Lab 8

In this lab, you should **NOT** change the given code in the description. **The lab is due 4/27 11:59 PM.**

1. Create a class `PFArrayD` which represents a dynamic array of double values. The class has a member variable `vec` which is of type `vector<double>`, a default constructor that initialize the capacity to 50 and a constructor that takes an `unsigned` argument to specify the initial capacity. It also has three member functions: `addElement()` to add an element to the end of the array, `getCapacity()` to return the capacity of the array, and `getNumberUsed()` to return the number of elements in the array. Complete the code below so that the main function provided below can be executed and output the same result as given below.

```
class PFArrayD {
private:
    vector<double> vec;

public:
    // Default constructor
    PFArrayD() {
        // Fill in the blank
    }
    // Constructor with unsigned argument
    PFArrayD(unsigned cap) {
        // Fill in the blank
    }

    void addElement(double ele) {
        // Fill in the blank
    }
    int getCapacity() {
        // Fill in the blank
    }
    int getNumberUsed() {
        // Fill in the blank
    }
};
```

Use the main function below to test your code:

(Next page)

```
int main() {
    PFArrayD pfa1;
    pfa1.addElement(1.0);
    pfa1.addElement(2.0);
    cout << "Capacity for pfa1: " << pfa1.getCapacity() << endl;
    cout << "Elements used in pfa1: " << pfa1.getNumberUsed() << endl;

    cout << "----------------" << endl;

    PFArrayD pfa2(30);
    pfa2.addElement(3.0);
    cout << "Capacity for pfa2: " << pfa2.getCapacity() << endl;
    cout << "Elements used in pfa2: " << pfa2.getNumberUsed() << endl;

    return 0;
}
```

The output should be **the same** as follows:

```
PFArrayD()
Allocate 50 doubles
Capacity for pfa1: 50
Elements used in pfa1: 2
----------------
PFArrayD(unsigned)
Allocate 30 doubles
Capacity for pfa2: 30
Elements used in pfa2: 1
```

2. Continuing from the previous question, add a **copy constructor** to the class `PFArrayD`. Use the main function below to test your code:

   (Next page)

```
int main() {
    PFArrayD pfa1;
    pfa1.addElement(1.0);
    pfa1.addElement(2.0);
    cout << "Capacity for pfa1: " << pfa1.getCapacity() << endl;
    cout << "Elements used in pfa1: " << pfa1.getNumberUsed() << endl;

    cout << "----------------" << endl;

    PFArrayD pfa2(30);
    pfa2.addElement(3.0);
    cout << "Capacity for pfa2: " << pfa2.getCapacity() << endl;
    cout << "Elements used in pfa2: " << pfa2.getNumberUsed() << endl;

    cout << "----------------" << endl;

    PFArrayD pfa3 = pfa2;
    cout << "Capacity for pfa3: " << pfa3.getCapacity() << endl;
    cout << "Elements used in pfa3: " << pfa3.getNumberUsed() << endl;

    return 0;
}
```

The output should be **the same** as follows:

```
PFArrayD()
Allocate 50 doubles
Capacity for pfa1: 50
Elements used in pfa1: 2
----------------
PFArrayD(unsigned)
Allocate 30 doubles
Capacity for pfa2: 30
Elements used in pfa2: 1
----------------
PFArrayD(const PFArrayD&)
Allocate 30 doubles
Capacity for pfa3: 30
Elements used in pfa3: 1
```

3. Continuing from the previous question, also add a **copy assignment operator** to the class `PFArrayD`. Use the main function below to test your code:

```
int main() {
    PFArrayD pfa1;
    pfa1.addElement(1.0);
    pfa1.addElement(2.0);
    cout << "Capacity for pfa1: " << pfa1.getCapacity() << endl;
    cout << "Elements used in pfa1: " << pfa1.getNumberUsed() << endl;

    cout << "----------------" << endl;

    PFArrayD pfa2(30);
    pfa2.addElement(3.0);
    cout << "Capacity for pfa2: " << pfa2.getCapacity() << endl;
    cout << "Elements used in pfa2: " << pfa2.getNumberUsed() << endl;

    cout << "----------------" << endl;

    PFArrayD pfa3 = pfa2;
    cout << "Capacity for pfa3: " << pfa3.getCapacity() << endl;
    cout << "Elements used in pfa3: " << pfa3.getNumberUsed() << endl;

    cout << "----------------" << endl;

    pfa3 = pfa1;
    cout << "Capacity for pfa3: " << pfa3.getCapacity() << endl;
    cout << "Elements used in pfa3: " << pfa3.getNumberUsed() << endl;

    return 0;
}
```

The output should be **the same** as follows:

(Next page)

```
PFArrayD()
Allocate 50 doubles
Capacity for pfa1: 50
Elements used in pfa1: 2
----------------
PFArrayD(unsigned)
Allocate 30 doubles
Capacity for pfa2: 30
Elements used in pfa2: 1
----------------
PFArrayD(const PFArrayD&)
Allocate 30 doubles
Capacity for pfa3: 30
Elements used in pfa3: 1
----------------
operator = (const PFArrayD&)
Release 30 doubles
Allocate 50 doubles
Capacity for pfa3: 50
Elements used in pfa3: 2
```

4. Continuing from the previous question, also add a **destructor** to the class `PFArrayD`. Use the main function below to test your code:
   (Next page)

```
int main() {
    PFArrayD pfa1;
    pfa1.addElement(1.0);
    pfa1.addElement(2.0);
    cout << "Capacity for pfa1: " << pfa1.getCapacity() << endl;
    cout << "Elements used in pfa1: " << pfa1.getNumberUsed() << endl;

    cout << "----------------" << endl;

    PFArrayD pfa2(30);
    pfa2.addElement(3.0);
    cout << "Capacity for pfa2: " << pfa2.getCapacity() << endl;
    cout << "Elements used in pfa2: " << pfa2.getNumberUsed() << endl;

    cout << "----------------" << endl;

    PFArrayD pfa3 = pfa2;
    cout << "Capacity for pfa3: " << pfa3.getCapacity() << endl;
    cout << "Elements used in pfa3: " << pfa3.getNumberUsed() << endl;

    cout << "----------------" << endl;

    pfa3 = pfa1;
    cout << "Capacity for pfa3: " << pfa3.getCapacity() << endl;
    cout << "Elements used in pfa3: " << pfa3.getNumberUsed() << endl;

    cout << "----------------" << endl;

    return 0;
}
```

The output should be **the same** as follows:

(Next page)

```
PFArrayD()
Allocate 50 doubles
Capacity for pfa1: 50
Elements used in pfa1: 2
----------------
PFArrayD(unsigned)
Allocate 30 doubles
Capacity for pfa2: 30
Elements used in pfa2: 1
----------------
PFArrayD(const PFArrayD&)
Allocate 30 doubles
Capacity for pfa3: 30
Elements used in pfa3: 1
----------------
operator = (const PFArrayD&)
Release 30 doubles
Allocate 50 doubles
Capacity for pfa3: 50
Elements used in pfa3: 2
----------------
~PFArrayD()
Release 50 doubles
~PFArrayD()
Release 30 doubles
~PFArrayD()
Release 50 doubles
```

5. Create a class `Move` with an integer pointer `data`. Complete the code below so that the main function provided below can be executed and output the same result as given below.

```
class Move {
private:
    int* data;

public:
    // Fill in the blank



};
```

Use the main function below to test your code:

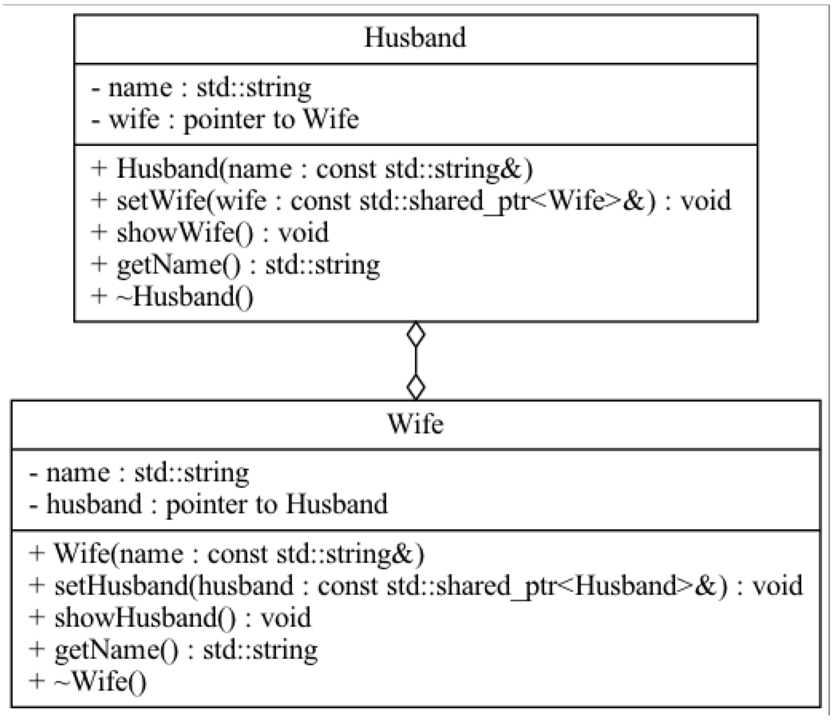
```
int main() {
    vector<Move> vec;
    vec.reserve(2);

    vec.push_back(Move(10));
    vec.push_back(Move(20));
    return 0;
}
```

The output should be **the same** as follows:

```
Constructor is called for 10
Move Constructor for 10
Destructor is called for nullptr
Constructor is called for 20
Move Constructor for 20
Destructor is called for nullptr
Destructor is called for 10
Destructor is called for 20
```

6. Create 2 classes `Husband` and `Wife`. The following is the UML class diagram. The hollow diamond represents aggregation. It means "has-a" relationship. An object of class `Husband` "has-a" object of class `Wife`. And vice versa. Hint: You might need forward declaration in this exercise. Additionally, the pointer type being used in the 2 classes might need some considerations.


Husband
- name : std::string
- wife : pointer to Wife
+ Husband(name : const std::string&)
+ setWife(wife : const std::shared_ptr<Wife>&) : void
+ showWife() : void
+ getName() : std::string
+ ~Husband()
Wife
- name : std::string
- husband : pointer to Husband
+ Wife(name : const std::string&)
+ setHusband(husband : const std::shared_ptr<Husband>&) : void
+ showHusband() : void
+ getName() : std::string
+ ~Wife()

```
int main() {
  shared_ptr<Husband> husband = make_shared<Husband>("John");
  shared_ptr<Wife> wife = make_shared<Wife>("Jane");

  husband->setWife(wife);
  wife->setHusband(husband);

  husband->showWife();
  wife->showHusband();

  return 0;
}
```

Output:

```
Husband John constructor called.
Wife Jane constructor called.
John has his wife set.
Jane has her husband set.
Husband John has wife named Jane
Wife Jane has husband named John
Husband John destructor called.
Wife Jane destructor called.
```