ECE 39595C Fall 2020, Test 3

The answer sheet is a separate sheet and can be edited, and therefore annotated with your answers.

The exam is 15 pages long with 39 questions worth 2.5 points. You get 2.5 points for free.

The first page is the answer sheet. You can annotate it and turn it in, you can turn in the entire exam with the answer sheet annotated, you can answer on a piece of scratch paper and turn in a .pdf (preferably) or as a .jpg.

You may begin the exam whenever it becomes available. I will give a 10 minute warning – at the end of that the exam answer sheet needs to have already been uploaded to Brightspace.

If you are not in zoom with video turned on you may recieve a 0 on the exam. I will be recording the exam. Check the Zoom chat box periodically for corrections.

Programs are be given without import statements for brevity. Assume all needed imports are present. You may use newlines in your answer, or not, without affecting your score.

This page has questions 1 2, below.

```
class Err {
protected:
   std::string msg;
public:
   Err(std::string _msg);
   friend std::ostream& operator<<(
      std::ostream& os, const Err& e);
};

Err::Err(std::string _msg) {
   msg = _msg;
}

std::ostream& operator<<(std::ostream& os,
                         const Err& e) {
   return os << "Err: " << e.msg;
}

class MyException : public Err {
public:
   MyException(std::string msg);
   friend std::ostream& operator<<(
      std::ostream& os, const MyException& e);
};

MyException::MyException(std::string msg) : Err(msg) { }

ostream& operator<<(ostream& os, const MyException& e) {
   return os << "My: " << e.msg;
}
```

```
void f(int j) {
   if (j==0) {
      throw MyException("0");
   }
   if (j == 1) {
      throw Err("1");
   }
   if (j == 2) {
      throw 2;
   }
}

int main( ) {
   try {
      f(0);
   } catch (Err eobj) {
      std::cout << eobj << std::endl; // S1
   } catch (MyException meobj) {
      std::cout << meobj << std::endl; // S2
   }
}
```

Q1. For S1, say what it prints if S1 executes, and "nothing" if it is not executed.

Q2. For S2, say what it prints if S2 executes, and "nothing" if it is not executed.

This page contains questions 3 and 4. All code except for the main function is identical to that on the previous page.

```
class Err {
protected:
   std::string msg;
public:
   Err(std::string _msg);
   friend std::ostream& operator<<(
      std::ostream& os, const Err& e);
};

Err::Err(std::string _msg) {
   msg = _msg;
}

std::ostream& operator<<(std::ostream& os,
                         const Err& e) {
   return os << "Err: " << e.msg;
}

class MyException : public Err {
public:
   MyException(std::string msg);
   friend std::ostream& operator<<(
      std::ostream& os, const MyException& e);
};

MyException::MyException(std::string msg) : Err(msg) { }

ostream& operator<<(ostream& os, const MyException& e) {
   return os << "My: " << e.msg;
}
```

```
void f(int j) {
   if (j==0) {
      throw MyException("0");
   }
   if (j == 1) {
      throw Err("1");
   }
   if (j == 2) {
      throw 2;
   }
}

int main( ) {
   try {
      for (int i = 0; i < 2; i++) {
         f(i);
      }
   } catch (MyException meobj) {
      std::cout << meobj << std::endl; // S1
   } catch (Err eobj) {
      std::cout << eobj << std::endl; // S2
   }
}
```

Q3. For S1, say what it prints if S1 executes, and "nothing" if it is not executed.

Q4. For S2, say what it prints if S2 executes, and "nothing" if it is not executed.

This page contains questions 5 and 6. All code except for the main function is identical to that on the previous page.

```
class Err {
protected:
   std::string msg;
public:
   Err(std::string _msg);
   friend std::ostream& operator<<(
      std::ostream& os, const Err& e);
};

Err::Err(std::string _msg) {
   msg = _msg;
}

std::ostream& operator<<(std::ostream& os,
                         const Err& e) {
   return os << "Err: " << e.msg;
}

class MyException : public Err {
public:
   MyException(std::string msg);
   friend std::ostream& operator<<(
      std::ostream& os, const MyException& e);
};

MyException::MyException(std::string msg) : Err(msg) { }

ostream& operator<<(ostream& os, const MyException& e) {
   return os << "My: " << e.msg;
}
```

```
void f(int j) {
   if (j==0) {
      throw MyException("0");
   }
   if (j == 1) {
      throw Err("1");
   }
   if (j == 2) {
      throw 2;
   }
}

int main( ) {
   for (int i = 0; i < 2; i++) {
      try {
         f(i);
      } catch (MyException meobj) {
         std::cout << meobj << std::endl; // S1
      } catch (Err eobj) {
         std::cout << eobj << std::endl; // S2
      }
   }
}
```

Q5. For S1, say what it prints if S1 executes, and "nothing" if it is not executed.

Q6. For S2, say what it prints if S2 executes, and "nothing" if it is not executed.

This page contains question 7. All code except for the main function is identical to that on the previous page.

```
class Err {
protected:
   std::string msg;
public:
   Err(std::string _msg);
   friend std::ostream& operator<<(
      std::ostream& os, const Err& e);
};

Err::Err(std::string _msg) {
   msg = _msg;
}

std::ostream& operator<<(std::ostream& os,
                         const Err& e) {
   return os << "Err: " << e.msg;
}

class MyException : public Err {
public:
   MyException(std::string msg);
   friend std::ostream& operator<<(
      std::ostream& os, const MyException& e);
};

MyException::MyException(std::string msg) : Err(msg) { }

ostream& operator<<(ostream& os, const MyException& e) {
   return os << "My: " << e.msg;
}
```

```
void f(int j) {
   if (j==0) {
      throw MyException("0");
   }
   if (j == 1) {
      throw Err("1");
   }
   if (j == 2) {
      throw 2;
   }
}

int main( ) {
   try {
      f(2);
   } catch (MyException meobj) {
      std::cout << "caught MyException\n";
   } catch (Err eobj) {
      std::cout << "caught Err" << std::endl;
   }
   std::cout << "done." << std::endl; // S1
}
```

Q7. Does S1 execute?

This page has questions 8 – 10, below. If something is printed by a line that is a question (has a Qx comment, where "x" is a natural number) say what is printed. If the line has an error at either compile or runtime, answer "Err" and assume the statement doesn't exist when answering other questions. If the statement prints nothing but is correct, answer "Ok". If a value is unitialized, answer "uninit".

```
class Base {
public:
   int val;
   Base(int _val);
   virtual void foo( );
};

Base::Base(int _val) : val(_val) { }

void Base::foo( ) {
   std::cout << "B::foo" << std::endl;
}

class D1 : public Base {
public:
   D1(int val);
};

D1::D1(int val) : Base(val) { }

class D2 : public Base {
public:
   D2(int val);
};

D2::D2(int val) : Base(val) { }

class Derived : public D1, public D2 {
public:
   Derived(int val);
};

Derived::Derived(int val) : D1(val), D2(-val) { }

int main(int argc, char** args) {
   Base b(0);
   D1 d1(10);
   D2 d2(20);
   Derived d(30);
   std::cout << b.val << std::endl; // Q8
   std::cout << d1.val << std::endl; // Q9
   std::cout << d.D2::val << std::endl; // Q10
}
```

This page has questions 11 – 13, below. Hint: note the presence of "virtual" on D1's and D2's inheritance of "Base".

If something is printed by a line that is a question (has a Qx comment, where "x" is a natural number) say what is printed. If the line has an error at either compile or runtime, answer "Err" and assume the statement doesn't exist when answering other questions. If the statement prints nothing but is correct, answer "Ok". If a value is unitialized, answer "uninit".

```
class Base {
public:
   int val;
   Base(int _val);
   virtual void foo( );
};

Base::Base(int _val) : val(_val) { }

void Base::foo( ) {
   std::cout << "B::foo" << std::endl;
}

class D1 : virtual public Base {
public:
   D1(int val);
};

D1::D1(int val) : Base(-val) { }

class D2 : virtual public Base {
public:
   D2(int val);
};

D2::D2(int val) : Base(-val) { }

class Derived : public D1, public D2 {
public:
   Derived(int val);
};

Derived::Derived(int val) : Base(val), D1(val), D2(val) { }

int main(int argc, char** args) {
   Base b(0);
   D1 d1(10);
   D2 d2(20);
   Derived d(30);
   std::cout << b.val << std::endl; // Q11
   std::cout << d1.val << std::endl; // Q12
   std::cout << d.D2::val << std::endl; // Q13
}
```

This page has questions 14 through 17, below.

```
const int MAXTHREADS = 2;
std::mutex m2;
int globalCount = 0;
std::thread* threads[MAXTHREADS];

void run1(int iters) {
   for (int i = 0; i < iters; i++) {
      std::lock_guard<std::mutex> lk(m2);
      int j = globalCount;
      usleep(10);
      globalCount = j+1;
   }
}

void run2(int iters) {
   std::mutex m1;
   for (int i = 0; i < iters; i++) {
      std::lock_guard<std::mutex> lk(m1);
      int j = globalCount;
      usleep(10);
      globalCount = j+1;
   }
}

void run3(int iters) {
   for (int i = 0; i < iters; i++) {
      int j = globalCount;
      usleep(10);
      globalCount = j+1;
   }
}

void joinAndReset(int run) {
   for (int threadNum = 0;
        threadNum < MAXTHREADS;
        threadNum++) {
      threads[threadNum]->join( );
   }
   std::cout << "run" << run << ": ";
   st d::cout << globalCount << std::endl;
   globalCount = 0;
}
```

Q14. When S1 executes, the value of globalCount printed is:

1. less than 2000
2. equal to 2000
3. less than or equal to 2000

```
int main(int argc, char *argv[]) {

   int numIters = 1000;

   for (int threadNum = 0; threadNum < MAXTHREADS;
        threadNum++) {
      threads[threadNum] = new std::thread(run1, numIters);
   }
   joinAndReset(1); // S1

   for (int threadNum = 0; threadNum < MAXTHREADS;
        threadNum++) {
      threads[threadNum] = new std::thread(run2, numIters);
   }
   joinAndReset(2); // S2

   for (int threadNum = 0; threadNum < MAXTHREADS;
        threadNum++) {
      threads[threadNum] = new std::thread(run3, numIters);
   }
   joinAndReset(3); // S3

   for (int threadNum = 0; threadNum < MAXTHREADS;
        threadNum++) {
      threads[threadNum] = new std::thread(run3, numIters);
      threads[threadNum]->join( );
   }
   std::cout << "run4: " << globalCount << std::endl; // S4
}
```

Q15. When S2 executes, the value of globalCount printed is:

1. less than 2000
2. equal to 2000
3. less than or equal to 2000

Q16. When S3 executes, the value of globalCount printed is:

1. less than 2000
2. equal to 2000
3. less than or equal to 2000

Q17. When S4 executes, the value of globalCount printed is:

1. less than 2000
2. equal to 2000
3. less than or equal to 2000

This page has Program A and question 18. `compare(std::string)` returns 0 if the strings are equal.

```
class Observer {
public:
   virtual void update(int i)=0;
};

class Subject {
   virtual void subscribe(Observer* o)=0;
   virtual void remove(Observer* o)=0;
};

class Buss : public Subject {
private:
   std::vector<Observer*> observers;
public:
   Buss( );
   virtual void subscribe(Observer* o);
   virtual void remove(Observer* o);
   virtual void notify(int i) const;
   virtual void putDataOnBuss(int i) const;
};

Buss::Buss( ) {
  observers = std::vector<Observer*>( );
}

void Buss::subscribe(Observer* o) {
   observers.push_back(o);
}

void Buss::remove(Observer* o) {
   std::vector<Observer*>::iterator it =
   std::find(observers.begin( ),
             observers.end( ), o);
   if (it != observers.end( ))
      observers.erase(it);
}

void Buss::notify(int i) const {
   for (Observer* o : observers) {
      o->update(i);
   }
}

void Buss::putDataOnBuss(int i) const {
   notify(i);
}

class OutputPort : public Observer {
private:
    int portId = -1;
public:
   OutputPort(int i);
   virtual void update(int i);
};

OutputPort::OutputPort(int i) {
  portId = i;
}

void OutputPort::update(int i) {
   std::cout << "Output port " << portId;
   std::cout << " receives " << i << std::endl;
}

void writeData(const Buss& buss,
               int low, int high) {
   for (int i = low; i < high; i++) {
      buss.putDataOnBuss(i);
   }
}

int main(int argc, char** args) {
   Buss buss;
   OutputPort* port1 = new OutputPort(1);
   OutputPort* port2 = new OutputPort(2);
   buss.subscribe(port1);
   buss.subscribe(port2);
   writeData(buss, 10, 12);

   buss.remove(port2);
   writeData(buss, 8, 10);

   buss.subscribe(new OutputPort(3));
   writeData(buss, 6, 8);
}
```

Q18. What is the name of the pattern used in this program?

This page has Program B and question 19, below. `compare(std::string) returns 0 if the strings are equal.`

```
class OutputPort {
private:
    int portId = -1;
public:
   OutputPort(int i);
   virtual void write(int i);
};

OutputPort::OutputPort(int i) {
  portId = i;
}

void OutputPort::write(int i) {
   std::cout << "Output port " << portId << " receives " << i << std::endl;
}

class Buss {
private:
   std::vector<OutputPort> ports;
   int len = -1;
public:
   Buss(int len, int portIds[ ]);
   virtual void notify(int i);
   virtual void putDataOnBuss(int i);
};

Buss::Buss(int len, int portIds[ ]) {
   ports = std::vector<OutputPort>( );
   for (int i = 0; i < len; i++) {
      ports.push_back(OutputPort(i));
   }
}

void Buss::notify(int i)  {
   for (OutputPort p : ports) {
      p.write(i);
   }
}

void Buss::putDataOnBuss(int i) {
   notify(i);
}
```

```
int main(int argc, char** args) {
   int portIds[ ] = {1, 3, 2};
   Buss buss = Buss(3, portIds);

   buss.putDataOnBuss(6);
   buss.putDataOnBuss(7);
}
```

Q19. Let it be true that the Buss will never have Output ports added or removed. We know that patterns often add complexity. Given these, should Program A or Program B be used in this case?

This page has Program C and questions 20 and 21, below. `compare(std::string) returns 0 if the strings are equal.`

```
class Disk {
private:
   std::string diskType;
public:
    Disk(std::string type);
    virtual std::string type( );
};

Disk::Disk(std::string type) {
   diskType = type;
}

std::string Disk::type( ) {
   return diskType;
}

class SSDDrive : public Disk {
public:
    SSDDrive( );
};

SSDDrive::SSDDrive( ) : Disk("SSD") { }

class SpinningDrive : public Disk {
public:
    SpinningDrive( );
};

SpinningDrive::SpinningDrive( ) :
   Disk("Spinning") { }

class Computer {
private:
   Disk* disk = nullptr;
public:
   Computer( );
   Computer(Disk* d);
   virtual void changeDisk(Disk* d);
   virtual std::string readDisk(int  addr);
};
```

```
Computer::Computer( ) {
    disk = nullptr;
}

Computer::Computer(Disk* d) {
    disk = d;
}

void Computer::changeDisk(Disk* d) {
   disk = d;
}

std::string Computer::readDisk(int  addr) {
   std::string str = std::string("reading addr ");
   str += std::to_string(addr);
   str += std::string(" on ")+disk->type( );
   str += std::string(" disk");
   return str;
}

int main(int argc, char** args) {
   Computer c;
   for (int i = 1; i < argc; i++) {
      std::string diskType = args[i];
      if (diskType.compare("ssd") == 0) {
         c = Computer(new SSDDrive( ));
      } else if (diskType.compare("spin") == 0) {
         c = Computer(new SpinningDrive( ));
      }
      std::cout << "read disk: " << c.readDisk(i);
   }
}
```

Q20. What pattern is used in Program C?

Q21. What classes would need to be changed if a new type of Disk is added? Answer with the letters for all changed classes.

A. Disk

B. SSDDrive

C. SpinningDrive

D. Computer

E. the main function

This page has Program D and questions 22 and 23, below. `compare(std::string) returns 0 if the strings are equal.`

```
class Disk {
private:
   std::string diskType;
public:
    Disk(std::string type);
    virtual std::string type( );
};

Disk::Disk(std::string type) {
   diskType = type;
}

std::string Disk::type( ) {
   return diskType;
}

class SSDDrive : public Disk {
public:
    SSDDrive( );
};

SSDDrive::SSDDrive( ) : Disk("SSD") { }

class SpinningDrive : public Disk {
public:
    SpinningDrive( );
};

SpinningDrive::SpinningDrive( ) :
               Disk("Spinning") { }

class Computer {
private:
   Disk* disk = nullptr;
public:
   Computer( );
   Computer(Disk* d);
   virtual void changeDisk(Disk* d);
   virtual std::string readDisk(int  addr);
};
```

```
Computer::Computer( ) {
    disk = nullptr;
}

Computer::Computer(Disk* d) {
    disk = d;
}

void Computer::changeDisk(Disk* d) {
   disk = d;
}

std::string Computer::readDisk(int  addr) {
   std::string str = std::string("reading addr ");
   str += std::to_string(addr);
   str += std::string(" on ")+disk->type( );
   str += std::string(" disk");
   return str;
}

class ComputerBuilder {
public:
    static Computer build(std::string driveType);
};

Computer ComputerBuilder::build(
   1std::string driveType) {
   Computer c;
   if (driveType.compare("ssd") == 0) {
         c = Computer(new SSDDrive( ));
   } else if (driveType.compare("spin") == 0) {
         c = Computer(new SpinningDrive( ));
   }
   return c;
}

int main(int argc, char** args) {
   for (int i = 1; i < argc; i++) {
      Computer c = ComputerBuilder::build(args[i]);
      std::cout << "reading: ";
      std::cout << c.readDisk(i) << std::endl;
   }
}
```

Q22. What pattern is used in Program D?

Q23. What classes would need to be changed if a new type of Disk is added?

A. Disk
B. SSDDrive
C. SpinningDrive
D. Computer
E. ComputerBuilder
F. the main function

This page has questions 24 – 29, below.

```
template <typename T, typename U>
   class Pair {
private:
   T data1;
   U data2;
public:
   Pair(T t, U u);
   bool operator<(const Pair<T,U>& p);
   friend std::ostream operator<<(
      std::ostream& os,
      const Pair<T, U>& p) {
      os << "(" << p.data1 << ", ";
      os << p.data2 << ")";
      return os;
   }
};

template <typename T, typename U>
   Pair<T, U>::Pair(const T t, const U u) :
       data1(t), data2(u) { }

template <typename T, typename U>
   bool Pair<T, U>::operator<(
        const Pair<T, U>& p) {
   if (data1 < p.data2) return true;
   return ((data1 == p.data1) &&
           (data2 < p.data2));
}

class Obj {
private:
   int data;
public:
   Obj(int);
   virtual bool operator<(const Obj& obj);
   virtual bool operator<(int i);
   virtual bool operator==(const Obj& obj);
   virtual bool operator==(int i);
};
```

```
Obj::Obj(int i) : data(i) { }

bool Obj::operator<(const Obj& obj) {
   return data < obj.data;
}

bool Obj::operator<(int i) {
   return data < i;
}

bool Obj::operator==(const Obj& obj) {
   return data == obj.data;
}

bool Obj::operator==(int i) {
   return data == i;
}

class C {
public:
   C( );
};

C::C( ) { }

int main(int argc, char** args) {
   Pair<int,int> ipair1 = Pair<int,int>(1,1);
   Pair<int,int> ipair2 = Pair<int,int>(2,2);
   std::cout << (ipair1 < ipair2) << std::endl; // Q24

   Pair<Obj,Obj> opair1 = Pair<Obj,Obj>(Obj(2),Obj(1));
   Pair<Obj,Obj> opair2 = Pair<Obj,Obj>(Obj(1),Obj(2));
   std::cout << (opair1 < opair2) << std::endl;

   Pair<Obj,int> oipair1 = Pair<Obj,int>(Obj(1),1); // S1
   Pair<Obj,int> oipair2 = Pair<Obj,int>(Obj(2),2);
   std::cout << (oipair1 < oipair2) << std::endl; // Q25

   Pair<C,C> ccpair1 = Pair<C,C>(C( ),C( ));
   Pair<C,C> ccpair2 = Pair<C,C>(C( ),C( ));
   std::cout << (ccpair1 < ccpair2) << std::endl; // Q26
}
```

Q27. What is the type of data1 in the Pair at S1?

Q28. What is the type of data2 in the Pair at S1?

Q29. What function or operation is used for "<" at the statement of Q25? Give the function name and arguments (e.g., `foo(int, int)` if a function, and say "primitive" if it is a built-in "<" operation.

This page and the next has questions 30 to 39, below.

```
class Base {
public:
   static void fstat( );
   virtual void f1( );
   void f3( );
   virtual void f4( );
private:
   virtual void f5( );
};

void Base::fstat( ) {
   std::cout << "B::fs" << std::endl;
}

void Base::f1( ) {
   std::cout << "B::f1" << std::endl;
}

void Base::f3( ) {
   std::cout << "B::f3" << std::endl;
}
void Base::f4( ) {
   std::cout << "B::f4 ";
   f5( );
}

void Base::f5( ) {
   std::cout << "B::f5" << std::endl;
}

class Derived : public Base {
public:
   static void fstat( );
   virtual void f1( );
   virtual void f2( );
   void f3( );
private:
   virtual void f5( );
};
```

```
void Derived::fstat( ) {
   std::cout << "D::fs" << std::endl;
}

void Derived::f1( ) {
   std::cout << "D::f1" << std::endl;
}

void Derived::f2( ) {
   std::cout << "D::f2" << std::endl;
}

void Derived::f3( ) {
   std::cout << "D::f3" << std::endl;
}

void Derived::f5( ) {
   std::cout << "D::f5" << std::endl;
}
```

```
// Main.cpp code
void f(int* ip) {
   *ip = 2;
   ip = nullptr;
}

void f(Derived& d) {
   std::cout << "D&&" << std::endl;
}

void f(Derived&& d) {
   std::cout << "D&&" << std::endl;
}

int main(int argc, char** args) {
   Base* b = new Base( );
   Base* bd = new Derived( );

   b->f4( ); // Q30
   bd->fstat( ); // Q31
   bd->f1( ); // Q32
   bd->f2( ); // Q33
   bd->f3( ); // Q34

   f(Derived( )); // Q35
   Derived d;
   f(d); // Q36

   int i = 4; // S1
   int* ip = &i;
   f(ip);
   std::cout << i << std::endl; // Q37
   std::cout << ip << std::endl; // S2
}
```

Q38. In statement S1, which is correct (write the letter for your answer):

A. i is an rvalue and 4 is the lvalue

B. 4 is an rvalue and i is the lvalue

C. i and 4 are lvalues

D. i and 4 are rvalues

Q39. Is ip == nullptr in S2? Answer true or false.