Declaring an array inside a class, and setting its size with the constructor

I haven't worked with c++ in a while, but I just started a project with it. This may not be possible, but Im trying to create a template class with an array that sets its size to the value of a constant which i'm trying to set with the constructor.

This is the code of the constructor:

```
Tarray(int s): start_size(s){
    }
```

This is the code that sets the array size:

```
const int start_size;
T this_array[start_size];
```

This is the entire file:

```
#ifndef TARRAY_H_
#define TARRAY_H_


template<typename T>
class Tarray {
private:
    const int start_size;
    T this_array[start_size];
    int array_size;
public:
    Tarray(int s): start_size(s){
    }
    ~Tarray(){
        delete[] this_array;
    }
    T & operator[](int i){
        return this_array[i];
    }
};



#endif /* TARRAY_H_ */
```

These are the errors I get:

```
..\template_array/Tarray.h:16:24: error: 'Tarray<T>::start_size' cannot appear in a
constant-expression
..\template_array/Tarray.h:16:34: error: 'new' cannot appear in a constant-expression
..\template_array/Tarray.h:16:34: error: ISO C++ forbids initialization of member
'this_array' [-fpermissive]
..\template_array/Tarray.h:16:34: error: making 'this_array' static [-fpermissive]
..\template_array/Tarray.h: In instantiation of 'Tarray<Person>':
..\Human.cpp:17:24:   instantiated from here
..\template_array/Tarray.h:16:34: error: invalid in-class initialization of static data
member of non-integral type 'Person*'
Build error occurred, build is stopped
Time consumed: 343  ms.
```

The error messages have been changing as I try to tweak the code, but these are the errors from this particular build.

Thanks for any help

c++ arrays class constructor constants

asked Mar 5 '12 at 2:23

sinθ
2,240 6 33 76

1 use std::vector<> – Michael Burr Mar 5 '12 at 2:26

Thanks, but i'm still wondering how you do this. I haven't used c++ in a while, and I'm trying to re-learn it. – sinθ Mar 5 '12 at 2:28

How would `sizeof` work if such a construct was allowed? – David Schwartz Mar 5 '12 at 2:36

C++ doesn't support variable length arrays in that way. C99 does, but not C++ (not even C++11). GNU supports them in C++ as an extension, but for automatic variables, not class members (as far as I know). You need to use `new` / `malloc` explicitly, or use `vector` and let that class manage the dynamic allocation for you (the better way to go in almost all cases). – Michael Burr Mar 5 '12 at 2:36

If the value is only known at runtime, then the array has to be dynamically allocated using new. If the value is known at compile time, then it can be a template parameter and the template parameter can be used for the array size. – Vaughn Cato Mar 5 '12 at 2:39

5 Answers

The reason you're getting compiler errors is this line:

```
T this_array[start_size];
```

This line would make your `Tarray` actually contain `start_size` instances of `T`. It wouldn't hold a pointer or reference to these instances - they would be part of same block of memory that contains the Tarray's other instance variables. This would make the class' size depend on start_size, and start_size is not known at compile time. The size of any C++ class must be known at compile time, this isn't possible.

There are two ways to solve this:

1. Allocate the array of `T` instances on the heap, using array new. This is what `std::vector` does. Writing such a class and getting it to behave right when it's copied/moved/expanded/etc is difficult and tedious, so I'd recommend just using `std::vector` instead.
2. Make the number of T instances fixed, and pass it as a template parameter

i.e.:

```
template<typename T, std::size_t N>
class TArray
{
    ...
    T this_array[N];
    ...
}
```

This is what std::array (C++11 only) and boost::array do. Again, I'd recommend using one of these instead of writing your own. Unless this is homework, of course...

Lastly, it's worth noting that this is an error:

```
~Tarray(){
    delete[] this_array;
}
```

`this_array` wasn't allocated with `new`, so you shouldn't `delete` it. If the array is part of the class as it is here (rather than being separately heap-allocated and owned by the class), then it will be destroyed along with the rest of the class by default. Calling `delete` is not only unnecessary, it will almost certainly cause a crash.

answered Mar 5 '12 at 3:12

je4d
4,851 11 34

`std::vector` is precisely the tool for this job:

```
template<typename T>
class Tarray {
private:
    std::vector<T> this_array;
public:
    Tarray(int s): this_array(s){
    }
    ~Tarray(){
    }
    T & operator[](int i){
        return this_array[i];
    }
};
```

answered Mar 5 '12 at 2:27

Rob_φ
62.2k 3 52 115

`this_array.reserve(s)` would have been better. – iammilind Mar 5 '12 at 2:47

@iammilind disagree, the semantics would be completely different as the elements would not be created if only reserve is called. – David Rodríguez - dribeas Mar 5 '12 at 3:21

@DavidRodríguez-dribeas, I know, it's just reserved. And that is the purpose. With `vector::reserve()` one can initialize the elements on need bases. If out of size `s` , only half is going to be used then no point in creating whole array. – iammilind Mar 5 '12 at 3:32

The following code does something similar but not using the constructor:

```
#ifndef TARRAY_H_
#define TARRAY_H_


template<int SizeT>
class Tarray {
private:
    T this_array[SizeT];
public:
    Tarray() {}
    ~Tarray() {}
    T & operator[](int i){
        return this_array[i];
    }
};

#endif /* TARRAY_H_ */
```

and you can use it like this:

```
TArray<10> myArray;
```

answered Mar 5 '12 at 2:33

alarouche
376 2 3

You have to create the array at run time.

```
template<typename T>
class Tarray {
private:
    const int start_size;
    T* this_array;
    int array_size;

    Tarray( const Tarrat& inObj ); // no copy

public:
    Tarray(int s): start_size(s), this_array( new T[s] ) {
    }
    ~Tarray(){
        delete[] this_array;
    }
    T & operator[](int i){
        return this_array[i];
    }
};
```

Note, for this to work, T must have a default constructor (that is, a constructor that takes no arguments).

edited Mar 5 '12 at 4:45

answered Mar 5 '12 at 2:31

claireware
1,319 6 16

This code will be very fragile - what happens if you accidentally copy it? – Arafangion Mar 5 '12 at 2:32

The question was how to create vectors in a template, which I answered. However, no accidental copying should occur because a constructor has been declared (which prevents the compiler from auto creating constructors), but no copy constructor has been declared. As a result, the compiler will complain if there were any implicit copies occurring. Fragility averted. I'd appreciate the -1 taken away, thank you. – claireware Mar 5 '12 at 3:28

The compiler will auto-generate a copy-constructor for you. If you want to specify no copy-constructor, you need to provide it as an explicitly private constructor, but then fail to provide an implementation. Also, in order for me to remove the -1, the answer needs to be updated. – Arafangion Mar 5 '12 at 4:22

My bad, I was confused with default constructors. Updated. – claireware Mar 5 '12 at 4:46

Use std::vector instead, and make life simple for yourself. :)

(If you want a fixed-size array, then std::array might be a possibility, I think that's in C++11, if not, then boost probably has an implementation).

If you insist on having that ordinary array syntax, though, as if you were using ye-olde C, then you will need to use a template parameter, such that your template class has two arguments - one for the 'T' it already has now, and another for the array size.

You are making life especially difficult by managing that array yourself - if you feel you have to define a destructor, you really should define the copy constructor in addition to the constructor. (That's called the Rule Of The Big Three, if I recall correctly), instead, rely on RAII and avoid having to ever explicitly call operator delete or delete[] yourself.

answered Mar 5 '12 at 2:27

Arafangion
6,090 16 44

Thanks, but I still get errors like "'Tarray<T>::start_size' cannot appear in a constant-expression" – sinθ Mar 5 '12 at 2:33

@Mike G: If you're using std::vector, then there is no point in keeping a ::start_size. That said, you've specified 'const' there without giving it a value. (And you can't do THAT in a class like that) – Arafangion Mar 5 '12 at 2:41