C++ class forward declaration

When I try to compile this code i get:

```
52 C:\Dev-Cpp\Projektyyy\strategy\Tiles.h invalid use of undefined type `struct
tile_tree_apple'
46 C:\Dev-Cpp\Projektyyy\strategy\Tiles.h forward declaration of `struct
tile_tree_apple'
```

some part of my code:

```
class tile_tree_apple;
```

```
class tile_tree : public tile
{
      public:
          tile onDestroy() {return *new tile_grass;};
          tile tick() {if (rand()%20==0) return *new tile_tree_apple;};
          void onCreate() {health=rand()%5+4; type=TILET_TREE;};
};

class tile_tree_apple : public tile
{
      public:
          tile onDestroy() {return *new tile_grass;};
          tile tick() {if (rand()%20==0) return *new tile_tree;};
          void onCreate() {health=rand()%5+4; type=TILET_TREE_APPLE;};
          tile onUse() {return *new tile_tree;};
};
```

I dont really know what to do, I searched for the solution but I couldnt find anything simmilar to my problem... Actually, i have more classes with parent "tile" and It was ok before... Thanx for any help.

EDIT:

I decided to change all returned types to pointers to avoid memory leaks, but now I got:

```
27 C:\Dev-Cpp\Projektyyy\strategy\Tiles.h ISO C++ forbids declaration of `tile'
with no type
27 C:\Dev-Cpp\Projektyyy\strategy\Tiles.h expected `;' before "tick"
```

Its only in base class, everything else is ok... Every function in tile class which return *tile has this error...

Some code:

```
class tile
{
      public:
          double health;
          tile_type type;
          *tile takeDamage(int ammount) {return this;};
          *tile onDestroy() {return this;};
          *tile onUse() {return this;};
          *tile tick() {return this};
          virtual void onCreate() {};
};
```

c++ class forward-declaration

edited Feb 2 '12 at 20:28

asked Feb 2 '12 at 20:10

noisy cat

1,038 3 19 41

9 Answers

In order for `new T` to compile, `T` must be a complete type. In your case, when you say `new tile_tree_apple` inside the definition of `tile_tree::tick`, `tile_tree_apple` is incomplete (it has been forward declared, but its definition is later in your file). Try moving the inline definitions of your functions to a separate source file, or at least move them after the class definitions.

Something like:

```
class A
{
    void f1();
    void f2();
};
class B
{
   void f3();
   void f4();
};

inline void A::f1() {...}
inline void A::f2() {...}
inline void B::f3() {...}
inline void B::f4() {...}
```

When you write your code this way, all references to A and B in these methods are guaranteed to refer to complete types, since there are no more forward references!

edited Nov 22 '16 at 21:47

Edward Z. Yang

18k 14 60 84

answered Feb 2 '12 at 20:13

Armen Tsirunyan

80.6k 39 231 358

Doesn't the `inline` also have to go in the class definition? I'm never too sure about this... – Kerrek SB Feb 2 '12 at 20:17

1 @KerrekSB: AFAIR, it should go either to the declaration or the definition, but doesn't matter which one – Armen Tsirunyan Feb 2 '12 at 20:18

3 @KerrekSB : It needs to go on the definition only; putting it on the declaration has no effect whatsoever. – ildjarn Feb 2 '12 at 20:19

2 @kittyPL: Use smart pointers to avoid memory leaks – Armen Tsirunyan Feb 2 '12 at 20:26

7 @kittyPL: Pointers don't cause memory leaks, poor coding causes memory leaks. – Clifford Feb 2 '12 at 20:51

Use forward declaration when possible.

Suppose you want to define a new class `B` that uses objects of class `A`.

1. `B` only uses references or pointers to `A`. Use forward declaration then you don't need to include `<A.h>`. This will in turn speed a little bit the compilation.

```
class A ;

class B
{
  private:
    A* fPtrA ;
  public:
    void mymethod(const& A) const ;
} ;
```

2. `B` derives from `A` or `B` explicitely (or implicitely) uses objects of class `A`. You then need to include `<A.h>`

```
#include <A.h>

class B : public A
{
};
```

```
class C
{
  private:
    A fA ;
  public:
    void mymethod(A par) ;
}
```

edited Nov 7 '12 at 11:24

PVitt
7,303 2 30 68

answered Nov 7 '12 at 10:56

kumaran
139 1 2

It should be: void mymethod(const A &) const. – KIIV Jun 9 '16 at 7:54

This answer (#1) contains error, so I downvoted. – John Boe Jun 9 '16 at 8:52

The forward declaration is an "*incomplete type*", the only thing you can do with such a type is instantiate a *pointer* to it, or reference it in a function *declaration* (i.e. and argument or return type in a function prototype). In line 52 in your code, you are attempting to instantiate an *object*.

At that point the compiler has no knowledge of the object's size nor its constructor, so cannot instantiate an object.

edited Feb 2 '12 at 20:22

answered Feb 2 '12 at 20:15

Clifford
47.4k 6 45 102

2 There are many more things you can do with an incomplete type. – Kerrek SB Feb 2 '12 at 20:16

1 How does one instantiate a pointer without instantiating an object? – Luchian Grigore Feb 2 '12 at 20:19

@Luchian: In this case: `tile_tree_apple* tta_ptr ;` Instantiates a pointer of type `tile_tree_apple*` , although of course it does not point to a valid object. The point is you might have such a pointer as a member of a class that later instantiates the object in the constructor for example, but either way, at point in the code where the *complete* type is visible. – Clifford Feb 2 '12 at 20:45

@Kerrek: Maybe, but perhaps none that are relevant to this discussion. It would be useful though perhaps if

you could elaborate. That said looking at your comment time, I may have already covered them in the last edit. – Clifford Feb 2 '12 at 20:55

I had this:

```
class paulzSprite;
...

struct spriteFrame
{
    spriteFrame(int, int, paulzSprite*, int, int);
    paulzSprite* pSprite; //points to the sprite class this struct frames
    static paulzSprite* pErase; //pointer to blanking sprite
    int x, y;
    int Xmin, Xmax, Ymin, Ymax; //limits, leave these to individual child classes,
according to bitmap size
    bool move(int, int);
    bool DrawAt(int, int);
    bool dead;
};

spriteFrame::spriteFrame(int initx, int inity, paulzSprite* pSpr, int winWidth, int
winHeight)
{
    x = initx;
    y= inity;
    pSprite = pSpr;
    Xmin = Ymin = 0;
    Xmax = winWidth - pSpr->width;
    Ymax = winHeight - pSpr->height;
    dead = false;
}
```

...

Got the same grief as in the original question. Only solved by moving the definition of paulzSprite to **after** that of spriteFrame. Shouldn't the compiler be smarter than this (VC++, VS 11 Beta)?

And btw, I wholeheartedly agree with Clifford's remark above "Pointers don't cause memory leaks, poor coding causes memory leaks". IMHO this is true of many other new "smart coding" features, which should not become a substitute for understanding what you are actually asking the computer to do.

edited Dec 9 '12 at 23:54

Jeff
4,257 3 18 43

answered Dec 9 '12 at 23:36

Paul Pignon
51 1 1

Hi, and welcome to Stack Overflow. We don't use signatures here, so please don't be offended if/when your signature is removed – Jeff Dec 9 '12 at 23:54

class tile_tree_apple should be defined in a separate .h file.

```
tta.h:
#include "tile.h"

class tile_tree_apple : public tile
{
      public:
          tile onDestroy() {return *new tile_grass;};
          tile tick() {if (rand()%20==0) return *new tile_tree;};
          void onCreate() {health=rand()%5+4; type=TILET_TREE_APPLE;};
          tile onUse() {return *new tile_tree;};
};

file tt.h
#include "tile.h"

class tile_tree : public tile
{
      public:
          tile onDestroy() {return *new tile_grass;};
          tile tick() {if (rand()%20==0) return *new tile_tree_apple;};
          void onCreate() {health=rand()%5+4; type=TILET_TREE;};
};
```

another thing: returning a tile and not a tile reference is not a good idea, unless a tile is a primitive or very "small" type.

answered Feb 2 '12 at 20:17

vulkanino
7,140 4 25 58

Won't tile_tree_apple need to know about tile_tree though an include and vice-versa? – Bren Mar 11 '16 at

16:01

The problem is that `tick()` needs to know the definition of `tile_tree_apple`, but all it has is a forward declaration of it. You should separate the declarations and definitions like so:

`tile_tree.h`

```
#ifndef TILE_TREE_H
#define TILE_TREE_H
#include "tile.h"

class tile_tree : public tile
{
public:
    tile onDestroy();
    tile tick();
    void onCreate();
};

#endif
```

`tile_tree.cpp`:

```
tile tile_tree::onDestroy() {
    return *new tile_grass;
}

tile tile_tree::tick() {
     if (rand() % 20 == 0)
         return *new tile_tree_apple;
}

void tile_tree::onCreate() {
    health = rand() % 5 + 4;
    type = TILET_TREE;
}
```

Except you have a major problem: you're allocating memory (with `new`), then copying the allocated object and returning the copy. This is called a *memory leak*, because there's no way for your program to free the memory it uses. Not only that, but you're copying a `tile_tree` into a `tile`, which discards the information that makes a `tile_tree` different from a `tile`; this is called *slicing*.

What you want is to return a pointer to a new `tile`, and make sure you call `delete` at some point to free the memory:

```
tile* tile_tree::tick() {
     if (rand() % 20 == 0)
         return new tile_tree_apple;
}
```

Even better would be to return a smart pointer that will handle the memory management for you:

```
#include <memory>

std::shared_ptr<tile> tile_tree::tick() {
     if (rand() % 20 == 0)
         return std::make_shared<tile_tree_apple>();
}
```

answered Feb 2 '12 at 20:18

Jon Purdy
30.2k 5 52 110

To do anything other than declare a pointer to an object, you need the full definition.

The best solution is to move the implementation in a separate file.

If you **must** keep this in a header, move the definition after both declarations:

```
class tile_tree_apple;

class tile_tree : public tile
{
  public:
      tile onDestroy();
      tile tick();
      void onCreate();
};

class tile_tree_apple : public tile
{
  public:
      tile onDestroy();
```

```
        tile tick();
        void onCreate();
        tile onUse();
};

tile tile_tree::onDestroy() {return *new tile_grass;};
tile tile_tree::tick() {if (rand()%20==0) return *new tile_tree_apple;};
void tile_tree::onCreate() {health=rand()%5+4; type=TILET_TREE;};

tile tile_tree_apple::onDestroy() {return *new tile_grass;};
tile tile_tree_apple::tick() {if (rand()%20==0) return *new tile_tree;};
void tile_tree_apple::onCreate() {health=rand()%5+4; type=TILET_TREE_APPLE;};
tile tile_tree_apple::onUse() {return *new tile_tree;};
```

Important

You have memory leaks:

```
tile tile_tree::onDestroy() {return *new tile_grass;};
```

will create an object on the heap, which you can't destroy afterwards, unless you do some ugly hacking. Also, your object will be sliced. **Don't do this, return a pointer.**

edited Feb 2 '12 at 20:18

answered Feb 2 '12 at 20:13

Luchian Grigore
175k 32 320 475

This isn't true. Look up the standard. You can, for example, declare (but not define) a function that takes T if T is an incomplete type. You can also declare references to T. – Armen Tsirunyan Feb 2 '12 at 20:13

So should i change everything in classes to pointers? – noisy cat Feb 2 '12 at 20:15

2 This should be useful, – Alok Save Feb 2 '12 at 20:15

The return type of a function may also be incomplete. – Kerrek SB Feb 2 '12 at 20:16

I meant the `new` isn't. – Luchian Grigore Feb 2 '12 at 20:17

To perform `*new tile_tree_apple` the constructor of `tile_tree_apple` should be called, but in this place compiler knows nothing about `tile_tree_apple` , so it can't use the constructor.

If you put

```
tile tile_tree::tick() {if (rand()%20==0) return *new tile_tree_apple;};
```

in separate cpp file which has the definition of class tile_tree_apple or includes the header file which has the definition everything will work fine.

answered Feb 2 '12 at 20:21

Seagull
1,545 11 25

I'm a CPP noob, but don't you have to specify the inheritance in the forward declaration?

```
class tile_tree_apple : public tile;
```

answered Feb 2 '12 at 20:15

Matthew
14.3k 3 39 67

3 No you don't . . – Joe Feb 2 '12 at 20:15

No, then the compiler would need the whole declaration... – noisy cat Feb 2 '12 at 20:16
