Machine Problem 1 - Stacks

[1] A stack can be used to recognize certain types of patterns. Consider the pattern STRING1#STRING2, where neither string contains "#". The STRING2 must be the reverse of STRING1. Write a program that reads strings until the user enters an empty string. The program should indicate whether each string matches the pattern.

Run:

```
Input a string: 1&A#A&1
1&A#A&1 matches the pattern

Input a string: 1&A#1&A
1&A#1&A does not match the pattern

Input a string: madamimadam#madamimadam
madamimadam#madamimadam matches the pattern

Input a string:
```

[2] Write a program that prompts the user to enter a non-negative decimal number and a base in the range 2 <= base <= 16. Write a function multibaseOutput() that displays the number in the specified base. The program terminates when the user enters a number of 0 and a base 0.

```
Run:
Enter a non-negative decimal number and base (2 <= B <= 16) or 0 0 to
terminate: 155 16
    155 base 16 is 9B
Enter a non-negative decimal number and base (2 <= B <= 16) or 0 0 to
terminate: 3553 8
    3553 base 8 is 6741
Enter a non-negative decimal number and base (2 <= B <= 16) or 0 0 to
terminate: 0 0
```

[3] The program prompts for a filename and then reads the file to check for balanced curly braces, {}; parentheses, (); and square brackets, []. The program should ignore any character that is not a parenthesis, curly brace, or square bracket. Note that the proper nesting is required.

```
Assume File "balance1.txt" has
((a+b))[{{c}}](){([])} * c[i]
(welcome to C++)
{while (i = 5) ;}
[z]
Run 1:
Enter the file name: balance1.txt
The symbols in balance1.txt are balanced.

Assume File "balance2.txt" has [a(b]c)
Run 2:
Enter the file name: balance2.txt
The symbols in balance2.txt are not balanced.
```

Please upload the following:

- The class .cpp file
- The main program

- The class .h file
- Output File

Machine Problem 2 - Queues

Write a program that simulates a checkout line at a supermarket. The line is a queue object. Customers (i.e. customer objects) arrive in random integer intervals of 1-4 minutes, also, each customer is served in random integers intervals of 1-4 minutes. Obviously, the rates need to be balanced. If the average arrival rate is larger than the average service rate, the queue will grow infinitely. Even with balanced rates, randomness can still cause long lines. Run the supermarket simulation for a 2-hour period (120 minutes) using the following algorithm:

1. Choose a random integer from 1 to 4 to determine the minute at which the first customer arrives
2. At the first customer's arrival time:
   a. Determine customer's service time
   b. Begin servicing the customer;
   c. Schedule arrival time of next customers
3. For each minute of the day
   a. If the next customer arrives, Say so, enqueue the customer, and schedule the arrival time of the next customer.
   b. If the services was completed for the last customer, Say so, dequeue next customer to be serviced and determined customer's service completion time (random integer 1 – 4 added to the current time).

Now run your simulation for 120 minutes, and answer each of the following:

a. What is the maximum number of customers in the queue at any time?
b. What is the longest wait any one customer experiences?
c. What happens if the arrival interval is changed from 1-4 minutes to 1-3 minutes?

Please upload the following:

- The class .cpp file
- The main program
- The class .h file
- Output File
- Answers to the Questions Above

Machine Problem 3 - Linked List

For this assignment you will write a program that inserts 20 random integers from 0 to 100 in order in a linked list object. The program will create another linked list, but with 15 random integers from 0 – 100 in order. The program then will merge those two ordered linked list into a single ordered list.

The function merge should receive references to each of the list objects to be merged and a reference to a list object into which the merged elements will be placed. There should be no duplicate numbers in the final list.

Calculate the sum of the elements and the floating-point average of the elements.

Don't use the STL linked list, you need to build your own linked list. You may use the one in the lecture's example.

An example of the output:

If the first list has
10, 22, 34, 45, 48, 55, 56, 57, 57, 69, 70, 72, 74, 74, 80, 83, 84, 85, 88, 88

And the second list has
50, 55, 57, 79, 81, 84, 87, 88, 90, 92, 95, 95, 95, 96, 99

The result will:
10, 22, 34, 45, 48, 50, 55, 56, 57, 69, 70, 72, 74, 79, 80, 81, 83, 84, 85, 87, 88, 90, 92, 95, 96, 99

The sum of the final list's elements is : xxxxx
The average of the final list is : xxxx.xx

Please upload the following:

- The class .cpp file
- The main program
- The class .h file
- Output File

Machine Problem 4 - Hashing

Write a program to do the following:

loads username/password sets from the file password.txt and insert them into the hash table until the end of file is reached on password.txt. The password.txt file will look something like this with one username/password set per line.

```
mary      changeMe!
```

The program will then present a login prompt, read one username, present a password prompt, and after looking up the username's password in the hash table, will print either "Authentication successful" or "Authentication failure".

The above step will be repeated until the end of the input data (EOF) is reached on the console input stream (cin). The EOF character on the PC's is the CTRL Z character.

To convert from a string to an integer, we can add the ascii value of each character together. For instance, Mary's conversion from string to integer might look something like this:

```
109('m') + 97('a') + 114('r') + 121('y')=441
```

We've converted the string to an integer, but we still need to convert the integer to an index. For an array of 10 elements we can divide by 10 and use the remainder as an index. Combining these two hash functions, we will get Mary's index to be: 441 % 10 = 1

Your primary tasks for this homework are to edit the login.cpp to replace the comments with lines so that it does the following:

1. Insert passwords into the Hash Table.
2. Retrieve one user's Password structure from the Hash Table.
3. Compare retrieved user password to input password and print "Authentication failure" or "Authentication successful."

```
//--------------------------------------------------------------------
//
//                            listlnk.h
//
//--------------------------------------------------------------------
#pragma warning( disable : 4290 )
#include <stdexcept>
#include <new>
#include <cstring>
#include <cmath>
#include <string>
#include <iostream>
#include <fstream>
using namespace std;

template < class T >         // Forward declaration of the List class
class List;

template < class T >
class ListNode                // Facilitator class for the List class
{
private:

    ListNode(const T &nodeData, ListNode *nextPtr);
    T dataItem;      // List data item
    ListNode *next;   // Pointer to the next list node

    friend class List<T>;
};

//--------------------------------------------------------------------

template < class T >
class List
{
public:

    List(int ignored = 0);
    ~List();
    void insert(const T &newData) throw (bad_alloc);        // Insert after
cursor
    void remove() throw (logic_error);                      // Remove data
item
    void replace(const T &newData)  throw (logic_error);    // Replace data
item
    void clear();

    bool isEmpty() const;
    bool isFull() const;

    // List iteration operations
    void gotoBeginning()   throw (logic_error);
    void gotoEnd()  throw (logic_error);
    bool gotoNext();
    bool gotoPrior();
```

```
    T getCursor() const throw (logic_error);                   // Return item
    void showStructure() const;
    void moveToBeginning() throw (logic_error);                        // Move to
beginning
    void insertBefore(const T &newElement)  throw (bad_alloc);  // Insert
before cursor

private:
    ListNode<T> *head,     // Pointer to the beginning of the list
        *cursor;   // Cursor pointer
};

//--------------------------------------------------------------------
//
//                          listlnk.cpp
//
//--------------------------------------------------------------------

template < class T >
ListNode<T>::ListNode(const T &nodeDataItem, ListNode<T> *nextPtr) :
dataItem(nodeDataItem), next(nextPtr)
{}

//--------------------------------------------------------------------

template < class T >
List<T>::List(int ignored) : head(0), cursor(0)
{}

//--------------------------------------------------------------------

template < class T >
List<T>:: ~List()
{
    clear();
}

//--------------------------------------------------------------------

template < class T >
void List<T>::insert(const T &newDataItem) throw (bad_alloc)
{
    if (head == 0)             // Empty list
    {
        head = new ListNode<T>(newDataItem, 0);
        cursor = head;
    }
    else                         // After cursor
    {
        cursor->next = new ListNode<T>(newDataItem, cursor->next);
        cursor = cursor->next;
    }
}

//--------------------------------------------------------------------

template < class T >
```

```
void List<T>::remove() throw (logic_error)
{
    ListNode<T> *p,   // Pointer to removed node
        *q;   // Pointer to prior node

     // Requires that the list is not empty
    if (head == 0)
        throw logic_error("list is empty");

    if (cursor == head)             // Remove first item
    {
        p = head;
        head = head->next;
        cursor = head;
    }
    else if (cursor->next != 0)     // Remove middle item
    {
        p = cursor->next;
        cursor->dataItem = p->dataItem;
        cursor->next = p->next;
    }
    else                              // Remove last item
    {
        p = cursor;
        for (q = head; q->next != cursor; q = q->next)
            ;
        q->next = 0;
        cursor = head;
    }

    delete p;
}

//----------------------------------------------------------------------

template < class T >
void List<T>::replace(const T &newDataItem) throw (logic_error)
{
    if (head == 0)
        throw logic_error("list is empty");

    cursor->dataItem = newDataItem;
}

//----------------------------------------------------------------------

template < class T >
void List<T>::clear()
{
    ListNode<T> *p,      // Points to successive nodes
    *nextP;              // Points to next node
    p = head;
    while (p != 0)
    {
        nextP = p->next;
        delete p;
        p = nextP;
```

```
    }

    head = 0;
    cursor = 0;
}

//--------------------------------------------------------------------

template < class T >
bool List<T>::isEmpty() const
{
    return (head == 0);
}
//--------------------------------------------------------------------

template < class T >
bool List<T>::isFull() const
{
    T testDataItem;
    ListNode<T> *p;

    try
    {
        p = new ListNode<T>(testDataItem, 0);
    }
    catch (bad_alloc)
    {
        return true;
    }

    delete p;
    return false;
}

//--------------------------------------------------------------------

template < class T >
void List<T>::gotoBeginning() throw (logic_error)
{
    if (head != 0)
        cursor = head;
    else
        throw logic_error("list is empty");
}

//--------------------------------------------------------------------

template < class T >
void List<T>::gotoEnd() throw (logic_error)
{
    if (head != 0)
        for (; cursor->next != 0; cursor = cursor->next)
            ;
    else
        throw logic_error("list is empty");
}
```

```
//--------------------------------------------------------------------

template < class T >
bool List<T>::gotoNext()
{
    bool result;   // Result returned

    if (cursor->next != 0)
    {
        cursor = cursor->next;
        result = true;
    }
    else
        result = false;

    return result;
}

//--------------------------------------------------------------------

template < class T >
bool List<T>::gotoPrior()

// If the cursor is not at the beginning of a list, then moves the
// cursor to the preceeding item in the list and returns 1.
// Otherwise, returns 0.

{
    ListNode<T> *p;   // Pointer to prior node
    int result;        // Result returned

    if (cursor != head)
    {
        for (p = head; p->next != cursor; p = p->next)
            ;
        cursor = p;
        result = true;
    }
    else
        result = false;

    return result;
}

//--------------------------------------------------------------------

template < class T >
T List<T>::getCursor() const throw (logic_error)
{
    if (head == 0)
        throw logic_error("list is empty");

    return cursor->dataItem;
}

//--------------------------------------------------------------------
```

```
template < class T >
void List<T>::showStructure() const
{
    ListNode<T> *p;   // Iterates through the list

    if (head == 0)
        cout << "Empty list" << endl;
    else
    {
        for (p = head; p != 0; p = p->next)
            if (p == cursor)
                cout << "[" << p->dataItem << "] ";
            else
                cout << p->dataItem << " ";
        cout << endl;
    }
}

//--------------------------------------------------------------------

template < class T >
void List<T>::moveToBeginning() throw (logic_error)

// Removes the item marked by the cursor from a list and
// reinserts it at the beginning of the list. Moves the cursor to the
// beginning of the list.
{
    ListNode<T> *p;   //  Pointer to prior node
                       // Requires that the list is not empty
    if (head == 0)
        throw logic_error("list is empty");

    if (cursor != head)
    {
        for (p = head; p->next != cursor; p = p->next)
            ;
        p->next = cursor->next;
        cursor->next = head;
        head = cursor;
    }
}

//--------------------------------------------------------------------

template < class T >
void List<T>::insertBefore(const T &newDataItem)
throw (bad_alloc)

// Inserts newDataItem before the cursor. If the list is empty, then
// newDataItem is inserted as the first (and only) item in the list.
// In either case, moves the cursor to newDataItem.

{
    if (head == 0)              // Empty list
    {
        head = new ListNode<T>(newDataItem, 0);
```

```
        cursor = head;
    }
    else                         // Before cursor
    {
        cursor->next = new ListNode<T>(cursor->dataItem, cursor->next);
        cursor->dataItem = newDataItem;
    }
}

//--------------------------------------------------------------------
//                        hashtbl.h
//--------------------------------------------------------------------


template < class T, class KF >
class HashTbl
{
public:
    HashTbl(int initTableSize);
    ~HashTbl();

    void insert(const T &newDataItem) throw (bad_alloc);
    bool remove(KF searchKey);
    bool retrieve(KF searchKey, T &dataItem);
    void clear();

    bool isEmpty() const;
    bool isFull() const;

    void showStructure() const;

private:
    int tableSize;
    List<T> *dataTable;
};

//--------------------------------------------------------------------
//                        hashtbl.cpp
//--------------------------------------------------------------------

template < class T, class KF >
HashTbl<T, KF>::HashTbl(int initTableSize) : tableSize(initTableSize)
{
    dataTable = new List<T>[tableSize];
}

template < class T, class KF >
HashTbl<T, KF>:: ~HashTbl()
{
    delete[] dataTable;
}

template < class T, class KF >
void HashTbl<T, KF>::insert(const T &newDataItem) throw (bad_alloc)
{
    int index = 0;
    index = newDataItem.hash(newDataItem.getKey()) % tableSize;
```

```
    if (dataTable[index].isEmpty())
        dataTable[index].insert(newDataItem);
    else
    {
        dataTable[index].gotoBeginning();
        do
        {
            if (dataTable[index].getCursor().getKey() ==
newDataItem.getKey())
            {
                dataTable[index].replace(newDataItem);
                return;
            }
        } while (dataTable[index].gotoNext());

        dataTable[index].insert(newDataItem);
    }
}

template < class T, class KF >
bool HashTbl<T, KF>::remove(KF searchKey)
{
    T temp;
    int index = 0;
    index = temp.hash(searchKey) % tableSize;

    if (dataTable[index].isEmpty())
        return false;

    dataTable[index].gotoBeginning();
    do
    {
        if (dataTable[index].getCursor().getKey() == searchKey)
        {
            dataTable[index].remove();
            return true;
        }
    } while (dataTable[index].gotoNext());

    return false;
}

template < class T, class KF >
bool HashTbl<T, KF>::retrieve(KF searchKey, T &dataItem)
{
    // apply two hash functions:
    // convert string (searchkey) to integer
    // and use the remainder method (% tableSize) to get the index

    int index = 0;
    index = dataItem.hash(searchKey) % tableSize;

    if (dataTable[index].isEmpty())
        return false;

    dataTable[index].gotoBeginning();
```

```
    do
    {
        if (dataTable[index].getCursor().getKey() == searchKey)
        {
            dataItem = dataTable[index].getCursor();
            return true;
        }
    } while (dataTable[index].gotoNext());

    return false;
}

template < class T, class KF >
void HashTbl<T, KF>::clear()
{
    for (int i = 0; i<tableSize; i++)
    {
        dataTable[i].clear();
    }
}

template < class T, class KF >
bool HashTbl<T, KF>::isEmpty() const
{
    for (int i = 0; i<tableSize; i++)
    {
        if (!dataTable[i].isEmpty())
            return false;
    }

    return true;
}

template < class T, class KF >
bool HashTbl<T, KF>::isFull() const
{
    for (int i = 0; i<tableSize; i++)
    {
        if (!dataTable[i].isFull())
            return false;
    }

    return true;
}

template < class T, class KF >
void HashTbl<T, KF>::showStructure() const
{
    cout << "The Hash Table has the following entries" << endl;
    for (int i = 0; i<tableSize; i++)
    {
        cout << i << ": ";
        if (dataTable[i].isEmpty())
            cout << "_";
        else
        {
            dataTable[i].gotoBeginning();
```

```
            do
            {
                cout << dataTable[i].getCursor().getKey() << " ";
            } while (dataTable[i].gotoNext());
        }
        cout << endl << endl;
    }
}

//--------------------------------------------------------------------
//                         login.cpp
//
//  program that reads in username/login pairs and then
//  performs authentication of usernames.
//--------------------------------------------------------------------

//This will be the data stored in the HashTbl (class T)
struct Password
{
    void setKey(string newKey) { username = newKey; }
    string getKey() const { return username; }

    //this hash converts a string to an integer
    int hash(const string str) const
    {
        int val = 0;

        for (unsigned int i = 0; i<str.length(); i++)
            val += str[i];
        return val;
    }
    string username,
        password;
};

void main()
{
    HashTbl<Password, string> passwords(10);
    Password tempPass;
    string name;      // user-supplied name
    string pass;      // user-supplied password
    //bool userFound;   // is user in table?

        //*****************************************************
        // Step 1: Read in the password file
        //*****************************************************
    ifstream passFile("password.txt");

    if (!passFile)
    {
        cout << "Unable to open 'password.txt'!" << endl;
        exit(0);
    }
    passFile >> tempPass.username;
    while (!passFile.eof() && !passwords.isFull())
    {
        //**add line here to insert passwords into the HashTbl
```

```
        passFile  >> tempPass.password;
    }

    cout << "Printing the hash table:..." << endl;
    //**add line here to show (print) the HashTbl


    //******************************************************
    // Step 2: Prompt for a Login and Password and check if valid
    //******************************************************
    cout << "Login: ";
    while (cin >> name)  // to quit, type CTRL Z in Visual C++
    {
        //**add line here to retrieve user from HashTbl


        cout << "Password: ";
        cin >> pass;

        //**add lines here to compare retrieved user password to
        //**input password and print "Authentication failure"
        //**or "Authentication successful"

        cout << "Login: ";
    }
    cout << endl;
}
```

Please upload the following:

- The class .cpp file
- The main program
- The class .h file
- Output File

Machine Problem 5 - Binary Trees

Write a C++ program to do the following:

1. Inputs a line of text.
2. Tokenizes the line into separate words.
3. Inserts the words into a binary search tree (BST), T1.
4. Do a postorder traversal of the tree T1 and print it, then insert them into T2.
5. Do a preorder traversal of the tree T2 and print it, then insert them into T3.
6. Do an inorder traversal of the tree T3 and print it.
7. Print the heights and the number of leafs in each of the three binary search trees.

Please upload the following:

- The class .cpp file
- The main program
- The class .h file
- Output File

Machine Problem 6 - Algorithms

Please either solve Part 1 or Part 2:

Part 1

For a random list of integers, the maximum number of comparisons required to find a target value by using the binary search is 2(1+ int($\log_2 n$)). This result can be tested experimentally. Modify the function binarySearch( ) to return the number of comparisons the algorithm executes in a successful search and the negative of the number of comparisons required for an unsuccessful search. We provide the prototype for binarySearch( ):

```
template <typename T>
int binarySearch(const T arr[], int first, int last, const T& target);
```

Write a program that declares an integer array ***table*** of ARRSIZE integers and two integer variables sumBinSearchSuccess and sumBinSearchFail.

```
const int ARRSIZE = 50000;
const int RANDOMVALUES = 100000, RANDOMLIMIT = 200000;
int table[ARRSIZE];
int sumBinSearchSuccess = 0, sumBinSearchFail = 0, success = 0;
```

After initializing table with ARRSIZE random integers in the range from 0 to RANDOMLIMIT - 1, apply the selection sort to ***table***. In a loop that executes RANDOMVALUES times, generate a random target in the range from 0 to RANDOMLIMIT – 1, and search for it in ***table*** using the modified binary search.

If the search is successful, increment the integer counter success, and increment sumBinSearchSuccess by the number of comparisons returned from binarySearch( ); otherwise, increment sumBinSearchFail by the negative of the number of comparisons returned from binarySearch( ). At the conclusion of the loop, output the following:

Empirical average case:
sumBinSearchSuccess / static_cast<double>(success)
Empirical worst case:
sumBinSearchFail / static_cast<double>(RANDOMVALUES – success))
Theoretical bound for worse case:
2.0 * (1.0 + int(log(static_cast<double>(ARRSIZE)) / log(2.0)))

Turn in your program and results. Study your results:
1. By how many iterations do the average and worse cases differ?
2. What is the difference between Empirical and Theoretical worst cases?

Part 2

Given the following class; write the implementation code of its member functions.

```
class Complex {
    friend ostream& operator<<(ostream& out, const Complex& theComplex);
```

```
    friend istream& operator>>(istream& in, Complex& theComplex);
    friend Complex operator+(const Complex& lhs, const Complex& rhs);
    friend Complex operator-(const Complex& lhs, const Complex& rhs);
    friend Complex operator*(const Complex& lhs, const Complex& rhs);
    friend Complex operator/(const Complex& lhs, const Complex& rhs);
public:
    Complex(double re = 0.0, double im = 0.0);
    double getReal(void) const;            // return real part
    double getImaginary(void) const;       // return imaginary part
    void setReal(double re);               // sets the real part
    void setImaginary(double im);          // sets the imaginary part
    void convertStringToComplex(const string& complexString);
private:
    double real;
    double imag;
};
```

Complex numbers are added by adding the real and imaginary parts of the summands. That is to say, assuming a + bi is the first complex number and c + di is the second complex number: (a + bi) + (c + di) = (a + c) + (b + d)i

Similarly, the subtraction is defined by: (a + bi) - (c + di) = (a - c) + (b - d)i

The multiplication of the two complex numbers is defined by the following formula: (a + bi)(c + di) = (ac - bd) + (b - d)i

The division of the two complex numbers is defined in terms of complex multiplication, which is described above, and real division. Where at least one of c and d is non-

zero:

Finally, two complex numbers are equal if (a == c) and (b == d).

Create a program to test the class.
For example:

- Enter first complex number: 4.2+3.0i
- Enter second complex number: 2.0+2.1i
- No spaces between the real and imaginary number in each string. Make sure you read the entry at once as a string.
  The output should test all the operators; for example, the addition operator will give the result:
  (4.2 + 3.0i) + (2.0 + 2.1i) = 6.2 + 5.1i

Please upload the following:

- The class .cpp file
- The main program
- The class .h file
- Output File