PROBLEM 1:  (Hidden in the Stack; (12 pts))

The Stack class discussed in lecture (StackAr.h) is given on the Exam 1 handout.

PART A (6 pts):
Write the function *Max* which returns the maximum value in a stack of integers. *Max* assumes that the stack is not empty and it does not change the stack in any way. The function *Max* is NOT a part of the Stack class, but rather part of a user’s program.

Complete *Max* below the following header.

```c
int Max(const Stack<int> & numbers)
// precondition: numbers is not empty
// postcondition: returns the maximum value on the stack numbers
{
}
```

PART B (6 pts):
Write the member function *Max* which returns the maximum value in a stack. *Max* assumes that the stack is not empty and it does not change the stack in any way. This time, the function *Max* is a part of the Stack class. Assume assignment and comparison operators are defined for whatever type the Stack is used with.

For example, if a stack named *scores* of type int is declared, then *scores.Max()* returns the maximum score on the stack. If a stack named *names* of type string is declared, then *names.Max()* returns the maximum name (the last name in alphabetical order).

Complete *Max* below the following header.

```c
template <class Etype>
Etype Stack<Etype>::Max()
// precondition: stack is not empty
// postcondition: returns maximum value on the stack
{
}
```

PROBLEM 2:  (Analysis: (18 pts))

1. (2 pts) What is the worst case running time (big-Oh, O(?)) of the following code segment?

```c
int k, j, h;
int sum = 0;
for (k = 1; k <= N; k++)
    for (j = 1; j <= P; j++)
        for (h = 1; h <= N; h++)
            sum = sum + h;
```
2. (2 pts) What is the worst case running time (big-Oh, O(?)) of the following code segment?

```c
int k, j;
int sum = 0;
for (k = 1; k < N; k++)
    for (j = k; j < N; j++)
        sum = sum + j;
```

3. (10 pts) Assume you are given a pointer to a singly linked list of elements in sorted order from smallest to largest. The pointer to the linked list points to the smallest element. The list may contain duplicate items.

For example, the list below contains 8 elements.

```
list
\[3\ 3\ 6\ 9\ 11\ 11\ 15\]
```

Consider the most efficient implementation of each of the following operations, given only a pointer to the front of the list. For example, find the maximum would be implemented as a function with the only parameter a pointer to the linked list. DO NOT modify the data structure. What is the worst case running time (big-Oh, O(?)) for each operation if the list contains N elements?

(a) Find the maximum element Running time:
(b) Find the minimum element Running time:
(c) Is X an element in the list Running time:
(d) Print the element that occurs most often Running time:
(e) List the elements from largest to smallest Running time:

4. (4 pts) Consider the following silly Mystery function but don’t attempt to decipher what it computes.

```c
int Mystery (int N)
// precondition: N>=0
{
    int tot = 0;
    int j, h;
    for (j=1; j<=N; j++)
    {
        tot += Mystery(N-1);
        h = N;
        while (h>0)
        {
            tot += j;
            h = h/2;
        }
    }
    return tot;
}
```
Write a recurrence relation describing the running time of Mystery. DO NOT solve the recurrence or compute the big-Oh, just give the recurrence relation.

PROBLEM 3: (Get a Job: (18 pts))

This problem refers to a linked list of companies, for which each company contains the names of students they plan to interview. Refer to the Exam 1 Handout for an example figure and the definition of CNode and SNode (note both nodes have constructors).

Part A (5 pts)
Write the function IsMember whose header is given below. IsMember returns true if the student name is a member of a linked list of students.

For example, on the Exam 1 handout, where interview is the pointer to the data structure, IsMember(interview->first, "Hendren") returns true since Hendren is in the IBM list of students, IsMember(interview->next->first, "Weber") returns true since Weber is in the GE list of students, and IsMember(interview->first, "Farlow") returns false, since Farlow is not in the IBM list of students.

Complete IsMember below the following header.

```cpp
bool IsMember(SNode * list, string name)
// postcondition: returns true if name is a member of the list,
// otherwise returns false
{
}
```

Part B (5 pts)
Write the function PrintCompanies whose header is given below. PrintCompanies prints the name of companies (on one line) the particular student is interviewing with.

Refering to the data structure on the Exam 1 handout where interview is the pointer to the data structure, PrintCompanies(interview, "Hendren") prints

IBM Glaxo

PrintCompanies(interview, "Dyer") prints

IBM Microsoft SAS

You may call the function IsMember that you wrote in PART A. Assume IsMember works correctly regardless of what you wrote for PART A.

Complete PrintCompanies below the following header.

```cpp
void PrintCompanies (CNode * list, string student)
// postcondition: Prints a list of companies on one line that
// student is interviewing with.
{
}
```
Part C (8 pts)

Write the function CopyList whose header is given below. CopyList is given a list of companies and the name of two companies, company1 and company2, and makes a copy of company1’s list with the new company name company2. Company2’s list is inserted into the list of companies to the immediate right of company 1. The students in company2’s list do not have to be in the same order as they were in company1’s list.

Refering to the first data structure on the Exam 1 handout, CopyList(interview, ”GE”, ”Intel”) might modify this structure to look like the second data structure. Note that the Intel list is a copy of the GE list; however, the students do not have to appear in the same order.

Complete CopyList below the following header.

```c
void CopyList(Cnode * list, string company1, string company2)
// precondition: company1 is a company name in list
// postcondition: makes a copy of company1’s student list and inserts the
// list to the immediate right of company1 in list with the new
// company name company2. The students in company2’s list do not
// have to be in the same order as they are in company1’s list.
{
}
```

PROBLEM 4: (Poly want another: (30 pts))

This problem refers to the linked list implementation of the Polynomial class. Refer to the Exam 1 Handout for the definitions of Node and the Polynomial class.

NOTE that the implementation contains two private data items myPoly and myNumNodes. You must use this implementation (and these variables) in the questions that follow.

You may assume that Nodes in the linked list are in **sorted order from largest degree to smallest degree**, and myPoly points to the Node of highest degree. You may also assume (except for Part F) that there is only one Node per degree and no nodes for terms with a zero coefficient, unless the Polynomial is equal to 0, in which case there is one Node with coefficient 0.

Part A (5 pts)

Write the member function LowestDegree whose header is given below. LowestDegree returns the smallest degree of the terms in the polynomial.

If Polynomial A was $3.2x^8 + 5.2x^7 + 5.4x^3$, then A would be represented by the linked list shown below and A.LowestDegree() would return 3 since $x^3$ is the lowest degree term. If Polynomial B was $3.7x^5 + 3.2x^2 + 1.4x^2 + 6.8$, then B would be represented by a linked list of 4 nodes and B.LowestDegree() would return 0 since 6.8 is the lowest degree term (which is the same thing as $6.8x^0$).

```
myPoly

3.2  8  5.2  7  5.4  3
```

Complete LowestDegree below the following header.

```c
int Polynomial::LowestDegree()
```
Part B (5 pts)

Write the member function Swap whose header is given below. Swap exchanges the values of two polynomials.

For example, if Polynomial A was $3.2x^8 + 5.2x^7 + 5.4x^3$, and Polynomial B was $3.7x^5 + 3.2x^2 + 1.4x^x + 6.8$, then A.Swap(B) would change Polynomial A to be $3.7x^5 + 3.2x^2 + 1.4x^x + 6.8$ (B’s value) and Polynomial B to be $3.2x^8 + 5.2x^7 + 5.4x^3$ (A’s value). Note that B.Swap(A) would accomplish the same thing.

Complete Swap below the following header.

```cpp
void Polynomial::Swap(Polynomial & Poly) {
    // postcondition: Swap the values of two Polynomials
}
```

Part C (4 pts)

Suppose that you did not write a copy constructor for the Polynomial class. Explain what would happen if you executed the statements shown below.

Polynomial A(3.2, 4.8, 1.2);
Polynomial B(A);

Part D (4 pts)

Consider the following code segment.

```cpp
void Process(Polynomial & P);
void DoIt(Polynomial Q);

main ()
{
    Polynomial Poly(3.2, 4.5, 2.1);
    Polynomial B(Poly);
    Polynomial C = B;
    Polynomial D;
    D = Poly;
    Process(C);
    DoIt(D);
}
```
Assume that a copy constructor and an "=" operator are written for the Polynomial class. For each line in the main function above, indicate to the right whether a constructor, copy constructor, or none is invoked.

**Part E (4 pts)**
Suppose that you did not write a destructor for the Polynomial class. Explain 1) when a destructor is invoked, and 2) what would happen in the Polynomial class.

**Part F (8 pts)**
Write the **member** function *Compress* whose header is given below. For this problem only, assume that the implementor has been lazy in some of the member functions. Terms are sorted by degree, however there may be several terms with the same degree. *Compress* compresses Nodes with the same degree into one Node.

For example, suppose Polynomial A was $3.2x^8 + 5.2x^7 + 5.4x^3$. It might be represented by the first polynomial shown below (there are three nodes for the $x^8$ term whose coeff sum is 3.2). Compress() would first compress nodes of like degree resulting in the second polynomial shown below. The last statement in Compress() should call RemoveZeros() to remove the Nodes with coeff zero from the list. You may assume the function RemoveZeros() already exists (you do not have to write it) and that it removes Nodes with coefficient 0 unless there is just one node in the list.

![Diagram](https://via.placeholder.com/150)

Complete *Compress* below the following header.

```cpp
void Polynomial::Compress()
// postcondition: Compresses Nodes with the same degree
{
}
```

---

**CPS 100** **Exam 1 Handout** **Fall 1996**

**PROBLEM 1: StackAr.h**

```cpp
template <class Etype>
class Stack
{
  public:
    Stack( ); // construct empty stack
    ~Stack( ); // destructor
    const Stack & operator=( const Stack & Rhs ); // assign Rhs to self
```
void Print() const;  // print stack to cout
void Push( const Etype & X );  // push X onto top of stack
void Pop( );  // pop top element
const Etype & Top( ) const;  // return top element (NO pop)
bool IsEmpty( ) const;  // return true if empty, else false
bool IsFull( ) const;  // return true if full, else false
void MakeEmpty( );  // empty stack (no elements)

int NumPushes();  // returns # of elements pushed
int Size();  // returns # of elements in stack

private:
int myMaxSize;  // largest possible size of stack
int myTop;  // index of top item
int myPushes;  // number of items pushed (may not be
               // current size of stack)

Vector<Etype> myArray;
}
PROBLEM 3: SNode, CNode, and interview list

struct SNode
{
    string name;
    SNode * next;
}

SNode (string nm, SNode * nx = NULL)
{ name = nm; next = nx }
};

struct CNode
{
    string name;
    CNode * next;
    SNode * first;
}

CNode (string nm, CNode * nx = NULL, SNode * fr = NULL)
{ name = nm; next = nx; first = fr; }
};

Parts A and B

Part C
After calling CopyList(interview, "GE", "Intel") on the above structure.
struct Node
{
    double coeff;
    int degree;
    Node * next;
};

// member functions

// Polynomial(double, double, double);
//     Create a polynomial of degree 3 or less, has default values
//     if fewer than 3 arguments are sent.
// ~Polynomial();
//     destructor
// int Degree();
//     return the coefficient of the highest term with nonzero degree

class Polynomial
{
    public:
        Polynomial(double, double, double); // constructor with defaults
        ~Polynomial(); // destructor
        int Degree(); // return degree of polynomial
        // not all Member functions shown

    private:
        Node * myPoly; // pointer to linked list
        int myNumNodes; // number of nodes in polynomial
};