Before you start, write your name on the top of this page. No calculators or course notes are allowed. Use the back of any page for scratch work.

Question 1: _____ out of _____ points

Question 2: _____ out of _____ points

Question 3: _____ out of _____ points

Question 4: _____ out of _____ points

Total Score:

Grade:
**Question 1: Some Warm-Up Problems**

Tell whether each of the following statements is true or false by checking the appropriate box. If you do not know the answer to a question, just give it your best guess.

<table>
<thead>
<tr>
<th>Statement</th>
<th>True</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>a) In C++, friend classes of a class A can access private members of A.</td>
<td>[X]</td>
<td>[ ]</td>
</tr>
<tr>
<td>b) If a C++ class has no default constructor, no array of objects of that class can be instantiated.</td>
<td>[X]</td>
<td>[ ]</td>
</tr>
<tr>
<td>c) The type <code>vector&lt;vector&lt;int&gt;&gt;</code> can be used to implement a two-dimensional array (or matrix) of integers.</td>
<td>[X]</td>
<td>[ ]</td>
</tr>
<tr>
<td>d) Destructors are only necessary in a class that uses arrays.</td>
<td>[ ]</td>
<td>[X]</td>
</tr>
<tr>
<td>e) A class can have more than one subclass.</td>
<td>[X]</td>
<td>[ ]</td>
</tr>
<tr>
<td>f) A class can have more than one destructor.</td>
<td>[ ]</td>
<td>[X]</td>
</tr>
<tr>
<td>g) In a C++ program, every “new” operation should have a corresponding “delete”.</td>
<td>[X]</td>
<td>[ ]</td>
</tr>
<tr>
<td>h) When deleting arrays that were allocated using “new”, the “delete []” command should be used.</td>
<td>[X]</td>
<td>[ ]</td>
</tr>
<tr>
<td>i) A C++ vector has to contain at least one element.</td>
<td>[ ]</td>
<td>[X]</td>
</tr>
<tr>
<td>j) “Overloading” means providing different functions that share the same name but have different signatures.</td>
<td>[X]</td>
<td>[ ]</td>
</tr>
</tbody>
</table>
Question 2: Solving the Mystery of the Riddle Class

Yet again, some old program has been discovered, and by a strange coincidence, it is yet again written in C++. It implements some mysterious Riddle class, and all that is known about it is the fact that it is bug-free and works on a quite famous type of data structure. The program creates and works on this data structure in a slightly different way than you have seen it before. This makes the program very short, but a bit more cryptic. So do not worry if you cannot figure out why, for example, the pointer-to-pointer type is used. Just try to find out which algorithm is programmed here. You probably know that algorithm. Then find more appropriate names for the Riddle class and its member functions and describe what each of them does. Also, find out what the output of the program is. In order to get bonus points, explain why Function2 returns a pointer to a pointer.

(a) Class Riddle should be named ______BinarySearchTree_____________________________ .

It implements the following thing (just give the name of the data structure/algorithm):

A binary search tree that can be instantiated for any given type of elements that are ordered, i.e., the <, >, ==, and != operations are defined on them.

(b) Member function Function1() should be named ____DeleteTree()____________________ .

It does the following thing:

It deletes the current binary search tree, i.e., frees all memory allocated by the current object.

(c) Member function Function2() should be named ____FindValue()___________________ .

It does the following thing:

It takes as its input a value “val” of the tree element type. Starting at the root, it moves to a node’s left child if val is less than the node’s label and to its right child if val is greater than its label. This process continues until either a null pointer is reached (i.e., the child does not exist) or val equals the node’s label. The function then returns a pointer to the final pointer, which is either null or points to the location where val was found.
(d) Member function `Function3()` should be named `__InsertValue()____________________`.

It does the following thing:

It takes an input “val” of the tree element type and calls the function `FindValue()` with it. If `FindValue()` does not find `val` in the tree, a new node with the label `val` is created at the appropriate position in the tree (pointed at by the return value of `FindValue()`). This way, `val` is inserted into the binary search tree. If `val` already exists in the tree, the function does not change anything.

(e) Member function `Function4()` should be named `___ContainsValue()__________________`.

It does the following thing:

It calls `FindValue()` to check whether its input `val` is contained in the tree. If so, it returns true and otherwise false.

(f) The output of the program is:

```
_1_ (or true)________________________
_0_ (or false)_______________________
```

(g) **Bonus question**: Why does `Function2()` return a pointer to a pointer? Explain (or speculate) why this data type was chosen.

`Function2()` locates the pointer that we need to modify if we want to insert a new node into the tree. If it only returned the pointer itself, i.e., the memory address it points to, we would not be able to modify that pointer, i.e., let it point to a newly allocated tree element that contains the new label. Instead, we need a pointer to that pointer, i.e., know the pointer’s memory location, to do that.
```cpp
#include <iostream>
#include <string>
using namespace std;

template <class TYPE>
struct Element
{
    TYPE value;
    Element *left;
    Element *right;
};

template <class TYPE>
class Riddle
{
public:
    Riddle() { root = NULL; }
    ~Riddle() { Function1(root); }
    void Function3(TYPE val);
    bool Function4(TYPE val);
private:
    Element<TYPE> *root;
    void Function1(Element<TYPE> *elem);
    Element<TYPE> **Function2(TYPE val);
};

template <class TYPE>
void Riddle<TYPE>::Function1(Element<TYPE> *elem)
{
    if (elem != NULL)
    {
        Function1(elem->left);
        Function1(elem->right);
        delete elem;
    }
}

template <class TYPE>
Element<TYPE> **Riddle<TYPE>::Function2(TYPE val)
{
    Element<TYPE> **elem = &root;
    while (*elem != NULL && (*elem)->value != val)
    {
        if (val < (*elem)->value)
            elem = &((*elem)->left);
        else
            elem = &((*elem)->right);
    }
    return elem;
}

(continued on next page)
```
template <class TYPE>
void Riddle<TYPE>::Function3(TYPE val)
{
    Element<TYPE> **elem = Function2(val);
    if (*elem == NULL)
    {
        *elem = new Element<TYPE>;
        (*elem)->value = val;
        (*elem)->left = (*elem)->right = NULL;
    }
}

template <class TYPE>
bool Riddle<TYPE>::Function4(TYPE val)
{
    return (*Function2(val) != NULL);
}

int main()
{
    Riddle<string> r;
    r.Function3("have");
    r.Function3("you");
    r.Function3("figured");
    r.Function3("this");
    r.Function3("program");
    r.Function3("out");
    r.Function3("yet?");
    cout << r.Function4("this") << endl << r.Function4("that") << endl;
    return 0;
}
Question 3: Prime Time

On the following page you will find a C++ program that is supposed to output the first 100 prime numbers. It defines a class PrimeNumber that is initialized with an integer n. The data member “number” will be set to the smallest prime number greater or equal n. The “++” operator is overloaded to set “number” to the next greater prime number following the current one.

Unfortunately, the program contains a lot of bugs. Sometimes there are some dots ( . . . ), indicating that you have to fill in some code there. But you may have to fill in code in other places, too.

Good luck!
```cpp
#include <iostream>
using namespace std;

class PrimeNumber
{
public:
    int PrimeNumber(int n): number(n - 1) { (*this)++; }
    bool IsMultipleOf(int n) { return (number%n == 0); }
    PrimeNumber operator++() { do number++; while (!IsPrime()); return *this; }
    operator int() { return number; }
private:
    bool IsPrime();
    int number;
};

void PrimeNumber::IsPrime()
{
    for (int i = 1; i <= number/2; i++)
        if (IsMultipleOf(i))
            return false;
    return true;
}

int main()
{
    int n;
    PrimeNumber p(2);
    for (int i = 1; i <= 100; i++)
    {
        n = p;
        cout << n << " ";
        p++;
    }
    cout << endl;
    return 0;
}
```
Question 4 (Bonus Question): Sentiment Analysis

(a) Describe the difference between a lexicon based approach vs machine learning approach for doing sentiment analysis

A lexicon based approach uses a dictionary of positive/negative words, whereas a machine learning approach creates a model based on a dataset with examples of positive/negative classes.

(b) What is the advantage of using an n-gram model compared to a bag of words model?

An n-gram model takes more word context into account.