GeeksforGeeks

Related Articles

Pure virtual destructor in C++

Difficulty Level : Medium • Last Updated : 09 Aug, 2019

Can a destructor be pure virtual in C++?
Yes, it is possible to have pure virtual destructor. Pure virtual destructors are legal in standard C++ and one of the most important things to remember is that if a class contains a pure virtual destructor, it must provide a function body for the pure virtual destructor. You may be wondering why a pure virtual function requires a function body. The reason is because destructors (unlike other functions) are not actually 'overridden', rather they are always called in the reverse order of the class derivation. This means that a derived class' destructor will be invoked first, then base class destructor will be called. If the definition of the pure virtual destructor is not provided, then what function body will be called during object destruction? Therefore the compiler and linker enforce the existence of a function body for pure virtual destructors.
Consider the following program:

```
#include <iostream>
class Base
{
public:
    virtual ~Base()=0; // Pure virtual destructor
};

class Derived : public Base
{
public:
    ~Derived()
    {
        std::cout << "~Derived() is executed";
    }
};

 nt main()
{
    Base *b=new Derived();
    delete b;
    return 0;
}
```

The linker will produce following error in the above program.

```
test.cpp:(.text$_ZN7DerivedD1Ev[__ZN7DerivedD1Ev]+0x4c):
undefined reference to `Base::~Base()'
```

Now if the definition for the pure virtual destructor is provided, then the program compiles & runs fine.

```
#include <iostream>
class Base
{
public:
    virtual ~Base()=0; // Pure virtual destructor
};
Base::~Base()
{
    std::cout << "Pure virtual destructor is called";
}

class Derived : public Base
{
public:
    ~Derived()
    {
        std::cout << "~Derived() is executed\n";
    }
};

int main()
{
    Base *b = new Derived();
    delete b;
    return 0;
}
```

Output:

```
~Derived() is executed
Pure virtual destructor is called
```

It is important to note that a class becomes abstract class when it contains a pure rtual destructor. For example, try to compile the following program.

```
#include <iostream>
class Test
{
public:
    virtual ~Test()=0; // Test now becomes abstract class
};
Test::~Test() { }

int main()
{
    Test p;
    Test* t1 = new Test;
    return 0;
}
```

The above program fails in compilation & shows following error messages.
[Error] cannot declare variable 'p' to be of abstract type 'Test'
[Note] because the following virtual functions are pure within 'Test':
[Note] virtual Test::~Test()
[Error] cannot allocate an object of abstract type 'Test'
[Note] since type 'Test' has pure virtual functions

Related Articles :
Constructors in C++
Destructors in C++
Virtual Destructor

Sources:
http://www.bogotobogo.com/cplusplus/virtualfunctions.php
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2005/n1905.pdf

This article was contributed by **Meet Pravasi**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Want to learn from the best curated videos and practice problems, check out the C++ Foundation Course for Basic to Advanced C++ and C++ STL Course for foundation plus STL. To complete your preparation from learning a language to DS Algo and many more, please refer **Complete Interview Preparation Course**.