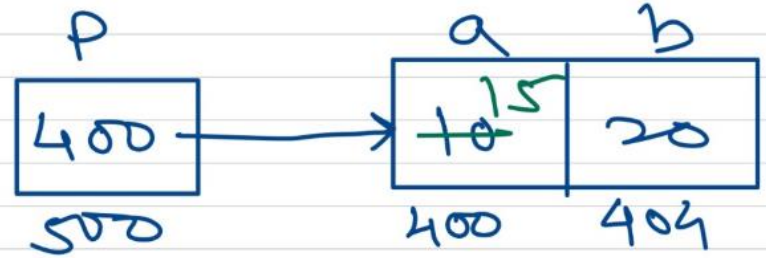
C++ INTERVIEW QUESTIONS

SUNBEAM INFOTECH

C++




```
class test {
    int a;
    int b;
public:
    test() {
        a=10;
        b=20;
    }
    void display() {
        cout << a;
        cout << b;
    }
};
```

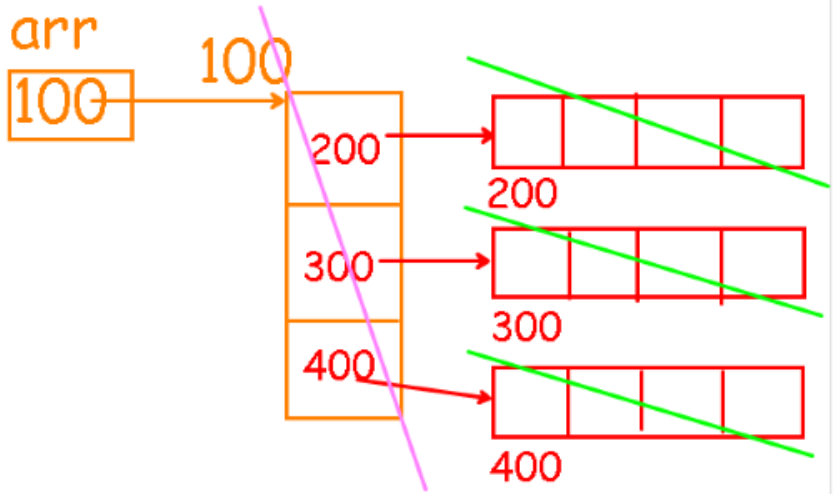
```
main() {
    test obj;
    int *p;
// p = &obj; //error.
    p = reinterpret_cast<int*>(&obj);
    *p = 15;
    obj.display(); → 15, 20
    return 0;
}
```

File Edit Selection View Go ··· private

CPP-Interview.md × Untitled-1 ●

C: > Nilesh > sep-23 > Interview > CPP-Interview.md > ### 2. Write a code to allocate and deallocate memory for multi-dimensional array using new and delete?

```cpp
int** allocateArray(int rows, int cols) {
    int** arr = new int*[rows];  // Allocate memory for array of row pointers
    for (int i = 0; i < rows; ++i) {
        arr[i] = new int[cols];  // Allocate memory for each row
    }
    return arr;
}

void deallocateArray(int** arr, int rows) {
    for (int i = 0; i < rows; ++i) {
        delete[] arr[i];  // Deallocate memory for each row
    }
    delete[] arr;  // Deallocate memory for array of row pointers
}

int main() {
    int rows = 3;
    int cols = 4;
    int** arr = allocateArray(rows, cols);

    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            arr[i][j] = i * cols + j;  // Example initialization
```



main ⊗ 0 ⚠ 0 0 Ln 45, Col 65 Spaces: 4 UTF-8 CRLF Markdown

Search 2:09 PM

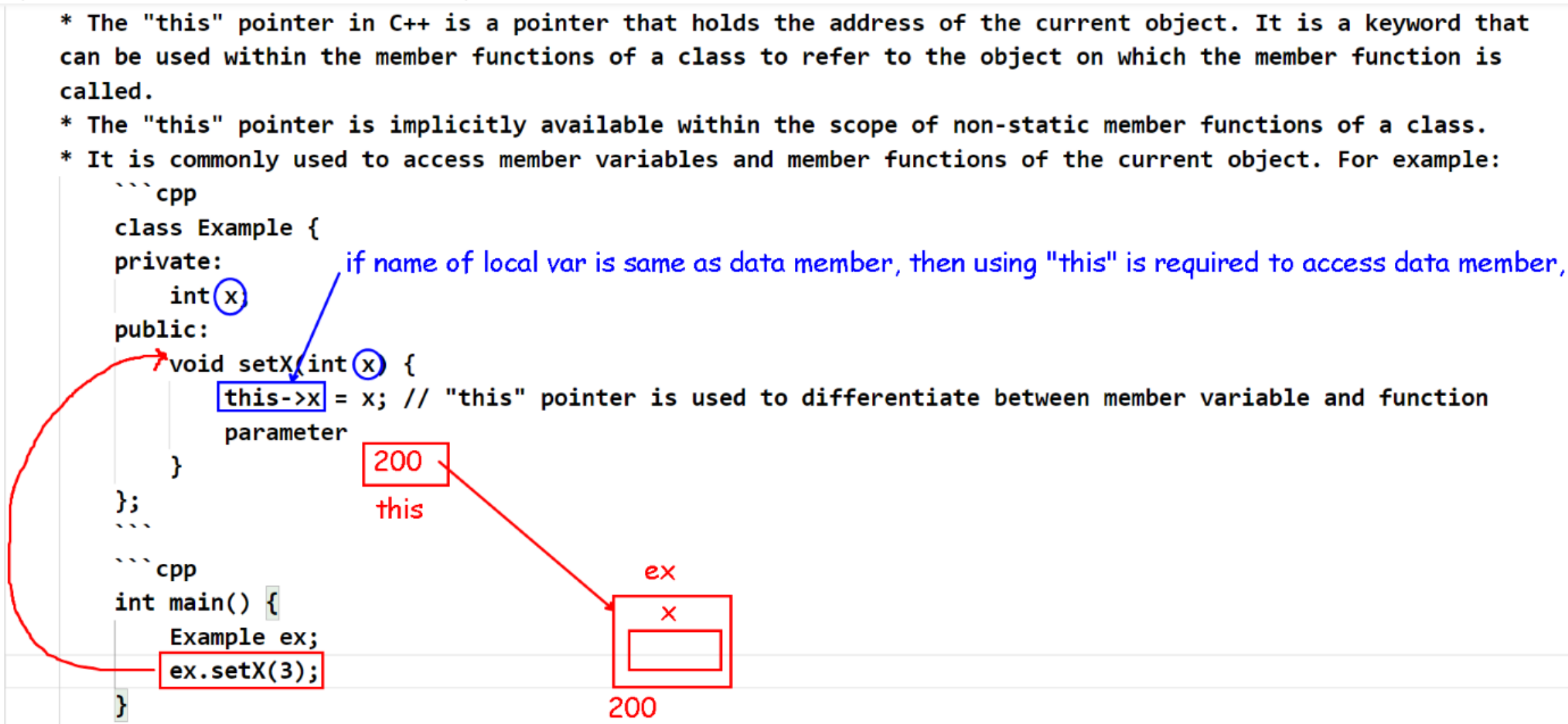
```
16
17  class myclass {
18      int a;
19  public:
20      // ...
21      void func1() {
22          // this -- myclass * const this;
23          this->a = 10; // okay
24      }
25      void func2() const {
26          // this -- const myclass * const this;
27          this->a = 10; // error
28      }
29  };
30
```

File Edit Selection View Go ··· private

CPP-Interview.md Untitled-1

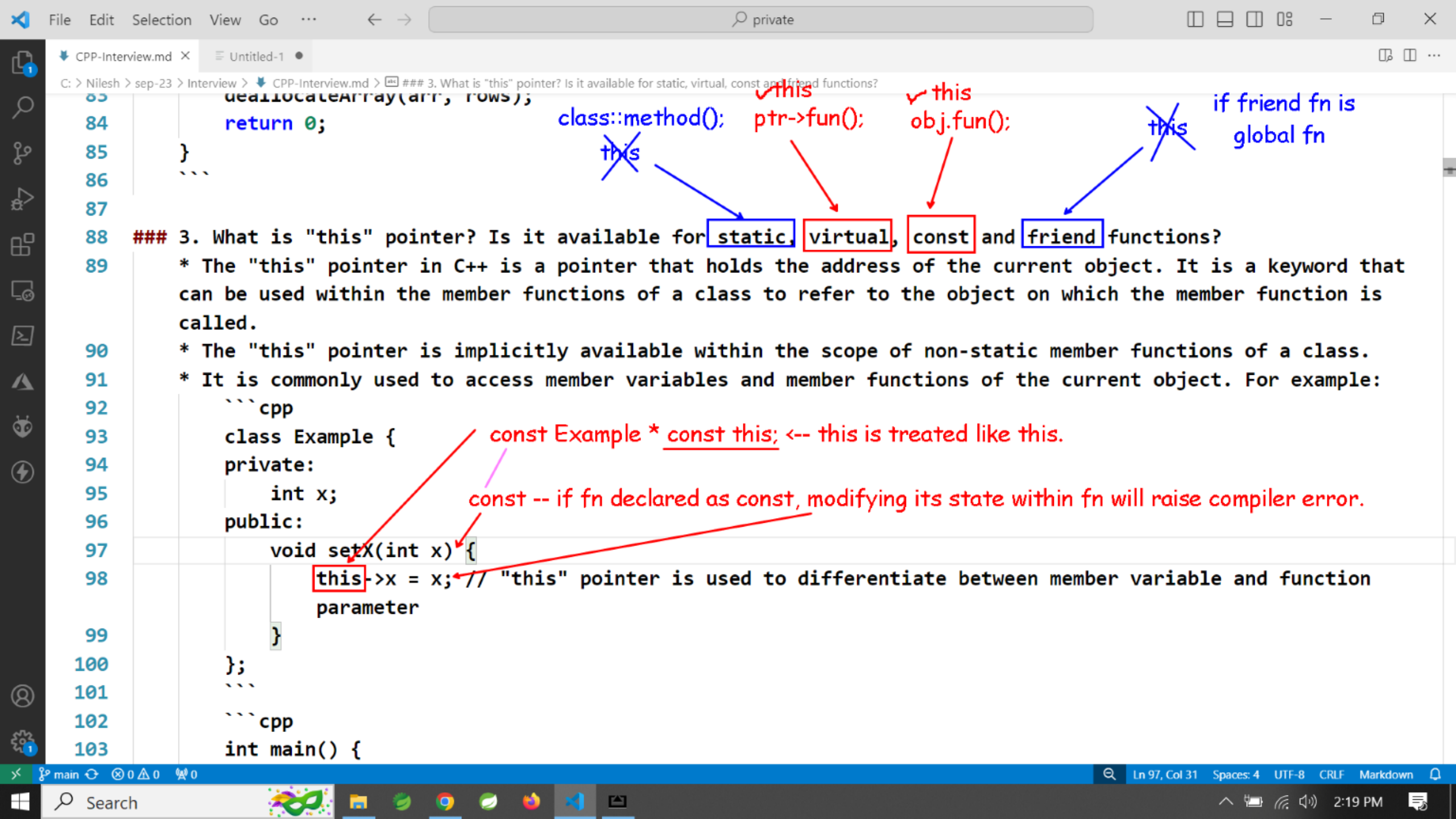
C: > Nilesh > sep-23 > Interview > CPP-Interview.md > ### 3. What is "this" pointer? Is it available for static, virtual, const and friend functions?

```
89  * The "this" pointer in C++ is a pointer that holds the address of the current object. It is a keyword that
    can be used within the member functions of a class to refer to the object on which the member function is
    called.
90  * The "this" pointer is implicitly available within the scope of non-static member functions of a class.
91  * It is commonly used to access member variables and member functions of the current object. For example:
92      ```cpp
93      class Example {
94      private:
95          int x;
96      public:
97          void setX(int x) {
98              this->x = x; // "this" pointer is used to differentiate between member variable and function
                parameter
99          }
100     };
101     ```
102     ```cpp
103     int main() {
104         Example ex;
105         ex.setX(3);
106     }
107     ```
108 * **Static Member Functions:** "this" pointer is not available within static member functions because static
    member functions do not operate on specific class instances/objects
```



main 0 0 0 105, Col 24 Spaces: 4 UTF-8 CRLF Markdown

Search 2:16 PM




```
    deallocateArray(arr, rows);
    return 0;
}
```

3. What is "this" pointer? Is it available for static, virtual, const and friend functions?

* The "this" pointer in C++ is a pointer that holds the address of the current object. It is a keyword that can be used within the member functions of a class to refer to the object on which the member function is called.
* The "this" pointer is implicitly available within the scope of non-static member functions of a class.
* It is commonly used to access member variables and member functions of the current object. For example:

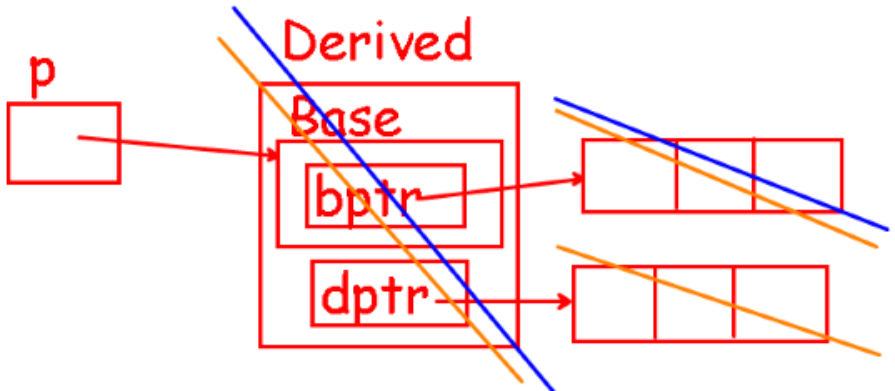
```cpp
class Example {
private:
    int x;
public:
    void setX(int x) {
        this->x = x; // "this" pointer is used to differentiate between member variable and function parameter
    }
};
```

```cpp
int main() {
```

```cpp
class Base {
    int *bptr;
public:
    Base() {
        bptr = new int[3];
        std::cout << "Base constructor called" << std::endl;
    }
    virtual ~Base() {
        delete[] bptr;
        std::cout << "Base destructor called" << std::endl;
    }
};
class Derived : public Base {
    int *dptr;
public:
    Derived() {
        dptr = new int[3];
        std::cout << "Derived constructor called" << std::endl;
    }
    ~Derived() {
        delete[] dptr;
        std::cout << "Derived destructor called" << std::endl;
    }
```

```
Base *p = new Derived;
// ...
delete p;
```

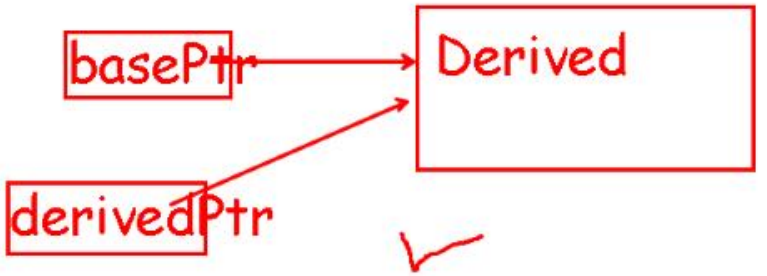
main()



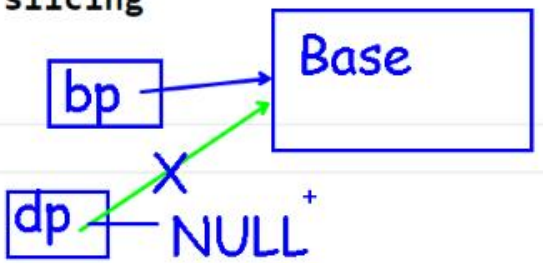
if destructor is not virtual, Derived cls destructor not called. Hence "dptr" pointing memory will leak.

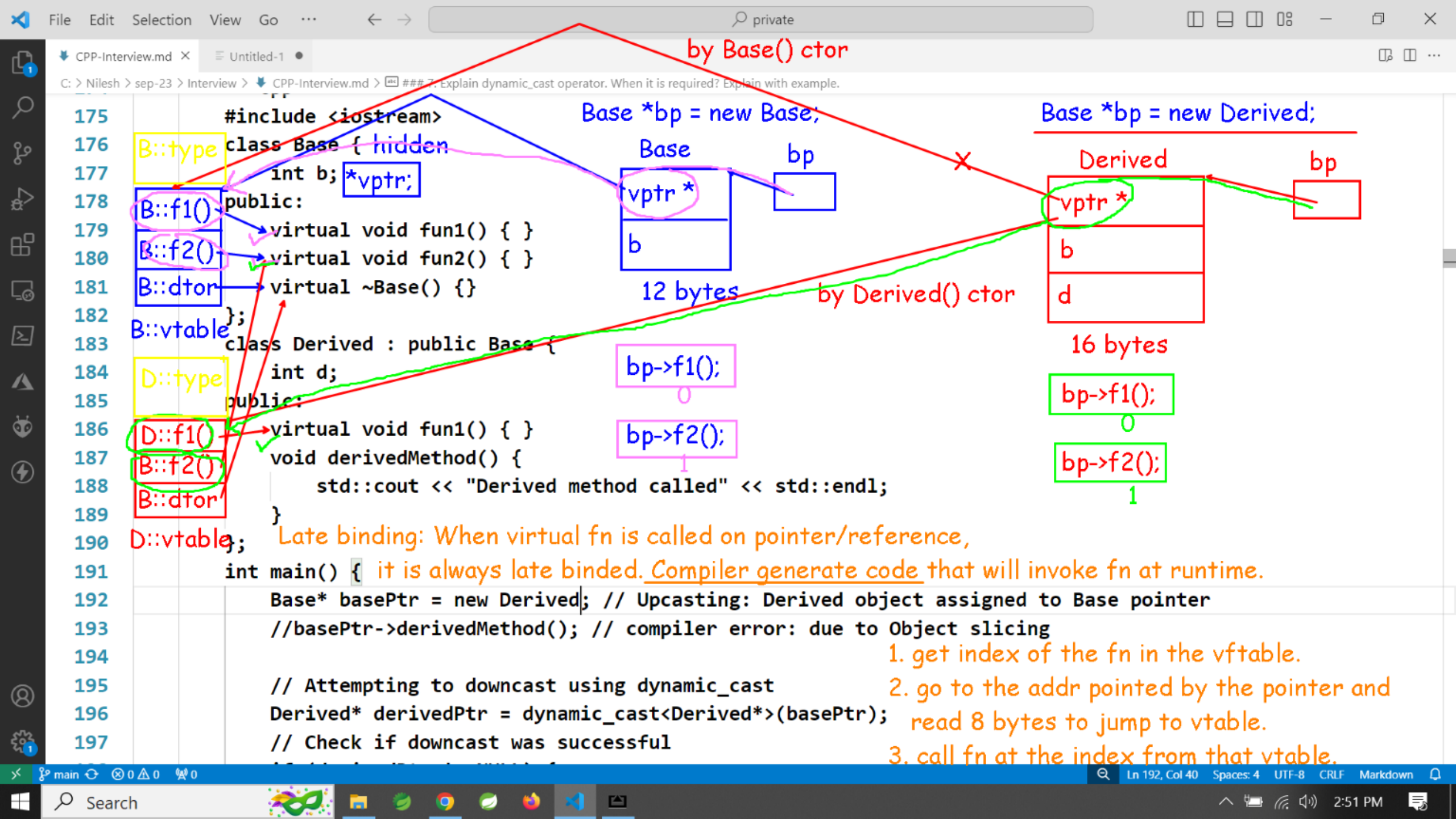
if destructor is virtual, first derived destructor, then base destructor called and then object memory released,

```cpp
class Base {
public:
    virtual ~Base() {}
};
class Derived : public Base {
public:
    void derivedMethod() {
        std::cout << "Derived method called" << std::endl;
    }
};
int main() {
    Base* basePtr = new Derived(); // Upcasting: Derived object assigned to Base pointer
    //basePtr->derivedMethod(); // compiler error: due to Object slicing

    // Attempting to downcast using dynamic_cast
    Derived* derivedPtr = dynamic_cast<Derived*>(basePtr);
    // Check if downcast was successful
    if (derivedPtr != NULL) {
        // Downcast successful, call derived class method
        derivedPtr->derivedMethod();
    } else {
        // Downcast failed, handle accordingly
        std::cout << "Dynamic cast failed" << std::endl;
    }
```



Base *bp = new Base;

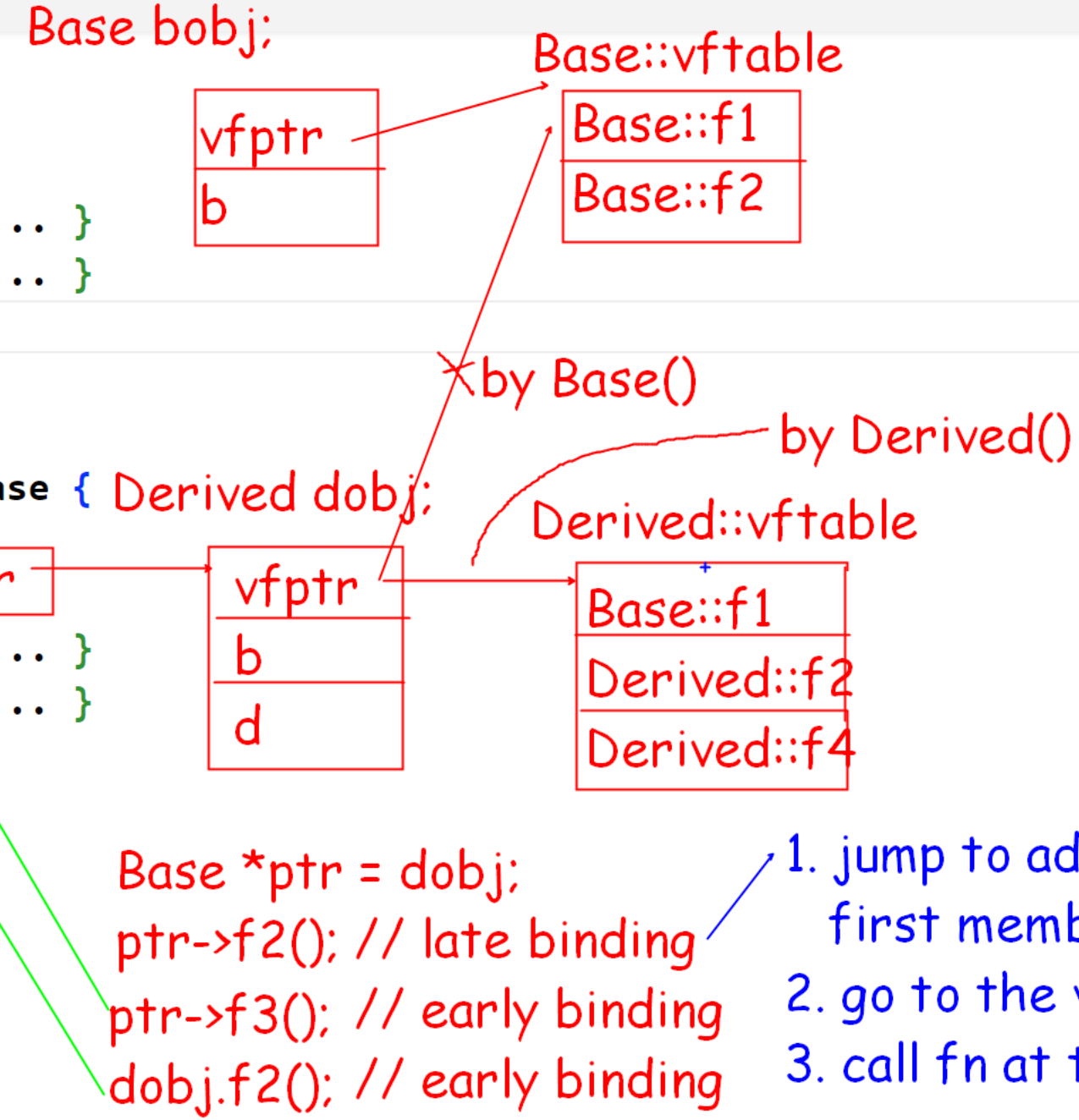




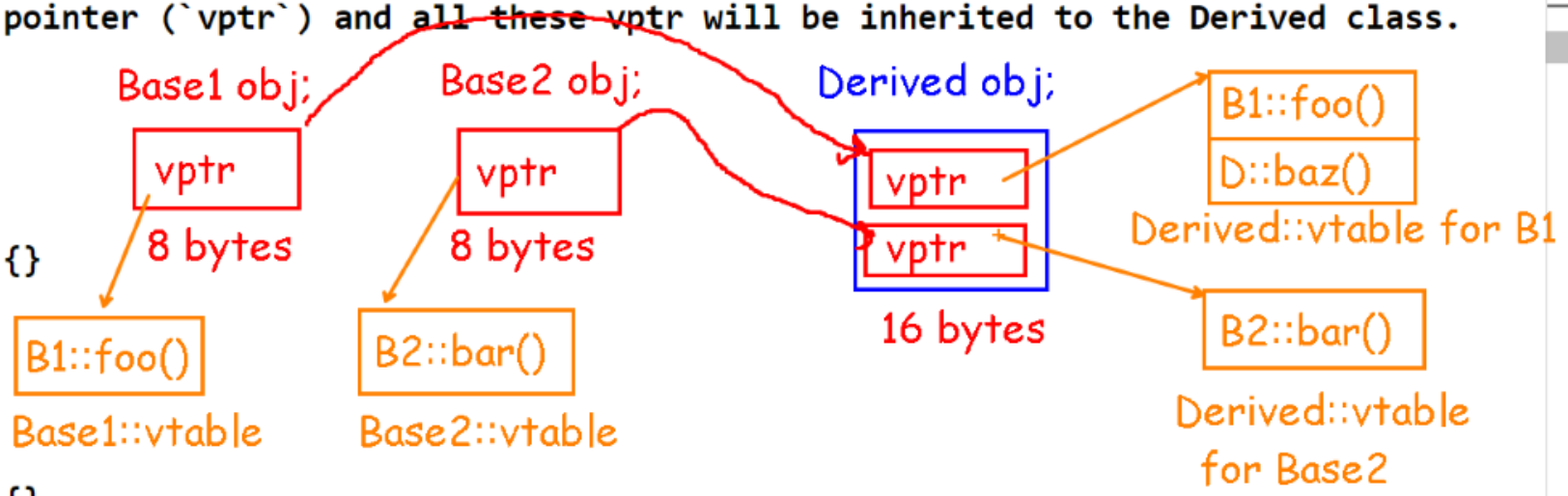

```
175  #include <iostream>
176  class Base {
177      int b;
178  public:
179      virtual void fun1() { }
180      virtual void fun2() { }
181      virtual ~Base() {}
182  };
183  class Derived : public Base {
184      int d;
185  public:
186      virtual void fun1() { }
187      void derivedMethod() {
188          std::cout << "Derived method called" << std::endl;
189      }
190  };
191  int main() {
192      Base* basePtr = new Derived; // Upcasting: Derived object assigned to Base pointer
193      //basePtr->derivedMethod(); // compiler error: due to Object slicing
194
195      // Attempting to downcast using dynamic_cast
196      Derived* derivedPtr = dynamic_cast<Derived*>(basePtr);
197      // Check if downcast was successful
```

CPP-Interview.md | OOP-Interview.md | Untitled-1 ●

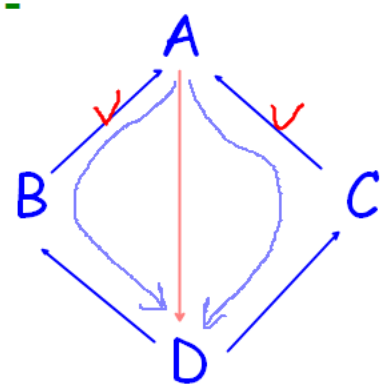
```
59  class Base {
60  public: void *vfptr;
61      int b;
62      virtual void f1() { ... }
63      virtual void f2() { ... }
64      void f3() { ... }
65  };
66
67  class Derived : public Base { Derived dobj;
68  public:
69      int d;
70      virtual void f2() { ... }
71      virtual void f4() { ... }
72      void f5() { ... }
73  };
74
75  Base *p = new Derived;
76  p->f2();
77  p->f1();
78  p->f3();
```



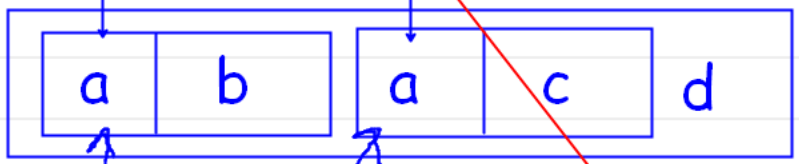
```
            e. vptr of Base is inherited to the Derived.
            return 0;
        }
        ```
* **Multiple Inheritance:**
    * In multiple inheritance, where a class has multiple direct base classes, each base class will have its own virtual function table pointer (`vptr`) and all these vptr will be inherited to the Derived class.
    * Example:
        ```cpp
        class Base1 {
        public:
            virtual void foo() {}
        };

        class Base2 {
        public:
            virtual void bar() {}
        };

        class Derived : public Base1, public Base2 {
        public:
            virtual void baz() {}
        };
```



```
80  // ----------------------------------------------
81  class A -- a
82
83  class B: A -- b
84
85  class C: A -- c
86
87  class D: B, C -- d
88
89  D obj;
90
```



multiple copies ??



obj.a = 10;

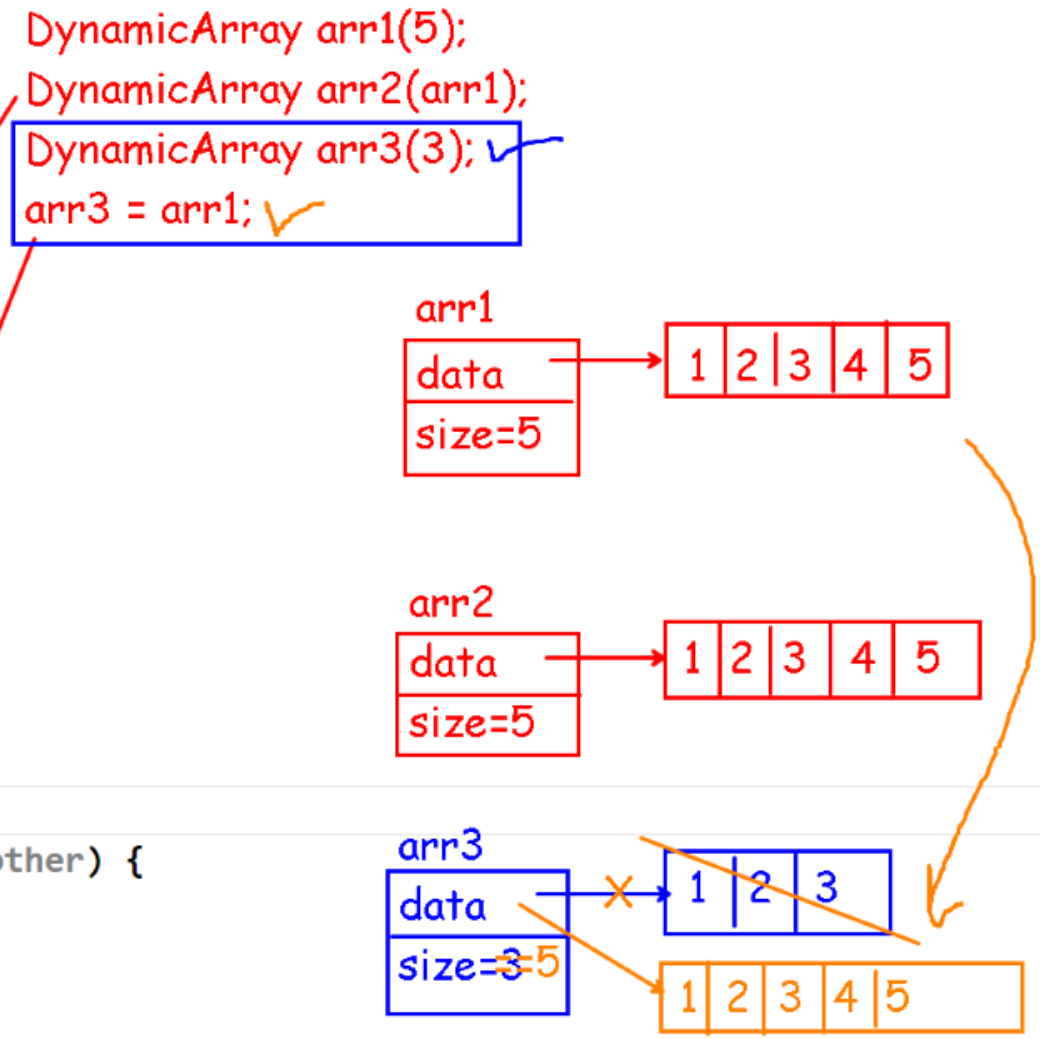
ambiguity...

obj::A = 10;

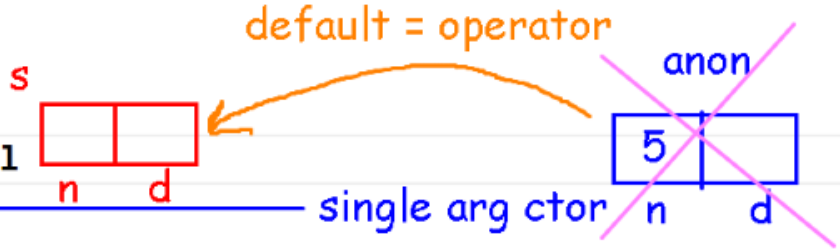
Virtual Inheritance

obj.a = 20; // no ambiguity

```cpp
class DynamicArray {
private:
    int* data;
    int size;
public:
    DynamicArray(int sz) {
        size = sz;
        data = new int[size];
        for (int i = 0; i < size; ++i)
            data[i] = i + 1;
    }
    DynamicArray(const DynamicArray& other) {
        size = other.size;
        data = new int[size];
        for (int i = 0; i < size; ++i)
            data[i] = other.data[i];
    }
    DynamicArray& operator=(const DynamicArray& other) {
        if (this != &other) {
            delete[] data;
            size = other.size;
            data = new int[size];
            for (int i = 0; i < size; ++i)
                data[i] = other.data[i];
        }
```



```
511         }
512         Rational(int num) {
513             numerator = num;
514             denominator = 1;
515         }
516         operator double() const {
517             return (double)numerator / denominator;
518         }
519     private:
520         int numerator;
521         int denominator;
522     };
523
524     int main() {
525         Rational r(3, 2);
526         double d = r;  // Implicit conversion to double
527             // d = r.operator double();
528         int x = 5;
529         Rational s;
530         s = x; // Implicit conversion from int to Rational
531             // x --> anonymous Rational object
532             // s = anonymous object (assignment)
533             // anonymous object destroyed
534         return 0;
```



THANK YOU

NILESH GHULE <NILESH@SUNBEAMINFO.COM>