Solutions to Assessment

Comment [1]: For reference - week 3 solutions https://docs.google.com/document/d/1er7ZuexGJGxSGus43uUXzTJ-CBFSScKGJSfnV9aXNCg/pub

1.
Assume that the structure of a Linked List node is as follows:

```
struct node{
    int data;
    struct node *next;
};
```

What does the following function print for a given Linked List which has elements in the order 1->2->3->4->5->6.

```
1.      void fun(struct node* head){
2.              if(head == NULL)
3.              return;
4.              printf("%d  ", head->data);
5.              if(head->next != NULL )
6.                      fun(head->next->next);
7.              printf("%d  ", head->data);
8.      }
```

A)1 2 3 4 5 6
B)6 5 4 3 2 1
C)1 3 5 5 3 1
D)1 3 5 6 4 2

Ans: C
Explanation:
The program prints alternative elements from head node, till it finds the end of the list and traces back the elements it has printed (last *printf* statement).

2.
What is the output of the following program in C and C++ respectively?

```
//assume required libraries are added
int main(){
        printf("%d",foo(5));          //comment the printf statement for C++
        std::cout<<foo(5);            //comment the cout statement for C
        return 0;
}
int foo(int k){
        if(k==1) return 1;
        else return k*foo(k-1);
}
```

Ignore any Compiler warnings for the above program and choose the correct answers.

1) The C program prints the output as 120
2) The C++ program prints the output as 120
3) C program does not compile.
4) C++ Program does not compile.

Ans:1,4
Explanation:

C++ programming standards requires the function to be declared before using it. Adding the following declaration before main() makes it recognize the function even though the function's code is after main function

int foo(int);

Writing code for function ***foo()*** before main is another way to correct it.

C Program also yields a compiler warning, as function ***foo()*** is not declared before its usage. But it can still find the symbol ***foo()*** even if it is defined after its usage in main.Adding the declaration line makes the warning disappear.

3.

```
class alpha{
   private:
        int data;
   public:
        alpha()
                {data = 10;}
        alpha(int x)
                {data = x;}
        void increase(int x){
        data = data + x;
        }
        void decrease(int x){
        data = data - x;
        }
}a;

int main(){
alpha b(2);
        int X;
        a.increase(10);
        X = a.data * b.data;
```

```
        cout<<X;
        return 0;
}
```

Assuming the required libraries are included, Which of the following statements is true about the above program?
A)The program compiles correctly and prints 20 as output.
B)The program compiles correctly and prints 40 as output.
C)The program compiles correctly and prints 0 as output.
D)The program does not compile.

Ans: D
Explanation:
In the line X=a.data * b.data, program is trying to access private member(a.data & b.data) of class objects outside the scope of class alpha.

4.
What is the time complexity of deleting the first node of a linked list? Assume that only the head pointer is given to you.
State your answer based on the algorithm followed in the lecture Linked Lists.
A)O(1)
B)O(n)
C)O(n^2)
D)O(log n)

Ans :
A)O(1)

Explanation:
_____temp = head;
_____head= head->next;
Above code require constant number of instructions(2 in this case). It does depend on the size(n) of linked list. Any constant time complexity can be represented as O(1) = k*1+c (k=2, c=0, k,c are independent of n(i.e. size)).

5.
Let's say we define an ADT **myInt**. addition(*) and multiplication(+) are the left associative operations with corresponding symbols defined on myInt. The priority of symbols is **+** > ***** . Then the value of the expression 2 * 3 + 4 * 5 is ____ (2,3,4, and 5 are of myInt datatype).

Ans :
19
Explanation:
Note the operator priority is defined as
multiplication(+) > addition(*)
2 * 3 + 4 * 5 is evaluated as shown
→ 2*(3+4)*5 (multiplication is evaluated first as it has higher priority.)
→ 2*12*5 (all are addition as per our myInt, do not confuse * as multiplication operator)
→ 19

6). Abstract Data Types are dependent on their implementation.

T

F

Ans : F

Explanation: An abstract data type (ADT) is defined as a mathematical model of the data objects that make up a data type, as well as the functions that operate on these objects. It is a theoretical concept and provides a logical view of a data type. There should be only one logical view of a data type; however, there may be many different approaches to implementing it.

7). Consider the following segment of code, in which p1 points to 4th element in a List l(1st element refers to the first non-empty element) and p2 points to 6th element. temp is of ElementType. p1 and p2 are of type Position as defined in the lecture.

```
temp = l.retrieve(l.p1); //retrieves Element at p1
l.insert(temp, p2);      //inserts temp, after p2
l.delete(p1);            //deletes the Element at p1
```

Given that the initial List is **abcdefgh**, with character as ElementType, What will be resultant List after running the above program. Specify your answer as a single string.

Ans : **abcefdgh**

Explanation: **d** is inserted after **f** and **d** in position p1 is deleted in the next step.

8. Assume that the List ADT is implemented using a Linked List. A private function is defined as part of the ADT as indicated below. Consider the following function in which P and Q are of type CellType*(as defined in the lecture). In addition, listHead points to the head of a list of integers(value of the list is of type integer).

```
CellType* operation (CellType *P){
    if(P==NULL){
       return P;
    }
    if(P->next==NULL){
       listHead=P;
    }
    else{
       Q=operation(P->next);
       Q->next=P;
       P->next = NULL;
    }
    return P;
}
```

Specify which one of the following statements is correct.

A. Calling this function with parameter as the head of the list "listHead", exchanges the first and last nodes of the list.
B. Calling this function with parameter as the head of the list "listHead",exchanges the last two nodes of the list.
C. Calling this function with parameter as the head of the list ‘listHead’, reverses the input list.
D. None of the above.

Ans : C

Explanation: operation() recursively reverses the list, by returning the tail element of the

sublist, which is already reversed. and sets the last element of the initial list as listHead.

9). The List ADT is defined as in the lecture. p1 and p2 are two instances of the List ADT that have been populated with elements of type ElementType. Consider the following segment of code:

```
List p1, p2;
    Position p;
    ElementType x;
    p = p2.first();                    // p = Position of first element of list p2
    while (p != p2.end()) {
        x = p2.retrieve(p);
        p1.insert(p1.end(), x);        //inserting the elements from p2 at the end
    of p1
        p = p2.next(p);                //updating Position p to its next
    }
```

Which of the following statements is correct?

A. Replaces elements from List p1 with that from List p2.
B. Appends the List p2 to the end of p1.
C. Merges the Lists p1 and p2 and removes duplicates from it.
D. It will lead to a segmentation fault.

Ans : B

Explanation:
p = p2.first(); //p is initialized to position of first element of p2.
In side while loop, elements of list p2 are inserted at the end of list p1.

Solutions to Programming Assignments

Question 1: Implementing a Hash Table ADT

A hash table is a data structure that maps a set of keys to a given set of values. It uses a hash function to compute an index into an array of buckets or slots, from which the correct value can be found or to which a correct value can be inserted. A hash function will assign each key to a unique bucket or slot. In practice, however, there may be more than one key that will hash to the same bucket. Such an assignment is said to result in a collision.

In this assignment, you have to implement a Hash Table ADT using arrays and linked lists. The algorithm should accommodate the features described below:

1. Your program should take as input 10 non-negative integers as key values and insert them to a hash table.
2. The hash table has to be implemented using the hash function h(x) = x%10, where x is the input value.
3. Input values that return the same value for h(x) have to be properly accommodated such that if there is a collision, the index from the array to the appropriate bucket should point to a linked list that can take in all the values that hash to the same bucket. i.e., you can have an array of linked lists such that each entry of the array is a pointer to a bucket. For e.g., when h(x) returns 4, the input value should go into the linked list starting from A[4].
4. Your program should report when there is a collision and must specify the position (starting at 1 for the 1st element) in the input list of that key value where the collision occurred. It should also report the bucket number and the number of entries in each bucket
5. If there are no collisions, the program should report 'NO'.
6. HashTable class has a printCollision function which you cannot re-write. The string you want to print has to be made available in the private data variable "collision" using listCollision function in class HashTable.

INPUT:
A list of 10 non-negative integers with a single space between the entries.

OUTPUT:
3-tuple indicating the position in the input list where the collision occurred, the corresponding bucket number and the number of elements in that bucket, each 3-tuple has to be separated by a single space.

CONSTRAINTS:
0 <= key <1000

Sample input:
77 23 231 785 900 345 287 16 5 234

Expected Output:
(6,5,2) (7,7,2) (9,5,3)

Test Cases:

Public test cases:

Input	Output
10 187 163 143 122 178 196 105 111 514	(4,3,2)
212 674 348 239 165 123 845 456 119 129	(7,5,2) (9,9,2) (10,9,3)
100 101 102 103 104 105 106 107 108 109	NO
11 22 31 42 51 62 71 82 91 102	(3,1,2) (4,2,2) (5,1,3) (6,2,3) (7,1,4) (8,2,4) (9,1,5) (10,2,5)
0 10 20 30 40 50 60 70 80 90	(2,0,2) (3,0,3) (4,0,4) (5,0,5) (6,0,6) (7,0,7) (8,0,8) (9,0,9) (10,0,10)

Private test cases:

Input	Output
0 0 0 0 0 0 0 0 0 0	(2,0,2) (3,0,3) (4,0,4) (5,0,5) (6,0,6) (7,0,7) (8,0,8) (9,0,9) (10,0,10)
1 22 333 4 55 666 7 88 999 000	NO
10 15 20 25 30 35 40 45 50 55	(3,0,2) (4,5,2) (5,0,3) (6,5,3) (7,0,4) (8,5,4) (9,0,5) (10,5,5)
818 218 812 212 817 217 816 216 814 214	(2,8,2) (4,2,2) (6,7,2) (8,6,2) (10,4,2)
1 11 111 2 22 222 3 33 333 000	(2,1,2) (3,1,3) (5,2,2) (6,2,3) (8,3,2) (9,3,3)

Solution:

```
#include <iostream>
#include <stdlib.h>
#include <string>
#include <sstream>

using namespace std;

/* Definitions as shown in the lecture */
typedef struct CellType* Position;
```

```
typedef int ElementType;

struct CellType{
    ElementType value;
    Position next;
};

/* *** Implements a List ADT with necessary functions.
You may make use of these functions (need not use all) to implement your
HashTable ADT ***/

class List{

    private:
        Position listHead;
        int count;

    public:
        //Initializes the number of nodes in the list
        void setCount(int num){
            count = num;
        }

        //Creates an empty list
        void makeEmptyList(){
            listHead = new CellType;
            listHead->next = NULL;
        }

        //Inserts an element after Position p
        int insertList(ElementType data, Position p){
            Position temp;
            temp = p->next;
            p->next = new CellType;
            p->next->next = temp;
            p->next->value = data;
            return ++count;
        }

        //Returns pointer to the last node
```

```
        Position end(){
            Position p;
            p = listHead;
            while (p->next != NULL){
                p = p->next;
            }
            return p;
        }

        //Returns number of elements in the list
        int getCount(){
            return count;
        }
};
/*UPTO THIS LINE, THE CODE WIL BE PREPENDED BEFORE YOUR FUNCTIONS*/
class HashTable{
      private:
            List bucket[10];
            int bucketIndex;
            int numElemBucket;
            Position posInsert;
            string collision;
            bool reportCol; //Helps to print a NO for no collisions

      public:

            HashTable(){
                  int i;
                  for (i=0;i<10;i++){
                        bucket[i].setCount(0);
                  }
                  collision = "";
                  reportCol = false;
            }

            int insert(int data){
                  //Hash Function
                  bucketIndex = data % 10;

                  //Creates a list on an initially empty array entry
```

```
                if (bucket[bucketIndex].getCount() == 0)
                      bucket[bucketIndex].makeEmptyList();


                //Find position to insert
                posInsert = new CellType();
                posInsert = bucket[bucketIndex].end();

                //Return number of elements after insert
                numElemBucket = bucket[bucketIndex].insertList(data,
posInsert);
                return numElemBucket;
          }

          void listCollision(int pos){
                reportCol = true;
                string str1;
                std::stringstream ss1, ss2, ss3;

                ss1 << pos;
                str1 = ss1.str();
                collision = collision + "(";
                collision = collision + str1;
                collision = collision + ",";

                ss2 << bucketIndex;
                str1 = ss2.str();
                collision = collision + str1;
                collision = collision + ",";

                ss3 << numElemBucket;
                str1 = ss3.str();
                collision = collision + str1;
                collision = collision + ") ";
          }

          void printCollision();

};
```

```
int main(){

      HashTable ht;
      int i, data;

      for (i=0;i<10;i++){
            cin>>data;
            if (ht.insert(data) != 1){
                  ht.listCollision(i+1);
            }
      }

//Prints the concatenated collision list
   ht.printCollision();

 }

void HashTable::printCollision(){
    if (reportCol == false)
        cout <<"NO";
    else
        cout<<collision;
}
```

Question 2: Big Int Addition

Primitive **int** data type handles integers each of which can be stored using 32 bits. Similarly **long** data type also has a limit. In this assignment you have to implement a data type which handles numbers larger than **int** , **long** and **long long.**

You are given a number in decimal form. You need to build a data structure called *BigInt* which can store that data, and perform the following basic operations on that data type:

1.BigInt::add(BigInt A), returns the result in another BigInt, without altering the numerical value of the caller.
2.BigInt::print(), this function must print Most significant digit at the beginning and must not print any leading zeroes.
Ex: 123 (correct)
0123 (incorrect)
3.populate(BigInt &A, char * str), populates the List in BigInt using public methods of BigInt.

Note that *populate(BigInt &A, char * str)* method is not a function of BigInt.

Input:

Input will have two lines, each line contains a single BigInt.

Sample Input:
8132393963283
216396739263921391

Output:

Output contains the result of the addition in a single line.
Output of the above sample input is as follows.

Sample Output:
216404871657884674

Constraints:

1 ≤ Input length of BigInt ≤ 40 digits.
Input for BigInt is always positive

Hints: (optional)

Store the numbers in reverse order, as addition is performed from Least Significant Digit. For example store the number 122 as 2->2->1, instead of 1->2->2.

Note: To get the initial template code, delete everything in editor and submit.

Test Cases:

Private Test cases:

Input	Output
0182638230003821323218326100237561800008163	1826382300061775032191424
0000 00000000000000000000000001	1
85000000000000000000000000 15000000000000000000000000	10000000000000000000000000
1450000000000000907887 0189093487289367482346283	190543487289367483254170
012000000000000000000000000555	120000000000000000000000010
000008132393963283 216396739263921391	216404871657884674
9921963823826300219638238263002196382382630021963823826300219638238499219638238263002196382382630021963823826300219638238426476526004392764765260043927647652600439276476	198439276476526004392764765260043927647666

Solution:

```
/***Prefix Code***/
void populate(BigInt &A, char * str){
    /*copy the numbers one by one at the beginning of the BigInt*/
    int i;
    for(i=0;i<(int)strlen(str);i++){
            A.prepend((int)(str[i]-'0'));
    }
}
void BigInt::append(int num){
   L.append(num);
}
void BigInt::prepend(int num){
   L.prepend(num);
}
/*****************************************************************************
*maintain two pointers temp1, temp2 each initialized at
*the beginning of two BigInt List.
*1.Keep iterating temp1, temp2 over the BigInt Lists until one of them is NULL
    val = temp1->getData() + temp2->getData() + carry
    res = val%10
    carry = val/10
    //Add res to the resultant BigInt(in this code, it is C) at the beginning.
*2.Evaluate the re, carry untill you reach the end of the other List.
*3.If the final carry > 0 add it to C.
```

```
**********************************************************************/
BigInt BigInt::add(BigInt A){
   BigInt C;
   int res, carry=0, value;

   ListNode * temp1, *temp2;
   temp1 = L.getHead();
   temp2 = A.L.getHead();

   while(temp1!=NULL || temp2!=NULL){
         value = (temp1==NULL? 0:temp1->getData()) + (temp2==NULL? 0:temp2->getData()) + carry;
         res = value % 10;
         carry = value / 10;

         C.L.append(res);
         if(temp1!=NULL)temp1=temp1->getNext();
         if(temp2!=NULL)temp2=temp2->getNext();
   }

   if(carry>0) C.L.append(carry);
   return C;

}
void BigInt::removeZeroes(){
  //helper removes additional zeroes at Most Significant bit. Not mandatory.
   //remove all zeroes from the end, until non-zero value or Head is reached.
   while(L.last()->getData()==0 && L.last()!=L.getHead()) L.popBack();
}
void BigInt::print(){
         /*prints a BigInt, and it must not have additional zeroes at most significant bit*/
         removeZeroes();
         L.printReverse();
}
/***PostFix Code****implementation of main***/
```

Question 3: Implement a Matrix ADT

You have to implement a matrix ADT using concepts of C++ *classes* taught in the lectures. The input matrices would be square matrices. The *class* must support the following functions:

1. Matrix Addition
2. Matrix Subtraction
3. Matrix Multiplication

Your program should take as input: dimension of a square matrix N, two matrices of size N x N with integer values, and one operator symbol (+, - ,*). It must perform the corresponding operation using member functions of Matrix class.

INPUT:
In the first line, one integer which is the dimension of the matrix and one operator (one of +, - or *)
Two NxN matrices one after the other, supplied one row at a time.

OUTPUT:
Resultant matrix after performing the specified operation, one row at a time. For subtraction, if A is the first matrix and B is the second matrix, perform A-B.

CONSTRAINTS:
The inputs will satisfy the following constraints:
1<=N<=10
There is no need to validate the value of N.
There are no constraints on the values of the entries of the matrices.

Test Cases:

Public test cases:

Input	Output
3 + 1 2 3 4 5 6 7 8 9 9 8 7 6 5 4 3 2 1	10 10 10 10 10 10 10 10 10 Done
2 * 3 4	31 30 23 30 Done

9 2 1 2 7 6	
4 - 11 22 33 44 12 23 34 45 13 24 35 46 14 25 36 47 11 22 33 44 12 23 34 45 13 24 35 46 14 25 36 47	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 Done
3 * 101 220 671 234 549 685 427 531 765 1 0 0 0 1 0 0 0 1	101 220 671 234 549 685 427 531 765 Done
2 - 3 4 9 2 1 2 7 6	2 2 2 -4 Done

Private test cases:

Input	Output
3 + 1 1 1 1 1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 -1	0 0 0 0 0 0 0 0 0 Done
5 - 100 100 100 100 100 -100 -100 -100 -100 -100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100	0 0 0 0 0 -200 -200 -200 -200 -200 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 Done
4 * 23 45 82 14	203 -1177 5907 2962 -1474 11381 1630 463

24 -98 17 10 -32 21 24 11 87 34 92 92 23 45 82 14 24 -98 17 10 -32 21 24 11 87 34 92 92	-43 -2620 -679 1038 7877 5643 18384 11034 Done
1 + 2 3	5 Done
4 - 23 45 82 14 24 -98 17 10 -32 21 24 11 87 34 92 92 12 13 14 15 21 23 24 15 17 18 19 31 76 54 29 56	11 32 68 -1 3 -121 -7 -5 -49 3 5 -20 11 -20 63 36 Done
1 * 5 6	30 Done

Solution:

```
#include <iostream>
#include <cstring>

using namespace std;

struct matrixType{
    int matDimension;
    int matValues[10][10];
};

class MatrixADT{

    private:
        matrixType resultMatrix;

    public:

      //Member function declarations
      void intializeResultMatrix(int);
      matrixType add(matrixType, matrixType);
```

```
        matrixType subtract(matrixType,matrixType);
        matrixType multiply(matrixType,matrixType);
        void printResult();
};

//Member function definitions

//ADD
matrixType MatrixADT::add(matrixType M1, matrixType M2){

      int i,j;
      for (i=0;i<M1.matDimension;i++)
            for (j=0; j<M1.matDimension;j++){
                  resultMatrix.matValues[i][j] =
M1.matValues[i][j]+M2.matValues[i][j];
            }
      return resultMatrix;
}

//SUBTRACT
matrixType MatrixADT::subtract(matrixType M1, matrixType M2){

      int i,j;
      for (i=0;i<M1.matDimension;i++)
            for (j=0; j<M1.matDimension;j++){
                  resultMatrix.matValues[i][j] = M1.matValues[i][j]-
M2.matValues[i][j];
            }
      return resultMatrix;
}

//MULTIPLY
matrixType MatrixADT::multiply(matrixType M1, matrixType M2){

      int i,j,k;
      for (i=0;i<M1.matDimension;i++)
            for (j=0; j<M1.matDimension;j++){
                  resultMatrix.matValues[i][j] = 0;
      }

      for (i=0;i<M1.matDimension;i++)
            for (j=0; j<M1.matDimension;j++)
                  for (k=0;k<M1.matDimension;k++){
                        resultMatrix.matValues[i][j] =
resultMatrix.matValues[i][j]+M1.matValues[i][k]*M2.matValues[k][j];
            }
      return resultMatrix;
```

```
}

//Intialize Result Matrix entries to zero
void MatrixADT::intializeResultMatrix(int dim){

      int i,j;

      resultMatrix.matDimension = dim;

      for (i=0;i<resultMatrix.matDimension;i++){
            for (j=0; j<resultMatrix.matDimension;j++){
                  resultMatrix.matValues[i][j] = 0;
            }
      }


}

int main(){

/*Enter your code here to accept two input matrices as instances of class
Matrix and
perform the operations using member functions, display the result matrix
using member function*/

      MatrixADT maX;
      matrixType M1, M2;
      char op;
      int i,j,dim;

      cin>>dim>>op;

      M1.matDimension = dim;
      M2.matDimension = dim;

      maX.intializeResultMatrix(dim);

      //Accept matrix entries
      for (i=0;i<dim;i++)
            for (j=0; j<dim;j++){
                  cin>>M1.matValues[i][j];
      }
      for (i=0;i<dim;i++)
            for (j=0; j<dim;j++){
                  cin>>M2.matValues[i][j];
      }

      //Call MatrixADT member function depending on operator
```

```
      if (op=='+'){
            maX.add(M1,M2);
      }
      else if (op=='-'){
            maX.subtract(M1,M2);
      }
      else{
            maX.multiply(M1,M2);
      }


/* DO NOT EDIT the code below; if you edit it, your program will not give
correct output */
      maX.printResult();

}


void MatrixADT::printResult(){

   int i,j;

   for (i=0;i<resultMatrix.matDimension;i++){
       for (j=0; j<resultMatrix.matDimension-1;j++){
           cout<<resultMatrix.matValues[i][j]<<" ";
       }
      cout <<resultMatrix.matValues[i][j]<<"\n";
   }
   cout <<"Done";

}
```