Instructions:
1. Print your name in the space provided
2. Print your student identifier in the space provided.
3. Print the class section number in the space provided.
4. Print the date in the space provided.
5. You have 55 minutes to complete this examination.
6. You may use a calculator.
7. Reference materials are prohibited. You must complete this test without the aid of course notes or reference texts.
8. Questions requiring written answers must be answered using standard American English. Answers containing spelling or grammatical errors will be given no credit.
9. Answers must be coded legibly. Answers that cannot be read by your instructor will be given no credit.
10. You must do your own work.

Scoring:
1. The table to the right lists the number of raw points available for each problem and the total number of raw points that can be earned on this test.
2. Your score will be normalized to a fraction of 150 points. If $n$ is your normalized score, $r$ is your raw score, and $T$ is the total number of raw points, then $n = \frac{150 \cdot r}{T}$.
1. (65 points) Complete function main and code member functions of class stack to print the contents of stack s. Function main and class stack are given in Figure 1. What does function main print?

<table>
<thead>
<tr>
<th>Task</th>
<th>Available</th>
<th>Missed</th>
<th>Earned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete function main</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete constructor stack</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete destructor ~stack</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete member function push</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete member function pop</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete member function empty</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete member function full</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>What does function main print?</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>65</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```cpp
#include <iostream.h>

struct stackerror {
    stackerror(char* m);
};

stackerror::stackerror(char* m) {
    cerr << "\nI am the stack and I am " << m << ".";
}

class stack {
    struct element {
        char v;
        element* antecedent;
    };
    element* top;

public:
    stack();
    ~stack();
    void push(char v);
    char pop(void);
    int empty(void);
    int full(void);
};

int main()
{ 
    stack s;
    char *evian="evian";
    while (*evian) s.push(*evian++);
    return 0;
}
```

Figure 1. Function main and class stack.
```cpp
stack::stack(): top(0) {}  
stack::~stack()  
{  
element* e=top;  
   while (e) {  
      element* p=e;  
      e=e->antecedent;  
      delete p;  
   }  
}

void stack::push(char v)  
{  
   if (full()) throw stackerror("full");  
   element* e=new element;  
   e->v=v;  
   e->antecedent=top;  
   top=e;  
}

char stack::pop(void)  
{  
   if (empty()) throw stackerror("empty");  
   element* e=top;  
   char v=e->v;  
   top=e->antecedent;  
   delete e;  
   return v;  
}

int stack::empty(void)  
{  
   return top==0;  
}

int stack::full(void)  
{  
   return 0;  
}

int main()  
{  
   stack s;  
   char *evian="evian";  
   while (*evian) s.push(*evian++);  
   while (!s.empty()) cout << s.pop();  
   return 0;  
}
```

Function main prints:

```
naïve
```
2. (105 points) Code class circle and struct circleerror so that the output given in Figure 2 is produced when function main is executed. You must code class circle member functions area and circumference. Function main is shown in Figure 3.

<table>
<thead>
<tr>
<th>Task</th>
<th>Available</th>
<th>Missed</th>
<th>Earned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Includes</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>struct circleerror</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constructor circleerror</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>class circle syntax</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>class circle member data</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>class circle member function prototypes</td>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>class circle member function implementations</td>
<td>55</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>105</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

circle: c, circumference=31.4159, area=78.5398
I’m too fat for a size –5 dress.

**Figure 2. Output for question 2.**

```cpp
int main()
{
    circle c(5);
    c.print(cout, "c");
    circle d(-5);
    c.print(cout, "d");
}
```

**Figure 3. Function main**
#include <iostream.h>
#include <fstream.h>

struct circleerror {
    circleerror(double r);
};

circleerror::circleerror(double r)
{ cerr << "\nI'm too fat for a size " << r << " dress.";
}

class circle {
    const double pi;
    double radius;

public:
    circle(double r);
    ~circle();
    double circumference(void);
    double area(void);
    void print(ostream& o, char* id);
};
circle::circle(double r): pi(3.141592654)
{ if (r<=0.0) throw circleerror(r);
    radius=r;
}
circle::~circle() {}

circle::circumference(void)
{ return 2*pi*radius;
}

circle::area(void)
{ return pi*radius*radius;
}

void circle::print(ostream& o, char* id)
{ o << "\n";
    o << "circle: " << id;
    o << ", ";
    o << "circumference=" << circumference();
    o << ", ";
    o << "area=" << area();
}
3. (30 points) Code member function `print` and the destructor in `class list`. Member function `print` produces the output shown in figure 4 given the list shown in figure 6. The `class list` destructor, function `~list`, removes every item in the list and associated identifiers. `Class list` is defined in figure 5.

<table>
<thead>
<tr>
<th>Task</th>
<th>Available</th>
<th>Missed</th>
<th>Earned</th>
</tr>
</thead>
<tbody>
<tr>
<td>print</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>~list</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>30</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
void print(ostream& o);
```

**Figure 4.** Output produced by member function `print`.

```
class list {
  struct item {
    item* next;
    char* id;
  };
  item* head;
  item* tail;
public:
  list();
  ~list();
  void append(char* id);
  void print(ostream& o);
};
```

**Figure 5. Class list**
void list::print(ostream& o) {
    item* e=head;
    o << "\n";
    while (e) {
        o << e->id;
        e=e->next;
    }
}

list::~list() {
    item* e=head;
    while (e) {
        item* p=e;
        e=e->next;
        if (p->id) delete[] p->id;
        delete p;
    }
}
Questions 4, 5, 6, 7 and 8 relate to simulation events recorded below. Assume the line is empty at time zero. The simulation is updated every 5 seconds.

a. Adele arrived at 7 seconds.
b. Bonita arrived at 14 seconds.
c. Cosette arrived at 21 seconds.
d. Adele departed at 28 seconds.
e. The simulation recorded that Denise arrived at 35 seconds.
f. Elise arrived 42 seconds
g. Bonita departed at 49 seconds
h. Cosette departed at 56 seconds.
i. Fantine arrived at 63 seconds.
j. The simulation recorded that Grace arrived at 70 seconds.
k. The simulation ended at 77 seconds.

4. (40 points) Assume the simulation is implemented using a queue having five (5) elements. Elements are stored in a dynamically allocated array. Complete the diagram of the queue in Figure 7 at the end of the simulation. Class queue is given in Figure 8.

<table>
<thead>
<tr>
<th>Task</th>
<th>Available</th>
<th>Missed</th>
<th>Earned</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue members</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>elements on the queue</td>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>40</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
</tr>
<tr>
<td>int</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

Figure 7. Queue at the end of the simulation.
class queue {
    int oldest;     //Index of oldest element on the queue
    int newest;     //Index of newest element on the queue
    int* q;         //Points to storage for elements
    int size;       //Maximum number of elements on the queue
    int count;      //Current number of elements on the queue

public:
    queue(int sz=100);  //Constructor
    ~queue();          //Destructor
    void enq(int v);   //Insert an element on the newest end of
                       //the queue
    void deq();        //Remove the oldest element on the queue
    int empty();       //Is the queue empty?
    int full();        //Is the queue full?
};

Figure 8. Class queue.

5. (5) How many customers were served?

3 customers were served

6. (15) Who waited the longest and how long did she wait?

Bonita and Cosette waited longest. They both waited 35 seconds.

7. (10) What was the average wait time?

30 seconds