Set-I

B.TECH. 4TH SEMESTER EXANIMATION, 2014

Answers to Short / Objective Questions

Q1.

i. B ii. B iii. C iv. D v. D vi. D vii. A viii. B ix. D x. A

B.TECH. 4TH SEMESTER EXANIMATION, 2014

Answers to Short / Objective Questions

B.Tech 4th Semester Examination, 2014

Model Answer

Subject:- Object Oriented Programming

Paper Code:- CS 1401

Sets (I) / (II)

Answer 2. a)

```
#include<iostream>
 using namespace std;
int recursiveLinearSearch(int array[],int key,int size){
 size=size-1;
  if(size <0){
  return -1;
  }
  else if(array[size]==key){
  return 1;
  }
  else{
  return recursiveLinearSearch(array,key,size);
  }
 }
 int main() {
 cout<<"Enter The Size Of Array:  ";
 int size;
 cin>>size;
 int array[size], key,i;
 // Taking Input In Array
 for(int j=0;j<size;j++){
 cout<<"Enter "<<j<<" Element : ";
 cin>>array[j];
```

B.Tech 4th Semester Examination, 2014
Model Answer

Subject:- **Paper Code:-**

Sets (I) / ~~(II)~~

```
}
//Your Entered Array Is
for(int a=0;a<size;a++){
  cout<<"array[ "<<a<<" ] = ";
  cout<<array[a]<<endl;
}
cout<<"Enter Key To Search  in Array";
cin>>key;
int result;
result=recursiveLinearSearch(array,key,size--);
if(result==1){
cout<<"Key Found in Array  ";
}
else{
cout<<"Key NOT Found in Array  ";
}
  return 0;
}
```

Answer 2 b.

```
#include<iostream>
using namespace std;
int main()
{
int armstrong=0,num=0,result=0,check;
```

B.Tech 4th Semester Examination, 2014
Model Answer

Subject:- Paper Code:-

Sets (I) / (II)

```
cout<<"Enter Number to find it is an Armstrong number?";
  cin>>num;
  check=num;
  for(int i=1;num!=0;i++){
    armstrong=num%10;
    num=num/10;
    armstrong=armstrong*armstrong*armstrong*armstrong;
    result=result+armstrong;
  }
  if(result==check){
  cout<<check<<" is an Armstrong Number";
  }
  else{
  cout<<check<<" is NOT an Armstrong Number";
  }
return 0;
}
```

Answer 3. a)

Since A and C have subclasses, it would make sense for them to have protected members (as protected members can be accessed directly in the class where they're created and in subclasses of that class). The protected members of A could be accessed in A, B, C, D, and E.

The protected members of C could be accessed in C, D, and E.

Answer 3. b)

B.Tech 4th Semester Examination, 2014
Model Answer

Subject:- **Paper Code:-**

Sets (I) / (II)

Any private members of C could only be accessed in C, as private members can only be accessed directly in the class which defines them.

Answer 3. c)

Since C has an abstract method, C must be abstract, and since C is the child of A and C is abstract, A must be abstract. The other classes would need to be concrete so that we could instantiate objects in this hierarchy.

Answer 3. d)

The first line makes an array of type A. It's possible to make an array of type A, even though A is abstract. (But it's not possible to put an A object in this array, because we can't instantiate an abstract class.) The second line is fine; it attempts to instantiate a concrete class, which is possible and desirable, and it attempts to put it an A container, which is possible since D is a kind of A. So, this code works just fine.

Answer 3. e)

The first line makes an array of type C. It's possible to make this array. The second line attempts to instantiate a concrete class, B, which is possible. However, it fails in its attempt to put a B in a C container, which is impossible because B is not a subclass of C.

Answer 4 a)

The problem is that the assignment in init(), namely x = y, uses the default copy assignment which simply copies each field from y into the corresponding field of x. This means that the address stored in y.a is simply copied into x.a, without the array being copied. At the end of init(), the destructor is called for the local variable y, which causes the array to be deleted. Thus x is left pointing to an array that is being reclaimed for other use by the program, and those locations in memory may be written to during the computation. Since p is passed to init() by reference, p is modified by init() so that p.a refers to the deleted array. To fix it, the copy assignment operator should redefined in A so that the elements of the array are copied, not the array pointer.

```
class A {
public:
...
A &operator=(const A &other)
{ for(i=0; i<3;i++) a[i] = other.a[i];
}
...
```

B.Tech 4th Semester Examination, 2014
Model Answer

Subject:- Paper Code:-

Sets (I) / (II)

}

Answer 4 b)

ADVANTAGES OF INHERITANCE

1) The most frequent use of inheritance is for deriving classes using existing classes, which provides reusability. The existing classes remain unchanged. By reusability, development time of software is reduced.
2) The derived classes extents the properties of base classes to generate more dominant object.
3) The same base classes can be used by number of derived classes in class hierarchy.
4) When a class is derived from more than one class, all the derived classes have the same properties as that of base classes.

DISADVANTAGES OF INHERITANCE

1) Though object oriented programming is frequently propagandized as an answer for complicated projects, inappropriate use of inheritance makes program more complicated.
2) Invoking member function using objects creates more compiler overheads.
3) In class hierarchy various data elements remains unused, the memory allocated to them is not utilized.

Answer 5 a)

It is a member function whose calls are dynamically dispatched. That is, given the declaration of the form

```
class C {
...
virtual .. f(..) ..
...
}
```

and a call of the form

```
... x.f(...) ...
```

where the variable x is declared to be of type C, the version of f that is executed depends on the actual type of the object that x refers to, not necessarily the declared type of x.

Rules for declaring virtual function

- The virtual functions should not be static and must be member of a class.
- A virtual function may be declared as friend for another class. Object pointer can access the virtual functions.

~~functions~~.

B.Tech 4th Semester Examination, 2014
Model Answer

Paper Code:-

Subject:-

Sets (I) / (II)

- Constructor cannot be declared as virtual, but destructor can be declared as virtual.
- The virtual function must be defined in public section of the class. It is also possible to define the virtual function outside the class. In such a case, the declaration is done inside the class and definition is outside the class. The virtual keyword is used in the declaration and not in function declarator.
- It is also possible to return a value from virtual function like other function.
- The prototype of virtual function in base class and derived class should be exactly same. In case miss
- match, the compiler neglects the virtual function mechanism and treats them as overloaded functions.
- Arithmetic operation cannot be used with base class pointer.
- If a base class contains virtual function and if the same function is not re-defined in the derived classes. In such a case, the base class function is invoked.
- The operator keyword used for operator overloading also supports virtual mechanism.

Answer 5 b)

If you leave the keyword virtual off , then no dynamic dispatching of the member function occurs. In the example above, no matter what the actual type of the object that x refers to is, the version of f that is called is the one defined in the class C.

Answer 5 c)

```
class A {
public: void f() { cout << "A's f\n"; }
virtual void g() { cout << "A's g\n"; }
}
class B: public A {
public: void f() { cout << "B's f\n"; }
virtual void g() { cout << "B's g\n"; }
}
void h(A &x)
{ x.f();
x.g();
}
int main()
{ B b;
```

```
h(a);
}
would print
A's f
B's g
```

Answer 6 a)

```
class A {
public:
int operator==(const A &other) { return 1; }
};
```

Answer 6 b)

In a child class, in order to override a method from the parent class, the signature of the method (i.e. parameter and return types) must be identical in the child and the parent classes. In this case, B's operator== is defined as taking a B parameter, but A's operator== is defined as taking an A parameter. Thus, this is not an overriding and so the occurrence of == in procedure f() is always A's operator==.

Answer 6 c)

This can be accomplished by writing f() as a function template, as follows:

```
template<class T>
void f(T &p, T &q)
{
if (p==q)
cout << "yes\n";
else
cout << "No\n";
}
```

Answer 7 a)

```
template<class T>
T sum(T &a, T &b, T &c)
{ return a+b+c;
}
```

B.Tech 4th Semester Examination, 2014
Model Answer

Paper Code:-

Subject:-

Sets (I) / (II)

Note the importance of the reference parameters, otherwise dynamic dispatching will not occur.

Answer 7b)

```
template<class T>
class myobj {
public:
        myobj() { cin >> val; } //not really necessary
        myobj operator+(myobj &m) {
        myobj r(*this);
        r.val = r.val + m.val;
        return r;
        }
protected:
        T val;
}
```

Answer 8 a)

```
#include<iostream.h>
        class A
        {
        protected :
        int i;
        public:
        A()
        {
        cout<<" Constructing default A class"<<endl;
        }
        A(int a)
        {
        i=a;
        cout<<" Constructing A class "<<endl;
        }
        ~A()
        {
        cout<<" Destructing A class"<<endl;
        }
        };
```

B.Tech 4th Semester Examination, 2014
Model Answer

Subject:- Paper Code:-

Sets (I) / (II)

```
class B
{
protected :
int j;
public:
B()
{
cout<<" Constructing default B class "<<endl;
}
B(int b)
{
j=b;
8
cout<<" Constructing B class "<<endl;
}
~B()
{
cout<<" Distructing B class"<<endl;
}
};
class C:public A,public B
{
protected:
int k;
public:
C()
{
cout<<" Constructing default derive C class"<<endl;
}
C(int a,int b,int c):A(a),B(b)
{
k=c; // b=i;
cout<<" Constructing C class"<<endl;
}
~C()
{
cout<<"\n Destructing C class"<<endl;
}
```

B.Tech 4th Semester Examination, 2014
Model Answer

Subject:- **Paper Code:-**

Sets (I) / (II)

```
void show()
{
cout<<"\n A i = "<<i<<endl;
cout<<" B j = "<<j<<endl;
cout<<" C k = "<<k<<endl;
}
};
void main()
{
C d(10,20,30);
d.show();
}
```

```
Constructing A class
Constructing B class
Constructing C class
A i = 10
B j = 20
C k = 30
Destructing C class
Destructing B class
Destructing A class
```

Explanation In the above program classes A, B, and C are defined. The class C is derived from classes A and B. In the function main () the object of class C is declared. As soon as the object is declared, the constructor of derived class and base classes are executed as shown in the output.

Answer 8 b)

Void pointer

Pointers can also be declared as void type. Void pointers cannot be dereferenced without explicit type conversion. This is because being void the compiler cannot determine the size of the object that the pointer points to.

Wild pointer

Pointers are used to store memory addresses. An improper use of pointer creates many errors in the program. Hence, pointer should be handled cautiously. When pointer points to an unallocated memory location or to data value, whose memory is de-allocated such a pointer is called as wild pointer. The wild pointer generates garbage memory location and pendent reference.

***this* pointer**

B.Tech 4th Semester Examination, 2014
Model Answer

Subject:- Paper Code:-

Sets (I) / (II)

The objects are used are used to invoke the non-static member function of the class. For xample, if ***p*** is an object of class ***P*** and ***get ()*** is a member function of ***P***, the statement ***p.get ()*** is used to call the function. The statement p.get *()* operates on ***p***. In the same way if ptr is a pointer to a ***P*** object, the function called ***ptr->get ()*** operates on ****ptr***. However, question is, how does the member function ***get ()*** understand which ***p*** it is functioning on? C++ compiler provides ***get ()*** with a pointer to ***p*** called ***this***. The pointer ***this*** is transferred as an unseen parameter in all calls to non-static member functions. The keyword ***this*** is a local variable always presents in the body of any non-static member function.

Answer 9 a)

The C++ provides a mechanism called ***inline*** function. When a function is declared as ***inline***, the compiler copies the code of the function in the calling function i.e. function body is inserted in place of function call during compilation. Passing of control between caller and callee functions is avoided. If the function is very large, in such a case inline function is not used because the compiler copies the contents in the called function that reduces the program execution speed. The ***inline*** function is mostly useful when calling function is small. It is advisable to use the inline function for only small functions. Inline mechanism increases execution performance in terms of speed. The overhead of repetitive function calls and returning values are removed. On the other hand, the program using inline functions needs more memory space. Since the inline function are copied at every point where the function is invoked.

Following are few situations where inline function may not work.

1) The function should not be recursive.
2) Function should not contain static variables.
3) Function containing control structure statements such as switch, if, for loop etc.
4) The function main () cannot be work as inline.

The inline functions are similar to macros of C. The main limitation of with macros is that they are not functions and errors are not checked at the time of compilation. The function offers better type testing and do not contain side effect as present in macros.

Answer 9 b)

An *lvalue* is an object locator. An expression points an object. An example of an *lvalue* expression is ****k*** that resulting to a non-null pointer. A changeable *lvalue* is an identifier or expression that relates to an object that can be accessed and suitably modified in computer memory. A ***const pointer to a constant*** is not a changeable *lvalue*. A pointer to a constant can be altered (its dereferenced value not). An *lvalue* could suitably stand on the left (the assignment side) of an assignment statement. Now, only changeable *lvalue* can legally stand on the left of an assignment

B.Tech 4th Semester Examination, 2014
Model Answer

Subject:- **Paper Code:-**

Sets (I) / (II)

statement. For example, Suppose *x* and *y* are nonconstant integer identifiers with appropriate allocated memory. Their lvalue are changeable. The following expressions are legal.

X = 1;

Y = x + y are legal expressions.

Rvalues (Right values)

The statement x + y is not a lvalue, x + y = z is invalid because the expression on the left is not related to a variable. Such expressions are often called rvalues.