RMIT University
School of Computer Science and Information Technology
COSC1252/1254 Object Oriented Programming
Semester 2 2008

Date: Thu 6 Nov
Time: 1:45pm to 4:00pm
Duration: 2:15 hours
Number of Pages: 4

Instructions to Candidates

This examination accounts for 40% of the total marks for the subject.
This examination totals 100 marks.
Answer all questions.
Marks for each question are shown.
This examination is closed book.
Calculators and other electronic devices are not permitted.

Question 1

Give a concise answer to each of the following questions:

(a) Give a command line to compile with `g++` a C++ program called `q1.cpp` with the executable being placed in a file called `q1` and with debugging support for `gdb` switched on.

(b) What is the purpose of having `#ifndef _GLIBCXX_VECTOR` in the `<vector>` header file?

(c) When does the `terminate()` function get called?

(d) What language is regarded as the first to support object oriented programming?

(e) What is the current C++ standard?

(f) Name a new feature to be supported in C++0x.

(g) What header file needs to be included in a C++ program order to use `for_each`?

(h) Is the declaration `std::vector<std::vector<int>> a;` allowed?

(i) Name three types of iterators.

(j) Can both the prefix and postfix increment operators be used on an iterator?

(k) In a C++ program how many objects can be declared of a class which has only static members?

(l) How is a class made abstract in C++?

(m) How are dynamically bound function calls made in C++?

(n) What is the difference between using `new` and `malloc` to allocate objects of a class?

(o) At which companies was early work on the STL performed?

(p) What is the value of the expression `cout << "hello world" << endl;`

(q) Do STL containers support range checked access using the `[]` operator?

(r) What are the two kinds of containers defined by the STL?

(s) Why might `f(a) = 3;` where `f(a)` is a function call be a valid assignment statement in C++?

(t) Can a derived class directly access `private` data members of its base class?

(1 mark each = 20 marks)

Question 2

Given the following definition of `Point`

```
class Point
{
  float _x;
  float _y;
public:
  Point(float x, float y) : _x(x), _y(y) {}
  float x() { return _x; }
  float y() { return _y; }
};
```

(a) What would be the difference if `Point` was declared a `struct` rather than a `Class`?

(b) Modify the constructor so it may be used as a default constructor.

(c) What is the effect of writing the `Point` constructor as

```
  Point(float x, float y) : _y(y), _x(x) {}
```

(d) Explain why the assignment statement below is not possible. Give changes to `Point` to make it possible.

```
  Point p1(4.2, 5.1);
  p1.x() += 0.1;
```

(e) Give and explain the changes necessary to `class Point` to support constant `Point`s, for example

```
const Point origin(0.0,0.0);
cout << origin.x() << " " << origin.y() << endl;
```

(f) Give a function which overloads the `<<` output operator so that the output statement in following code will work:

```
  Point p1(3.2,5.0), p2(2.0,4.5);
  cout << p1 << " " << p2 << endl;
```

(g) Modify `class Point` to be a template class, i.e. `template <class T> class Point` where the template parameter `T` is used to specify the data type used to store the (x, y) coordinates.

(h) If a two element array is instead used to represent the coordinates as below, give the changes necessary to the constructor `Point(float x, float y)` and the member function `float x()`.

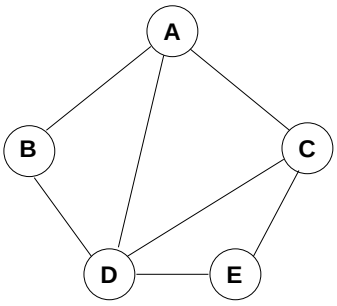
```
class Point {
  float coord[2];
  ...
}
```

(i) Give a definition for `class ColoredPoint` which is derived from `Point`, and adds red, green and blue data members as `float`s.

(1+2+2+3+4+3+4+3+3 = 25 marks)

Question 3

An undirected graph consisting of vertices and edges is shown below



In a particular application a `Vertex` class is using a dynamically allocated array to store the list of adjacent vertices

```
class Vertex
{
  int *adj;
  int size;
 public
  Vertex(int n);
  ...
}
```

(a) Give the definition of a constructor `Vertex::Vertex(int n)` which allocates an array of `int`s of size `n` storing the array pointer in the `adj` data member and the size in the `size` data member.

(b) Give the definition of a constructor `Vertex::Vertex(int n, int a[])` which does the same as the constructor in (a) but which also initialises the adjacency list using the contents of `a`.

(c) What problems occur when one vertex object is assigned to another as below? Explain your answer.

```
v2 = v1;
```

(d) Write a member function which overloads the assignment operator and fixes the problems when assigning one `Vertex` object to another, explaining how it does so.

(e) Provide a copy constructor for the `Vertex` class.

(f) Provide a member function which overloads the array subscript operator `[]` to give access to the adjacent vertices.

(g) Give and explain a solution to the problem which occurs in the following function.

```
void f()
{
  Vertex v(10);
  ...
}
```

(h) Show how using static members along with other changes to the `Vertex` class a count of the total number of array elements allocated for adjacent vertices for all vertex objects in existence can be implemented.

(2+3+3+4+4+3+4+4 = 27 marks)

Question 4

(a) Give and discuss operations supported by the `vector` and `list` containers, including their computational complexity.

(b) Give a definition for a class `Graph` with the following members: (i) the number of vertices in the graph (ii) a dynamically allocated array for the vertices which have `float` (x, y) coordinates and an `int visited` flag and (iii) a dynamically allocated array of STL `list`s of `int`s for the adjacency lists.

(c) Provide a constructor `Graph::Graph(int nv)` where nv is the number of vertices in the graph.

(d) Write member function `Graph::add_edge(int u, int v)` which adds edge (u, v) to a graph object.

(e) Give and explain changes needed to your `add_edge` if the `Graph` class uses an array of `vector`s instead of an array of `list`s.

(f) Explain how to make the `Graph` class a singleton, including code as appropriate.

(g) The Boost Graph Library (BGL) has a template `adjacency_list` class where each template argument has a default value:

```
adjacency_list<OutEdgeList, VertexList, Directed,
               VertexProperties, EdgeProperties,
               GraphProperties, EdgeList>
```

Give a `typedef` for a `Graph` using the `adjacency_list` class which uses a `vector` for the vertices, a `list` for the edges and selects an undirected graph.

(h) Part of a simple graph example program is given below

```
int main(int argc, char** argv)
{
   typedef boost::adjacency_list<> Graph;

   enum { A, B, C, D, E, N };
   const int num_vertices = N;
   const char *name = "ABCDE";

   typedef std::pair<int, int> Edge;
   Edge edge_array[] = { Edge(A, B), Edge(A, D), Edge(C, A), Edge(D, C),
      Edge(C, E), Edge(B, D), Edge(D, E) };

   const int num_edges = sizeof(edge_array) / sizeof(edge_array[0]);
   Graph g(num_vertices);

   for(int i = 0; i < num_edges; i++)
      boost::add_edge(edge_array[i].first, edge_array[i].second, g);
   ...
```

The BGL has a breadth first search algorithm implementation

```
// named parameter version
template <class Graph, class P, class T, class R>
void breadth_first_search(Graph& G,
  typename graph_traits<Graph>::vertex_descriptor s,
  const bgl_named_params<P, T, R>& params);
```

Give and explain code which uses `breadth_first_search` to search the graph in the example program above, recording predecessors/parents as it proceeds. Include in your explanation a discussion of *visitors*.

(5+3+3+3+2+3+2+7 = 28 marks)

THE END