OOP
Lecture 27

27/01/2022

typeid operator :

typeid is an operator in C++. It is used to get the runtime data type info of any constant =, variable or object. It is included in the **<typeinfo>** library. Hence in order to use the **typeid** operator, this library should be included in the program. i.e **#include <typeinfo>.** We must use this operator in an if condition, to verify the type information of any expression or object.

Parameters :

typeid operator accepts a parameter, based on the syntax used in the program:

type: This parameter is passed when the runtime type information of a variable or an object is needed. In this, there is no evaluation that needs to be done inside type and simply the type information is to be known.

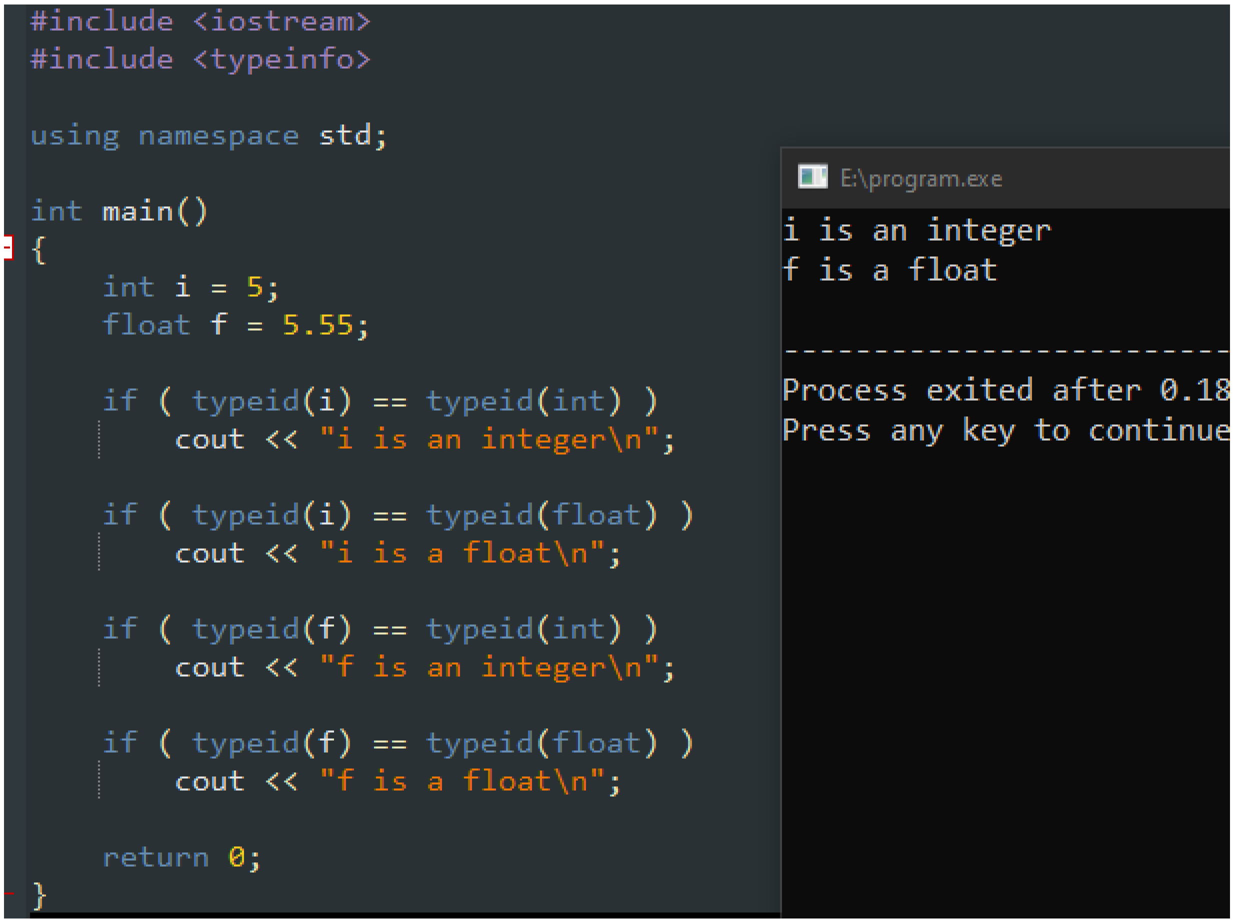
expression: This parameter is passed when the runtime type information of an expression is needed. In this, the expression is first evaluated. Then the type information of the final result is then provided.

Syntax :

if (typeid(expression) == typeid(type))

Example :

if (typeid(5) == typeid(int))

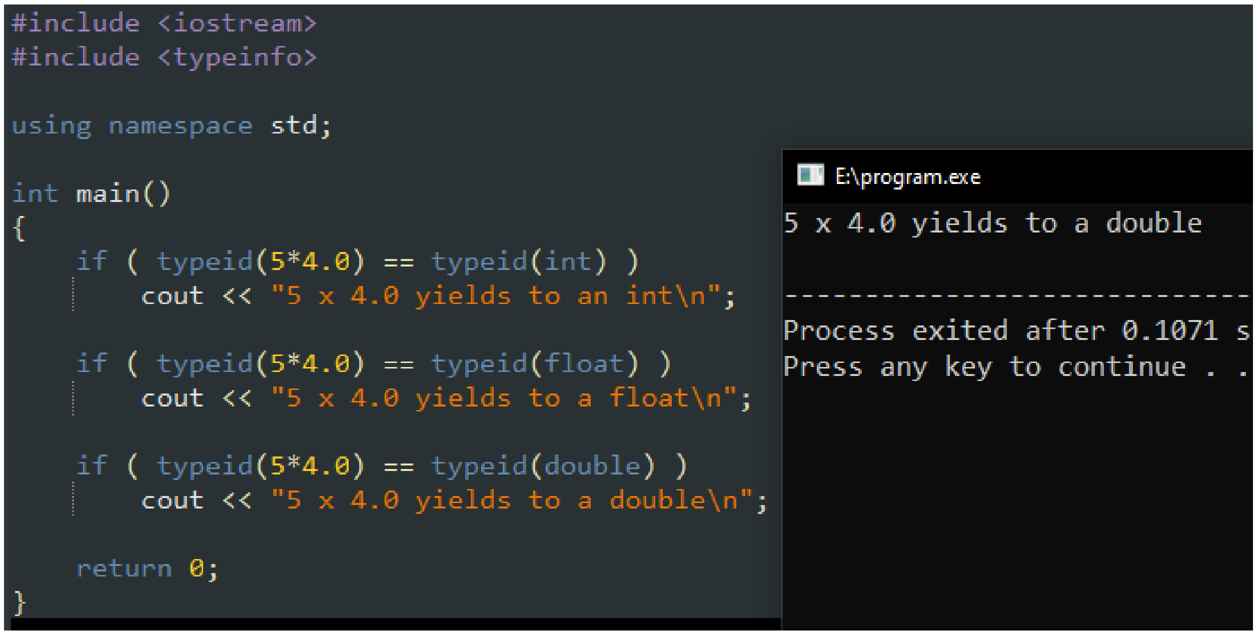


```cpp
#include <iostream>
#include <typeinfo>

using namespace std;

int main()
{
    int i = 5;
    float f = 5.55;

    if ( typeid(i) == typeid(int) )
        cout << "i is an integer\n";

    if ( typeid(i) == typeid(float) )
        cout << "i is a float\n";

    if ( typeid(f) == typeid(int) )
        cout << "f is an integer\n";

    if ( typeid(f) == typeid(float) )
        cout << "f is a float\n";

    return 0;
}
```

```
i is an integer
f is a float

---------------------------
Process exited after 0.18
Press any key to continue
```

Numeric constants have some fixed data types in C++. It means that a number with no floating point are integers and the numbers with floating point even if we write it like this, 5.0, it will be considered as double.

In an expression, when different types of numeric constants or variables are used, there will be implicit type casting and the lower data type variable will be converted to the superior data type.

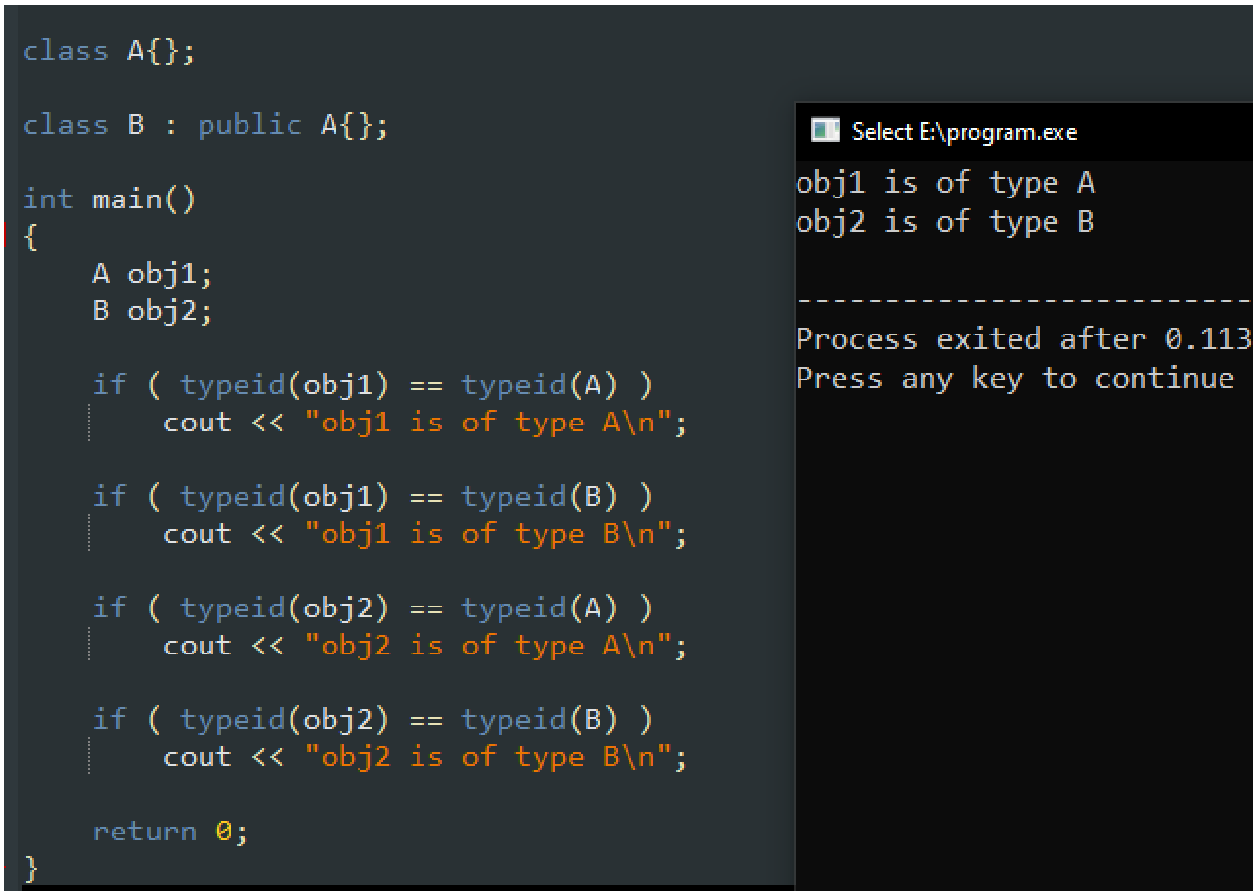


```cpp
#include <iostream>
#include <typeinfo>

using namespace std;

int main()
{
    if ( typeid(5*4.0) == typeid(int) )
        cout << "5 x 4.0 yields to an int\n";

    if ( typeid(5*4.0) == typeid(float) )
        cout << "5 x 4.0 yields to a float\n";

    if ( typeid(5*4.0) == typeid(double) )
        cout << "5 x 4.0 yields to a double\n";

    return 0;
}
```

E:\program.exe

```
5 x 4.0 yields to a double

-----------------------------
Process exited after 0.1071 s
Press any key to continue . .
```

The same principle applies on the objects of different types as well.

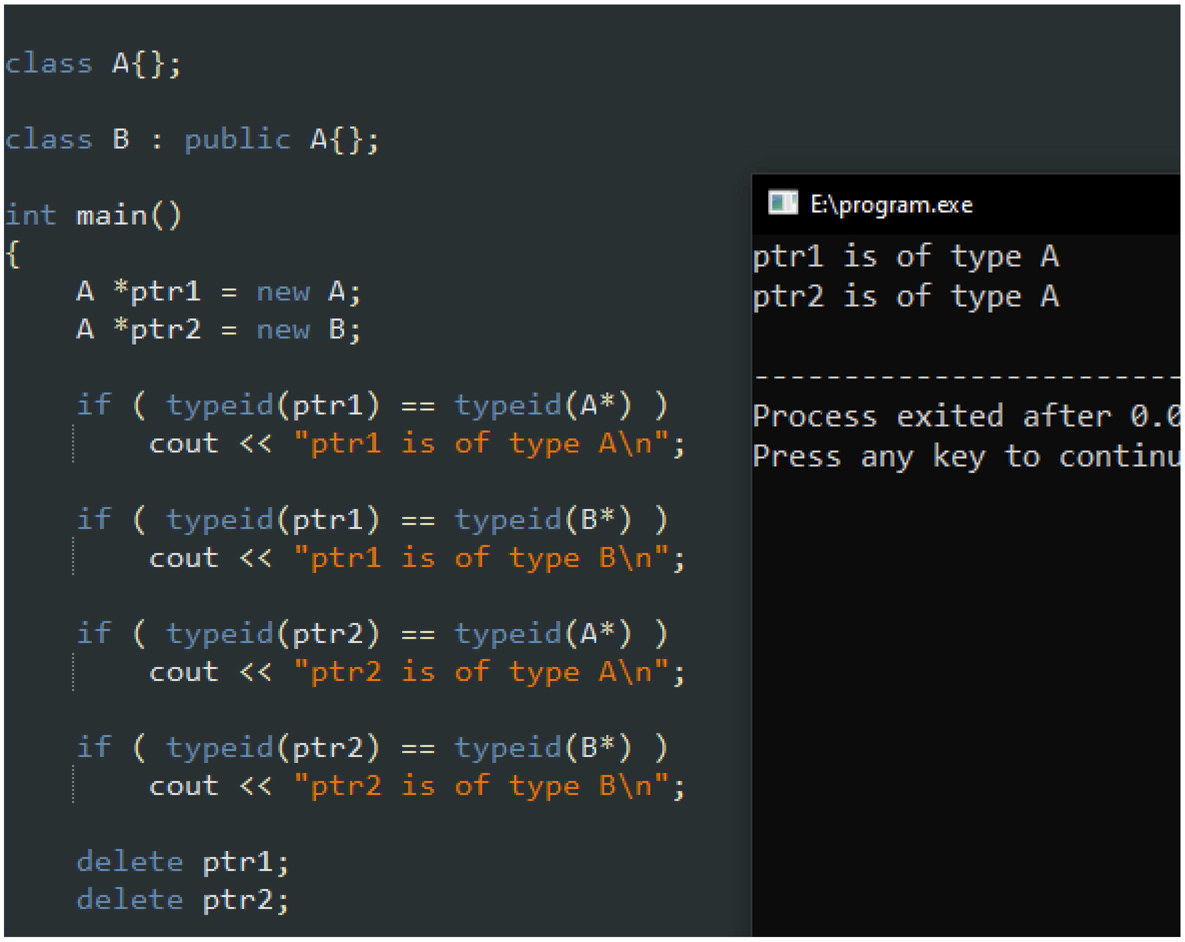


```cpp
class A{};

class B : public A{};

int main()
{
    A obj1;
    B obj2;

    if ( typeid(obj1) == typeid(A) )
        cout << "obj1 is of type A\n";

    if ( typeid(obj1) == typeid(B) )
        cout << "obj1 is of type B\n";

    if ( typeid(obj2) == typeid(A) )
        cout << "obj2 is of type A\n";

    if ( typeid(obj2) == typeid(B) )
        cout << "obj2 is of type B\n";

    return 0;
}
```

Select E:\program.exe

```
obj1 is of type A
obj2 is of type B

-------------------------
Process exited after 0.113
Press any key to continue
```

But the problem arises when we use pointers and when we come to the polymorphism,



```
class A{};

class B : public A{};

int main()
{
    A *ptr1 = new A;
    A *ptr2 = new B;

    if ( typeid(ptr1) == typeid(A*) )
        cout << "ptr1 is of type A\n";

    if ( typeid(ptr1) == typeid(B*) )
        cout << "ptr1 is of type B\n";

    if ( typeid(ptr2) == typeid(A*) )
        cout << "ptr2 is of type A\n";

    if ( typeid(ptr2) == typeid(B*) )
        cout << "ptr2 is of type B\n";

    delete ptr1;
    delete ptr2;
```

E:\program.exe

```
ptr1 is of type A
ptr2 is of type A

------------------------
Process exited after 0.0
Press any key to continu
```

typeid fails when we do polymorphism as it cannot check the object type of the pointer, rather it just checks the data type of the pointer and workers according to it. But, what if we need to access the child class function and the parent class has multiple childs? How do we know which parent class it belongs to? In the previous lecture we discussed type casting and its effects on the objects and pointers. In the last lecture we discussed only C-style type casting which is valid but not most accurate, as it just type casts the whole object into another data type, but what if it was not valid. We also discussed a scenario in the previous lecture, where we concluded that the C-style type is not always the solution. So, in this lecture we are going to learn dynamic casting.

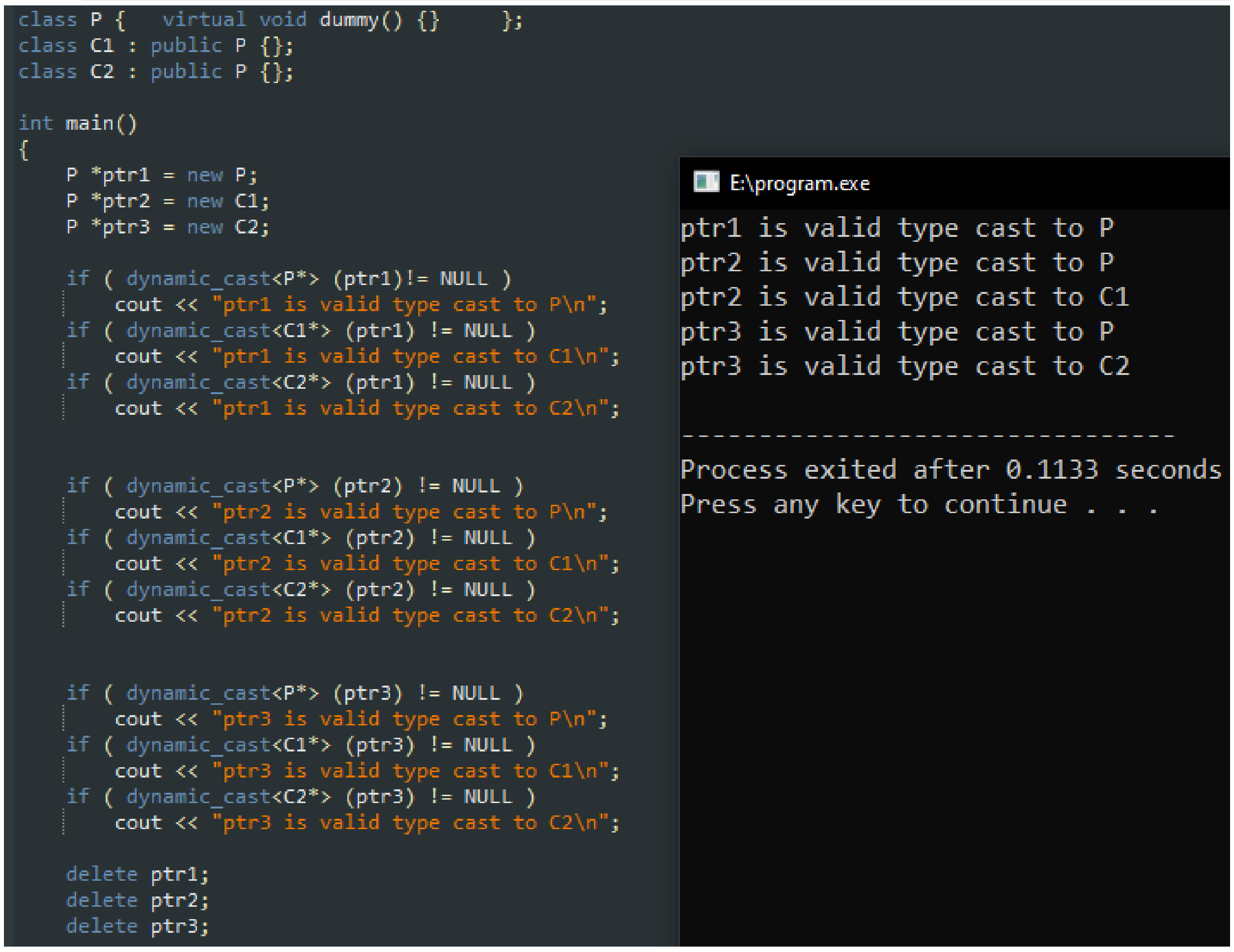
Dynamic Cast :

Dynamic cast is used in inheritance hierarchy It is used with polymorphism. It means that there must exist at least 1 virtual function in the parent class otherwise it will generate an error. The conversion of child to parent class of object is implicit type cast. And with the help of dynamic cast we can identify the type of parent class pointer or reference and type cast it into the child class pointer or reference.

Syntax :

dynamic_cast<type> (object/pointer);

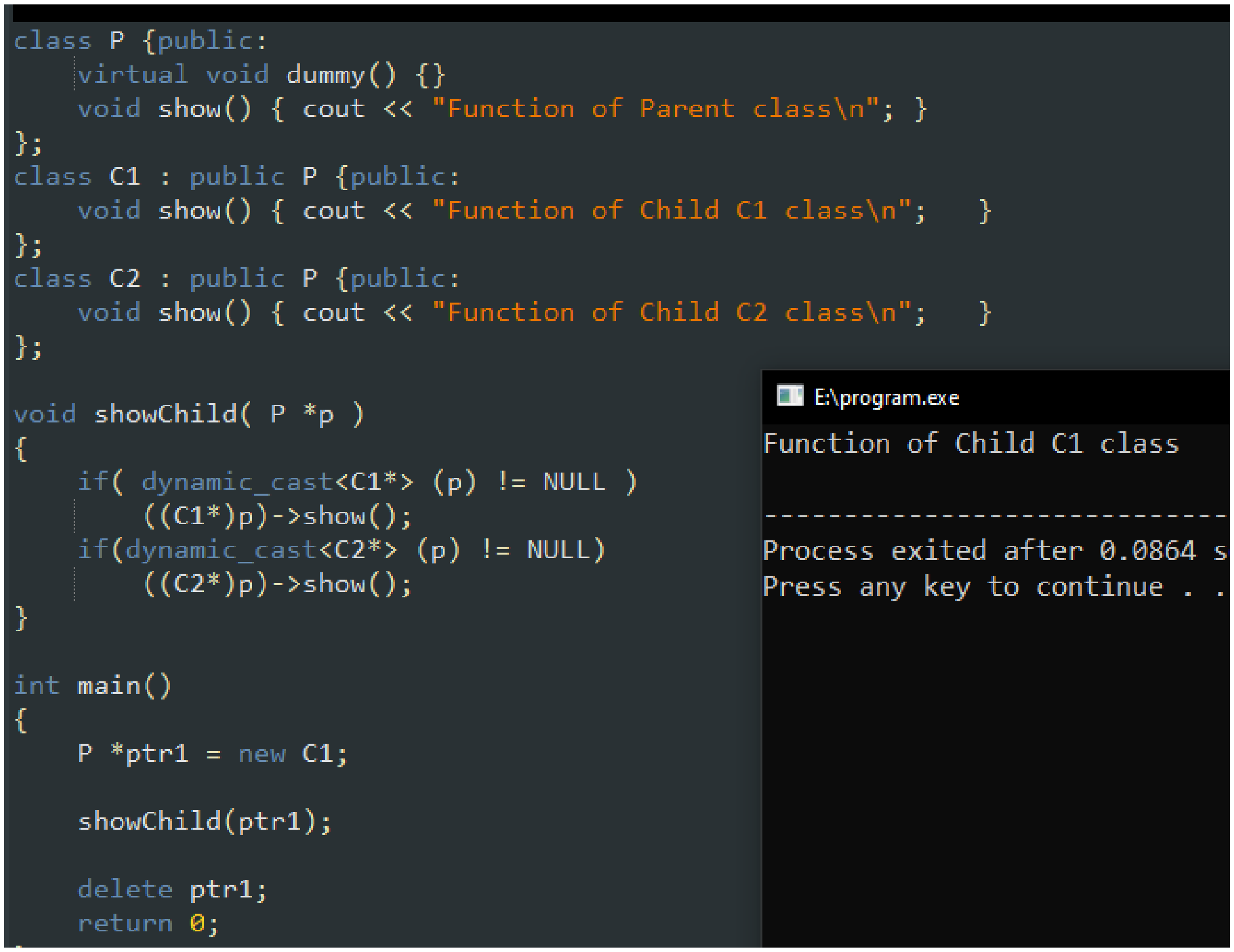
The advantage of dynamic cast is that it will only type cast the object/pointer only if it is a valid transformation. It means that if a parent class pointer contains an object of child class only then it can be type cast to the child class otherwise the dynamic cast will return a NULL.



```
class P {   virtual void dummy() {}     };
class C1 : public P {};
class C2 : public P {};

int main()
{
    P *ptr1 = new P;
    P *ptr2 = new C1;
    P *ptr3 = new C2;

    if ( dynamic_cast<P*> (ptr1)!= NULL )
        cout << "ptr1 is valid type cast to P\n";
    if ( dynamic_cast<C1*> (ptr1) != NULL )
        cout << "ptr1 is valid type cast to C1\n";
    if ( dynamic_cast<C2*> (ptr1) != NULL )
        cout << "ptr1 is valid type cast to C2\n";


    if ( dynamic_cast<P*> (ptr2) != NULL )
        cout << "ptr2 is valid type cast to P\n";
    if ( dynamic_cast<C1*> (ptr2) != NULL )
        cout << "ptr2 is valid type cast to C1\n";
    if ( dynamic_cast<C2*> (ptr2) != NULL )
        cout << "ptr2 is valid type cast to C2\n";


    if ( dynamic_cast<P*> (ptr3) != NULL )
        cout << "ptr3 is valid type cast to P\n";
    if ( dynamic_cast<C1*> (ptr3) != NULL )
        cout << "ptr3 is valid type cast to C1\n";
    if ( dynamic_cast<C2*> (ptr3) != NULL )
        cout << "ptr3 is valid type cast to C2\n";

    delete ptr1;
    delete ptr2;
    delete ptr3;
```

E:\program.exe

```
ptr1 is valid type cast to P
ptr2 is valid type cast to P
ptr2 is valid type cast to C1
ptr3 is valid type cast to P
ptr3 is valid type cast to C2

--------------------------------
Process exited after 0.1133 seconds
Press any key to continue . . .
```



```cpp
class P {public:
    virtual void dummy() {}
    void show() { cout << "Function of Parent class\n"; }
};
class C1 : public P {public:
    void show() { cout << "Function of Child C1 class\n";   }
};
class C2 : public P {public:
    void show() { cout << "Function of Child C2 class\n";   }
};

void showChild( P *p )
{
    if( dynamic_cast<C1*> (p) != NULL )
        ((C1*)p)->show();
    if(dynamic_cast<C2*> (p) != NULL)
        ((C2*)p)->show();
}

int main()
{
    P *ptr1 = new C1;

    showChild(ptr1);

    delete ptr1;
    return 0;
```

E:\program.exe

```
Function of Child C1 class

-----------------------------
Process exited after 0.0864 s
Press any key to continue . .
```

-THE END-