

## Re: pointers to elements in containers

---

*Source:* <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.language/2006-09/msg00103.html>

---

- *From:* "Carl Daniel [VC++ MVP]" <[cpdaniel\\_remove\\_this\\_and\\_nospam@xxxxxxxxxxxxxxxxxxx](mailto:cpdaniel_remove_this_and_nospam@xxxxxxxxxxxxxxxxxxx)>
  - *Date:* Mon, 4 Sep 2006 11:40:48 -0700
- 

Paul wrote:

I have a class that looks like that:

```
struct Detail {
    std::list<const std::string> fields;
    std::list<const std::string*> key_fields;
};
```

The 'key\_fields' container is meant to store pointers to the std:string's in 'fields', so Detail's constructor looks like the following:

```
Detail::Detail()
{
    fields.push_back(std::string("string 1"));
    key_fields.push_back(&fields.back());

    fields.push_back(std::string("string 2"));
    key_fields.push_back(&fields.back());

    //...
}
```

This does not work: pointers to elements in 'fields' point to valid locations only until the end of the constructor after which they change (if addresses of elements stored in 'fields' were to be taken once the constructor has completed). (I have tried variations on the same theme but whatever the container – list or vector – and whatever the method of obtaining the address the underlying principle seems to remain the same: addresses of elements in a container storing objects change once the constructor has completed.)

Working on the above, I assumed the following:

1) Addresses of the elements in a vector may change after `resize()`, `push_back()`, `insert()`, `erase()` – perhaps a few others of the similar nature – but if you take care and call `resize()` or `reserve()` beforehand and then assign (through `[]`), they should not. (Initially I tried it with a vector, although later realised having a vector was

## Re: pointers to elements in containers

not the issue.)

That's correct.

2) Lists (and maps) are not affected by (1), as far as elements other than those immediately affected by the operation are concerned.

That's correct.

Could it be that real copies are created upon the exit from the constructor? How is this generally defined?

Is your Detail struct being copied? If it is, key\_fields will be broken – the pointers therein will continue to point into the original container, while the contents of fields will have been copied. A struct defined like this must be strictly non-copyable, unless you provide a suitable copy-constructor and assignment operator that adjusts the pointers (which could be tricky, depending on how hard it is to identify the "key" fields).

(The reason why I have two containers above is because not all of the fields required are actually keys – supposed to be unique and to contain data – so I thought I would store keys separately for ease of manipulation.)

Thank you.

–cd

.