```
// Shaun Chemplavil U08713628
// shaun.chemplavil@gmail.com
// C/C++ Programming IV : Advanced Programming with Objects
// 152488 Raymond L. Mitchell III
// hw5.cpp
// Win10
// Visual C++ 19.0
//

#include <iostream>
#include <exception>

// To avoid auto_ptr ambiguity we need to avoid using std namespace
using std::cout;
using std::clog;

template <typename X>
class auto_ptr
{
public:
   // Constructors
   explicit auto_ptr(X * = 0) throw();
   auto_ptr(auto_ptr &) throw();
   template <typename Y> auto_ptr(auto_ptr<Y> &) throw();
   // Destructor
   ~auto_ptr() throw();
   // Access
   X *get() const throw();
   X &operator*() const throw();
   X *operator->() const throw();
   // Assignment
   auto_ptr &operator=(auto_ptr &) throw();
   template <typename Y> auto_ptr &operator=(auto_ptr<Y> &) throw();
   // Release & Reset
   X *release() throw();
   void reset(X * = 0) throw();

private:
   X *aPtr;
   template<typename Y>
   friend class auto_ptr;  // make all auto_ptr classes
                           // friends of one another
};

// Explict Contructor
template<typename X>
auto_ptr<X>::auto_ptr(X *ptr)
   : aPtr(ptr) {}

// Copy Constructor
template<typename X>
auto_ptr<X>::auto_ptr(auto_ptr &rhs)
```

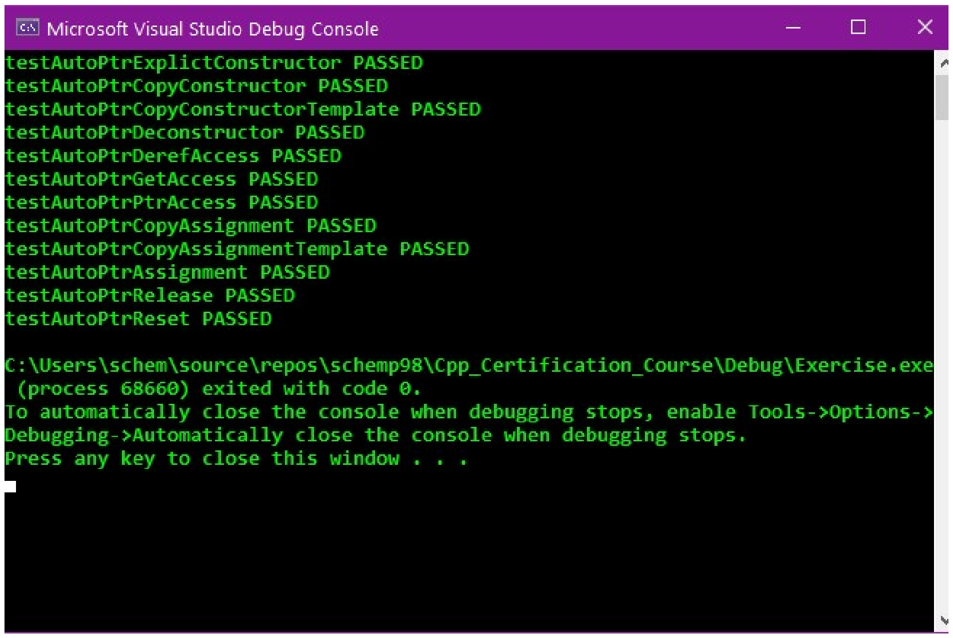
```cpp
   : aPtr(rhs.release()) {}

// Constructor Taking Ownership from auto_ptr
template<typename X>
template<typename Y>
auto_ptr<X>::auto_ptr(auto_ptr<Y>& rhs)
   : aPtr(rhs.release()) {}

// Destructor
template<typename X>
auto_ptr<X>::~auto_ptr()
{
   delete aPtr;
}

template<typename X>
auto_ptr<X> &auto_ptr<X>::operator=(auto_ptr& rhs)
{
   reset(rhs.release());
   return *this;
}

// Copy Assignment
template<typename X>
template<typename Y>
auto_ptr<X>& auto_ptr<X>::operator=(auto_ptr<Y>& rhs)
{
   // Check for self-assign
   if (this != &rhs)
      reset(rhs.release());
   return *this;
}

// Dereference Operator
template<typename X>
X& auto_ptr<X>::operator*() const
{
   return *aPtr;
}

// Pointer Access Operator
template<typename X>
X* auto_ptr<X>::operator->() const
{
   return aPtr;
}

// Get function (Pointer Access Operator)
template<typename X>
X* auto_ptr<X>::get() const
{
   return aPtr;
```

```cpp
}

template<typename X>
X* auto_ptr<X>::release()
{
   X *aPtrOld = aPtr;
   // Set pointer to Null and output original address
   aPtr = 0;
   return aPtrOld;
}

// Reset Auto Pointer and Point to input address
template<typename X>
void auto_ptr<X>::reset(X *ptr)
{
   if (aPtr != ptr) {
      delete aPtr;
      aPtr = ptr;
   }
}

// Unit Tests:
void testAutoPtrExplictConstructor()
{
   try
   {
      const int testInput(9);
      auto_ptr<int> testAutoPtr(new int(testInput));
      clog << "testAutoPtrExplictConstructor PASSED\n";
   }
   catch (...)
   {
      clog << "testAutoPtrExplictConstructor FAILED\n";
   }
}

void testAutoPtrCopyConstructor()
{
   try
   {
      const int testSource(9);
      auto_ptr<int> sourceAutoPtr(new int(testSource));
      auto_ptr<int> sinkAutoPtr(sourceAutoPtr);

      if ((testSource == *sinkAutoPtr) && (sourceAutoPtr.get() == 0))
         clog << "testAutoPtrCopyConstructor PASSED\n";
      else
         clog << "testAutoPtrCopyConstructor FAILED : Expected output "
         << testSource << " instead saw " << *sinkAutoPtr << "\n";
   }
   catch (...)
   {
      clog << "testAutoPtrCopyConstructor FAILED\n";
   }
}

void testAutoPtrCopyConstructorTemplate()
{
   try
   {
      const int testSource(9);
      auto_ptr<int> sourceAutoPtr(new int(testSource));
      auto_ptr<const int> sinkAutoPtr(sourceAutoPtr);

      if ((testSource == *sinkAutoPtr) && (sourceAutoPtr.get() == 0))
         clog << "testAutoPtrCopyConstructorTemplate PASSED\n";
      else
         clog << "testAutoPtrCopyConstructorTemplate FAILED : Expected output "
         << testSource << " instead saw " << *sinkAutoPtr << "\n";
   }
   catch (...)
   {
      clog << "testAutoPtrCopyConstructorNewType FAILED\n";
   }
}

void testAutoPtrDeconstructor()
{
   const int testInput(9);
   auto_ptr<int> *testAutoPtr = new auto_ptr<int>(new int(testInput));

   try
   {
      delete testAutoPtr;
      clog << "testAutoPtrDeconstructor PASSED\n";
   }
   catch (...)
   {
      clog << "testAutoPtrDeconstructor FAILED\n";
   }
}

void testAutoPtrDerefAccess()
{
   try
   {
      const int testSource(9);
      auto_ptr<int> sourceAutoPtr(new int(testSource));

      if (testSource == *sourceAutoPtr)
         clog << "testAutoPtrDerefAccess PASSED\n";
      else
         clog << "testAutoPtrDerefAccess FAILED : Expected output "
         << testSource << " instead saw " << *sourceAutoPtr << "\n";
   }
   catch (...)
   {
      clog << "testAutoPtrDerefAccess FAILED\n";
   }
}

void testAutoPtrGetAccess()
{
   try
   {
      const int testSource(9);
      auto_ptr<int> sourceAutoPtr(new int(testSource));
      int * sinkPtr = sourceAutoPtr.get();

      if (sinkPtr == sourceAutoPtr.get())
         clog << "testAutoPtrGetAccess PASSED\n";
      else
         clog << "testAutoPtrGetAccess FAILED : Expected output "
         << sinkPtr << " instead saw " << sourceAutoPtr.get() << "\n";
   }
   catch (...)
   {
      clog << "testAutoPtrGetAccess FAILED\n";
   }
}

void testAutoPtrPtrAccess()
{
   try
   {
      const int testSource(9);
      auto_ptr<int> sourceAutoPtr;

      int *sinkPtr = sourceAutoPtr.operator->();

      if (sinkPtr == sourceAutoPtr.get())
         clog << "testAutoPtrPtrAccess PASSED\n";
      else
         clog << "testAutoPtrPtrAccess FAILED : Expected output "
         << sinkPtr << " instead saw " << sourceAutoPtr.get() << "\n";
   }
   catch (...)
   {
      clog << "testAutoPtrPtrAccess FAILED\n";
   }
}

void testAutoPtrCopyAssignment()
{
   try
   {
```

```cpp
      const int testSource(9);
      auto_ptr<int> sourceAutoPtr(new int(testSource));

      //auto_ptr<int> sinkAutoPtr(new int(0));
      auto_ptr<int> sinkAutoPtr = sourceAutoPtr;
      //sinkAutoPtr = sourceAutoPtr;

      if ((testSource == *sinkAutoPtr) && (sourceAutoPtr.get() == 0))
         clog << "testAutoPtrCopyAssignment PASSED\n";
      else
         clog << "testAutoPtrCopyAssignment FAILED : Expected output "
         << testSource << " instead saw " << *sinkAutoPtr << "\n";
   }
   catch (...)
   {
      clog << "testAutoPtrCopyAssignment FAILED\n";
   }
}

void testAutoPtrCopyAssignmentTemplate()
{
   try
   {
      const int testSource(9);
      auto_ptr<int> sourceAutoPtr(new int(testSource));

      auto_ptr<const int> sinkAutoPtr = sourceAutoPtr;

      if ((testSource == *sinkAutoPtr) && (sourceAutoPtr.get() == 0))
         clog << "testAutoPtrCopyAssignmentTemplate PASSED\n";
      else
         clog << "testAutoPtrCopyAssignmentTemplate FAILED : Expected output "
         << testSource << " instead saw " << *sinkAutoPtr << "\n";
   }
   catch (...)
   {
      clog << "testAutoPtrCopyAssignmentTemplate FAILED\n";
   }
}
void testAutoPtrAssignment()
{
   try
   {
      const int testSource(9);
      auto_ptr<int> sourceAutoPtr(new int(testSource));

      auto_ptr<int> sinkAutoPtr(new int(0));
      sinkAutoPtr = sourceAutoPtr;

      if ((testSource == *sinkAutoPtr) && (sourceAutoPtr.get() == 0))
         clog << "testAutoPtrAssignment PASSED\n";
      else
```

```
         clog << "testAutoPtrAssignment FAILED : Expected output "
         << testSource << " instead saw " << *sinkAutoPtr << "\n";
   }
   catch (...)
   {
      clog << "testAutoPtrAssignment FAILED\n";
   }
}

void testAutoPtrRelease()
{
   try
   {
      const int testSource(9);
      auto_ptr<int> sourceAutoPtr(new int(testSource));
      int *sinkVal(sourceAutoPtr.release());

      if ((testSource == *sinkVal) && (sourceAutoPtr.get() == 0))
         clog << "testAutoPtrRelease PASSED\n";
      else
         clog << "testAutoPtrRelease FAILED : Expected output "
         << testSource << " instead saw " << sinkVal << "\n";
   }
   catch (...)
   {
      clog << "testAutoPtrRelease FAILED\n";
   }
}

void testAutoPtrReset()
{
   try
   {
      const int testSource(9);
      int * sourcePtr = new int(testSource * testSource);
      auto_ptr<int> sinkAutoPtr(new int(testSource));

      sinkAutoPtr.reset(sourcePtr);

      if (sinkAutoPtr.get() == sourcePtr)
         clog << "testAutoPtrReset PASSED\n";
      else
         clog << "testAutoPtrReset FAILED : Expected output "
         << sourcePtr << " instead saw " << sinkAutoPtr.get() << "\n";
   }
   catch (...)
   {
      clog << "testAutoPtrReset FAILED\n";
   }
}

int main(void)
```

```
{
    // 1)
    testAutoPtrExplictConstructor();
    testAutoPtrCopyConstructor();
    testAutoPtrCopyConstructorTemplate();

    // 2)
    testAutoPtrDeconstructor();

    // 3)
    testAutoPtrDerefAccess();
    testAutoPtrGetAccess();
    testAutoPtrPtrAccess();

    //4)
    testAutoPtrCopyAssignment();
    testAutoPtrCopyAssignmentTemplate();
    testAutoPtrAssignment();

    //5)
    testAutoPtrRelease();
    testAutoPtrReset();
}

```



```
testAutoPtrExplictConstructor PASSED
testAutoPtrCopyConstructor PASSED
testAutoPtrCopyConstructorTemplate PASSED
testAutoPtrDeconstructor PASSED
testAutoPtrDerefAccess PASSED
testAutoPtrGetAccess PASSED
testAutoPtrPtrAccess PASSED
testAutoPtrCopyAssignment PASSED
testAutoPtrCopyAssignmentTemplate PASSED
testAutoPtrAssignment PASSED
testAutoPtrRelease PASSED
testAutoPtrReset PASSED

C:\Users\schem\source\repos\schemp98\Cpp_Certification_Course\Debug\Exercise.exe
 (process 68660) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->
Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```