Instructions:
1. Print your name in the space provided
2. Print your student identifier in the space provided.
3. Print the section number of the section in which you are enrolled 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 $n = 150 \frac{r}{T}$.
1. (50 points) Code class list. Do not code implementations for member functions.
   1.1. Class list is composed of dynamically allocated elements.
   1.2. Each element has pointers to elements containing smaller and larger keys.
   1.3. The list is circular.
   1.4. The list contains a sentinel.
   1.5. The sentinel contains the smallest key.
   1.6. Keys are pointers to strings.
   1.7. Separate storage is dynamically allocated for keys.
   1.8. Access to the list is via member smallest that points to the sentinel.
   1.9. Code function prototypes for the
       1.9.1. constructor
       1.9.2. destructor
       1.9.3. member function insert that inserts a key in its proper position on the list
       1.9.4. member function remove that deletes the element whose key is given as a parameter.

```cpp
class list {
    struct element {
        element* smaller;
        char* key;
        element* larger;
    };
    element* smallest;
public:
    list();
    ~list();
    void insert(char* key);
    void remove(char* key);
};
```

**Figure 1.** Answer to problem 1.

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private struct element</td>
<td>10</td>
</tr>
<tr>
<td>Character keys</td>
<td>5</td>
</tr>
<tr>
<td>Member smallest</td>
<td>5</td>
</tr>
<tr>
<td>constructor</td>
<td>5</td>
</tr>
<tr>
<td>destructor</td>
<td>5</td>
</tr>
<tr>
<td>insert</td>
<td>10</td>
</tr>
<tr>
<td>remove</td>
<td>10</td>
</tr>
<tr>
<td>Total</td>
<td>50</td>
</tr>
</tbody>
</table>
2. (30 points) Complete the diagram in Figure 2. The diagram represents data structures created as a result of calling the constructor `list()` as defined in problem 1. The constructor creates an empty list. An empty list consists of the sentinel-element. Define a string, `MsMin` that will serve as a key smaller than any that might reasonably be inserted on the list. Use only the portion of the diagram needed. Label the type of all structures. Include member names and types. Use directed edges to show the values of pointers. Write the value of arithmetic types in the space provided.

Figure 2. Diagram for Problem 2

Figure 3. Answer for Problem 3

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fifteen fields each worth 2 points</td>
<td>30</td>
</tr>
<tr>
<td>Total</td>
<td>30</td>
</tr>
</tbody>
</table>
3. (40 points) Write member function `int list::count(void)` that returns the number of elements on the list. Class list is defined in problem 1. The sentinel is not counted as an element of the list.

```cpp
int list::count(void)
{
    element* e=smallest->smaller;
    int c=0;
    while (strcmp(e->key,MsMin)<0) {
        e=e->smaller;
        c++;
    }
    return c;
}
```

**Figure 4.** Answer for problem 3.

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>5</td>
</tr>
<tr>
<td>Start traversing the list with smallest-&gt;smaller</td>
<td>10</td>
</tr>
<tr>
<td>string comparison</td>
<td>10</td>
</tr>
<tr>
<td>Traverse using the smaller link</td>
<td>10</td>
</tr>
<tr>
<td>return c</td>
<td>5</td>
</tr>
<tr>
<td>Total</td>
<td>40</td>
</tr>
</tbody>
</table>
4. (70 points) Write function `void list::insert(char* key)` that inserts an element into a list. Keys are placed on the list in descending order. Class `list` is defined in problem 1.

```cpp
char* list::makekey(char* key)
{
    char* KEY = new char[strlen(key)+1];
    return strcpy(KEY, key);
}

void list::insert(char* key)
{
    element* e = smallest->smaller;
    while (strcmp(key, e->key) < 0) e = e->smaller;
    if (strcmp(key, e->key) == 0) return;
    element* n = new element;
    n->key = makekey(key);
    n->smaller = e;
    n->larger = e->larger;
    e->larger = n;
    e->larger = n;
}
```

Figure 5. Answer to problem 4.

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>5</td>
</tr>
<tr>
<td>makekey or equivalent</td>
<td>15</td>
</tr>
<tr>
<td>start traversing the list with smallest-&gt;smaller</td>
<td>10</td>
</tr>
<tr>
<td>string comparisons</td>
<td>10</td>
</tr>
<tr>
<td>The next element is found via the smaller link</td>
<td>10</td>
</tr>
<tr>
<td>Bind the element to the list</td>
<td>10</td>
</tr>
<tr>
<td>Bind the list to the element</td>
<td>10</td>
</tr>
<tr>
<td>Total</td>
<td>70</td>
</tr>
</tbody>
</table>
6. (161 points) Complete the diagrams in Figures 6 to show how an element on the list is inserted. The list is defined by class list in problem 1. The list consists of elements that reference keys "dee", "sue", and "zoe". The sentinel key is the empty string. The key to be inserted is "ilse". Use the diagram on the left in Figure 6 to show the list before the element is inserted. Complete the diagram on the right in Figure 6 to show the list after the element is inserted. Diagram local variables e and n. Variable e points to the element on the list where the new element referencing "ilse" is inserted. Variable n points to the new element. Use only those portions of the diagrams that are required. Label structure types, member names, member types, and member values. Use directed edges to indicate the values of pointers.

Figure 7. Diagram for problem 6.
(36 points) Insert the identifiers given below in a binary tree. Draw a diagram of the tree above the title Figure 4.

nadine tammy gloria qian daphne wendy julia yvette vera stacia pia linda hallie fantine bonita alice cosette elise

Figure 4. Binary tree for problem 7

7. (36 points) List the identifiers as they are printed in a pre-order traversal opposite the numbers given.

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>nadine</td>
<td>6</td>
<td>cosette</td>
<td>11</td>
<td>linda</td>
<td>16</td>
<td>wendy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>gloria</td>
<td>7</td>
<td>fantine</td>
<td>12</td>
<td>tammy</td>
<td>17</td>
<td>vera</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>daphne</td>
<td>8</td>
<td>elise</td>
<td>13</td>
<td>qian</td>
<td>18</td>
<td>yvette</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>bonita</td>
<td>9</td>
<td>julia</td>
<td>14</td>
<td>pia</td>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>alice</td>
<td>10</td>
<td>hallie</td>
<td>15</td>
<td>stacia</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
8. (5 points) What is the height of the tree produced in problem 7?

4

9. (5 points) What is the depth of the node referencing the identifier pia in the tree of problem 7?

3

10. (5 points) What is the maximum number of comparisons required finding an identifier in the tree of problem 8?

5

11. (5 points) How many times is a recursive implementation of a function which performs an in order traversal of the tree produced in problem 8 called?

37

12. (20 points) Derive the number of nodes in a quaternary tree of height $h$. Each node in the tree has four children.

\[ N = \sum_{i=0}^{h} 4^i \]

\[ N = 1 + 4^1 + 4^2 + \cdots + 4^h \]

\[ 4N = 4 + 4^2 + \cdots + 4^h + 4^{h+1} \]

\[ 3N = -1 + 4^{h+1} \]

\[ N = \frac{4^{h+1} - 1}{3} \]