Strings in C++

Session 11, Wednesday January 19th, 2011

Michael Pereira Neves

Overview

- cstring
- std::string
- String Implementations
- String Optimizations
- stringstream
- Some String-Functions (find, multibyte)
- Performance Java vs. C++

Libraries & Compiler

- There are different String-Libraries in C++
  - string.h (C Standard Library)
  - std::string (Standard Template Library)
  - qString
- Their implementation is Compiler dependant

cstring

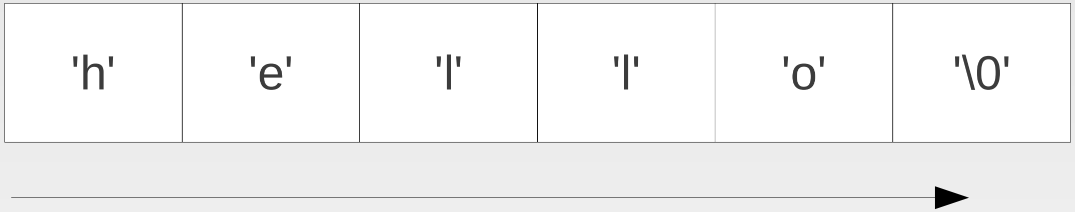
- Is part of the C Standard Library
- Stored as a char array
- Example:
  - char message[] = “hello”;

'h'	'e'	'l'	'l'	'o'	'\0'

- Null Terminated Character Sequence
  - '\0' marks the end of the String
  - Because of limited memory at that time

strlen

- The string-length is determined by searching for the first ocurrance of the '\0’ character
  - strlen(“Hello”) → 5



'h'	'e'	'l'	'l'	'o'	'\0'

  - Which takes O(n) and is therefore rather bad for String processing

Dynamic strings

- The size of a cstring can be defined dynamically during runtime
  - char *msg = new char[length]
- But cannot be resized afterwards...
- You have to manually allocate memory
  - char* newmsg = malloc(strlen(msg) + 1);
- And use strcpy or strcat to copy or concatenate the string into the new one
  - strcpy(newmsg, msg);
- Which is unhandy and prone to errors...

cstring

- Common errors when using cstrings
  - Not allocating additional space for '\0'
    - char msg[5];
    - *msg = “Hallo”;   //terminating '\0' is not in msg
  - Causing buffer overflows
    - char msg[1];   //@0x7fffc71e2a4f
    - char msg2[1];   //@0x7fffc71e2a4e
    - strcpy(msg2,”abc”);   //copy “abc” in msg2
    - cout << msg[0];   //returns “b”
  - Both can result in undefined behaviour!

std::string

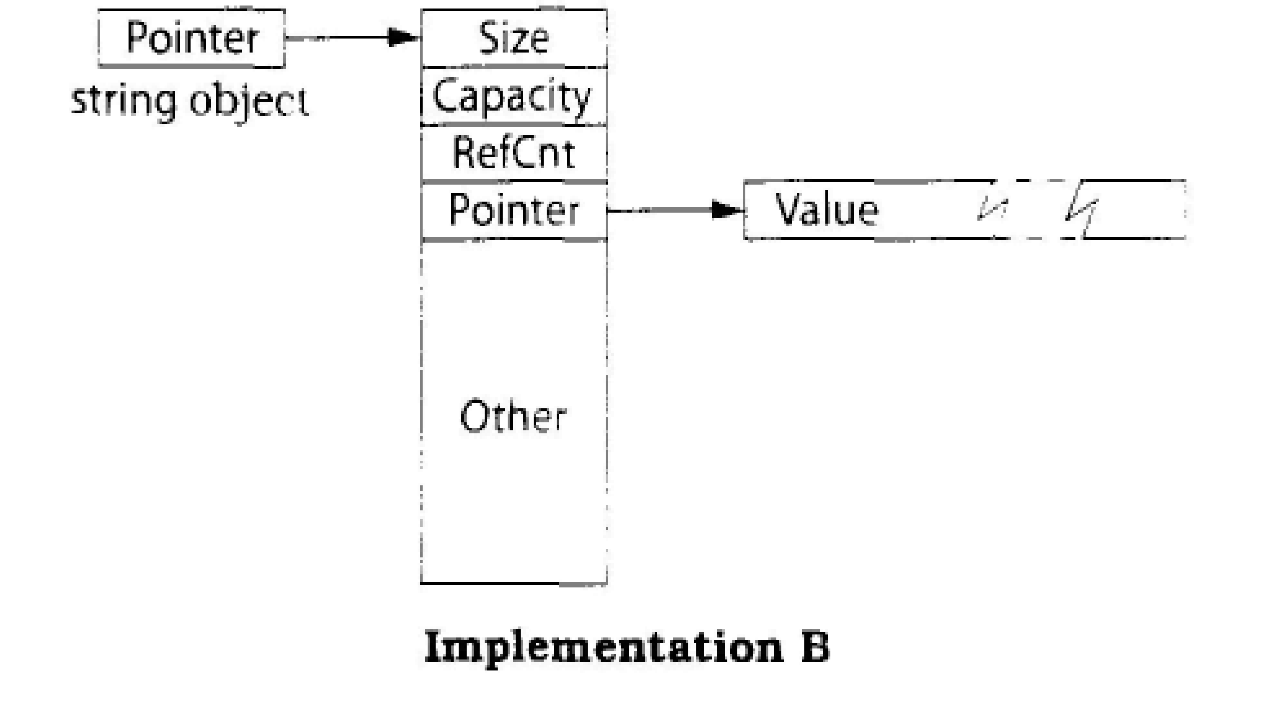
- Is a class of the C++ Standard Template Library
- Removes many of the problems with cstrings
  - memory allocation, null termination
- Offers useful String-functions like
  - Comparison, concatenation, find,...
- Can be constructed from a cstring and converted to a cstring again

std::string

- The string class is an instantiation of the basic_string class template
  - typedef basic_string<char> string;
  - template<charT, char_traits<charT>, allocator<charT> >
- Thus the string class can handle different character types, like char (8bit) or wchar (32bit)
- Or even user defined objects

String Implementation

- The String Implementation in gcc



Overview

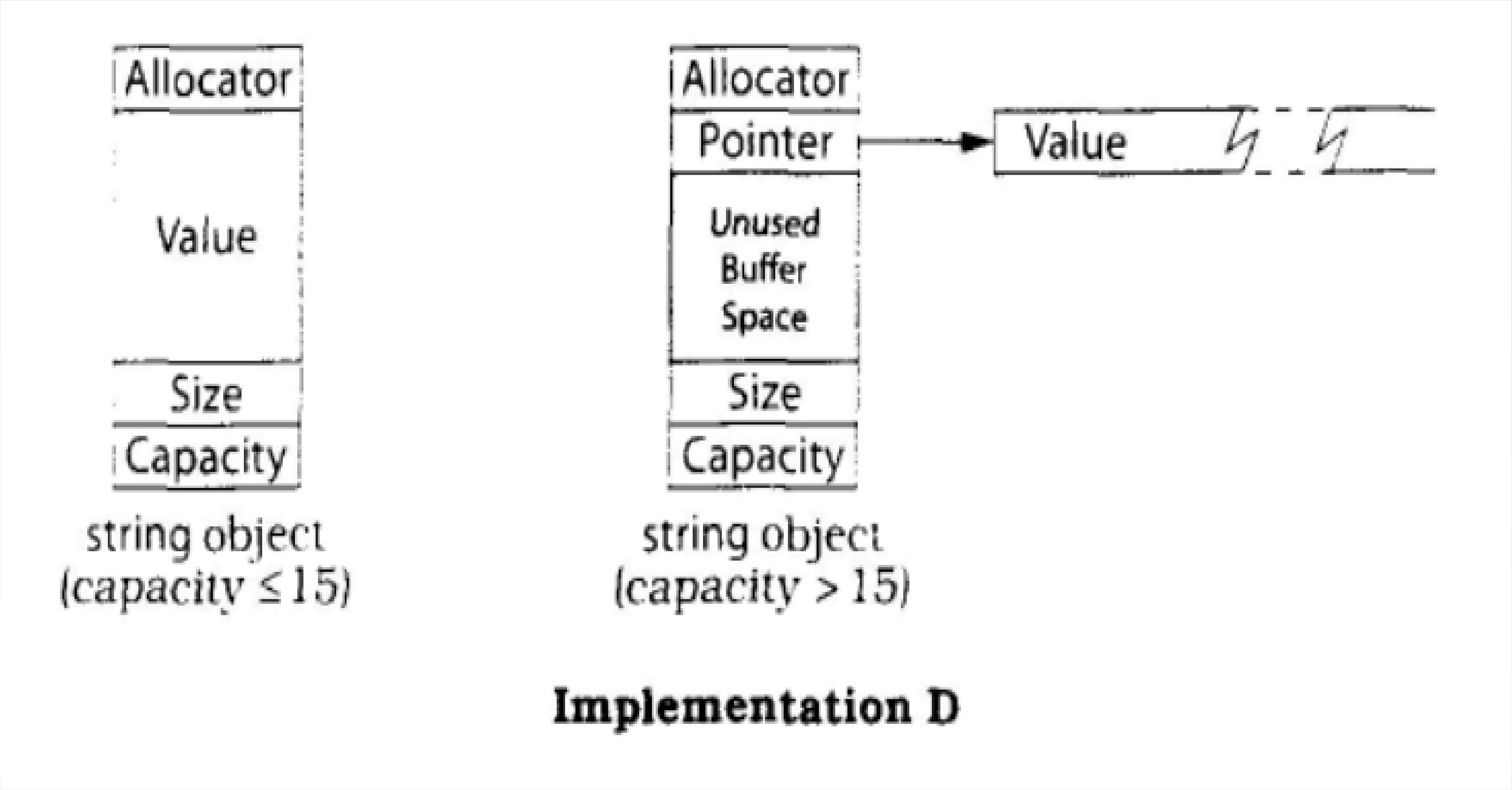
- cstring
- std::string
- String Implementations
- String Optimizations
- stringstream
- Some String-Functions (find, multibyte)
- Performance Java vs. C++

String Optimizations

- Small-String-Optimization
- Reserve
- Shrink-to-fit
- Lazy Evaluation (Copy-On-Write)
- Reference Counter
- Call-By-Reference

Small string optimization

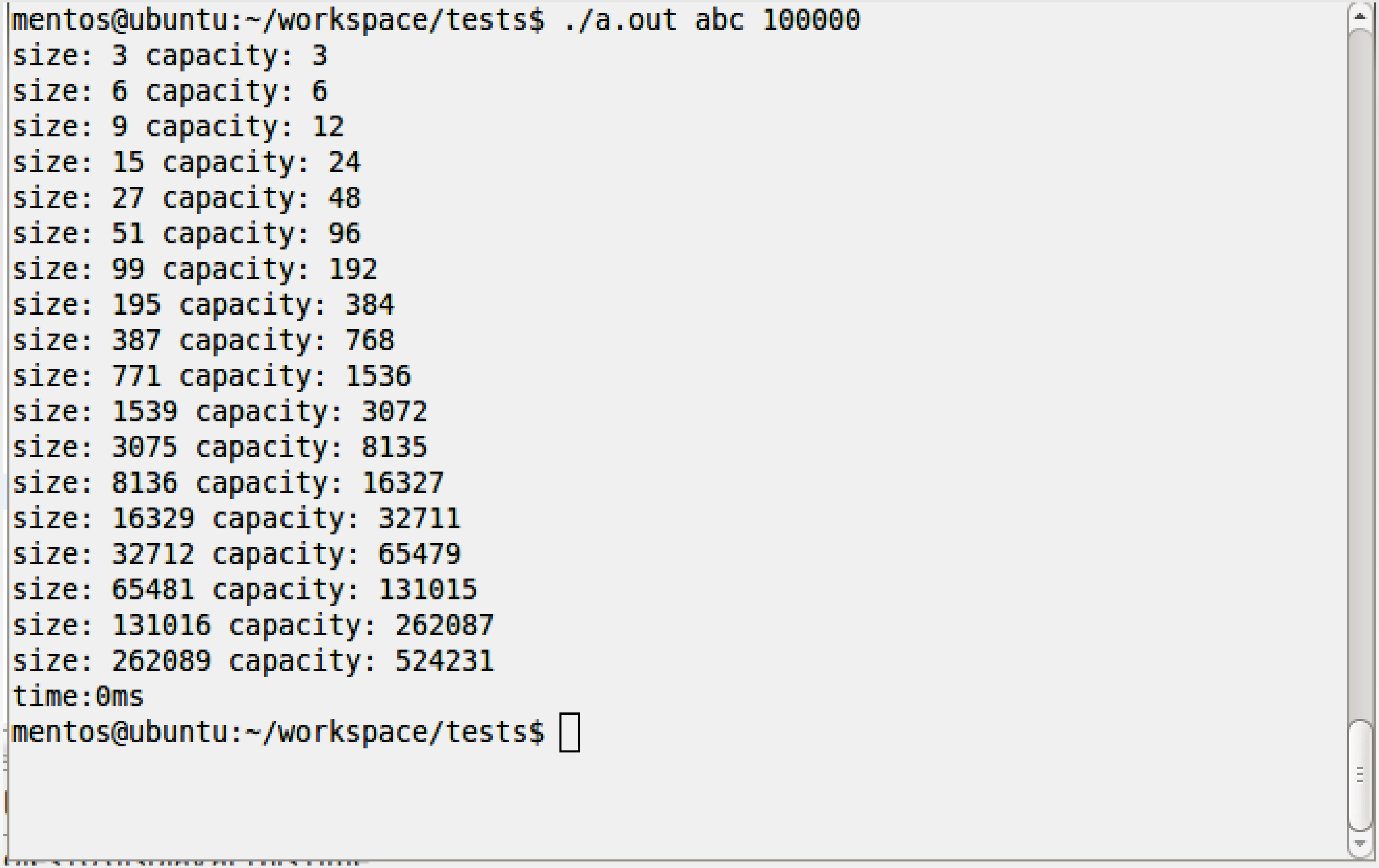
- Some Libraries implement small string optimization (like boost::const_string<>)



String growth

- Strings can grow dynamically with append or the operator +=
  - String text = “Hello “; text += “World”;
- A “realloc”-like operation is triggered
  - Allocate a new block of memory
    - Multiple of the current capacity (e.g. factor of 2)
  - Copy all elements from the String's old memory to the new one.
  - Destroy the object in the old memory
  - Deallocate the memory

String growth



```
mentos@ubuntu:~/workspace/tests$ ./a.out abc 100000
size: 3 capacity: 3
size: 6 capacity: 6
size: 9 capacity: 12
size: 15 capacity: 24
size: 27 capacity: 48
size: 51 capacity: 96
size: 99 capacity: 192
size: 195 capacity: 384
size: 387 capacity: 768
size: 771 capacity: 1536
size: 1539 capacity: 3072
size: 3075 capacity: 8135
size: 8136 capacity: 16327
size: 16329 capacity: 32711
size: 32712 capacity: 65479
size: 65481 capacity: 131015
size: 131016 capacity: 262087
size: 262089 capacity: 524231
time:0ms
mentos@ubuntu:~/workspace/tests$ 
```

reserve

- Use reserve to avoid unnecessary allocations
- Reserve(size_t n) forces the container to change it's capacity to at least n.
- As long as str.size() < str.capacity there is no need to reallocate memory
- If you can approximate how many elements will end up in your container, use reserve!
- Another Strategy is to reserve the maximum space you could ever need, then once you've added all your data, trim off any excess capacity

Shrink-to-fit

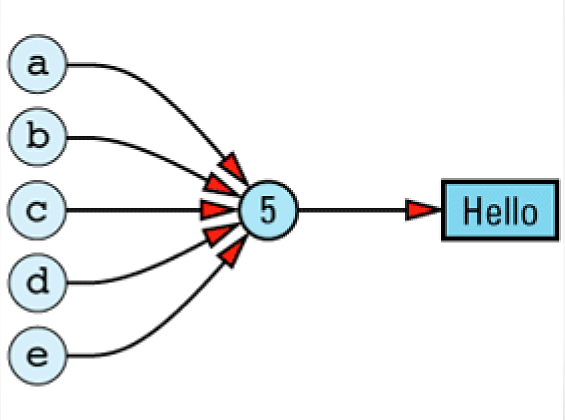
- Use swap() to fit the capacity to your actual string size.
  - String s;
  - s.reserve(1000);
  - //fill s...
  - s.swap(s); or s.reserve(0);
- All Elements are copied by s's copy constructor, but only as much memory is allocated as needed for all elements.

Lazy Evalution (Copy-on-Write)

- Naïve approach:
  - String s1 = “Hello”;
  - String s2 = s1;   //copy constructor of s2 is called
- Why doing an expensive copy when s2 hasn't been used yet?
- Better: make s2 a reference to s1!
- And just defer the copy work until s2 is really modified!

Reference counting

- Count how many references are made to an object.
- When nobody refers to that object, it destroys itself
- Saves Memory and time, no need to construct and destruct copies of the same object value.



Call-By-Reference

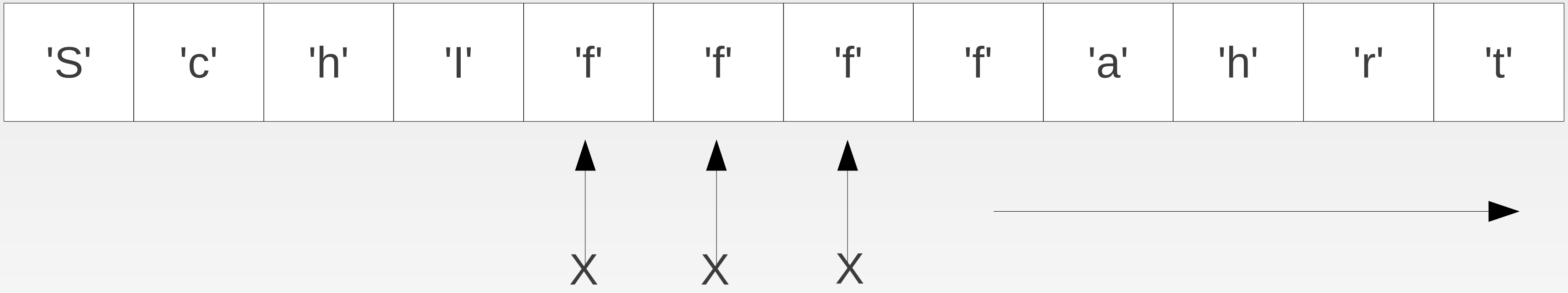
- call-by-reference
  - Passing Strings to functions:
    - Void print_the_string(string str);
    - A temporary string object is generated and the copy constructor is called
  - Copying the String takes O(n) + time to allocate the heap memory
  - Better: Use a reference when passing Strings
    - void print_the_string(string& str)
    - Local variable str now refers to the String

Overview

- cstring
- std::string
- String Implementations
- String Optimizations
- stringstream
- Some String-Functions (find, multibyte)
- Performance Java vs. C++

string::find

- size_t find (const string& str, size_t pos = 0) const;
- string str("Schifffahrt");
- size_t found = str.find(“fahrt”);



'S'	'c'	'h'	'l'	'f'	'f'	'f'	'f'	'a'	'h'	'r'	't'

stringstream

- Provides an interface to manipulate strings as if they were input/output streams
- Maintains pointer to a stringbuf object
- The stringbuffer associates the input or output sequence with a sequence of arbitrary characters
- When characters are written to the stream, if the write position goes beyond the buffer end, stringstream automatically increase the buffer size

Multibyte Functions

- wctomb (char *string, wchar_t wchar)
  - converts the wide character code wchar to its corresponding multibyte character sequence
- Example:
  - Character: M
    - UTF-8: 0x4D
    - UTF-32:0x0000004D
  - Character: 二 (Japan thing)
    - UTF-8: 0xE4BA8C
    - UTF-32:0x00004E8C

Sources

- Scott Meyers. 1998. Effective C++ (2nd Ed.): 50 Specific Ways to Improve Your Programs and Designs. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
- Scott Meyers. 1995. More Effective C++: 35 New Ways to Improve Your Programs and Designs. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
- Scott Meyers. 2001. Effective STL: 50 Specific Ways to Improve Your Use of the Standard Template Library. Addison-Wesley Longman Ltd., Essex, UK, UK.
- Optimizations That Aren't (In a Multithreaded World)
  - http://www.gotw.ca/publications/optimizations.htm
- Google ;-)