

PSP0.1 Project Plan Summary

Name: James Small
 Program: 2A
 Instructor: Dr. Concepcion

Number: 2
 Language: C++

<i>Program Size (LOC)</i>	<i>Plan</i>	<i>Actual</i>	<i>To Date</i>
<i>Base (B)</i>		<u>0</u> (Measured)	
<i>Deleted (D)</i>		<u>0</u> (Counted)	
<i>Modified (M)</i>		<u>0</u> (Counted)	
<i>Added (A)</i>		<u>46</u> (T-B+D-R)	
<i>Reused (R)</i>		<u>0</u> (Counted)	<u>0</u>
<i>New and Changed (N)</i>	<u>50</u>	<u>46</u> (A+M)	<u>0</u>
<i>Total LOC (T)</i>		<u>46</u> (Measured)	<u>143</u>
<i>Total New Reusable</i>		<u>0</u>	

<i>Time in Phase (min.)</i>	<i>Plan</i>	<i>Actual</i>	<i>To Date</i>	<i>To Date %</i>
Planning	<u>8</u>	<u>2</u>	<u>9</u>	<u>8.4</u>
Design	<u>8</u>	<u>7</u>	<u>14</u>	<u>13.1</u>
Code	<u>14</u>	<u>21</u>	<u>33</u>	<u>30.8</u>
Compile	<u>11</u>	<u>13</u>	<u>22</u>	<u>20.6</u>
Test	<u>17</u>	<u>4</u>	<u>18</u>	<u>16.8</u>
Postmortem	<u>6</u>	<u>6</u>	<u>11</u>	<u>10.3</u>
Total	<u>65</u>	<u>53</u>	<u>107</u>	<u>100</u>

<i>Defects Injected</i>	<i>Actual</i>	<i>To Date</i>	<i>To Date %</i>
Planning	<u>0</u>	<u>0</u>	<u>0</u>
Design	<u>0</u>	<u>1</u>	<u>16.7</u>
Code	<u>3</u>	<u>5</u>	<u>83.3</u>
Compile	<u>0</u>	<u>0</u>	<u>0</u>
Test	<u>0</u>	<u>0</u>	<u>0</u>
Total Development	<u>3</u>	<u>6</u>	<u>100</u>

<i>Defects Removed</i>	<i>Actual</i>	<i>To Date</i>	<i>To Date %</i>
Planning	<u>0</u>	<u>0</u>	<u>0</u>
Design	<u>0</u>	<u>0</u>	<u>0</u>
Code	<u>0</u>	<u>0</u>	<u>0</u>
Compile	<u>1</u>	<u>3</u>	<u>50</u>
Test	<u>2</u>	<u>3</u>	<u>50</u>
Total Development	<u>3</u>	<u>6</u>	<u>100</u>
After Development			

Process Improvement ProposalName: James SmallProgram: 2AInstructor: Dr. ConcepcionNumber: 2Language: C++

Problem Description Briefly describe the problems that you encountered.
A problem that I am noticing is that just about all my errors thus far have been in the coding stage. Most of the problems encountered are simple errors, either in syntax, or general logic that are easy to fix, yet I keep making them.
Proposal Description Briefly describe the process improvements that you propose.
The improvements that I propose to solve this is to put more time into the code section and not rush through it like I'm tending to do. By doing this, I think I can eliminate the obvious coding errors which waste time to fix. This won't eliminate design errors, but will fix the more common errors that I'm encountering.
Other Notes and Comments Note any other comments or observations that describe your experiences or improvement ideas.

Name: James Small

Program: 2A

Instructor: Dr. Concepcion

Number: 2

Language: C++

[illegible]

Total: 53

Defect Recording Log

Name: James Small
 Program: 2A
 Instructor: Dr. Concepcion

Number: 2
 Language: C++

Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
2014-01-28	1	20 - Syntax	Code	Compile	1	

Description: forgot correct syntax for getline command

Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
2014-01-28	2	80 - Function	Code	Test	2	

Description: Forgot () around the following, currentString[0] == '/' when using ! operator which flipped expected results

Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
2014-01-28	3	80 - Function	Code	Test	2	

Description: In line that determines if string contains all blanks, I put a ! in front which was returning blank lines instead of not blank lines

Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description:

Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description:

Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description:

Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description:

```

// Name: James Small
// Program: 2A
// Class: CSE455
// Description: Main Program to count lines of code in a file

#include <iostream>
#include <string>
#include "Counter.h"

using namespace std;

int main()
{
    string file;

    cout << "Enter the file name to count lines from: ";
    cin >> file;

    Counter count(file);

    cout << "LOC for " << file << " = " << count.calculateLOC()
<< endl;

    return 0;
}

```

```
// Name: James Small
// Program: 2A
// Class: CSE455
// Description: Counter class Header File

#ifndef COUNTER_H
#define COUNTER_H

#include <string>

using namespace std;

class Counter
{
    public:
        Counter(string fileName);
        int calculateLOC();

    private :
        string fileToRead;
};
#endif
```

```

// Name: James Small
// Program: 2A
// Class: CSE455
// Description: Counter class Implementation File

#include "Counter.h"
#include <fstream>

using namespace std;

// This is the default constructor
Counter::Counter(string fileName)
{
    this->fileToRead = fileName;
}

// This method calculates the LOC in a file
int Counter::calculateLOC()
{
    ifstream infile;

    infile.open(fileToRead.c_str());

    int count = 0;
    string currentString;

    while (getline(infile, currentString))
        if ((currentString.find_first_not_of(' ') !=
string::npos))
            if (!(currentString[0] == '/'))
                count++;

    infile.close();

    return count;
}

```

Report R1: LOC Standard - James Small

Definition Name:	C++ LOC std.	Language:	C++
Author:	James Small	Date:	1/29/14

Count Type	Type	Comments
Physical/Logical	Physical	
Statement Type	Included	Comments
Executable	Yes	Count any executable line. See note 1 below.
Nonexecutable:		
Declarations	Yes	Count declarations as they are used in the running of the program
Compiler Directives	Yes	Count declarations as they are used in the running of the program
Comments	No	Do not count comments as they are not used in the running of the program
Blank lines	No	Do not count comments as they are not used in the running of the program
Notes		
Note 1		This includes brackets that exist on their own line. They will be counted as one line of code in the LOC

Report R2: Coding Standard - James Small

Purpose	Show Coding Standard for C++ Programs
Program Headers	Begin all files with a header
Header Format	<p>Header to contain the following, with each line starting on the left side of the file</p> <pre>// Name: James Small // Program: 2A // Class: CSE455 // Description: Main Program to count lines of code in a file</pre>
Method Headers	<p>Methods must contain a header in the format of a comment. The header will consist of one or more comment lines with a brief description of the method. See comments section on formatting rules</p> <pre>// This function calculates the LOC of a file</pre>
Identifiers	Use Descriptive names for identifiers. No abbreviations or single letter for variables, functions, or any other identifiers.
Identifiers Example	<pre>int a = 1 // this is bad int letterCount = 1 // this is good</pre>
Comments	Document code where needed using comments. Comments on their own line must begin with // and start on the far left side of the file. Comments on same line of code must start 2 spaces after the end of the line of code. Never use /* */ style of commenting
Blank Spaces	Program must be written so different logical sections of code must be separated by a blank line. If lines of code are related, then no blank lines between them.
Indenting	Indent every level of brace from the previous one. Indenting is done using 4 spaces at all times. See examples below for indenting for different types of commands.
Capitalization	Class names start with a capital. Functions, methods, and variables start with a lower case letter and follow standard camel case structure. Any define statements are all capitalized.
Brackets	<p>For class and method declarations, the opening and closing {} must be on their own lines. For all other brackets, the opening { will be on the same line as the command, while the closing bracket will be on it's own line. See examples in the sections below. For all brackets used besides those on classes and methods, if the number of items inside the brackets is greater than 1 line, then go ahead and use the opening and closing brackets. If there is only one line of code that would exist inside the brackets, then no opening or closing brackets are used. The one line will be indented as usual on the next line below the command. Example:</p> <pre>while (getline(infile,currentString)) if ((currentString.find_first_not_of(' ') != string::npos)) if (!(currentString[0] == '/')) count++;</pre> <p>The while and both if statements each contain one line below them, so no brackets are used.</p>

Report R2: Coding Standard - James Small

Variables	<p>variables of the same type can be declared on the same line</p> <pre>int count, score, points;</pre> <p>An initial value can be specified on the declaration of the variable.</p> <pre>int count = 0;</pre>
Class Interface	<p>Begin class interface with header above. Then show <code>#ifndef</code>, <code>#define</code>, and end with <code>#endif</code>. Next will be all <code>#include</code>'s. Separate <code>public:</code> <code>private:</code> sections. If no items exist for one of those sections, don't list the section with no items in it. If both sections exist, put a space between them.</p>
Class Interface Example:	<pre>// Name: James Small // Program: 2A // Class: CSE455 // Description: Counter class Header File #ifndef COUNTER_H #define COUNTER_H #include <string> using namespace std; class Counter { public: Counter(string fileName); int calculateLOC(); private : string fileToRead; }; #endif</pre>
if, if/else, if/else if/else	<p>examples of if constructs:</p> <p>if example:</p> <pre>if (count == 5) { count += 2; result += count; }</pre> <p>if/else example:</p> <pre>if (count == 5) { count += 2; result += count;</pre>

Report R2: Coding Standard - James Small

	<pre>} else { count += 3; result += count; } if/else if/else example: if (count == 5) { count += 2; result += count; } else if (count == 6) { count += 3; result += count; }</pre>
while Statement	<p>Example of While Statement</p> <pre>while (count != 5) { count++; result += count; }</pre>
do while statement	<p>Example of do while statement</p> <pre>do { count++; result += count; } while (count != 5);</pre>
for statement	<p>Example of for statement</p> <pre>for (int i = 0; i < 5; i++) { count += i; result += count; }</pre>
switch statement	<p>example of switch statement</p> <pre>switch (grade) { case 'A': cout << "Excellent!" << endl; break; case 'B': case 'C': cout << "Well done" << endl; break; case 'D': cout << "You passed" << endl; break; case 'F': cout << "Better try again" << endl; break; default :</pre>

Report R2: Coding Standard - James Small

	<pre>cout << "Invalid grade" << endl; }</pre> <p>Switch statement must contain a default section at the bottom.</p>
--	---

Other Requested Materials

Compilation Image

```
james-iac:program AcousticTime$ g++ -c program2a.cpp
james-iac:program AcousticTime$ g++ -c Counter.cpp
james-iac:program AcousticTime$ g++ -o program2a program2a.o Counter.o
```

Program1B LOC

```
james-iac:program AcousticTime$ ./program2a
Enter the file name to count lines from: Counter.h
LOC for Counter.h = 13
james-iac:program AcousticTime$ ./program2a
Enter the file name to count lines from: Counter.cpp
LOC for Counter.cpp = 20
james-iac:program AcousticTime$ ./program2a
Enter the file name to count lines from: program2a.cpp
LOC for program2a.cpp = 13
```

Program2A LOC

```
james-iac:program AcousticTime$ ./program2a
Enter the file name to count lines from: Input.h
LOC for Input.h = 14
james-iac:program AcousticTime$ ./program2a
Enter the file name to count lines from: Input.cpp
LOC for Input.cpp = 37
james-iac:program AcousticTime$ ./program2a
Enter the file name to count lines from: program1b.cpp
LOC for program1b.cpp = 25
```

Other Requested Materials

Test Results Table

<u>Program Number</u>	<u>LOC</u>
1B	Counter.h = 13 Counter.cpp = 20 Program1B.cpp = 13 Total LOC = 46
2A	Input.h = 14 Input.cpp = 37 Program2a.cpp = 25 Total LOC = 76