ECE 29595C Fall 2019 First Exam Answer Sheet
Write your name and test version in the space above! It's worth 5 points.

1.	**21.**
2.	**22.**
3.	**23.**
4.	**24.**
5.	**25.**
6.	**26.**
7.	**27.**
8.	**28.**
9.	**29.**
10.	**30.**
11.	**31.**
12.	**32.**
13.	**33.**
14.	**34.**
15.	**35.**
16.	
17.	
18.	
19.	
20.	
21.	
22.	
23.	
24.	

ECE 39595 Java Fall 2019, Test 1

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 "#include" statements for brevity. Assume all needed includes are present. "std::endl" may left off for brevity. You may use newlines in your answer, or not, without affecting your score.

Each question is worth 2.5 points except for the last question which is worth 10 points. You will recieve 5 points for putting your name and test version on the answer sheet.

By taking and turning in a test answer sheet to be graded, you agree that: I have neither given nor received help during this exam from any other person or electronic source (other than my own notes, viewing the exam and using an electronic device to annotate the answer sheet with my answers), and I understand that if I have I will be guilty of cheating and will fail the exam and possibly the course.

The code on this page and the facing page are used for questions 1 - 8. If something is printed on 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 for the rest of the program. If the statement prints nothing but is correct, answer ”Ok”. If a value is uninitialized, answer ”uninit”.

```
class Base {
public:
   Base(int);
   Base( );
   virtual ~Base( );
};

Base::Base(int i) {
   std::cout << "B(i)" << std::endl;
}

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

Base::~Base( ) {
   std::cout << "B dtor" << std::endl;
}

class Derived : public Base {
public:
   Derived(int, int);
   Derived(int);
   Derived( );
   virtual ~Derived( );
};

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

Derived::Derived(int i) : Base(i) {
   std::cout << "D(i)" << std::endl;
}

Derived::Derived(int i, int j) : Base( ) {
   std::cout << "D(i, i)" << std::endl;
}

Derived::~Derived( ) {
   std::cout << "D dtor" << std::endl;
}
```

```
int main(int argc, char* argv[ ]) {
   Derived d1; // Q1
   Derived d2(4); // Q2
   Derived* d3 = new Derived(0, 1); // Q3
   Base* b1 = new Base( ); // Q4
   Base b2(3);  // Q5
   delete d3; // Q6
   delete b1; // Q7
   // Q8 Answer what is printed before main
   // exits, or "nothing if nothing is
   // printed.
}
```

The code on this page and the facing page are used for questions 9 - 20. If something is printed on 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 for the rest of the program. If the statement prints nothing but is correct, answer ”Ok”. If a value is uninitialized, answer ”uninit”.

```
class Base {
public:
   Base(int, int);
   void m0( );
   virtual void m1( );
   void m2( );
   virtual void m3( );
   void m4( );
   int i;
private:
   virtual void m5( );
   int j;
};

Base::Base(int i, int j) {
   this->i = i;
   this->j = j;
}

void Base::m0( ) {
   std::cout << "Bm0" << std::endl;
}

void Base::m1( ) {
   std::cout << "Bm1" << std::endl;
}

void Base::m2( ) {
   std::cout << "Bm2" << std::endl;
}

void Base::m3( ) {
   std::cout << "Bm3" << std::endl;
}

void Base::m4( ) {
   std::cout << "Bm4" << std::endl;
}

void Base::m5( ) {
   std::cout << "Bm5" << std::endl;
}
```

```
class Derived : public Base {
public:
   Derived(int, int, int);
   void m0( ); // m3
   void m1( ); // m4
   void m2( ); // m5
   void m3( ); // m6
   virtual void m6( ); // m2
   int i;
   int k;
private:
   void m5( ); // m0
};
Derived:: Derived(int i, int j, int _k) :
   Base(i, j), k(_k) {
   this->i = i*100;
}

void Derived::m0( ) {
   std::cout << "Dm0" << std::endl;
}

void Derived::m1( ) {
   std::cout << "Dm1" << std::endl;
}

void Derived::m2( ) {
   std::cout << "Dm2" << std::endl;
}

void Derived::m3( ) {
   std::cout << "Dm3" << std::endl;
}

void Derived::m5( ) {
   std::cout << "Dm5" << std::endl;
}

void Derived::m6( ) {
   std::cout << "Dm6" << std::endl;
}
```

```
int main(int argc, char* argv[ ]) {
   Base* bp = new Base(0, 1);
   bp->m0( );  // Q9
   bp->m1( );  // Q10
   bp->m5( );  // Q11
   bp->m6( );  // Q12
   std::cout << bp->i << std::endl;  // Q13

   bp = new Derived(2, 3, 4);
   bp->m0( );  // Q14
   bp->m1( );  // Q15
   bp->m2( );  // Q16
   bp->m3( );  // Q17
   bp->m4( );  // Q18
   std::cout << bp->i << std::endl;  // Q19

   Derived* d = (Derived*) bp; // this is legal
   d->m6( ); // Q20
}
```

The code on this page and the facing page are used for questions 21 - 34. If something is printed on 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 for the rest of the program. If the statement prints nothing but is correct, answer ”Ok”. If a value is uninitialized, answer ”uninit”.

```
class Base {
public:
   Base(int, int);
   void m0( );
   virtual void m1( );
   void m2( );
   int i;
private:
   void m3( );
   int j;
};

Base::Base(int i, int j) {
   this->i = i;
   this->j = j;
}

void Base::m0( ) {
   std::cout << "Bm0" << std::endl;
}

void Base::m1( ) {
   std::cout << "Bm1" << std::endl;
}

void Base::m2( ) {
   std::cout << "Bm2" << std::endl;
}

void Base::m3( ) {
   std::cout << "Bm3" << std::endl;
}
```

```
class Derived : public Base {
public:
   Derived(int, int, int);
   void m0( );
   void m1( );
   void m2( );
   virtual void m4( );
   int i;
   int k;
private:
   void m3( );
};

Derived:: Derived(int i, int j, int _k) :
   Base(i, j), k(_k) {
   this->i = i*100;
}

void Derived::m0( ) {
   std::cout << "Dm0" << std::endl;
}

void Derived::m1( ) {
   std::cout << "Dm1" << std::endl;
}

void Derived::m2( ) {
   std::cout << "Dm2" << std::endl;
}

void Derived::m3( ) {
   std::cout << "Dm3" << std::endl;
}

void Derived::m4( ) {
   std::cout << "Dm4" << std::endl;
}
```

```
int main(int argc, char* argv[ ]) {
   Base b(0,1);

   Derived d(2, 3, 4);
   Base& bd = d; // Q21
   bd.m0( ); // Q22
   bd.m1( ); // Q23
   bd.m2( ); // Q24
   bd.m3( ); // Q25
   bd.m4( ); // Q26
   std::cout << bd.i << std::endl; // Q27
   std::cout << bd.k << std::endl; // Q28

   Derived d1 = bd; // Q29
   d.m4( ); // Q30
   b = d; // Q31
   b.m0( ); // Q32
   b.m1( ); // Q33

   Base& b2 = b;
   b2 = bd;
   bd.m1( ); // Q34
}
```

Draw the VFT (virtual function table) for the two classes below. You can draw it on your answer sheet. If turning in a scan, and y ou use two sheets, either combine them into a single .pdf document or put them in a directory named your userid, zip it, and turn that in. This is worth 10 points.

```
class B {
public:
   virtual void m0( );
   virtual void m1( );
   virtual void m2( );
   void m3( );
private:
   virtual void m4( );
};

void B::m0( ) {
   std::cout << "m0" << std::endl;
}

void B::m1( ) {
   std::cout << "m1" << std::endl;
}

void B::m2( ) {
   std::cout << "m2" << std::endl;
}

void B::m3( ) {
   std::cout << "m3" << std::endl;
}

void B::m4( ) {
   std::cout << "m4" << std::endl;
}
```

```
class D : public B {
public:
   void m0( );
   virtual void m5( );
   virtual void m6( );

private:
   virtual void m4( );
};

void D::m0( ) {
   std::cout << "m0" << std::endl;
}

void D::m4( ) {
   std::cout << "m2" << std::endl;
}

void D::m5( ) {
   std::cout << "m5" << std::endl;
}

void D::m6( ) {
   std::cout << "m6" << std::endl;
}
```