```
 1: // $Id: astring.cpp,v 1.30 2015-01-26 14:28:34-08 - - $
 2:
 3: //
 4: // NAME
 5: //    astring - trivial implementation of a string using an array
 6: //
 7: // DESCRIPTION
 8: //    We show how to implement a simple string class.
 9: //
10:
11: #include <cstdlib>
12: #include <cstring>
13: #include <iostream>
14: #include <sstream>
15: #include <stdexcept>
16: #include <string>
17:
18: using namespace std;
19:
20: //////////////////////////////////////////////////////////////////
21: // astring.h
22: //////////////////////////////////////////////////////////////////
23:
24: class astring {
25:    private:
26:       static constexpr size_t DEFAULT_CAPACITY = 16;
27:       size_t capacity_;
28:       size_t size_;
29:       char* buffer_;
30:       void range_check (size_t pos, const char* id) const;
31:       void copy_from (const astring&);
32:       void clear_that (astring&);
33:    public:
34:
35:       // override implicit members
36:       astring();                                  // default ctor
37:       astring (const astring&);                   // copy ctor
38:       astring& operator= (const astring&);        // operator=
39:       ~astring();                                 // dtor
40:       astring (astring&&);                        // move ctor
41:       astring& operator= (astring&&);             // move operator=
42:
43:       // other members
44:       astring (const char* );                   // "" ctor
45:       explicit astring (size_t);                // length reservation
46:       astring& operator= (const char*);         // operator=
47:       astring& operator+= (const char);         // += char
48:       astring& operator+= (const char*);        // += char*
49:       char operator[] (size_t pos) const;       // const subscript =[]
50:       char& operator[] (size_t pos);            // ref subscript []=
51:       void reserve (size_t);                    // ensure buffer size;
52:       size_t size() const;                      // strlen
53:       size_t capacity() const;
54:       const char* c_str() const;                // borrow string in C fmt
55:       friend ostream& operator<< (ostream&, const astring&);
56: };
```

```
 57:
 58: //////////////////////////////////////////////////////////////////
 59: // astring.cpp
 60: //////////////////////////////////////////////////////////////////
 61:
 62: void astring::range_check (size_t pos, const char* id) const {
 63:    if (pos < size_) return;
 64:    throw out_of_range (id);
 65: }
 66:
 67: void astring::copy_from (const astring& that) {
 68:    reserve (that.size_ + 1);
 69:    size_ = that.size_;
 70:    strcpy (buffer_, that.buffer_);
 71: }
 72:
 73: void astring::clear_that (astring& that) {
 74:    that.size_ = that.capacity_ = 0;
 75:    that.buffer_ = NULL;
 76: }
 77:
 78: astring::astring(): capacity_ (DEFAULT_CAPACITY), size_ (0),
 79:                     buffer_ (new char[DEFAULT_CAPACITY]) {
 80:    buffer_[size_] = '\0';
 81: }
 82:
 83: astring::astring (const astring& that): capacity_ (that.capacity_),
 84:                     buffer_ (new char[that.capacity_]) {
 85:    copy_from (that);
 86: }
 87:
 88: astring& astring::operator= (const astring& that) {
 89:    if (this !=& that) copy_from (that);
 90:    return *this;
 91: }
 92:
 93: astring::astring (astring&& that): capacity_ (that.capacity_),
 94:                     size_ (that.size_), buffer_ (that.buffer_) {
 95:    clear_that (that);
 96: }
 97:
 98: astring& astring::operator= (astring&& that) {
 99:    if (this !=& that) {
100:       capacity_ = that.capacity_;
101:       size_ = that.size_;
102:       buffer_ = that.buffer_;
103:       clear_that (that);
104:    }
105:    return *this;
106: }
107:
108: astring::~astring() {
109:    if (buffer_ != NULL) delete[] buffer_;
110: }
111:
```

```
112:
113: astring::astring (const char* that) {
114:    size_ = strlen (that);
115:    capacity_ = size_ + 1;
116:    buffer_ = new char [capacity_];
117:    strcpy (buffer_, that);
118: }
119:
120: astring::astring (size_t capacity): capacity_ (capacity), size_ (0),
121:                     buffer_ (new char[size_]) {
122:    buffer_[size_] = '\0';
123: }
124:
125: astring& astring::operator= (const char* that) {
126:    size_ = strlen (that);
127:    reserve (size_ + 1);
128:    strcpy (buffer_, that);
129:    return *this;
130: }
131:
132: astring& astring::operator+= (const char achar) {
133:    ++size_;
134:    reserve (size_ + 1);
135:    buffer_[size_ - 1] = achar;
136:    buffer_[size_] = '\0';
137:    return *this;
138: }
139:
140: astring& astring::operator+= (const char* cstr) {
141:    size_ += strlen (cstr);
142:    reserve (size_ + 1);
143:    strcat (buffer_, cstr);
144:    return *this;
145: }
146:
147: char astring::operator[] (size_t pos) const {
148:    range_check (pos, "operator[]");
149:    return buffer_[pos]; // no bounds check
150: }
151:
152: char& astring::operator[] (size_t pos) {
153:    range_check (pos, "operator[]");
154:    return buffer_[pos]; // no bounds check
155: }
156:
157: void astring::reserve (size_t capacity) {
158:    if (capacity < capacity_) return;
159:    capacity_ *= 2;
160:    if (capacity_ < capacity) capacity_ = capacity + 1;
161:    char* oldbuffer_ = buffer_;
162:    buffer_ = new char[capacity_];
163:    strcpy (buffer_, oldbuffer_);
164:    delete[] oldbuffer_;
165: }
166:
```

```
167:
168: size_t astring::size() const {
169:    return size_;
170: }
171:
172: const char* astring::c_str() const {
173:    return buffer_;
174: }
175:
176: ostream& operator<< (ostream& out, const astring& that) {
177:    out << that.buffer_;
178:    return out;
179: }
180:
181: //////////////////////////////////////////////////////////////////
182: // main.cpp
183: //////////////////////////////////////////////////////////////////
184:
185: int main (int argc, char** argv) {
186:    astring first = "Hello, World!";
187:    cout << "first=" << first << endl;
188:    astring second;
189:    second = first;
190:    second += 'x'; second += 'y';
191:    for (int i = 0; i < 3; ++i) second[i] = i + '1';
192:    cout << "second=" << second << endl;
193:    for (size_t i = 5; i < second.size(); ++i) {
194:       cout << second[i] << endl;
195:    }
196:    astring allargs = "args:";
197:    for (char** arg = &argv[1]; arg < &argv[argc]; ++arg) {
198:       (allargs += " ") += *arg;
199:    }
200:    cout << allargs << endl;
201:    cout << allargs.c_str() << endl;
202:    return EXIT_SUCCESS;
203: }
204:
205: /*
206: //TEST// valgrind --leak-check=full --show-reachable=yes \
207: //TEST//       --log-file=astring.out.grind \
208: //TEST//       astring foo bar baz >astring.out 2>&1
209: //TEST// mkpspdf astring.ps astring.cpp* astring.out*
210: */
211:
```

```
    1: @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: starting astring.cpp
    2: astring.cpp:
    3:       $Id: astring.cpp,v 1.30 2015-01-26 14:28:34-08 - - $
    4: g++ -g -O0 -Wall -Wextra -rdynamic -std=gnu++11 astring.cpp -o astring -
lglut -lGLU -lGL -lX11 -lrt -lm
    5: rm -f astring.o
    6: @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: finished astring.cpp
```

```
 1: first=Hello, World!
 2: second=123lo, World!xy
 3: ,
 4:
 5: W
 6: o
 7: r
 8: l
 9: d
10: !
11: x
12: y
13: args: foo bar baz
14: args: foo bar baz
```

```
    1: ==6215== Memcheck, a memory error detector
    2: ==6215== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
    3: ==6215== Using Valgrind-3.9.0 and LibVEX; rerun with -h for copyright in
fo
    4: ==6215== Command: astring foo bar baz
    5: ==6215== Parent PID: 6213
    6: ==6215==
    7: ==6215==
    8: ==6215== HEAP SUMMARY:
    9: ==6215==     in use at exit: 0 bytes in 0 blocks
   10: ==6215==   total heap usage: 7 allocs, 7 frees, 113 bytes allocated
   11: ==6215==
   12: ==6215== All heap blocks were freed -- no leaks are possible
   13: ==6215==
   14: ==6215== For counts of detected and suppressed errors, rerun with: -v
   15: ==6215== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 6 from 6)
```