1.124J Foundations of Software Engineering

Problem Set 2 - Solution

Due Date: Tuesday 9/26/00

Problem 1:[10%]

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

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

b. *void printMyF(double)*

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

a. they are assumed to be private.

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.
d. Pointers of different types may not be assigned to one another without a cast operation.

4. Which of the following is/are True?

e. None of the above

5. Which of the following is/are True?

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

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

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

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

Problem 2:[30%]

sol2_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
```

sol2_2.C

```
 // Problem Set#2 - Problem#2 solution  [sol2_2.C]

#include "sol2_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()
{
  cout << "\n Using the default constructor \n";
  position = 0 ;
  totalWeight = 0.0;
}



ElevatorStack::~ElevatorStack()       //  Destructor
{
  cout << "\nReleasing the memory for the array of structures \n";
  for(int i=0 ; i<position;i++)
    delete []  guards[i].name;
}


void ElevatorStack::push(char *name, double weight)
{
```

```
  if(position >= MAX_PERSONS && (totalWeight+weight)>MAX_WEIGHT)
    cout << "\n  Guard " << name << " cannot enter the elevator \n"
  << "to avoid exceeding of both allowable weight and "
  << "number of persons";

  else if(position >= MAX_PERSONS)
    cout << "\n  Guard " << name << " cannot enter the elevator \n"
  << "to avoid exceeding the allowable number of persons";

  else if((totalWeight+weight)>MAX_WEIGHT)
    cout << "\n Guard " << name << " cannot enter the elevator"
  << "to avoid exceeding the maximum weight";
  else
    {
      cout << "\n - Pushing a guard into the elevator " << endl;
      guards[position].name = new char[strlen(name)+1] ;
      strcpy(guards[position].name,name) ;
      guards[position].weight = weight ;
      position++ ;
      totalWeight += weight;
      cout << "  There are " << position
    << " guard in the elevator " ;
      cout  << setiosflags(ios::fixed) << setprecision(1)
     << " with total weight of " << totalWeight
     << " pounds"<< endl ;
    }
}



void  ElevatorStack::pop(void)
{
  if(position==0)
    cout << "\n Stack is empty \n";
  else
    {
      cout << "\n - Poping a guard from the elevator" << endl ;

      position--;
      totalWeight -= guards[position].weight ;
      delete []  guards[position].name;

      cout << "  There are " << position << " guards in the elevator" ;
```

```
        cout << setiosflags(ios::fixed) << setprecision(1)
       << " with total weight of " << totalWeight
       << " pounds"<< endl ;
       }
    }
```

```
/********************** Solution output  ************************
 sol2_2 < dat2_2
---------------
 Using the default constructor

 Guard's name :
 Weight :
 - Pushing a guard into the elevator
  There are 1 guard in the elevator  with total weight of 180.5 pounds

 Guard's name :
 Weight :
 - Pushing a guard into the elevator
  There are 2 guard in the elevator  with total weight of 346.1 pounds

 Guard's name :
 Weight :
 - Pushing a guard into the elevator
  There are 3 guard in the elevator  with total weight of 553.1 pounds

 Guard's name :
 - Poping a guard from the elevator
  There are 2 guards in the elevator with total weight of 346.1 pounds

 Guard's name :
 Weight :
 - Pushing a guard into the elevator
  There are 3 guard in the elevator  with total weight of 524.1 pounds

 Guard's name :
 Weight :
 - Pushing a guard into the elevator
  There are 4 guard in the elevator  with total weight of 719.6 pounds
```

Guard's name :
Weight :
- Pushing a guard into the elevator
There are 5 guard in the elevator with total weight of 904.6 pounds

Guard's name :
Weight :
Guard Olivia cannot enter the elevator
to avoid exceeding the allowable number of persons

Guard's name :
- Poping a guard from the elevator
There are 4 guards in the elevator with total weight of 719.6 pounds

Guard's name :
- Poping a guard from the elevator
There are 3 guards in the elevator with total weight of 524.1 pounds

Guard's name :
- Poping a guard from the elevator
There are 2 guards in the elevator with total weight of 346.1 pounds

Guard's name :
- Poping a guard from the elevator
There are 1 guards in the elevator with total weight of 180.5 pounds

Guard's name :
- Poping a guard from the elevator
There are 0 guards in the elevator with total weight of 0.0 pounds

Guard's name :
Stack is empty

Guard's name :
Weight :
- Pushing a guard into the elevator
There are 1 guard in the elevator with total weight of 246.0 pounds

Guard's name :
Weight :
- Pushing a guard into the elevator
There are 2 guard in the elevator with total weight of 481.4 pounds

Guard's name :
Weight :
- Pushing a guard into the elevator
There are 3 guard in the elevator with total weight of 715.4 pounds

Guard's name :
Weight :
- Pushing a guard into the elevator
There are 4 guard in the elevator with total weight of 960.4 pounds

Guard's name :
Weight :
Guard Paul cannot enter the elevatorto avoid exceeding the maximum weight

Guard's name :
- Poping a guard from the elevator
There are 3 guards in the elevator with total weight of 715.4 pounds

Guard's name :
- Poping a guard from the elevator
There are 2 guards in the elevator with total weight of 481.4 pounds

Guard's name :
- Poping a guard from the elevator
There are 1 guards in the elevator with total weight of 246.0 pounds

Guard's name :
Weight :
- Pushing a guard into the elevator
There are 2 guard in the elevator with total weight of 413.4 pounds

Guard's name :
Weight :
- Pushing a guard into the elevator
There are 3 guard in the elevator with total weight of 586.4 pounds

Guard's name :
Weight :
- Pushing a guard into the elevator
There are 4 guard in the elevator with total weight of 761.9 pounds

Guard's name :

Weight :
- Pushing a guard into the elevator
There are 5 guard in the elevator with total weight of 956.9 pounds

Guard's name :
Weight :
Guard Bob cannot enter the elevator
to avoid exceeding of both allowable weight and number of persons

Guard's name :
- Poping a guard from the elevator
There are 4 guards in the elevator with total weight of 761.9 pounds

Guard's name :
- Poping a guard from the elevator
There are 3 guards in the elevator with total weight of 586.4 pounds

Guard's name :

Exiting the program normally

Releasing the memory for the array of structures

**********/

Problem 3:[70%]

Makefile

```
#!gmake
#============================================
#
#          Makefile for Problem Set # 2
#!gmake
#
# To use this makefile:    % gmake -f makeSol2 program_name
#
#      Fall - 2000
```

```
#
#============================================================

#     V a r i a b l e     D e f i n i t i o n s
#     ----------------------------------------
MACHINE_TYPE = `/bin/athena/machtype`
CXX = g++

CXXINCLUDE = -I.
CXXFLAGS = -g -ansi -pedantic -Wall
LDLIBS =  -lm

SRC  =  sol2_2.C sol2_3.C cable.C
PROG =  sol2_2 sol2_3
OBJ =  $(SRC:%.C=%.o)

#          E x p l i c i t       R u l e s
#          ------------------------------
#------------------------------------------------------------------
all: ${PROG}
.PHONY: all
${PROG}: makeSol2
${OBJ}: makeSol2
#------------------------------------------------------------------
sol2_2: sol2_2.o
 @ echo "     Linking  to create $@"
 $(CXX)  sol2_2.o  -o sol2_2  ${LDLIBS}

sol2_2.o:sol2_2.C sol223.h
 @ echo "     Compiling $< to create $@ "
 $(CXX) ${CXXFLAGS}  -c sol2_2.C
#------------------------------------------------------------------
sol2_3: sol2_3.o cable.o
 @ echo "     Linking  to create $@"
 $(CXX) sol2_3.o cable.o -o sol2_3  ${LDLIBS}

sol2_3.o:sol2_3.C sol2_3.h
 @ echo "     Compiling $< to create $@ "
 $(CXX) ${CXXFLAGS}  -c sol2_3.C

cable.o:cable.C cable.h
 @ echo "     Compiling $< to create $@ "
```

```
 $(CXX) ${CXXFLAGS}  -c cable.C
#-------------------------------------------------------------
.PHONY: clean clean_o clean_p
clean:
 @echo "     Cleaning all ~, executable and object files"
 -rm -f $(PROG) *.o a.out *.*~ *~
clean_o:
 @echo "     Cleaning all object files"
 -rm  -f *.o
clean_p:
 @echo "     Cleaning all executables"
 -rm -f $(PROG)

#-----------------------------------------------------------

#          I m p l i c i t       R u l e s
#          ------------------------------
#-------------------------------------------------------------
%: %.o
 @ echo "     Linking  to create $@"
 $(CXX) $<  -o   $@ ${LDLIBS}
%.o:%.C
 @ echo "     Compiling $< to create $@ "
 $(CXX) ${CXXFLAGS}  -c $< -o $@
#-----------------------------------------------------------
```

sol2_3.h

```
// Problem Set#2 - Problem#3 [sol2_3.h]

#ifndef SOL2_3_H
#define SOL2_3_H


#include "cable.h"


int main();

int readCableData( Cable **c);

void printCableData(Cable *cableAssemblage, int numberCables, double weight);
```

```
double readWeight();

bool checkStrength(Cable *cableAssemblage, int numberCables, double weight);

void determineExtensions(Cable *cableAssemblage, int numberCables, double weight);

void releaseMemory(Cable *cableAssemblag);


#endif
```

sol2_3.C

```
// Problem Set#2 - Problem#3 [sol2_3.C]

#include "sol2_3.h"
#include <iostream.h>
#include <stdlib.h>
#include <iomanip.h>


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;
}


int readCableData( Cable **c)
```

```
{
  int n,i ;

  cout << "Enter the number of cables in the assemblage: ";
  cin >> n;

  *c = new Cable[n];

  for (i=0; i<n; i++)
    {
      cout<< "\nEnter the data for cable " << i+1;
      cin >> *(*c+i);
    }

  return n;
}


void printCableData(Cable *cableAssemblage, int numberCables, double weight)
{
  for (int i=0; i<numberCables; i++)
    cout << "\nCable " << i+1 << cableAssemblage[i];
}



double readWeight()
{
  double weight;

  do
    {
      cout << "\nEnter the weight of the machinery: " << endl;
      cin >> weight;

      if (weight < 0)
        cout << "Weight must be greater than zero.  Try again"<< endl<<endl;

    }while (weight < 0);

  return weight;
}
```

```
bool checkStrength(Cable *cableAssemblage, int numberCables, double weight)
{
  for (int i=0; i<numberCables; i++)
    {
      if(cableAssemblage[i].fail(weight))
 {
   cout << "\n This assemblage cannot support the machinery." << endl;
   cout << "\n Cable " << i+1 << " will fail!!!" << endl;
   return false;
 }
    }
  return true;

}


void determineExtensions(Cable *cableAssemblage, int numberCables, double weight)
{
  double inversesSum=0.0, kEq, dl;

  for (int i=0; i<numberCables; i++)
    inversesSum += 1/cableAssemblage[i].kConstant();

  kEq = 1/inversesSum;
  cout << "\n Equivalent stiffness constant:  Keq = " << kEq << endl;

  dl = weight / kEq;

  cout << "\n\n The assemblage will extend "
       << setprecision(3) << dl
       << " units beyond its original length.\n" << endl;

  for(int i=0; i<numberCables; i++)
    {
      cout << "Cable " << i+1 << ":     stress="
    <<  setprecision(3)
    << cableAssemblage[i].stress(weight)
    << "        Elongation="  << setprecision(3)
    << cableAssemblage[i].elongation(weight)
    << endl << endl;
```

```
    }

}



void releaseMemory(Cable *cableAssemblage)
{
 delete []cableAssemblage;
}
```

cable.h

```
// Problem Set#2 - Problem#3 [cable.h]

#include <iostream>

#ifndef CABLE_H
#define CABLE_H

// Class definition

class Cable
{
 public:

  double getLength() { return length;}

  double kConstant();
  double stress(double force);
  double elongation(double force);
  bool fail(double force);

  friend istream& operator >> (istream &i, Cable &c);
  friend ostream& operator << (ostream &o, Cable &c);

 private:

  double area, elasticModulus, length, strength;

};
```

```
#endif
```

cable.C

```
// Problem Set#2 - Problem#3 [cable.C]
/************ Externally defined member functions  **************/


#include "cable.h"


// Formulas for physical quantities

double Cable :: kConstant()
{
  return area*elasticModulus/length;
}


double Cable :: stress(double force)
{
  return force/area;
}


double Cable :: elongation(double force)
{
  return force/kConstant();
}


bool Cable::fail(double force)
{
  if ( stress(force) > strength )
    return true;

  return false;
}

/****************   Friend functions  ************************/
```

```
istream& operator >> (istream &i, Cable &c)
{
  do
    {
      cout << "\n  Area:  A = " ;
      i >> c.area ;

      cout << "  Modulus of elasticity:  E = " ;
      i >> c.elasticModulus ;

      cout << "  Length:  L = " ;
      i >> c.length ;

      cout << "  Strength:  S = " ;
      i >> c.strength ;
    }while(c.area<=0 || c.elasticModulus<=0 || c.length<=0 || c.strength<0);

  return i;
}

ostream& operator << (ostream &o, Cable &c)
{
  cout << "\n Area:  A = " ;
  o << c.area ;

  cout << "\n Modulus of elasticity:  E = " ;
  o << c.elasticModulus ;

  cout << "\n Length:  L = " ;
  o << c.length ;

  cout << "\n Strength:  S = " ;
  o << c.strength << endl;

  return o;
}
```