Keep the following overall goals of the resulting code in mind:

  - keep it minimal and simple, but complete enough to be practically useful
  - elegant and convenient to use public API
  - elegant, readable, and maintainable implementation
  - use modern C++ where appropriate (exceptions, templates, STL, etc)
  - support both 32 and 64 bit platforms

In other words, you should be able to look at your code 10 years
later and still feel proud ;-).

Do not use any external libraries (e.g, Boost, ACE, etc.), except
what comes with the standard C++ compiler (STL, iostream, etc) and
libc/POSIX. Provide an example and/or a test as well as a way to
build everything on a GNU/Linux machine with g++ (preferably
makefiles).

Project:

  Implement a buffer class that can be used to manage a binary
  data buffer. It should support copying into/from it, appending,
  resizing, and getting the underlying buffer as char*. The goal
  is to come up with a functional and clean interface. For example,
  here is how this buffer class can be used.

```
  const char* buf = ...
  size_t n = ...
  size_t c = ...

  buffer b; // Allocate an empty buffer.

  b.copy (buf, n); // Copy data into the buffer, growing the underlying
                   // buffer if necessary. After this call b.size () == n.

  buffer b1 (n); // Allocate a buffer 'n' bytes long.
  memcpy (b1.data (), buf, n);

  buffer b2 (0, 1024); // Allocate a buffer 0 bytes long with capacity
                       // to grow without reallocation to 1024 bytes.
  b2.size (n);
  memcpy (b2.data (), buf, n);


  buffer b3 (1024, 5 * 1024); // Allocate a buffer 1024 bytes long with
                              // capacity to grow without reallocation to
                              // 5*1024 bytes.
  memset (b3.data (), 0, 1024);

  for (int i = 0; i < 5; ++i)
    b3.append (buf, n); // Append the data to the buffer.

  if (b3.capacity () < 10 * 1024)
    b3.capacity (10 * 1024); // Make sure the buffer has this capacity.

  buffer b4 (buf, n); // Allocate a buffer 'n' bytes long and copy the
                      // data.

  buffer b5 (buf, n, 5 * 1024); // Allocate a buffer 'n' bytes long and
                                // with capacity to grow without
                                // reallocation to 5*1024 bytes. Copy
                                // the date over.

  buffer b6 (buf, n, n, true);  // Assume ownership of the buffer with
                                // size 'n' (second argument) and capacity
                                // 'n' (third argument).
```

  Extend this class (or create a new implementation) to support the
  "multiple underlying buffers" strategy instead of/in addition to
  "one contiguous underlying buffer" (the idea is to minimize

reallocation and copying).