Experiment Name: Enhancing Code Flexibility with Virtual Functions in C++.

Objective: To demonstrate the use of virtual functions in C++ programming.

Description: In this experiment, we will learn about virtual functions and their use in C++ programming. We will create a base class and derived class and demonstrate how virtual functions work in the context of inheritance. We will also solve four programming problems that utilize virtual functions.

Problem 1: Implementing a Base Class and Derived Class with Virtual Functions.

Source Code:

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

class base{
      public:
         int i;
         base(int x){i=x;}
         inline void virtual func(){cout<<i<<endl;}
};
class derived1 : public base{
      public:
         derived1(int x):base(x){}
         inline void func(){cout<<i*i<<endl;}
};
class derived2 : public base{
      public:
         derived2(int x):base(x){}
         void func(){cout<<i+i<<endl;}
};

int main(){
      base      *p;
      base ob(10);
      derived1 d1(20);
      derived2 d2(30);
      p=&ob;
      p->func();
      p=&d1;
      p->func();
      p=&d2;
      p->func();
     return 0;
}
```

Sample Output:

```
10
400
60
```

Problem 2: Using Virtual Functions to Implement Polymorphism.

Source Code:

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

class base{
      public:
          int i;
          base(int x){i=x;}
          inline void virtual func(){cout<<i<<endl;}
};
class derived1 : public base{
      public:
          derived1(int x):base(x){}
          inline void func(){cout<<i*i<<endl;}
};
    class derived2 : public base{
      public:
          derived2(int x):base(x){}
};

int main(){
      base       *p;
      base ob(10);
      derived1 d1(20);
      derived2 d2(30);
      p=&ob;
      p->func();
      p=&d1;
      p->func();
      p=&d2;
      p->func();
    return 0;
}
```

Sample Output:

```
10
400
30
```

Problem 3: Using Virtual Functions to Implement a Bank Account.

Source Code:

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

```
class base{
        public:
            int i;
            base(int x){i=x;}
            inline void virtual func(){cout<<i<<endl;}
};
class derived1 : public base{
        public:
            derived1(int x):base(x){}
            inline void func(){cout<<i*i<<endl;}
};
class derived2 : public base{
        public:
            derived2(int x):base(x){}
            inline void func(){cout<<i+i<<endl;}
};
int main(){
        base *p;
        base ob(10);
        derived1 d1(20);
        derived2 d2(30);
        for(int i=0,j; i<4; i++){
            j=rand();
            if(j%2) p = &d1;
            else p = &d2;
            p->func();
        }
    return 0;
}
```

Sample Output:

```
400
400
60
60
```

Problem 4: Area of Shapes Using Virtual Function.

Source Code:

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

class Shape {
        public:
                virtual void calculate_area() {
                        cout << "Area calculation not implemented for this shape." << endl;
                }
                float area;
};
```

```
class Rectangle : public Shape {
   public:
      void calculate_area() {
         float length, breadth;
         cin >> length >> breadth;
         area = length * breadth;
      }
};
class Circle : public Shape {
   public:
      void calculate_area() {
         float radius;
         cin >> radius;
         area = 3.14 * radius * radius;
      }
};

int main() {
   Shape *s;
   Rectangle r;
   Circle c;
   s = &r;
   s->calculate_area();
   cout << "Area of rectangle: " << s->area << endl;
   s = &c;
   s->calculate_area();
   cout << "Area of circle: " << s->area << endl;
   return 0;
}
```

Sample Input:

```
Enter length and breadth: 5 3
Enter radius: 4
```

Sample Output:

```
Area: 15
Area: 50.24
```

Conclusion:

We have successfully demonstrated how virtual functions can be used to implement runtime polymorphism in C++ programs. Through the use of virtual functions, we were able to write more flexible and extensible code that can be easily modified and extended in the future. Our experiment has also highlighted some limitations of virtual functions, such as the overhead associated with dynamic dispatch and the potential for object slicing. These limitations should be considered when designing C++ programs that make use of virtual functions.