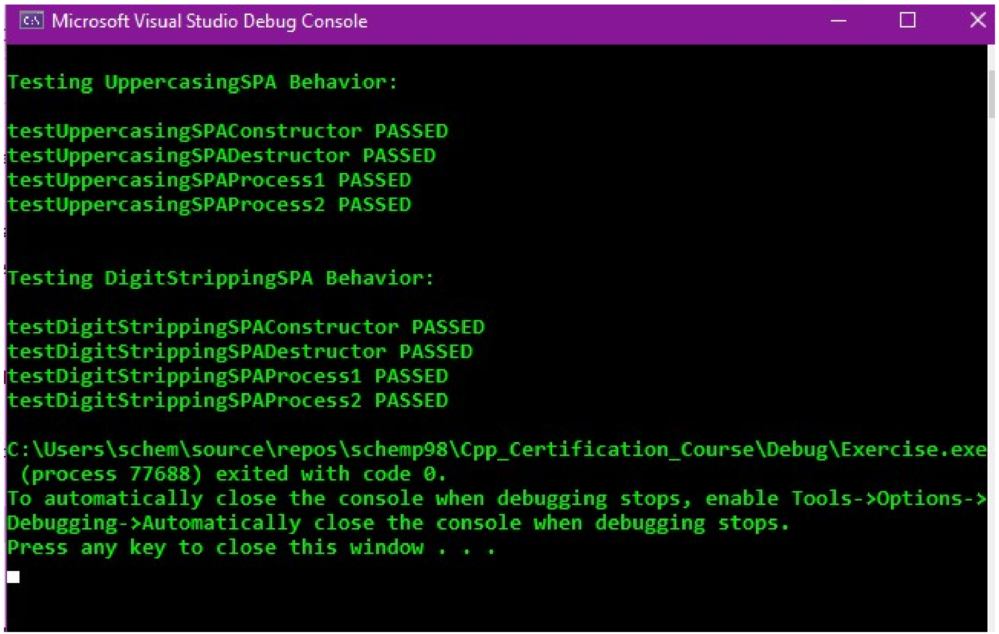
```
// Shaun Chemplavil U08713628
// shaun.chemplavil@gmail.com
// C/C++ Programming IV : Advanced Programming with Objects
// 152488 Raymond L. Mitchell III
// hw3.cpp
// Win10
// Visual C++ 19.0
//

#include <iostream>
#include <sstream>
#include <cctype> // isdigit, toupper
#include <string>
#include <algorithm>  // transform
#include <exception>

using namespace std;

class StreamProcessorAlgorithm
{
public:
   //Default Constructor
   StreamProcessorAlgorithm(istream &in, ostream &out) :in_(in), out_(out) {}
   virtual ~StreamProcessorAlgorithm() {}
   void process();
private:
   virtual bool filterToken(const string &token) const = 0;
   virtual void processToken(string &token) const = 0;
   istream &in_;
   ostream &out_;
};

void StreamProcessorAlgorithm::process()
{
   // For each whitespace separated string (token) read from the input stream:
   while (in_)
   {
      string token;

      //extract input stream in to string
      in_ >> token;

      // If the token passes through the filter
      if (filterToken(token))
      {
         // Process the token and output it to the output stream
         processToken(token);
         out_ << token;
      }
   }
}

```

```cpp
// Stream Processing Algorithm to Upper Case ALL input tokens
class UppercasingSPA : public StreamProcessorAlgorithm
{
public:
   UppercasingSPA(istream &in, ostream &out) :StreamProcessorAlgorithm(in, out)
   {}

   ~UppercasingSPA() {}

private:
   bool filterToken(const string &token) const
   {
      // Allows ALL tokens to pass through
      return true;
   }

   void processToken(string &token) const
   {
      transform(token.begin(), token.end(), token.begin(), toupper);
   }
};

// Stream Processing Algorithm to strip out digits from input token
class DigitStrippingSPA : public StreamProcessorAlgorithm
{
public:
   DigitStrippingSPA(istream &in, ostream &out) :StreamProcessorAlgorithm(in, 
    out)
   {}

   ~DigitStrippingSPA() {}

private:
   bool filterToken(const string &token) const
   {
      // Allows tokens containing at least one digit to pass through
      return find_if(token.begin(), token.end(), isdigit) != token.end();
   }

   void processToken(string &token) const
   {
      // Erase character from string if it is a digit
      token.erase(remove_if(token.begin(), token.end(), isdigit), token.end());
   }
};

// Unit Tests:
void testUppercasingSPAConstructor()
{
   try
   {
      UppercasingSPA testSPA(cin, cout);
      clog << "testUppercasingSPAConstructor PASSED\n";
   }
   catch (...)
   {
      clog << "testUppercasingSPAConstructor FAILED\n";
   }
}
void testUppercasingSPADestructor()
{
   UppercasingSPA *testSPA = new UppercasingSPA(cin, cout);

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

// "standard" input
void testUppercasingSPAProcess1()
{
   const string TEST_INPUT = "This is my 1st Test!";
   const string VALID_OUTPUT = "THISISMY1STTEST!";
   istringstream testInputStream(TEST_INPUT);
   ostringstream testOutputStream;

   UppercasingSPA testUpper(testInputStream, testOutputStream);

   testUpper.process();

   // Check if expected output is present
   if (testOutputStream.str().compare(VALID_OUTPUT) == 0)
      clog << "testUppercasingSPAProcess1 PASSED\n";
   else
      clog << "testUppercasingSPAProcess1 FAILED : Expected output "
      << VALID_OUTPUT << " instead saw " << testOutputStream.str() << "\n";
}

// non-alpha character input
void testUppercasingSPAProcess2()
{
   const string TEST_INPUT = "1232$*)(_*1543!";
   const string VALID_OUTPUT = TEST_INPUT;
   istringstream testInputStream(TEST_INPUT);
   ostringstream testOutputStream;

   UppercasingSPA testUpper(testInputStream, testOutputStream);

   testUpper.process();

   // Check if expected output is present
   if (testOutputStream.str().compare(VALID_OUTPUT) == 0)
      clog << "testUppercasingSPAProcess2 PASSED\n";
   else
      clog << "testUppercasingSPAProcess2 FAILED : Expected output "
      << VALID_OUTPUT << " instead saw " << testOutputStream.str() << "\n";
}

void testDigitStrippingSPAConstructor()
{
   try
   {
      DigitStrippingSPA testSPA(cin, cout);
      clog << "testDigitStrippingSPAConstructor PASSED\n";
   }
   catch (...)
   {
      clog << "testDigitStrippingSPAConstructor FAILED\n";
   }
}
void testDigitStrippingSPADestructor()
{
   DigitStrippingSPA *testSPA = new DigitStrippingSPA(cin, cout);

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

// alpha numeric token
void testDigitStrippingSPAProcess1()
{
   const string TEST_INPUT = "Th1s 1s my 1st D1g1t T3st";
   const string VALID_OUTPUT = "ThssstDgtTst";
   istringstream testInputStream(TEST_INPUT);
   ostringstream testOutputStream;

   DigitStrippingSPA testSPA(testInputStream, testOutputStream);

   testSPA.process();

   // Check if expected output is present
   if (testOutputStream.str().compare(VALID_OUTPUT) == 0)
      clog << "testDigitStrippingSPAProcess1 PASSED\n";
   else
      clog << "testDigitStrippingSPAProcess1 FAILED : Expected output "
      << VALID_OUTPUT << " instead saw " << testOutputStream.str() << "\n";
}

// non-numeric token
void testDigitStrippingSPAProcess2()
{
   const string TEST_INPUT = "This is MY second tEst!!$";
   const string VALID_OUTPUT = "";
   istringstream testInputStream(TEST_INPUT);
   ostringstream testOutputStream;

   DigitStrippingSPA testSPA(testInputStream, testOutputStream);

   testSPA.process();

   // Check if expected output is present
   if (testOutputStream.str().compare(VALID_OUTPUT) == 0)
      clog << "testDigitStrippingSPAProcess2 PASSED\n";
   else
      clog << "testDigitStrippingSPAProcess2 FAILED : Expected output "
      << VALID_OUTPUT << " instead saw " << testOutputStream.str() << "\n";
}

int main(void)
{
   cout << "\nTesting UppercasingSPA Behavior:\n\n";
   testUppercasingSPAConstructor();
   testUppercasingSPADestructor();
   testUppercasingSPAProcess1();
   testUppercasingSPAProcess2();

   cout << "\n\nTesting DigitStrippingSPA Behavior:\n\n";
   testDigitStrippingSPAConstructor();
   testDigitStrippingSPADestructor();
   testDigitStrippingSPAProcess1();
   testDigitStrippingSPAProcess2();
}
```



```
Testing UppercasingSPA Behavior:

testUppercasingSPAConstructor PASSED
testUppercasingSPADestructor PASSED
testUppercasingSPAProcess1 PASSED
testUppercasingSPAProcess2 PASSED


Testing DigitStrippingSPA Behavior:

testDigitStrippingSPAConstructor PASSED
testDigitStrippingSPADestructor PASSED
testDigitStrippingSPAProcess1 PASSED
testDigitStrippingSPAProcess2 PASSED

C:\Users\schem\source\repos\schemp98\Cpp_Certification_Course\Debug\Exercise.exe
 (process 77688) 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 . . .
```