SOFTWARE MEASUREMENTS , METRICS & MODELLING

LAB FILE

ITE 311

Submitted in partial fulfilment of the requirements

for the reward of the degree

of

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

SUBMITTED BY:

Name: Manvendra Singh

Roll no.: 00216407722(LE)

SUBMITTED TO:

Dr. Priyanka Bhutani

UNIVERSITY SCHOOL OF INFORMATION, COMMUNICATION AND TECHNOLOGY

GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY

DECEMBER - 2023

INDEX

PRACTICAL - 1

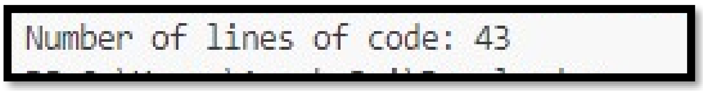
Aim: To find the Lines of code of a given C/C++ program input from a text file.

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;
}
```

<u>OUTPUT</u>



```
Number of lines of code: 43
```

PRACTICAL - 2

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

CODE

```
#include <bits/stdc++.h>
using namespace std;
// Function
// For rounding off float to int
int fround(float x)
{
        int a;
        x = x + 0.5;
        a = x;
        return (a);
}

// Function to calculate parameters of Basic COCOMO
void calculate(float table[][4], int n, char mode[][15],
                        int size)
{
        float effort, time, staff;

        int model;

        // Check the mode according to size

        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];

        // Calculate Effort
        effort = table[model][0] * pow(size, table[model][1]);

        // Calculate Time
        time = table[model][2] * pow(effort, table[model][3]);

        // Calculate Persons Required
        staff = effort / time;

        // Output the values calculated
        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 = 4;
	calculate(table, 3, mode, size);
	return 0;
}
```

OUTPUT

```
The mode is Organic

Effort = 10.289 Person-Month

Development Time = 6.06237 Months

Average Staff Required = 2 Persons
```

PRACTICAL - 3

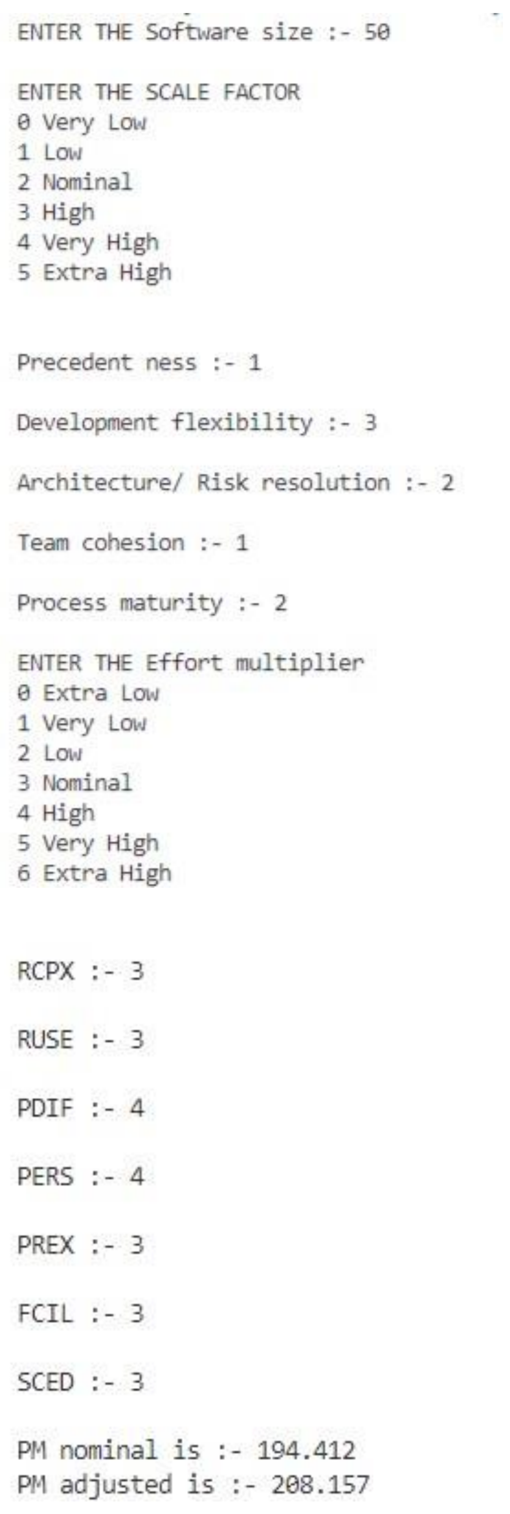
Aim: To implement the COCOMO II model of software measurement.

CODE

```
#include<bits/stdc++.h>
using namespace std;
double calculateB(int *value){
     // double Precedent_ness[6] = {6.20,4.96,3.72,2.48,1.24,0.00};
   // double Development_flexibility[6] = {5.07,4.05,3.04,2.03,1.01,0.00};
   // double Risk_resolution[6] = {7.07,5.65,4.24,2.83,1.41,0.00};
   // double Team_cohesion[6] = {5.48,4.38,3.29,2.19,1.10,0.00};
   // double Process_maturity[6] = {7.80,6.24,4.68,3.12,1.56,0.00};

   double Scaling_factor[5][6] = {6.20,4.96,3.72,2.48,1.24,0.00,
                              5.07,4.05,3.04,2.03,1.01,0.00,
                              7.07,5.65,4.24,2.83,1.41,0.00,
                              5.48,4.38,3.29,2.19,1.10,0.00,
                              7.80,6.24,4.68,3.12,1.56,0.00};
  double sum=0.0;

   for(int i=0 ; i<5 ; i++){
      sum += Scaling_factor[i][value[i]];


      // cout<<"***"<<Scaling_factor[i][value[i]];
   }


   return (0.91+(0.01*sum));
}

double calculate_effort_multiplier(int *value_effortMultiplier){
   double matrix[7][7] = {0.73,0.81,0.98,1.00,1.30,1.74,2.38,
                    0.00,0.00,0.95,1.00,1.07,1.15,1.24,
                    0.00,0.00,0.87,1.00,1.29,1.81,2.61,
                    2.12,1.62,1.26,1.00,0.83,0.63,0.50,
                    1.59,1.33,1.12,1.00,0.87,0.71,0.62,
                    1.43,1.30,1.10,1.00,0.87,0.73,0.62,
                    0.00,1.43,1.14,1.00,1.00,1.00,0.00};

   double effortMultiplier = 1;

   for(int i=0 ; i<7 ; i++){
      effortMultiplier *= matrix[i][value_effortMultiplier[i]];
   }
   return effortMultiplier;
}
```

```
int main(){
   double A = 2.5;
      int size;
cout<<"ENTER THE Software size :- ";
   cin>>size;

   cout<<"\nENTER THE SCALE FACTOR \n";
   cout<<"0 Very Low\n";
   cout<<"1 Low\n";
   cout<<"2 Nominal\n";
   cout<<"3 High\n";
   cout<<"4 Very High\n";
   cout<<"5 Extra High\n\n";

   string Scaling_factors[5] = {"Precedent ness","Development flexibility","Architecture/ Risk
resolution","Team cohesion","Process maturity"};

   int value_scalingFactor[5];
   for(int i=0 ; i<5 ; i++){
      cout<<endl<<Scaling_factors[i]<<" :- ";
      cin>>value_scalingFactor[i];
   }
   double B = calculateB(value_scalingFactor);

   double PM_nominal = A*pow(size,B);


   string Early_design_cost_driver[7]={"RCPX","RUSE","PDIF","PERS","PREX","FCIL","SCED"};
   cout<<"\nENTER THE Effort multiplier \n";
   cout<<"0 Extra Low\n";
   cout<<"1 Very Low\n";
   cout<<"2 Low\n";
   cout<<"3 Nominal\n";
   cout<<"4 High\n";
   cout<<"5 Very High\n";
   cout<<"6 Extra High\n\n";

   int value_effortMultiplier[7];
   for(int i=0 ; i<7 ; i++){
      cout<<endl<<Early_design_cost_driver[i]<<" :- ";
      cin>>value_effortMultiplier[i];
   }

   double effort_multiplier = calculate_effort_multiplier(value_effortMultiplier);

   double PM_adjusted = PM_nominal*effort_multiplier;
   cout<<"\nPM nominal is :- "<<PM_nominal;
   cout<<"\nPM adjusted is :- "<<PM_adjusted;

   return 0;
}
```

OUTPUT



```
ENTER THE Software size :- 50

ENTER THE SCALE FACTOR
0 Very Low
1 Low
2 Nominal
3 High
4 Very High
5 Extra High


Precedent ness :- 1

Development flexibility :- 3

Architecture/ Risk resolution :- 2

Team cohesion :- 1

Process maturity :- 2

ENTER THE Effort multiplier
0 Extra Low
1 Very Low
2 Low
3 Nominal
4 High
5 Very High
6 Extra High


RCPX :- 3

RUSE :- 3

PDIF :- 4

PERS :- 4

PREX :- 3

FCIL :- 3

SCED :- 3

PM nominal is :- 194.412
PM adjusted is :- 208.157
```

PRACTICAL - 4

Aim: Case **s**tudy **o**f **I**ntelliJ

CODE

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.

Metrics and Measurements:

- Present key metrics to quantify the impact of IntelliJ IDEA. For example:
  - **Development Time**: Measure the reduction in development time compared to the previous tools or methods.
  - **Bug Reduction**: Highlight any decrease in the number of bugs or issues identified during development.
  - **Code Quality**: Use code analysis tools to showcase improvements in code quality metrics.
  - **Collaboratio**n: Measure the impact on team collaboration and communication.
  - **Learning Curve**: Evaluate the ease with which the development team adapted to IntelliJ IDEA.

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.

<u>Solution using IntelliJ IDEA:</u>

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.

<u>Results:</u>

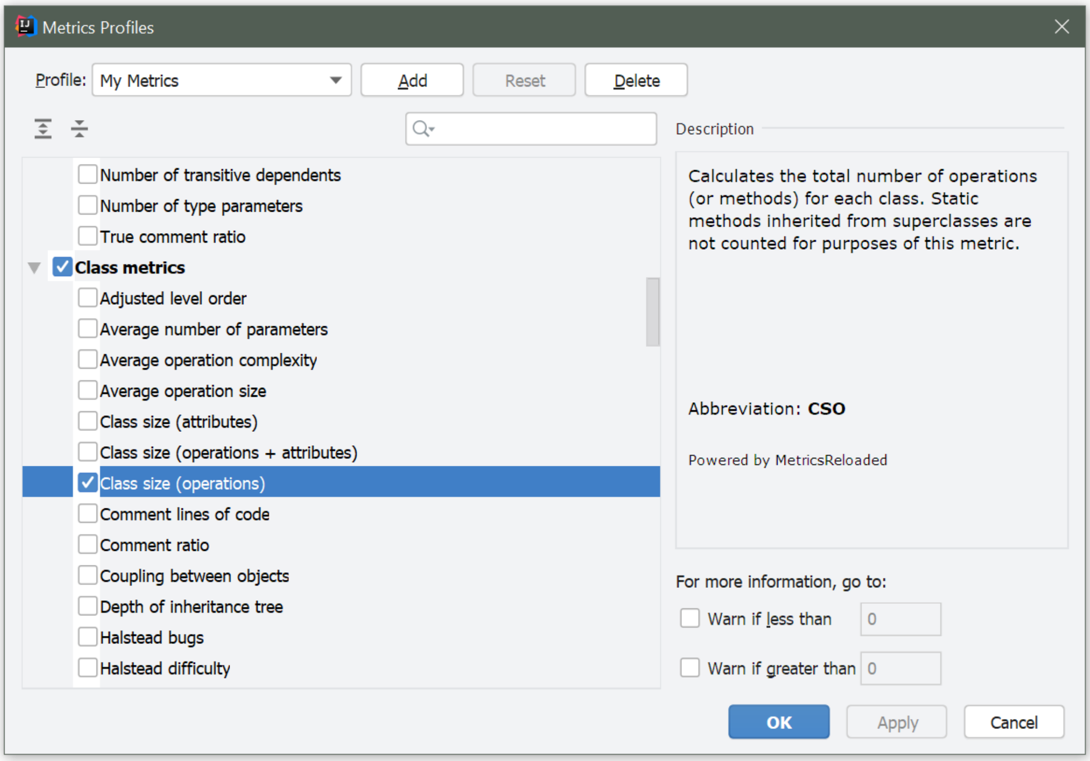
Increased Developer Productivity: The intuitive interface and robust feature set of IntelliJ IDEA significantly boosted developers' productivity, enabling them to focus more on coding logic and less on mundane tasks.

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.



Profile: My Metrics

- [ ] Number of transitive dependents
- [ ] Number of type parameters
- [ ] True comment ratio
- [x] **Class metrics**
  - [ ] Adjusted level order
  - [ ] Average number of parameters
  - [ ] Average operation complexity
  - [ ] Average operation size
  - [ ] Class size (attributes)
  - [ ] Class size (operations + attributes)
  - [x] Class size (operations)
  - [ ] Comment lines of code
  - [ ] Comment ratio
  - [ ] Coupling between objects
  - [ ] Depth of inheritance tree
  - [ ] Halstead bugs
  - [ ] Halstead difficulty

Calculates the total number of operations (or methods) for each class. Static methods inherited from superclasses are not counted for purposes of this metric.

Abbreviation: **CSO**

Powered by MetricsReloaded

For more information, go to:

- [ ] Warn if less than 0
- [ ] Warn if greater than 0

PRACTICAL - 5

Aim: Case study of Designite tool

CODE

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.

Metrics and Measurements:

The several key metrics throughout this process:

- **Design Flaws Detected:** [X%] increase in the identification of design flaws leading to proactive problem-solving.
- **Code Duplication Reduction:** A substantial [X%] decrease in code duplication, streamlining our codebase for better maintainability.
- **Code Smells Addressed:** Designite helped us address [X%] more code smells, contributing to improved code readability and maintainability.

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:

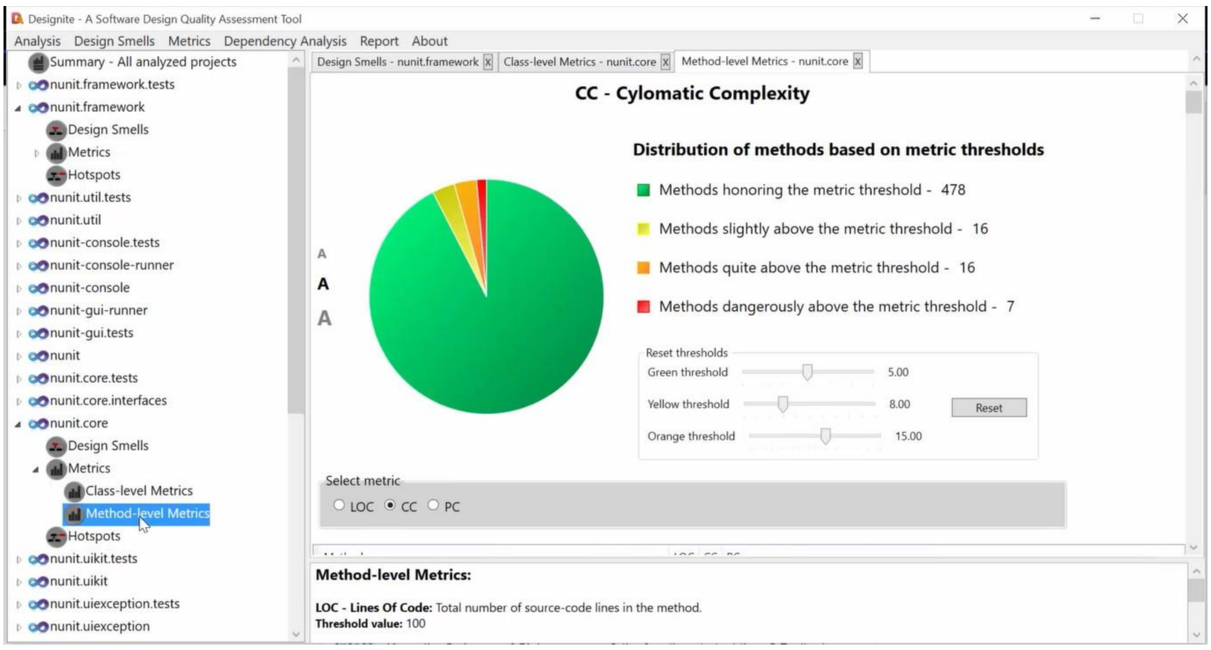
Enhanced Code Maintainability: Designite's analysis and suggestions aided in restructuring the codebase, improving modularity and reducing complex interdependencies, thereby enhancing its maintainability.

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.



CC - Cylomatic Complexity

Distribution of methods based on metric thresholds

🟩 Methods honoring the metric threshold - 478

🟨 Methods slightly above the metric threshold - 16

🟧 Methods quite above the metric threshold - 16

🟥 Methods dangerously above the metric threshold - 7

Reset thresholds

Green threshold	5.00
Yellow threshold	8.00
Orange threshold	15.00

Select metric

○ LOC [✓ CC] ○ PC

Method-level Metrics:

LOC - Lines Of Code: Total number of source-code lines in the method.
Threshold value: 100

15

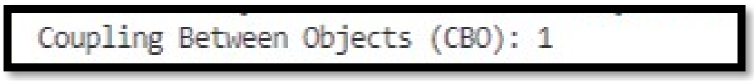
PRACTICAL - 6

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

CODE

```
#include <iostream>
#include <vector>
#include <unordered_set>
class ClassA {
public:
   void methodA() {
      std::cout << "Method A in ClassA\n";
   }};
class ClassB {
public:
   void methodB(ClassA& objA) {
      objA.methodA();
      std::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() {
   std::vector<std::unordered_set<std::string>> dependencies;
   std::unordered_set<std::string> dependencySet;
   dependencySet.insert("ClassA");
   dependencies.push_back(dependencySet);
   int cbo = calculateCBO(dependencies);
   std::cout << "Coupling Between Objects (CBO): " << cbo << std::endl;
   return 0;
}
```

OUTPUT



```
Coupling Between Objects (CBO): 1
```

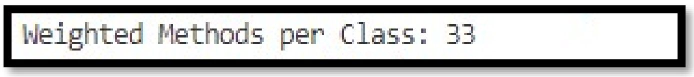
PRACTICAL - 7

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

CODE

```
#include <iostream>
#include <vector>

class MyClass {
private:
   std::vector<int> methodComplexities;

public:

   void addMethodComplexity(int complexity) {
      methodComplexities.push_back(complexity);
   }


   int calculateWMC() const {
      int totalComplexity = 0;
      for (auto i: methodComplexities) {
         totalComplexity +=i;
      }
      return totalComplexity;
   }
};

int main() {

   MyClass myClass;


   myClass.addMethodComplexity(10);  // Complexity of method 1
   myClass.addMethodComplexity(15);  // Complexity of method 2
   myClass.addMethodComplexity(8);   // Complexity of method 3


   std::cout << "Weighted Methods per Class: " << myClass.calculateWMC() << std::endl;

   return 0;
}
```

OUTPUT



```
Weighted Methods per Class: 33
```

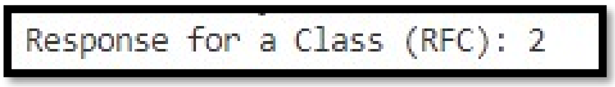
PRACTICAL - 8

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

CODE

```
#include <iostream>
#include <vector>
#include <string>
class MyClass {
public:
   void method1() {
      responseFunctions.push_back("method1");
   }
   void method2() {
      responseFunctions.push_back("method2");
   }
   int calculateRFC() {
      responseFunctions.clear();
      method1();
      method2();
      return responseFunctions.size();
   }

private:
   std::vector<std::string> responseFunctions;
};

int main() {
   MyClass myObject;
   int rfc = myObject.calculateRFC();
   std::cout << "Response for a Class (RFC): " << rfc << std::endl;

   return 0;
}
```

OUTPUT



```
Response for a Class (RFC): 2
```

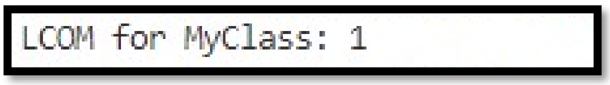
PRACTICAL - 9

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

CODE

```
#include <iostream>

class MyClass {
private:
   int variable1;
   int variable2;
public:
   void method1() {
        std::cout << "Method 1: " << variable1 << std::endl;
   }
  void method2() {
        std::cout << "Method 2: " << variable2 << std::endl;
   }
  void method3() {
      std::cout << "Method 3: " << variable1 << ", " << variable2 << std::endl;
   }

   double calculateLCOM() {
      int m = 0;
      int p = 0;


      if (shareVariables(&MyClass::method1, &MyClass::method2)) {
         p++;
      }
      if (shareVariables(&MyClass::method1, &MyClass::method3)) {
         p++;
      }
      if (shareVariables(&MyClass::method2, &MyClass::method3)) {
         p++;
      }

      m = 3; // Number of methods in MyClass


      if (m == 0) {
         return 0; // Avoid division by zero
      } else {
         return int((p) / m);
      }
   }


   bool shareVariables(void (MyClass::*method1)(), void (MyClass::*method2)()) {
        return true;
    }
};

int main() {
    MyClass myObject;

    double lcom = myObject.calculateLCOM();

    std::cout << "LCOM for MyClass: " << lcom << std::endl;

    return 0;
}
```

OUTPUT



```
LCOM for MyClass: 1
```

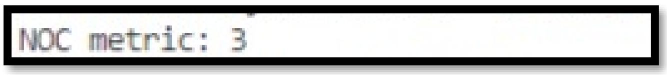
PRACTICAL - 10

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:

};
class ChildClass1 : public BaseClass {
public:

};
class ChildClass2 : public BaseClass {
public:

};
class ChildClass3 : public BaseClass {
public:

};
int calculateNOC(const std::vector<BaseClass*>& classes) {
   return classes.size();
}
int main() {
   ChildClass1 child1;
   ChildClass2 child2;
   ChildClass3 child3;
   std::vector<BaseClass*> classVector = {&child1, &child2, &child3};
   int nocValue = calculateNOC(classVector);
   std::cout << "NOC metric: " << nocValue << std::endl;
   return 0;
}
```

OUTPUT



```
NOC metric: 3
```

PRACTICAL - 11

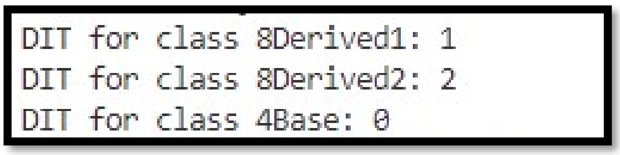
Aim: Write a program to find the Depth of Inheritance (DIT) metric given in Chidamber & Kemerer Metrics.

CODE

```
include <iostream>
#include <unordered_set>
#include <typeinfo>

class Base {

};

class Derived1 : public Base {

};

class Derived2 : public Derived1 {

};

class DITCalculator {
public:
   static int calculateDIT(std::unordered_set<const std::type_info*>& visitedTypes, const
std::type_info& type) {
dependencies
      if (visitedTypes.count(&type) > 0) {
         return 0;
      }

      visitedTypes.insert(&type);

      const std::type_info* baseType = getBaseType(type);

      if (baseType != nullptr) {
         int baseTypeDepth = calculateDIT(visitedTypes, *baseType);
         return baseTypeDepth + 1;
      }

      return 0;
   }

private:
   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() {
    std::unordered_set<const std::type_info*> types;
    types.insert(&typeid(Base));
    types.insert(&typeid(Derived1));
    types.insert(&typeid(Derived2));
    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;
}
```

<u>OUTPUT</u>



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