Practical No. 11

Date: - / /

Q.1) Destructor.

```
#include <iostream>

using namespace std;

class Employee

{

public:

    Employee()

    {

        cout << "Constructor Invoked" << endl;

    }

    ~Employee()

    {

        cout << "Destructor Invoked" << endl;

    }

};

int main(void)

{

    Employee e1; // creating an object of Employee

    Employee e2; // creating an object of Employee

    return 0;

}
```

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\91930\Desktop\cpp journal> g++ PractNo11.cpp
PS C:\Users\91930\Desktop\cpp journal> .\a.exe
Constructor Invoked
Constructor Invoked
Destructor Invoked
Destructor Invoked
PS C:\Users\91930\Desktop\cpp journal>
```

Practical No. 12

Date: - / /

Q.1) Dynamic initialisation of object.

```
#include <iostream>

using namespace std;

class MyClass {

public:

    MyClass(int val) : value(val) {

        cout << "Constructor called with value: " << value << endl;

    }

    ~MyClass() {

        cout << "Destructor called for value: " << value << endl;

    }

    void display() {

        cout << "Value: " << value << endl;

    }

private:

    int value;

};

int main() {

    // Dynamic initialization of object

    MyClass *objPtr = new MyClass(10);

    // Accessing member function

    objPtr->display();
```

```
    // Deallocating memory

    delete objPtr;

    return 0;

}
```

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\91930\Desktop\cpp journal> g++ PractNo12.cpp
PS C:\Users\91930\Desktop\cpp journal> .\a.exe
Constructor called with value: 10
Value: 10
Destructor called for value: 10
PS C:\Users\91930\Desktop\cpp journal>
```

Practical No. 13

Date: - / /

Q.1) Illustrating inheritance.

- **single inheritance.**

```
#include <iostream>

using namespace std;

class Account

{

public:

    float salary = 60000;

};

class Programmer : public Account

{

public:

    float bonus = 5000;

};

int main(void)

{

    Programmer p1;

    cout << "Salary: " << p1.salary << endl;

    cout << "Bonus: " << p1.bonus << endl;

    return 0;

}
```

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\91930\Desktop\cpp journal> g++ PractNo13.cpp
PS C:\Users\91930\Desktop\cpp journal> .\a.exe
Salary: 60000
Bonus: 5000
PS C:\Users\91930\Desktop\cpp journal>
```

- **Multilevel inheritance.**

```
#include <iostream>
using namespace std;
class Animal
{
public:
    void eat()
    {
        cout << "Eating..." << endl;
    }
};
class Dog : public Animal
{
public:
    void bark()
    {
        cout << "Barking..." << endl;
    }
};
class BabyDog : public Dog
{
public:
```

```
    void weep()
    {
        cout << "Weeping...";
    }
};
int main(void)
{
    BabyDog d1;
    d1.eat();
    d1.bark();
    d1.weep();
    return 0;
}
```

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\91930\Desktop\cpp journal> g++ PractNo13b.cpp
PS C:\Users\91930\Desktop\cpp journal> .\a.exe
Eating...
Barking...
Weeping...
PS C:\Users\91930\Desktop\cpp journal>
```

Practical No. 14

Date: - / /

Q.1) Perform static and dynamic polymorphism.

```
#include <iostream>

using namespace std;

// Base class

class Animal {

public:

    virtual void sound() {

        cout << "Animal makes a sound" << endl;

    }

    void eat() {

        cout << "Animal eats food" << endl;

    }

};

// Derived class

class Dog : public Animal {

public:

    void sound() override {

        cout << "Dog barks" << endl;

    }

    void eat() {

        cout << "Dog eats Meat" << endl;

    }
```

```
};
int main() {
    // Static polymorphism
    Animal animal;
    Dog dog;
    animal.eat();
    dog.eat();
    std::cout << std::endl;
    // Dynamic polymorphism
    Animal* ptr = &animal;
    ptr->sound();
    ptr = &dog;
    ptr->sound();
    return 0;
}
```

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\91930\Desktop\cpp journal> g++ PractNo14.cpp
PS C:\Users\91930\Desktop\cpp journal> .\a.exe
Animal eats food
Dog eats Meat

Animal makes a sound
Dog barks
PS C:\Users\91930\Desktop\cpp journal>
```

Practical No. 15

Date: - / /

Q.1) Demonstrate virtual and pure virtual function.

```
#include<iostream>

using namespace std;


class baseClass {

    public:

        int bvalue;


        virtual void display(){

            cout<<"the value of base class is "<<bvalue<<endl;

        }

};


class derivedClass: public baseClass{

    public:

    int dvalue = 23;

    void display(){

        cout<<"the value of derived class is "<<dvalue<<endl;

    }

};


int main(){
```

```
    baseClass * basePtr;

    baseClass b1;

    derivedClass d1;

    basePtr = &d1;

    // basePtr->dvalue;   throw an error without using vertual function

    basePtr->display();

    return 0;

}

}
```

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\91930\Desktop\Cpp_BCA\Practical Qpractice> cd "c:\Users\91930\Desktop\Cp
 g++ vf.cpp -o vf } ; if ($?) { .\vf }
the value of derived class is 23
PS C:\Users\91930\Desktop\Cpp_BCA\Practical Qpractice>
```

#pure virtual

```
#include<iostream>

using namespace std;


class baseClass{

    public:

        int bv;


        virtual void display() = 0;

};


class derivedClass: public baseClass{

    public:

        int dv;

        void display(){

            cout<<"the value of derived class is "<<dv<<endl;

        }

};


class SDClass: public baseClass{

    public:

        int Sdv;

        void display(){

            cout<<"the value of second derived class is "<<Sdv<<endl;
```

```
        }
};

int main(){
    baseClass *bptr;
    derivedClass d1;
    SDClass sd1;
    // bptr[0] = &d1;
    bptr = &d1;

    bptr->display();

    return 0;
}
```

Output :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\91930\Desktop\Cpp_BCA\Practical Qpractice> cd "c:\Users\91930\Desktop\Cpp_BCA
 g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
the value of derived class is 0
PS C:\Users\91930\Desktop\Cpp_BCA\Practical Qpractice>
```