ECE 39595C Fall 2020 Second Exam Answer Sheet

Write your name in the space above – it is 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.	**36.**
17.	**37.**
18.	**38.**
19.	**39.**
20.	**40.**

This page intentionally left almost blank

ECE 39595 C++ Fall 2020, Test 2

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 and there are 38 questions in the test. 5 points are reserved for putting your name on your 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). I will not show this test to anyone that has not taken it until it and the answer sheet are posted. I understand that if I am found to have done any of these prohibited actions I will be guilty of cheating and will fail the exam and probably the course.

The code on this page and the facing page are used for questions 1 - 11. If something is printed on a line that is a question (has a Qx comment, where "x" is a 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 Stringy {

private:
   std::string stringData;
public:
   Stringy(std::string _data);
   Stringy(const Stringy& old);
   Stringy& operator=(const Stringy& s);
   Stringy& operator+(const Stringy& s) const;
   Stringy& operator+=(const Stringy& s);
   Stringy& operator-(int i) const;
   std::string getData( ) const;
   friend Stringy& operator*(const Stringy& s, int n);
   friend std::ostream& operator<<(
          std::ostream& os, const Stringy& s);
};

Stringy::Stringy(std::string _data) {
   stringData = _data;
}

Stringy::Stringy(const Stringy& old) {
   stringData = old.getData( ) + "a";
}

Stringy& Stringy::operator=(const Stringy& s) {
   stringData = s.getData( );
   Stringy* stringy = new Stringy("return");
   return *stringy;
}

Stringy& Stringy::operator+(const Stringy& s) const {
   Stringy* stringy =
              new Stringy(stringData + s.getData( ));
   return *stringy;
}

Stringy& Stringy::operator+=(const Stringy& s) {
   stringData = stringData + s.getData( );
   return *this;
}

Stringy& Stringy::operator-(int i) const {
   int newLen = (stringData.size( ) - i) + 1;
   Stringy* retVal;
   if (newLen >= 0) {
      retVal = new Stringy(
                 stringData.substr(0, newLen));
   }
   return *retVal;
}

Stringy& operator*(const Stringy& s, int n) {
   std::string tmp = "";
   for (int i = 0; i < n; i++) {
      tmp += s.getData( );
   }
   Stringy* retVal = new Stringy(tmp);
   return *retVal;
}

std::string Stringy::getData( ) const {
   return stringData;
}

std::ostream& operator<<(
         std::ostream& os, const Stringy& s) {
   return os << s.getData( );
}
```

```
int main(int argc, char** args) {
   Stringy a("a");
   Stringy b("b");
   Stringy c("c");
   Stringy s1(a);
   std::cout << s1 << std::endl; // Q1

   a = b = c;
   std::cout << a << std::endl; // Q2
   std::cout << b << std::endl; // Q3

   Stringy d("ddd");
   Stringy e("eee");
   std::cout << d+e << std::endl; // Q4
   std::cout << e-1 << std::endl; // Q5

   Stringy f("f");
   std::cout << f*3 << std::endl; // Q6

   Stringy s2 = d += e;
   std::cout << s2 << std::endl; // Q7
   std::cout << d << std::endl; // Q8
}
```

Q9. In operator+, what does the first const mean? (answer A, B, C or D)

A. the object pointed to by this cannot be changed

B. the object passed in as s cannot be changed

C. the referende cannot be changed, but the referenced object can be

D. the return value must be the same as the the input argument

Q10. In operator+, what does the second const mean? (answer A, B, C or D)

A. the object pointed to by this cannot be changed

B. the object passed in as s cannot be changed

C. the referende cannot be changed, but the referenced object can be

D. the return value must be the same as the the input argument

Q11. In operator+, in the expresson "getData + s.getData( )", what is the + operator?(answer A, B, C or D)

A. the overloaded opertor+ in the Stringy class

B. the std::string concatenation operator

C. C++ is free to pick whatever is convenient under "catch fire" semantics

The code on this page and the facing page are used for questions 12 - 24. Note that questions 12, 13, 14 and 15 are in the functions D2::f5( ) and bar( ). If something is printed on a line that is a question (has a Qx comment, where "x" is a 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:
   static int count;
   Base( );
   ~Base( );
   static void printCount( );
   static void printAll( );
   virtual void f1( );
   void f2( );
   virtual void f3( );
private:
   virtual void f4( );
};

int Base::count = 0;
   Base::Base( ) {
      count++;
   }

   Base::~Base( ) {
      --count;
      std::cout << count << std::endl;
   }

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

   void Base::f2( ) {
      std::cout << "B::f2" << std::endl;
   }

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

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

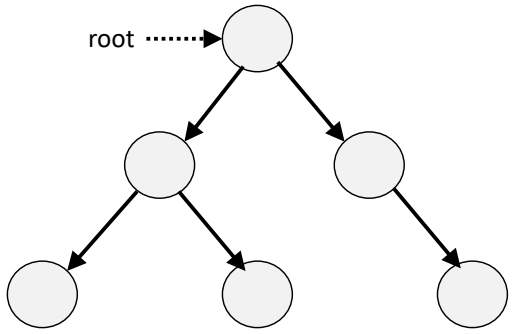
```
class D1 : public Base {
public:
   static int count;
   void f2( );
   virtual void f3( );
   virtual void f5( );
private:
   virtual void f4( );
};

void D1::f2( ) {
   std::cout << "D1::f2" << std::endl;
}

void D1::f3( ) {
   std::cout << "D1::f3" << std::endl;
}

void D1::f4( ) {
   std::cout << "D1::f4" << std::endl;
}

void D1::f5( ) {
   std::cout << "D1::f5" << std::endl;
}
```

```
class D2 : private Base {
public:
   static int sv;
   int v1 = 0;
   void f2( );
   virtual void f3( );
   static void f5( );
   friend void bar(D2 d2);
private:
   int v2 = 0;
};

int D2::sv = 0;

void D2::f2( ) {
   std::cout << "D2::f2" << std::endl;
}

void D2::f3( ) {
   std::cout << "D2::f3" << std::endl;
}

// Q12 & Q13 --  answer "Ok" if legal,
// "Err" if not.
void D2::f5( ) {
   std::cout << v2 << std::endl; // Q12
   std::cout << sv << std::endl; // Q13
}
```

```
// main.cpp
void bar(D2 d2) {
   std::cout << "main " << std::endl;
   std::cout << d2.v1 << std::endl; // Q S14
   std::cout << d2.v2 << std::endl; // Q S15
}

int main(int argc, char** args) {
   Base b;
   Base& bR = b;

   D1 d1;
   Base& d1R = d1;

   bR = d1R;
   bR.f3( ); // Q16
   bR.f5( ); // Q17

   d1R.f1( ); // Q18
   d1R.f2( ); // Q19
   d1R.f3( ); // Q20
   d1R.f4( ); // Q21

   D2 d2;
   D2& d2R = d2;
   d2.f1( ); // Q22
   d2.f3( ); // Q23
}
```

Q24. What, if anything, is printed after the statement for Q23 is finished executing, and before the program terminates?

In the following questions, pick the answer that (a) uses pointers that allow the fewest pointers to the object (i.e, favor unique over shared) and are correct (e.g., no cycles of shared pointers)



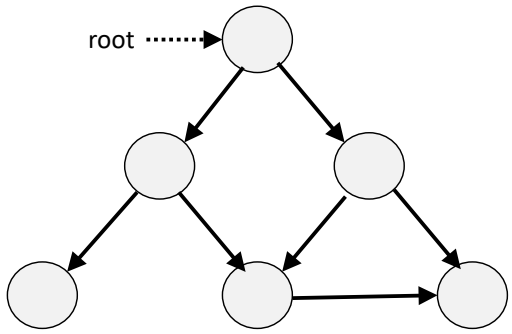
Q25. In the drawing to the left, which is the best types of pointers to use (answer A, B, C or D)

A. the dotted line is a unique_ptr, the solid is a shared_ptr.

B. the dotted and solide lines are shared_ptrs.

C. the dotted and solide lines are unique_ptrs.

D. the dotted line is a weak_ptr, the solid line is a unique_ptr.



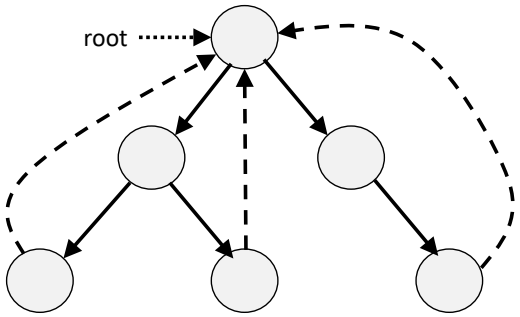
Q26. In the drawing to the left, which is the best types of pointers to use (answer A, B, C or D)

A. the dotted line is a unique_ptr, the solid is a shared_ptr.

B. the dotted and solide lines are shared_ptrs.

C. the dotted and solide lines are unique_ptrs.

D. the dotted line is a weak_ptr, the solid line is a unique_ptr.



Q27. In the drawing to the left, which is the best types of pointers to use (answer A, B or C)

A. the dotted line is a shared_ptr, the dashed line is a weak_ptr, and the solid line is a shared_ptr.

B. the dotted line is a unique_ptr, the dashed line is a unique_ptr, and the solid line is a shared_ptr.

C. the dotted line is a unique_ptr, the dashed line is a shared_ptr, and the solid line is a weakd_ptr.

The code on this page is used for questions 28 - 34. If something is printed on a line that is a question (has a Qx comment, where "x" is a 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:
   static int count;
   Base( );
   virtual void h(double);
   virtual void h(int);
   virtual void h( );
};

int Base::Base::count = 0;;
Base::Base( ) {
   count++;
}

void Base::h(double d) {
   std::cout << "B::hd" << std::endl;
}

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

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

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

void D1::h(int i) {
   std::cout << "D1::hi" << std::endl;
}
```

```
class D2 : public Base {
public:
   virtual void h(int);
   using Base::h;
};

void D2::h(int) {
   std::cout << "D2::hi" << std::endl;
}

int main(int argc, char** args) {

   D1* d1 = new D1( ); // 28
   d1->h(2.0); // 29
   d1->h( ); // 30


   D2* d2 = new D2( ); // 31
   d2->h(1); // 32
   d2->h( ); // 33

   Base b;

   std::cout << b.count << std::endl; // 34
}
```

The code on this page is used for questions 35 - 38. If something is printed on a line that is a question (has a Qx comment, where "x" is a 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 {
   virtual void h( );
};

void Base::h( ) {
   std::cout << "h" << std::endl;
}
class X : public Base {
public:
   virtual void f( );
};

void X::f( ) {
   std::cout << "f" << std::endl;
}

class Y : public Base {

public:
   virtual void g( );
};

void Y::g( ) {
   std::cout << "g" << std::endl;
}
```

```
int main(int argc, char** args) {
   Base* b = new Base( );
   X* x = new X( );
   Y* y = new Y( );

   y = new Y( );
   y = static_cast<Y*>(b); // S1

   y = new Y( );
   y = static_cast<Y*>(x); // S2

   y = new Y( );
   y = dynamic_cast<Y*>(b); // S3
   y->g( ); // S4
}
```

Q35. Does S1 give an error at compile time? (Answer yes or no.)

Q36. Does S2 give an error at compile time? (Answer yes or no.)

Q37. Does S3 give an error at compile time? (Answer yes or no.)

Q38. Does S3 give an error at run time? (Answer yes or no.)