Subject: C strings
Posted by Shire on Sun, 08 Oct 2006 16:31:03 GMT
View Forum Message <> Reply to Message

C-like strings have a long history. But them have no length information, which can be unsafe (while(*p) {...}) and performance loss (many strlen() calls). Microsoft marks many of c-like functions as deprecated.
U++ code have many functions with only const char* parameter. Is reasonable to change type of this parameters to some like ConstString?

```
#define cs_(buff) ConstString(buff, sizeof(buff))
#define s_(buff) ConstString(buff, strlen(buff))

class ConstString
{
  const char* pBuffer;
  unsigned length;
public
  ConstString(const char* p, unsigned length);
  // operators, accessors, etc..
}
```

Subject: Re: C strings
Posted by mirek on Tue, 10 Oct 2006 07:37:34 GMT
View Forum Message <> Reply to Message

char * is used in chances are that string itself will be supplied as constant or other char * (avoiding performance and code size loss of char * -> String conversion) AND there is no further string length testing (or when it is negligible performance wise).

Subject: Re: C strings
Posted by Shire on Tue, 10 Oct 2006 19:38:59 GMT
View Forum Message <> Reply to Message

Quote:char * is used in chances are that string itself will be supplied as constant or other char * (avoiding performance and code size loss of char * -> String conversion) AND there is no further string length testing (or when it is negligible performance wise).

Not in all cases. For example,

```
void CodeEditor::Enclose(const char *c1, const char *c2)
{
 int l, h;
```

```
 if(!GetSelection(l, h))
  return;
 Insert(l, WString(c1));
 Insert(h + strlen(c1), WString(c2));
 ClearSelection();
 SetCursor(h + strlen(c1) + strlen(c2));
}
```

Calculating the size of const string is already overhead.
For non-const strings, determination of the end of string by zero terminator is unsafe, because it may be lost, and work with this string can make security hole.
I know and understand, that removing C-like strings is huge work (and code works without it, really?), but IMHO, if you attempted on STL, you can make library safer and faster.

Subject: Re: C strings
Posted by mirek on Wed, 11 Oct 2006 02:26:59 GMT
View Forum Message <> Reply to Message

Quote:
(or when it is negligible performance wise).

Quote:
Not in all cases. For example,

```
void CodeEditor::Enclose(const char *c1, const char *c2)
{
 int l, h;
 if(!GetSelection(l, h))
  return;
 Insert(l, WString(c1));
 Insert(h + strlen(c1), WString(c2));
 ClearSelection();
 SetCursor(h + strlen(c1) + strlen(c2));
}
```

Calculating the size of const string is already overhead.

You are not seeing the whole story. Enclose is always called with const char * literal parameters (of short lentgth). So makeing parameters String would enlarge the code size, as strlen has to be done anyway and you would just have to call String constructor twice and before each Enclose call.

Quote:
For non-const strings, determination of the end of string by zero terminator is unsafe, because it may be lost, and work with this string can make security hole.

I do not think this is true. How could that be lost?

The fixed buffers for strings are security reasons, not trailing '\0' sentinels.

Quote:
I know and understand, that removing C-like strings is huge work (and code works without it, really?), but IMHO, if you attempted on STL, you can make library safer and faster.

Actually, that would be nice, but is a bit problematic because of nature of C++ string literals and because of external environment. (C++ string literals being the major problem).
