Subject: Request: please make Range classes compatible with Vector!
Posted by piotr5 on Thu, 12 Oct 2017 05:20:23 GMT
View Forum Message <> Reply to Message

well, range isn't really moveable, actually it inherits moveable from the iterator it contains. however, I use it as Key to a map with iterator being just a pointer to some const characters. for that purpose I had to use NTL_MOVEABLE, create a "GetHashValue(const SubRangeClass&){return xxHash(...);}" and a specialization of SubRangeClass<const char*> containing an empty constructor.

the latter I see as a bug, SubRangeClass IMHO should have a constructor taking no arguments marked as protected, useable for its Vector friend. i.e. 2 changes: add containers making use of Create() or Add() or whatever to a list of friends for range classes, maybe put these additions into a friend-macro for others to use in their classes too. and then add SubRangeClass() constructor as a protected (or private) member.

The Range classes are a great addition to Upp! finally I can work with substrings without having to create yet another instance of the same text over and over again. however, in my observation substrings I actually need mostly for Keys in a Map, for example in a command-parser taking long commands or short ones which are a substring of the long commands. another use-case is indexing stuff for faster substring-search. it improves cache usage, often less memory is used too. if I had to enclose them in Value objects for storage, guess the overhead would kill some of these advantages...

Subject: Re: Request: please make Range classes compatible with Vector!
Posted by mirek on Thu, 12 Oct 2017 12:32:36 GMT
View Forum Message <> Reply to Message

piotr5 wrote on Thu, 12 October 2017 07:20well, range isn't really moveable, actually it inherits moveable from the iterator it contains. however, I use it as Key to a map with iterator being just a pointer to some const characters. for that purpose I had to use NTL_MOVEABLE, create a "GetHashValue(const SubRangeClass&){return xxHash(...);}" and a specialization of SubRangeClass<const char*> containing an empty constructor.

the latter I see as a bug, SubRangeClass IMHO should have a constructor taking no arguments marked as protected, useable for its Vector friend. i.e. 2 changes: add containers making use of Create() or Add() or whatever to a list of friends for range classes, maybe put these additions into a friend-macro for others to use in their classes too. and then add SubRangeClass() constructor as a protected (or private) member.

The Range classes are a great addition to Upp! finally I can work with substrings without having to create yet another instance of the same text over and over again. however, in my observation substrings I actually need mostly for Keys in a Map, for example in a command-parser taking long commands or short ones which are a substring of the long commands. another use-case is indexing stuff for faster substring-search. it improves cache usage, often less memory is used too. if I had to enclose them in Value objects for storage, guess the overhead would kill some of these advantages...

You got me scared...

I am afraid your usage goes far beyound the original intent, which is basically about having unified interface for algorithms, so that I can have the same simple algorithm to sort a vector or just slice of it.

The problem with range is that it is temporary view which exists only as long as its parent "range" (ultimately, a container) exists. I am afraid that using Range as anything else might be quite error prone.

A comment about substrings. Consider that you are fetching keys from some long text and storing to Map somehow. You will store the slice in something like

```
struct SubString {
  const char *begin;
  const char *end;
};
```

It might sound like saving the memory, but often is not, because sizeof(SubString) == 16, which is the same size as consumed by String for up to 14 characters. Worse, comparing small SubStrings is order of magnitude slower than comparing regular small Strings.

So the above optimisation would work only if you have "long" keys.

Mirek

Subject: Re: Request: please make Range classes compatible with Vector!
Posted by piotr5 on Thu, 12 Oct 2017 21:22:36 GMT
View Forum Message <> Reply to Message

thanks for the reminder I'm on a 64-bit system now with pointers taking up 8 bytes each. you're absolutely right it requires some caution, people aren't used to the idea of something like pointers or iterators losing their meaning when the container changes. I too experienced problems in past that got solved by switching from Vector to Array because the data I used wasn't truely moveable...

I admire the range classes because I tried to implement them myself for a file-management program: the string stores the full pathname, while the range alike class just stores the filename and maybe the directory-name they're in. so for example I'd have a directory with files named [0-9]*.pdf each representing pages scanned, and the directory-name contains the whole book-title and author-name and isbn and so on. the two together are definitely longer than 16 characters, and storing them in some Range class could help in calling some algorithms on them. another use-case I found was to store a 2D object in a Vector<SubRangeClass>, with each element pointing at a row of an image, thereby describing an outline of a (non-rectangular) area selected

by the user or an algorithm. again such ranges are definitely not really small, and even if some more compressed method for storing that info would exist, it's still quicker to alter the colours of some container of other container-alike objects than having to navigate in the actual image. (I hate drawing programs which insist that no image can ever be larger than 4GB, what am I supposed to use for high-resolution scans?...)

I agree that the speed-up isn't always noticable. if I'd store a substring in a range-alike class, the cpu-cache must load another 16 bytes in addition to the actual string, quite some waste when the same string is referred to from many places. so even a 32 byte string would see a speed-up in search compared to the SubString class -- unless string and SubString are stored within the same cache-line. but I must emphasize that strings aren't the only use-case that come into mind. I've used various objects as Keys into a Map object. floating-point numbers (or big rational numbers) for example take up quite some space, an std::array of them might determine some coordinates in an n-dimensional space, and I could take a look at a smaller sub-space by selecting a SubRangeClass or ViewRangeClass as a key mapping to further information of that particular place. the idea is, when working with the same vectors over and over again, in different sub-spaces, it might save on cache-consumption if I avoid copying those values...