```
1  // An implementation of a simplified STL Vector
2  // George F. Riley, Georgia Tech, Fall 2009
3
4  template<class T> class GFRVec
5  {
6  public:
7    GFRVec() : first(0), last(0), end(0) {}
8    GFRVec(size_t n)
9      { // Create a GFRVec with "n" copies of T, with default constructor
10       first = new T[n];
11       last = first + n;
12       end = last;
13     }
14   GFRVec(size_t n, const T& t)
15     { // Create a GFRVec with "n" copies of t
16       first = new T[n];
17       for (size_t i = 0; i < n; ++i) first[i] = t; // Populate the vector
18       last = first + n;
19       end = last;
20     }
21   GFRVec(const GFRVec& c)
22     { // Copy Constructor
23       first = new T[c.size()]; // Allocate memory
24       for (size_t i = 0; i < c.size(); ++i) first[i] = c[i]; // Copy elements
25       last = first + c.size();
26       end = last;
27     }
28   ~GFRVec()
29     { // Destructor, remove all elements
30       clear();
31     }
32   GFRVec& operator=(const GFRVec& rhs)
33     { // Assignment operator
34       if (this == &rhs) return *this; // Self assignment
35       delete [] first;                // Free old memory
36       first = new T[rhs.size()]; // Allocate memory
37       for (size_t i = 0; i < rhs.size(); ++i)
38         first[i] = rhs[i]; // Copy the elements
39       last = first + rhs.size();
40       end = last;
41     }
42   T& operator[](size_t i) const
43     { // Indexing operator
44       return first[i];
45     }
46   T& back() const
47     { // Return last element
48       return first[size()-1];
49     }
50   T& front() const
51     { // Return last element
52       return first[0];
53     }
54   void pop_back()
55     { // Remove last element
56       last--;
```

Program GFRVec.h

```
57        first[size()].~T(); // Call destructor on just popped object
58      }
59    void push_back(const T& t)
60      { // Add new element
61        if (last != end)
62          { // Room for new object without re-allocating
63            first[size()] = t;
64            last++;
65          }
66        else
67          { // Need to re-allocate
68            T* tmp = new T[end-first+1];
69            for (size_t i = 0; i < size(); ++i) tmp[i] = first[i];
70            tmp[size()] = t; // Add new element
71            last = tmp + (last - first) + 1;
72            end = last;
73            delete [] first;   // Delete and destroy old objects
74            first = tmp;
75          }
76      }
77    size_t size() const
78      { // Number of elements in the vector
79        return last - first;
80      }
81    void reserve(size_t n)
82      { // Reserve space for "n" elements
83        if (n <= (end-first)) return; // Less than already reserved
84        T* tmp = new T[n];            // Allocate new memory
85        for (size_t i = 0; i < size(); ++i) tmp[i] = first[i];
86        last = tmp + last - first;
87        delete [] first;
88        first = tmp;
89        end = first + n;
90      }
91    void clear()
92    { // Erase all elements
93      while(size()) pop_back();
94    }
95
96  private:
97    T* first; // Initial element
98    T* last;  // Last element
99    T* end;   // End of allocated storage
100 };
101
102
103
104
```

Program GFRVec.h (continued)

```
1  // An implementation of a simplified STL Vector
2  // This one uses uninitialized alloc and placement new operator
3  // George F. Riley, Georgia Tech, Fall 2009
4
5  template<class T> class GFRVec
6  {
7  public:
8    GFRVec() : first(0), last(0), end(0) {}
9    GFRVec(size_t n)
10     { // Create a GFRVec with "n" copies of T, with default constructor
11       first = (T*)malloc(n * sizeof(T));
12       for (size_t i = 0; i < n; ++i)
13         {
14           new (&first[i]) T();
15         }
16       last = first + n;
17       end = last;
18     }
19   GFRVec(size_t n, const T& t)
20     { // Create a GFRVec with "n" copies of t
21       first = (T*)malloc(n * sizeof(T));
22       for (size_t i = 0; i < n; ++i)
23         { // Use copy constructor to populate
24           new (&first[i]) T(t);
25         }
26       last = first + n;
27       end = last;
28     }
29   GFRVec(const GFRVec& c)
30     { // Copy Constructor
31       first = (T*)malloc(c.size() * sizeof(T));
32       for (size_t i = 0; i < c.size(); ++i)
33         {
34           new (&first[i]) T(c[i]); // Copy elements
35         }
36
37       last = first + c.size();
38       end = last;
39     }
40   ~GFRVec()
41     { // Destructor, remove all elements
42       clear();
43     }
44   GFRVec& operator=(const GFRVec& rhs)
45     { // Assignment operator
46       if (this == &rhs) return *this; // Self assignment
47       free(first);
48       first = (T*)malloc(rhs.size() * sizeof(T));
49       for (size_t i = 0; i < rhs.size(); ++i)
50         {
51           new (&first[i]) T(rhs[i]); // Copy the elements
52         }
53       last = first + rhs.size();
54       end = last;
55     }
56   T& operator[](size_t i) const
```

Program GFRVec1.h

```
57      { // Indexing operator
58        return first[i];
59      }
60    T& back() const
61      { // Return last element
62        return first[size()-1];
63      }
64    T& front() const
65      { // Return last element
66        return first[0];
67      }
68    void pop_back()
69      { // Remove last element
70        last--;
71        first[size()].~T(); // Call destructor on just popped object
72      }
73    void push_back(const T& t)
74      { // Add new element
75        if (last != end)
76          { // Room for new object without re-allocating
77            new (&first[size()]) T(t);
78            last++;
79          }
80        else
81          { // Need to re-allocate
82            T* tmp = (T*)malloc((size() + 1) * sizeof(T));
83            for (size_t i = 0; i < size(); ++i)
84              { // Copy old elements
85                new (&tmp[i]) T(first[i]);
86              }
87            new (&tmp[size()]) T(t);
88            for (size_t i = 0; i < size(); ++i)
89              { // Destroy old elements
90                first[i].~T();
91              }
92            last = tmp + (last - first) + 1;
93            end = last;
94            free(first);
95            first = tmp;
96          }
97      }
98    size_t size() const
99      { // Number of elements in the vector
100       return last - first;
101     }
102   void reserve(size_t n)
103     { // Reserve space for "n" elements
104       if (n <= (end-first)) return; // Less than already reserved
105       T* tmp = (T*)malloc(n * sizeof(T));
106       for (size_t i = 0; i < size(); ++i)
107         { // Copy elements to new space
108           new (&tmp[i]) T(first[i]);
109         }
110       last = tmp + last - first;
111       free(first);
112       first = tmp;
```

Program GFRVec1.h (continued)

```
      end = first + n;
    }
  void clear()
  { // Erase all elements
    while(size()) pop_back();
    free(first);
    first = 0;
    last = 0;
    end = 0;
  }

private:
  T* first; // Initial element
  T* last;  // Last element
  T* end;   // End of allocated storage
};
```

Program GFRVec1.h (continued)