Final Review

Computer Systems Programming, Spring 2023

Instructor: Travis McGaha

TAs:

Kevin Bernat
Jialin Cai
Mati Davis
Donglun He
Chandravaran Kunjeti
Heyi Liu
Shufan Liu
Eddy Yang

Logistics

- Late Policy:
  - You can still use the same late policy for HW4 and the final project
  - I can grant extensions into reading days
  - I REALLY don’t want to grant extensions into finals week
  - Email me (Travis) at least a day in advance of the deadline so that I have time to process the extension
- Final Exam: May 2nd @noon to May 6th @noon
  - Cumulative & Midterm Clobber policy
- Travis’ OH TODAY from 5-7 pm
  - May have some during next week, TBD

Logistics

- HW4 Posted Due Thursday 4/20 @ 11:59

  Extended to 5/5 @11:59 pm

  I WILL GRANT FEW (if any) EXTENSIONS PAST THIS

- Project Released! Due Wednesday 4/26 @ 11:59

  Extended to 5/5 @11:59 pm

  I WILL GRANT FEW (if any) EXTENSIONS PAST THIS

- HW2 grades & Midterm grades posted
  - Can fix HW2 submissions
  - Midterm has regrades & the clobber policy

Lecture Outline

- **Final Exam Review**

Review Topics

- Scheduling
- Threads
- IPC
- Networks (P1, P2, P3)
- C++ Casting
- Smart Pointers
- Inheritance (P1 & P2)
- C++ Copying

NOTE: These are not all the topics that ***could*** be on the final. List is trimmed for review due to time constraints.

In What order do the processes finish?

Scheduling

- The following processes are scheduled using a standard **Priority Round Robin** scheme.

Process Name	Arrival Time	Execution Time	Priority
Ape	0	7	medium
Bear	1	3	medium
Chinchilla	3	4	medium
Dolphin	4	4	low
Elephant	7	2	high
Flamingo	21	2	medium

  - You may assume the following:
    - the quantum for all processes (regardless of priority) is 2
    - context switching is instantaneous
    - if a process arrives and its priority is **higher** than that of the process that is currently running, the newly-arrived process is immediately scheduled; in that case, the process that is preempted goes to the end of its queue, but is able to run for a full quantum the next time it is scheduled
    - if a process' time slice ends at the same time as another process of the same priority arrives, the one that just arrived goes into the queue **before** the one that just finished its time slice

In What order do the processes finish?

Scheduling

EBACDF

- The following processes are scheduled using a standard **Priority Round Robin** scheme.

Process Name	Arrival Time	Execution Time	Priority
Ape	0	7	medium
Bear	1	3	medium
Chinchilla	3	4	medium
Dolphin	4	4	low
Elephant	7	2	high
Flamingo	21	2	medium

  - You may assume the following:
    - the quantum for all processes (regardless of priority) is 2
    - context switching is instantaneous
    - if a process arrives and its priority is **higher** than that of the process that is currently running, the newly-arrived process is immediately scheduled; in that case, the process that is preempted goes to the end of its queue, but is able to run for a full quantum the next time it is scheduled
    - if a process' time slice ends at the same time as another process of the same priority arrives, the one that just arrived goes into the queue **before** the one that just finished its time slice

Threads

- The code below has three functions that could be executed in separate threads. Note that these are not thread entry points, just functions used by threads:
  - Assume that "lock" has been initialized

```
1  // global variables
2  pthread_mutex_t lock;
3  int g =     0;
4  int k = 0;
5
6  void fun1() {
7    pthread_mutex_lock(&lock);
8    g += 3;
9    pthread_mutex_unlock(&lock);
10   k++;
11 }
12
13 void fun2(int a, int b) {
14   g += a;
15   a += b;
16   k = a;
17 }
18
19 void fun3() {
20   pthread_mutex_lock(&lock);
21   g = k + 2;
22   pthread_mutex_unlock(&lock);
23 }
```

- Thread-1 executes line 8 while Thread-2 executes line 21. Choose one:
  - Could lead to a race condition.
  - There is no possible race condition.
  - The situation cannot occur.

- Thread-1 executes line 15 while Thread-2 executes line 15. Choose one:
  - Could lead to a race condition.
  - There is no possible race condition.
  - The situation cannot occur.

Threads

- The code below has three functions that could be executed in separate threads. Note that these are not thread entry points, just functions used by threads:
  - Assume that "lock" has been initialized

```
1  // global variables
2  pthread_mutex_t lock;
3  int g =    0;
4  int k = 0;
5
6  void fun1() {
7    pthread_mutex_lock(&lock);
8    g += 3;
9    pthread_mutex_unlock(&lock);
10   k++;
11 }
12
13 void fun2(int a, int b) {
14   g += a;
15   a += b;
16   k = a;
17 }
18
19 void fun3() {
20   pthread_mutex_lock(&lock);
21   g = k + 2;
22   pthread_mutex_unlock(&lock);
23 }
```

- Thread-1 executes line 8 while Thread-2 executes line 21. Choose one:
  - Could lead to a race condition.
  - There is no possible race condition.
  - The situation cannot occur.

- Thread-1 executes line 15 while Thread-2 executes line 15. Choose one:
  - Could lead to a race condition.
  - There is no possible race condition.
  - The situation cannot occur.

Threads

- The code below has three functions that could be executed in separate threads. Note that these are not thread entry points, just functions used by threads:
  - Assume that "lock" has been initialized

- Thread-1 executes line 8 while Thread-2 executes line 14
  Choose one:
  - Could lead to a race condition.
  - There is no possible race condition.
  - The situation cannot occur.

- Thread-1 executes line 14 while Thread-2 executes line 16.
  Choose one:
  - Could lead to a race condition.
  - There is no possible race condition.
  - The situation cannot occur.

```
1   // global variables
2   pthread_mutex_t lock;
3   int g =    0;
4   int k = 0;
5
6   void fun1() {
7     pthread_mutex_lock(&lock);
8     g += 3;
9     pthread_mutex_unlock(&lock);
10    k++;
11  }
12
13  void fun2(int a, int b) {
14    g += a;
15    a += b;
16    k = a;
17  }
18
19  void fun3() {
20    pthread_mutex_lock(&lock);
21    g = k + 2;
22    pthread_mutex_unlock(&lock);
23  }
```

Threads

- The code below has three functions that could be executed in separate threads. Note that these are not thread entry points, just functions used by threads:
  - Assume that "lock" has been initialized

```c
1  // global variables
2  pthread_mutex_t lock;
3  int g =     0;
4  int k = 0;
5
6  void fun1() {
7    pthread_mutex_lock(&lock);
8    g += 3;
9    pthread_mutex_unlock(&lock);
10   k++;
11 }
12
13 void fun2(int a, int b) {
14   g += a;
15   a += b;
16   k = a;
17 }
18
19 void fun3() {
20   pthread_mutex_lock(&lock);
21   g = k + 2;
22   pthread_mutex_unlock(&lock);
23 }
```

- Thread-1 executes line 8 while Thread-2 executes line 14
  Choose one:
  - Could lead to a race condition.
  - There is no possible race condition.
  - The situation cannot occur.

- Thread-1 executes line 14 while Thread-2 executes line 16.
  Choose one:
  - Could lead to a race condition.
  - There is no possible race condition.
  - The situation cannot occur.

IPC

- The following code intends to use a global variable so that a child process reads a string and the parent prints it.

- Briefly describe two reasons why this program won't work. You can assume it compiles.

```
string message;

void child();
void parent();

int main() {
  pid_t pid = fork();
  if (pid == 0) {
    child();
  } else {
    parent();
  }
}
void child() {
  cin >> message;
}
void parent() {
  cout << message;
}
```

IPC

- The following code intends to use a global variable so that a child process reads a string and the parent prints it.

- Briefly describe two reasons why this program won't work. You can assume it compiles.
  - After fork is called, global variables are no longer shared. Each process has its own "message"
  - There is no synchronization to know if the parent prints after the child reads.

```
string message;

void child();
void parent();

int main() {
  pid_t pid = fork();
  if (pid == 0) {
    child();
  } else {
    parent();
  }
}
void child() {
  cin >> message;
}
void parent() {
  cout << message;
}
```

IPC

- Describe how we would have to rewrite the code if we wanted it to work. Keeping the multiple processes and calls to fork(). Be specific about where you would add the new lines of code.

```
string message;

void child();
void parent();

int main() {
  pid_t pid = fork();
  if (pid == 0) {
    child();
  } else {
    parent();
  }
}
void child() {
  cin >> message;
}
void parent() {
  cout << message;
}
```

IPC

- Describe how we would have to rewrite the code if we wanted it to work. Keeping the multiple processes and calls to fork(). Be specific about where you would add the new lines of code.

- ONE ANSWER:

```
string message;
int fds[2];

void child();
void parent();

int main() {
  pipe(fds);
  pid_t pid = fork();
  if (pid == 0) {
    close(fds[0]);
    child();
  } else {
    close(fds[1]);
    parent();
  }
}
void child() {
  cin >> message;
  wrapped_write(fds[1], message);
}
void parent() {
  wrapped_read(fds[0], message);
  cout << message;
}
```

Networking: pt. 1

- TCP guarantees reliable delivery of the packets that make up a stream, assuming that the socket doesn't fail because of an I/O error.
- IP guarantees reliable delivery of packets, assuming that the socket doesn't fail because of an I/O error.
- Given a particular hostname (like www.amazon.com), getaddrinfo() will return a single IP address corresponding to that name.
- A single server machine can handle connection requests sent to multiple IP addresses.
- A struct sockaddr_in6 contains only an ipv6 address.
- The HTTP payload takes up a larger percentage of the overall packet sent over the network than the IP payload.

Networking: pt. 1

- TCP guarantees reliable delivery of the packets that make up a stream, assuming that the socket doesn't fail because of an I/O error.
  - True
- IP guarantees reliable delivery of packets, assuming that the socket doesn't fail because of an I/O error.
  - False
- Given a particular hostname (like www.amazon.com), getaddrinfo() will return a single IP address corresponding to that name.
  - False
- A single server machine can handle connection requests sent to multiple IP addresses.
  - True
- A struct sockaddr_in6 contains only an ipv6 address.
  - False
- The HTTP payload takes up a larger percentage of the overall packet sent over the network than the IP payload.
  - False

Networking pt. 2

- For each of the following behaviors, identify what networking layer is most closely thought of as being responsible for handling that behavior.
  - Host A tries to send a long message to Host B in another city, broken up into many packets. A packet in the middle does not arrive, so Host A sends it again.
  - Host A tries to send a message to Host B, but Host C and Host D are also trying to communicate on the same network, so Host A must avoid interfering

Networking pt. 2

- For each of the following behaviors, identify what networking layer is most closely thought of as being responsible for handling that behavior.
  - Host A tries to send a long message to Host B in another city, broken up into many packets. A packet in the middle does not arrive, so Host A sends it again.
    - Transport Layer (Protocol commonly associated with this: TCP)
  - Host A tries to send a message to Host B, but Host C and Host D are also trying to communicate on the same network, so Host A must avoid interfering
    - Data Link Layer (Protocol commonly associated with this: MAC)

Networking pt. 3

- The original versions of HTTP (including 1.1) were designed to use plain text characters sent over the network instead of alternatives like a binary encoding for the request and response. Describe one advantage of this design decision and one disadvantage.

- Advantage:

- Disadvantage:

Networking pt. 3

- The original versions of HTTP (including 1.1) were designed to use plain text characters sent over the network instead of alternatives like a binary encoding for the request and response. Describe one advantage of this design decision and one disadvantage.

- Advantage:
  - Interpretable by humans
  - Easy to experiment with and adopt
- Disadvantage:
  - Might be less efficient (for some definition of efficient) than a well-packed binary format

C++ Casting

- For each of these casts in C++, will it be okay, cause a compile time error, or cause a runtime error?

```
struct A {
    int x;
};
struct B {
    float y;
};
struct C : public B {
    char z;
};
```

```
void modify(A* aptr);

int main() {
    A a;
    B b;
    C c;

    B* bptr = static_cast<B*>(&c);
      // ^ OK, CT Err, RT Err

    C* cptr = static_cast<C*>(&b); // OK, CT Err, RT Err

    A* aptr = static_cast<A*>(&b); // OK, CT Err, RT Err

    bptr = &c;
    C* cptr_dyn = dynamic_cast<C*>(bptr); // OK, CT Err, RT Err
```

Could cause a RT error if we try to access cptr->z

C++ Casting

- For each of these casts in C++, will it be okay, cause a compile time error, or cause a runtime error?

```
struct A {
    int x;
};
struct B {
    float y;
};
struct C : public B {
    char z;
};
```

```
void modify(A* aptr);

int main() {
    A a;
    B b;
    C c;

    // ...
    cptr_dyn = dynamic_cast<C*>(&b); // OK, CT Err, RT Err

    const A const_a;
    modify(&const_a); // OK, CT Err, RT Err

    modify(const_cast<A*>(&const_a)); // OK, CT Err, RT Err
```

Could cause a RT error if we try to use cptr_dyn without checking for it being nullptr

C++ Casting

- For each of these casts in C++, will it be okay, cause a compile time error, or cause a runtime error?

```
void modify(A* aptr);

int main() {
    // ...
    int64_t u64 = 0;
    int32_t u32_r = reinterpret_cast<int32_t>(u64);
                    // ^ OK, CT Err, RT Err
    int32_t u32_s = static_cast<int32_t>(u64);
                    // ^ OK, CT Err, RT Err

    float f32 = static_cast<float>(u64);
                    // ^ OK, CT Err, RT Err
    double f64 = reinterpret_cast<double>(u64);
                    // ^ OK, CT Err, RT Err
    double* f64_ptr = reinterpret_cast<double*>(&u64);
                    // ^ OK, CT Err, RT Err
}
```

Double and uint64_t are the same size, but still not allowed

Smart Pointers

- Suppose we have the following declarations at the beginning of a C++ program:

```
int n = 17;
int *x = &n;
int *y = new int(42);
```

- For each part, indicate whether if we were to add just that line(s) after the code above, whether there is a compiler error, some sort of run time error, or memory leak.
  - `unique_ptr a(n);`
  - `unique_ptr b(x);`
  - `unique_ptr c(y);`
  - `unique_ptr d(&n);`
  - `unique_ptr e(new int(333));`
  - `unique_ptr temp(new int(0));`
    `unique_ptr f(temp.get());`

Smart Pointers

- Suppose we have the following declarations at the beginning of a C++ program:

```
int n = 17;
int *x = &n;
int *y = new int(42);
```

- For each part, indicate whether if we were to add just that line(s) after the code above, whether there is a compiler error, some sort of run time error, or memory leak.
  - `unique_ptr a(n);` Won't compile.
  - `unique_ptr b(x);` Compiles, but fails during execution
  - `unique_ptr c(y);` Works
  - `unique_ptr d(&n);` Compiles, but fails during execution
  - `unique_ptr e(new int(333));` Works, but y leaks
  - `unique_ptr temp(new int(0));` Compiles,
    `unique_ptr f(temp.get());` but fails during execution

Inheritance

- Consider the following C++ classes and declared variables.
- What do each of function calls print? (if it compiles)

```
class Animal {
 public:
  virtual void Eat() { cout << "A::E" << endl; }
};

class Dog : public Animal {
 public:
  void Eat() { cout << "D::E" << endl; Bark(); }
  void Bark() { cout << "D::B" << endl; }
};

class Husky : public Dog {
 public:
  virtual void Bark() { cout << "H::B" << endl; }
};
```

```
Dog d;
Husky h;
Dog *d2d = &d;
Animal *a2h = &h;
Dog *d2h = &h;

d2d->Eat();


a2h->Eat();


a2h->Bark();


d2h->Eat();


d2h->Bark();
```

Inheritance

- Consider the following C++ classes and declared variables.
- What do each of function calls print? (if it compiles)

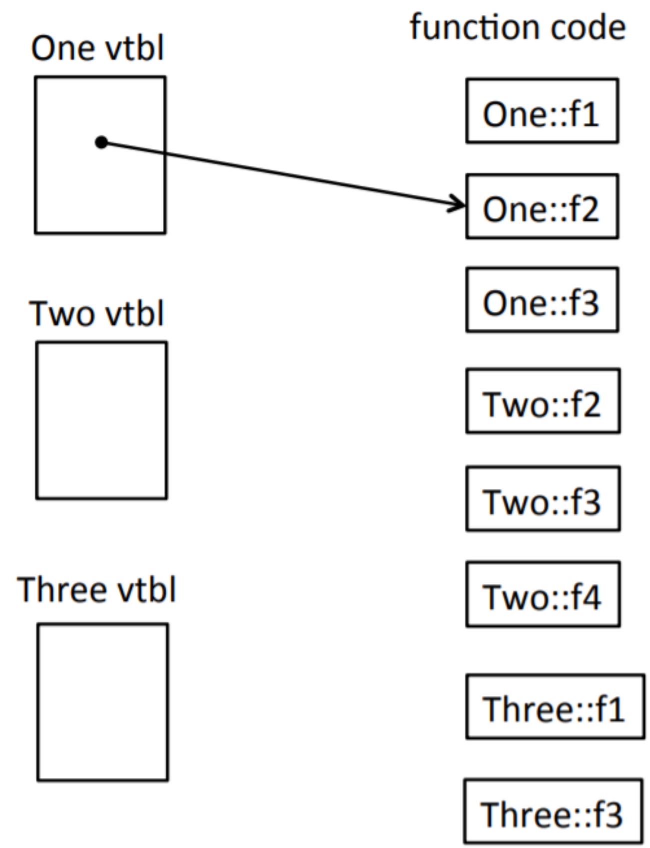
```
class Animal {
 public:
  virtual void Eat() { cout << "A::E" << endl; }
};

class Dog : public Animal {
 public:
  void Eat() { cout << "D::E" << endl; Bark(); }
  void Bark() { cout << "D::B" << endl; }
};

class Husky : public Dog {
 public:
  virtual void Bark() { cout << "H::B" << endl; }
};
```

```
Dog d;
Husky h;
Dog *d2d = &d;
Animal *a2h = &h;
Dog *d2h = &h;

d2d->Eat();
// D::E
// D::B
a2h->Eat();
// D::E
// D::B
a2h->Bark();
// compiler
// error
d2h->Eat();
// D::E
// D::B
d2h->Bark();
// D::B
```

Inheritance

- Complete the diagram below to show the layout of the virtual function tables for the classes given on the previous page. Be sure that the order of pointers in the virtual function tables is clear!

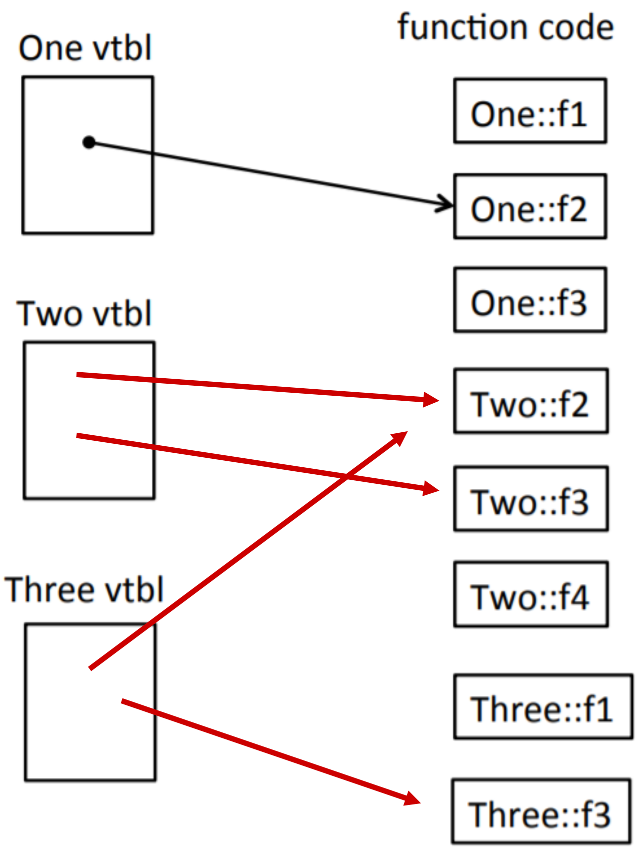
```
class One {
public:
     void f1() { f3(); cout << "One::f1" << endl; }
     virtual void f2() { cout << "One::f2" << endl; }
     void f3() { cout << "One::f3" << endl; }
};


class Two: public One {
public:
     void f4() { cout << "Two::f4" << endl; }
     void f2() { f1(); cout << "Two::f2" << endl; }
     virtual void f3() { f4(); cout << "Two::f3" << endl; }
};
class Three: public Two {
public:
     void f3() { f2(); cout << "Three::f3" << endl; }
     void f1() { cout << "Three::f1" << endl; }
};
```



Inheritance

- Complete the diagram below to show the layout of the virtual function tables for the classes given on the previous page. Be sure that the order of pointers in the virtual function tables is clear!

```
class One {
public:
     void f1() { f3(); cout << "One::f1" << endl; }
     virtual void f2() { cout << "One::f2" << endl; }
     void f3() { cout << "One::f3" << endl; }
};


class Two: public One {
public:
     void f4() { cout << "Two::f4" << endl; }
     void f2() { f1(); cout << "Two::f2" << endl; }
     virtual void f3() { f4(); cout << "Two::f3" << endl; }
};
class Three: public Two {
public:
     void f3() { f2(); cout << "Three::f3" << endl; }
     void f1() { cout << "Three::f1" << endl; }
};
```



Inheritance

- Now, for each of the following sequences of code, assume that we try to run the program with the given lines of code replacing the empty box in main. Either write the output that is produced when that program is executed, or, if an error occurs, give a concise description of the problem.

```
One *x = new Two();
x->f1();
```

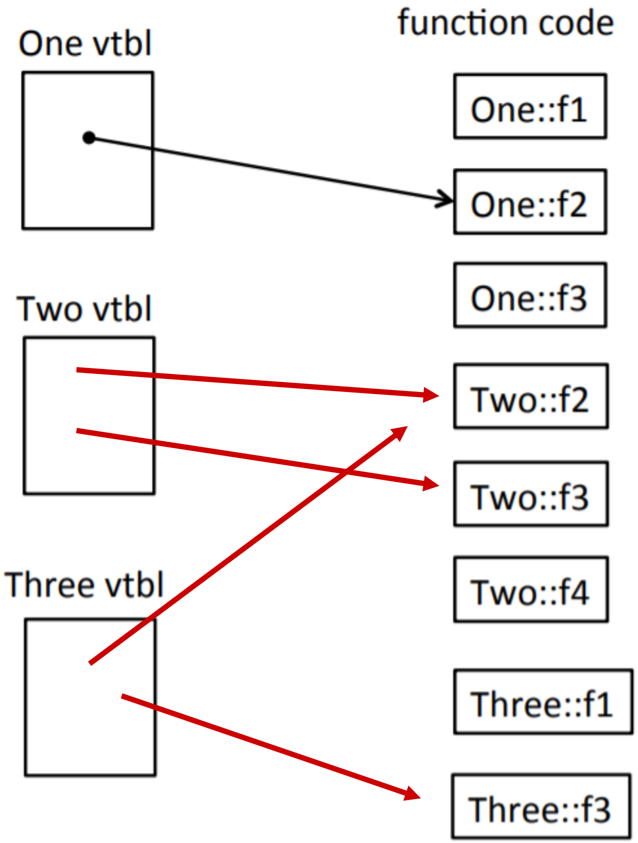
One::f3
One::f1

```
One *x = new Two();
x->f3();
```

One::f3

```
Two *x = new Two();
x->f3();
```

Two::f4
Two::f3



Inheritance

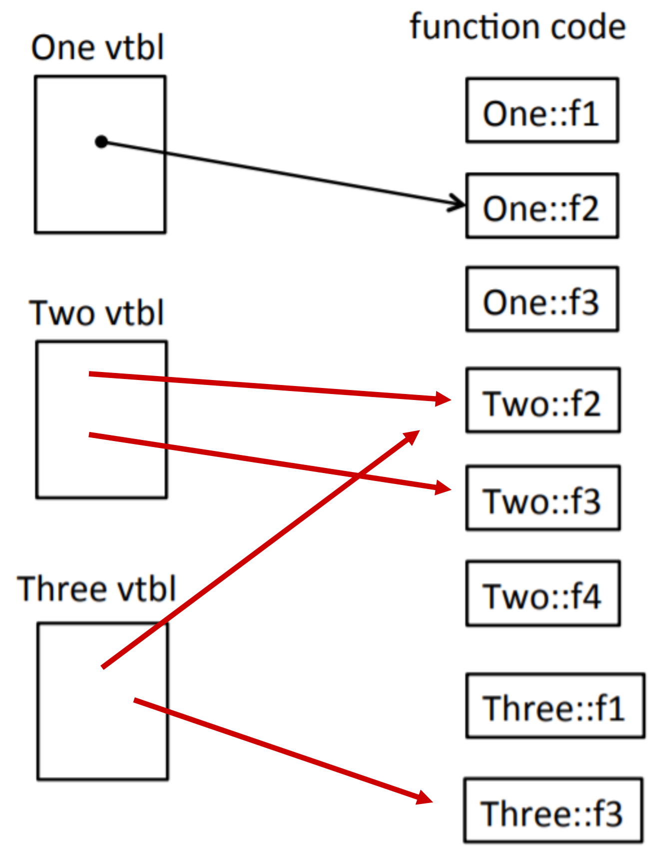
- Now, for each of the following sequences of code, assume that we try to run the program with the given lines of code replacing the empty box in main. Either write the output that is produced when that program is executed, or, if an error occurs, give a concise description of the problem.

```
One *x = new Three();
x->f4();
```

compiler error

```
Three *x = new Three();
x->f3();
```

One::f3
One::f1
Two::f2
Three::f3



Inheritance

- Show the output produced after changing all of the member functions in the classes to be virtual and then replacing the empty box in main with each of the following sequences of code. Either write the output that is produced when the program is executed, or, if an error occurs, give a concise description of the problem.

```
One *x = new Two();
x->f1();
Two::f4
Two::f3
One::f1
```

```
One *x = new Two();
x->f3();
Two::f4
Two::f3
```

Inheritance

- Show the output produced after changing all of the member functions in the classes to be virtual and then replacing the empty box in main with each of the following sequences of code. Either write the output that is produced when the program is executed, or, if an error occurs, give a concise description of the problem.

```
Two *x = new Two();
x->f3();
```

Two::f4
Two::f3

```
One *x = new Three();
x->f4();
```

compiler error

```
Three *x = new Three();
x->f3();
```

Three::f1
Two::f2
Three::f3

C++ Copying

- Below is a class that represents a Multiple Choice answer

```
class MC {
 public:
  MC() : resp_(' ') { }
  MC(char resp) : resp_(resp) { }
  char get_resp() const { return resp_; }
  bool Compare(MC mc) const;
 private:
  char resp_;
}; // class MC
```

- How many times are each of the following invoked:
  - MC constructor
  - MC copy constructor
  - MC operator=
  - MC destructor

```
int QS 2
// this works
MC key[2] = {'D', 'A'};

size_t Score(const MC *ans) {
  size_t score = 0;
  for (int i = 0; i < QS; i++) {
    if (ans->Compare(key[i])) {
      score++;
    }
    ans++;
  }
  return score;
}
int main(int argc, char **argv) {
  MC myAns[QS];
  myAns[0] = MC('B');
  myAns[1] = MC('A');
  cout << "Score: ";
  cout << Score(myAns) << endl;
  return 0;
}
```

C++ Copying

- Below is a class that represents a Multiple Choice answer

```
class MC {
 public:
  MC() : resp_(' ') { }
  MC(char resp) : resp_(resp) { }
  char get_resp() const { return resp_; }
  bool Compare(MC mc) const;
 private:
  char resp_;
}; // class MC
```

- How many times are each of the following invoked:
  - MC constructor
  - MC copy constructor
  - MC operator=
  - MC destructor

```
int QS 2
// this works
MC key[2] = {'D', 'A'}; // ctor x2

size_t Score(const MC *ans) {
  size_t score = 0;
  for (int i = 0; i < QS; i++) {
    if (ans->Compare(key[i])) {
      score++;
    }
    ans++;
  }
  return score;
}
int main(int argc, char **argv) {
  MC myAns[QS];
  myAns[0] = MC('B');
  myAns[1] = MC('A');
  cout << "Score: ";
  cout << Score(myAns) << endl;
  return 0;
}
```

C++ Copying

- Below is a class that represents a Multiple Choice answer

```
class MC {
 public:
  MC() : resp_(' ') { }
  MC(char resp) : resp_(resp) { }
  char get_resp() const { return resp_; }
  bool Compare(MC mc) const;
 private:
  char resp_;
}; // class MC
```

- How many times are each of the following invoked:
  - MC constructor
  - MC copy constructor
  - MC operator=
  - MC destructor

```
int QS 2
// this works
MC key[2] = {'D', 'A'}; // ctor x2

size_t Score(const MC *ans) {
  size_t score = 0;
  for (int i = 0; i < QS; i++) {
    if (ans->Compare(key[i])) {
      score++;
    }
    ans++;
  }
  return score;
}
int main(int argc, char **argv) {
  MC myAns[QS];  // defulat ctor x2
  myAns[0] = MC('B');
  myAns[1] = MC('A');
  cout << "Score: ";
  cout << Score(myAns) << endl;
  return 0;
}
```

C++ Copying

- Below is a class that represents a Multiple Choice answer

```
class MC {
 public:
  MC() : resp_(' ') { }
  MC(char resp) : resp_(resp) { }
  char get_resp() const { return resp_; }
  bool Compare(MC mc) const;
 private:
  char resp_;
}; // class MC
```

- How many times are each of the following invoked:
  - MC constructor
  - MC copy constructor
  - MC operator=
  - MC destructor

```
int QS 2
// this works
MC key[2] = {'D', 'A'}; // ctor x2

size_t Score(const MC *ans) {
  size_t score = 0;
  for (int i = 0; i < QS; i++) {
    if (ans->Compare(key[i])) {
      score++;
    } // cctor in loop 2x for param
    ans++;
  }
  return score;
}
int main(int argc, char **argv) {
  MC myAns[QS];  // defulat ctor x2
  myAns[0] = MC('B'); // ctor then =
  myAns[1] = MC('A'); // ctor then =
  cout << "Score: ";
  cout << Score(myAns) << endl;
  return 0;
}
```

C++ Copying

- Below is a class that represents a Multiple Choice answer

```
class MC {
 public:
  MC() : resp_(' ') { }
  MC(char resp) : resp_(resp) { }
  char get_resp() const { return resp_; }
  bool Compare(MC mc) const;
 private:
  char resp_;
}; // class MC
```

- How many times are each of the following invoked:
  - MC constructor
  - MC copy constructor
  - MC operator=
  - MC destructor

```
int QS 2
// this works
MC key[2] = {'D', 'A'}; // ctor x2

size_t Score(const MC *ans) {
  size_t score = 0;
  for (int i = 0; i < QS; i++) {
    if (ans->Compare(key[i])) {
      score++;
    }
    ans++;
  }
  return score;
}
int main(int argc, char **argv) {
  MC myAns[QS];  // defulat ctor x2
  myAns[0] = MC('B'); // ctor then =
  myAns[1] = MC('A'); // ctor then =
  cout << "Score: ";
  cout << Score(myAns) << endl;
  return 0;
}
```

C++ Copying

- Below is a class that represents a Multiple Choice answer

```
class MC {
 public:
  MC() : resp_(' ') { }
  MC(char resp) : resp_(resp) { }
  char get_resp() const { return resp_; }
  bool Compare(MC mc) const;
 private:
  char resp_;
}; // class MC
```

- How many times are each of the following invoked:
  - MC constructor
  - MC copy constructor
  - MC operator=
  - MC destructor

```
int QS 2
// this works
MC key[2] = {'D', 'A'}; // ctor x2

size_t Score(const MC *ans) {
  size_t score = 0;
  for (int i = 0; i < QS; i++) {
    if (ans->Compare(key[i])) {
      score++;
    } // cctor in loop 2x for param
    ans++;
  }
  return score;
}
int main(int argc, char **argv) {
  MC myAns[QS];  // defulat ctor x2
  myAns[0] = MC('B'); // ctor then =
  myAns[1] = MC('A'); // ctor then =
  cout << "Score: ";
  cout << Score(myAns) << endl;
  return 0;
}
```

C++ Copying

- Below is a class that represents a Multiple Choice answer

```
class MC {
 public:
  MC() : resp_(' ') { }
  MC(char resp) : resp_(resp) { }
  char get_resp() const { return resp_; }
  bool Compare(MC mc) const;
 private:
  char resp_;
}; // class MC
```

```
int QS 2
// this works
MC key[2] = {'D', 'A'}; // ctor x2

size_t Score(const MC *ans) {
  size_t score = 0;
  for (int i = 0; i < QS; i++) {
    if (ans->Compare(key[i])) {
      score++;
    } // cctor in loop 2x for param
    ans++;
  }
  return score;
}
int main(int argc, char **argv) {
  MC myAns[QS];  // defulat ctor x2
  myAns[0] = MC('B'); // ctor then =
  myAns[1] = MC('A'); // ctor then =
  cout << "Score: ";
  cout << Score(myAns) << endl;
  return 0;
}
```

- How many times are each of the following invoked:
  - MC constructor 6
  - MC copy constructor 2
  - MC operator= 2
  - MC destructor 8