual then only one copy of its data member and member function is passed to the classes inheriting it. So in our example when we will make class “Student” a virtual class then only one copy of data member and member function will be passed to the classes “Test” and “Sports” which will be shared between all classes. This will help to solve the ambiguity.

An example program of the following diagram is shown in a code snippet below,

#include<iostream>

using namespace std;

class Student{

protected:

int roll\_no;

public:

void set\_number(int a){

roll\_no = a;

}

void print\_number(void){

cout<<"Your roll no is "<< roll\_no<<endl;

}

};

class Test : public Student{

protected:

float maths, physics;

public:

void set\_marks(float m1, float m2){

maths = m1;

physics = m2;

}

void print\_marks(void){

cout << "You result is here: "<<endl

<< "Maths: "<< maths<<endl

<< "Physics: "<< physics<<endl;

}

};

class Sports: public Student{

protected:

float score;

public:

void set\_score(float sc){

score = sc;

}

void print\_score(void){

cout<<"Your PT score is "<<score<<endl;

}

};

class Result : public Test, public Sports{

private:

float total;

public:

void display(void){

total = maths + physics + score;

print\_number();

print\_marks();

print\_score();

cout<< "Your total score is: "<<total<<endl;

}

};

Copy

**Code Snippet 1:**Virtual Base Class Example Program

As shown in a code snippet 1,

1. We have created a “Student” class that consists of protected data member “roll\_no” and member functions “set\_number” and “print\_number”. The function “set\_number” will assign the value to the protected data member “roll\_no” and the function “print\_number” will print the value of data member “roll\_no”.
2. We have created a “Test” class that is inheriting the virtual base class “Student”. The “Test” consists of protected data members “maths” and “physics” and member functions “set\_marks” and “print\_marks”. The function “set\_number” will assign the values to the protected data members “maths” and “physics” and the function “print\_marks” will print the value of data members “maths” and “physics”.
3. We have created a “Sports” class that is inheriting the virtual base class “Student”. The “Sports” consists of protected data member “score” and member functions “set\_score” and “print\_score”. The function “set\_score” will assign the values to the protected data members “score” and “physics” and the function “print\_score” will print the value of data members “score”.
4. We have created a “Result” class which is inheriting base classes “Test” and “Sports”. The “Result” consists of protected data member “total” and member functions “display”. The function “display” will first add the values of data members “math”, “physics”, and “score” and assign the value to the protected data members “total” and second the “display” function will call the functions “print\_number”, ”print\_marks”, and “print\_score” and also print the value of the data member “total”.

The code of the main function is shown below,

int main(){

Result harry;

harry.set\_number(4200);

harry.set\_marks(78.9, 99.5);

harry.set\_score(9);

harry.display();

return 0;

}

Copy

**Code Snippet 2:**Main Program

As shown in code snippet 2,

1. Object “harry” is created of the “Result” data type.
2. The function “set\_number” is called by the object “harry” and the value “4200” is passed.
3. The function “set\_marks” is called by the object “harry” and the values “48.9” and “99.5” are passed.
4. The function “set\_score” is called by the object “harry” and the value “9” is passed.
5. The function “display” is called by the object “harry”.

The main thing to note here is that there will be no ambiguity because we have made the “Student” class as a virtual base class but if we remove the “virtual” keyword then the compare will throw an error. The output of the following program is shown below.

Text

Description automatically generated

**Constructors in Derived Class in C++ | C++ Tutorials for Beginners #46**

In this tutorial, we will discuss constructors in derived class in C++

**Constructors in Derived Class in C++**

* We can use constructors in derived classes in C++
* If the base class constructor does not have any arguments, there is no need for any constructor in the derived class
* But if there are one or more arguments in the base class constructor, derived class need to pass argument to the base class constructor
* If both base and derived classes have constructors, base class constructor is executed first

**Constructors in Multiple Inheritances**

* In multiple inheritances, base classes are constructed in the order in which they appear in the class deceleration. For example if there are three classes “A”, “B”, and “C”, and the class “C” is inheriting classes “A” and “B”. If the class “A” is written before class “B” then the constructor of class “A” will be executed first. But if the class “B” is written before class “A” then the constructor of class “B” will be executed first.
* In multilevel inheritance, the constructors are executed in the order of inheritance. For example if there are three classes “A”, “B”, and “C”, and the class “B” is inheriting classes “A” and the class “C” is inheriting classes “B”. Then the constructor will run according to the order of inheritance such as the constructor of class “A” will be called first then the constructor of class “B” will be called and at the end constructor of class “C” will be called.

**Special Syntax**

* C++ supports a special syntax for passing arguments to multiple base classes
* The constructor of the derived class receives all the arguments at once and then will pass the call to the respective base classes
* The body is called after the constructors is finished executing

**Syntax Example:**

Derived-Constructor (arg1, arg2, arg3….): Base 1-Constructor (arg1,arg2), Base 2-Constructor(arg3,arg4)

{

….

} Base 1-Constructor (arg1,arg2)

Copy

**Special Case of Virtual Base Class**

* The constructors for virtual base classes are invoked before a non-virtual base class
* If there are multiple virtual base classes, they are invoked in the order declared
* Any non-virtual base class are then constructed before the derived class constructor is executed

**Code Example: Constructors in Derived Class in Cpp | C++ Tutorials for Beginners #48**

In this tutorial, we will discuss constructors in derived class with code example in C++

**Constructors in Derived Class in C++**

As we have discussed before about the constructors in derived class in a code snippet below three cases are given to clarify the execution of constructors.

/\*

Case1:

class B: public A{

// Order of execution of constructor -> first A() then B()

};

Case2:

class A: public B, public C{

// Order of execution of constructor -> B() then C() and A()

};

Case3:

class A: public B, virtual public C{

// Order of execution of constructor -> C() then B() and A()

};

\*/

Copy

**Code Snippet 1:**Constructors Execution Example Cases

As shown in Code Snippet 1,

1. In case 1, class “B” is inheriting class “A”, so the order of execution will be that first the constructor of class “A” will be executed and then the constructor of class “B” will be executed.
2. In case 2, class “A” is inheriting two classes “B” and “C”, so the order of execution will be that first constructor of class “B” will be executed and then the constructor of class “C” will be executed and at the end constructor of class “A” will be executed.
3. In case 3, class “A” is inheriting two classes “B” and virtual class “C”, so the order of execution will be that first constructor of class “C” will be executed because it is a virtual class and it is given more preference and then the constructor of class “B” will be executed and at the end constructor of class “A” will be executed.

To demonstrate the concept of constructors in derived classes an example program is shown below.

class Base1{

int data1;

public:

Base1(int i){

data1 = i;

cout<<"Base1 class constructor called"<<endl;

}

void printDataBase1(void){

cout<<"The value of data1 is "<<data1<<endl;

}

};

class Base2{

int data2;

public:

Base2(int i){

data2 = i;

cout << "Base2 class constructor called" << endl;

}

void printDataBase2(void){

cout << "The value of data2 is " << data2 << endl;

}

};

class Derived: public Base2, public Base1{

int derived1, derived2;

public:

Derived(int a, int b, int c, int d) : Base2(b), Base1(a)

{

derived1 = c;

derived2 = d;

cout<< "Derived class constructor called"<<endl;

}

void printDataDerived(void)

{

cout << "The value of derived1 is " << derived1 << endl;

cout << "The value of derived2 is " << derived2 << endl;

}

};

Copy

**Code Snippet 2:** Constructors in Derived Class Example Program

As shown in code snippet 2,

1. We have created a “Base1” class which consists of private data member “data1” and parameterized constructor which takes only one argument and set the value of data member “data1”. The “Base1” class also contains the member function “printDataBase1” which will print the value of data member “data1”.
2. We have created a “Base2” class which consists of private data member “data2” and parameterized constructor which takes only one argument and set the value of data member “data2”. The “Base2” class also contains the member function “printDataBase2” which will print the value of data member “data2”.
3. We have created a “Derived” class that is inheriting base classes “Base1” and “Base2”. The “Derived” class consists of private data members “derived1” and “derived2”. The “Derived” class contains parameterized constructor which calls the “Base1” and “Base2” class constructors to pass the values, it also assigns the values to the data members “derived1” and “derived2”. The “Derived” class also contains member functions “printDataDerived”. The function “printDataDerived” will print the values of the data member “derived1” and “derived2”.

The main thing to note here is that the constructors will be executed in the order in which the classes are being inherited. As in the example program above the “Base2” class is being inherited first and then “Base1” class is being inherited, so the constructor of “Base2” class will be executed first. The main program of the following example code is shown below.

int main(){

Derived harry(1, 2, 3, 4);

harry.printDataBase1();

harry.printDataBase2();

harry.printDataDerived();

return 0;

}

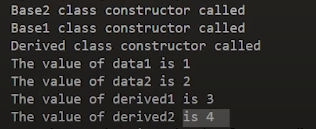
Copy

**Code Snippet 3:** Main Program

As shown in code snippet 3,

1. Object “harry” is created of the “Derived” data type and the values (1, 2, 3, 4) are passed.
2. The function “printDataBase1” is called by the object “harry”.
3. The function “printDataBase2” is called by the object “harry”.
4. The function “printDataDerived” is called by the object “harry”.

The output of the following program is shown below,



**Figure 1:** Program Output

**Initialization list in Constructors in Cpp | C++ Tutorials for Beginners #49**

In this tutorial, we will discuss the Initialization list in Constructors in C++

**Initialization list in Constructors in C++**

The initialization list in constructors is another concept of initializing the data members of the class. The syntax of the initialization list in constructors is shown below.

/\*

Syntax for initialization list in constructor:

constructor (argument-list) : initilization-section

{

assignment + other code;

}

Copy

**Code Snippet 1:** Initialization list in Constructors Syntax

As shown in a code snippet 1,

1. A constructor is written first and then the initializations section is written
2. In the initialization section, the data members are initialized

To demonstrate the concept of Initialization list in Constructors an example program is shown below,

class Test

{

int a;

int b;

public:

Test(int i, int j) : a(i), b(j)

{

cout << "Constructor executed"<<endl;

cout << "Value of a is "<<a<<endl;

cout << "Value of b is "<<b<<endl;

}

};

int main()

{

Test t(4, 6);

return 0;

}

Copy

**Code Snippet 2:** Initialization list in Constructors Example Program 1

As shown in code snippet 2,

1. We have created a “test” class that consists of private data member “a” and “b” and parameterized constructor which takes two arguments and sets the value of data member “a” and “b” by using the initialization list. The constructor will also print the value of data member “a” and “b”.
2. In the main program object “t” is created of the “test” data type and the values (4, 6) are passed.

The output of the following program is shown below,

Text

Description automatically generated

**Figure 1:** Program Output

**Main Points**

The main thing to note here is that if we use the code shown below to initialize data members the compiler will throw an error because the data member “a” is being initialized first and the “b” is being initialized second so we have to assign the value to “a” data member first.

Test(int i, int j) : b(j), a(i+b)

Copy

**Code Snippet 3:**Initialization list in Constructors Example 1

But if we use the code shown below to initialize data members the compiler will not throw an error because the data member “a” is being initialized first and we are assigning the value to the data member “a” first.

Test(int i, int j) : a(i), b(a + j)

Copy

**Code Snippet 4:** Initialization list in Constructors Example 2

**Revisiting Pointers: new and delete Keywords in CPP | C++ Tutorials for Beginners #50**

In this tutorial, we will discuss pointers and new, delete keywords in C++

**Pointers in C++**

Pointers are variables that are used to store the address. Pointers are created using “\*”. An example program of pointers is shown below

#include<iostream>

using namespace std;

int main(){

// Basic Example

int a = 4;

int\* ptr = &a;

cout<<"The value of a is "<<\*(ptr)<<endl;

return 0;

}

Copy

**Code Snippet 1:**Pointer Example Program 1

As shown in a code snippet 1,

1. We created an integer variable “a” and assign the value “4” to it
2. We created an integer pointer “ptr” and assign the address of variable “a”
3. And printed the value at the address of pointer “ptr”

The output of the following program is shown below,



**Figure 1:** Pointer Program 1 Output

As shown in figure 1, we get the output value “4” because pointer “ptr” is pointing to the variable “a” and the value of the variable “a” is “4” that is why we get the output “4”.

**New Keyword**

Another example program for pointers and the use of a “new” keyword is shown below.

#include<iostream>

using namespace std;

int main(){

float \*p = new float(40.78);

cout << "The value at address p is " << \*(p) << endl;

return 0;

}

Copy

**Code Snippet 2:**Pointer Example Program 2

As shown in code snippet 2,

1. We created a float pointer “p” and dynamically created a float which has value “40.78” and assigned that value to pointer “p”
2. And printed the value at the address of pointer “p”

The output of the following program is shown below,



**Figure 2:**Pointer Program 2 Output

As shown in figure 2, we get the output value “40.78” because pointer “p” is pointing to an address whose value is “40.78”.

Another example program for pointers array and the use of a “new” keyword with an array is shown below.

#include<iostream>

using namespace std;

int main(){

int \*arr = new int[3];

arr[0] = 10;

arr[1] = 20;

arr[2] = 30;

cout << "The value of arr[0] is " << arr[0] << endl;

cout << "The value of arr[1] is " << arr[1] << endl;

cout << "The value of arr[2] is " << arr[2] << endl;

return 0;

}

Copy

**Code Snippet 3:**Pointer Example Program 3

As shown in a code snippet 3,

1. We created an integer pointer “arr” and dynamically created an array of size three which is assigned to the pointer “arr”
2. The values “10”, “20”, and “30” are assigned to the ”1”, “2”, and “3” indexes of an array
3. And printed the value at the array indexes “1”, “2”, and “3”

The output of the following program is shown below,

A picture containing text, electronics, close

Description automatically generated

**Figure 3:**Pointer Program 2 Output

As shown in figure 3, we get the output values “10”, “20”, and “30”.

**Delete Keyword**

Another example program for pointers array and the use of the “delete” keyword with an array is shown below.

#include<iostream>

using namespace std;

int main(){

int \*arr = new int[3];

arr[0] = 10;

arr[1] = 20;

arr[2] = 30;

delete[] arr;

cout << "The value of arr[0] is " << arr[0] << endl;

cout << "The value of arr[1] is " << arr[1] << endl;

cout << "The value of arr[2] is " << arr[2] << endl;

return 0;

}

Copy

**Code Snippet 4:** Pointer Example Program 4

As shown in code snippet 4,

1. We created an integer pointer “arr” and dynamically created an array of size three which is assigned to the pointer “arr”
2. The values “10”, “20”, and “30” are assigned to the ”1”, “2”, and “3” indexes of an array
3. Before printing the values we used the “delete” keyword
4. And printed the value at the array indexes “1”, “2”, and “3”

The output of the following program is shown below,

Text

Description automatically generated

**Figure 4:** Pointer Program 2 Output

As shown in figure 2, we get the garbage value in the output instead of “10”, “20”, and “30” because we have used “delete” keyword before printing the values due to which the space used by an array gets free and we get the garbage value in return.

**Pointers to Objects and Arrow Operator in CPP | C++ Tutorials for Beginners #51**

In this tutorial, we will discuss pointers to objects and arrow operator in C++

**Pointer to objects in C++**

As discussed before pointers are used to store addresses of variables which have data types like int, float, double etc. But pointer can also store the address of an object. An example program is shown below to demonstrate the concept of pointer to objects.

#include<iostream>

using namespace std;

class Complex{

int real, imaginary;

public:

void getData(){

cout<<"The real part is "<< real<<endl;

cout<<"The imaginary part is "<< imaginary<<endl;

}

void setData(int a, int b){

real = a;

imaginary = b;

}

};

int main(){

Complex \*ptr = new Complex;

(\*ptr).setData(1, 54); is exactly same as

(\*ptr).getData(); is as good as

return 0;

}

Copy

**Code Snippet 1:** Pointer to objects Example Program 1

As shown in a code snippet 1,

1. We created a class “Complex”, which contains two private data members “real” and “imaginary”.
2. The class “complex” contains two member functions “getdata” and “setdata”
3. The Function “setdata” will take two parameters and assign the values of parameters to the private data members “real” and “imaginary”
4. The Function “getdata” will print the values of private data members “real” and “imaginary”
5. In the main program object is created dynamically by using the “new” keyword and its address is assigned to the pointer “ptr”
6. The member function “setdata” is called using the pointer “ptr” and the values “1, 54” are passed.
7. The member function “getdata” is called using the pointer “ptr” and it will print the values of data members.

The main thing to note here is that we called the member function with pointers instead of object but still it will give same result because pointer is pointing to the address of that object.

The output of the following program is shown below,



**Figure 1:**Pointer to Objects Program 1 Output

**Arrow Operator in C++**

Another example program for the pointer to Objects and the use of the “Arrow” Operator is shown below.

#include<iostream>

using namespace std;

class Complex{

int real, imaginary;

public:

void getData(){

cout<<"The real part is "<< real<<endl;

cout<<"The imaginary part is "<< imaginary<<endl;

}

void setData(int a, int b){

real = a;

imaginary = b;

}

};

int main(){

Complex \*ptr = new Complex;

ptr->setData(1, 54);

ptr->getData();

// Array of Objects

Complex \*ptr1 = new Complex[4];

ptr1->setData(1, 4);

ptr1->getData();

return 0;

}

Copy

**Code Snippet 2:** Pointer to Objects with Arrow Operator Example Program 2

As shown in code snippet 2,

1. We created a class “Complex”, which contains two private data members “real” and “imaginary”.
2. The class “complex” contains two member functions “getdata” and “setdata”
3. The Function “setdata” will take two parameters and assign the values of parameters to the private data members “real” and “imaginary”
4. The Function “getdata” will print the values of private data members “real” and “imaginary”
5. In the main program object is created dynamically by using the “new” keyword and its address is assigned to the pointer “ptr”
6. The member function “setdata” is called using the pointer “ptr” with the arrow operator “->” and the values “1, 54” are passed.
7. The member function “getdata” is called using the pointer “ptr” with the arrow operator “->” and it will print the values of data members.
8. Array of objects is created dynamically by using the “new” keyword and its address is assigned to the pointer “ptr1”
9. The member function “setdata” is called using the pointer “ptr1” with the arrow operator “->” and the values “1, 4” are passed.
10. The member function “getdata” is called using the pointer “ptr1” with the arrow operator “->” and it will print the values of data members.

The main thing to note here is that we called the member function with pointers by using arrow operator “->” instead of the dot operator “.” but still it will give the same results.

The output of the following program is shown below,

Text

Description automatically generated

**Figure 2:**Pointer to Objects Program 2 Output

**Array of Objects Using Pointers in C++ | C++ Tutorials for Beginners #52**

In this tutorial, we will discuss an array of objects using pointers in C++

**Array of Objects Using Pointers in C++**

Array of objects can be defined as an array that’s each element is an object of the class. In this tutorial, we will use the pointer to store the address of an array of objects.  An example program is shown below to demonstrate the concept of an array of objects using pointers.

#include<iostream>

using namespace std;

class ShopItem

{

int id;

float price;

public:

void setData(int a, float b){

id = a;

price = b;

}

void getData(void){

cout<<"Code of this item is "<< id<<endl;

cout<<"Price of this item is "<<price<<endl;

}

};

Copy

**Code Snippet 1:**Array of Objects Using Pointers Example Program

As shown in a code snippet 1,

1. We created a class “ShopItem”, which contains two private data members “id” and “price”.
2. The class “ShopItem” contains two member functions “setdata” and “getdata”
3. The Function “setdata” will take two parameters and assign the values of parameters to the private data members “id” and “price”
4. The Function “getdata” will print the values of private data members “id” and “price”

int main(){

int size = 3;

ShopItem \*ptr = new ShopItem [size];

ShopItem \*ptrTemp = ptr;

int p, i;

float q;

for (i = 0; i < size; i++)

{

cout<<"Enter Id and price of item "<< i+1<<endl;

cin>>p>>q;

// (\*ptr).setData(p, q);

ptr->setData(p, q);

ptr++;

}

for (i = 0; i < size; i++)

{

cout<<"Item number: "<<i+1<<endl;

ptrTemp->getData();

ptrTemp++;

}

return 0;

}

Copy

**Code Snippet 2:** Main Program

As shown in code snippet 2,

1. We created an integer variable “size” and assigned the value “3” to it.
2. Array of objects of size “3” is created dynamically by using the “new” keyword and its address is assigned to the pointer “ptr”
3. The address of pointer “ptr” is assigned to another pointer “ptrTemp”
4. Two integer variables “p” and “i” are declared and one float variable ”q” is declared
5. We created a “for” loop which will run till the size of array and will take input for “id” and “price” from user at run time. In this “for” loop “setdata” function is called using pointer “ptr”; the function will set the values of “id” and “price” which user will enter. The value of the pointer “ptr” is incremented by 1 in every iteration of loop.
6. We created another “for” loop which will run till the size of array and will print the number of the item. In this “for” loop “getdata” function is called using pointer “ptr”; the function will print the values of “id” and “price”. The value of the pointer “ptrTemp” is incremented by 1 in every iteration of loop.

The main thing to note here is that in the first “for” loop we are incrementing the value of the pointer “ptr” because it is pointing to the address of array of objects and when loop will run every time the function “setdata” will be called by the different object. If we don’t increment the value of the pointer “ptr” the each time function “setdata” will be called by the same object. Likewise in the second loop we are incrementing the pointer “ptrTemp” so that the function “getdata” could be called by each object in the array.

The input and output of the following program is shown below,

Text

Description automatically generated

**Figure 1:**Array of Objects Using Pointer Program Input

Text

Description automatically generated

**Figure 2:**Array of Objects Using Pointer Program Output

**this Pointer in C++ | C++ Tutorials for Beginners #53**

In this tutorial, we will discuss ‘this’ pointer in C++

**‘this’ Pointer in C++**

“this” is a keyword that is an implicit pointer. “this” pointer points to the object which calls the member function. An example program is shown below to demonstrate the concept of “this” pointer.

#include<iostream>

using namespace std;

class A{

int a;

public:

void setData(int a){

this->a = a;

}

void getData(){

cout<<"The value of a is "<<a<<endl;

}

};

Copy

**Code Snippet 1:**“this” Pointer Example Program

As shown in a code snippet 1,

1. We created a class “A”, which contains private data members “a”.
2. The class “A” contains two member functions “setData” and “getData”
3. The Function “setData” will take one parameters and assign the values of parameter to the private data members “a” using “this” pointer. As we know that one copy of member function is shared between all object. The use of “this” pointer helps to points to the object which invokes the member function.
4. The Function “getData” will print the values of private data members “a”

The code for the main program is shown below,

int main(){

A a;

a.setData(4);

a.getData();

return 0;

}

Copy

**Code Snippet 2:** Main Program

As shown in code snippet 2,

1. Object “a” is of data type “A” is created
2. The function “setData” is called using object “a” and the value “4” is passed to the function
3. The function “getData” is called using object “a”

The input and output of the following program is shown below,



**Figure 1:** Program Output

“this” pointer can be used to return a reference to the invoking object. An example program is shown below.

class A{

int a;

public:

A & setData(int a){

this->a = a;

return \*this;

}

void getData(){

cout<<"The value of a is "<<a<<endl;

}

};

int main(){

A a;

a.setData(4).getData();

return 0;

}

Copy

**Code Snippet 3:** Return Reference to Invoking Object Example Program

As shown in Code Snippet 3,

1. In the function “setData” the reference of the object is returned using “this” pointer.
2. In the main program by using a single object we have made a chain of the function calls. The main thing to note here is that the function “setData” is returning an object on which we have used the “getData” function. so we don’t need to call the function “getData” explicitly.

**Polymorphism in C++ | C++ Tutorials for Beginners #54**

In this tutorial, we will discuss polymorphism in C++

**Polymorphism in C++**

“Poly” means several and “morphism” means form. So we can say that polymorphism is something that has several forms or we can say it as one name and multiple forms. There are two types of polymorphism:

* Compile-time polymorphism
* Run time polymorphism

**Compile Time Polymorphism**

In compile-time polymorphism, it is already known which function will run. Compile-time polymorphism is also called early binding, which means that you are already bound to the function call and you know that this function is going to run. There are two types of compile-time polymorphism:

1. Function Overloading

This is a feature that lets us create more than one function and the functions have the same names but their parameters need to be different. If function overloading is done in the program and function calls are made the compiler already knows that which functions to execute.

1. Operator Overloading

This is a feature that lets us define operators working for some specific tasks. For example, we can overload the operator “+” and define its functionality to add two strings. Operator loading is also an example of compile-time polymorphism because the compiler already knows at the compile time which operator has to perform the task.

**Run Time Polymorphism**

In the run-time polymorphism, the compiler doesn’t know already what will happen at run time. Run time polymorphism is also called late binding. The run time polymorphism is considered slow because function calls are decided at run time. Run time polymorphism can be achieved from the virtual function.

1. Virtual Function

A function that is in the parent class but redefined in the child class is called a virtual function. “virtual” keyword is used to declare a virtual function.

**Pointers to Derived Classes in C++ | C++ Tutorials for Beginners #55**

In this tutorial, we will discuss pointer to derived class in C++

**Pointer to Derived Class in C++**

In C++ we are provided with the functionality to point the pointer to derived class or base class. An example program is shown below to demonstrate the concept of pointer to a derived class in C++

#include<iostream>

using namespace std;

class BaseClass{

public:

int var\_base;

void display(){

cout<<"Dispalying Base class variable var\_base "<<var\_base<<endl;

}

};

class DerivedClass : public BaseClass{

public:

int var\_derived;

void display(){

cout<<"Dispalying Base class variable var\_base "<<var\_base<<endl;

cout<<"Dispalying Derived class variable var\_derived "<<var\_derived<<endl;

}

};

Copy

**Code Snippet 1:**Pointer to Derived Class Program Example

As shown in Code snippet 1,

1. We created a class “BaseClass” which contains public data member “var\_base” and member function “display”. The member function “display” will print the value of data member “var\_base”
2. We created another class “DerivedClass” which is inheriting “BaseClass” and contains data member “var\_derived” and member function “display”. The member function “display” will print the values of data members “var\_base” and “var\_derived”

The code for the main program is shown below,

int main(){

BaseClass \* base\_class\_pointer;

BaseClass obj\_base;

DerivedClass obj\_derived;

base\_class\_pointer = &obj\_derived; // Pointing base class pointer to derived class

base\_class\_pointer->var\_base = 34;

// base\_class\_pointer->var\_derived= 134; // Will throw an error

base\_class\_pointer->display();

base\_class\_pointer->var\_base = 3400;

base\_class\_pointer->display();

DerivedClass \* derived\_class\_pointer;

derived\_class\_pointer = &obj\_derived;

derived\_class\_pointer->var\_base = 9448;

derived\_class\_pointer->var\_derived = 98;

derived\_class\_pointer->display();

return 0;

}

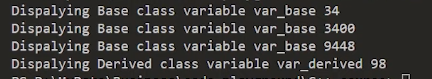
Copy

**Code Snippet 2:**Main Program

As shown in code snippet 2,

1. We created a pointer “base\_class\_pointer” of the data type “Baseclass”.
2. Object “obj\_base” of the data type “BaseClass” is created.
3. Object “obj\_derived” of the data type “DerivedClass” is created
4. Pointer “base\_class\_pointer” of the base class is pointing to the object “obj\_derived” of the derived class
5. By using the pointer “base\_class\_pointer” of the base class we have set the value of the data member “var\_base” by “34”. The main thing to note here is that we cannot set the value of the derived class data member by using the base class pointer otherwise the compiler will throw an error.
6. The function “display” is called using a base class pointer. The main thing to note here is that the base class “display” function will run here.
7. Again by using the pointer “base\_class\_pointer” of the base class we have set the value of the data member “var\_base” by “3400” which will update the previous value and the function “display” is called.
8. We created a pointer “derived\_class\_pointer” of the data type “DerivedClass”
9. Pointer “Derived\_class\_pointer” of the derived class is pointing to the object “obj\_derived” of the derived class
10. By using pointer “Derived\_class\_pointer” of the derived class we have set the value of the data member “var\_base” of the base class by “9448”. The main thing to note here is that this will not throw an error because we can set the value of base class data member by using derived class pointer but we cannot set the value of derived class data member by using base class pointer
11. By using pointer “Derived\_class\_pointer” of the derived class we have set the value of the data member “var\_derived” of the derived class by “98”.
12. The function “display” is called using a derived class pointer. The main thing to note here is that the derived class “display” function will run here.

The output of the following program is shown in figure 1,



**Figure 1:**Program Output

**Virtual Functions in C++ | C++ Tutorials for Beginners #56**

In this tutorial, we will discuss virtual functions in C++

**Virtual Functions in C++**

A member function in the base class which is declared using virtual keyword is called virtual functions. They can be redefined in the derived class. To demonstrate the concept of virtual functions an example program is shown below

#include<iostream>

using namespace std;

class BaseClass{

public:

int var\_base=1;

virtual void display(){

cout<<"1 Dispalying Base class variable var\_base "<<var\_base<<endl;

}

};

class DerivedClass : public BaseClass{

public:

int var\_derived=2;

void display(){

cout<<"2 Dispalying Base class variable var\_base "<<var\_base<<endl;

cout<<"2 Dispalying Derived class variable var\_derived "<<var\_derived<<endl;

}

};

Copy

***Code Snippet 1: Virtual Function Example Program***

As shown in code snippet 1,

1. We created a class “BaseClass” which contains public data member “var\_base” which has the value “1” and member function “display”. The member function “display” will print the value of data member “var\_base”
2. We created another class “DerivedClass” which is inheriting “BaseClass” and contains data member “var\_derived” which has the value “2” and member function “display”. The member function “display” will print the values of data members “var\_base” and “var\_derived”

The code for the main program is shown below

int main(){

BaseClass \* base\_class\_pointer;

BaseClass obj\_base;

DerivedClass obj\_derived;

base\_class\_pointer = &obj\_derived;

base\_class\_pointer->display();

return 0;

}

Copy

***Code Snippet 2: Main Program***

As shown in code snippet 2,

1. We created a pointer “base\_class\_pointer” of the data type “Baseclass”
2. Object “obj\_base” of the data type “BaseClass” is created.
3. Object “obj\_derived” of the data type “DerivedClass” is created
4. Pointer “base\_class\_pointer” of the base class is pointing to the object “obj\_derived” of the derived class
5. The pointer “base\_class\_pointer” is pointed to the object “obj\_derived” of the derived class.
6. The function “display” is called using the pointer “base\_class\_pointer” of the base class.

The main thing to note here is that if we don’t use the “virtual” keyword with the “display” function of the base class then beside of the point that we have pointed our base call pointer to derived class object still the compiler would have called the “display” function of the base class because this is its default behavior as we have seen in the previous tutorial.

But we have used the “virtual” keyword with the “display” function of the base class to make is **virtual function**so when the display function is called by using the base class pointer the display function of the derived class will run because the base class pointer is pointing to the derived class object.

The output of the following program is shown in figure 1

**Virtual Functions Example + Creation Rules in C++ | C++ Tutorials for Beginners #57**

In this tutorial, we will discuss virtual functions example and its creation rules in C++

**Virtual Functions Example in C++**

As we have seen in the previous tutorial that how virtual functions are used to implement run-time polymorphism. In this tutorial, we will see an example of virtual functions.

class CWH{

protected:

string title;

float rating;

public:

CWH(string s, float r){

title = s;

rating = r;

}

virtual void display(){}

};

Copy

***Code Snippet 1: Code with Harry Class***

As shown in a code snippet 1,

1. We created a class “CHW” which contains protected data members “title” which has a “string” data type and “rating” which has a “float” data type.
2. The class “CWH” has a parameterized constructor which takes two parameters “s” and “r” and assign their values to the data members “title” and “rating”
3. The class “CHW” has a virtual function void “display” which does nothing

class CWHVideo: public CWH

{

float videoLength;

public:

CWHVideo(string s, float r, float vl): CWH(s, r){

videoLength = vl;

}

void display(){

cout<<"This is an amazing video with title "<<title<<endl;

cout<<"Ratings: "<<rating<<" out of 5 stars"<<endl;

cout<<"Length of this video is: "<<videoLength<<" minutes"<<endl;

}

};

Copy

***Code Snippet 2: Code with Harry Video Class***

As shown in a code snippet 2,

1. We created a class “CHWVideo” which is inheriting the “CWH” class and contains private data members “videoLength” which has a “float” data type.
2. The class “CWHVideo” has a parameterized constructor which takes three parameters “s”, “r” and “vl”. The constructor of the base class is called in the derived class and the values of the variables “s” and “r” are passed to it. The value of the parameter “vl” will be assigned  to the data members “videoLength”
3. The class “CHWVideo” has a function void “display” which will print the values of the data members “title”, “rating” and “videoLength”

class CWHText: public CWH

{

int words;

public:

CWHText(string s, float r, int wc): CWH(s, r){

words = wc;

}

void display(){

cout<<"This is an amazing text tutorial with title "<<title<<endl;

cout<<"Ratings of this text tutorial: "<<rating<<" out of 5 stars"<<endl;

cout<<"No of words in this text tutorial is: "<<words<<" words"<<endl;

}

};

Copy

***Code Snippet 3: Code with Harry Text Class***

As shown in a code snippet 3,

1. We created a class “CHWText” which is inheriting the “CWH” class and contains private data members “words” which has an “int” data type.
2. The class “CWHText” has a parameterized constructor which takes three parameters “s”, “r” and “wc”. The constructor of the base class is called in the derived class and the values of the variables “s” and “r” are passed to it. The value of the parameter “wc” will be assigned  to the data members “words”
3. The class “CHWText” has a function void “display” which will print the values of the data members “title”, “rating” and “words”

int main(){

string title;

float rating, vlen;

int words;

// for Code With Harry Video

title = "Django tutorial";

vlen = 4.56;

rating = 4.89;

CWHVideo djVideo(title, rating, vlen);

// for Code With Harry Text

title = "Django tutorial Text";

words = 433;

rating = 4.19;

CWHText djText(title, rating, words);

CWH\* tuts[2];

tuts[0] = &djVideo;

tuts[1] = &djText;

tuts[0]->display();

tuts[1]->display();

return 0;

}

Copy

***Code Snippet 4: Main Program***

As shown in a code snippet 4,

1. We created a string variable “title”, float variables “rating”, “vlen” and integer variable “words”
2. For the code with harry video class, we have assigned “Django tutorial” to the string “title”, “4.56” to the float “vlen” and “4.89” to the float “rating”.
3. An object “djVideo” is created of the data type “CWHVideo” and the variables “title”, “rating” and “vlen” are passed to it.
4. For the code with harry text class, we have assigned “Django tutorial text” to the string “title”, “433” to the integer “words” and “4.19” to the float “rating”.
5. An object “djText” is created of the data type “CWHText” and the variables “title”, “rating” and “words” are passed to it.
6. Two pointers array “tuts” is created of the “CWH” type
7. The address of the “djVideo” is assigned to “tuts[0]” and the address of the “djText” is assigned to “tuts[1]”
8. The function “display” is called using pointers “tuts[0]” and “tuts[1]”

The main thing to note here is that if we don’t use the “virtual” keyword with the “display” function of the base class then the “display” function of the base class will run.

But we have used the “virtual” keyword with the “display” function of the base class to make is a **virtual function**so when the display function is called by using the base class pointer the display function of the derived class will run because the base class pointer is pointing to the derived class object.

The output of the following program is shown in figure 1

Text

Description automatically generated

***Figure 1: Program Output***

**Rules for virtual functions**

1. They cannot be static
2. They are accessed by object pointers
3. Virtual functions can be a friend of another class
4. A virtual function in the base class might not be used.
5. If a virtual function is defined in a base class, there is no necessity of redefining it in the derived class

**Abstract Base Class & Pure Virtual Functions in C++ | C++ Tutorials for Beginners #58**

In this tutorial, we will discuss abstract base class and pure virtual functions in C++

**Pure Virtual Functions in C++**

Pure virtual function is a function that doesn’t perform any operation and the function is declared by assigning the value 0 to it. Pure virtual functions are declared in abstract classes.

**Abstract Base Class in C++**

Abstract base class is a class that has at least one pure virtual function in its body. The classes which are inheriting the base class must need to override the virtual function of the abstract class otherwise compiler will throw an error.

To demonstrate the concept of abstract class and pure virtual function an example program is shown below.

class CWH{

protected:

string title;

float rating;

public:

CWH(string s, float r){

title = s;

rating = r;

}

virtual void display()=0;

};

Copy

***Code Snippet 1: Code with Harry Class***

As shown in code snippet 1,

1. We created a class “CHW” which contains protected data members “title” which has “string” data type and “rating” which has “float” data type.
2. The class “CWH” has a parameterized constructor which takes two parameters “s” and “r” and assign their values to the data members “title” and “rating”
3. The class “CHW” has a pure virtual function void “display” which is declared by 0. The main thing to note here is that as the “display” function is a pure virtual function it is compulsory to redefine it in the derived classes.

class CWHVideo: public CWH

{

float videoLength;

public:

CWHVideo(string s, float r, float vl): CWH(s, r){

videoLength = vl;

}

void display(){

cout<<"This is an amazing video with title "<<title<<endl;

cout<<"Ratings: "<<rating<<" out of 5 stars"<<endl;

cout<<"Length of this video is: "<<videoLength<<" minutes"<<endl;

}

};

Copy

***Code Snippet 2: Code with Harry Video Class***

As shown in code snippet 2,

1. We created a class “CHWVideo” which is inheriting “CWH” class and contains private data members “videoLength” which has “float” data type.
2. The class “CWHVideo” has a parameterized constructor which takes three parameters “s”, “r” and “vl”. The constructor of the base class is called in the derived class and the values of the variables “s” and “r” are passed to it. The value of the parameter “vl” will be assigned  to the data members “videoLength”
3. The class “CHWVideo” has a function void “display” which will print the values of the data members “title”, “rating” and “videoLength”

class CWHText: public CWH

{

int words;

public:

CWHText(string s, float r, int wc): CWH(s, r){

words = wc;

}

void display(){

cout<<"This is an amazing text tutorial with title "<<title<<endl;

cout<<"Ratings of this text tutorial: "<<rating<<" out of 5 stars"<<endl;

cout<<"No of words in this text tutorial is: "<<words<<" words"<<endl;

}

};

Copy

***Code Snippet 3: Code with Harry Text Class***

As shown in code snippet 3,

1. We created a class “CHWText” which is inheriting “CWH” class and contains private data members “words” which has “int” data type.
2. The class “CWHText” has a parameterized constructor which takes three parameters “s”, “r” and “wc”. The constructor of the base class is called in the derived class and the values of the variables “s” and “r” are passed to it. The value of the parameter “wc” will be assigned  to the data members “words”
3. The class “CHWText” has a function void “display” which will print the values of the data members “title”, “rating” and “words”

int main(){

string title;

float rating, vlen;

int words;

// for Code With Harry Video

title = "Django tutorial";

vlen = 4.56;

rating = 4.89;

CWHVideo djVideo(title, rating, vlen);

// for Code With Harry Text

title = "Django tutorial Text";

words = 433;

rating = 4.19;

CWHText djText(title, rating, words);

CWH\* tuts[2];

tuts[0] = &djVideo;

tuts[1] = &djText;

tuts[0]->display();

tuts[1]->display();

return 0;

}

Copy

***Code Snippet 4: Main Program***

As shown in code snippet 4,

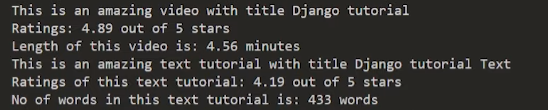
1. We created a string variable “title”, float variables “rating”, “vlen” and integer variable “words”
2. For the code with harry video class we have assigned “Django tutorial” to the string “title”, “4.56” to the float “vlen” and “4.89” to the float “rating”.
3. An object “djVideo” is created of the data type “CWHVideo” and the variables “title”, “rating” and “vlen” are passed to it.
4. For the code with harry text class we have assigned “Django tutorial text” to the string “title”, “433” to the integer “words” and “4.19” to the float “rating”.
5. An object “djText” is created of the data type “CWHText” and the variables “title”, “rating” and “words” are passed to it.
6. Two pointers array “tuts” is created of the “CWH” type
7. The address of the “djVideo” is assigned to “tuts[0]” and the address of the “djText” is assigned to “tuts[1]”
8. The function “display” is called using pointers “tuts[0]” and “tuts[1]”

The main thing to note here is that if we don’t override the pure virtual function in the derived class the compiler will throw an error as shown in figure 1.



***Figure 1: Program Error***

The output of the following program is shown in figure 2



***Figure 2: Program Output***