

```
1 // Shaun Chemplavil U08713628
2 // shaun.chemplavil@gmail.com
3 // C/C++ Programming IV : Advanced Programming with Objects
4 // 152488 Raymond L. Mitchell III
5 // hw3.cpp
6 // Win10
7 // Visual C++ 19.0
8 //
9
10 #include <iostream>
11 #include <sstream>
12 #include <cctype> // isdigit, toupper
13 #include <string>
14 #include <algorithm> // transform
15 #include <exception>
16
17 using namespace std;
18
19 class StreamProcessorAlgorithm
20 {
21 public:
22     //Default Constructor
23     StreamProcessorAlgorithm(istream &in, ostream &out) :in_(in), out_(out) {}
24     virtual ~StreamProcessorAlgorithm() {}
25     void process();
26 private:
27     virtual bool filterToken(const string &token) const = 0;
28     virtual void processToken(string &token) const = 0;
29     istream &in_;
30     ostream &out_;
31 };
32
33 void StreamProcessorAlgorithm::process()
34 {
35     // For each whitespace separated string (token) read from the input stream:
36     while (in_)
37     {
38         string token;
39
40         //extract input stream in to string
41         in_ >> token;
42
43         // If the token passes through the filter
44         if (filterToken(token))
45         {
46             // Process the token and output it to the output stream
47             processToken(token);
48             out_ << token;
49         }
50     }
51 }
52
```

```
53 // Stream Processing Algorithm to Upper Case ALL input tokens
54 class UppercasingSPA : public StreamProcessorAlgorithm
55 {
56 public:
57     UppercasingSPA(istream &in, ostream &out) :StreamProcessorAlgorithm(in, out)
58     {}
59
60     ~UppercasingSPA() {}
61
62 private:
63     bool filterToken(const string &token) const
64     {
65         // Allows ALL tokens to pass through
66         return true;
67     }
68
69     void processToken(string &token) const
70     {
71         transform(token.begin(), token.end(), token.begin(), toupper);
72     }
73 };
74
75 // Stream Processing Algorithm to strip out digits from input token
76 class DigitStrippingSPA : public StreamProcessorAlgorithm
77 {
78 public:
79     DigitStrippingSPA(istream &in, ostream &out) :StreamProcessorAlgorithm(in, out)
80     {}
81
82     ~DigitStrippingSPA() {}
83
84 private:
85     bool filterToken(const string &token) const
86     {
87         // Allows tokens containing at least one digit to pass through
88         return find_if(token.begin(), token.end(), isdigit) != token.end();
89     }
90
91     void processToken(string &token) const
92     {
93         // Erase character from string if it is a digit
94         token.erase(remove_if(token.begin(), token.end(), isdigit), token.end());
95     }
96 };
97
98 // Unit Tests:
99 void testUppercasingSPAConstructor()
100 {
101     try
102     {
103         UppercasingSPA testSPA(cin, cout);
```

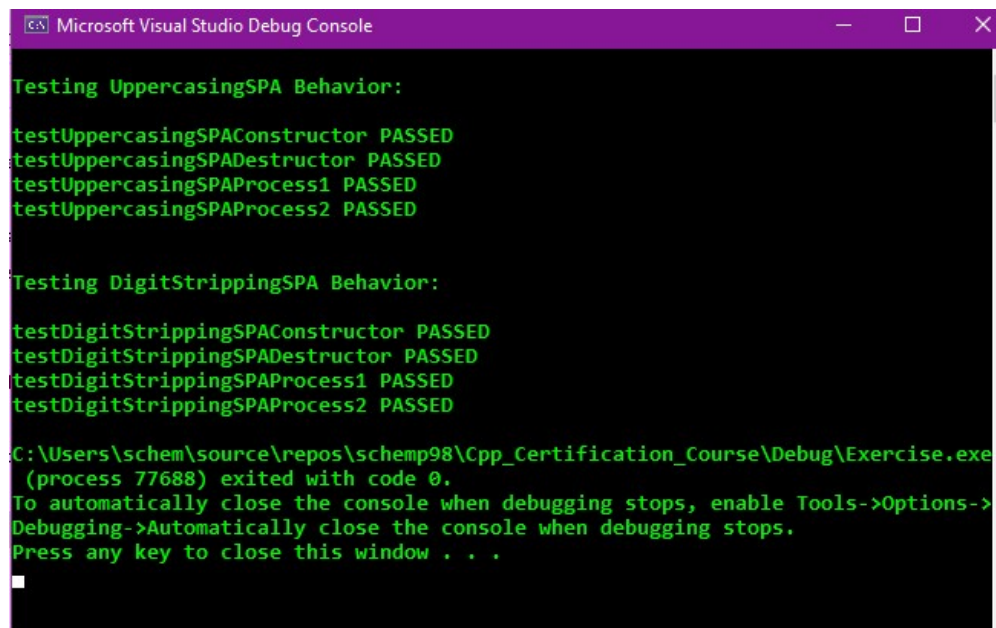
```
104     clog << "testUppercasingSPAConstructor PASSED\n";
105 }
106 catch (...)
107 {
108     clog << "testUppercasingSPAConstructor FAILED\n";
109 }
110 }
111 void testUppercasingSPADestructor()
112 {
113     UppercasingSPA *testSPA = new UppercasingSPA(cin, cout);
114
115     try
116     {
117         delete testSPA;
118         clog << "testUppercasingSPADestructor PASSED\n";
119     }
120     catch (...)
121     {
122         clog << "testUppercasingSPADestructor FAILED\n";
123     }
124 }
125
126 // "standard" input
127 void testUppercasingSPAProcess1()
128 {
129     const string TEST_INPUT = "This is my 1st Test!";
130     const string VALID_OUTPUT = "THISISMY1STTEST!";
131     istream testInputStream(TEST_INPUT);
132     ostream testOutputStream;
133
134     UppercasingSPA testUpper(testInputStream, testOutputStream);
135
136     testUpper.process();
137
138     // Check if expected output is present
139     if (testOutputStream.str().compare(VALID_OUTPUT) == 0)
140         clog << "testUppercasingSPAProcess1 PASSED\n";
141     else
142         clog << "testUppercasingSPAProcess1 FAILED : Expected output "
143         << VALID_OUTPUT << " instead saw " << testOutputStream.str() << "\n";
144 }
145
146 // non-alpha character input
147 void testUppercasingSPAProcess2()
148 {
149     const string TEST_INPUT = "1232$(*)_1543!";
150     const string VALID_OUTPUT = TEST_INPUT;
151     istream testInputStream(TEST_INPUT);
152     ostream testOutputStream;
153
154     UppercasingSPA testUpper(testInputStream, testOutputStream);
155 }
```

```
156     testUpper.process();
157
158     // Check if expected output is present
159     if (testOutputStream.str().compare(VALID_OUTPUT) == 0)
160         clog << "testUppercasingSPAProcess2 PASSED\n";
161     else
162         clog << "testUppercasingSPAProcess2 FAILED : Expected output "
163         << VALID_OUTPUT << " instead saw " << testOutputStream.str() << "\n";
164 }
165
166 void testDigitStrippingSPAConstructor()
167 {
168     try
169     {
170         DigitStrippingSPA testSPA(cin, cout);
171         clog << "testDigitStrippingSPAConstructor PASSED\n";
172     }
173     catch (...)
174     {
175         clog << "testDigitStrippingSPAConstructor FAILED\n";
176     }
177 }
178 void testDigitStrippingSPADestructor()
179 {
180     DigitStrippingSPA *testSPA = new DigitStrippingSPA(cin, cout);
181
182     try
183     {
184         delete testSPA;
185         clog << "testDigitStrippingSPADestructor PASSED\n";
186     }
187     catch (...)
188     {
189         clog << "testDigitStrippingSPADestructor FAILED\n";
190     }
191 }
192
193 // alpha numeric token
194 void testDigitStrippingSPAProcess1()
195 {
196     const string TEST_INPUT = "Th1s 1s my 1st D1g1t T3st";
197     const string VALID_OUTPUT = "ThssstDgtTst";
198     istream testInputStream(TEST_INPUT);
199     ostream testOutputStream;
200
201     DigitStrippingSPA testSPA(testInputStream, testOutputStream);
202
203     testSPA.process();
204
205     // Check if expected output is present
206     if (testOutputStream.str().compare(VALID_OUTPUT) == 0)
207         clog << "testDigitStrippingSPAProcess1 PASSED\n";
```

```

208     else
209         clog << "testDigitStrippingSPAProcess1 FAILED : Expected output "
210         << VALID_OUTPUT << " instead saw " << testOutputStream.str() << "\n";
211 }
212
213 // non-numeric token
214 void testDigitStrippingSPAProcess2()
215 {
216     const string TEST_INPUT = "This is MY second tEst!!$";
217     const string VALID_OUTPUT = "";
218     istream testInputStream(TEST_INPUT);
219     ostream testOutputStream;
220
221     DigitStrippingSPA testSPA(testInputStream, testOutputStream);
222
223     testSPA.process();
224
225     // Check if expected output is present
226     if (testOutputStream.str().compare(VALID_OUTPUT) == 0)
227         clog << "testDigitStrippingSPAProcess2 PASSED\n";
228     else
229         clog << "testDigitStrippingSPAProcess2 FAILED : Expected output "
230         << VALID_OUTPUT << " instead saw " << testOutputStream.str() << "\n";
231 }
232
233 int main(void)
234 {
235     cout << "\nTesting UpperCasingSPA Behavior:\n\n";
236     testUpperCasingSPAConstructor();
237     testUpperCasingSPADestructor();
238     testUpperCasingSPAProcess1();
239     testUpperCasingSPAProcess2();
240
241     cout << "\n\nTesting DigitStrippingSPA Behavior:\n\n";
242     testDigitStrippingSPAConstructor();
243     testDigitStrippingSPADestructor();
244     testDigitStrippingSPAProcess1();
245     testDigitStrippingSPAProcess2();
246 }
247

```



```

Microsoft Visual Studio Debug Console

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 . . .

```