ECE3090 **Fall Semester, 2004**

Midterm Exam 2

Nov 9, 2004 *NAME:*_____

OPEN BOOK, OPEN NOTES, NO INTERNET ACCESS PLEASE!.

1. STL Vectors **20 Points**

   Consider the code snippet using the STL vector class below. For your answers in this question, assume an `int` is 4 bytes, and all pointers are 4 bytes.

   (a) What is printed at line 14 of the program? (Explain)

   (b) What is printed at line 15. **NOTE:** You don't have sufficient information to answer this definitively. Make an educated guess and explain your assumptions and how you arrived at the answer.

```
 1  #include <iostream>
 2  #include <vector>
 3
 4  using namespace std;
 5
 6  int main()
 7  {
 8    vector<int> v;
 9
10    for (int i = 0; i < 100; ++i)
11      {
12        v.push_back(i);
13      }
14    cout << "v.size() " << v.size() << endl;
15    cout << "sizeof(v) " << sizeof(v) << endl;
16  }
```

Program q1.cc

2. Template Subroutine **20 Points**

The code snippet below implements a Sum subroutine, that computes a sum elements specified by a pair of parameters specifying the first and (last + 1) elements to be summed. Note that this subroutine is *generic* in the sense that we implemented it for any arbitrary type T. The Sum routine simply dereferences each element between first and (last - 1), computes the sum and returns it.

At lines 41 through 44 we attempt to instantiate the Sum routine with four different parameter types. Which of the four instantiations of Sum will compile and which will not? State reasons why you think the call will compile properly or not. Use the back of this sheet for your answer if needed.

```
1  // Define a template subroutine to compute the sum of elements
2  // specified by two iterators
3
4  template <class T> int Sum (T start, T end)
5  { // Compute the sum of all elements specified by iterators "start" and "end"
6    int tot = 0;
7    while(start != end)
8      {
9        tot += *start++; // Add to total
10     }
11   return tot;
12 }
13
14 class A {
15 public:
16   A() : a(0) {};
17   A(int a0) : a(a0) {};
18 public:
19   int a;
20 };
21
22 class B
23 {
24 public:
25   B() : b(0) {};
26   B(int b0) : b(b0) {};
27   // Define a typecast operator to cast to an int
28   operator int() { return b;}
29 public:
30   int b;
31 };
32
33 int main()
34 {
35   char s[] = "This is a test";
36   A a1[10];
37   B b1[10];
38
39   // Now call the Sum routine with differing parameters
40   // Will all of these compile?
41   int r1 = Sum(s, s + strlen(s));
42   int r2 = Sum(&a1[0], &a1[10]);
43   int r3 = Sum(&b1[0], &b1[10]);
44   int r4 = Sum( b1[0],  b1[10]);
45 }
```

Program q2.cc

3. Object Cloning **20 Points**

What is printed by the program below? Explain your answer.

```
1  // Object cloning
2  #include <iostream>
3  using namespace std;
4
5  class Base {
6  public:
7    virtual void  Hello() { cout << "Hello from Base" << endl;}
8    virtual Base* Clone() { return new Base(*this);}
9  };
10
11 // A derives from Base
12 class A : public Base {
13 public:
14   virtual void  Hello() { cout << "Hello from A" << endl;}
15   virtual Base* Clone() { return new A(*this);}
16 };
17
18 // B derives from Base
19 class B : public Base {
20 public:
21   virtual void  Hello() { cout << "Hello from B" << endl;}
22 };
23
24 // C derives from B
25 class C : public B
26 {
27 public:
28   virtual void  Hello() { cout << "Hello from C" << endl;}
29   virtual Base* Clone() { return new C(*this);}
30 };
31
32 void Sub1(Base& b)
33 {
34   Base* newb = b.Clone();
35   newb->Hello();
36 }
37
38 int main()
39 {
40   Base base;
41   A a;
42   B b;
43   C c;
44   Sub1(base);
45   Sub1(a);
46   Sub1(b);
47   Sub1(c);
48 }
```

Program q3.cc

4. Iterators **20 Points**

Code snippets from the ListIterator class we designed and discussed in class are given below. In the main program at lines 34 and 36, there are two different (but similar) loops to iterate over the entire list. Which is more efficient? Explain your answer

```
1  template <class T> class ListIterator
2  { // Define an "iterator" to access list elements
3  public:
4    // Lots of code omitted here for brevity
5    ListIterator operator++(int) // Postfix increment
6      { // Postfix increment
7        // Create a temporary to return the value prior to advance
8        ListIterator tmp(*this);
9        if  (current) current = current->next;
10       return tmp;
11     }
12
13   ListIterator operator++()    // Prefix increment
14     { // Prefix  increment
15       if  (current) current = current->next;
16       return *this;
17     }
18   // Lots of code omitted here for brevity
19 };
20
21 template <class T> class List {
22 public:
23   // Lots of code omitted here for brevity
24   ListIterator<T> Begin()
25       // implementation omitted
26   ListIterator<T> End()
27       // implementation omitted
28 };
29
30 int main()
31 {
32   List list;
33   // Code omitted that add things to the List list.
34   for (ListIterator k = list.Begin(); k != list.End(); ++k)
35     // omitted
36   for (ListIterator n = list.Begin(); n != list.End(); n++)
37     // omitted
38 }
```

Program q4.cc

5. Discrete Fourier Transform **20 Points**

Suppose that we implemented Discrete Fourier Transform (DFT) using a naive approach, simply using the basic definition for the DFT given below:

$$H[n] = \sum_{k=0}^{N-1} W^{nk} h[k] \qquad \text{where } W = e^{-j2\pi/N} = \cos(2\pi/N) - j\sin(2\pi/N) \qquad \text{where } j = \sqrt{-1}$$

We executed this implementation on a sample set consisting of 65,536 elements, and observed an execution time of 30 minutes. Since this execution time seems too long, we decided to implement a more efficient version of the algorithm, using the Danielson–Lanczos binary decomposition method and the Cooley–Tukey bit–reversal trick that we used for our lab 4.

Estimate the running time for the new, efficient algorithm, when running on the same sample set of 65,536 elements. Assume that for both implementations we pre–computed the W array (as we did in our implementation). Also, you can ignore any one–time overhead such as the time needed to read in the sample set from disk, to print out results, and to pre–compute the W values. Explain your answer, state any assumptions you made, and show your work. Clearly, there is no exact "right answer". Your answer must be reasonable, your assumptions must be valid, and your answer must be consistent with your assumptions.