Section: B **Quiz 13** Roll number: BSSE-23008

Q1 Word polymorphism means having different shapes

Q2 If derived class defines same function as defined in its base class, it is known as

a) *Function overloading* b) *friend function*
c) *dynamic polymorphism* d) *none of above*

Q3 What is the key difference between function overloading and function overriding in C++ polymorphism?

(a) Overloading involves functions with the same name but different parameter lists, while overriding involves functions with the same name and parameter lists in a base and derived class.
(b) Overloading is static binding (resolved at compile time), while overriding uses runtime binding (virtual functions).
(c) Overloading can occur within the same class, while overriding requires inheritance.
(d) Both overloading and overriding are used for dynamic memory allocation.

Q4 Polymorphism is achieved through

a) Inheritance *b) Poly programming*

c) Encapsulation *d) Overloading*

9

Q5 What is an abstract class in C++?

a) A class that cannot be instantiated
b) A class that contains only pure virtual functions
c) A class that contains at least one virtual function
d) A class that is defined with the keyword "abstract"

Q6 In C++, when is dynamic polymorphism achieved?

a) During compile-time
b) During runtime
c) During both compile-time and runtime
d) Dynamic polymorphism is not possible in C++

Q7 In compile-time polymorphism, a compiler is able to select the appropriate function for a particular call at the compile time itself, which is known as

a) early binding *b) static binding*
c) dynamic binding *d) both A & B*

Section: B **Quiz 13** Roll number: 23008

Q8 What will be the output of following code snippet **(3)**

```
#include <iostream>
using namespace std;

class base {
 public:
      base (){
        cout<<"Hello from Base"<<endl;
    }
      virtual void display(){
    cout<<"Displaying Base"<<endl;
}
};

class derived :public base{
    public:
derived(){
        cout<<"Hello from derived"<<endl;

    }
    void display(){
        cout<<"Displaying derived"<<endl;
    }
};
int main() {
    base *b = new derived();
    b->display();
    delete b;

    return 0;
}
```

Output:

Hellor from Base.
Hello from derived.
Displaying derived