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
