FIFO implementation

While implementing a FIFO I have used the following structure:

```
struct Node
{
    T info_;
    Node* link_;
    Node(T info, Node* link=0): info_(info), link_(link)
    {}
};
```

I think this a well known trick for lots of STL containers (for example for List). Is this a good practice? What it means for compiler when you say that Node has a member with a type of it's pointer? Is this a kind of infinite loop?

And finally, if this is a bad practice, how I could implement a better FIFO.

EDIT: People, this is all about implemenation. I am enough familiar with STL library, and know a plenty of containers from several libraries. Just I want to discuss with people who can gave a good implementation or a good advice.

c++ implementation fifo

edited Jun 13 '10 at 19:29

asked Jun 13 '10 at 19:20

Narek

15.1k 44 152 266

5 Answers

Pointers to objects of type that is being declared is fine in both C and C++. This is based on the fact that pointers are objects of fixed size (say, always 32-bit integers on 32-bit platform) so you don't need the full size of the pointed-to type to be known.

In fact, you don't even need a full type declaration to declare a pointer. A forward declaration would suffice:

```
class A; // forward declared type

struct B
{
    A* pa; //< pointer to A - perfectly legal
};
```

Of course, you need a full declaration in scope at the point where you actually access members:

```
#include <A.hpp> // bring in full declaration of class A
...
B b;
b.pa = &a; // address of some instance of A
...
b.pa->func(); // invoke A's member function - this needs full declaration
```

For FIFO look into `std::queue`. Both `std::list`, `std::deque`, and `std::vector` could be used for that purpose, but also provide other facilities.

edited Jun 13 '10 at 21:01

answered Jun 13 '10 at 19:33

Nikolai Fetissov

64.5k 7 71 135

> Is this a good practice?

I don't see anything in particular wrong with it.

> What it means for compiler when you say that Node has a member with a type of it's pointer?

There's nothing wrong with a class storing a pointer to an object of the same class.

> And finally, if this is a bad practice, how I could implement a better FIFO.

I'd use `std::queue` ;)

answered Jun 13 '10 at 19:25
Cogwheel
12.9k 3 33 63

1 deque is way cooler than queue – Inverse Jun 13 '10 at 19:36

Obviously you are using linked-list as the underlying implementation of your queue. There's nothing particularly bad about that.

Just FYI though, that in terms of implementation, std::queue itself is using std::deque as its underlying implementation. std::deque is a more sophisticated data structure that consists of blocks of dynamic arrays that are cleverly managed. It ends up being better than linked list because:

1. With linked-list, each insertion means you have to do an expensive dynamic memory allocation. With dynamic arrays, you don't. You only allocate memory when the buffer has to grow.
2. Array elements are contiguous and that means elements access can be cached easily in hardware.

answered Jun 13 '10 at 19:45
ryaner
1,672 4 13 22

You can use the existing FIFO, std::queue.

answered Jun 13 '10 at 19:23
Stephen
27.1k 6 42 59

It is about implemenation. I know where to find a good container ;). – Narek Jun 13 '10 at 19:26

@Narek: I kinda figured that'd be the case, but didn't have time to write more :) I agree with the other comments - there's nothing wrong with your implementation, but using a `deque` would be better for performance. – Stephen Jun 13 '10 at 23:21

This is one good way of implementing a node. The node pointer is used to create the link to the next node in the container. You're right though, it can be used to create a loop. If the last node in the container references the first, iterating that container would loop through all of the nodes.

For example, if the container is a FIFO queue the pointer would reference the next node in the queue. That is, the value of `link_` would be the address of another instance of class `Node` .

If the value type `T` implemented an expensive copy constructor, a more efficient `Node` class would be

```
struct Node
{
    T * info_;
    Node* link_;
    Node(T * info, Node* link=0): info_(info), link_(link)
    {}
};
```

Note that `info_` is now a pointer to an instance of `T`. The idea behind using a pointer is that assigning a pointer is less expensive than copying complex objects.

answered Jun 13 '10 at 19:30

Nick Strupat
2,896 2 27 45

Ok, but pointers bring up ownership issues. Now who's responsible for deleting `info`? – Steven Sudit Aug 6 '10 at 4:55