```
 1: // $Id: gnu-string.cpp,v 1.21 2016-08-03 16:23:58-07 - - $
 2:
 3: // G++ basic_string for gcc 4.*.
 4: // Documentation taken from source code.
 5: // Code cleaned up a little.
 6:
 7: #include <cstring>
 8: #include <iostream>
 9: using namespace std;
10:
11: template <typename char_type>
12: struct basic_gnustr {
13:    struct repr_ {
14:       size_t size_;
15:       size_t capacity_;
16:       size_t refcount_;
17:    }; // NOTE:  not a field.
18:    static constexpr size_t repr_size = sizeof (repr_);
19:    static constexpr size_t char_size = sizeof (char_type);
20:    static constexpr size_t repr_chars = repr_size / char_size;
21:    char_type* pointer_;
22:
23:    repr_* repr() {
24:       repr_* repr_addr = reinterpret_cast<repr_*>(pointer_);
25:       return &repr_addr[-1];
26:    }
27:
28:    size_t size() { return repr()->size_; }
29:    size_t capacity() { return repr()->capacity_; }
30:    char_type& operator[] (size_t index) { return pointer_[index]; }
31:    const char_type* c_str() { return pointer_; }
32:
33:    basic_gnustr(): pointer_(nullptr) {}
34:    ~basic_gnustr() { if (pointer_) delete[] repr(); }
35:
36:    basic_gnustr (size_t size) {
37:       pointer_ = new char_type[repr_chars + size + 1] + repr_chars;
38:       repr()->size_ = size;
39:       repr()->capacity_ = size + 1;
40:       repr()->refcount_ = 1;
41:       pointer_[0] = 0;
42:    }
43:
44:    basic_gnustr (const char_type* str): basic_gnustr (
45:       [](const char_type* begin) {
46:          const char_type* end = begin;
47:          while(*end++) continue;
48:          return end - begin;
49:       }(str)
50:    ) {
51:       memcpy (pointer_, str, sizeof (char_type) * size());
52:       pointer_[size()] = 0;
53:    }
54:
55: };
56:
```

```
 57:
 58: using gnu_string = basic_gnustr<char>;
 59:
 60: #define SHOW(X) cout << #X << " = " << X << endl;
 61: int main() {
 62:    gnu_string s ("Hello");
 63:    SHOW (s.repr());
 64:    SHOW (static_cast<void*>(s.pointer_));
 65:    SHOW (s.repr()->size_);
 66:    SHOW (s.repr()->capacity_);
 67:    SHOW (s.repr()->refcount_);
 68:    SHOW (s.pointer_);
 69:    SHOW (s.c_str());
 70: }
 71:
 72: //TEST// valgrind gnu-string >gnu-string.out 2>gnu-string.err
 73: //TEST// more gnu-string.out gnu-string.err >gnu-string.lis </dev/null
 74: //TEST// rm gnu-string.out gnu-string.err
 75: //TEST// mkpspdf gnu-string.ps gnu-string.cpp* gnu-string.lis
 76:
 77: /**
 78: * @class basic_string basic_string.h <string>
 79: * @brief  Managing sequences of characters and character-like objects.
 80: *
 81: * @ingroup strings
 82: * @ingroup sequences
 83: *
 84: * @tparam _CharT  Type of character
 85: * @tparam _Traits  Traits for character type, defaults to
 86: *                  char_traits<_CharT>.
 87: * @tparam _Alloc  Allocator type, defaults to allocator<_CharT>.
 88: *
 89: * Meets the requirements of a <a href="tables.html#65">container</a>, a
 90: * <a href="tables.html#66">reversible container</a>, and a
 91: * <a href="tables.html#67">sequence</a>.  Of the
 92: * <a href="tables.html#68">optional sequence requirements</a>, only
 93: * @c push_back, @c at, and @c %array access are supported.
 94: *
 95: * @doctodo
 96: *
 97: *
 98: * Documentation?  What's that?
 99: * Nathan Myers <ncm@cantrip.org>.
100: *
```

```
101:
102: * A string looks like this:
103: *
104: * @code
105: *                                        [_Rep]
106: *                                        _M_length
107: *  [basic_string<char_type>]             _M_capacity
108: *  _M_dataplus                           _M_refcount
109: *  _M_p ---------------->                unnamed array of char_type
110: * @endcode
111: *
112: * Where the _M_p points to the first character in the string, and
113: * you cast it to a pointer-to-_Rep and subtract 1 to get a
114: * pointer to the header.
115: *
116: * This approach has the enormous advantage that a string object
117: * requires only one allocation.  All the ugliness is confined
118: * within a single %pair of inline functions, which each compile to
119: * a single @a add instruction: _Rep::_M_data(), and
120: * string::_M_rep(); and the allocation function which gets a
121: * block of raw bytes and with room enough and constructs a _Rep
122: * object at the front.
123: *
124: * The reason you want _M_data pointing to the character %array and
125: * not the _Rep is so that the debugger can see the string
126: * contents. (Probably we should add a non-inline member to get
127: * the _Rep for the debugger to use, so users can check the actual
128: * string length.)
129: *
130: * Note that the _Rep object is a POD so that you can have a
131: * static <em>empty string</em> _Rep object already @a constructed before
132: * static constructors have run.  The reference-count encoding is
133: * chosen so that a 0 indicates one reference, so you never try to
134: * destroy the empty-string _Rep object.
135: *
136: * All but the last paragraph is considered pretty conventional
137: * for a C++ string implementation.
138: */
```

```
    1: @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: starting gnu-string.cpp
    2: gnu-string.cpp: $Id: gnu-string.cpp,v 1.21 2016-08-03 16:23:58-07 - - $
    3: g++ -g -O0 -std=gnu++14 -rdynamic -Wall -Wextra -Wold-style-cast gnu-str
ing.cpp -o gnu-string -lglut -lGLU -lGL -lX11 -lrt -lm
    4: cpplint.py.perl gnu-string.cpp
    5: Done processing gnu-string.cpp
    6: @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: finished gnu-string.cpp
```

```
    1: ::::::::::::::
    2: gnu-string.out
    3: ::::::::::::::
    4: s.repr() = 0x9c9b090
    5: static_cast<void*>(s.pointer_) = 0x9c9b0a8
    6: s.repr()->size_ = 6
    7: s.repr()->capacity_ = 7
    8: s.repr()->refcount_ = 1
    9: s.pointer_ = Hello
   10: s.c_str() = Hello
   11: ::::::::::::::
   12: gnu-string.err
   13: ::::::::::::::
   14: ==15487== Memcheck, a memory error detector
   15: ==15487== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al
.
   16: ==15487== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright
info
   17: ==15487== Command: gnu-string
   18: ==15487==
   19: ==15487==
   20: ==15487== HEAP SUMMARY:
   21: ==15487==     in use at exit: 0 bytes in 0 blocks
   22: ==15487==   total heap usage: 2 allocs, 2 frees, 47 bytes allocated
   23: ==15487==
   24: ==15487== All heap blocks were freed -- no leaks are possible
   25: ==15487==
   26: ==15487== For counts of detected and suppressed errors, rerun with: -v
   27: ==15487== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 1 from 1)
```