DEPARTMENT OF COMPUTER ENGINEERING

LAB MANUAL
Academic year 2022-23

T.E. COMPUTER (SEM – I)

Laboratory Practice-I (310248)

TEACHING SCHEME
PRACTICAL: 4 HRS/WEEK

EXAMINATION SCHEME
TERM WORK: 25 MARKS
ORAL EXAM: 25 MARKS

Program Outcomes: POs

Students are expected to know and be able –

PO1- *Engineering Knowledge*: - To apply knowledge of mathematics, science, engineering fundamentals, problem solving skills, algorithmic analysis and mathematical modelling to the solution of complex engineering problems.

PO2- *Problem Analysis*: - Identify, formulate, review research literature and analyze complex engineering problems reaching substantiated conclusion using first principals of mathematics, natural sciences and engineering sciences.

PO3- *Design / Development of solutions*: - Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate considerations for the public health and safety, and the cultural, social and environmental considerations.

PO4- *Conduct Investigations of Complex Problems*: - Use research based knowledge and research methods including design of experiments, analysis and interpretation of data, and modeling to complex engineering activities with an understanding of the limitations.

PO5- *Modern Tool Usage*: - Create, select and apply appropriate techniques, resources and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6- *the Engineer and Society*: - Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7- *Environment and Sustainability*:- Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8- *Ethics*:- Apply ethical principles and commit to professional ethics and responsibilities and norms of engineering practice.

PO9- *Individual and Team work* :-Function effectively as an individual and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10- *Communication Skill*:- Communicate effectively on complex engineering activities with the engineering community and with society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive dear instructions.

PO11- *Project management Finance*: - Demonstrate knowledge and understanding of engineering and management principles and apply these to one's own work as a member and leader in a team to manage projects and in multidisciplinary environment.

PO12- *Life-long Learning*: - Recognize the need for, and have the preparations and ability to engage in independent and lifelong learning in the broadest context of technological change.

Program Specific Outcomes: PSOs

A graduate of the Computer Engineering Program will demonstrate-

PSO1- *Professional Skills*-The ability to understand, analyze and develop computer programs in the areas related to algorithms, system software, multimedia, web design, big data analytics, and networking for efficient design of computer-based systems of varying.

PSO2- *Problem-Solving Skills*- The ability to apply standard practices and strategies in software project development using open-ended programming environments to deliver a quality product for business success.

PSO3- *Successful Career and Entrepreneurship*- The ability to employ modern computer languages, environments, and platforms in creating innovative career paths to be an entrepreneur, and a zest for higher studies.

Course Objectives:

1) To learn system programming tools
2) To learn modern operating system
3) To learn various techniques, tools, application in IOT and embedded system. HCI/ DS/ SPM.

Course Outcomes:

On completion of the course, learners will be able to

System programming and operating system

CO1: Implement language translators

CO2: use tools like LEX and YACC

CO3: Implement internals and functionalities of operating system.

A- Internet of Things and Embedded system

CO4: Design IoT and Embedded system based application

CO5: Develop smart application using IoT

CO6: Develop IoT application based on cloud environment

DEPARTMENT OF COMPUTER ENGINEERING

CERTIFICATE

This is to certify that, Mr. /Miss_____of class TE Roll No._____Exam Seat No._____

has completed all the practical work in the subject *Laboratory Practices-I*, satisfactorily, as prescribed by Savitribai Phule Pune University, Pune (SPPU) in the Academic Year 2022-23.

Subject In-charge **Head of Department**

Guidelines for Laboratory /Term Work Assessment

Continuous assessment of laboratory work is based on overall performance and Laboratory assignments performance of student. Each Laboratory assignment assessment will assign grade/marks based on parameters with appropriate weightage. Suggested parameters for overall assessment as well as each Laboratory assignment assessment include- timely completion, performance, innovation, efficient codes, punctuality and neatness

Guidelines for Laboratory Conduction

The instructor is expected to frame the assignments by understanding the prerequisites, technological aspects, utility and recent trends related to the topic. The assignment framing policy need to address the average students and inclusive of an element to attract and promote the intelligent students. The instructor may set multiple sets of assignments and distribute among batches of students. It is appreciated if the assignments are based on real world problems/applications. Use of open source software is encouraged. In addition to these, instructor may assign one real life application in the form of a mini-project based on the concepts learned. Instructor may also set one assignment or mini-project that is suitable to respective branch beyond the scope of syllabus.

Programming Tools:
Internet of things and embedded system: - Raspberry Pi / Arduino Programming; Arduino IDE / Python.

Guidelines for Oral Examination

. During oral assessment, the expert evaluator should give the maximum weightage to the satisfactory implementation of the problem statement. The supplementary and relevant questions may be asked at the time of evaluation to test the student's for advanced learning, understanding of the fundamentals, effective and efficient implementation. So encouraging efforts, transparent evaluation and fair approach of the evaluator will not create any uncertainty or doubt in the minds of the students. So adhering to these principles will consummate our team efforts to the promising start of the student's academics.

INDEX

Sr. no.	Assignment	Date of performance	Date of Submission	Marks Obtained	Sign of Faculty
1	*Design suitable Data structures and implement Pass-I and Pass-II of a two-pass assembler for pseudo-machine. Implementation should consist of a few instructions from each category and few assembler directives.*				
2	*Design suitable data structures and implement Pass-I and Pass-II of a two-pass macro- processor.*				
3	*Write a program to simulate CPU Scheduling Algorithms: FCFS, SJF (Preemptive), Priority (Non-Preemptive) and Round Robin (Preemptive).*				
4	*Write a program to simulate Memory placement strategies – best fit, first fit, next fit and worst fit.*				
5	*Design a user interface in Python*				
6	*To Redesign existing Graphical User Interface with screen complexity*				
			Total Marks		

Total marks: - /

Date: **Subject Teacher Sign**

Assignment-1

Aim: Design suitable data structures and implement pass-I of a two-pass assembler for pseudo-machine. Implementation should consist of a few instructions from each category and few assembler directives.

Objective :

1) To understand data structures to be used in pass I of an assembler.
2) To Implement Two Pass Assembler in C++

Theory:

A language translator bridges an execution gap to machine language of computer system. An assembler is a language translator whose source language is assembly language. An Assembler is a program that accepts as input an Assembly language program and converts it into machine language.

Two Pass Assembler Scheme

In a 2-pass assembler, the first pass constructs an intermediate representation of the source program for use by the second pass. This representation consists of two main components - data structures like Symbol table, Literal table and processed form of the source program called as intermediate code(IC). This intermediate code is represented by the syntax of Variant –I.

Forward reference of a program entity is a reference to the entity, which precedes its definition in the program. While processing a statement containing a forward reference, language processor does not possess all relevant information concerning referenced entity. This creates difficulties in synthesizing the equivalent target statements. This problem can be solved by postponing the generation of target code until more information concerning the entity is available. This also reduces memory requirements of LP and simplifies its organization. This leads to multi-pass model of language processing.

Data Structure for Two Pass Assembler Scheme

Data Structure of Assembler:

a) Operation code table (OPTAB) :This is used for storing mnemonic,
operation code and class of instruction

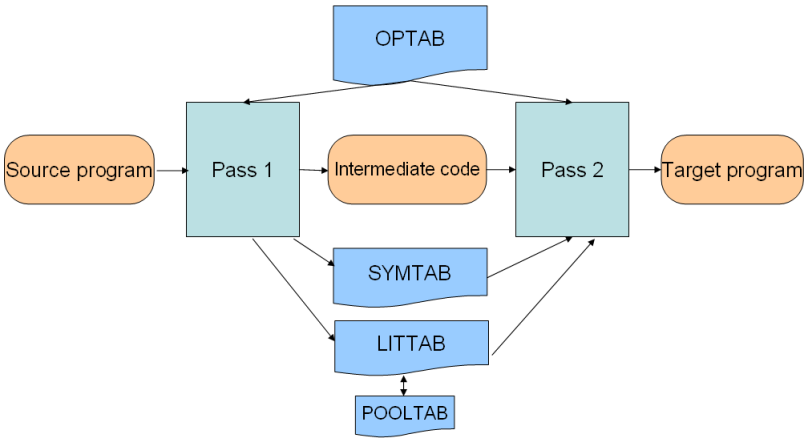
Structure of OPTAB is as follows

b) Data structure updated during translation: Also called as translation time data structure. They are

I. SYMBOL TABLE (SYMTAB) : Ii contains entries such as symbol, it's address and value.

II. LITERAL TABLE (LITTAB) : it contains entries such as literal and it's value.

III . POOL TABLE (POOLTAB): Contains literal number of the starting literal of each literal pool.



Design of Two Pass Assembler Scheme

Tasks performed by the passes of two-pass assembler are as follows:

Pass I: -

Separate the symbol, mnemonic opcode and operand fields.

Determine the storage-required for every assembly language statement and update the location counter.

Build the symbol table and the literal table.

Construct the intermediate code for every assembly language statement.

Pass II: -

Synthesize the target code by processing the intermediate code generated during pass1

INTERMEDIATE CODE REPRESENTATION

The intermediate code consists of a set of IC units, each IC unit consisting of the following three fields

1. Address
2. Representation of the mnemonic opcode
3. Representation of operands

Where statement class can be one of IS,DL and AD standing for imperative statement,

declaration statement and assembler directive respectively.

Algorithms(procedure) :

PASS 1

• Initialize location counter, entries of all tables as zero.

• Read statements from input file one by one.

• While next statement is not END statement

I. Tokenize or separate out input statement as label,numonic,operand1,operand2

II. If label is present insert label into symbol table.

III. If the statement is LTORG statement processes it by making it's entry into literal table, pool table and allocate memory.

IV. If statement is START or ORIGEN Process location counter accordingly.

V. If an EQU statement, assign value to symbol by correcting entry in symbol table.

VI. For declarative statement update code, size and location counter.

VII. Generate intermediate code.

VIII. Pass this intermediate code to pass -2.

Conclusions:

Program:

```
/*
* Program 1: Implementation of 2 pass assemblers for x86 machine.
* The Text Editor: VIM - Vi Improved 9.0
* The Compiler: Apple clang version 13.1.6 (clang-1316.0.21.2.3)
* The Machine: MacBook Air M1 (Running ARM64 architecture)
*/
#include <iostream>#
  include <cstdio >#
  include <cstring>

int main() {
const char* code[9][4] = {
  {"PRG1", "START", "",""},
  {"", "USING", "*", "15"},
  {"", "L", "", ""},
  {"", "A", "", ""},
  {"", "ST", "", ""},
  {"FOUR", "DC", "F", ""},
  {"FIVE", "DC", "1F", ""},
  {"TEMP", "DS", "1F", ""},
  {"", "END", "", ""}
  };

    char av[2], avail[15] = {'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N'};
                int i, j, k, count[3], lc[9] = {0, 0, 0, 0, 0, 0, 0, 0, 0}, loc = 0;

                std::cout << "-------------------------------------------------" <<
                                    std::endl;

                    std::cout << "LABEL\t\tOPCODE" << std::endl;
    std::cout << "-------------------------------------------------------------------------------" <<
                                    std::endl;

                                for (i = 0; i <= 8; i++) {
                                  for (j = 0; j <= 3; j++) {
                                std::cout << code[i][j] << "\t\t";
                                    }
                                  j = 0;
                                std::cout << std::endl;
                                    }

                                    std::cin.get();

    std::cout << "-------------------------------------------------------------------------------" ;
          std::cout << std::endl << "VALUES FOR LC:" << std::endl << std::e              ndl;

                                for (j = 0; j <= 8; j++) {
                  if ((strcmp(code[j][1], "START") != 0) && (strcmp(code[j][1], "
              USING") != 0) && (strcmp(code[j][1], "L") != 0))lc[j] = lc[j -
                                    1] + 4;
                                std::cout << lc[j] << "\t";
                                    }

    std::cout << std::endl << std::endl << "SYMBOL TABLE:" << std::endl;std::cout << "---
                        ------------------------------------------" <<
                                    std::endl;
```

```
52                  std::cout << "SYMBOL\t\tVALUE\t\tLENGTH\t\tR/A" << std::endl;
53                      std::cout << "-----------------------------------------------" <<
                                                    std::endl;
54
55      for                             (i = 0; i < 9; i++) {
56                                  if (strcmp(code[i][1], "START") == 0) {
57                                      std::cout << code[i][0] << "\t\t" << loc << "\t\t" << "4" << "\
                                                t\t" << "R" << std::endl;
58                                                  }
59                                    else if (strcmp(code[i][0], "") != 0) {
60                                      std::cout << code[i][0] << "\t\t" << loc << "\t\t" << "4" << "\
                                                t\t" << "R" << std::endl;
61                                                  loc = 4 + loc;
62                                                  }
63                                  else if (strcmp(code[i][1], "USING") == 0) {
64                                                // Emptye if-statement
65                                                  }
66                                                  else {
67                                                  loc = 4 + loc;
68                                                  }
69       }
70                      std::cout << "-----------------------------------------------" <<
                                                    std::endl;
71
72      std:: cout << std:: endl << std:: endl << " BASE TABLE:" << std:: endl; std::cout << "-----
73                                  ----------------------------------------- " <<
                                                    std::endl;
74        std::cout << "REG NO\t\tAVALIABILITY\tCONTENTS OF BASE TABLE" << std::
                                                    endl;
75      std::cout << "-------------------------------------------------" <<
            std::endl;
76
77      for (j = 0; j <= 8; j++) {
78            if (strcmp(code[j][1], "USING") != 0) {
79                  // Empty if-statement
80            }
81            else {
82                  strcpy(av, code[j][3]);
83            }
84      }
85
86      count[0] = (int)av[0] - 48;
87      count[1] = (int)av[1] - 48;
88      count[2] = count[0] * 10 + count[1];
89      avail[count[2] - 1] = 'Y';
90
91      for (k = 0; k < 16; k++) {
92            std::cout << k << "\t\t" << avail[k - 1] << std::endl;
93      }
94
95      std::cout << "-------------------------------------------------" <<
            std::endl;
96      std::cout << "Continue..??" << std::endl;
97      std::cin.get();
98
99      std::cout << "PASS2 TABLE:" << std::endl << std::endl;
100     std::cout << "LABLE\t\tOP1\t\tLC\t\t" << std::endl;
```

```
    std::cout << "-------------------------------------------" <<
        std::endl;

    loc                                         = 0;
    for                                     (i = 0; i <= 8; i++) {
                                          for (j = 0; j <= 3; j++) {
                                         std::cout << code[i][j] << "\t\t";
                                                }
                                              j = 0;

                                          std::cout << std::endl;
                                               std::cin.get();
                                                }
                        std::cout << "-----------------------------------------------" <<
                                                std::endl;
}
```

(commends for linux or MacOS or Unix-Like OS)

Linux OS

```
$ touch main.cpp                    // For creating file
(open that file in any text editor you want and write above code)

$ g++ main.cpp -o main              // For compiling the file into executable

$ ./main                            // For running that output fileMacOSX



$ touch main.cpp                    // For creating file
(open that file in any text editor you want and write above code)

$ clang++ main.cpp -o main          // For compiling the file into executable

$ ./main                            // For running that output file
```

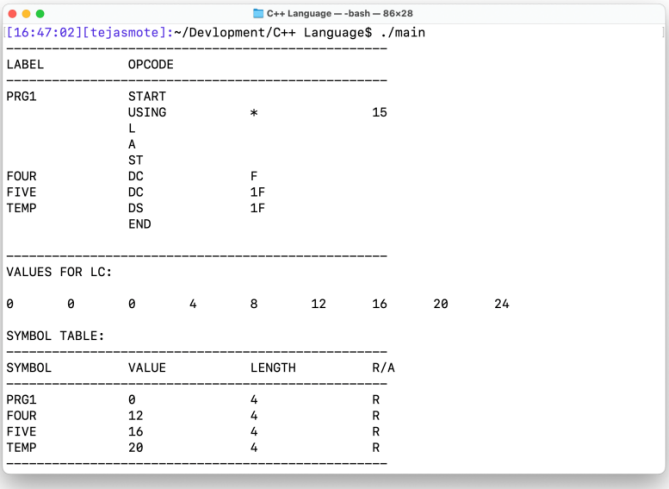


Figure 1: The output terminal snap 1

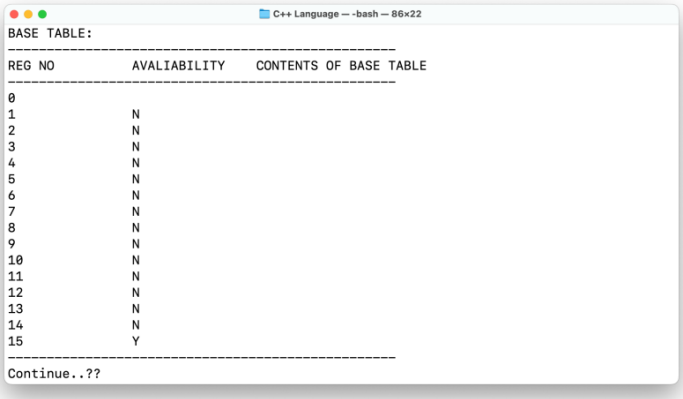


Figure 2: The output terminal snap 2

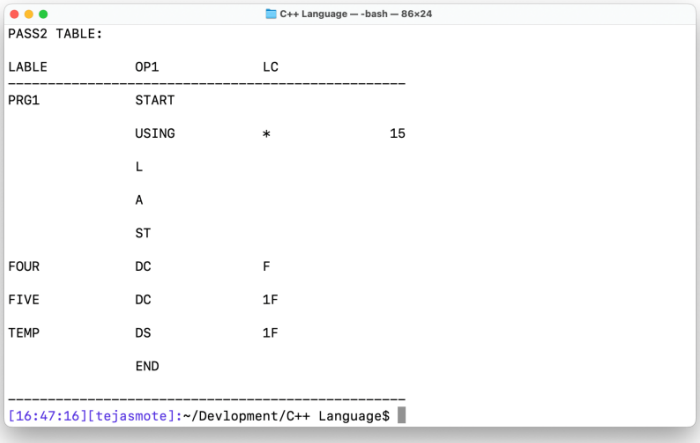


Figure 3: The output terminal snap 3

Aim: Design suitable data structures and implement Pass-I and Pass-II of a two-pass macro-processor.

Objective:

1) To Identify and create the data structures required in the design of macro processor
2) Implement Pass-I and Pass-II of a two-pass macro- processor using C++

Theory :

- In Pass-I the macro definitions are searched and stored in the macro definition table and the entry is made in macro name table
- In Pass-II the macro calls are identified and the arguments are placed in the appropriate place and the macro calls are replaced by macro definitions.

Pass 1:-

• The input macro source program.

• The output macro source program to be used by Pass2.

• Macro-Definition Table (MDT), to store the body of macro definition .

• Macro-Definition Table Counter (MDTC), to mark next available entry MDT.

• Macro- Name Table (MNT) - store names of macros.

• Macro Name Table counter (MNTC) - indicate the next available entry in MNT.

• Argument List Array (ALA) - substitute index markers for dummy arguments before storing a macro-definition.

Pass 2:-

• The input is from Pass1.

• The output is expanded source to be given to assembler.

• MDT and MNT are created by Pass1.

• Macro-Definition Table Pointer (MDTP), used to indicate the next line of text to be used during macro expansion.

• Argument List Array (ALA), used to substitute macro call arguments for the index markers in the stored macro-definitions.

Design of two pass Macro processor

PASS I - ALGORITHM

• Pass1 of macro processor makes a line-by-line scan over its input.

• Set MDTC = 1 as well as MNTC = 1.

• Read next line from input program.

• If it is a MACRO pseudo-op, the entire macro definition except this (MACRO) line is stored in MDT.

• The name is entered into Macro Name Table along with a pointer to the first location of MDT entry of the definition.

• When the END pseudo-op is encountered all the macro-definitions have been processed, so control is transferred to pass2

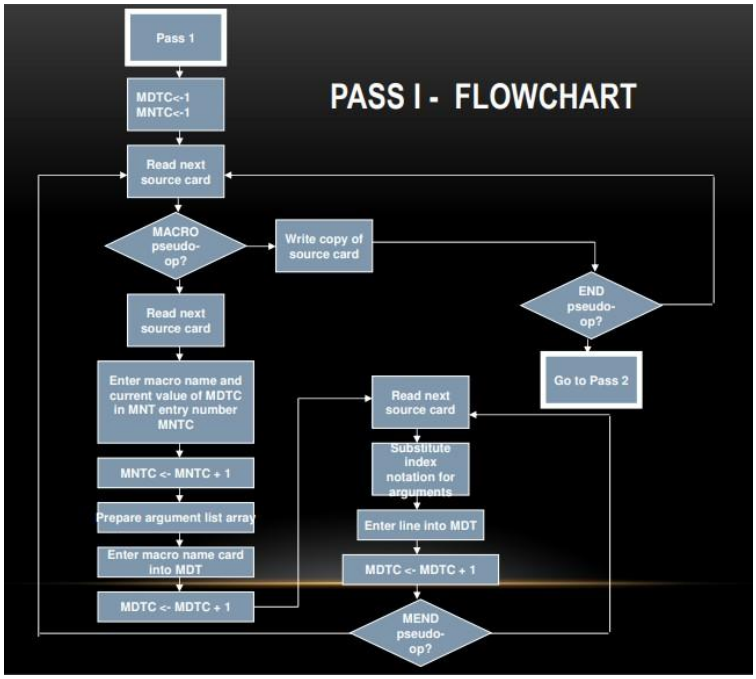


Figure : Pass I processing Macro Definitions

PASS II - ALGORITHM

• This algorithm reads one line of input program at a time.

• for each Line it checks if op-code of that line matches any of the MNT entry.

• When match is found (i.e. when call is pointer called MDTF to corresponding macro definitions stored in MDT.

• The initial value of MDTP is obtained from MDT index field of MNT entry.

•The macro expander prepares the ALA consisting of a table of dummy argument indices & corresponding arguments to the call.

•Reading proceeds from the MDT, as each successive line is read, The values form the argument list one substituted for dummy arguments indices in the macro definition.

• Reading MEND line in MDT terminates expansion of macro & scanning continues from the input file.

• When END pseudo-op encountered , the expanded source program is given to the assembler.

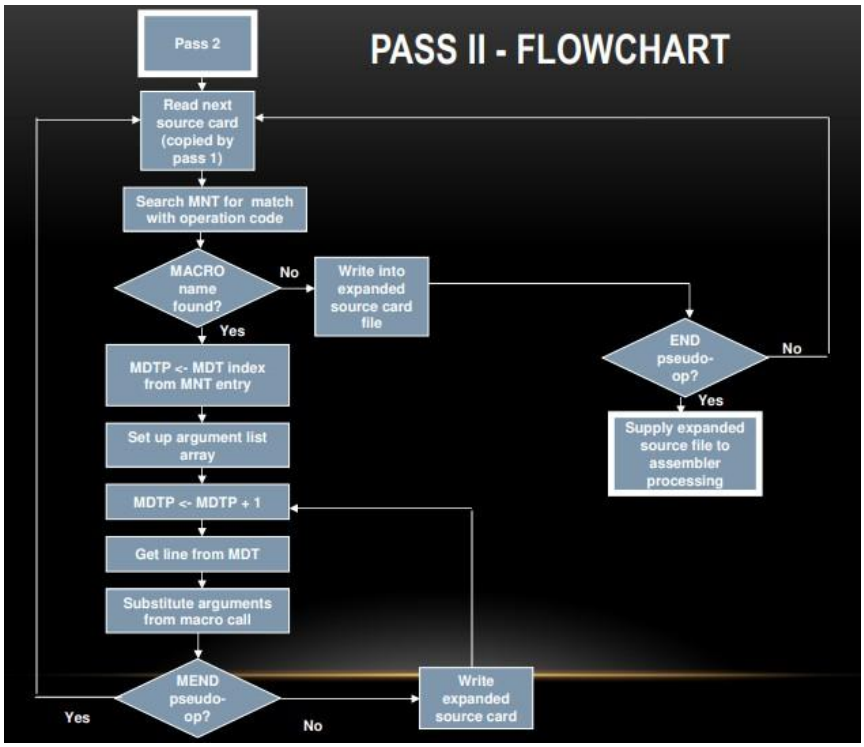


Figure : Pass II processing Macro calls and expansion

Conclusions:

Aim: Write a program to simulate CPU Scheduling Algorithms: FCFS, SJF (Preemptive), Priority (Non- Preemptive) and Round Robin (Preemptive).

Objectives: Implement CPU scheduling algorithms FCFS, SJF (Preemptive), Priority (Non-Preemptive) and Round Robin (Preemptive) using Python.

Theory:

A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms.

- First-Come, First-Served (FCFS) Scheduling
- Shortest-Job-Next (SJN) Scheduling or Shortest-Job-First (SJF) Scheduling
- Priority Scheduling
- Round Robin(RR) Scheduling

These algorithms are either non-preemptive or preemptive. Non-preemptive algorithms are designed so that once a process enters the running state, it cannot be preempted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

1) First-Come, First-Served (FCFS) Scheduling

- Jobs are executed on first come, first serve basis.
- It is a non-preemptive, pre-emptive scheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- Poor in performance as average wait time is high.

Algorithm

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time

Step 4: Set the waiting of the first process as '0' and its burst time as its turnaround time

Step 5: for each process in the Ready Q calculate

(a) Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)

(b) Turnaround time for Process(n)= waiting time of Process(n)+ Burst time for process(n)

Step 6: Calculate

(a) Average waiting time = Total waiting Time / Number of process

(b) Average Turnaround time = Total Turnaround Time / Number of process Step 7: Stop the process

2) Shortest-Job-Next (SJN) Scheduling or Shortest-Job-First (SJF) Scheduling

- This is also known as shortest job first, or SJF
- This is a non-preemptive, pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.
- The processer should know in advance how much time process will take.

Algorithm

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time

Step 4: Start the Ready Q according the shortest Burst time by sorting according to lowest to highest burst time.

Step 5: Set the waiting time of the first process as '0' and its turnaround time as its burst time.

Step 6: For each process in the ready queue, calculate

(c) Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)

(d) Turnaround time for Process(n)= waiting time of Process(n)+ Burst time for process(n)

Step 6: Calculate

(c) Average waiting time = Total waiting Time / Number of process

(d) Average Turnaround time = Total Turnaround Time / Number of process

Step 7: Stop the process

3) Priority Scheduling

- Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.
- Each process is assigned a priority. Process with highest priority is to be executed first and so on.
- Processes with same priority are executed on first come first served basis.

- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

Algorithm

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time, priority

Step 4: Start the Ready Q according the priority by sorting according to lowest to highest burst time and process.

Step 5: Set the waiting time of the first process as '0' and its turnaround time as its burst time.

Step 6: For each process in the ready queue, calculate

(e) Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)

(f) Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)

Step 6: Calculate

(g) Average waiting time = Total waiting Time / Number of process

(h) Average Turnaround time = Total Turnaround Time / Number of process

Step 7: Stop the process

4) Round Robin(RR) Scheduling

- Round Robin is the preemptive process scheduling algorithm.
- Each process is provided a fix time to execute, it is called a quantum.
- Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
- Context switching is used to save states of preempted processes.

Algorithm

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue and time quantum (or) time slice

Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time

Step 4: Calculate the no. of time slices for each process where

No. of time slice for process(n) = burst time process(n)/time slice

Step 5: If the burst time is less than the time slice then the no. of time slices =1.

Step 6: Consider the ready queue is a circular Q, calculate

(a) Waiting time for process(n) = waiting time of process(n-1)+ burst time of process(n-1) + the time difference in getting the CPU from process(n-1)

(b) Turnaround time for process(n) = waiting time of process(n) + burst time of process(n)+ the time difference in getting CPU from process(n).

Step 7: Calculate

(e) Average waiting time = Total waiting Time / Number of process

(f) Average Turnaround time = Total Turnaround Time / Number of process

Step 8: Stop the process.

Conclusions:

1) Python program for implementation of FCFS scheduling

```
# Function to find the waiting time for all processes
def findWaitingTime(processes, n, bt, wt):

    # waiting time for first process is 0
    wt[0] = 0

    # calculating waiting time
    for i in range(1, n ):
        wt[i] = bt[i - 1] + wt[i - 1]

# Function to calculate turnaround time
def findTurnAroundTime(processes, n, bt, wt, tat):

    # calculating turnaround time by adding bt[i] + wt[i]
    for i in range(n):
        tat[i] = bt[i] + wt[i]

# Function to calculate average time
def findavgTime( processes, n, bt):

    wt = [0] * n
    tat = [0] * n
    total_wt = 0
    total_tat = 0

    # Function to find waiting time of all processes
    findWaitingTime(processes, n, bt, wt)

    # Function to find turnaround time for all processes
    findTurnAroundTime(processes, n, bt, wt, tat)

    # Display processes along with all details
    print( "Processes Burst time " +
                  " Waiting time " +
                " Turn around time")

    # Calculate total waiting time And total turnaround time
    for i in range(n):

        total_wt = total_wt + wt[i]
        total_tat = total_tat + tat[i]
        print(" " + str(i + 1) + "\t\t" +
                    str(bt[i]) + "\t " +
```

```
                    str(wt[i]) + "\t\t " +
                    str(tat[i]))

    print( "Average waiting time = "+
                   str(total_wt / n))
    print("Average turn around time = "+
                     str(total_tat / n))

# Driver code
if __name__=="__main__":

    # process id's
    processes = [ 1, 2, 3]
    n = len(processes)

    # Burst time of all processes
    burst_time = [10, 5, 8]

    findavgTime(processes, n, burst_time)
```

Output:

```
Processes Burst time  Waiting time  Turnaround time
 1          10     0          10
 2          5      10         15
 3          8      15         23
Average waiting time = 8.333333333333334
Average turnaround time = 16.0
```

2) Python program to implement Shortest Remaining Time First Shortest Remaining Time First (SRTF)/ Shortest Job First(SJF)/ Shortest Job Next(SJN)

```
# Function to find the waiting time for all processes
def findWaitingTime(processes, n, wt):
    rt = [0] * n

    # Copy the burst time into rt[]
    for i in range(n):
        rt[i] = processes[i][1]
    complete = 0
    t = 0
    minm = 999999999
    short = 0
    check = False

    # Process until all processes gets completed
    while (complete != n):

        # Find process with minimum remaining time among the processes that
        # arrives till the current time`
        for j in range(n):
            if ((processes[j][2] <= t) and
                (rt[j] < minm) and rt[j] > 0):
                minm = rt[j]
                short = j
                check = True
        if (check == False):
            t += 1
            continue

        # Reduce remaining time by one
        rt[short] -= 1

        # Update minimum
        minm = rt[short]
        if (minm == 0):
            minm = 999999999

        # If a process gets completely executed
        if (rt[short] == 0):

            # Increment complete
            complete += 1
            check = False

            # Find finish time of current process
            fint = t + 1
```

```
            # Calculate waiting time
            wt[short] = (fint - proc[short][1] -
                                proc[short][2])

            if (wt[short] < 0):
                wt[short] = 0

        # Increment time
        t += 1

# Function to calculate turnaround time
def findTurnAroundTime(processes, n, wt, tat):

    # Calculating turnaround time
    for i in range(n):
        tat[i] = processes[i][1] + wt[i]

# Function to calculate average waiting and turn-around times.
def findavgTime(processes, n):
    wt = [0] * n
    tat = [0] * n

    # Function to find waiting time of all processes
    findWaitingTime(processes, n, wt)

    # Function to find turn around time for all processes
    findTurnAroundTime(processes, n, wt, tat)

    # Display processes along with all details
    print("Processes    Burst Time     Waiting",
                   "Time     Turn-Around Time")
    total_wt = 0
    total_tat = 0
    for i in range(n):

        total_wt = total_wt + wt[i]
        total_tat = total_tat + tat[i]
        print(" ", processes[i][0], "\t\t",
                   processes[i][1], "\t\t",
                   wt[i], "\t\t", tat[i])

    print("\nAverage waiting time = %.5f "%(total_wt /n) )
    print("Average turn around time = ", total_tat / n)

# Driver code
if __name___=="__main__":

    # Process id's
    proc = [[1, 6, 1], [2, 8, 1],
            [3, 7, 2], [4, 3, 3]]
```

```
    n = 4
    findavgTime(proc, n)
```

Output:

```
Processes    Burst Time     Waiting Time     Turn-Around Time
  1          6           3           9
  2          8           16          24
  3          7           8           15
  4          3           0           3

Average waiting time = 6.75000
Average turnaround time =  12.75
```

3) Python program for implementation of Priority Scheduling

```
# Function to find the waiting time for all processes
def findWaitingTime(processes, n, wt):
    wt[0] = 0

    # calculating waiting time
    for i in range(1, n):
        wt[i] = processes[i - 1][1] + wt[i - 1]

# Function to calculate turnaround time
def findTurnAroundTime(processes, n, wt, tat):

    # Calculating turnaround time by adding bt[i] + wt[i]
    for i in range(n):
        tat[i] = processes[i][1] + wt[i]

# Function to calculate average waiting and turn-around times.
def findavgTime(processes, n):
    wt = [0] * n
    tat = [0] * n

    # Function to find waiting time of all processes
    findWaitingTime(processes, n, wt)

    # Function to find turnaround time for all processes
    findTurnAroundTime(processes, n, wt, tat)

    # Display processes along with all details
    print("\nProcesses    Burst Time    Waiting",
          "Time    Turn-Around Time")
    total_wt = 0
```

```
    total_tat = 0
    for i in range(n):

        total_wt = total_wt + wt[i]
        total_tat = total_tat + tat[i]
        print(" ", processes[i][0], "\t\t",
                   processes[i][1], "\t\t",
                   wt[i], "\t\t", tat[i])

    print("\nAverage waiting time = %.5f "%(total_wt /n))
    print("Average turn around time = ", total_tat / n)

def priorityScheduling(proc, n):

    # Sort processes by priority
    proc = sorted(proc, key = lambda proc:proc[2],
                                   reverse = True);

    print("Order in which processes gets executed")
    for i in proc:
        print(i[0], end = " ")
    findavgTime(proc, n)

# Driver code
if __name__=="__main__":

    #  Process  id's
    proc = [[1, 10, 1],
            [2, 5, 0],
            [3, 8, 1]]
    n = 3
    priorityScheduling(proc, n)
```

Output:

```
Order in which processes gets executed
1 3 2
Processes    Burst Time    Waiting Time    Turn-Around Time
  1          10          0           10
  3          8           10          18
  2          5           18          23

Average waiting time = 9.33333
Average turnaround time = 17.0
```

4) Python program for implementation of RR scheduling

```
# Function to find the waiting time for all processes
def findWaitingTime(processes, n, bt, wt, quantum):
    rem_bt = [0] * n

    # Copy the burst time into rt[]
    for i in range(n):
        rem_bt[i] = bt[i]
    t = 0 # Current time

    # Keep traversing processes in round  robin manner until all of them are
    # not done.
    while(1):
        done = True

        # Traverse all processes one by one repeatedly
        for i in range(n):

            # If burst time of a process is greater
           # than 0 then only need to process further
            if (rem_bt[i] > 0) :
                done = False # There is a pending process

                if (rem_bt[i] > quantum) :

                    # Increase the value of t i.e. shows
                    # how much time a process has been processed
                    t += quantum

                    # Decrease the burst_time of current
                    # process by quantum
                    rem_bt[i] -= quantum

                # If burst time is smaller than or equal
                # to quantum. Last cycle for this process
                else:

                    # Increase the value of t i.e. shows
                    # how much time a process has been processed
                    t = t + rem_bt[i]

                    # Waiting time is current time minus
                    # time used by this process
                    wt[i] = t - bt[i]

                    # As the process gets fully executed
                    # make its remaining burst time = 0
                    rem_bt[i] = 0
```

```
        # If all processes are done
        if (done == True):
            break

# Function to calculate turn around time
def findTurnAroundTime(processes, n, bt, wt, tat):

    # Calculating turnaround time
    for i in range(n):
        tat[i] = bt[i] + wt[i]


# Function to calculate average waiting
# and turn-around times.
def findavgTime(processes, n, bt, quantum):
    wt = [0] * n
    tat = [0] * n

    # Function to find waiting time
    # of all processes
    findWaitingTime(processes, n, bt,
                        wt, quantum)

    # Function to find turn around time
    # for all processes
    findTurnAroundTime(processes, n, bt,
                                wt, tat)

    # Display processes along with all details
    print("Processes    Burst Time     Waiting",
                    "Time    Turn-Around Time")
    total_wt = 0
    total_tat = 0
    for i in range(n):

        total_wt = total_wt + wt[i]
        total_tat = total_tat + tat[i]
        print(" ", i + 1, "\t\t", bt[i],
              "\t\t", wt[i], "\t\t", tat[i])

    print("\nAverage waiting time = %.5f "%(total_wt /n) )
    print("Average turn around time = %.5f "% (total_tat / n))

# Driver code
if __name__=="__main__":

    # Process id's
    proc = [1, 2, 3]
    n = 3
```

```
    # Burst time of all processes
    burst_time = [10, 5, 8]

    # Time quantum
    quantum = 2;
    findavgTime(proc, n, burst_time, quantum)
```

Output:

```
Processes    Burst Time     Waiting Time    Turn-Around Time
  1          10          13          23
  2          5           10          15
  3          8           13          21

Average waiting time = 12.00000
Average turnaround time = 19.66667
```

Aim: Write a program to simulate Memory placement strategies – best fit, first fit, next fit and Worst fit

Objective : Implement Memory Placement Strategies best fit, first fit, next fit and Worst fit using Python

Theory :

There are various placement strategies which are implemented by the Operating System in order to find out the holes in the linked list and allocate them to the processes.

1. First Fit

First Fit scans the linked list and whenever it finds the first big enough hole to store a process, it stops scanning and load the process into that hole. This procedure produces two partitions. Out of them, one partition will be a hole while the other partition will store the process. First Fit maintains the linked list according to the increasing order of starting index. This is the simplest to implement among all the algorithms and produces bigger holes as compare to the other algorithms.

2.Next Fit

Next Fit algorithm is similar to First Fit algorithm except the fact that, Next fit scans the linked list from the node where it previously allocated a hole.

Next fit doesn't scan the whole list, it starts scanning the list from the next node. The idea behind the next fit is the fact that the list has been scanned once therefore the probability of finding the hole is larger in the remaining part of the list.

Experiments over the algorithm have shown that the next fit is not better then the first fit. So it is not being used these days in most of the cases.

3. Best Fit

The Best Fit algorithm tries to find out the smallest hole possible in the list that can accommodate the size requirement of the process.

Using Best Fit has some disadvantages.

- It is slower because it scans the entire list every time and tries to find out the smallest hole which can satisfy the requirement the process.
- Due to the fact that the difference between the whole size and the process size is very small, the holes produced will be as small as it cannot be used to load any process and therefore it remains useless.

- Despite of the fact that the name of the algorithm is best fit, It is not the best algorithm among all.

4. Worst Fit

The worst fit algorithm scans the entire list every time and tries to find out the biggest hole in the list which can fulfill the requirement of the process.

Despite of the fact that this algorithm produces the larger holes to load the other processes, this is not the better approach due to the fact that it is slower because it searches the entire list every time again and again.

The first fit algorithm is the best algorithm among all because

1. It takes lesser time compare to the other algorithms.
2. It produces bigger holes that can be used to load other processes later on.
3. It is easiest to implement.

Conclusions:

1) Python implementation of First-Fit algorithm

```
# Function to allocate memory to
# blocks as per First fit algorithm
def firstFit(blockSize, m, processSize, n):

    # Stores block id of the
    # block allocated to a process
    allocation = [-1] * n

    # Initially no block is assigned to any process

    # pick each process and find suitable blocks
    # according to its size ad assign to it
    for i in range(n):
        for j in range(m):
            if blockSize[j] >= processSize[i]:

                # allocate block j to p[i] process
                allocation[i] = j

                # Reduce available memory in this block.
                blockSize[j] -= processSize[i]

                break

    print(" Process No. Process Size      Block no.")
    for i in range(n):
        print(" ", i + 1, "         ", processSize[i],
                          "         ", end = " ")
        if allocation[i] != -1:
            print(allocation[i] + 1)
        else:
            print("Not Allocated")

# Driver code
if __name__ == '__main__':
    blockSize = [100, 500, 200, 300, 600]
    processSize = [212, 417, 112, 426]
    m = len(blockSize)
    n = len(processSize)

    firstFit(blockSize, m, processSize, n)
```

Output:

```
Process No. Process Size      Block no.
  1           212           2
  2           417           5
  3           112           2
  4           426           Not Allocated
```

2) Python program for next fit memory management algorithm

```
# Function to allocate memory to blocks as per Next fit algorithm
def NextFit(blockSize, m, processSize, n):

    # Stores block id of the block
    # allocated to a process

    # Initially no block is assigned
    # to any process
    allocation = [-1] * n
    j = 0
    t = m-1
    # pick each process and find suitable blocks
    # according to its size ad assign to it
    for i in range(n):

        # Do not start from beginning
        while j < m:
            if blockSize[j] >= processSize[i]:

                # allocate block j to p[i] process
                allocation[i] = j

                # Reduce available memory in this block.
                blockSize[j] -= processSize[i]

                # sets a new end point
                t = (j - 1) % m
                break
            if t == j:
                # sets a new end point
                t = (j - 1) % m
                # breaks the loop after going through all memory block
                break
```

```
            # mod m will help in traversing the
            # blocks from starting block after
            # we reach the end.
            j = (j + 1) % m

    print("Process No. Process Size Block no.")

    for i in range(n):
        print(i + 1, "         ", processSize[i],end = "     ")
        if allocation[i] != -1:
            print(allocation[i] + 1)
        else:
            print("Not Allocated")

# Driver Code
if_name_== '__main_':
    blockSize = [5, 10, 20]
    processSize = [10, 20, 5]
    m = len(blockSize)
    n = len(processSize)
     NextFit(blockSize, m, processSize, n)
```

Output :

```
Process No. Process Size Block no.
      1         10        2
      2         20        3
      3         5         1
```

3) Python implementation of worst - Fit algorithm

```
# Function to allocate memory to blocks as per worst fit algorithm

def worstFit(blockSize, m, processSize, n):

    # Stores block id of the block
    # allocated to a process

    # Initially no block is assigned
    # to any process
    allocation = [-1] * n

    # pick each process and find suitable blocks
    # according to its size ad assign to it
    for i in range(n):
```

```
        # Find the best fit block for
        # current process
        wstIdx = -1
        for j in range(m):
            if blockSize[j] >= processSize[i]:
                if wstIdx == -1:
                    wstIdx = j
                elif blockSize[wstIdx] < blockSize[j]:
                    wstIdx = j

        # If we could find a block for
        # current process
        if wstIdx != -1:

            # allocate block j to p[i] process
            allocation[i] = wstIdx

            # Reduce available memory in this block.
            blockSize[wstIdx] -= processSize[i]

    print("Process No. Process Size Block no.")
    for i in range(n):
        print(i + 1, "         ",
              processSize[i], end = "     ")
        if allocation[i] != -1:
            print(allocation[i] + 1)
        else:
            print("Not Allocated")

# Driver code
if __name__ == '__main__':
    blockSize = [100, 500, 200, 300, 600]
    processSize = [212, 417, 112, 426]
    m = len(blockSize)
    n = len(processSize)

    worstFit(blockSize, m, processSize, n)
```

Output:

```
Process No. Process Size Block no.
1           212     5
2           417     2
3           112     5
4           426     Not Allocated
```

4) implementation of Best- Fit algorithm using C

```
#include <iostream>
#include <memory>
#include <cstring>
using namespace std;
// To allocate the memory to blocks as per Best fit algorithm
void bestfit(int bsize[], int m, int psize[], int n) {
   // To store block id of the block allocated to a process
   int alloc[n];
   // Initially no block is assigned to any process
    memset(alloc, -1, sizeof(alloc));
   // pick each process and find suitable blocks
   // according to its size ad assign to it
   for (int i=0; i<n; i++) {
      // Find the best fit block for current process
      int bestIdx = -1;
      for (int j=0; j<m; j++) {
         if (bsize[j] >= psize[i]) {
            if (bestIdx == -1)
               bestIdx = j;
            else if (bsize[bestIdx] > bsize[j])
               bestIdx = j;
         }
      }
      // If we could find a block for current process
      if (bestIdx != -1) {
         // allocate block j to p[i] process
         alloc[i] = bestIdx;
         // Reduce available memory in this block.
         bsize[bestIdx] -= psize[i];
      }
   }
   cout << "\nProcess No.\tProcess Size\tBlock no.\n";
   for (int i = 0; i < n; i++) {
      cout << " " << i+1 << "\t\t\t\t" << psize[i] << "\t\t\t\t";
      if (alloc[i] != -1)
```

```
            cout << alloc[i] + 1;
         else
            cout << "Not Allocated";
            cout << endl;
      }
}
// Driver code
int main() {
   int bsize[] = {100, 500, 200, 300, 400};
   int psize[] = {112, 518, 110, 526};
   int m = sizeof(bsize)/sizeof(bsize[0]);
   int n = sizeof(psize)/sizeof(psize[0]);
   bestfit(bsize, m, psize, n);
   return 0 ;
}
```

Output:

```
Process No. Process Size     Block no.
 1          112              3
 2          518              Not Allocated
 3          110              4
 4          526              Not Allocated
```

Aim- Design a user interface in Python

Objective:-
To design a user interface in Python

To learn simplicity, user centric approach of a GUI in designing

Learning Outcomes:

A simple GUI designed using Tkinter library in Python**.**

Requirements:

Tkinter - standard GUI library for Python

Implementation Steps:

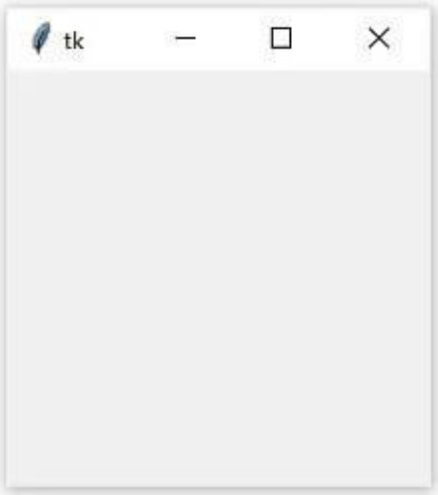
import the Python GUI Tkinter module:

```
>>> import tkinter as tk
```

A **window** is an instance of Tkinter's Tk class. Go ahead and create a new window and assign into the variable window:

```
>>> window = tk.Tk()
```

When you execute the above code, a new window pops up on your screen. How it looks dependson your operating system:



(a) Windows

Adding a Widget

Use the tk.Label class to add some text to a window.
Create a Label widget with the text "Hello,Tkinter" and assign it to a variable called greeting:

```
>>>
>>> greeting = tk.Label(text="Hello, Tkinter")
```

Working With Widgets

Each **widget** in Tkinter is defined by a class. Here are some of the widgets available:

Widget Class		Description
Label	A	widget used to display text on the screen
Button	A	button that can contain text and can perform an action when clicked
Entry	A	text entry widget that allows only a single line of text
Text	A	text entry widget that allows multiline text entry
Frame	A	rectangular region used to group related widgets or provide padding between windows and Images With Label Widgets

Label widgets are used to **display text or images**. The text displayed by a Label widget can't beedited by the user. It's for display purposes only. As you saw in the example at the beginning of this tutorial, you can create a Label widget by instantiating the Label class and passing a string tothe text parameter:

```
label = tk.Label(text="Hello, Tkinter")
```

abel widgets display text with the default system text color and the default system text background color. These are typically black and white, respectively, but you may see differentcolors if you have changed these settings in your operating system.

You can control Label text and background colors
usingthe foreground and background parameters:

```
label = tk.Label(
    text="Hello,
    Tkinter",
    foreground="white", # Set the text color to white
    background="black" # Set the background color to
    black
)
```

ou can also control the width and height of a label with the width and height parameters:

```
label = tk.Label(
    text="Hello,
    Tkinter",
    fg="white",
    bg="black",
    width=10,
    height=10
```

Here's what this label looks like in a window:

Displaying Clickable Buttons With Button

```
Widgetsbutton = tk.Button(
    text="Click
    me!",
    width=25,
    height=5,
    bg="blue",
    fg="yellow",
)
```

Getting User Input With Entry Widgets

he following code creates a widget with a blue background, some yellow text, and a widthof 50 text units:

```
entry = tk.Entry(fg="yellow", bg="blue", width=50)
```

The best way to get an understanding of Entry widgets is to create one and interact with it. Openup a Python shell and follow along with the examples in this section. First, import tkinter and create a new window:

```
>>> import tkinter as tk
>>> window = tk.Tk()
```

Now create a Label and an Entry widget:

```
>>> label = tk.Label(text="Name")
>>> entry = tk.Entry()
```

The Label describes what sort of text should go in the Entry widget. It doesn't enforce any sort ofrequirements on the Entry, but it tells the user what your program expects them to put there. You need to .pack() the widgets into the window so that they're visible:

```
>>> label.pack()
>>> entry.pack()
```

Here's what that looks like:

Conclusion: GUI created using Python

.....

Conclusion-

……

Program- **Attach Program with Comment.**

Output- **Attach Output**

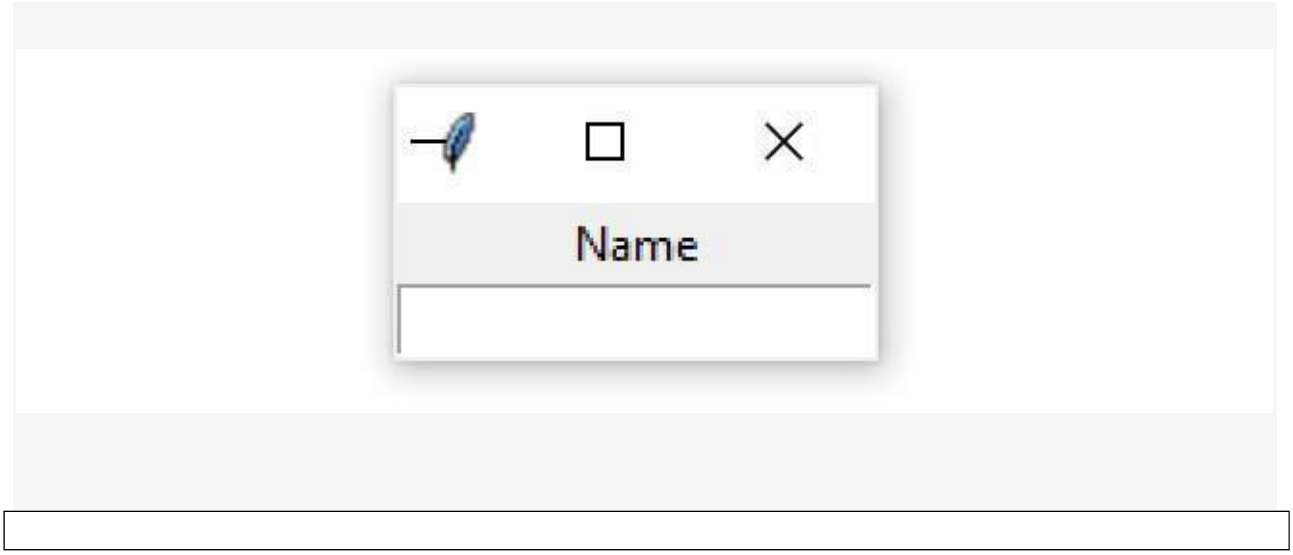
Aim:- To Redesign existing Graphical User Interface with screen complexity

Objective:- To study principles of Good screen design
To apply the screen complexity rules to a GUI to improvise it.
To analyse the human considerations in Interface and screen design.

Learning Outcomes:

- ✓ Design better screens in interfaces based on visually pleasing structure
- ✓ Learn to organize the elements on an interface screen by properly calculating the screencomplexity.

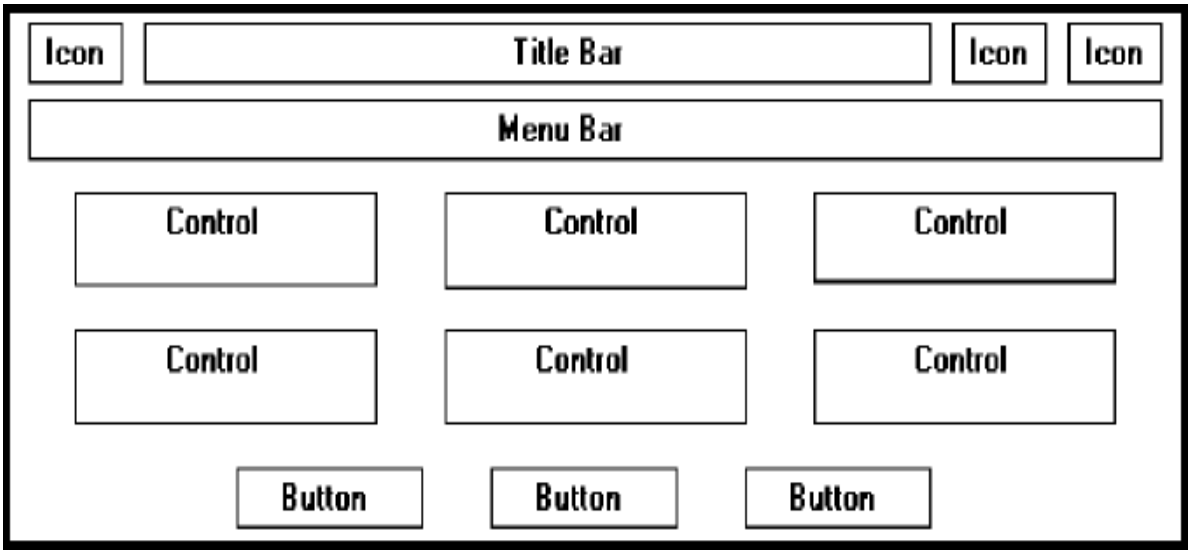
Learning the factors that affect the screen design quality with respect to user expectations



Requirements:

Any GUI screen from a selected application.

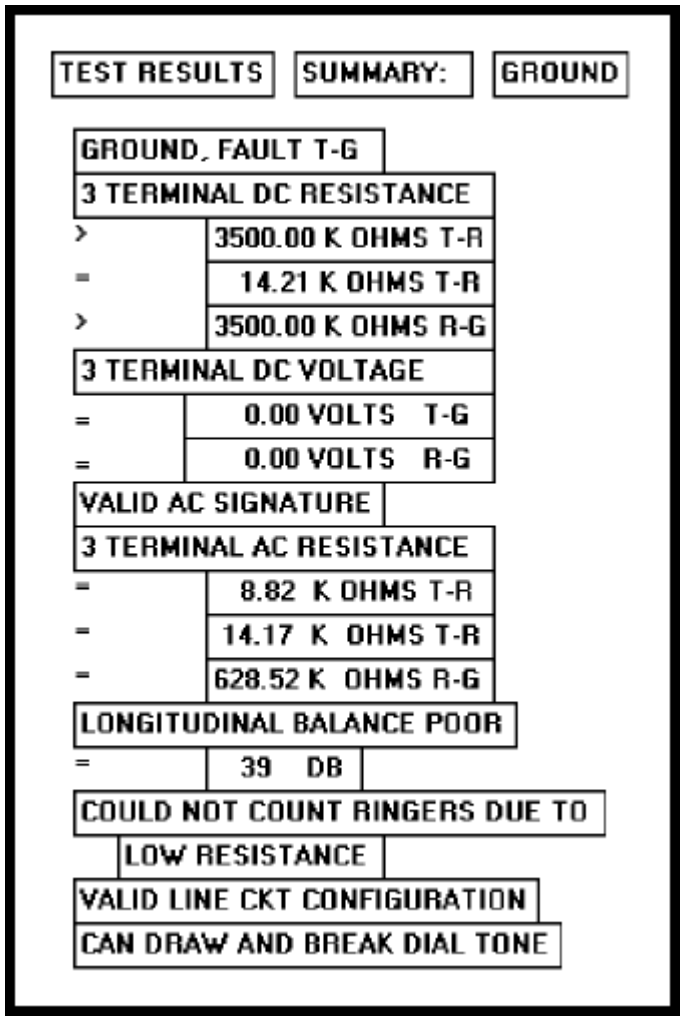
General structure of the elements on the screen to measure complexity factors.



To calculate the complexity first determine the following:

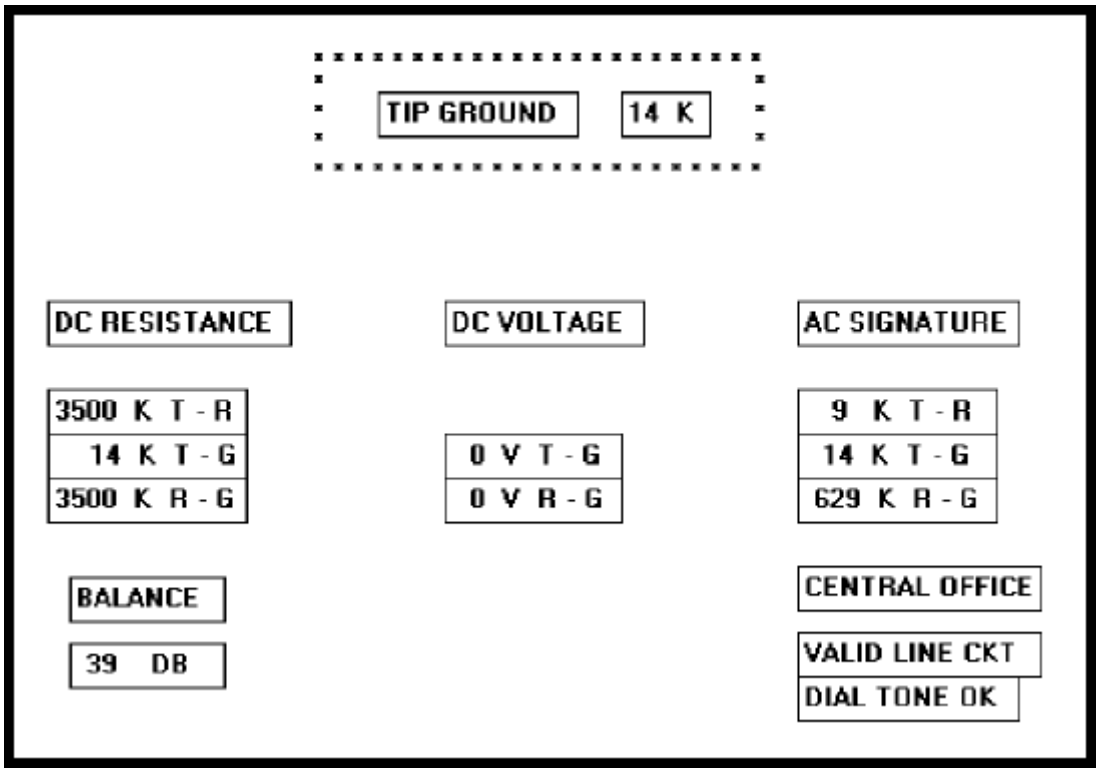
1. the number of elements on the screen
2. the number of horizontal (column) alignment points
3. the number of vertical (row) alignment points

An example is given below:



Original Design of the GUI

- ✓ In the above screen the elements are not placed in a proper symmetry, which creates userconfusion and loss of interest in the interface.
- ✓ The first requirement is to identify the text boxes and their places on the screen and thenplace them in a proper order , also group them as per requirement.
- ✓ The re-designed screen for the above example is shown in the figure below.
  To validate the improved screen, complexity of the screen is calculated which shows The optimization of screen space as well as the user friendly interface.



Re-designed Screen design

Calculation of complexity:

1. Original Design
2. elements
3. horizontal (column) alignment points
4. vertical (row) alignment points

1. 48 = complexity (addition of all counts))

Redesigned Screen

1. 18 elements
2. 7 horizontal (column) alignment points
3. 8 vertical (row) alignment points
4. 33 = complexity

Students have to choose an application like the above GUI and calculate the screen complexityand re-design it by re-arranging the elements on the screen

Conclusion:

Brief description of the studied method for improving screen design complexity and the improvement in design by applying this method.