1.124J Foundations of Software Engineering

Problem Set 2

Due Date: Tuesday 9/26/00

Reference Readings: *From C++ Primer (in addition to the reference reading from PS1):*

- *Chapters 6, 7, 8*

Problem 1:[10%]

This is a multiple choice question. You need to select the correct answer(s), or provide the answer, as indicated in the following questions. There may be one or more than one correct answers in the multiple choice questions. Please submit this page, stapled together with the hardcopies for the other problems, with the selected answer(s) to each question circled. Please, write your name and username clearly on this page as well. Each question counts for 2 point.

1. Which of the following functions, whose declarations are given below, will be called:

```
float myF;
printMyF(2.0*myF);
```

a. *void printMyF(void)*
b. *void printMyF(double)*
c. *void printMyF(float)*
d. *float printMyF(float)*
e. none of the above

2. If you declare members inside a class without labeling them public, private, or protected

a. they are assumed to be private.
b. they are assumed to be protected
c. they are assumed to be public
d. it is illegal not to explicitly specify the access level

e. none of the above

3. Which of the following is/are True?

a. The definition, and not only the declaration, of an inline function needs to be available in each source code file that uses that function.
b. Only a member function or a friend function can access a private member of the class.
c. The memory location where the object which invoked a member function is given by &this inside the member function.
d. Pointers of different types may not be assigned to one another without a cast operation.
e. All of the above

4. Which of the following is/are True?

a. Since in a member function "*this*" is a pointer to an object of that class it can be used to store a dynamically allocated object in the body of the member function definition.
b. It is not allowable to define (i.e. allocate memory) a static data member within the declaration of the class.
c. The copy constructor takes an object of the same class as argument passed by value.
d. All of the above
e. None of the above

5. Which of the following is/are True?

a. The division operator(/) can be overloaded as a member function with no parameters.
b. The return data type of a type conversion operator may optionally be specified.
c. The statement: *delete px;* deletes the pointer *px* (i.e. release the memory allocated to store *px*).
d. Member data and functions defined in the private part of a class are accessible by member functions of a class derived from the former class.
e. None of the above

6. Which of the following give(s) the element A[3][4] of an array A of size 10x10?

a. *(&A[0][0]+3*4)*
b. *(A[3]+4)*
c. *(*(A+3))[4]*
d. *((*(A+3))+4)*
e. All of the above.

7. 8,9,10. Indicate which of the following statements are True and which are False:

7. It is not allowable to define a constructor to have *void* return type since it returns nothing: T / F

8. It is allowable to specify a destructor to have void as parameters, since it does not take any arguments: T / F

9. It is not possible to initialize a constant member data in the body of a constructor of the class: T / F

10. The definition *double *a[100]* causes C++ to allocate storage for 100 doubles: T / F

Problem 2:[30%]

In this problem you will implement two member functions of a class, named ***ElevatorStack***, which can be used to create a stack.

The declaration of the class is given in the file ***ps2_2.h***. It consists of an array of structures, named *Guard*, an integer which indicates the current position in the stack, named *position*, and a double, named *weight*, to store the current total weight of the "guards" currently in the elevator. The declaration of the structure Guard is also given in the file *ps2_2.h*. The prototypes of the functions *push()* and *pop()*, which are used for the stack operations, are also given in the file *ps2_2.h*.

The *main()* program in the file ***ps2_2.C*** is supposed to simulate a stack to represent the entrance (push) and the exit (pop) of guards in an elevator. You should also provide in the file *ps2_2.C* the definitions of the functions *push()* and *pop()*, whose declarations are given in the file *ps2_2.h*, and the missing code from the externally defined constructor and destructor of the class *ElevatorStack***.**

For the ***push(char *name, double weight)*** make sure that you check whether the maximum allowable weight or/and the maximum allowable number of persons in the elevator have been exceeded. In that case the function should provide the user with an informative message for not allowing a *push()*, i. e. another person to enter the elevator. When it is allowed the data provided by the user should be stored

in the members of the structure in the current position. The memory for the name is dynamically allocated according to its length.

For the ***pop()***, which is called whenever the provided name is "POP", you should check whether the stack is empty and have accordingly a message. The function must release the dynamically allocated memory for the name of the guard who is exiting (according to last-in first-out) and reduce the total weight by its individual weight.

Before exiting both *push()* and *pop()* the function must inform the user with the current status, i.e. the total weight and number of guards in the elevator.

Please, submit both the printout of the modified code and the output when executing the completed program with the data in the provided file ***dat2_2***.

You may use redirection for both the input and output. e.g.

% ps2_2 < dat2_2 > res2_2 ; ensript -2Gr res2_2

You need to submit both ***ps2_2.h***. and ***ps2_2.C*** files, although you should make changes only to the latter file.

ps2_2.h

```
// Problem Set#2 - Problem#2  [ps2_2.h]

#ifndef PS_2_2_H
#define PS_2_2_H

#define MAX_PERSONS 4
#define MAX_WEIGHT 900

int main (void) ;

struct Guard
 {
   char *name;
   double weight;
 };


class ElevatorStack
```

```
{
private:
  Guard guards[MAX_PERSONS];
  int position;
  double totalWeight;

public:
  ElevatorStack();      //  Constructor
  ~ElevatorStack();     //  Destructor
  void push(char *name, double weight);
  void pop(void);
};

#endif
```

ps2_2.C

```
 // Problem Set#2 - Problem#2  [ps2_2.C]

#include  <iostream.h>
#include  <stdlib.h>
#include <string.h>
#include <iomanip.h>
#include "ps2_2.h"

int main (void)
{
  ElevatorStack elevatorStack;
  char name[20];
  double weight;

  cin.clear();
  while(1)
    {
      cout << "\n\n Guard's name : " ;
      cin >> name ;

      if(cin.eof())       break;

      if(strcmp(name,"POP"))
  {
```

```
   cout << "\n Weight : " ;
   cin >> weight ;
   elevatorStack.push(name,weight);
  }
     else
 elevatorStack.pop();
   }

  cout << "\n\n Exiting the program normally" << endl << endl;
  return  EXIT_SUCCESS ;
}



ElevatorStack:: ElevatorStack()
{
  // The constructor should set the position and
  //   totalWeight to 0 and 0.0, respectively.
}


ElevatorStack::~ElevatorStack()       //  Destructor
{
  // The destructor should releasing the dynamically allocated
  //   memory for name of each guard in the array of structures

}


/*****************************************************

Here you should provide the externally defined
member functions push() and pop().

push()
------
  - The push member function should check whether the
    number of people or their weight has exceeded the
    allowable limits.
      - In each of the above case it should print an
        informative message why push is not possible.
  - If it is allowable a new "Guard" structure should
```

be added to the stack and the number of people in the elevator and their weight should be updated. This information should be printed to the user.

pop()

- The pop() member function should check whether the stack is empty and in that case print a message
- Whenever the message is not empty a "Guard" should be popped out of the stack and the updated information for the number of people in the elevator and their weight should be printed to the user.

**********/

Problem 3:[60%]

To lower heavy testing equipment for deep underground experiments, it is often necessary to connect a number of cables in series to reach great depths. Supporting the weight of the machinery, the cables will undergo axial stress and extend beyond their original length. Your task is to write a program to determine how much a series of cables will stretch under a given load and to see if the cables will be strong enough to support that load.

Bar and cable elements in series subjected to axial load are governed by the equation of springs in series:

$$F = \left(\frac{1}{\frac{1}{k_1} + \frac{1}{k_2} + \ldots + \frac{1}{k_n}} \right) \Delta x$$

where the stiffness, *k*, of each element is equal to its area, *A*, multiplied by its modulus of elasticity, *E*, divided by its length, *L*,: $k = AE/L$.

The program that you need to develop should be able to compute the axial deformation of the element assemblage. In addition, it should check the strength of each cable to ensure its safety against breaking.

In order to simplify the problem, you can assume that the weight of the cables is negligible in comparison to the machinery being lowered and that each cable behaves perfectly elastically until failure.

You should use a class named *Cable* with 3 doubles as private member data: the area, the modulus of elasticity, the length, and the strength of the cable. The class that you should use for a cable element should look as the one below. The member variables *area, elastic_modulus, length, and strength* **MUST** be private.

```
class Cable
{
public:
 // your functions here

private:
 double area;              // area of the cable
 double elastic_modulus;   // modulus of elasticity of the cable
 double length;            // length of the cable
 double strength;          // strength of the cable
};
```

A cable will fail if its stress exceeds its strength. Since each cable in a series assemblage must support the entire weight, the stress in each cable can be determined by dividing the supported weight by the area of the cable:

$$Stress = weight / area$$

If a cable does not fail, its elongation can be calculated using the following formula derived from Hooke's Law:

$$Elongation = \left(\frac{weight \cdot length}{area \cdot modulus}\right) = \frac{weight}{k}$$

The total elongation of the system can be determined by adding the elongation calculated for each cable.

Alternatively, the total elongation of the system can be determined using the following formula:

$$Elongation = \left(\frac{weight \cdot length}{area \cdot modulus}\right) = \frac{weight}{k_{eq}}$$

The equivalent stiffness k_{eq} is given by the formula:

$$k_{eq} = \frac{1}{\frac{1}{k_1} + \frac{1}{k_2} + \ldots + \frac{1}{k_n}}$$

The data should be stored in an array of *Cables* , i.e. objects of the class *Cable*. The array should be dynamically allocated after the user specifies the number of cables. First have the user enter the number of cables in the assemblage and then read the data for each cable and the weight of the machinery. After reading all the information the data should be printed out using overloaded I/O operators.

Then, check whether any of the cable will fail under the provided weight. In case of exceeding its strength a warning message should be printed out and the program should exit.

If no cable will fail, loop over the array to determine the equivalent stiffness constant for the assemblage, which you should print out to the user. You also need to print the individual elongation and stress for each cable. You should print out all results with ***3 decimal point accuracy***.

You should provide your source code for the Cable class in the file *cable.C* and the declarations in the file *cable.h*, and the general code in the files *ps2_3.C* and *ps2_3.h*. You also need to write a proper makefile, named *makePS2,* that can be used to compile and link your program.

ps2_3.h

In this header file you should provide all the declarations of the non-member functions, i.e. all functions that are provided in the file *ps2_3.C*.

ps2_3.C

In this file you should provide the main function, which should be kept very short, e.g. less than 10 lines of code. In main you should first call a function to read the cables information and a function to read the weight of the machinery. The first information that should be provided in the former function should be the number of cables in order to allocate dynamically the array of cables. Then, the area, elastic modulus, length and strength of each cable. The input for each cable should be read using an overloaded input operator. The latter function should read the weight of the machinery supported by the cables. In both function you should check that the provided data are positive numbers. Having read all input data another function should be invoked to print out the data of each cable. The output operator should be used for this purpose.

Then, a function should be called to check if any cables will fail. If a cable will fail then this information should be provided to the user and the program should exit after it releases the dynamically allocated memory. If no cable will fail, the elongation and equivalent stiffness constant of the cable assemblage should be printed. The stress and elongation in each individual cable should then also be printed. Finally, the dynamically allocated memory should be released.

A sample main() function is presented below:

```
int main()
{
  Cable *cableAssemblage;
  int numberCables;
  double weight;

  numberCables = readCableData(&cableAssemblage);
  weight = readWeight();

  printCableData(cableAssemblage,numberCables,weight);

  if(checkStrength(cableAssemblage,numberCables,weight))
    determineExtensions(cableAssemblage,numberCables,weight);

  releaseMemory(cableAssemblage);
  return EXIT_SUCCESS;
}
```

In the file *ps2_3.C* you should also provide the functions to implement the above functionalities, i.e. check for failure, determine the elongation of the assemblage, and the elongation and stress of each individual cable.

cable.h

In this header file you should provide the definition of the class *Cable*. All externally defined member function should be provided in the file *cable.C*.

cable.C

All externally defined member functions as well as the overloaded input and output operators should be defined in this file. You may need to provide functions to calculate the stiffness constant (k), stress and elongation of a single cable, and a function to check if a cable has failed under a given load.

Two sample runs of a program that performs the above functionalities are provided below. In the first one no cable fails so the elongations and stresses are provided as indicated above.

Enter the number of cables in the assemblage: 2

Enter the data for cable 1
Area: A = 0.0025
Modulus of elasticity: E = 200000000
Length: L = 5
Strength: S = 300000

Enter the data for cable 2
Area: A = 0.002
Modulus of elasticity: E = 210000000
Length: L = 6.5
Strength: S = 325000

Enter the weight of the machinery:
500

Cable 1
Area: A = 0.0025
Modulus of elasticity: E = 2e+08
Length: L = 5
Strength: S = 300000

Cable 2
Area: A = 0.002
Modulus of elasticity: E = 2.1e+08
Length: L = 6.5

Strength: S = 325000

Equivalent stiffness constant: Keq = 39252.3

The assemblage will extend 0.0127 units beyond its original length.

Cable 1: stress=2e+05 Elongation=0.005

Cable 2: stress=2.5e+05 Elongation=0.00774

In the second case the stress exceeds the strength of the cable, and, therefore, the program exits after printing an appropriate message.

Enter the number of cables in the assemblage: 1

Enter the data for cable 1
Area: A = 0.002
Modulus of elasticity: E = 200000000
Length: L = 7
Strength: S = 250000

Enter the weight of the machinery:
745

Cable 1
Area: A = 0.002
Modulus of elasticity: E = 2e+08
Length: L = 7
Strength: S = 250000

This assemblage cannot support the machinery.

Cable 1 will fail!!!

Submission requirements:

You should provide your source code in the files ***cable.C*** and ***cable.h*** *(*for the Cable class as explained above), and ***ps2_3.C*** and ***ps2_3.h*** (the general code). In addition you **need** to submit a proper makefile, named ***makePS2***, that can be used to compile and link your program with the following command:

gmake -f makePS2 ps2_3

Note:

Please submit **both** printouts of the source code you have written (preferably using *% enscript -2Gr -Pprinter filename*) and screen dumps of the execution output (using *%xdpr -Pprinter*), with your name and username clearly written on the first page of the **stapled** submitted problem set. The submitted code must be identical to that electronically turned in (as described above).

If you can submit the problem set one day late 10% of the total grade (i.e. 10 points) will be deducted. More than one day late problem sets will **not** be accepted.
