UNIVERSITY SCHOOL OF INFORMATION, COMMUNICATION AND TECHNOLOGY GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY

Software Measurement, Metrics, And Modelling Lab

Subject Code: ITE-311

Submitted to: DR. PRIYANKA BHUTANI

Submitted by:Priyanshu

Course: B.TECH. CSE 5thSemester

Enrolment No:03516403221

INDEX

Practical-1

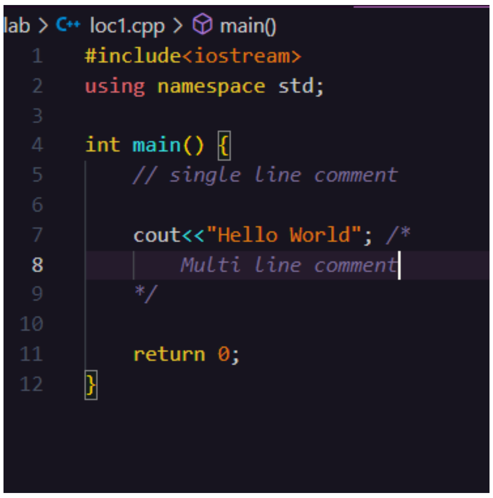
AIM:To find the Lines of code of a given C/C++ program input from a text file.

Code:

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main(){
  ifstream inputFile("loc1.cpp");
  if (!inputFile){
    cout << "Failed to open the input file." << endl;
    return 1;
  }
  string line;
  int codeLines = 0;
  bool inMultiLineComment = false;
  while (getline(inputFile, line)) {
    string trimmedLine;
    for (char c : line){
      if (!isspace(c)){
        trimmedLine += c;
      }
    }
    if (inMultiLineComment){
      int commentEnd = trimmedLine.find("*/");
      if (commentEnd == -1){
        continue;
      }
      else      {
        inMultiLineComment = false;
        trimmedLine = trimmedLine.substr(commentEnd + 2);
```

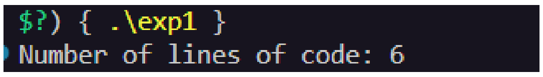
```
            }
        }
        if (trimmedLine.empty() || (trimmedLine.find("//") == 0))  {
            continue;
        }
        int commentStart = trimmedLine.find("/*");
        if (commentStart != -1){
            inMultiLineComment = true;
            trimmedLine = trimmedLine.substr(0, commentStart);
        }
        if (!trimmedLine.empty()) {
            codeLines++;
        }
    }
    cout << "Number of lines of code: " << codeLines << endl;
    inputFile.close();
    return 0;
}
```

OUTPUT:



lab > loc1.cpp > main()

```
#include<iostream>
using namespace std;

int main() {
    // single line comment

    cout<<"Hello World"; /*
        Multi line comment
    */

    return 0;
}
```



```
$?) { .\exp1 }
Number of lines of code: 6
```

Practical-2

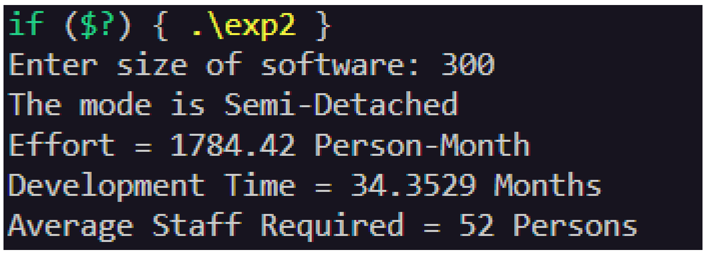
AIM:To implement the COCOMO I model of software measurement.

Code:

```
#include <bits/stdc++.h>
using namespace std;
int fround(float x){
        int a;
        x = x + 0.5;
        a = x;
        return (a);
}
void calculate(float table[][4], int n, char mode[][15],
                        int size){
        float effort, time, staff;
        int model;
        if (size >= 2 && size <= 50)
                model = 0; // organic
        else if (size > 50 && size <= 300)
                model = 1; // semi-detached
        else if (size > 300)
                model = 2; // embedded
        cout << "The mode is " << mode[model];
        effort = table[model][0] * pow(size, table[model][1]);
        time = table[model][2] * pow(effort, table[model][3]);
        staff = effort / time;
        cout << "\nEffort = " << effort << " Person-Month";
        cout << "\nDevelopment Time = " << time << " Months";
        cout << "\nAverage Staff Required = " << fround(staff)
                << " Persons";
}
int main()
```

```
{
        float table[3][4] = { 2.4, 1.05, 2.5, 0.38, 3.0, 1.12,
                                                    2.5, 0.35, 3.6, 1.20, 2.5, 0.32 };
        char mode[][15]
                = { "Organic", "Semi-Detached", "Embedded" };
        int size;
        cout<<"Enter size of software: ";
        cin>>size;
        calculate(table, 3, mode, size);
        return 0;
}
```

Output:



```
if ($?) { .\exp2 }
Enter size of software: 300
The mode is Semi-Detached
Effort = 1784.42 Person-Month
Development Time = 34.3529 Months
Average Staff Required = 52 Persons
```

Practical-3

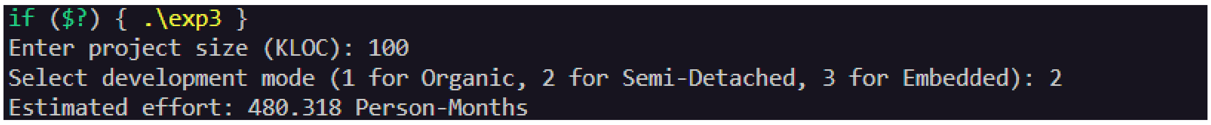
AIM:To implement the COCOMO II model of software measurement.

Code:

```
#include <iostream>
#include <math.h>
class COCOMOII {
public:
  // Function to estimate effort in Person-Months
  double estimateEffort(double size, int mode) {
    // Basic formula: Effort = A * (Size ^ B) * EAF
    double A = 2.94;  // Example value, would be calibrated based on the project
    double B = 1.067; // Example value, would be calibrated based on the project
    double EAF = calculateEAF(mode); // Environmental Adjustment Factor
    double effort = A * pow(size, B) * EAF;
    return effort;
  }
private:
  // Function to calculate the Environmental Adjustment Factor (EAF)
  double calculateEAF(int mode) {
    // EAF is based on various factors such as personnel capability, process maturity, etc.
    // This is a simplified example, in reality, you would need to consider more factors.
    double eaf = 1.0;
    // Adjust based on the development mode
    switch (mode) {
      case 1: // Organic
        eaf *= 1.0;
        break;
      case 2: // Semi-Detached
        eaf *= 1.2;
        break;
      case 3: // Embedded
```

```
                eaf *= 1.5;
                break;
            default:
                std::cerr << "Invalid mode." << std::endl;
                break;
        }
        return eaf;
    }
};
int main() {
    COCOMOII cocomo;
    double projectSize;
    int developmentMode;
    std::cout << "Enter project size (KLOC): ";
    std::cin >> projectSize;
    std::cout << "Select development mode (1 for Organic, 2 for Semi-Detached, 3 for Embedded): ";
    std::cin >> developmentMode;
    double effort = cocomo.estimateEffort(projectSize, developmentMode);
    std::cout << "Estimated effort: " << effort << " Person-Months" << std::endl;
    return 0;
}
```

Output:



```
if ($?) { .\exp3 }
Enter project size (KLOC): 100
Select development mode (1 for Organic, 2 for Semi-Detached, 3 for Embedded): 2
Estimated effort: 480.318 Person-Months
```

Practical-4

AIM:Case study of IntelliJ tool

IntelliJ IDEA is a powerful Integrated Development Environment (IDE) developed by JetBrains. It's widely used for Java development, but it supports multiple programming languages like Kotlin, Groovy, Scala, and others. Below is a case study showcasing the features and benefits of IntelliJ IDEA in a software development project:

Company Background:

XYZ Corp, a software development company, aimed to build a robust and scalable web application for managing financial transactions. The project required a proficient development environment to ensure efficiency, code quality, and timely delivery.

Challenges Faced:

Complexity of the Project: Developing a financial application involves handling intricate logic, data processing, and security measures.

Collaboration and Code Management: The team consisted of developers working on various modules simultaneously. Efficient collaboration and code management were essential.

Code Quality Assurance: Ensuring high code quality, adhering to coding standards, and minimising errors were crucial to the project's success.

Solution using IntelliJ IDEA:

Smart Code Assistance and Refactoring Tools: The developers leveraged IntelliJ IDEA's smart code completion, quick-fix suggestions, and powerful refactoring tools. This streamlined the coding process, improving productivity and reducing errors. Features like 'Extract Method,' 'Rename,' and 'Inline' helped in code maintenance and readability.

Version Control Integration: IntelliJ IDEA seamlessly integrated with version control systems like Git, allowing the team to collaborate efficiently. Developers could manage branches, merge changes, and resolve conflicts directly within the IDE.

Code Inspection and Analysis: The IDE's built-in code inspection tools helped identify potential bugs, performance issues, and code smells. This proactive approach ensured better code quality and reduced the chances of runtime errors.

Testing and Debugging Capabilities: The integrated testing framework support and debugging tools enabled developers to run unit tests, debug code, and track issues effectively. This ensured robustness and reliability in the application.

Support for Frameworks and Libraries: IntelliJ IDEA provided excellent support for various frameworks and libraries, such as Spring, Hibernate, and JPA, easing integration and enhancing development speed.

Results:

Increased Developer Productivity: The intuitive interface and robust feature set of IntelliJ IDEA significantly boosted developers' productivity, enabling them to focus more on coding logic and less on mundane tasks.

Improved Code Quality and Stability: By leveraging code inspections, refactoring tools, and testing capabilities, the team maintained high code quality, reducing the number of bugs and enhancing the application's stability.

Streamlined Collaboration: Seamless integration with version control systems facilitated smoother collaboration among team members, allowing for efficient code sharing and management.

Timely Project Delivery: With enhanced productivity, better code quality, and effective collaboration, the project met its deadlines and delivered a reliable financial application.

In conclusion, leveraging IntelliJ IDEA's advanced features and robust capabilities significantly contributed to the success of XYZ Corp's financial application project by ensuring efficient development, high-quality code, and timely delivery.

Practical-5

AIM:Case **s**tudy **o**f **D**esignite **t**ool

Designite is a software design quality assessment tool that analyses code quality, detects design issues, and provides insights to improve software maintainability and extensibility. While specific case studies may not be widely available for Designite, I can outline a hypothetical scenario showcasing its potential benefits in a software development project:

Company Background:

XYZ Tech Solutions, a software development firm, aimed to enhance the quality of its flagship product, an e-commerce platform experiencing scalability and maintainability challenges. They sought a tool to analyse and improve the codebase's design quality.

Challenges Faced:

Complex Codebase: The e-commerce platform had evolved over time, resulting in a complex and intertwined code structure that was becoming hard to maintain.

Scalability Issues: The existing design inhibited scalability, making it difficult to incorporate new features and modifications swiftly.

Code Quality and Maintainability: The lack of adherence to design principles led to reduced code maintainability, making it challenging to identify and rectify potential issues.

Solution using Designite:

Design Quality Analysis: Designite's capabilities to analyse code quality against various design metrics and principles were leveraged to identify design flaws, code smells, and anti-patterns within the codebase.

Identifying Hotspots: Designite's hotspot analysis helped pinpoint the most critical and problematic areas within the code, allowing the team to prioritise refactoring efforts.

Maintainability Improvement Suggestions: By leveraging Designite's insights and recommendations, the development team received actionable suggestions to refactor code, improve modularity, and enhance maintainability.

Custom Rule Definitions: The tool's flexibility allowed the team to define custom rules tailored to the specific requirements of the e-commerce platform, enabling targeted design improvement strategies.

Results:

Enhanced Code Maintainability: Designite's analysis and suggestions aided in restructuring the codebase, improving modularity and reducing complex interdependencies, thereby enhancing its maintainability.

Identification and Mitigation of Design Issues: The tool's analysis identified design flaws, anti-patterns, and areas of improvement, enabling the team to proactively rectify these issues.

Improved Scalability: Through targeted refactoring based on Designite's insights, the e-commerce platform's architecture became more scalable, allowing for easier incorporation of new features and modifications.

Reduced Technical Debt: Addressing design issues and code smells helped in reducing technical debt, leading to a more robust and maintainable codebase over time.

In conclusion, while specific case studies might not be available, the adoption of Designite by XYZ Tech Solutions facilitated the identification and rectification of design flaws, improving code maintainability, scalability, and reducing technical debt within their e-commerce platform, thereby contributing to its long-term sustainability and quality.

Practical-6

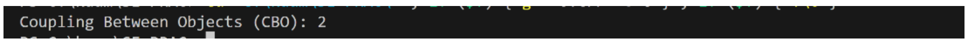
Aim: To Calculate the Coupling between Objects (CBO) given in Chidamber and Kemerer Metric Suite.

Code:

```
#include <iostream>

#include <vector>

#include <unordered_set>

using namespace std;

class ClassA {

public:

   void methodA() {

      cout << "Method A in ClassA\n";

   }};

class ClassB {

public:

   void methodB(ClassA& objA) {

      objA.methodA();

      cout << "Method B in ClassB\n";

   }}


int calculateCBO(const std::vector<std::unordered_set<std::string>>& dependencies) {

   int cbo = 0;

   for (const auto& set : dependencies) {

      cbo += set.size();

   }

   return cbo;

}


int main() {

   vector<std::unordered_set<std::string>> dependencies;

   unordered_set<std::string> dependencySet;

   dependencySet.insert("ClassA");
```

```
    dependencies.push_back(dependencySet);

    int cbo = calculateCBO(dependencies);

    cout << "Coupling Between Objects (CBO): " << cbo << endl;

    return 0;

}
```

<u>OUTPUT:</u>



```
Coupling Between Objects (CBO): 2
```

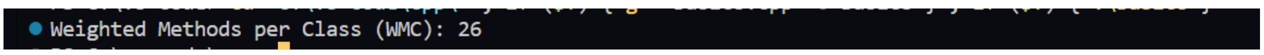
Practical 7

Aim: Write a program to find the Weighted Methods per Class given in Chidamber & Kemerer Metric Suite.

Code:

```
#include <iostream>

#include <vector>

#include <string>

using namespace std;

class MyClass {

public:

   void method1() {

      complexity += 10;

   }

   void method2() {

      complexity += 16;

   }

   int calculateWMC() {

      complexity = 0;

      method1();

      method2();

      return complexity;

   }

private:

   int complexity = 0;

};

int main() {

   MyClass myObject;

   int wmc = myObject.calculateWMC();

   cout << "Weighted Methods per Class (WMC): " << wmc << endl;return 0;}
```

OUTPUT:



```
Weighted Methods per Class (WMC): 26
```

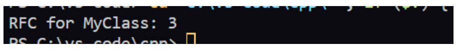
Practical 8

Aim: Write a program to find the Response for a Class metric given in Chidamber & Kemerer Metric Suite.

Code:

```
#include <iostream>
#include <vector>
#include <set>
using namespace std;
class ClassInfo;
class MethodInfo{
public:
string methodName;
set<string> calledMethods;
};
class ClassInfo{
public:
string className;
vector<MethodInfo> methods;
};
int calculateRFC(const ClassInfo &classInfo){
set<string> uniqueMethods;
for (const auto &method : classInfo.methods){
uniqueMethods.insert(method.methodName);
for (const auto &calledMethod : method.calledMethods){
uniqueMethods.insert(calledMethod);
}
}
return static_cast<int>(uniqueMethods.size());
}
int main(){
ClassInfo myClass;
myClass.className = "MyClass";
MethodInfo method1;
method1.methodName = "method1";
method1.calledMethods.insert("method2");
method1.calledMethods.insert("method3");
MethodInfo method2;
method2.methodName = "method2";
method2.calledMethods.insert("method1");
MethodInfo method3;
method3.methodName = "method3";
myClass.methods.push_back(method1);
myClass.methods.push_back(method2);
myClass.methods.push_back(method3);
int rfc = calculateRFC(myClass);
cout << "RFC for " << myClass.className << ": " << rfc << endl;
return 0;
}
```

OUTPUT:



```
RFC for MyClass: 3
PS C:\vs code\cpp>
```

Practical 9

Aim: Write a program to find the Lack of Cohesion in Methods (LCOM) metric given in Chidamber & Kemerer Metric Suite.

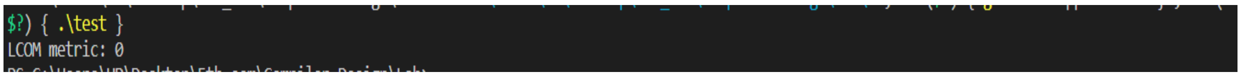
Code:

```
#include <iostream>
#include <vector>
#include <set>
using namespace std;
class MyClass {
private:
   int variable1;
   float variable2;
   char variable3;
public:
   void method1() {
      variable1 = 10;
   }
   void method2() {
      variable2 = 3.14f;
   }
   void method3() {
      variable3 = 'A';
   }
   int getVariable1() const {
      return variable1;
   }
   float getVariable2() const {
      return variable2;
   }
```

```
char getVariable3() const {
    return variable3;
}
int calculateLCOM() const {
    vector< set<string>> methodSets;
    for (const auto& method : {"method1", "method2", "method3"}) {
        set< string> methodSet;
        methodSet.insert(method);
        methodSets.push_back(methodSet);
    }
    if (variable1 == 10) {
        methodSets[0].insert("method2");
        methodSets[0].insert("method3");
    }
    if (variable2 == 3.14f) {
        methodSets[1].insert("method1");
        methodSets[1].insert("method3");
    }
    if (variable3 == 'A') {
        methodSets[2].insert("method1");
        methodSets[2].insert("method2");
    }
    int disjointSets = 0;
    for (const auto& set : methodSets) {
        if (!set.empty()) {
            disjointSets++;
        }
    }
    int lcom = methodSets.size()- disjointSets;
    return lcom;
}
```

```
};

int main() {

   MyClass myObject;

   int lcomValue = myObject.calculateLCOM();

   cout << "LCOM metric: " << lcomValue << std::endl;

   return 0;

}
```

Output:



```
$?) { .\test }
LCOM metric: 0
```

Practical 10

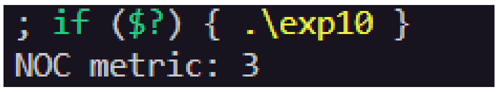
Aim: Write a program to find the Number of Children (NOC) metric given in Chidamber & Kemerer Metric Suite.

Code:

```
#include <iostream>
#include <vector>
class BaseClass {
public:
   // Base class methods and members
};
class ChildClass1 : public BaseClass {
public:
   // Child class 1 methods and members
};
class ChildClass2 : public BaseClass {
public:
   // Child class 2 methods and members
};
class ChildClass3 : public BaseClass {
public:
   // Child class 3 methods and members
};
// Calculate NOC metric
int calculateNOC(const std::vector<BaseClass*>& classes) {
   return classes.size();
}
int main() {
   // Create instances of child classes
   ChildClass1 child1;
   ChildClass2 child2;
```

```
    ChildClass3 child3;

    // Store pointers to base class objects in a vector

    std::vector<BaseClass*> classVector = {&child1, &child2, &child3};

    // Calculate and display NOC metric

    int nocValue = calculateNOC(classVector);

    std::cout << "NOC metric: " << nocValue << std::endl;

    return 0;

}
```

Output:



```
; if ($?) { .\exp10 }
NOC metric: 3
```

Practical 11

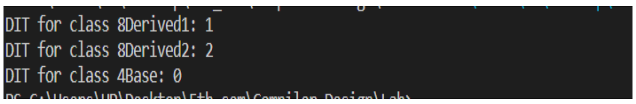
Aim: Write a program to find the Depth of Inheritance (DIT) metric given in Chidamber & Kemerer Metrics.

Code:

```
#include <iostream>
#include <unordered_set>
#include <typeinfo>
class Base {
   // Your base class implementation goes here
};
class Derived1 : public Base {
   // Your first derived class implementation goes here
};
class Derived2 : public Derived1 {
   // Your second derived class implementation goes here
};
// Utility class to calculate DIT
class DITCalculator {
public:
   static int calculateDIT(std::unordered_set<const std::type_info*>& visitedTypes, const std::type_info&
type) {
      // Check if the type has already been visited to avoid infinite loops in case of cyclic dependencies
      if (visitedTypes.count(&type) > 0) {
         return 0;
      }
      // Add the current type to the set of visited types
      visitedTypes.insert(&type);
      // Get the base type (super class)
      const std::type_info* baseType = getBaseType(type);
      // Recursively calculate the DIT for the base type
      if (baseType != nullptr) {
```

```
      int baseTypeDepth = calculateDIT(visitedTypes, *baseType);
      return baseTypeDepth + 1;
    }
    return 0;
  }
private:
  // Utility function to get the base type (super class) of a given type
  static const std::type_info* getBaseType(const std::type_info& derivedType) {
    if (derivedType == typeid(Derived2)) {
      return &typeid(Derived1);
    } else if (derivedType == typeid(Derived1)) {
      return &typeid(Base);
    }
    return nullptr;
  }
};
int main() {
  // Add your classes to the set
  std::unordered_set<const std::type_info*> types;
  types.insert(&typeid(Base));
  types.insert(&typeid(Derived1));
  types.insert(&typeid(Derived2));
  // Add other classes as needed
  // Calculate DIT for each class
  for (const auto& type : types) {
    std::unordered_set<const std::type_info*> visitedTypes;
    int dit = DITCalculator::calculateDIT(visitedTypes, *type);
    std::cout << "DIT for class " << type->name() << ": " << dit << std::endl;
  }   return 0;}
```

Output:



```
DIT for class 8Derived1: 1
DIT for class 8Derived2: 2
DIT for class 4Base: 0
```