


## C++

[Information](#)  
[Tutorials](#)  
[Reference](#)  
[Articles](#)  
[Forum](#)

## Forum

[Beginners](#)  
[Windows Programming](#)  
[UNIX/Linux Programming](#)  
[General C++ Programming](#)  
[Lounge](#)  
[Jobs](#)

## Creating copies of pointers to objects

Pages: [1](#) [2](#)**Cuddly Kittens 11** (12) Feb 22, 2011 at 9:57am

Hey There.

I am trying to create a class called "enemyManager", that well, manages the enemies in my game (deployment, memory deletion, etc.).

Well, for the function that adds an "enemy" class to the manager, I'm having difficulty trying to create a copy of "refEnemy", then putting that copy into an array that manages the enemy data.


Here is the function:

```
1 // Add an enemy to the manager
2 void enemyManager::addEnemy(enemy *refEnemy, int instances) {
3     if ((instances < 1) || (_numEnemies >= EM_MAX_NUM_ENEMIES)) // Just to makesure we don't add any extra stuff
4         return;
5
6     int tempVerticesArray[64][2];
7
8     for (int i = 0; i < instances; i++) {
9         enemies[_curAvailableSlot]->enemyPtr = refEnemy; // <-- Issue is right here
10        enemies[_curAvailableSlot]->ptrClearedFromMem = false;
11
12        _curAvailableSlot++;
13        _numEnemies++;
14    }
15 }
```


All I know is that I'm putting the address of "refEnemy" into "enemyPtr" right now which is what I don't want. I want it to copy all the data from "refEnemy" to "enemyPtr".

If needed I'll provide the whole header file.


Thanks.

**hamsterman** (4538) Feb 22, 2011 at 10:27am

If enemy isn't abstract, `enemies[_curAvailableSlot]->enemyPtr = new enemy(*refEnemy);`


**Cuddly Kittens 11** (12) Feb 22, 2011 at 10:58am

Wow, thanks, that worked out perfect.


**ne555** (9314) Feb 22, 2011 at 2:07pm

But then you are losing the polymorphism  
Wouldn't be better something like

```
1 class enemy{
2     virtual enemy* clone() = 0;
3 };
```

**jsmith** (5804) Feb 22, 2011 at 2:46pm

Yes, hamsterman's example will compile, but will not work in practice. The clone pattern is necessary.

**Cuddly Kittens 11** (12) Feb 22, 2011 at 3:22pm

ne555 and jsmith, could you elaborate on this please? (Specifically what I'll need to do to copy all the data into "enemyPtr" properly.)

Also, lets say that I have a class called "boss":

```
1 class boss : public enemy {
2 public:
3     boss();
4     ~boss();
5
6     doStuff();
7 private:
8     _ID;
9 };
```

Would I be able to add it to the enemyManger as of right now? Like this:

```
1 enemyManager *em = new enemyManager();
2 boss *b1 = new boss();
3 em->addEnemy(b1);
```

Would I need to overload the function at all?

jsmith (5804)

Feb 22, 2011 at 4:42pm

Consider this code:

```
1 struct Base {
2     virtual ~Base() {}
3     int x;
4     int y;
5 };
6
7 struct Derived : Base {
8     int z;
9 };
10
11 void copy_it( Base* b )
12 {
13     Base* copy_of_it = new Base( b );
14 }
15
16 int main()
17 {
18     Derived* d = new Derived;
19     copy_it( d );
20 }
```

copy\_it does what is called "slicing" -- an instance of Derived has two "layers" -- the bottom-most layer is the data members of Base and the topmost layer is the data members of Derived. When copy\_it is called, it makes a copy not of the two-layer cake, but only of the bottom-most layer. This is the problem with polymorphism. To solve that problem -- to have copy\_it make a complete copy of what is passed to it -- you have to write a virtual clone() method that derived instances must override.

Cuddly Kittens 11 (12)

Feb 22, 2011 at 5:38pm

So what would I want to include in my virtual clone() method?

And wouldn't that example code (if compiled and ran) create a memory leak?

EDIT:

I forgot to mention that the "enemy" class contains pointers to a lot of other classes (like "sprite", "sound", etc). So would I want the clone function/method to return clones of these other classes?

Last edited on Feb 22, 2011 at 6:02pm

jsmith (5804)

Feb 22, 2011 at 6:10pm

The clone() method is quite simple, see below.

Yes, I did not delete copy\_of\_it, so it would leak. Fixed below.

```
1 struct Base {
2     virtual ~Base() {}
3     virtual Base* clone() const { return new Base( *this ); }
4     int x;
5     int y;
6 };
7
8 struct Derived : Base {
9     virtual Derived* clone() const { return new Derived( *this ); }
10    int z;
11 };
12
13 void copy_it( Base* b )
14 {
15     Base* copy_of_it = b->clone();
16     // use copy_of_it ...
17     delete copy_of_it;
18 }
19
20 int main()
21 {
22     Derived* d = new Derived;
23     copy_it( d );
24 }
```

Whether you want to implement a deep copy or a shallow copy is up to your design. If the enemy class owns the pointers, then probably yes, you'll want a deep copy. If sprite and sound are also base classes, you'll need a clone method on them too. If they are not polymorphic, then you don't need the clone() method.

The clone() pattern exists solely to get around the slicing issue that results from casting a pointer up to a base class type and then attempting to copy the entire object through the base class pointer.

Cuddly Kittens 11 (12)

Feb 23, 2011 at 12:18am

Well for my game, the basic variable structure of "enemy" is this:

```
1 class enemy : public sprite {
2 public:
3     static int sNextEnemyID;
4
5     bool invul;
```

```

6         int speed;
7         int damage;
8         int pointValue;
9
10    private:
11        int _enemyID;
12        int _curHealth;
13        int _maxHealth;
14        bool _isAlive;
15        bool _hitNoiseAdded;
16        bool _deathNoiseAdded;
17        bool _deathSpriteAdded;
18        sound * _hitNoise;
19        sound * _deathNoise;
20        sprite * _deathSprite;
21        int _deathSpriteOffsetX;
22        int _deathSpriteOffsetY;
23        int _deathSpriteTimer;
24        bool _deathRoutineStarted;
25
26    };

```

Sprite and Sound are base classes, but as you can see, "enemy" is derived from "sprite".

I'm also using The DarkGDK game library, and each time I create a new sprite, it calls a DarkGDK function where it loads an image file into memory and then creates a sprite for it. Both of the base classes don't contain any pointers.

So I'm guessing a deep copy here?

*Last edited on Feb 23, 2011 at 12:21am*

**jsmith (5804)**

Feb 23, 2011 at 12:50am

Who is responsible for freeing the memory allocated to \_hitNoise, \_deathNoise, and \_deathSprite?

If it is the enemy class, then yes, you'll need a deep copy.

**Cuddly Kittens 11 (12)**

Feb 23, 2011 at 10:08am

In the "enemy" class destructor, it uses the "delete" keyword.

Ex:

```

1 // Enemy class destructor
2 // cleans up some of the stuff for us
3 enemy::~enemy() {
4     // Delete the hit noise
5     if (_hitNoiseAdded)
6         delete _hitNoise;
7
8     // Delete the death noise
9     if (_deathNoiseAdded)
10        delete _deathNoise;
11
12    // Delete the death Sprite
13    if (_deathSpriteAdded)
14        delete _deathSprite;
15 }

```

**Cuddly Kittens 11 (12)**

Feb 27, 2011 at 10:58pm

Okay, sorry for both double-posting and necro-posting.

I got my "clone()" method written for both my "sprite" and "enemy" class, but I'm running into a little bit of a problem.

I'm getting these compiler errors:

```

1 error C2143: syntax error : missing ';' before '*'
2 error C4430: missing type specifier - int assumed. Note: C++ does not support default-int
3 error C4430: missing type specifier - int assumed. Note: C++ does not support default-int
4 error C2556: 'int *enemy::clone(void)' : overloaded function differs only by return type from 'enemy *enemy::clone(void)'
5 error C2371: 'enemy::clone' : redefinition; different basic types see declaration of 'enemy::clone'

```

I think it has something to do with my function declarations and definitions.

For "sprite" class: (<- This works weel by itself.)

```

1 class sprite {
2     public:
3         // ...
4
5         virtual sprite *clone();
6
7         // ...
8 };
9
10 sprite *sprite::clone() {
11     sprite *cpy = new sprite();
12
13     // ...
14
15     return cpy;
16 }

```

For "enemy" class: (<- Errors upon compiling.)

```

1 class enemy : public sprite {

```

```

2      public:
3          // ..
4
5          virtual enemy *clone();
6
7          // ..
8      };
9
10
11 enemy *enemy::clone() {
12     enemy *cpy = new enemy();
13
14     // ...
15
16     return cpy;
17 }


```

Any help is really appreciated.

Thanks.

*Last edited on Feb 27, 2011 at 11:00pm*

**simeonz** (490)


 Feb 27, 2011 at 11:27pm

```

enemy *enemy::clone() { :)

```

**ne555** (9314)

 Feb 28, 2011 at 8:07am


The clone method is supposed to return a pointer to the base class (in this case sprite). You can't modify that prototype.

However it seems a little weird to say that an enemy **is** a sprite. Wouldn't be better to say that enemy **has** a sprite?

Edit: sorry about the misinformation.  
Thanks for the correction.

*Last edited on Feb 28, 2011 at 12:22pm*

**jsmith** (5804)

 Feb 28, 2011 at 9:19am

No, you can modify the prototype. Remember that the return type of the function is not part of the function's type, so that


```

1 struct Base{
2     virtual int frobnicate() = 0;
3 };
4
5 struct Derived : base {
6     virtual float frobnicate() {}
7 };

```

makes Derived a concrete class since Derived's frobnicate() is an override of Base's pure virtual method. It's perfectly ok for the derived class therefore to return a pointer to its type instead of the base type.

**ne555** (9314)


 Feb 28, 2011 at 12:01pm

```

1 struct Base{
2     virtual int frobnicate() = 0; //error: overriding 'virtual int Base::frobnicate()'
3 };
4
5 struct Derived : public Base {
6     virtual float frobnicate() {} //error: conflicting return type specified for 'virtual float Derived::frobnicate()'
7 };
8
9 int main(int argc, char *argv[]) {
10     Base *b;
11     b = new Derived;
12     b->frobnicate();
13     return 0;
14 }


```

**jsmith** (5804)

 Feb 28, 2011 at 2:03pm

Sorry, doesn't like the int/float thing, but replacing int with Base\* and float with Derived\*, it compiles fine.


**jsmith** (5804)

 Feb 28, 2011 at 2:12pm

Sorry, my bad.

If you replace "int" with "Base\*" and "float" with "Derived\*" (then return 0) it compiles fine.

**Cuddly Kittens 11** (12)

 Feb 28, 2011 at 9:17pm

@simeonz  
Thanks. (Wow I feel like and idiot.)

@Ne555  
The reason why I derived "enemy" from "sprite" is that the original class for "enemy" contained many similar things that "sprite" had, so why not just derive it. Lot of work saved.

Thanks everyone.

OPEN A RETIREMENT  
ACCOUNT AND GET UP TO \$600.

C  
I  
T