Exercises 1

Part 1:

Explain the output of the following code without running it:

```
struct data
{
      float z;
      char type;
};

int main()
{
   data D1 = {20, 'P'};
   cout << D1.z++ << D1.type << endl;
   data D2;D2.type = D1.type;
   D2.z = 4 * D1.z;
   Cout << ++D2.z << D2.type << endl;
   Cout << D1.z--<<D1.type;
   return 0;
}
```

Part 2:

Debug the following program. Include the entire fully functional program as your answer.

```
class member
{       int memberNum = 25;
        float memberPay;
        public
        void Input(cin >> memberNum >> memberPay);
        void Output;
}

int main()
{
     Member mem;
     cin >> mem.memberNum >> mem.memberPay;
     cout << mem.memberNum << '\t' << mem. memberPay;
}

void Output::Member()
  {cout << mem.memberNum << '\t' << mem.memberPay;}
```

Part 3:

1. Define a class Account (to indicate bank accounts) with the followingspecifications:
   a. Data members in private visibility mode (remember that the default visibility mode is private):
   i. Account number (long integer)
   ii. Name of the depositor (string object)
   iii. Type of account (character type value which stores either ‘S’ for saving and ‘C’ for checking)
   iv. Balance Amount (double)

   b. Member functions in public:
   v. To input the value of all data members based on values inputted by the user.
   vi. To deposit money (the function should take an argument called amountToDeposit, increment the current balance amount by this value, and return the new balance)
   vii. To withdraw money from the account. It should do what the previous case did with a similar argument named amountToWithdraw. However, before doing the withdrawal, it should make sure that the balance after withdrawing will not drop below $500. It should return the amount actually withdrawn (0 if no money was withdrawn).
   viii. To return the Balance Amount
   ix. To print all the data members.

2. Write a function which takes two Accountobjects as arguments, and sets the data members of these objects based on values inputted by the user. The object in the first argument should retain these values after the function is called, but the object in the second argument should not. Remember that since this function will not be a member of the class, it only has publicaccess to the class members.

3. Write another function which again takes two objects as arguments, compares their balance amount (using the function you defined in question 1b.viii), and returns that object whose balance amount is greater. Define the function such that modifying the Account object returned will modify the original object passed as an argument.

Part 4:

Classes can also have arrays data members inside them. For instance, if we have an array A of size 3 as a data member of the class Employee we discussed about in lecture, an object of the class Emp1 will have 3 sub-variables, namely Emp1.A[0], Emp2.A[1], Emp3.A[2], similar to the regular sub-variables Emp1.Id, Emp1.Name etc.

Define a class SchoolStudent with the following specifications. You may assume every student has 8 grades.

1. Data members in private visibility mode-
    a. studentID, of type long integer
    b. studentName, of type string
    c. average, of type float
    d. grades, an array of type double

2. Member function in private visibility mode. (If a member function is defined in private visibility mode, then it can only be accessed by other member functions inside the class but not by the objects of the class).
    a. CalculateAvg() that returns the average of the students' grades.
3. Member functions in public:
    a. Assign() that asks the user to input the ID, Name and Grades and sets the average equal to the value returned after calling the Calculate() function.
    b. Output() that outputs all of the data on to the screen

Part 5:

An array of class objects works the same way as a normal array. For example, if we declare an array of objects of the class Employee discussed in lecture: Employee Emp[10];Each element of the array will be an object, i.e Emp[0], Emp[1], and Emp[2] are all Employee objects (or *instances* of the Employee class). If we wish to call the Input()function for object Emp[1], the syntax will be just as in the case of a normal object: Emp[1].Input();.

In classes, data members are usually made private. If you want to allow outside users of the class to see or modify these values, the class usually should make sure this happens on the class's own terms – it doesn't want any other code making its internal data inconsistent (say, a SchoolStudent's average being set to a number not reflective of the grades). To selectively allow access, classes often have "getter" and "setter" functions.For example, to allow users of the Employee class to retrieve and modify the employee Id, we might define:

```
class Employee  {  …
                   int getId()
                   { return Id; }
                   void setId(int newId)
                   { Id = newId; }
                     …   };
```

The names of these functions are usually of the form getSomeVariable or setSomeVariable.

In the class defined in the previous question, add a getter function in public visibility mode to return the percentage of a student. Write a program to declare an array of

objects of class SchoolStudentand perform the following using separate global functions (i.e. non-member functions) for each. (Use dynamic memory allocation to allow the user to decide how many students are in the array.)

1. Assigning values to all the elements of the array (Calling the Assign() function for each of the objects)
2. Sorting the array of class objects on the basis of average using bubble sort method (you will have to call your average getter in the if-statement of the sort)
3. Displaying the array elements (You will have to call the Output() function for each object)

Part 6:

1. Write a Rectangle class that has as data members two arrays of doubles, one for x coordinates and the other for y coordinates. The class should have an init() function that inputs these coordinates from the user. (We will see a more elegant way to initialize in the next lecture.)
2. Write a RectDrawing class that has as a data member an array of Rectangles. Create an init() function that allocates a new array of Rectangles of a size entered by the user, and allows the user to initialize each Rectangle.
3. What problem will occur if a RectDrawing is initialized more than once? Assuming you had some well-implemented hasBeenInitialized() member function, and you were adding the code below to the init function of RectDrawing, what would you have to put in the blank line below to solve this problem?

```
if ( hasBeenInitialized() )
{// Insert your code here}
```

4. Show, in one line, how you would allocate a new RectDrawing class using the new operator (assuming you wanted to be able to refer to the new object later

Part 1

1. The program first defines a structure `data` containing two variables `z` (of type float), and `type` (of datatype char).
2. In the main function, a structure type variable of the structure data is declared, and then initialized (D1.z is initialized to 20, and D1.type is initialized to P)
3. D1.z++ increments D1.z to 21, but since the increment is done by the post-increment operator, the cout prints 20, followed by 'P' (value stored in D1.type), on the screen.
4. A new structure type variable of structure `data` is declared. The value of D2.type is initialized to the value stored in D1.type (i.e P). The value of D2.z is initialized to 4 times the value stored in D1.z(i.e 21*4=84).
5. ++D2.z increments the value of D2.z by 1 (i.e to 85), and as the increment is done by the pre-increment operator, cout print 85 is on the screen, followed by 'P' (value stored in D2.type).
6. Finally, in the next line, D1.z is decremented by 1, but due to post-increment, its previous value (i.e 20) is printed to the screen by the cout, followed by printing of the character 'P' (D1.type).

Output: 20P
85P
21P

Part 2

```
class member
{
  int memberNum;
  float memberPay;
public:
  void Input(){cin >> memberNum >> memberPay};
  void Output();
};

int main()
{
  Member mem;
  mem.Input();
  mem.Output();

  return 0;
}

void Member::Output()
{
  cout << memberNum << '\t' << memberPay;
}
```

Part 3

```
class Account
{
```

```
  long AccNum;
  string Depositor;
  char AccType;
  double Amount;
  public:
  void Input();
  double Deposit( double amountToDeposit);
  double Withdraw (double amountToWithdraw);
  double retBalance();
  void Output();
}
void Account::Input()
{
  cin >> AccNum >> Depositor >> AccType >> Amount;
}
void Account::Output()
{
  cout << AccNum << ":" << Depositor << ":"
       << AccType << ":" << Amount << endl;
}
double Account::Deposit (double amountToDeposit)
{
  Balance += amountToDeposit;
  return Balance;
}
double Account::Withdraw (double amountToWithdraw)
{
  if(Balance-amountToWithdraw >= 500)
  {
    Balance -= amountToWithdraw;
    return Balance;
  }
  else
  {
    cerr << "Not allowed to withdraw" << endl;
    return 0;
  }
}
double Account::retBalance()
{
  return Balance;
}

void Input(Account &Acc1, Account Acc2)
{
  Acc1.Input();
  Acc2.Input();
}
Account CompareBal(Account &Acc1, Account &Acc2)
{
  if(Acc1.retBalance()>Acc2.retBalance())
    return Acc1;
  else
    return Acc2;
}
```

Part 4:

```
class SchoolStudent
{
private:    // Optional
  long studentID;
  string studentName;
  float average;
  double grades[8];
  float CalculateAvg();
public:
  void Assign();
  void Output();
};
void SchoolStudent::Assign()
{
  cin >> studentID >> studentName;
  for(int i=0;i<8;i++)
    cin >> grades[i];
  average = CalculateAvg();
}
float SchoolStudent::CalculateAvg()
{
  float sum=0;
  for(int i =0; i < 8; i++)
    sum += grades[i];
  return sum/8.0;
}
void SchoolStudent::Output()
{
  cout << studentID << ":" << studentName << ":"
       << average << ":";
  for(int i = 0; i < 8; i++)
    cout << grades[i];
}
```

Part 5:

This question uses the same class as in the previous question, with an additional member function in public, i.e. `GetAvg()`.

```
float SchoolStudent::GetAvg()
{
  return average;
}

void Assign( SchoolStudent *Student, int n)
{
  for(int i=0;i<n;i++)
  Student[i].Assign();
}

void Sort (SchoolStudent *Student, int n)
{
```

```
 for(int i=0;i<5;i++)
 {
   for( int j=0;j<i;j++)
   {
     if(Student[i].GetAvg() > Student[j].GetAvg())
     {
       Student temp=Student[i]; //swap
       Student[i]=Student[j];
       Student[j]=temp;
     }
    }
  }
}

void Display(SchoolStudent *Student, int n)
{
  for(int i = 0; i < n; i++)
  Student[i].Output();
}

int main()
{
  int n;
  cout << "Enter the number of students:";
  cin >> n;
  SchoolStudent *Student1 = new SchoolStudent[n];
  Assign(Student1, n);
  Sort(Student1, n);
  Display(Student1, n);
}
```

Part 6:

```
class Rectangle
{
  double x[4], y[4];
public:
  void init();
};

Rectangle::init()
{
 for (int i = 0; i < 4; i++)
 cin >> x[i] >> y[i];
}

class RectDrawing
{
 Rectangle *ArrRect;
 public:
 void init();
};

void Rectdrawing::init()
{
```

```
 int n;
 cin >> n;
 ArrRect = new Rectangle[n];
}
```

If we keep initializing without deleting, then each previous array will be lost, but will continue to sit in memory. More and more memory will keep being allocated to the arrays, and eventually the program will run out of memory.

```
void RectDrawing::init()
{
 if( hasBeenInitialized() )
   delete[] ArrRect;
 int n;
 cin >> n;
 Rectangle *ArrRect= new Rectangle[n];
}
```

```
RectDrawing *Arr=new RectDrawing;
```