SOFTWARE PROJECT MANAGEMENT LAB - G2
EXPERIMENT 1

- ASHISH KUMAR
- 2K18/SE/041

AIM:- Write a program to implement function point analysis.

THEORY:-

Function Point Analysis (FPA) is a method or set of rules of Functional Size Measurement. It assesses the functionality delivered to its users, based on the user's external view of the functional requirements. It measures the logical view of an application not the physically implemented view or technical view.

Components of Function Point Analysis

Based on the interaction of the system components internally and with external users, applications, etc they are categorized into five types:

- **External Inputs (EI):** This is the process of capturing inputs from users like control information or business information and store it as internal/external logic database files.
- **External Outputs (EO):** This is the process of sending out data to external users or systems. The data might be directly grabbed from database files or might undergo some system-level processing.
- **Inquiries (EQ):** This process includes both input and output components. The data is then processed to extract relevant information from internal/external database files.
- **Internal Logic File (ILF):** This is the set of data present within the system. The majority of the data will be interrelated and are captured via the inputs received from the external sources.
- **External Logic File (ELF):** This is the set of data from external resources or external applications. The majority of the data from the external resource is used by the system for reference purpose.

CODE:-

```
#include <bits/stdc++.h>
#include<iostream>
using namespace std;

// Function to calculate Function Point
void calfp(int frates[][3], int fac_rate)
{
 // Function Units
 string funUnits[5] = {
         "External Inputs",
         "External Outputs",
         "External Inquiries",
         "Internal Logical Files",
         "External Interface Files"
 };

 // Weight Rates
 string wtRates[3] = { "Low", "Average", "High" };

 // Weight Factors
 int wtFactors[5][3] = {
         { 3, 4, 6 },
         { 4, 5, 7 },
         { 3, 4, 6 },
         { 7, 10, 15 },
         { 5, 7, 10 },
 };


int UFP = 0;

 // Calculating UFP (Unadjusted Function Point)
 for (int i = 0; i < 5; i++) {

         for (int j = 0; j < 3; j++) {

                 int freq = frates[i][j];

                 UFP += freq * wtFactors[i][j];
         }
 }

 // 14 factors
 string aspects[14] = {
         "reliable backup and recovery required ?",
         "data communication required ?",
         "are there distributed processing functions ?",
         "is performance critical ?",
         "will the system run in an existing heavily utilized operational environment ?",
         "on line data entry required ?",
         "does the on line data entry require the input transaction to be built over multiple
           screens or operations ?",
         "are the master files updated on line ?",
            "is the inputs, outputs, files or inquiries complex ?",
            "is the internal processing complex ?",
            "is the code designed to be reusable ?",
            "are the conversion and installation included in the design ?",
            "is the system designed for multiple installations in different organizations ?",
            "is the application designed to facilitate change and ease of use by the user ?"
        };

        /*
        Rate Scale of Factors
        Rate the following aspects on a scale of 0-5 :-
        0 - No influence
        1 - Incidental
        2 - Moderate
        3 - Average
        4 - Significant
        5 - Essential
        */

        int sumF = 0;

        // Taking Input of factors rate
        for (int i = 0; i < 14; i++) {

            int rate = fac_rate;

            sumF += rate;
        }

        // Calculate CFP
        double CAF = 0.65 + 0.01 * sumF;

        // Calculate Function Point (FP)
        double FP = UFP * CAF;

        // Output Values
        cout << "Function Point Analysis :-" << endl;

        cout << "Unadjusted Function Points (UFP) : " << UFP << endl;

        cout << "Complexity Adjustment Factor (CAF) : " << CAF << endl;

        cout << "Function Points (FP) : " << FP << endl;
}
```

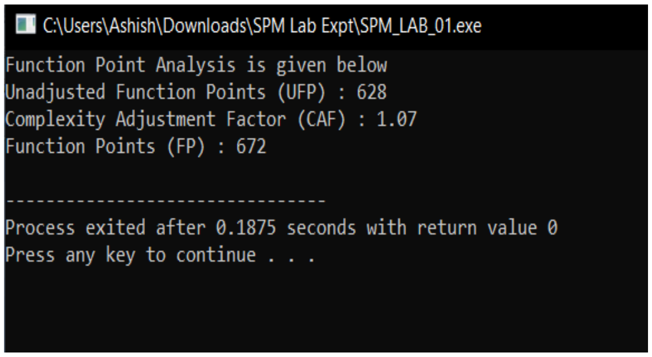
```
int main()
{
        int frates[5][3] = {
                { 0, 50, 0 },
                { 0, 40, 0 },
                { 0, 35, 0 },
                { 0, 6, 0 },
                { 0, 4, 0 }
        };

        int fac_rate = 3;

        calfp(frates, fac_rate);

        return 0;
}
```

OUTPUT:-



```
C:\Users\Ashish\Downloads\SPM Lab Expt\SPM_LAB_01.exe
Function Point Analysis is given below
Unadjusted Function Points (UFP) : 628
Complexity Adjustment Factor (CAF) : 1.07
Function Points (FP) : 672

--------------------------------
Process exited after 0.1875 seconds with return value 0
Press any key to continue . . .
```

CONCLUSION:- FPA method is suitable for estimating the size, cost, productivity, development time, documentation and quality for a project.

Findings and Learning:-

Function points are useful-

- Independent of technology
- In estimating overall costs, schedule and effort.
- In estimating testing projects.