Do not award half marks.
In all cases give credit for appropriate alternative answers.

Question 1 (Compulsory)

(a) Briefly explain **one purpose** of each of the following terms used in the Object-oriented C++ programming language context. [4]

class

- **contains the related data and operations that act on the data together**

friend function

- **to access private data in a class which is a non member function**

static member function

- **for accessing static data member of a class**

virtual functions

- **allow derived classes to replace the implementation provided by the base class.**

Award 1 mark for correct explanation.
Accept alternative correct explanation. [4 marks]

(b) Name ***one feature*** of Object-oriented programming that promotes code reuse. Briefly explain how the feature supports code reuse as one of the benefits of Object-oriented programming. [2]

Inheritance [1]
New classes can be derived altering or adding new properties to an existing one. [1]

Accept alternative correct explanation.

(c) How do ***message*** and ***method*** in Object-oriented programming relate to each other to your understanding? [2]

Message is a request to invoke a method in a class through an object. [1]
Method is an operation in the class to act on the data member. [1]

Award 1 mark for correct explanation.
Accept alternative correct explanation.

(d) Identify **one similarity** and **one difference** between constructor and copy constructor in their purposes. [3]

Constructor and copy constructor are for initialization [1]
Constructor is for initializing data contained in object when created.[1]
Copy constructor is to make a copy of existing object. [1]

Award 1 mark for correct explanation.
Accept alternative correct explanation.

(e) Create a class named Job that holds the following members:

- an array of 20 characters, **jobId**
- a float pointer, **cost**

Both member variables are not made accessible to any other class. [3]

```
class Job {    [1]
               char jobId[20];                                    [1]
               float *cost;                                       [1]
          }
```

(f) Implement a constructor that takes two default parameters to initialize properly the member variables: **jobId,** which is a character pointer, and a float **c**. It uses the "new" operator to allocate memory storage for member variable cost. **jobId** should hold **"default",** cost is **0**. [3]

```
Job :: Job ( char *jobId = "default", float c = 0) [1]
          {    strcpy( this->jobId, jobId); [1]
               cost = new float( c ); [1] }
```

(g) Implement a destructor for the class **Job**. [2]

```
Job :: ~Job() [1] { delete cost; }                                [1]
```

(h) Implement a method **setCost** for Job class that takes in input a parameter float **amtReduce.** If cost is greater than amtReduce it reduces member variable cost by amtReduce and returns 1; otherwise it returns 0.

int Job :: setCost (float amtReduce) [3]

```
int Job :: setCost (float amtReduce)
        { if ( amtReduce < cost) [1]  { cost -= amtReduce;
                                        return 1; }
                        return 0; }
```

(i) Should the programmer need to implement a copy constructor for Job class to override the default copy constructor? Explain why. [2]

Yes [1]
One of the data members is a pointer [1]

(j) Create an array of 10 objects of type Job named **engrProject**. [1]

Job engrProject[10]; [1]

(k) Implement a recursive function named **aboveCost** – whose signature is given below – that takes an array called **project** of n objects of type Job and returns an integer that represents the number of projects that are equal to or greater than the tgtcost.

int aboveCost(int n, Job project[], float tgtcost) [5]

```
int aboveCost( int n, Job project[], float tgtcost)
        { int count ;
                if (n = = -1) return 0; [1]
                else  count = aboveCost(n-1, project, tgtcost) [2]
                                if ( project[n].getCost() >= tgtcost) [1]
                                        return count +1;
                                else
                                        return count;   [1]
                }
```

Question 2

(a) Briefly explain the term "**multiple inheritance**" in C++ object-oriented programming. [1]

A derived class inherits more than one base class. [1]

Accept alternative correct explanation.

(b) When a created class is made to inherit the data members and member functions of another class, what are the member functions that cannot be inherited? [3]

constructors [1]
destructors [1]
assignment operator = [1]

(c) Given the declaration of classes as follows.

```
class  Research {
                private:
                        char *projectTitle;
                        float hrs;
                public:
                void display( );
                int getResearchHrs( );
                research(float  h, char *pt);

        };

class Teaching {
                protected:
                        char unitTitle[20];
                        int hrs;
        // the default constructor by the C++ compiler is not overridden

        };
```

(i) Create a derived class named Lecturer that inherits the classes Research and Teaching in a protected way. [2]

```
class Lecturer : protect Research, Teaching {}
```

1 mark for correct multiple inheritance syntax
1 mark for protect

(ii) Implement a suitable constructor for the class Lecturer that takes appropriate parameters for initializing inherited data members. [4]

```
Lecturer :: Lecturer(char *pt, float hr, int ht, char uT[] ) [1] :
Research(hr, pt) [1]
 {      Teaching::hrs = ht; [1]
        strcpy(unitTitle, uT); [1]
 };
```

(iii) Implement a method getWorkHrs( ) for Lecturer class that returns a type float containing the total hours of both the base classes. [3]

```
float getWorkHrs()  [1] { return Teaching::hrs +
                              (float)  getResearchHrs( ); }
```

1 mark for choosing correctly the hours in both base classes.
1 mark for casting getResearchHrs

(iv) Implement a method named display( ) which has the same name and parameter declaration as the display method in Research class, and displays the data members in both classes. [2]

```
void Lecturer :: display( )
{  Research::display( ); [1]
   cout <<  “Unit Title”  << unitTitle << “Hours worked” <<
   Teaching::hrs << endl; [1] }
```

Question 3

(a) Encapsulation promotes data hiding. Briefly explain the term *encapsulation* and identify **one advantage** for hiding the data. [2]

Encapsulation is the process of hiding the details of an object that do not contribute to its (abstract) essential characteristics. [1] Smaller objects can be combined into a larger element that can be treated as a whole. [1]

Accept alternative correct explanation.

(b) Given the declaration of String class as below.

```
class String {
                char *buf;
                int len;
public:
                String(char *c="")
              { len = strlen(c);
                buf = new char[len+1];
                strcpy(buf, c);         }
};
```

(i) Implement an iterative method **countChar** that returns a type integer of the number of occurrences variable c in an object s of type String.

int countchar(String s, char c) [3]

```
int countChar( String s, char c)
{       int count = 0;
        for (int x=0; x<s.length(); x++)  [1]
                if (s[x] == c)   [1]
                        count ++;
                return count;        }  [1]
```

(ii) Implement a method for String to overload the assignment operator = to perform a *deep* copy. [4]

```
String &operator=(const String &s) [1]
{       if ( this == &s )  [1]
                return *this;
        delete [] buf;
        int len = s.length;
        buf = new char[len+1];
        strcpy(buf, s.buf);  [1]
        return *this;    [1]
}
```

(c) A class can be a friend to another class. Briefly explain what you have understood in the context of C++ object-oriented programming. [2]

All the functions in the friend class can access all the private elements of the other class. This is useful when objects of a class are managed by another class. [1]

Friendship status is one way. A class specifies a friendship relationship by placing the function prototype with the friend keyword. [1]

Accept alternative correct explanation.

(d) Given the declaration of classes as follows.

```
class Customer {
          public:
              char *custId; }

class CustAccount {
              Customer cust;
              double paymentMadeToDate;
              double updatePayment( CustTransaction c);     }

class CustTransaction {
              Customer cust;
              double amtPaid; }
```

Implement a method **updatePayment** for CustAccount class that is a non-member function of CustTransaction class to deduct paymentMadeToDate by amtPaid. It returns the updated payment. [4]

```
double CustAccount:: updatePayMent( CustTransaction c) [1]
    { if (strcmp(c.cust.custId,cust.custId)== 0 ) [1]
         paymentMadeToDate =paymentMadeToDate - c.amtPaid; [1]
              return paymentMadeToDate; [1]
              }
```

Question 4

(a) Constructors execute automatically in two phases:

1. Initialization
2. Assignment

Name two types of data member that require initialization list syntax other than reference data type. [2]

constant data type [1]
another class object that has a constructor and either the constructor requires parameters or we want to override the default values. [1]

(b) Given the declaration of customer class.

```
class Customer {
        private: char *name;
        };
```

(i) Create a class named CreditCard that has private two members are a reference object named **cust** of type customer and a static member **issuseno** that is initialized to 10000.

Include a constructor that takes an appropriate parameter for initializing the data member and increments member issueno by one for each new instance created. [4]

```
class CreditCard {
                Customer &cust;   [1]
                static int issueno = 10000;   [1]

        public: CreditCard(Customer &c) : cust( c ) {   [1]
                issueno++; }  [1]

        }
```

(ii) Implement a static member function getIssueNo that returns the static member issueno. *Note: The definition getIssueNo is part of the CreditCard class.* [2]

```
static int getIssueNo( )                                        [1]
        { return issueno;  }                                    [1]
```

(iii) Briefly explain why static member functions cannot call non-static member functions. [1]

static member functions have no 'this' pointer.

(iv) Assume the existence of method display( ) in **CreditCard** class, but the C++ compiler reports an error on the main( ) below. Briefly explain the error. [2]

```
void main()
            {  Customer cust("DBS");
            const CreditCard DBS(cust);
            DBS.display(); }
```

constant object DBS gives an error [1] when attempting to call non const member display( ). [1]

(v) Create a class named **CreditList** that has two private members: a **creditHolder** which is a pointer to CreditCard and an integer **creditNo**.

Include a constructor that takes an integer **s** to allocating exact memory **s** arrays for creaditHolder and **s** is assigned to creditNo. [4]

```
class CreditList {
        private:
              CreditCard *creditHolder; [1]
              int  creditNo; }
```

[1m for correct creditNo and class syntax]

```
        public: CreditList( int s)
        { creditHolder = new CreditCard[s]; [1]
                creditNo = s;                }
```

[1 for correct signature and proper s assigning]

Question 5

(a) Polymorphism literally means many forms. Briefly explain the relationship between parametric and polymorphism. [2]

Parametric means the class and/or functions are created without being specifying the type and they are later instantiated. [1] As such, the class and/or functions may appear in different types. [1]

Accept alternative correct explanation.

(b) Given the generic class as below.

```
template <class U, class T>
class Generic {
        U data;
        T key;
   public:
        Generic ( U data, T key);
        U getData();
        T getKey();
   }
```

(i) Create a class named GenericObj that has a private object objData of type Generic and define a public method **getData( )** that returns the data member of class Generic. [4]

```
template <class U, class T>                                   [1]
  class GeneriObj {                                           [1]
                    Generic<U, T> objData; [1]
                    public: U getData( ) [1] { return objdata.getData();} [1]
                    }
```

(ii) Implement a pure virtual method named setObj that takes in a parameter newData of type U. [2]

```
template<class U, class T>                                    [1]
virtual void setObj( U newData) = 0;                          [1]
```

(iii) Implement a method that overloads the operator + which takes in an object obj of type Generic and returns an object of type GenericObj which is the sum of the two objects. [2]

```
Generic<U, T> operator+ ( Generic<U,T> obj ) [1]
    { return  Generic ( obj.data + data, obj.key + key); } [1]
```

(iv) Implement a friend function that overloads the relational operator less than which takes in two arguments obj1 and obj2 of type Generic and returns an integer 1 if the member key of obj1 is less than obj2's key; 0 otherwise. You may define this friend function in the GenericObj class. [3]

```
friend int operator<(generic<U, T> obj1, generic<U, T> obj2 )    [1]
    { if (obj1.key < obj2.key) return 1                          [1]
      return 0;}                                                 [1]
```

(c) Does the C compiler report any error if an instance of type GenericObj is created? Why? [2]

Yes [1] because pure virtual functions cannot be instantiated [1]

- END OF PAPER -