Attachments for ECE2036 Final Exam, Fall 2012

```
1  // Code for ECE3090 midterm, Constructors and Destructors question
2
3  class A {
4  public:
5    A();          // Default constructor
6    A(int);       // int Constructor
7    A(const A&); // Copy constructor
8    ~A();         // Destructor
9    A operator+(A rhs) const; // Addition operator
10 public:
11   int x;        // Single data member
12 };
13
14 A A::operator+(A rhs) const
15 {
16   A r(x + rhs.x);
17   return r;
18 }
19
20 class B {
21 public:
22   B();          // Default Constructor
23   B(int);       // int Constructor
24   B(const B&); // Copy constructor
25   ~B();         // Destructor
26   B operator+(const B& rhs) const; // Addition operator
27 public:
28   int x;        // Single data member
29 };
30
31 B B::operator+(const B& rhs) const
32 {
33   return B(x + rhs.x);
34 }
35
36 void Sub1(const A& a)
37 {
38   A a2(a + a);
39 }
40
41 void Sub2(B b)
42 {
43   B b2(b + b);
44 }
45
46 int main()
47 {
48   A a(1);
49   B b(2);
50
51   Sub1(a);
52   Sub2(b);
53 }
```

Program Constructors-Destructors.cc

```
1  // Inheritance program for ECE2036 final exam
2  //ECE2036, Fall 2012
3  #include <iostream>
4  class Base {
5  public:
6    virtual void Func1() const = 0;
7             void Func2() const;
8    virtual void Func3() const;
9   };
10
11 class Sub1 : public Base {
12 public:
13   void Func1() const;
14   void Func2() const;
15 };
16
17
18 class Sub2 : public Base {
19 public:
20   void Func2() const;
21   void Func3() const;
22 };
23
24 class Sub3 : public Sub1 {
25 public:
26   void Func1() const;
27   void Func2() const;
28 };
29
30 class Sub4 : public Sub2 {
31 public:
32   void Func1() const;
33 };
```

Program inheritance.cc

```
34  // Implementations
35  // Base
36  void Base::Func2() const
37  {
38    std::cout << "Hello from Base::Func2()" << std::endl;
39  };
40  void Base::Func3() const
41  {
42    std::cout << "Hello from Base::Func3()" << std::endl;
43  };
44  // Sub1
45  void Sub1::Func1() const
46  {
47    std::cout << "Hello from Sub1::Func1()" << std::endl;
48  };
49  void Sub1::Func2() const
50  {
51    std::cout << "Hello from Sub1::Func2()" << std::endl;
52  };
53  // Sub2
54  void Sub2::Func2() const
55  {
56    std::cout << "Hello from Sub2::Func2()" << std::endl;
57  };
58  void Sub2::Func3() const
59  {
60    std::cout << "Hello from Sub2::Func3()" << std::endl;
61  };
62  // Sub3
63  void Sub3::Func1() const
64  {
65    std::cout << "Hello from Sub3::Func1()" << std::endl;
66  };
67  void Sub3::Func2() const
68  {
69    std::cout << "Hello from Sub3::Func2()" << std::endl;
70  };
71  // Sub4
72  void Sub4::Func1() const
73  {
74    std::cout << "Hello from Sub4::Func1()" << std::endl;
75  };
```

Program inheritance.cc (continued)

```
76  // Functions for testing
77  void BaseRef(const Base& b)
78  {
79    b.Func2();
80    b.Func1();
81  }
82
83  void Sub1Val(Sub1 s1)
84  {
85    s1.Func1();
86    s1.Func2();
87    s1.Func3();
88  }
89
90  void Sub2Ref(const Sub2& s2)
91  {
92    s2.Func1();
93    s2.Func2();
94    s2.Func3();
95  }
96
97
98  int main()
99  {
100   Sub1 s1;
101   Sub3 s3;
102   Sub4 s4;
103   BaseRef(s1);
104   BaseRef(s3);
105   Sub1Val(s3);
106   Sub2Ref(s4);
107 }
```

Program inheritance.cc (continued)

```
1  //
2  // ECE2036  Matrix and MatrixRow Class implementation
3  //
4  // All #include commands and namespace omitted for brevity.
5  //
6
7  typedef int Element_t;
8  typedef unsigned int Index_t;
9
10 class Matrix
11 { // Matrix class declaration
12 public:
13   //Constructors;
14   Matrix();             // Constructs an empty  0x0 matrix with NaM
15   Matrix(Index_t r, Index_t c); // Construct with specified rows, columns
16   Matrix(const std::string&);   // Construct from string "(i,j,k..),(a,b,c..)"
17   Matrix(const Matrix&);// Copy constructor
18   ~Matrix();            // Destructor
19   // Operators
20   Matrix& operator=(const Matrix&);
21   Element_t* operator[](int whichRow) const;
22   // Arithmetic operators
23   Matrix  operator+(Matrix );
24   // Other operators omitted for brevity
25 public:
26   Index_t nRows;
27   Index_t nCols;
28   Element_t* elements;  // Points to the elements
29   bool   NaM;   // True if "Not a matrix".
30 };
31
32 Matrix::Matrix()
33   : nRows(0), nCols(0), NaM(true)
34 { // Default constructor
35   // Nothing else needed
36 }
37
38 Matrix::Matrix(Index_t r, Index_t c)
39     : nRows(r), nCols(c), NaM(false)
40 { // Construct with specified rowCount and colCount
41   elements = new Element_t[nRows * nCols];
42   // Initialize to all zeros
43   for (Index_t i = 0; i < nRows * nCols; ++i) elements[i] = 0;
44 }
45
46 Matrix::Matrix(const std::string& st)
47 { // string constructor omitted for brevity. Assume it is correct
48 }
49
50 Matrix::Matrix(const Matrix& rhs)
51     : nRows(rhs.nRows), nCols(rhs.nCols), NaM(rhs.NaM)
52 { // Copy constructor
53   elements = rhs.elements;
54 }
```

Program matrix.cc

```
55
56  Matrix::~Matrix()
57  {
58  }
59
60  Matrix& Matrix::operator=(const Matrix& rhs)
61  { // Assignment operator
62    nRows = rhs.nRows;
63    nCols = rhs.nCols;
64    NaM = rhs.NaM;
65    // Allocate and copy elements
66    elements = new Element_t[nRows * nCols];
67    for (Index_t i = 0; i < nRows * nCols; ++i)
68      {
69        elements[i] = rhs.elements[i];
70      }
71    return *this;
72  }
73
74  // Indexing operator
75  Element_t* Matrix::operator[](int whichRow) const
76  {
77    return &elements[whichRow * nCols];
78  }
79
80
81  // Operators
82  Matrix Matrix::operator+(Matrix rhs)
83  {
84    Matrix ret(nRows, nCols); // Return matrix of correct shape
85    for (Index_t r = 0; r < nRows; ++r)
86      {
87        for (Index_t c = 0; c < nCols; ++c)
88          {
89            ret[r][c] = (*this)[r][c] + rhs[r][c];
90          }
91      }
92    return ret;
93  }
94
95  // Other operators omitted for brevity
```

Program matrix.cc (continued)

```
1  // Demonstrate use of Templates to make a linked list.
2  // ECE2036 - Fall 2012
3  // Includes omitted for brevity
4
5  template <typename T> class List;
6
7  // Define a templated class that contains the user's data
8  // and a "next" pointer.
9  template <typename T> class ListNode
10 {
11 public:
12   ListNode(const T& e) : next(0), element(e) {}
13 public:
14   ListNode<T>* next;
15   T            element;  // The user's data
16 };
17
18 template <typename T> class ListIterator
19 { // Define an "iterator" to access list elements
20 public:
21   ListIterator() : current(0) {}
22   ListIterator(ListNode<T>* b) :  current(b) {}
23
24   // Define not equal operator
25   bool operator !=(const ListIterator & rhs)
26     {
27       return current != rhs.current;
28     }
29
30   ListIterator operator++(int) // Postfix increment
31     { // Postfix increment
32       ListIterator tmp(*this);
33       current = current->next;
34       return tmp;
35     }
36
37   ListIterator operator++()    // Prefix increment
38     { // Prefix  increment
39       current = current->next;
40       return *this;
41     }
42
43   T& operator*() const // Dereference operator
44     {
45       return current->element;
46     }
47
48 private:
49   ListNode<T>* current;
50   friend class List<T>;
51 };
52 // Now define the "List" class
53 template <typename T> class List {
54 public:
```

Program templatelinkedlist.cc

```
55    List() : head(0), tail(0) {}
56    void PushBack(const T& e)
57    { // Add to end
58      ListNode<T>* n = new ListNode<T>(e); // Make a new node
59      if (!head) head = n;                 // If list is empty, n is new head
60      else tail->next = n;                 // Otherwise, old tail -> next is n
61      tail = n;                            // n is always new tail
62    }
63
64    void PushFront(const T& e)
65    { // Add to beginning
66      ListNode<T>* n = new ListNode<T>(e); // Make a new node
67      n->next = head;                      // New element next is old head
68      if (!tail) tail = n;                 // List was empty, n is new tail
69      head = n;                            // n is always new head
70    }
71
72    void Insert(const ListIterator<T>& i, const T& e)
73      { // Insert an element after specified iterator
74        ListNode<T>* n = new ListNode<T>(e);
75        n->next = i.current->next;
76        i.current->next = n;
77        if (i.current == tail) tail = n;
78      }
79
80    ListIterator<T> Begin()
81    { // Return an iterator starting at the first element
82      return ListIterator<T>(head);
83    }
84
85    ListIterator<T> End()
86    { // Return an iterator representing one beyond end
87      return ListIterator<T>(0);
88    }
89
90  private:
91    ListNode<T>* head;
92    ListNode<T>* tail;
93  };
```

Program templatelinkedlist.cc (continued)

```
1  // Define a template subroutine to compute the sort a collection of elements
2  // specified by two iterators
3  #include <algorithm>
4  using namespace std;
5
6  template <class T>
7  void Sort(T b, T e)
8  { // This sort is inefficient, and used for illustrative purposes only
9    while(b != e)
10     {
11       T i = b;
12       while(i != e)
13         {
14           if (*i < *b)
15             { // Need to swap.  This iter_swap is defined in "algorithm"
16               iter_swap(i, b); // Swap the two values
17             }
18           ++i;
19         }
20       ++b;
21     }
22 }
23
24 class A {
25 public:
26   A() : a(0) {};
27   A(int a0) : a(a0) {};
28   bool operator<(const A& rhs) { return a < rhs.a; } // Less-than operator
29 public:
30   int a;
31 };
32
33 class B
34 {
35 public:
36   B() : b(0) {};
37   B(int b0) : b(b0) {};
38   bool operator>(const B& rhs) { return b > rhs.b; } // Greater-than operator
39 public:
40   int b;
41 };
42
43 int main()
44 {
45   char s[] = "This is a test";
46   A a1[10] = {9, 8, 7, 6, 5, 4, 3, 2, 1, 0};
47   B b1[10] = {0, 1, 7, 6, 5, 4, 6, 7, 8, 9};
48
49   // Now call the Sort routine with differing parameters
50   Sort(s, s + 14);
51   Sort(&a1[0], &a1[10]);
52   Sort(&b1[0], &b1[10]);
53   Sort(&a1[10], &b1[10]);
54 }
```

Program sort.cc

Program sort.cc (continued)

```
1  // ECE2036 Final Exam
2  // Containers question
3  // (Includes and namespace omitted for brevity)
4  int main()
5  {
6    typedef map<char, int>      MyMap_t;
7    typedef MyMap_t::value_type MyMapPair_t;
8
9    vector<int> v1;
10   deque<int>  d1;
11   MyMap_t     m1;
12
13   // Populate the vector
14   for (int k = 0; k < 5; ++k)
15     {
16       v1.push_back(k);
17       v1.push_back(-k);
18     }
19   vector<int> v2(v1);  // Copy construtor
20   // VECTOR PRINT HERE
21   cout << "v1 size " << v1.size() << " v2 size " << v2.size()
22        << " v1 front() " << v1.front() << " v2.back() " << v2.back() << endl;
23   // END VECTOR PRINT
24
25   // Populate the deque
26   for (int k = 0; k < 5; ++k)
27     {
28       d1.push_back(k);
29       d1.push_front(-k);
30     }
31   // DEQUE PRINT HERE
32   cout << "d1.front() " << d1.front() << " d1.back() " << d1.back()
33        << " d1[5] " << d1[5] << endl;
34   // END DEQUE PRINT
35
36   // Populate the map
37   m1.insert(MyMapPair_t('B', 3));
38   m1.insert(MyMapPair_t('A', 4));
39   m1.insert(MyMapPair_t('D', 1));
40   m1.insert(MyMapPair_t('A', 1));
41   m1['K'] = 10;
42   // MAP PRINT HERE
43   cout << "m1.size() " << m1.size()
44        << " begin.first " << m1.begin()->first
45        << " m1.begin.second " << m1.begin()->second << endl
46        << " --(m1.end).first " << (--m1.end())->first
47        << " --(m1.end).second " << (--m1.end())->second << endl;
48   // END MAP PRINT
49 }
50
51
52
```

Program Containers.cc

```
1  // Static Members attachment
2  // ECE2036 Final Exam
3
4  #include <iostream>
5
6  using namespace std;
7
8  class MyClass
9  {
10 public:
11   MyClass() : a(0), b(0) { } // Constructor
12   // member variables
13   int a;
14   int b;
15   // member functions
16   void Func1();
17   void Func2();
18 };
19
20 // Member function implementations
21 void MyClass::Func1()
22 {
23   cout << "Hello from MyClass::Func1()" << endl;
24 }
25
26 void MyClass::Func2()
27 {
28   cout << "Sum of a and b is " << a + b << endl;
29 }
30
31 int main()
32 {
33
34 }
35
36
37
38
```

Program Static.cc