Subject: String improvements
Posted by hojtsy on Sat, 25 Feb 2006 17:12:23 GMT
View Forum Message <> Reply to Message

I added a feature to String::Find and String::ReverseFind that position could be also given relative to the end of the String, with -1 being equal to the length.

s.Find('R', -3) == s.Find('R', s.GetCount()+1-3);
s.ReverseFind('R', -3) == s.ReverseFind('R', s.GetCount()+1-3);

Implementation is below:template <class T, class S>
int AString<T, S>::Find(int chr, int from) const
{
 ASSERT(from >= -GetLength()-1 && from <= GetLength());
 if(from < 0)
  from += GetLength()+1;
 const T *e = End();
 for(const T *s = ptr + from; s < e; s++)
  if(*s == chr)
   return s - ptr;
 return -1;
}

template <class T, class S>
int AString<T, S>::ReverseFind(int chr, int from) const
{
 ASSERT(from >= -GetLength()-1 && from <= GetLength());
 if(from < 0)
  from += GetLength()+1;
 const T *s = ptr + from;
 while(--s >= ptr)
  if(*s == chr)
   return s - ptr;
 return -1;
}
This makes the AString::ReverseFind(int chr) unnecessary, because the ReverseFind(int chr, int from) could have -1 as default value for "from". Could you please add this improvement to uppsrc?

There is one trick, but it comes from the old implementation. If you invoke Find('R', x) it will find 'R' if it present at pos x, but ReverseFind('R', x) only starts to check at position x-1. So to start reverese checking from the last char you can use ReverseFind('R', -1), but to start forward checking from the last char you need to use Find('R', -2). Even though I find this unintuitive this is the existing behaviour of Find and ReverseFind for positive numbers too and changing it would break existing implementations. What is your opinion about changing this behaviour so that ReverseFind('R', x) starts checking at position x, and not x-1?

By the way, why is the chr parameter int? Shouldn't it rather be T?

Subject: Re: String improvements
Posted by mirek on Sat, 25 Feb 2006 21:59:07 GMT
View Forum Message <> Reply to Message

hojtsy wrote on Sat, 25 February 2006 12:12I added a feature to String::Find and String::ReverseFind that position could be also given relative to the end of the String, with -1 being equal to the length.

s.Find('R', -3) == s.Find('R', s.GetCount()+1-3);
s.ReverseFind('R', -3) == s.ReverseFind('R', s.GetCount()+1-3);

Implementation is below:template <class T, class S>
int AString<T, S>::Find(int chr, int from) const
{
 ASSERT(from >= -GetLength()-1 && from <= GetLength());
 if(from < 0)
  from += GetLength()+1;
 const T *e = End();
 for(const T *s = ptr + from; s < e; s++)
  if(*s == chr)
   return s - ptr;
 return -1;
}

template <class T, class S>
int AString<T, S>::ReverseFind(int chr, int from) const
{
 ASSERT(from >= -GetLength()-1 && from <= GetLength());
 if(from < 0)
  from += GetLength()+1;
 const T *s = ptr + from;
 while(--s >= ptr)
  if(*s == chr)
   return s - ptr;
 return -1;
}
This makes the AString::ReverseFind(int chr) unnecessary, because the ReverseFind(int chr, int from) could have -1 as default value for "from". Could you please add this improvement to uppsrc?

I will have think about it a little. It seems as logical extension, however there is one thing I do not exactly like about this:

So far, supplying "-1" as parameter there lead to runtime error, which was quite reasonable contract. I have catched some bugs via that definition.

My experience is that you are quite often playing arithmetic games with "from", and sometimes they go wrong. Allowing negative values would make some of them silently hidden.

But I have to check the code for "GetCount()" in the "String::Find" before makeing any final decision.

BTW, if negative values are to be allowed with this semantics, we should probably extend this to other methods....

Mirek

Subject: Re: String improvements
Posted by mirek on Sun, 26 Feb 2006 07:00:12 GMT
View Forum Message <> Reply to Message

[quote title=luzr wrote on Sat, 25 February 2006 16:59][quote

But I have to check the code for "GetCount()" in the "String::Find" before makeing any final decision.

Mirek[/quote]

Well, I have checked the whole codebase and there were _zero_ cases (unless I missed something) where negative offset would make things easier. Given the bug-catching ability of negative values, for the moment I vote for not adding negative offsets to the U++...

What usage scenario (particalar case) led you to this extension?

Mirek

Subject: Re: String improvements
Posted by hojtsy on Sun, 26 Feb 2006 11:10:40 GMT
View Forum Message <> Reply to Message

OK, if negative offsets will not be supported, would it be possible to include these other minor modifications:
 - ReverseFind should ASSERT for invalid values of "from"
 - the loop in ReverseFind should be fixed to avoid reading into the memory before the first char. See the fix in my post above.
 - the type of chr parameter should be T and not int.

Also I raised another problem with ReverseFind: ReverseFind(c, x) starts to search at position x-1. This is unintuitive and different from both std::string and QString, in which the reverese find method starts from the position given as parameter. Would it be possible to change this? (backward incompatible change)

As for the reasons to support negative offsets:
- brevity: If I would like to search in the last x chars of a String calculated with a complex

exression, it would be shorter to write:
longExpression->something().Format(somemore).Find(something, -3) instead of creating separate local variable to store the String, and doing s.Find(something, s.GetLength()-3).
- conformance to expectations: QString has it. People coming from Qt will expect this behaviour.

Maybe an other parameter could solve the problem of brevity:
Find(int chr, int from = 0, OffsetDirection dir = FromStart)
So instead of negative offset, you can use the third param to specify if the offset is calculated relative to the start or the end.

Here is another suggestion for a new method in Stringtemplate <class T, class S>
bool AString<T, S>::ContainsAt(const S &s, int pos) const
{
 ASSERT(pos >= 0);
 if(pos < 0 || pos + s.GetCount() > GetCount()) return false;
 return memcmp(ptr + pos, s.Begin(), s.GetCount()) == 0;
}

Subject: Re: String improvements
Posted by mirek on Sun, 26 Feb 2006 11:36:00 GMT
View Forum Message <> Reply to Message

hojtsy wrote on Sun, 26 February 2006 06:10OK, if negative offsets will not be supported, would it be possible to include these other minor modifications:
 - ReverseFind should ASSERT for invalid values of "from"
 - the loop in ReverseFind should be fixed to avoid reading into the memory before the first char. See the fix in my post above.
 - the type of chr parameter should be T and not int.

Definitely!

Quote:
Also I raised another problem with ReverseFind: ReverseFind(c, x) starts to search at position x-1. This is unintuitive and different from both std::string and QString, in which the reverese find method starts from the position given as parameter. Would it be possible to change this? (backward incompatible change)

Well, I think we should be able to afford that change. I will have to debate it this with Tom a little though...

Quote:
As for the reasons to support negative offsets:
- brevity: If I would like to search in the last x chars of a String calculated with a complex exression, it would be shorter to write:
longExpression->something().Format(somemore).Find(something, -3)

instead of creating separate local variable to store the String, and doing s.Find(something, s.GetLength()-3).

The tradeoff there is how often you need something like that vs. bug catching. All I can say at the moment is that I never needed to search through last n characters (by checking my codebase).

Generally, I am quite reluctant adding features that just "could be useful sometimes". Means, I need real-world example before considering it further.

Quote:
Maybe an other parameter could solve the problem of brevity:
Find(int chr, int from = 0, OffsetDirection dir = FromStart)

So instead of negative offset, you can use the third param to specify if the offset is calculated relative to the start or the end.

Well, considering usage scenarios you have gave me so far (I mean, the above example), I would prefer separate method like
"FindInLast" or something like that (less typing, simple inliner,, bug catching ability retained).

In any case, this has to wait after 602... (after AttrText disaster, I would like to keep the code-base as stable as possible during next week...).

Quote:
Here is another suggestion for a new method in Stringtemplate <class T, class S>
bool AString<T, S>::ContainsAt(const S &s, int pos) const
{
 ASSERT(pos >= 0);
 if(pos < 0 || pos + s.GetCount() > GetCount()) return false;
 return memcmp(ptr + pos, s.Begin(), s.GetCount()) == 0;
}
[/quote]

Well, there is a couple of methods (or external functions) I am missing in String and something like this would certainly be useful.

Mirek

Subject: Re: String improvements
Posted by rylek on Sun, 26 Feb 2006 14:39:55 GMT
View Forum Message <> Reply to Message

As concerns the exact meaning of the "start character position" in ReverseFind, I'm quite for the proposed change (to search from character with the index n, not n-1). In the matter of negative string offsets (e.g. in the Find / ReverseFind method) I must admit that I used to be a big fan of

such "useful hacks", but over the years I moved a lot in Mirek's direction and I'm more and more willing to sacrifice a line or two of code if I gain a cleaner and more intuitive interface.

I believe the negative string offsets are a good example, a very similar situation arises also in the container interfaces. If we decide to implement negative character offsets for Find / ReverseFind, it seems to me counter-intuitive not to implement them everywhere else, like in the Left / Right / Mid methods for starters. After that we'll start to consider using the same syntax for the [] operator, like using s[-1] to address the last string character, and this will be the moment when the real trouble begins, because the necessary runtime check will potentially cost us some performance, with the dubious by-effect of introducing a feature which is quite as likely to come handy in certain situations as it's error-prone in the sense that it may hide erroneous code design (like searching a string which is not long enough) in a relatively tricky way.

Come to it, I don't have anything against a new method to search a terminal portion of the string. I'm afraid that my experience and opinion in this matter is a bit similar to Mirek's in that I use even the existing Find / ReverseFind methods relatively seldom (by a simple check I've just found a single occurrence of ReverseFind in all my "commercial" project source code). However I'm not very fond of this "statistical programming" (I don't use a feature, ipso facto it is not necessary or useful) and a few practical examples are quite likely to convince me that we do good to the U++ String interface by enriching it with a few additional search functions (by the way, the one I really do miss is searching a string for a substring, which, if I'm not mistaken, is still missing).

Regards

Tomas

Subject: Re: String improvements
Posted by mirek on Sun, 26 Feb 2006 15:27:05 GMT
View Forum Message <> Reply to Message

OK, I did it again... (last minute additions):

String now has altered ReverseFind "from" and I have (finally) added substring search.

I hope I have not broken the 602 release

Mirek
