**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,

https://cwh-full-next-space.fra1.digitaloceanspaces.com/videos/cpp-tutorials-in-hindi-50/Pointer_Program_1_Output.webp

**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,

https://cwh-full-next-space.fra1.digitaloceanspaces.com/videos/cpp-tutorials-in-hindi-50/Pointer_Program_2_Output.webp

**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;

}

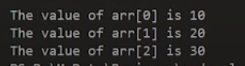
Copy

**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,



**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,

https://cwh-full-next-space.fra1.digitaloceanspaces.com/videos/cpp-tutorials-in-hindi-50/Pointer_Program_4_Output.webp

**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,

https://cwh-full-next-space.fra1.digitaloceanspaces.com/videos/cpp-tutorials-in-hindi-51/Pointer_to_Objects_Program_1_Output.webp

**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

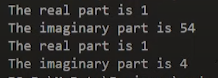
**Code Snippet 2:** Pointer to Objects with Arrow Operator Example Program 2

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,



**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

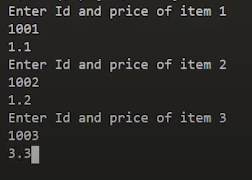
**Code Snippet 2:** Main Program

As shown in code snippet 2,

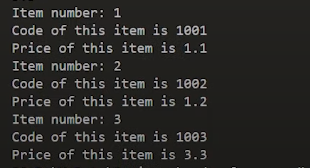
1. We created an integer variable “size” and assigned the value “3” to it.
2. Array of objects of size “3” is created dynamically by using the “new” keyword and its address is assigned to the pointer “ptr”
3. The address of pointer “ptr” is assigned to another pointer “ptrTemp”
4. Two integer variables “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,



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



**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,

https://cwh-full-next-space.fra1.digitaloceanspaces.com/videos/cpp-tutorials-in-hindi-53/Program_output.webp

**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;

}

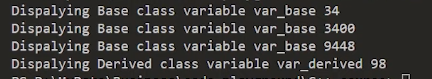
Copy

**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

https://cwh-full-next-space.fra1.digitaloceanspaces.com/videos/cpp-tutorials-in-hindi-56/base64.webp

***Figure 1: Program Output***

**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***

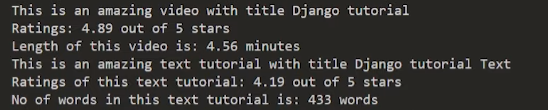
As shown in a code snippet 4,

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



***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,

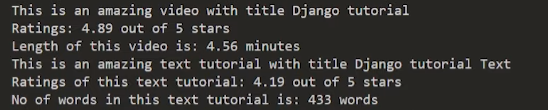
1. We created a string variable “title”, float variables “rating”, “vlen” and integer variable “words”
2. For the code with harry video class we have assigned “Django tutorial” to the string “title”, “4.56” to the float “vlen” and “4.89” to the float “rating”.
3. An object “djVideo” is created of the 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.

https://cwh-full-next-space.fra1.digitaloceanspaces.com/videos/cpp-tutorials-in-hindi-58/base64.webp

***Figure 1: Program Error***

The output of the following program is shown in figure 2



***Figure 2: Program Output***