PSP0.1 Project Plan Summary

Name: James Small

Program: 3A Number: 3

Instructor: Dr. Concepcion Language: C++

Program Size (LOC)	***Plan***	***Actual***	***To Date***
Base (B)		46 (Measured)	
Deleted (D)		2 (Counted)	
Modified (M)		5 (Counted)	
Added (A)		145 (T-B+D-R)	
Reused (R)		0 (Counted)	0
New and Changed (N)	85	150 (A+M)	0
Total LOC (T)		189 (Measured)	332
Total New Reusable		0	

Time in Phase (min.)	**Plan**	**Actual**	**To Date**	**To Date %**
Planning	9	3	12	4.2
Design	14	9	23	8
Code	34	66	99	34.5
Compile	23	6	28	9.8
Test	19	82	100	34.8
Postmortem	11	14	25	8.7
Total	110	180	287	100

Defects Injected	Actual	To Date	To Date %
Planning	0	0	0
Design	0	1	9.1
Code	5	10	90.9
Compile	0	0	0
Test	0	0	0
Total Development	5	11	100

Defects Removed	Actual	To Date	To Date %
Planning	0	0	0
Design	0	0	0
Code	0	0	0
Compile	3	6	54.5
Test	2	5	45.5
Total Development	5	11	100
After Development			

Process Improvement Proposal

Name: James Small

Program: 3A Number: 3

Instructor: Dr. Concepcion Language: C++

Problem Description Briefly describe the problems that you encountered.
I spent the time to make some pseduo code showing the basic outline and flow of my program. I felt it was pretty well done, but after coding, I missed an important key which causes and infinite loop, and 20 minutes of searching to figure out the cause.
Proposal Description Briefly describe the process improvements that you propose.
My plan is to review the pseduo code I write after I write it. Of course I looked through it a bunch of times when I was writing the pseduo code, but I need to take a look the next day and go back through my logic. More than likely I will find some problems with what I'm doing when taking a look a second time.
Other Notes and Comments Note any other comments or observations that describe your experiences or improvement ideas.

Time Recording Log

Name: James Small
Program: 3A Number: 3
Instructor: Dr. Concepcion Language: C++

Date	Start	Stop	Int. Time	Delta Time	Phase	Comments
2014-01-31	11:13	11:16	0	3	Planning	Setup forms and made time and LOC estimates
2014-01-31	11:16	11:25	0	9	Design	Wrote out design in pseudo code
2014-02-01	1:01	1:27	0	26	Code	Started Writing Code
2014-02-01	10:12	10:52	0	40	Code	Continued Writing Code
2014-02-02	12:47	12:53	0	6	Compile	Compiled and Fixed Errors
2014-02-02	12:53	14:15	0	82	Test	Tested Code and Fixed Errors
2014-02-03	12:11	12:25	0	14	Postmortem	Made screenshots and finished paperwork

Total: 180

Defect Recording Log

Name: James Small
Program: 3A Number: 3
Instructor: Dr. Concepcion Language: C++

Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
2014-02-02	1	20 - Syntax	Code	Compile	1	

Description: forgot to delcare fstream variable before using it

Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
2014-02-02	2	20 - Syntax	Code	Compile	1	

Description: forgot () on length method on string

Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
2014-02-02	3	20 - Syntax	Code	Compile	2	

Description: mistyped variable name

Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
2014-02-02	4	80 - Function	Code	Test	10	

Description: needed to break out of the for loop when condition was found

Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
2014-02-02	5	80 - Function	Code	Test	19	

Description: forgot to test weather class had been previously found causing infinte loop

Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description:

Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description:

```cpp
// Name: James Small
// Program: 3A
// Class: CSE455
// Description: Main Program to count lines of code in all files of a program

#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);
    count.calculateLOC();
    count.displayReport();

    return 0;
}
```

```
// Name: James Small
// Program: 3A
// Class: CSE455
// Description: Counter class Header File

#ifndef COUNTER_H
#define COUNTER_H

#include <string>
#include <vector>
#include "ClassInfo.h"

using namespace std;

class Counter
{
    public:
        Counter(string fileName);
        void calculateLOC();
        void displayReport();

    private :
        string currentFileName;
        int methodCount();
        vector<ClassInfo> classVector;
        vector<string> classListVector();
        string classNameWithoutExtension(string className);
};
#endif
```

```cpp
// Name: James Small
// Program: 3A
// Class: CSE455
// Description: Counter class Implementation File

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

using namespace std;

// This is the default constructor

Counter::Counter(string fileName)
{
    this->currentFileName = classNameWithoutExtension(fileName);
}

// This method calcualtes the LOC in a file

void Counter::calculateLOC()
{
    ifstream infile;

    int count = 0;
    string currentString;

    currentFileName.append(".h");

    for (int i = 0; i < 2; i++) {
        infile.open(currentFileName.c_str());

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

        infile.close();

        currentFileName = classNameWithoutExtension(currentFileName);
        currentFileName.append(".cpp");
    }

    int methodC = methodCount();

    ClassInfo currentClass(classNameWithoutExtension(currentFileName), methodC, count);

    classVector.push_back(currentClass);

    vector<string> currentClasses = classListVector();

    for (int i = 0; i < currentClasses.size(); i++) {
        currentFileName = classNameWithoutExtension(currentClasses[i]);
        calculateLOC();
    }
}

// This method returns the number of methods in a given class

int Counter::methodCount()
{
    string currentString;

    string stringToCheck = classNameWithoutExtension(currentFileName);
    stringToCheck.append("::");
```

```
    currentFileName = classNameWithoutExtension(currentFileName);
    currentFileName.append(".cpp");

    int methodCount = 0;

    ifstream infile;

    infile.open(currentFileName.c_str());

    while (getline(infile,currentString))
        if (currentString.find(stringToCheck) != string::npos)
            methodCount++;

    infile.close();

    return methodCount;
}

// This method returns a vector listing the names of all classes found in current class

vector<string> Counter::classListVector()
{
    string currentString;

    string stringToCheck = "#include \"";

    vector<string> classList;

    currentFileName = classNameWithoutExtension(currentFileName);

    currentFileName.append(".h");

    ifstream infile;

    for (int i = 0; i < 2; i++) {
        infile.open(currentFileName.c_str());

        while (getline(infile,currentString)) {

            size_t found = currentString.find(stringToCheck);

            if (found != string::npos) {

                string temp;

                for (int i = currentString.find('\"') + 1; i < currentString.length(); i++) {

                    if (currentString[i] != '\"')
                        temp.push_back(currentString[i]);
                    else
                        break;
                }

                bool notFound = true;

                for (int i = 0; i < classList.size(); i++) {
                    if (classList[i] == temp) {
                        notFound = false;
                        break;
                    }
                }

                for (int i = 0; i < classVector.size(); i++) {
                    if (classVector[i].getClassName() == classNameWithoutExtension(temp)) {
                        notFound = false;
                        break;
```

```
                    }
                }

                if (notFound)
                    classList.push_back(temp);
            }
        }

        infile.close();

        currentFileName = classNameWithoutExtension(currentFileName);
        currentFileName.append(".cpp");
    }

    return classList;
}

// This method returns the class name without an extension on it

string Counter::classNameWithoutExtension(string className)
{
    string temp;

    for (int i = 0; i < className.length(); i++) {
        if (className[i] != '.')
            temp.push_back(className[i]);
        else
            break;
    }

    return temp;
}

// This method displays the results of the LOC for all files in the program

void Counter::displayReport()
{
    int masterCount = 0;

    cout << "\nProgram Name: " << classVector[0].getClassName() << "\n\n";
    masterCount += classVector[0].getLineCount();

    for (int i = 1; i < classVector.size(); i++) {

        cout << "Class Name: " << classVector[i].getClassName() << endl;
        cout << "Method Count: " << classVector[i].getMethodCount() << endl;
        cout << "Class Line Count: " << classVector[i].getLineCount() << "\n\n";
        masterCount += classVector[i].getLineCount();
    }

    cout << classVector[0].getClassName() << " Master Count: " << masterCount << "\n\n";
}
```

```
// Name: James Small
// Program: 3A
// Class: CSE455
// Description: ClassInfo class Header File

#ifndef CLASSINFO_H
#define CLASSINFO_H

#include <string>

using namespace std;

class ClassInfo
{
    public:
        ClassInfo(string className, int methodCount, int lineCount);
        string getClassName();
        int getMethodCount();
        int getLineCount();

    private :
        string className;
        int methodCount;
        int lineCount;
};
#endif
```

```cpp
// Name: James Small
// Program: 3A
// Class: CSE455
// Description: ClassInfo class Implementation File

#include "ClassInfo.h"

using namespace std;

// This is the default constructor

ClassInfo::ClassInfo(string className, int methodCount, int lineCount)
{
    this->className = className;
    this->methodCount = methodCount;
    this->lineCount = lineCount;
}

// This is the getter method for className

string ClassInfo::getClassName()
{
    return className;
}

// This is the getter method for methodCount

int ClassInfo::getMethodCount()
{
    return methodCount;
}

// This is the getter method for lineCount

int ClassInfo::getLineCount()
{
    return lineCount;
}
```

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	Header to contain the following, with each line starting on the left side of the file
	
	// Name: James Small
	// Program: 2A
	// Class: CSE455
	// Description: Main Program to count lines of code in a file

Method Headers	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 // This function calculates the LOC of a file
Identifiers	Use Descriptive names for identifiers. No abbreviations or single letter for variables, functions, or any other identifiers.
Identifiers Example	int a = 1 // this is bad int letterCount = 1 // this is good
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	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:
	
	while (getline(infile,currentString))
	  if ((currentString.find_first_not_of(' ') != string::npos))
	  if (!(currentString[0] == '/'))
	  count++;
	
	The while and both if statements each contain one line below them, so no backets are used.

Variables	variables of the same type can be declared on the same line int count, score, points; An initial value can be specified on the declaration of the variable. int count = 0;
Class Interface	Begin class interface with header above. Then show #ifndef, #define, and end with #endif. Next will be all #include's. Separate public: private: 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.

Class Interface Example:	// 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

if, if/else, if/else if/else	examples of if constructs:
	
	if example:
	
	if (count == 5) {
	count += 2;
	result += count;
	}
	
	if/else example:
	
	if (count == 5) {
	count += 2;
	result += count;

	} 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;
	}

while Statement	Example of While Statement
	
	while (count != 5) {
	  count++;
	  result += count;
	}

do while statement	Example of do while statement
	
	do {
	  count++;
	  result += count;
	} while (count != 5);

for statement	Example of for statement
	
	for (int i = 0; i < 5; i++) {
	  count += i;
	  result =+ count;
	}

switch statement	example of switch statement
	
	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 :

	cout << "Invalid grade" << endl; } Switch statement must contain a default section at the bottom.

Compilation

```
jamess-imac:program AcousticTime$ g++ -c ClassInfo.cpp
jamess-imac:program AcousticTime$ g++ -c Counter.cpp
jamess-imac:program AcousticTime$ g++ -o program3a program3a.cpp ClassInfo.o Counter.o
```

Program 1B

```
jamess-imac:Program2A AcousticTime$ ./program3a
Enter the file name to count lines from: program1b.cpp

Program Name: program1b

Class Name: Input
Method Count: 3
Class Line Count: 51

program1b Master Count: 76
```

Program 2A

```
jamess-imac:Program2A AcousticTime$ ./program3a
Enter the file name to count lines from: program2a.cpp

Program Name: program2a

Class Name: Counter
Method Count: 2
Class Line Count: 33

program2a Master Count: 46
```

Program 3A

```
jamess-imac:program AcousticTime$ ./program3a
Enter the file name to count lines from: program3a.cpp

Program Name: program3a

Class Name: Counter
Method Count: 6
Class Line Count: 138

Class Name: ClassInfo
Method Count: 4
Class Line Count: 37

program3a Master Count: 189
```

Results Table

Program #	Object Name	# of Methods	Object LOC	Total Prog LOC
1B	Input	3	51	
				76
2A	Counter	2	33	
				46
3A	Counter	6	138	
	ClassInfo	4	37	
				189

Pseudo Code

Main Program

create Count object
pass it program filename
count method called
display report

Count Lines Method - Recursive

create new ClassInfo Object
    add name of current class to object – extension
count lines and store result in ClassInfo Object
count methods and store result in ClassInfo Object
add ClassInfo object to vector
search for classes and store each name in local vector
for each class found,
    run count lines method with new name which repeats above recursively