1. Inheritance and Virtual Functions  **20 Points**

What is printed by the program q2 attached? Explain your answer.

*Hint*: There should be 10 “Hello from” outputs.
2. Memory Management **20 Points** The code snippet in program q4.cc attached below has a fatal flaw and results in possible corruption of the heap and unexplained crashes. Explain what the flaw is and describe two possible solutions. One of your solutions should be efficient and avoid needless string copying. The other solution need not be concerned with efficiency. *You don’t need to provide code or pseudo-code for your solutions, just describe them.* You might not be familiar with the `strncpy` function at line 21. It simply says “copy 99 bytes from address \(s\) to address \(str\).”
(a) Smart Pointers 30 points A copy of the *Smart Pointers* handout is attached for reference. A code snippet is given below using the *Smart Pointers* concept. In the table below, fill in the value the specified variable at the specified line number. For at least one entry, you will have to make some assumptions and an educated guess. A few of the rows of the table are filled in for you as a starting point. Note: The last entry at line 21 means after all local variable destructors have been called, but before subroutine *main* actually exits.

<table>
<thead>
<tr>
<th>After Line Number</th>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>sp1.st</td>
<td>0xB0001000</td>
</tr>
<tr>
<td>15</td>
<td>*(sp1.refCount)</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>sp2.st</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>*(sp2.refCount)</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>sp3.st</td>
<td>0xB0001020</td>
</tr>
<tr>
<td>17</td>
<td>*(sp3.refCount)</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>sp3.st</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>*(sp3.refCount)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>sp.st</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>*(sp.refCount)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>sp.st</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>*(sp.refCount)</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>sp.st</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>*(sp.refCount)</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>sp.st</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>*(sp.refCount)</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>sp1.st</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>*(sp1.refCount)</td>
<td></td>
</tr>
</tbody>
</table>

```plaintext
1 // SPointer class and implementation defined here.
2
3 void PassByValue(SPointer sp)
4 {
5     sp.Set(0, 'K'); // Change the zeroth char of the string
6 }
7
8 void PassByReference(SPointer& sp)
9 {
10     sp.Set(0, 'K'); // Change the zeroth char of the string
11 }
12
13 int main()
14 {
15     SPointer sp1("This is a test");
16     SPointer sp2(sp1); // Copy constructor
17     SPointer sp3("ShortString");
18     sp3 = sp2;
19     PassByValue(sp2);
20     PassByReference(sp1);
21 }
```

Program q-sp.cc
// Code for ECE3090 midterm, QUESTION 2 - Inheritance and Virtual Functions

class Base
{
   // Define a base class
   public:
      void Sub1();
      virtual void Sub2() = 0;
      virtual void Sub3();
      virtual void Sub4();
};
class A : public Base
{
   // Class A derives from Base
   public:
      void Sub1();
      void Sub2();
};
class B : public Base
{
   // Class B derives from Base
   public:
      void Sub1();
      void Sub2();
      void Sub4();
};
// Base Class Methods
void Base::Sub1()
{
   cout << "Hello from Base::Sub1()" << endl;
}
void Base::Sub3()
{
   cout << "Hello from Base::Sub3()" << endl;
   Sub1(); // DON'T MISS THIS CALL IN YOUR ANSWER
   Sub4(); // DON'T MISS THIS CALL IN YOUR ANSWER
}
void Base::Sub4()
{
   cout << "Hello from Base::Sub4()" << endl;
}

// A Helper Subroutine
void Sub(Base& x)
{
   x.Sub3();
   x.Sub2();
   x.Sub1();
}

// Class A Methods
void A::Sub1() { cout << "Hello from A:Sub1()" << endl; }
void A::Sub2() { cout << "Hello from A:Sub2()" << endl; }

// Class B Methods
void B::Sub1() { cout << "Hello from B:Sub1()" << endl; }
void B::Sub2() { cout << "Hello from B:Sub2()" << endl; }
void B::Sub4() { cout << "Hello from B:Sub4()" << endl; }

// A Helper Subroutine
void Sub(Base& x)
{
   x.Sub3();
   x.Sub2();
   x.Sub1();
}

int main()
{
   A a;
   B b;
   Sub(a);
   Sub(b);
}

Program q2.cc
class A {
public:
A(); // Constructor
A(char*); // Constructor
~A(); // Destructor
public:
char* str;
};

A::A()
{
    str = new char[100];
    str[0] = '\0'; // String initially empty
}

A::A(char* s)
{
    str = new char[100];
    strncpy(str, s, 99); // Set initial string value
    str[99] = '\0';
}

A::~A()
{
    delete [] str; // Return the memory to heap
}

int main()
{
    A a1;
    A a2(a1);
    A a3 = a2;
    A a4;
    a4 = a1;
}

Program q4.cc
class SPointer
{
public:
  SPointer(char*);
  SPointer(const SPointer&); // Need a copy constructor
  ~SPointer(); // Need a destructor
  SPointer& operator=(const SPointer& rhs); // Need an assignment operator

  char Get(int i); // Get the "i’th" character from the array
  void Set(int i, char c); // Set the "i’th" character to the char ‘c’

private:
  char* st; // This is the "shared" pointer
  int* refCount; // This keeps up with how many references to st
  int lth;
};

// Constructor
SPointer::SPointer(char* s)
  : lth(strlen(s) + 1) // lth is set to the length of the string "s"
{  
  st = new char[lth]; // Allocate the dynamic memory from heap
  strcpy(st, s); // Copy the string s to the heap
  refCount = new int(1); // Create the reference count variable, set to 1
}

// Copy constructor
SPointer::SPointer(const SPointer& c)
  : st(c.st), refCount(c.refCount), lth(c.lth)
{  
  (*refCount)++; // Increment the reference count
}

// Destructor
SPointer::~SPointer()
{  
  (*refCount)--; // Decrement the reference count
  if (*refCount == 0) // This is the last reference, delete
  {  
    delete [] st;
    delete refCount;
    st = 0;
  }
}

// Assignment operator
SPointer& SPointer::operator=(const SPointer& rhs)
{  
  if (st != rhs.st) // not self assignment
  {  
    (*refCount)--; // Decrement the reference count
    if (*refCount == 0) // Last reference to my string, delete
    {  
      delete [] st;
      delete refCount;
    }
  }

  return *this;
}
57     st = rhs.st; // Use same pointer as right-hand-side
58     refCount = rhs.refCount; // Use same reference count as rhs
59     lth = rhs.lth; // Use same length as rhs
60     (*refCount)++; // Count this reference
61     return *this;
62 }
63
64 // Get and Set functions
65 char SPointer::Get(int i)
66 {
67     if (i < lth) return st[i];
68     return '0'; // Out of range, just return 0
69 }
70
71 void SPointer::Set(int i, char c)// Set the "i’th" character to the char ’c’
72 // This is called "Copy on Write" semantics
73 {
74     if (i >= lth) return; // Out of range
75     if (*refCount > 1)
76         (*refCount)--; // Decrement reference count
77         char* newSt = new char[lth]; // Get a new memory array
78         refCount = new int(1); // Get a new reference count = 1
79         memcpy(newSt, st, lth); // Copy the data
80         st = newSt; // And set the new pointer
81     }
82     st[i] = c; // Change specified character
83 }

Program smartpointers.cc (continued)