SOFTWARE ENGINEERING
PRACTICAL FILE
(CO 301)

NAME - PRINCE PIYUSH

ROLL NO - 2K16/CO/236

BRANCH - CO

BATCH - A4(GROUP G1)

SUBJECT - SOFTWARE ENGINEERING

FACULTY – MR. RAHUL GUPTA

Experiment – 1

AIM- Write a program to count number of lines in a text file in c/c++.

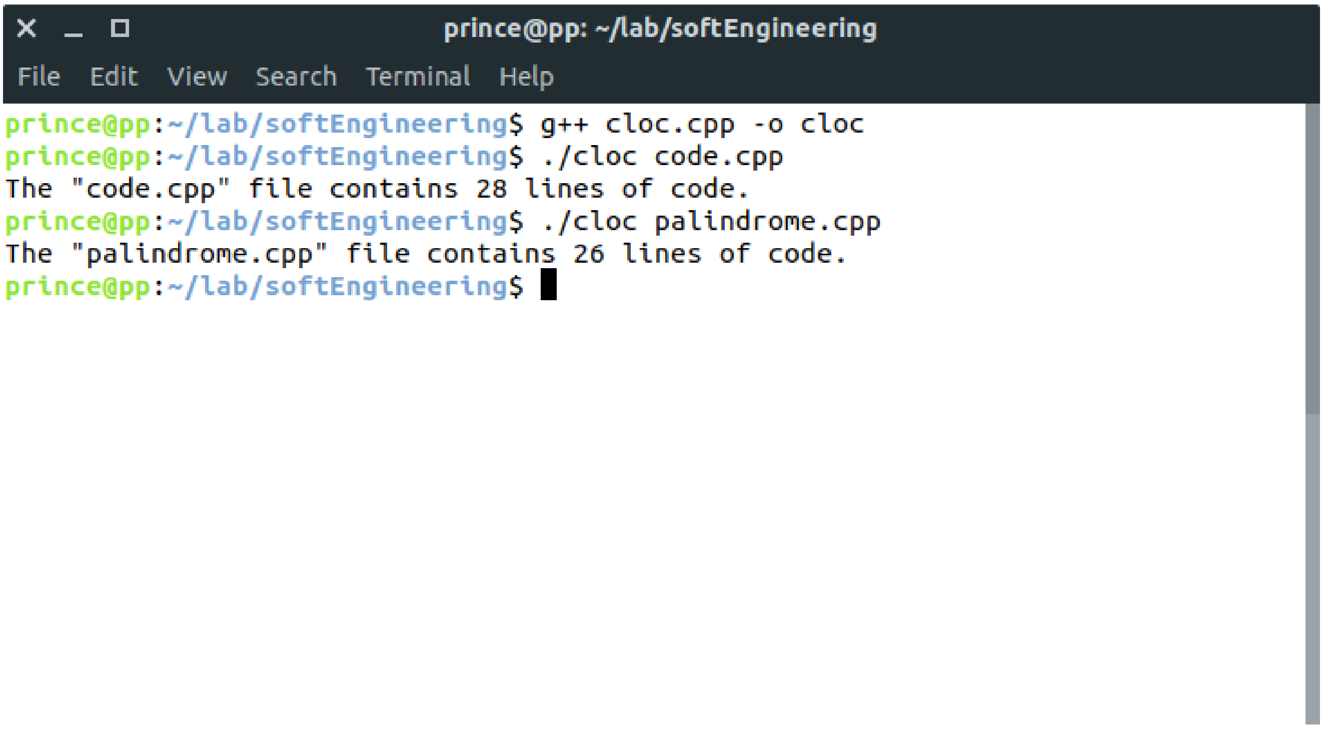
Description -
1.Open the file in an object of the fstream class.
2. Read the file line by line using the getline() function.
3. Check whether the line is a blank line.
4. Check whether the line starts with a comment.
5. If the line is a valid line of code, increase the count.

Code -

```
#include <iostream>
#include <fstream>
using namespace std;
int main(int argc,char *argv[]){
        string filename ;
        if(argc>1){
                filename = argv[1];
        }else{
                filename = "code.cpp";
        }
        ifstream file(filename,ios::in);
        string line;
        int count=0;
        while(getline(file,line)){
                bool isValid = true;
                if(line.length() == 0 ){
                        isValid = false;
                        continue;
                }
                for(int i=0;i<line.length()-1;i++){
                        if(     (line[i] == '/' && line[i+1] == '/') ||
                                (line[i] == '/' && line[i+1] == '*') ||
                                (line[i] == '*' && line[i+1] == '/')       ) {
```

```
                    isValid = false;
                    break;
            }
        }
        if(isValid){
            count++;
        }
    }
    cout<<"The \""<<filename<<"\" file contains "<<count
        <<" lines of code."<<endl;
    return 0;
}
```

Output –



```
prince@pp:~/lab/softEngineering$ g++ cloc.cpp -o cloc
prince@pp:~/lab/softEngineering$ ./cloc code.cpp
The "code.cpp" file contains 28 lines of code.
prince@pp:~/lab/softEngineering$ ./cloc palindrome.cpp
The "palindrome.cpp" file contains 26 lines of code.
prince@pp:~/lab/softEngineering$ 
```

Learnings –We learnt that it is possible to find out the exact number of lines of code in a program without including comments and blank spaces using file handling in C++.

Experiment – 2

AIM- Write a program to calculate effort using Cocomo model (Basic and Intermediate).

Description -

Cocomo (Constructive Cost Model) is a regression model based on LOC, i.e. number of Lines of Code. It is a procedural cost estimate model for software projects and often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time and quality.

Types of Models: COCOMO consists of a hierarchy of three increasingly detailed and accurate forms. Any of the three forms can be adopted according to our requirements. These are types of COCOMO model:

1. Basic COCOMO Model
2. Intermediate COCOMO Model
3. Detailed COCOMO Model

The first level, Basic COCOMO can be used for quick and slightly rough calculations of Software Costs. Its accuracy is somewhat restricted due to the absence of sufficient factor considerations. Boehm’s definition of organic, semidetached, and embedded systems:

1. Organic – A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past and also the team members have a nominal experience regarding the problem.

2. Semi-detached – A software project is said to be a Semi-detached type if the vital characteristics such as team-size, experience, knowledge of the various programming environment lie in between that of organic and Embedded. The projects classified as Semi-detached are comparatively less familiar and difficult to develop compared to the organic ones and require more experience and better guidance and creativity. E.g.: Compilers or different Embedded Systems can be considered of Semi-Detached type.

3. Embedded – A software project with requiring the highest level of complexity, creativity, and experience requirement fall under this category. Such software requires a larger team size than the other two models and also the developers need to be sufficiently experienced and creative to develop such complex models.

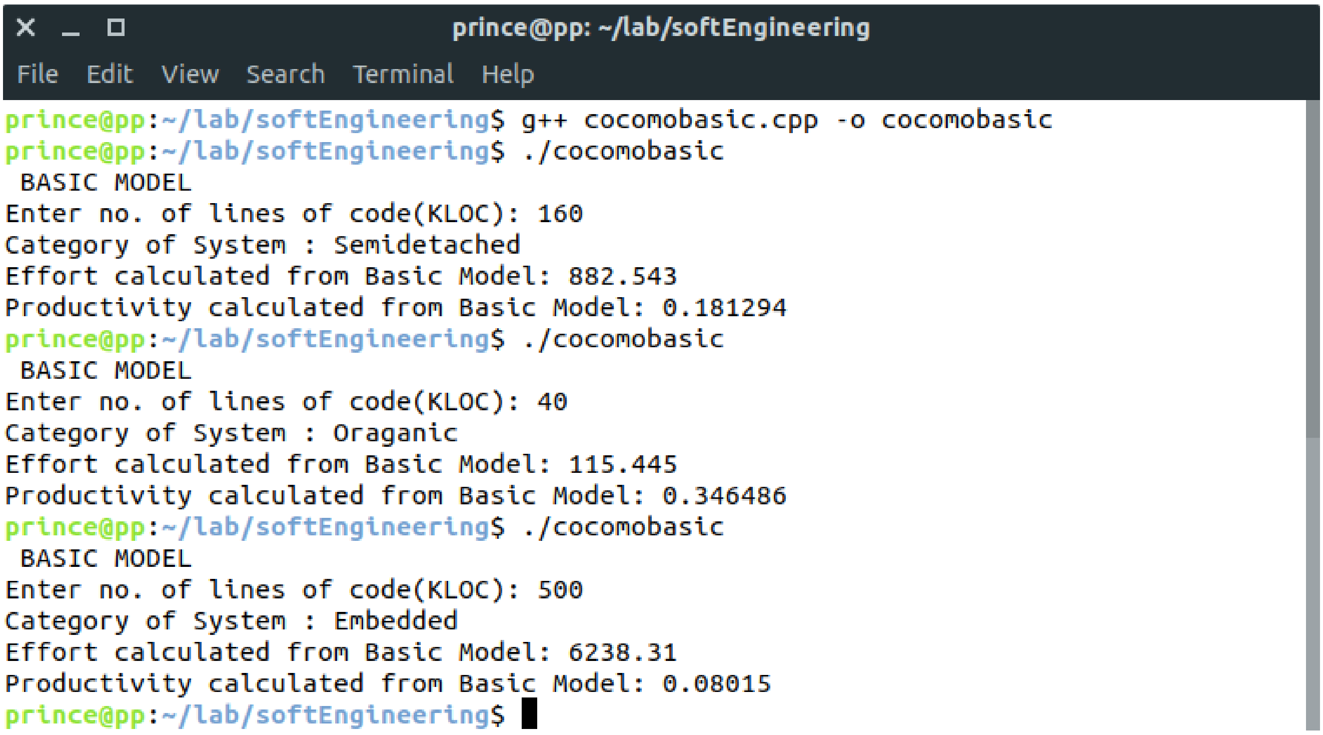
Formula for calculation of effort in basic model is E=a(KLOC)^b

CODE-

“cocomobasic.cpp” :-

```
#include<iostream>
#include<math.h>
using namespace std;
int main(){
    int ch;
    float kloc;
    cout<<" BASIC MODEL";
    cout<<"\nEnter no. of lines of code(KLOC): ";
    cin>>kloc;

    cout<<"Category of System : ";
    if(kloc<50){
        cout<<"Oraganic";
        ch = 1;
    }else if(kloc<300){
        cout<<"Semidetached";
        ch = 2;
    }else{
        cout<<"Embedded";
        ch = 3;
    }
    float a[3] = {2.4,3.0,3.6};
    float b[3] = {1.05,1.12,1.20};
    float effort,prod;
    effort = a[ch-1] * pow(kloc,b[ch-1]);
    prod = kloc / effort;
    cout<<"\nEffort calculated from Basic Model: "
        <<effort<<endl;
    cout<<"Productivity calculated from Basic Model: "
        <<prod<<endl;
    return 0;
}
```

Output :


prince@pp: ~/lab/softEngineering
File Edit View Search Terminal Help

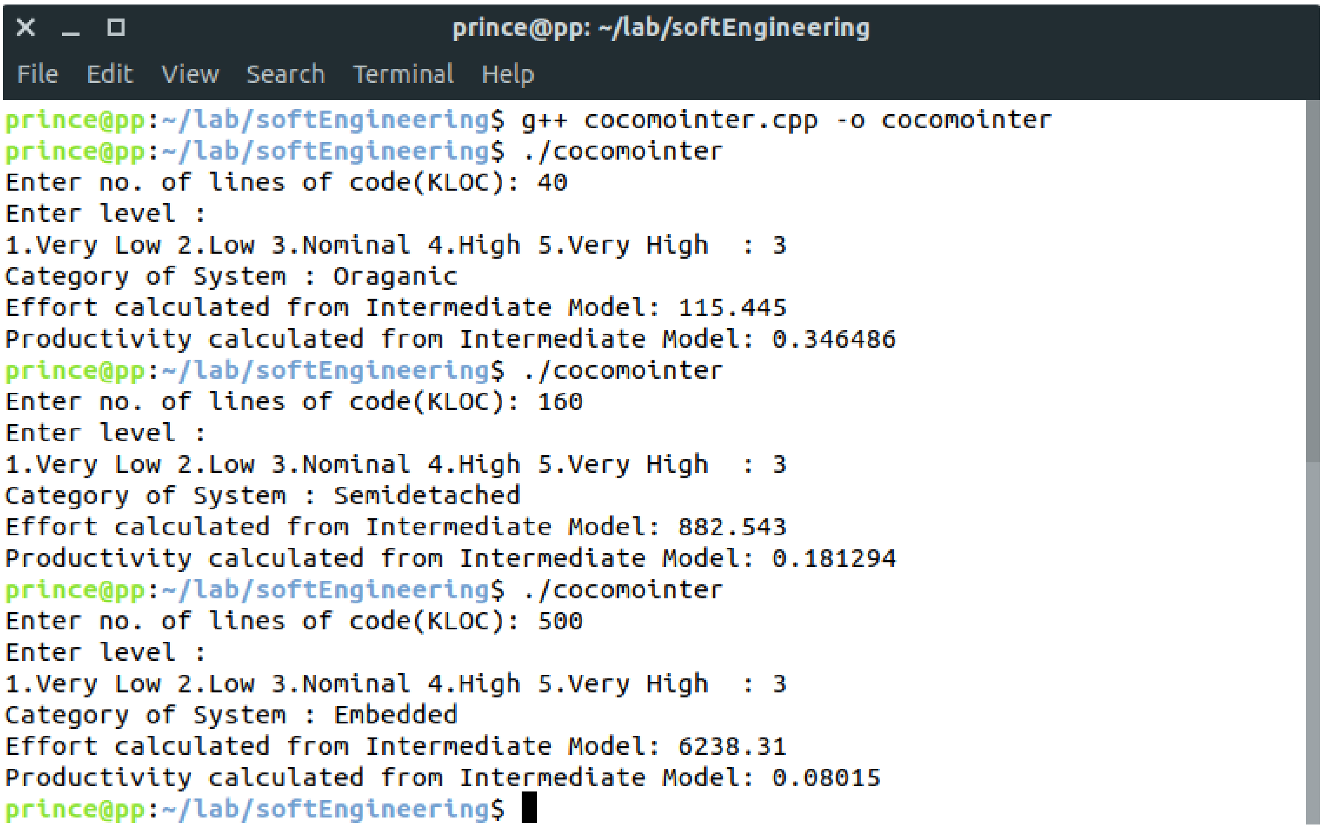
```
prince@pp:~/lab/softEngineering$ g++ cocomobasic.cpp -o cocomobasic
prince@pp:~/lab/softEngineering$ ./cocomobasic
 BASIC MODEL
Enter no. of lines of code(KLOC): 160
Category of System : Semidetached
Effort calculated from Basic Model: 882.543
Productivity calculated from Basic Model: 0.181294
prince@pp:~/lab/softEngineering$ ./cocomobasic
 BASIC MODEL
Enter no. of lines of code(KLOC): 40
Category of System : Oraganic
Effort calculated from Basic Model: 115.445
Productivity calculated from Basic Model: 0.346486
prince@pp:~/lab/softEngineering$ ./cocomobasic
 BASIC MODEL
Enter no. of lines of code(KLOC): 500
Category of System : Embedded
Effort calculated from Basic Model: 6238.31
Productivity calculated from Basic Model: 0.08015
prince@pp:~/lab/softEngineering$ 
```

“cocomointer.cpp” :-

```
#include<iostream>
#include<math.h>
using namespace std;
int main(){
    float values[15][5] = { 0.75,   0.88,   1.00,   1.15,   1.40,
                            1.00,   0.94,   1.00,   1.08,   1.16,
                            0.70,   0.85,   1.00,   1.15,   1.30,
                            1.00,   1.00,   1.00,   1.11,   1.30,
                            1.00,   1.00,   1.00,   1.06,   1.21,
                            1.00,   0.87,   1.00,   1.15,   1.30,
                            1.00,   0.94,   1.00,   1.07,   1.15,
                            1.46,   1.19,   1.00,   0.86,   0.71,
                         1.29,   1.13,   1.00,   0.91,   0.82,
                         1.42,   1.17,   1.00,   0.86,   0.70,
                         1.21,   1.10,   1.00,   0.90,   1.00,
                         1.14,   1.07,   1.00,   0.95,   1.00,
                         1.24,   1.10,   1.00,   0.91,   0.82,
                         1.24,   1.10,   1.00,   0.91,   0.83,
                         1.23,   1.08,   1.00,   1.04,   1.10
                         };
float kloc;
int level,ch;
cout<<"INTERMEDIATE MODEL";
cout<<"Enter no. of lines of code(KLOC): ";
cin>>kloc;
cout<<"Enter level :\n";
cout<<"1.Very Low 2.Low 3.Nominal 4.High 5.Very High :";
cin>> level;

cout<<"Category of System : ";
if(kloc<50){
   cout<<"Organic";
   ch = 1;
}else if(kloc<300){
   cout<<"Semidetached";
   ch = 2;
}else{
   cout<<"Embedded";
   ch = 3;
}
float a[3] = {2.4,3.0,3.6};
float b[3] = {1.05,1.12,1.20};
float effort,prod;
float eaf =1 ;
for(int i=0;i<15;i++){
   eaf *= values[i][level-1];
}
```

```
    effort = a[ch-1] * pow(kloc,b[ch-1]) * eaf;
    prod = kloc / effort;

    cout<<"\nEffort calculated from Intermediate Model: "
        <<effort<<endl;
    cout<<"Productivity calculated from Intermediate Model:"
        <<prod<<endl;
    return 0;
}
```

Output:



prince@pp: ~/lab/softEngineering
File Edit View Search Terminal Help

```
prince@pp:~/lab/softEngineering$ g++ cocomointer.cpp -o cocomointer
prince@pp:~/lab/softEngineering$ ./cocomointer
Enter no. of lines of code(KLOC): 40
Enter level :
1.Very Low 2.Low 3.Nominal 4.High 5.Very High  : 3
Category of System : Oraganic
Effort calculated from Intermediate Model: 115.445
Productivity calculated from Intermediate Model: 0.346486
prince@pp:~/lab/softEngineering$ ./cocomointer
Enter no. of lines of code(KLOC): 160
Enter level :
1.Very Low 2.Low 3.Nominal 4.High 5.Very High  : 3
Category of System : Semidetached
Effort calculated from Intermediate Model: 882.543
Productivity calculated from Intermediate Model: 0.181294
prince@pp:~/lab/softEngineering$ ./cocomointer
Enter no. of lines of code(KLOC): 500
Enter level :
1.Very Low 2.Low 3.Nominal 4.High 5.Very High  : 3
Category of System : Embedded
Effort calculated from Intermediate Model: 6238.31
Productivity calculated from Intermediate Model: 0.08015
prince@pp:~/lab/softEngineering$
```

Learnings:

We came to know about the different types of COCOMO Model, and the parameters which are taken into account in the Basic and Intermediate Models to calculate the Effort and Productivity.

Experiment – 3

AIM- Write a program for Project size estimation by Function Point Analysis.

Descriptions:

Function Point Analysis is a structured technique of problem solving. It is a method to break systems into smaller components, so they can be better understood and analysed.

Function points are a unit measure for software much like an hour is to measuring time, miles are to measuring distance or Celsius is to measuring temperature. Function Points are an ordinal measure much like other measures such as kilometres, Fahrenheit, hours, so on and so forth.

Function Point Analysis can provide a mechanism to track and monitor scope creep. Function Point Counts at the end of requirements, analysis, design, code, testing and implementation can be compared. The function point count at the end of requirements and/or designs can be compared to function points actually delivered. If the project has grown, there has been scope creep. The amount of growth is an indication of how well requirements were gathered by and/or communicated to the project team. If the amount of growth of projects declines over time it is a natural assumption that communication with the user has improved.

The Five Major Components

External Inputs (EI) - is an elementary process in which data crosses the boundary from outside to inside. This data may come from a data input screen or another application. The data may be used to maintain one or more internal logical files. The data can be either control information or business information. If the data is control information it does not have to update an internal logical file. The graphic represents a simple EI that updates 2 ILF's (FTR's).

External Outputs (EO) - an elementary process in which derived data passes across the boundary from inside to outside. Additionally, an EO may update an ILF. The data creates reports or output files sent to other applications. These reports and files are created from one or more internal logical files and external interface file. The following graphic represents on EO with 2 FTR's there is derived information (green) that has been derived from the ILF's

External Inquiry (EQ) - an elementary process with both input and output components that result in data retrieval from one or more internal logical files and external interface files. The input process does not update any Internal Logical Files, and the output side does not contain derived data. The graphic below represents an EQ with two ILF's and no derived data.

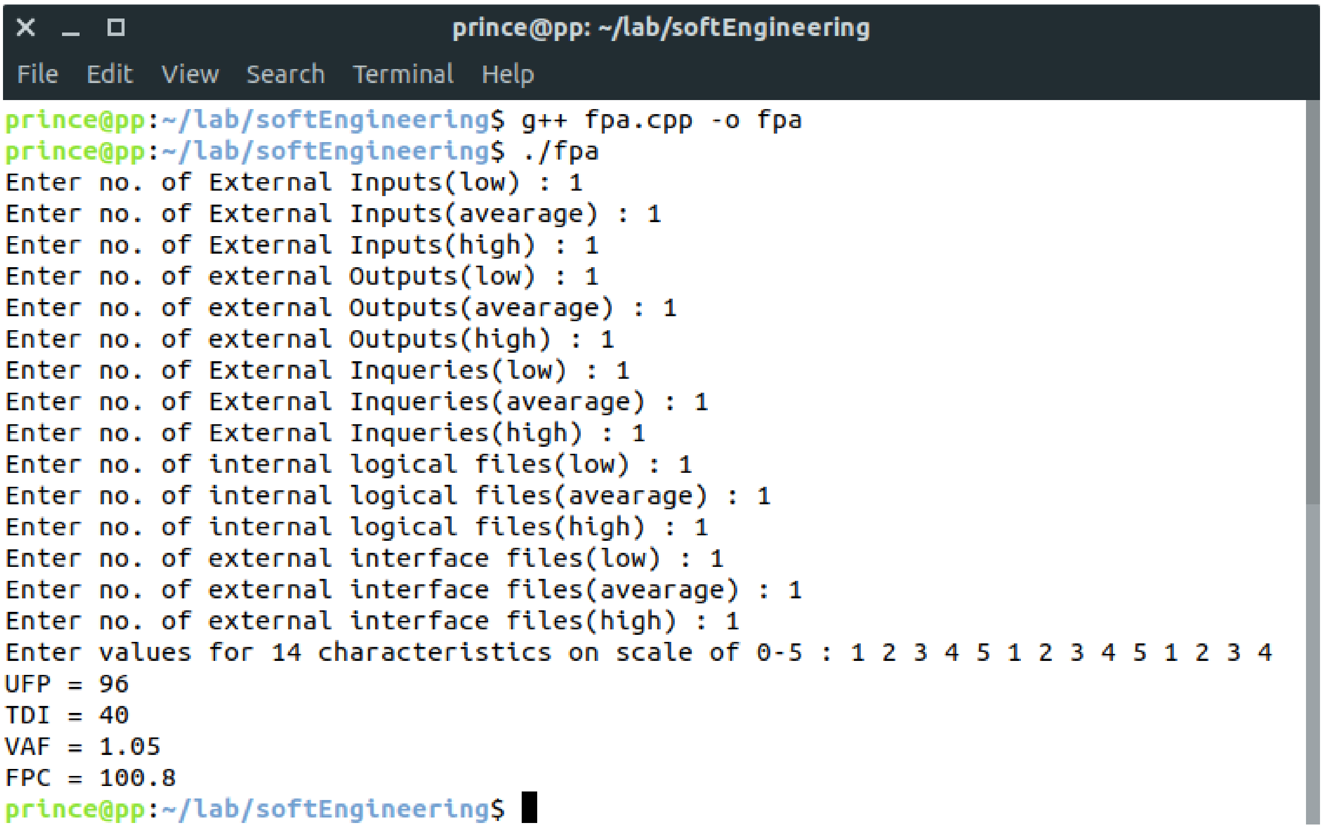
Internal Logical Files (ILF’s) - a user identifiable group of logically related data that resides entirely within the applications boundary and is maintained through external inputs.

External Interface Files (EIF’s) - a user identifiable group of logically related data that is used for reference purposes only. The data resides entirely outside the application and is maintained by another application. The external interface file is an internal logical file for another application.

Code:

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    int weights[5][3] = {3,4,6,4,5,7,3,4,6,7,10,15,5,7,10} ;
    int UFP = 0, tdi=0,rating;
    string f[] = {"External Inputs",
                        "external Outputs",
                        "External Inqueries",
                        "internal logical files",
                        "external interface files"};
    string c[] = {"low","avearage","high"};
    int input[5][3];
    for (int i = 0; i < 5; ++i){
        for (int j = 0; j < 3; ++j){
            cout<<"Enter no. of "<<f[i]<<"("<<c[j]<<") : ";
            cin>>input[i][j];
            UFP += input[i][j]*weights[i][j];
        }
    }

int val;
    cout<<"Enter values for 14 characteristics on scale of 0-5 : ";
    for (int i = 0; i < 14; ++i){
        cin>>val;
        tdi += val;
    }
    float VAF = (tdi* 0.01) + 0.65;
    float FPC = UFP*VAF;
    cout<<"UFP = "<<UFP<<endl;
    cout<<"TDI = "<<tdi<<endl;
    cout<<"VAF = "<<VAF<<endl;
    cout<<"FPC = "<<FPC<<endl;
    return 0;
}
```

Output:



prince@pp: ~/lab/softEngineering
File Edit View Search Terminal Help

```
prince@pp:~/lab/softEngineering$ g++ fpa.cpp -o fpa
prince@pp:~/lab/softEngineering$ ./fpa
Enter no. of External Inputs(low) : 1
Enter no. of External Inputs(avearage) : 1
Enter no. of External Inputs(high) : 1
Enter no. of external Outputs(low) : 1
Enter no. of external Outputs(avearage) : 1
Enter no. of external Outputs(high) : 1
Enter no. of External Inqueries(low) : 1
Enter no. of External Inqueries(avearage) : 1
Enter no. of External Inqueries(high) : 1
Enter no. of internal logical files(low) : 1
Enter no. of internal logical files(avearage) : 1
Enter no. of internal logical files(high) : 1
Enter no. of external interface files(low) : 1
Enter no. of external interface files(avearage) : 1
Enter no. of external interface files(high) : 1
Enter values for 14 characteristics on scale of 0-5 : 1 2 3 4 5 1 2 3 4 5 1 2 3 4
UFP = 96
TDI = 40
VAF = 1.05
FPC = 100.8
prince@pp:~/lab/softEngineering$ 
```

Learnings:

We came to know about the FPA method for Project Size Estimation and the different steps involved in it.

Experiment - 4

AIM: Program to evaluate the Halstead Metrics

DESCRIPTION:

A computer program is an implementation of an algorithm considered to be a collection of tokens which can be classified as either operators or operands. Halstead's metrics are included in a number of current commercial tools that count software lines of code. By counting the tokens and determining which are operators and which are operands, the following base measures can be collected:

n1 = Number of distinct operators.
n2 = Number of distinct operands.
N1 = Total number of occurrences of operators.
N2 = Total number of occurrences of operands.

In addition to the above, Halstead defines the following:

n1* = Number of potential operators.
n2* = Number of potential operands.

Halstead refers to n1* and n2* as the minimum possible number of operators and operands for a module and a program respectively. This minimum number would be embodied in the programming language itself, in which the required operation would already exist (for example, in C language, any program must contain at least the definition of the function main()), possibly as a function or as a procedure: n1* = 2, since at least 2 operators must appear for any function or procedure : 1 for the name of the function and 1 to serve as an assignment or grouping symbol, and n2* represents the number of parameters, without repetition, which would need to be passed on to the function or the procedure.

Halstead metrics are:

- Halstead Program Length – The total number of operator occurrences and the total number of operand occurrences.
  N = N1 + N2
  And estimated program length is, $N^{\wedge} = n1\log_2 n1 + n2\log_2$
- Halstead Vocabulary – The total number of unique operator and unique operand occurrences.
  n = n1 + n2
- Program Volume – Proportional to program size, represents the size, in bits, of space necessary for storing the program. This parameter is

dependent on specific algorithm implementation. The properties V, N, and the number of lines in the code are shown to be linearly connected and equally valid for measuring relative program size.

V = Size * (log$_2$ vocabulary) = N * log2(n)

The unit of measurement of volume is the common unit for size “bits”. It is the actual size of a program if a uniform binary encoding for the vocabulary is used. And error = Volume / 3000

- Program Difficulty – This parameter shows how difficult to handle the program is.
  D = (n1 / 2) * (N2 / n2)
  D = 1 / L
  As the volume of an implementation of a program increases, the program level decreases and the difficulty increases. Thus, programming practices such as redundant usage of operands, or the failure to use higher-level control constructs will tend to increase the volume as well as the difficulty.
- Programming Effort – Measures the amount of mental activity needed to translate the existing algorithm into implementation in the specified program language.
  E = V / L = D * V = Difficulty * Volume
- Language Level – Shows the algorithm implementation program language level. The same algorithm demands additional effort if it is written in a low level program language. For example, it is easier to program in Pascal than in Assembler.
  L’ = V / D / D
  And estimated program level is L^ =2 * (n2) / (n1) (N2)
- Intelligence Content – Determines the amount of intelligence presented (stated) in the program This parameter provides a measurement of program complexity, independently of the program language in which it was implemented.
  I = V / D
- Programming Time – Shows time (in minutes) needed to translate the existing algorithm into implementation in the specified program language.
  T = E / (f * S)

The concept of the processing rate of the human brain, developed by the psychologist John Stroud, is also used. Stoud defined a moment as the time required by the human brain requires to carry out the most elementary decision. The Stoud number S is therefore Stoud’s moments per second with:

5 <= S <= 20. Halstead uses 18. Stroud number S = 18 moments / second
seconds-to-minutes factor f = 60

ALGORITHM

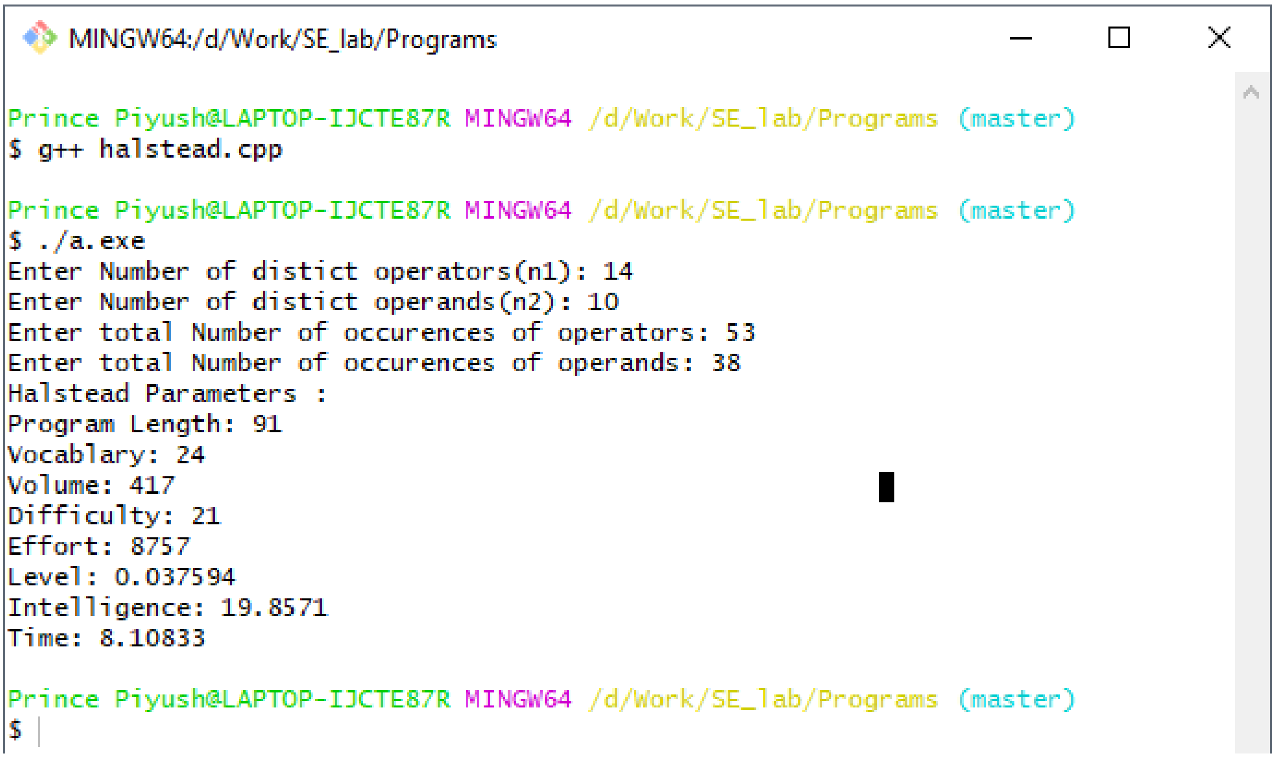
1. Calculate the distinct number of operands and operators and the total occurences of all the operands and operators.
2. Using the values calculated in step 1, calculate the following Halstead metrics as:
    - Halstead Program Length – N1+N2
    - Halstead Vocabulary(n) – n1+n2
    - Program Volume V = Size * ($\log_2$ vocabulary) = N * log2(n)
    - Program Difficulty D = (n1 / 2) * (N2 / n2)
    - Programming Effort E = V / L = D * V = Difficulty * Volume
    - Language Level L' = V / D / D
    - Intelligence Content I = V / D
    - Programming Time T = E / (f * S)

CODE

```
#include <bits/stdc++.h>

using namespace std;
float log2(int n){
   return log(n)/log(2);
}
int length(int N1,int N2){
   return N1 + N2;
}
int vocabulary(int n1,int n2){
   return n1 + n2;
}
int volume(int n,int N){
   return N*log2(n);
}
float dificulty(int n1,int n2,int N1,int N2){
   return (float(n1)/2)*(N2/n2);
```

```
}
float effort(float D,float V){
    return D*V;
}
float level(int n1,int n2,int N1,int N2){
    return (2*float(n2))/(n1*N2);
}
float intelligence(float V,float D){
    return V/D;
}
float ptime(float E){
    int f = 60;
    int S = 18;
    return E/(f*S);
}
int main(){
    int n1,n2,N1,N2;
    cout<<"Enter Number of distict operators(n1): ";
    cin>>n1;
    cout<<"Enter Number of distict operands(n2): ";
    cin>>n2;
    cout<<"Enter total Number of occurences of operators: ";
    cin>>N1;
    cout<<"Enter total Number of occurences of operands: ";
    cin>>N2;
    int N = length(N1,N2);
    int n = vocabulary(n1,n2);
    float D = dificulty(n1,n2,N1,N2);
    int V = volume(n,N);
    float e = effort(D,V);
    float l = level(n1,n2,N1,N2);
    float i = intelligence(V,D);
    float t = ptime(e);

    cout<<"Halstead Parameters :";
    cout<<"\nProgram Length: "<<N;
    cout<<"\nVocablary: "<<n;
    cout<<"\nVolume: "<<V;
    cout<<"\nDifficulty: "<<D;
    cout<<"\nEffort: "<<e;
    cout<<"\nLevel: "<<l;
    cout<<"\nIntelligence: "<<i;
    cout<<"\nTime: "<<t;

    cout<<endl;
    return 0;
}
```

OUTPUT



```
Prince Piyush@LAPTOP-IJCTE87R MINGW64 /d/Work/SE_lab/Programs (master)
$ g++ halstead.cpp

Prince Piyush@LAPTOP-IJCTE87R MINGW64 /d/Work/SE_lab/Programs (master)
$ ./a.exe
Enter Number of distict operators(n1): 14
Enter Number of distict operands(n2): 10
Enter total Number of occurences of operators: 53
Enter total Number of occurences of operands: 38
Halstead Parameters :
Program Length: 91
Vocablary: 24
Volume: 417
Difficulty: 21
Effort: 8757
Level: 0.037594
Intelligence: 19.8571
Time: 8.10833

Prince Piyush@LAPTOP-IJCTE87R MINGW64 /d/Work/SE_lab/Programs (master)
$
```

LEARNING OUTCOMES

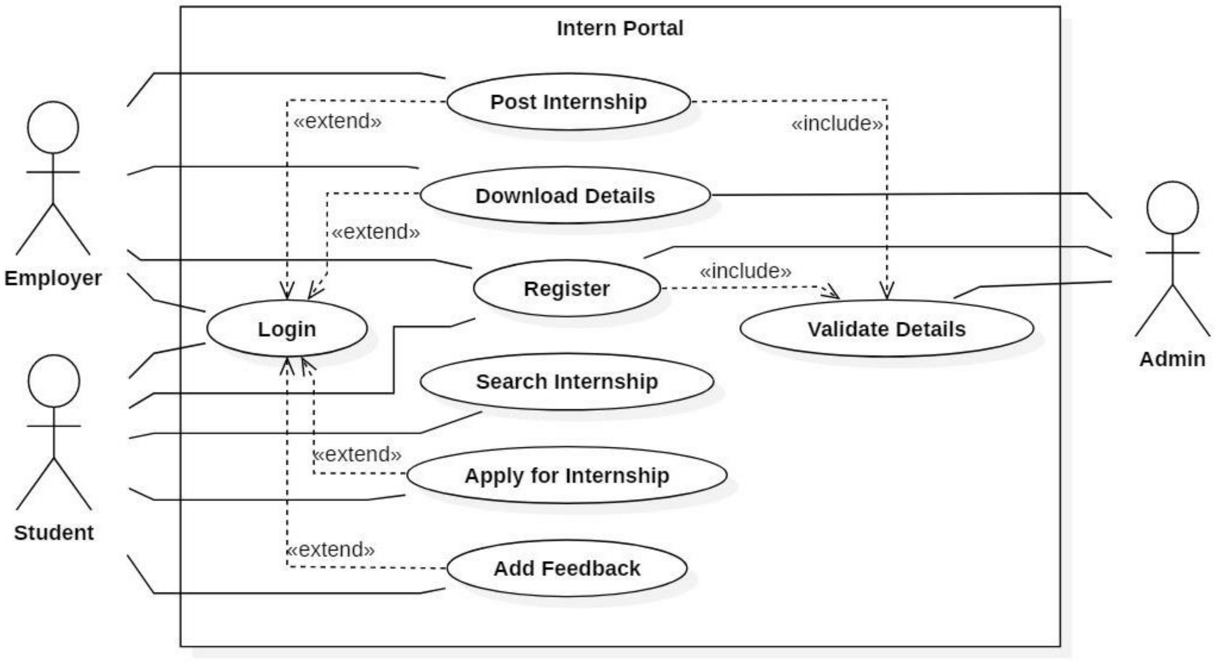
We learnt about the Halstead Metrics which evaluate our software on various parameters which are dependent directly or indirectly on the lines of code in our program and the number of operands and the number of operators and their total occurences.

Experiment - 5

AIM: To draw a Use-Case Diagram

DESCRIPTION: A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses. Here, the use case diagram of Intern Portal has been made.

USE CASE DIAGRAM (Intern Portal)



LEARNING OUTCOMES: We learnt how to make a use case diagram and we came to know about the different concepts involved while making a use case diagram for example the actors involved, the different entities etc.

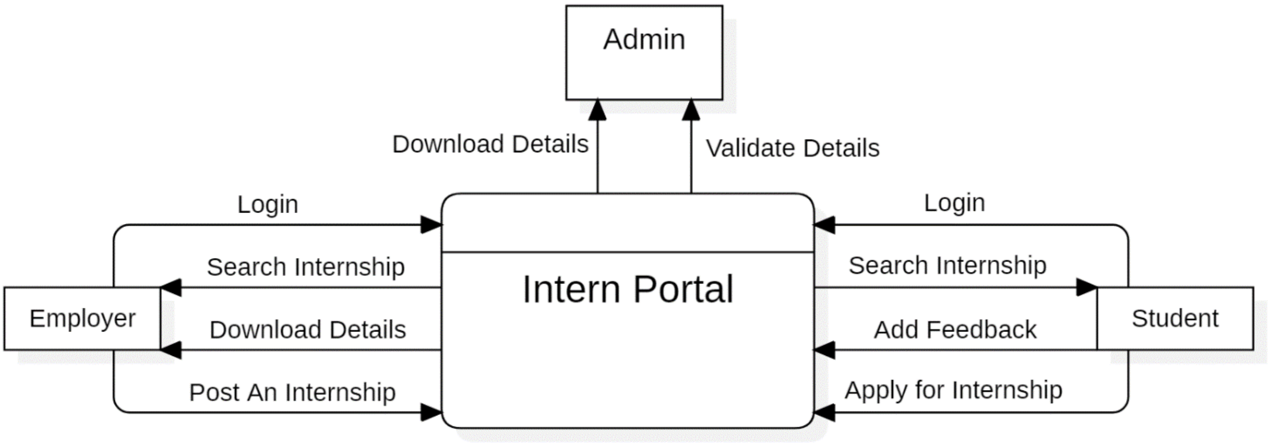
Experiment - 6

AIM: To make a Level-0 and Level-1 DFD

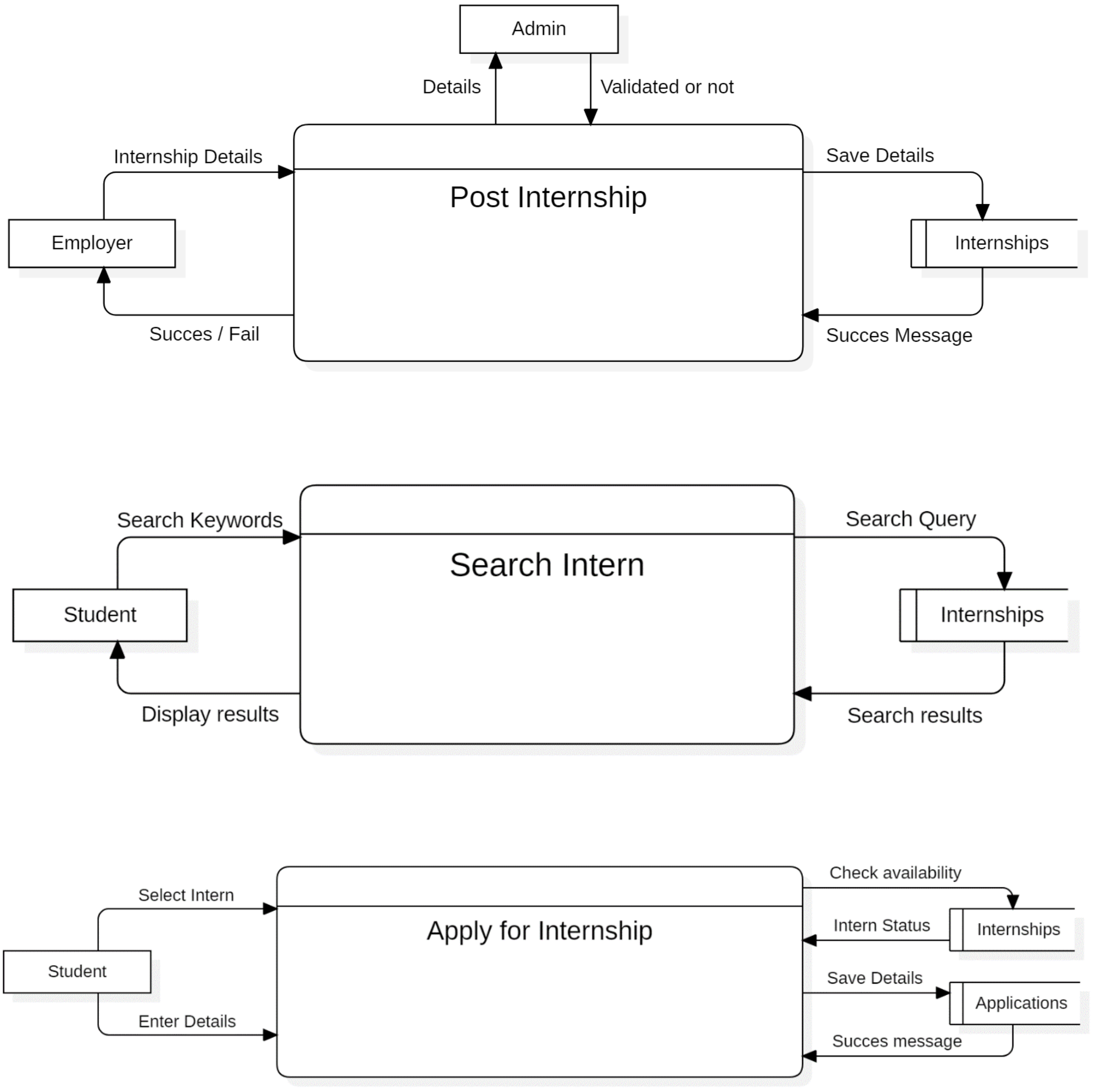
DESCRIPTION: A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modelling its process aspects. A DFD is often used as a preliminary step to create an overview of the system without going into great detail, which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design).

A DFD shows what kind of information will be input to and output from the system, how the data will advance through the system, and where the data will be stored. It does not show information about process timing or whether processes will operate in sequence or in parallel, unlike a traditional structured flowchart which focuses on control flow, or a UML activity workflow diagram, which presents both control and data flows as a unified model. Here, the Level -0 and Level-1 DFD of the Intern Portal has been made.

LEVEL-0 DFD (Intern Portal)



LEVEL-1 DFD (Intern Portal)



LEARNING OUTCOMES: We learnt how to make Level-0 DFD and Level-1 DFD where we also learnt the different types of objects involved, for example, the database entity etc.

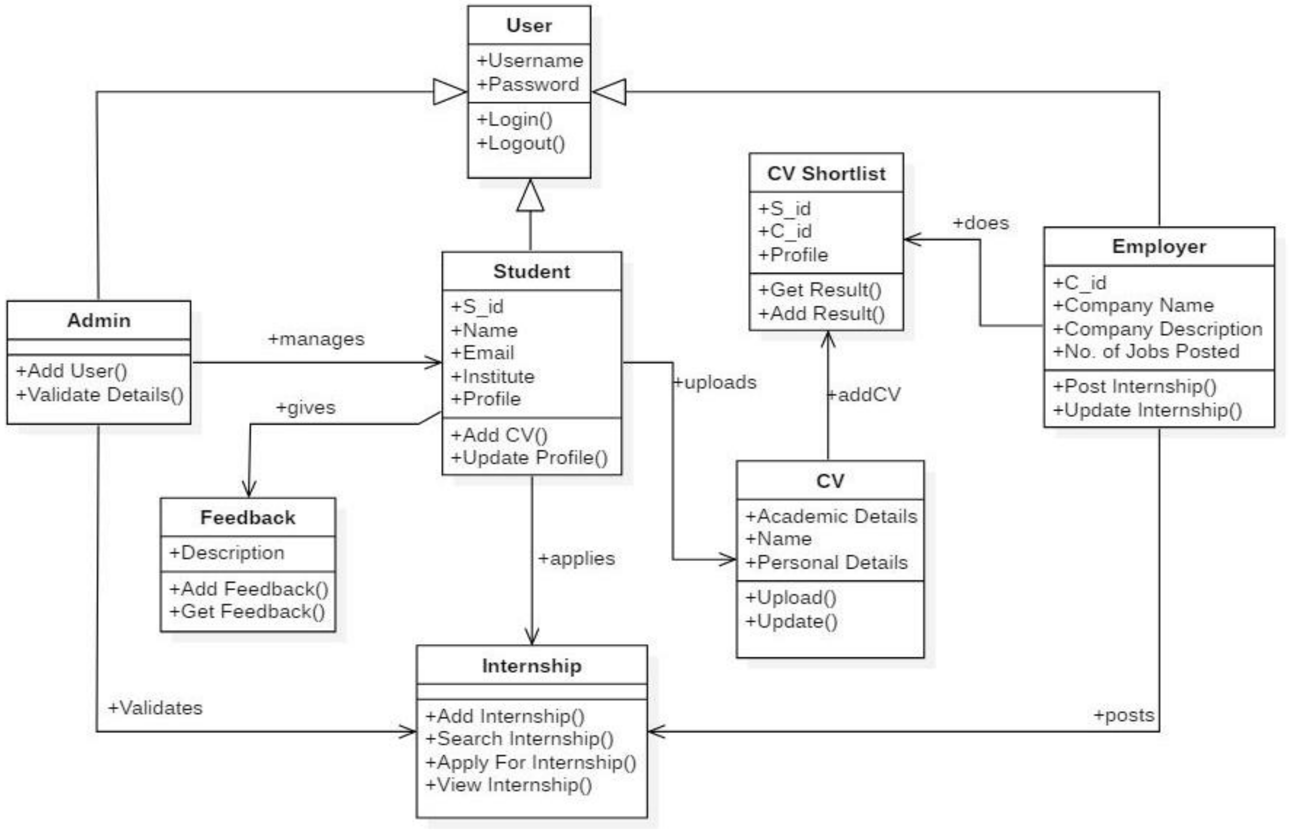
Experiment – 7

AIM: To make a class diagram

DESCRIPTION: The class diagram is the main building block of object-oriented modelling. It is used for general conceptual modelling of the systematic of the application, and for detailed modelling translating the models into programming code. Class diagrams can also be used for data modelling. The classes in a class diagram represent both the main elements, interactions in the application, and the classes to be programmed. In the diagram, classes are represented with boxes that contain three compartments:

- The top compartment contains the name of the class. It is printed in bold and centred, and the first letter is capitalized.
- The middle compartment contains the attributes of the class. They are left-aligned and the first letter is lowercase.
- The bottom compartment contains the operations the class can execute. They are also left-aligned and the first letter is lowercase.

CLASS DIAGRAM (Intern Portal)



LEARNING OUTCOMES: We learnt how to make a class diagram and the different kinds of associations in a class diagram and the different parts of a class entity.

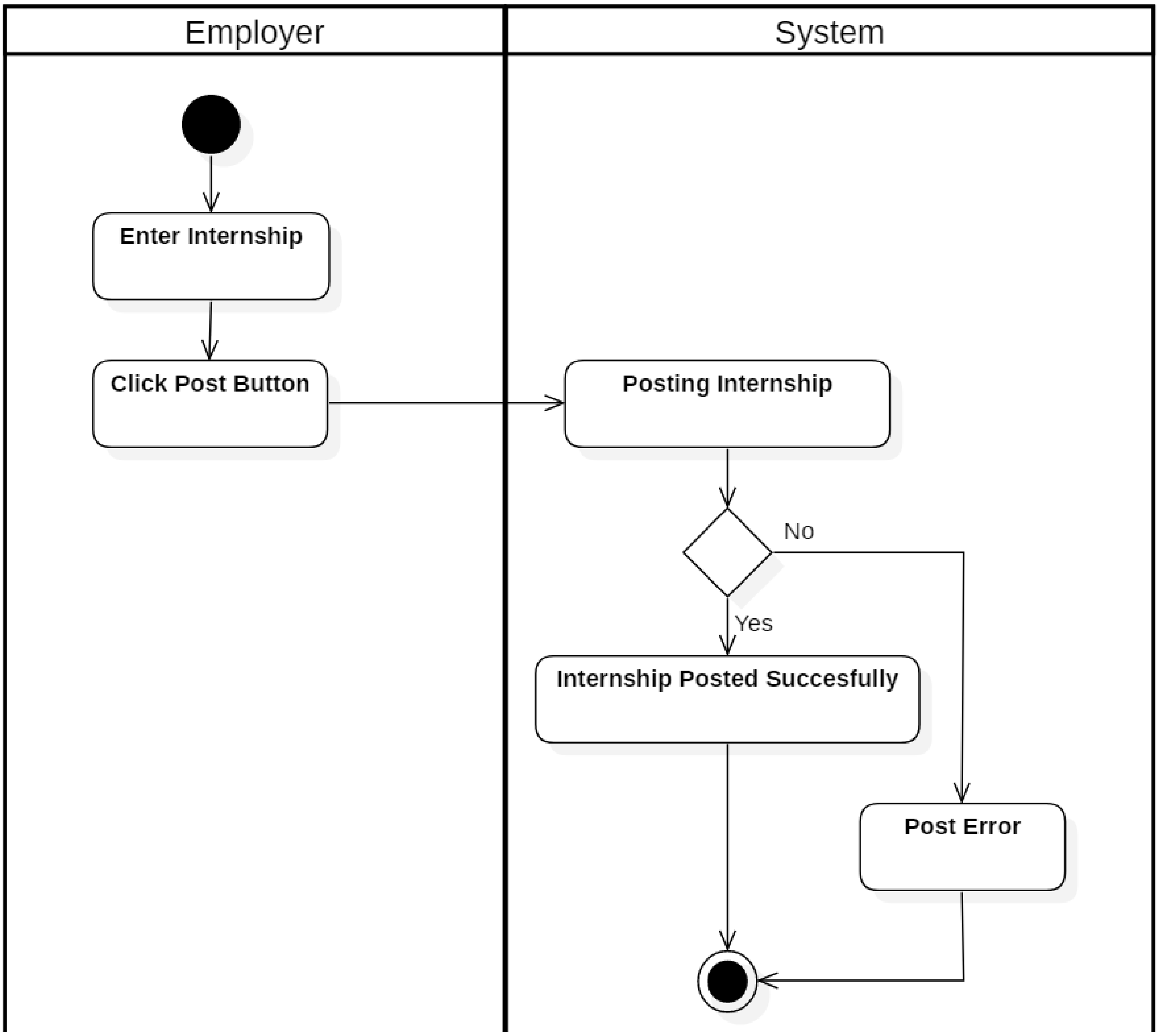
Experiment – 8

AIM: To make an activity diagram

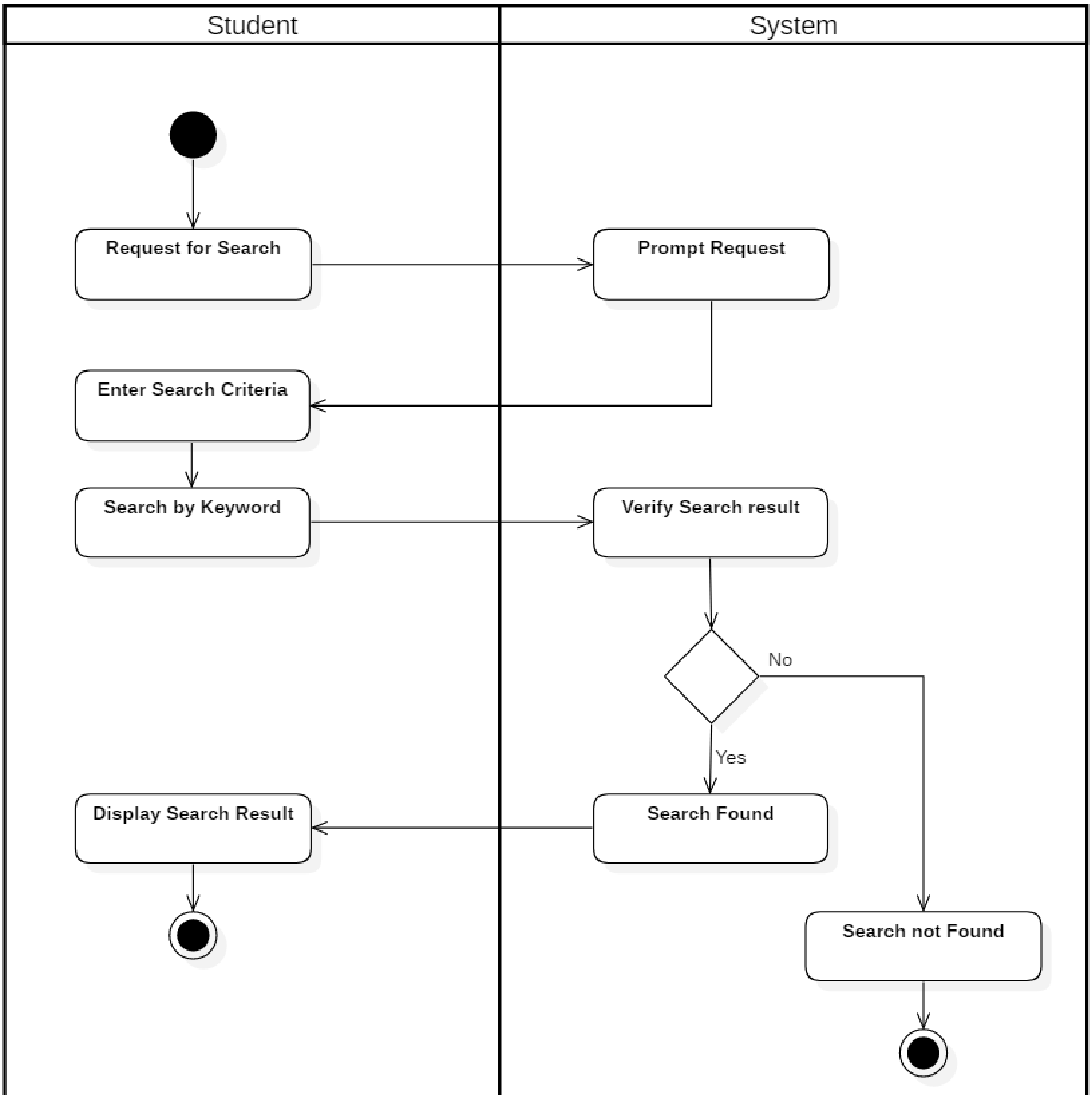
DESCRIPTION: Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modelling Language, activity diagrams are intended to model both computational and organizational processes (i.e., workflows), as well as the data flows intersecting with the related activities. Although activity diagrams primarily show the overall flow of control, they can also include elements showing the flow of data between activities through one or more data stores. Here, we have made an Activity diagram representing the Intern Portal.

ACTIVITY DIAGRAM (Intern Portal)

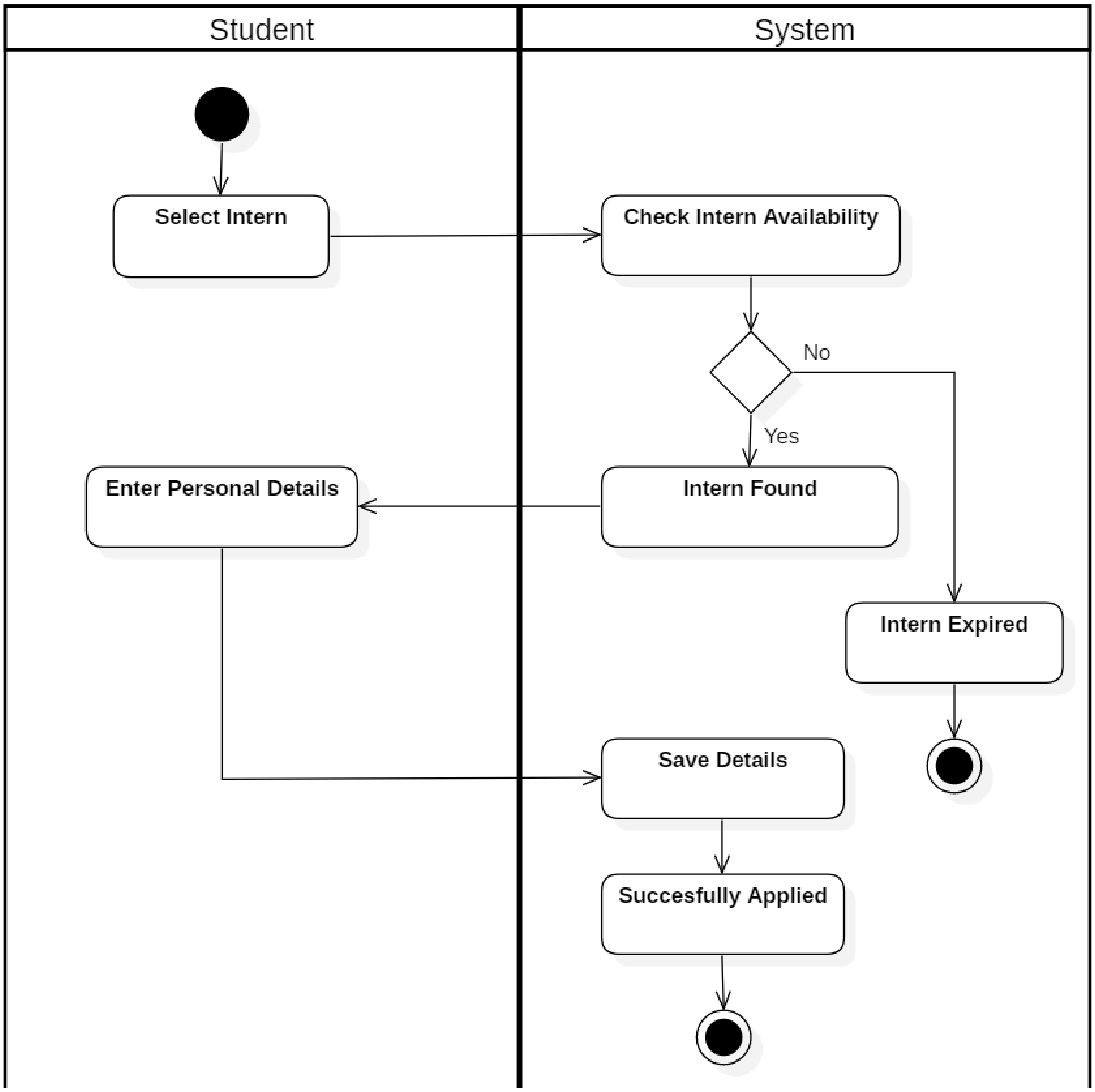
Post an Internship



Search Internship



Apply for Internship



LEARNING OUTCOMES: We learnt how to make an activity diagram and the different kind of activities that are involved.

Experiment – 9

AIM: To make a sequence diagram

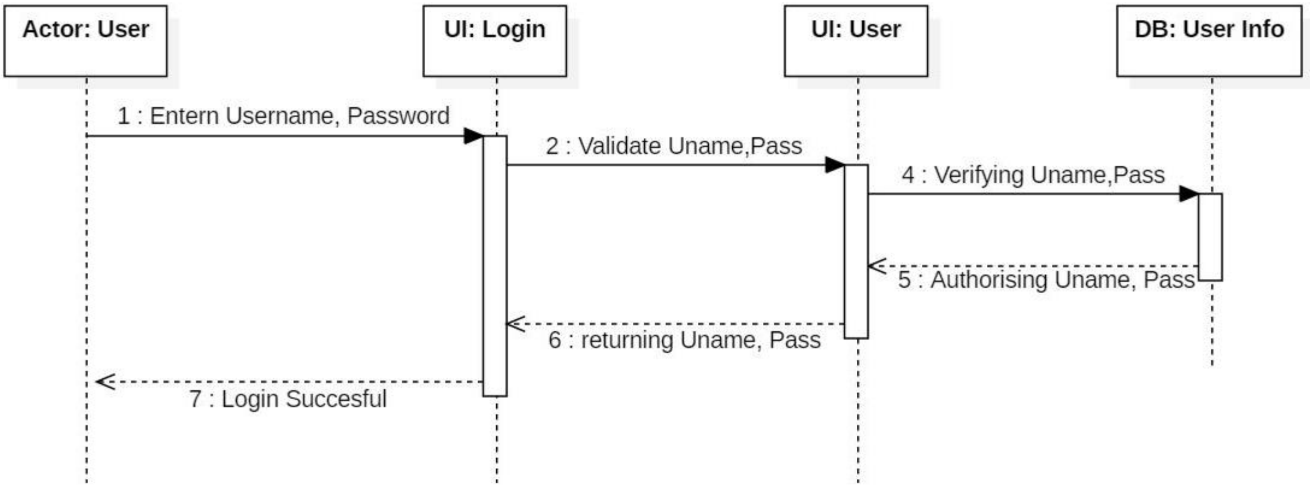
DESCRIPTION: A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios.

A sequence diagram shows, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

Here, we have made the sequence diagrams for some use cases of the Intern Portal.

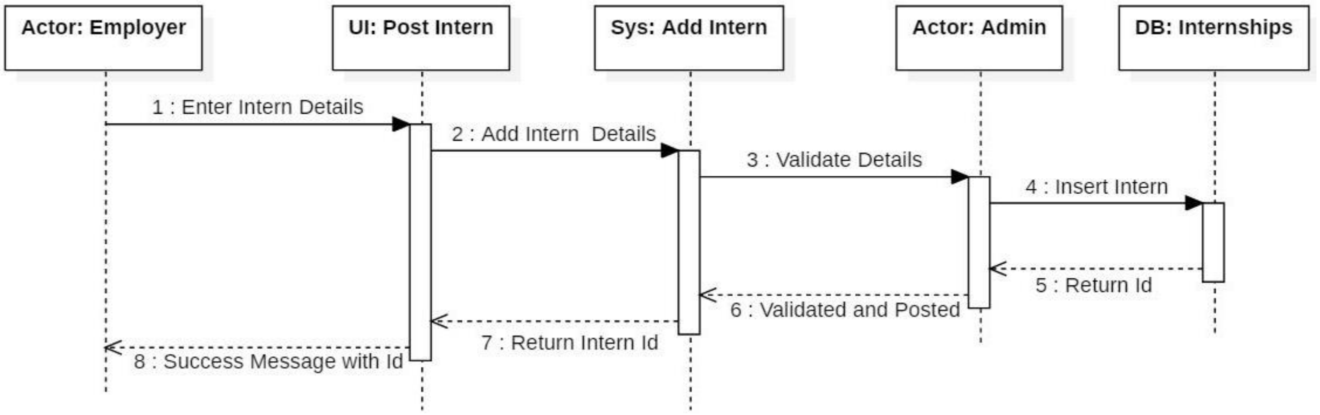
SEQUENCE DIAGRAM (Intern Portal)

Login

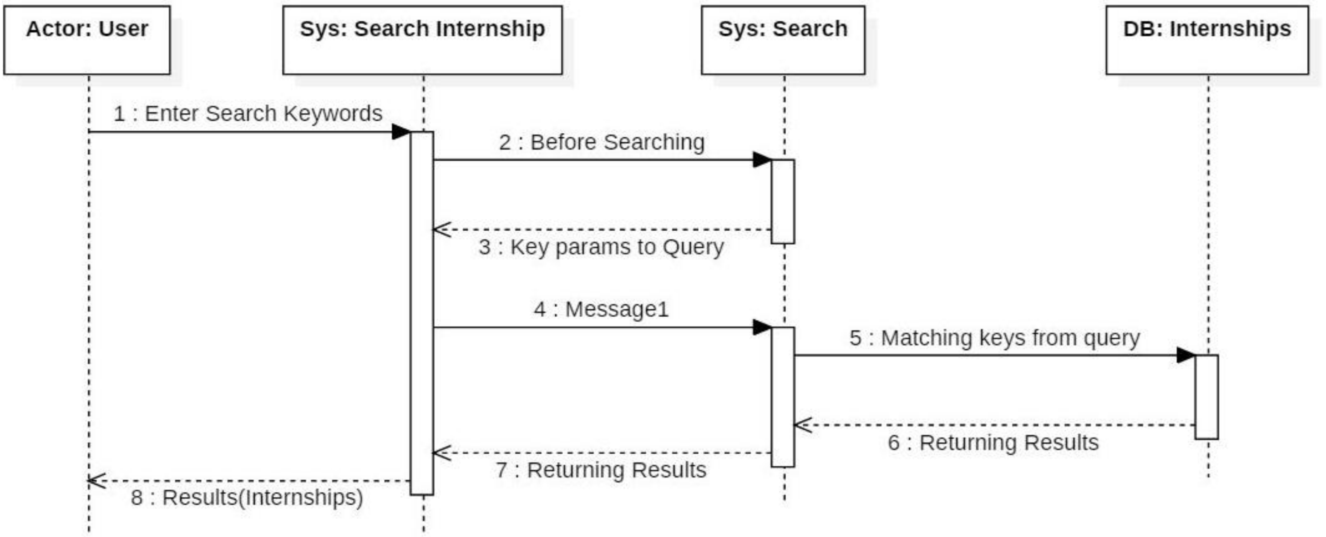


LEARNING OUTCOMES: We learnt how to make the Sequence Diagrams for different use cases of our model. We also came to know about the different kinds of objects in the Sequence Diagrams.

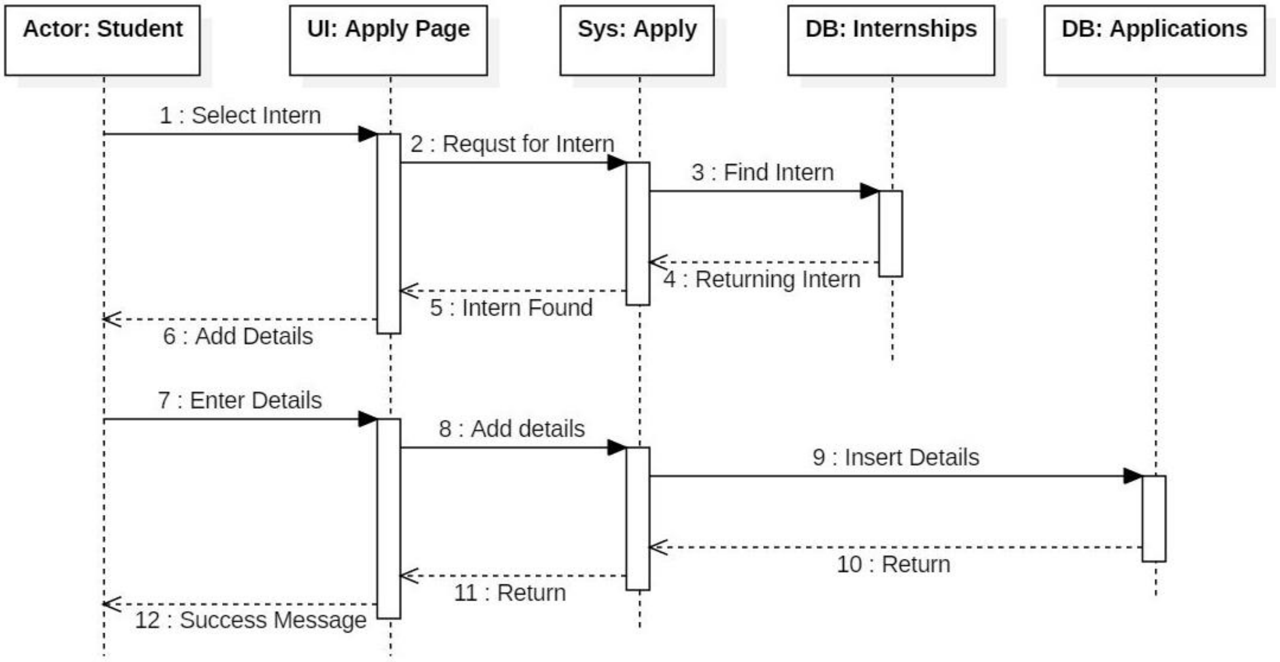
Post an Internship



Search Internship



Apply for Internship



Actor: Student
UI: Apply Page
Sys: Apply
DB: Internships
DB: Applications
1 : Select Intern
2 : Requst for Intern
3 : Find Intern
4 : Returning Intern
5 : Intern Found
6 : Add Details
7 : Enter Details
8 : Add details
9 : Insert Details
10 : Return
11 : Return
12 : Success Message