

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 -

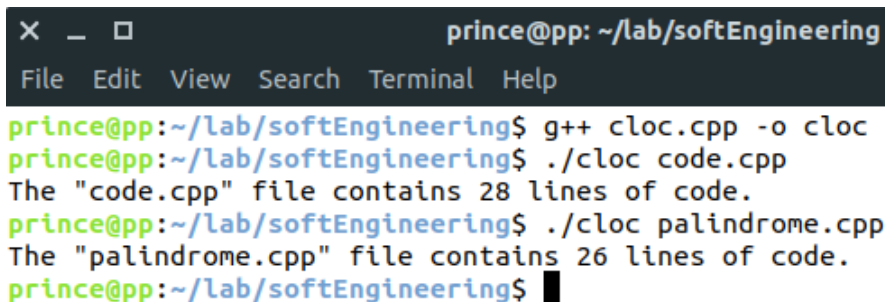
```
#include <iostream>
#include <fstream>
using namespace std;
int main(int argc,char *argv[]){
    string filename ;
    if(argc>1){
        filename = argv[1];
    }else{
        filename = "code.cpp";
    }
    ifstream file(filename,ios::in);
    string line;
    int count=0;
    while(getline(file,line)){
        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;
            }
        }
        count++;
    }
}
```

```

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

```

Output -



```

prince@pp: ~/lab/softEngineering
File Edit View Search Terminal Help
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 SemiDetached are comparatively less familiar and difficult to develop compared to the organic ones and require more experience and better guidance and creativity. Eg: 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<<"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;
    effort = a[ch-1] * pow(kloc,b[ch-1]);
    prod = kloc / effort;
```

}

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” :-

[illegible]

```

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<<"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;
float eaf =1 ;
for(int i=0;i<15;i++){

```

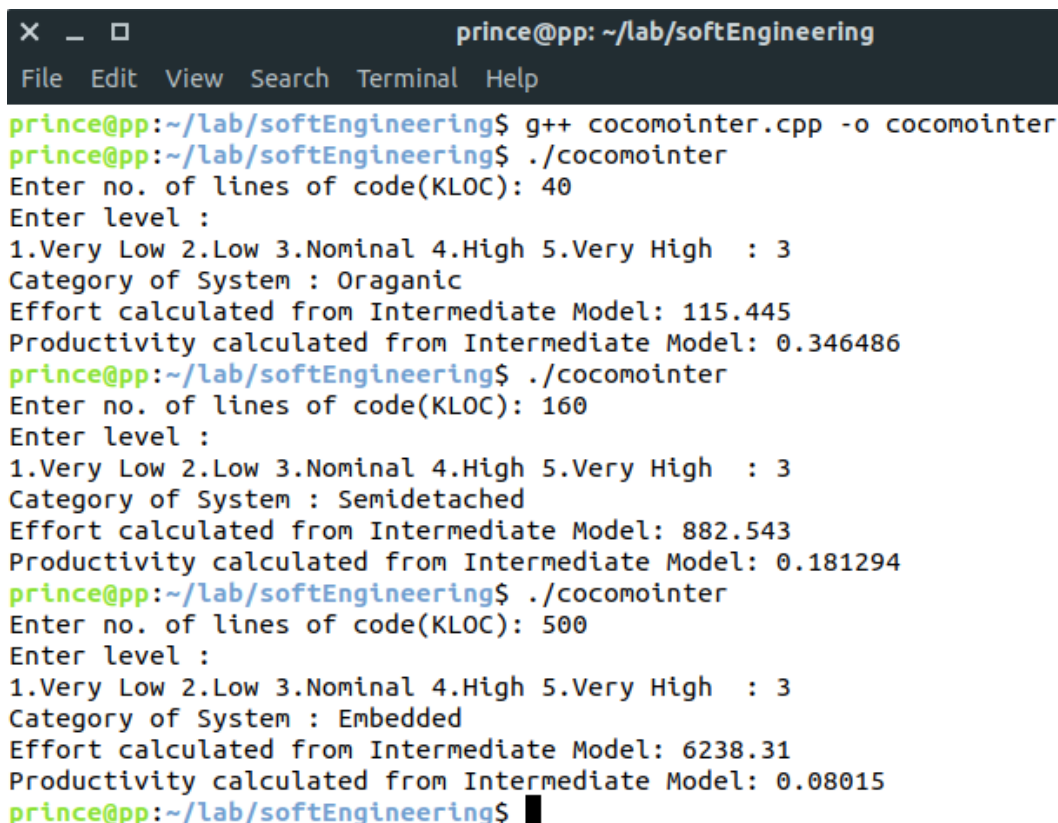
```

    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 analyzed.

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 kilometers, 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 an 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 Inquiries",
                  "internal logical files",
                  "external interface files"};
    string c[] = {"low", "average", "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];
        }
    }
}
```

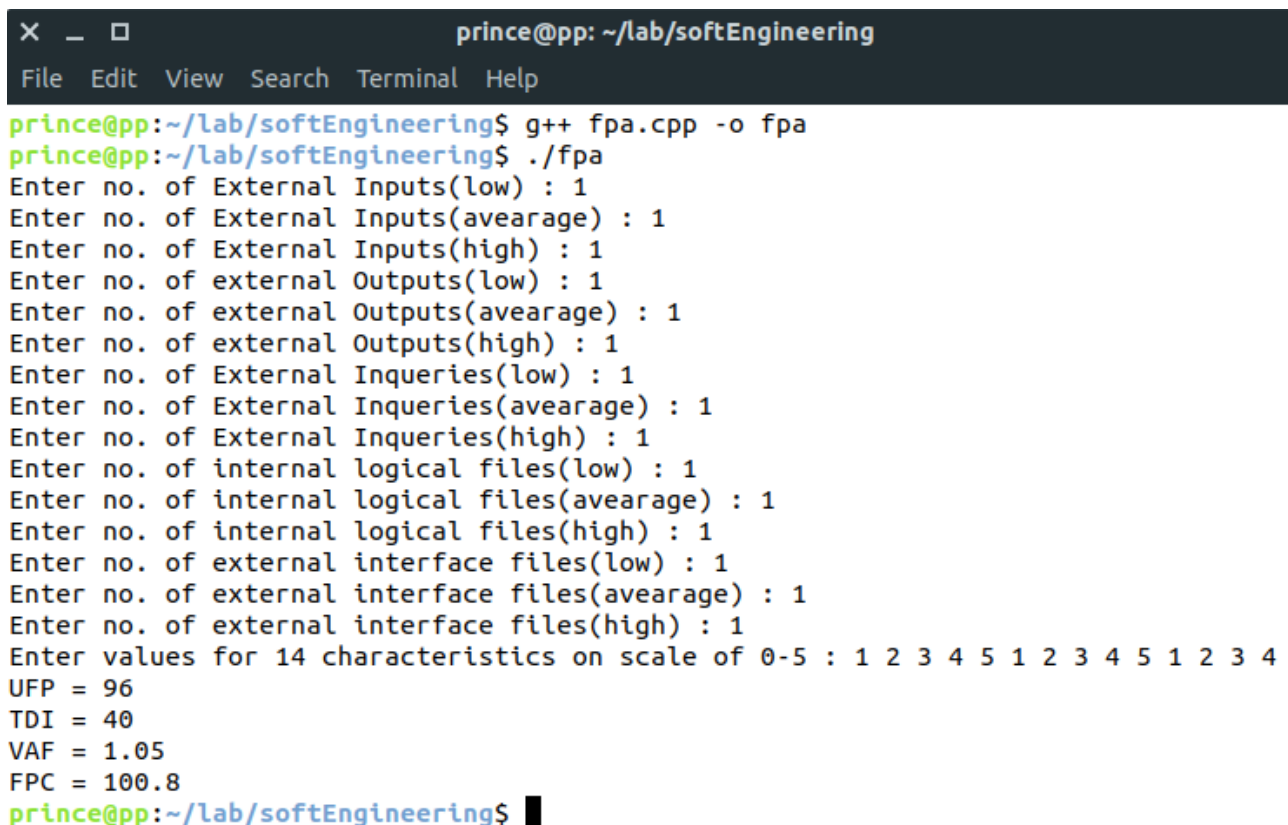
```

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 Inquiries(low) : 1
Enter no. of External Inquiries(avearage) : 1
Enter no. of External Inquiries(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.