

**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**Belgaum, Karnataka-590 014**



**Laboratory Manual**

**System Software Laboratory**

**(18CSL66)**

**Compiled by**

**1. Prof. Prakash A**

**2. Prof. Vijayalakshmi S A**

**3. Prof. Rajeev Bilagi**

**4. Prof. Rakshitha B T**



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**(ACCREDITED BY NBA)**

**ACHARYA INSTITUTE OF TECHNOLOGY**

**Soldevanahalli, Bengaluru-560107**

**2022-2023**

## Table of contents

|                                       |     |
|---------------------------------------|-----|
| Vision, Mission, Motto of Institute   | I   |
| Vision, Mission of Department         | II  |
| Program Educational Objectives (PEOs) | III |
| Program Specific Outcomes (PSOs)      | III |
| Program outcomes (POs)                | IV  |
| Course outcomes of course (COs)       | VI  |

| SL | Name of Program   | Page No |
|----|---|---------|
| 1  | a) Write a LEX program to recognize valid arithmetic expression. Identifiers in the expression could be only integers and operators could be + and *. Count the identifiers & operators present and print them separately.<br>b) Write YACC program to evaluate arithmetic expression involving operators: +, -, *, and / |         |
| 2  | Develop, Implement and Execute a program using YACC tool to recognize all strings ending with b preceded by n a's using the grammar $a^n b$ (note: input n value)   |         |
| 3  | Design, develop and implement YACC/C program to construct Predictive / LL(1) Parsing Table for the grammar rules: $A \rightarrow aBa$ , $B \rightarrow bB \mid \epsilon$ . Use this table to parse the sentence: abba\$   |         |
| 4  | Design, develop and implement YACC/C program to demonstrate Shift Reduce Parsing technique for the grammar rules: $E \rightarrow E+T \mid T$ , $T \rightarrow T * F \mid F$ , $F \rightarrow (E) \mid id$ and parse the sentence: id + id * id.   |         |
| 5  | Design, develop and implement a C/Java program to generate the machine code using Triples for the statement $A = -B * (C + D)$ whose intermediate code in three-address form:<br>$T1 = -B$ $T2 = C + D$ $T3 = T1 + T2$ $A = T3$   |         |
| 6  | a) Write a LEX program to eliminate comment lines in a C program and copy the resulting program into a separate file.<br>b) Write YACC program to recognize valid identifier, operators and keywords in the given text (C program) file.  |         |
| 7  | Design, develop and implement a C/C++/Java program to simulate the working of Shortest remaining time and Round Robin (RR) scheduling algorithms. Experiment with different quantum sizes for RR algorithm.   |         |
| 8  | Design, develop and implement a C/C++/Java program to implement Banker's algorithm. Assume suitable input required to demonstrate the results.  |         |
| 9  | Design, develop and implement a C/C++/Java program to implement page replacement algorithms LRU and FIFO. Assume suitable input required to demonstrate the results.  |         |

- **VISION OF THE INSTITUTE**

Acharya Institute of Technology, committed to the cause of value-based education in all disciplines, envisions itself as fountainhead of innovative human enterprise, with inspirational initiatives for Academic Excellence

- **MISSION OF THE INSTITUTE**

Acharya Institute of Technology, strives to provide excellent academic ambiance to the students for achieving global standards of technical education, foster intellectual and personal development, meaningful research and ethical service to sustainable societal needs.

- **VISION OF THE DEPARTMENT**

Envisions to be recognized for quality education and research in the field of Computing, leading to creation of globally competent engineers, who are innovative and adaptable to the changing demands of industry and society.

- **MISSION OF THE DEPARTMENT**

- ✓ Act as a nurturing ground for young computing aspirants to attain the excellence by imparting quality education.
- ✓ Collaborate with industries and provide exposure to latest tools/ technologies.
- ✓ Create an environment conducive for research and continuous learning

## **PROGRAM EDUCATIONAL OBJECTIVES (PEOs)**

Students shall

- Have a successful career in academia, R&D organizations, IT industry or pursue higher studies in specialized field of Computer Science & Engineering and allied disciplines.
- Be competent, creative and a valued professional in the chosen field
- Engage in life-long learning, professional development and adapt to the working environment quickly
- Become effective collaborators and exhibit high level of professionalism by leading or participating in addressing technical, business, environmental and societal challenges.

## **PROGRAM SPECIFIC OUTCOMES (PSOs)**

**PSO**

**statement**

Students shall

- PSO-1:** Apply the knowledge of hardware, system software, algorithms, networking and data bases.
- PSO-2:** Design, analyze and develop efficient and secure algorithms using appropriate data structures, databases for processing of data.
- PSO-3** Be capable of developing stand alone, embedded and web-based solutions having easy to operate interface using software engineering practices and contemporary computer programming languages.

## **PROGRAM OUTCOMES (POs)**

Engineering Graduates will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **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.
6. **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.

7. **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.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** 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 clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the 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 environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

## 1. INTRODUCTION TO LEX

Lex and YACC helps you write programs that transforms structured input. Lex generates C code for lexical analyzer whereas YACC generates Code for Syntax analyzer. Lexical analyzer is build using a tool called LEX. Input is given to LEX and lexical analyzer is generated. Lex is a UNIX utility. It is a program generator designed for lexical processing of character input streams. Lex generates C code for lexical analyzer. It uses the patterns that match strings in the input and converts the strings to tokens. Lex helps you by taking a set of descriptions of possible tokens and producing a C routine, which we call a lexical analyzer. The token descriptions that Lex uses are known as regular expressions.

### 1.1 Steps in writing LEX Program:

1st Step: Using gedit create a file with extension l. For example: prg1.l

2nd Step: lex prg1.l

3rd Step: cc lex.yy.c -ll

4th Step: ./a.out

### 1.2 Structure of LEX source program:

{definitions}

%%

{rules}

%%

{user subroutines/code section}

%% is a delimiter to the mark the beginning of the Rule section. The second %% is optional, but the first is required to mark the beginning of the rules. The definitions and the code /subroutines are often omitted.

#### ***Lex variables***

|       |   |
|-------|---|
| yyin  | Of the type FILE*. This points to the current file being parsed by the lexer.   |
| yyout | Of the type FILE*. This points to the location where the output of the lexer will be written. By default, both yyin and yyout point to standard input and output. |

|          |   |
|----------|---|
| yytext   | The text of the matched pattern is stored in this variable (char*).                   |
| yytext   | Gives the length of the matched pattern.  |
| yylineno | Provides current line number information. (May or may not be supported by the lexer.) |

### ***Lex functions***

|               |  |
|---------------|--|
| yylex( )      | The function that starts the analysis. It is automatically generated by Lex.   |
| yywrap( )     | This function is called when end of file (or input) is encountered. If this function returns 1, the parsing stops. So, this can be used to parse multiple files. Code can be written in the third section, which will allow multiple files to be parsed. The strategy is to make yyin file pointer (see the preceding table) point to a different file until all the files are parsed. At the end, yywrap() can return 1 to indicate end of parsing. |
| yyless(int n) | This function can be used to push back all but first 'n' characters of the read token.   |
| yymore( )     | This function tells the lexer to append the next token to the current token.   |

### **Regular Expressions**

It is used to describe the pattern. It is widely used to in lex. It uses meta language. The character used in this Meta language are part of the standard ASCII character set. An expression is made up of symbols. Normal symbols are characters and numbers, but there are other symbols that have special meaning in Lex. The following two tables define some of the symbols used in Lex and give a few typical examples.

| Character     | Meaning   |
|---------------|---|
| A-Z, 0-9, a-z | Characters and numbers that form part of the pattern.   |
| .             | Matches any character except \n.  |
| -             | Used to denote range. Example: A-Z implies all characters from A to Z.  |
| [ ]           | A character class. Matches any character in the brackets. If the first character is ^ then it indicates a negation pattern. Example: [abc] matches either of a, b, c. |



| Character    | Meaning   |
|--------------|---|
| *            | Match zero or more occurrences of the preceding pattern.  |
| +            | Matches one or more occurrences of the preceding pattern. (no empty string) Ex: [0-9]+ matches "1","111" or "123456" but not an empty string.   |
| ?            | Matches zero or one occurrences of the preceding pattern.<br>Ex: -?[0-9]+ matches a signed number including an optional leading minus.  |
| ?            | Matches zero or one occurrences of the preceding pattern.<br>Ex: -?[0-9]+ matches a signed number including an optional leading minus.  |
| \$           | Matches end of line as the last character of the pattern.   |
| { }          | 1) Indicates how many times a pattern can be present. Example:<br>A{1,3} implies one to three occurrences of A may be present.<br>2) If they contain name, they refer to a substitution by that name. Ex: {digit} |
| \            | Used to escape meta characters. Also used to remove the special meaning of characters as defined in this table.<br>Ex: \n is a newline character, while "\*" is a literal asterisk.                               |
| ^            | Negation.   |
|              | Matches either the preceding regular expression or the following regular expression. Ex: cow sheep pig matches any of the three words.  |
| "< symbols>" | Literal meanings of characters. Meta characters hold.   |
| /            | Look ahead. Matches the preceding pattern only if followed by the succeeding expression. Example: A0/1 matches A0 only if A01 is the input.   |
| ( )          | Groups a series of regular expressions together into a new regular expression. Ex: (01) represents the character sequence 01. Parentheses are useful when building up complex patterns with *,+ and               |

*Examples of regular expressions*

| Regular expression | Meaning   |
|--------------------|---|
| joke[rs]           | Matches either jokes or joker.  |
| A{1,2}shis+        | Matches AAshis, Ashis, AAshi, Ashi.   |
| (A[b-e])+          | Matches zero or one occurrences of A followed by any character from b to e. |
| [0-9]              | 0 or 1 or 2 or .....9   |
| [0-9]+             | 1 or 111 or 12345 or ...At least one occurrence of preceding exp            |
| [0-9]*             | Empty string (no digits at all) or one or more occurrence.                  |
| -?[0-9]+           | -1 or +1 or +2 .....  |
| [0.9]*\.[0.9]+     | 0.0,4.5 or .31415 But won't match 0 or 2                                    |

*Examples of token declarations*

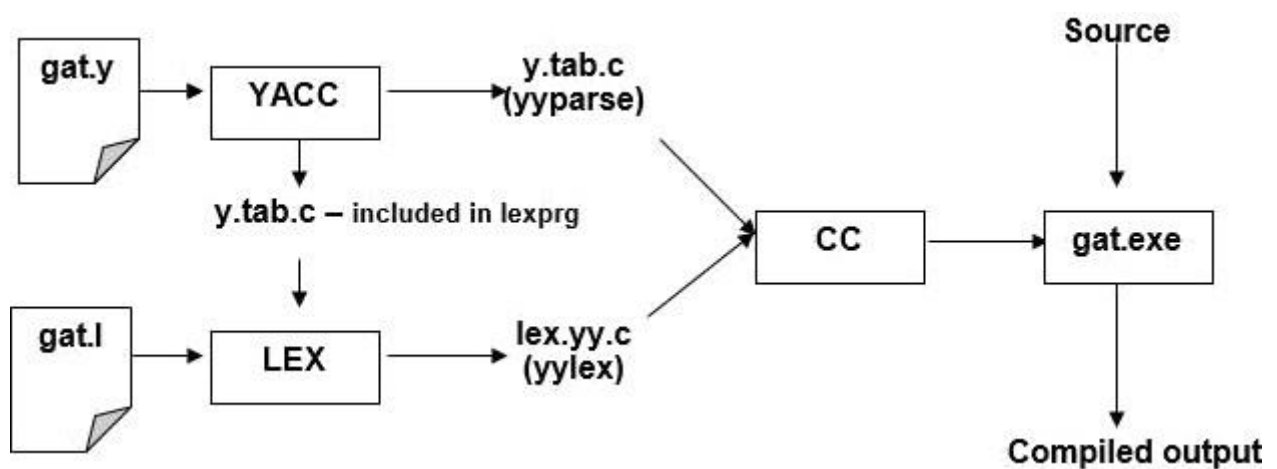
| Token    | Associated expression               | Meaning                          |
|----------|-------------------------------------|----------------------------------|
| number   | ([0-9])+                            | 1 or more occurrences of a digit |
| chars    | [A-Za-z]                            | Any character                    |
| blank    | " "                                 | A blank space                    |
| word     | (chars)+                            | 1 or more occurrences of chars   |
| variable | (chars)+(number)*(chars)*( number)* |                                  |

## 2. Introduction to YACC

YACC provides a general tool for imposing structure on the input to a computer program. The input specification is a collection of grammar rules. Each rule describes an allowable structure and gives it a name. YACC prepares a specification of the input process. YACC generates a function to control the input process. This function is called a parser.

The name is an acronym for “Yet Another Compiler Compiler”. YACC generates the code for the parser in the C programming language. YACC was developed at AT& T for the UNIX operating system. YACC has also been rewritten for other languages, including Java, Ada.

The function parser calls the lexical analyzer to pick up the tokens from the input stream. These tokens are organized according to the input structure rules. The input structure rule is called as grammar. When one of the rule is recognized, then user code supplied for this rule ( user code is action) is invoked. Actions have the ability to return values and makes use of the values of other actions.



### Steps in writing and Executing YACC Program:

**1<sup>st</sup> step:** Using gedit editor create a file with extension `.y`.

For example: `prg1.y`

**2<sup>nd</sup> Step:** `lex prg1.l`

**3<sup>rd</sup> Step:** YACC -d prg1.y

**4<sup>th</sup> Step:** cc lex.yy.c y.tab.c -ll

**5<sup>th</sup> Step:** ./a.out

When we run YACC, it generates a parser in file y.tab.c and also creates an include file y.tab.h. To obtain tokens, YACC calls yylex. Function yylex has a return type of int, and returns the token. Values associated with the token are returned by lex in variable yylval.

### Structure of YACC source program:

#### Basic Specification:

Every YACC specification file consists of three sections. The declarations, Rules (of grammars), programs. The sections are separated by double percent “%%” marks. The % is generally used in YACC specification as an escape character.

The general format for the YACC file is very similar to that of the Lex file.

| 1 <sup>st</sup> Col | 2 <sup>nd</sup> Col | 3 <sup>rd</sup> Col | 4 <sup>th</sup> Col |
|---------------------|---------------------|---------------------|---------------------|
|                     | DEFINITION SECTION  |                     |                     |
| %%                  |                     |                     |                     |
|                     | RULE SECTION        |                     |                     |
| %%                  |                     |                     |                     |
|                     | CODE SECTION        |                     |                     |

%% is a delimiter to mark the beginning of the Rule section.

#### Definition Section

|        |   |
|--------|---|
| %union | It defines the Stack type for the Parser. It is a union of various datas/structures/ objects  |
| %token | These are the terminals returned by the yylex function to the YACC. A token can also have type associated with it for good type checking and syntax directed translation. A type of a token can be specified as %token <stack member>tokenName.<br>Ex: %token NAME NUMBER |
| %type  | The type of a non-terminal symbol in the Grammar rule can be specified with this mode. The format is %type <stack member>non-terminal.  |

|           |  |
|-----------|--|
| %no assoc | Specifies that there is no associativity of a terminal symbol.   |
| %left     | Specifies the left associativity of a Terminal Symbol  |
| %right    | Specifies the right associativity of a Terminal Symbol.  |
| %start    | Specifies the L.H.S non-terminal symbol of a production rule which should be taken as the starting point of the grammar rules. |
| %prec     | Changes the precedence level associated with a particular rule to that of the following token name or literal                  |

### Rules Section

The rules section simply consists of a list of grammar rules. A grammar rule has the form: A:

#### BODY

A represents a nonterminal name, the colon and the semicolon are YACC punctuation and BODY represents names and literals. The names used in the body of a grammar rule may represent tokens or nonterminal symbols. The literal consists of a character enclosed in single quotes.

Names representing tokens must be declared as follows in the declaration sections:

#### %token name1 name2...

Every name not defined in the declarations section is assumed to represent a non-terminal symbol. Every non-terminal symbol must appear on the left side of at least one rule. Of all the non-terminal symbols, one, called the start symbol has a particular importance. The parser is designed to recognize the start symbol. By default the start symbol is taken to be the left hand side of the first grammar rule in the rules section.

With each grammar rule, the user may associate actions to be. These actions may return values, and may obtain the values returned by the previous actions. Lexical analyzer can return values for tokens, if desired. An action is an arbitrary C statement. Actions are enclosed in curly braces.

**File Utilities**

| Action   | UNIX options & filespec            |
|--|------------------------------------|
| View a file  | vi file.txt                        |
| Concatenate files  | cat file1file2 to standard output. |
| Counts-lines,-words, and- characters in a file                 | wc -l                              |
| Displays line-by-line differences between pairs of text files. | Diff                               |
| calculator   | Bc                                 |
| calendar for September, 1752 (when leap years began)           | cal 9 1752                         |

**Controlling program execution for C-shell**

|      |   |
|------|---|
| &    | Run job in background                     |
| ^c   | Kill job in foreground                    |
| ^z   | Suspend job in foreground                 |
| Fg   | Restart suspended job in foreground       |
| Bg   | Run suspended job in background           |
| ;    | Delimit commands on same line             |
| ()   | Group commands on same line               |
| !    | re-run earlier commands from history list |
| jobs | List current jobs                         |

**Controlling program input/output for C-shell**

|        |  |
|--------|--|
|        | Pipe output to input                             |
| >      | Redirect output to a storage file                |
| <      | Redirect input from a storage file               |
| >>     | Append redirected output to a storage file       |
| Tee    | Copy input to both file and next program in pipe |
| Script | Make file record of al terminal activity         |

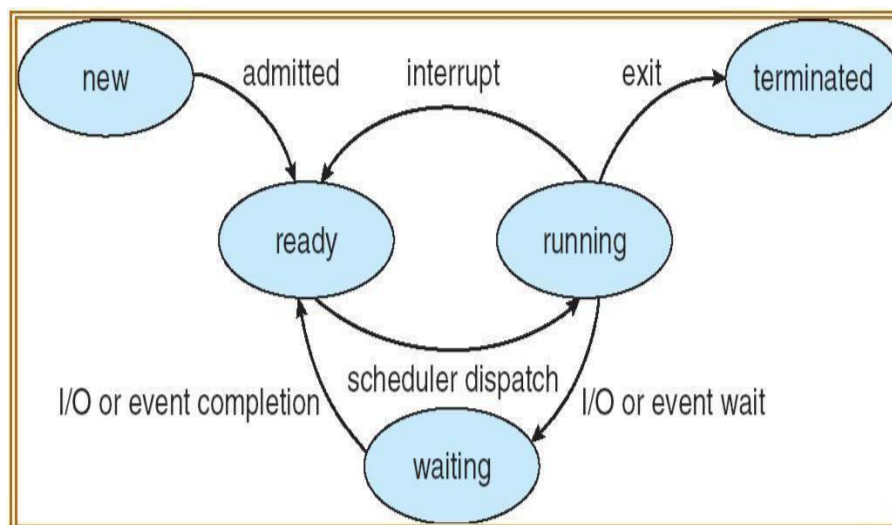
### 3. Introduction to Operating Systems

#### *Introduction*

An Operating System is a program that manages the Computer hardware. It controls and coordinates the use of the hardware among the various application programs for the various users.

A Process is a program in execution. As a process executes, it changes *state*

- New : The process is being created
- Running : Instructions are being executed
- Waiting : The process is waiting for some event to occur
- Ready : The process is waiting to be assigned to a process
- Terminated : The process has finished execution



Apart from the program code, it includes the current activity represented by Program Counter, Contents of Processor registers, Process Stack which contains temporary data like function parameters, return addresses and local variables Data section which contains global variables, Heap for dynamic memory allocation.

A Multi-programmed system can have many processes running simultaneously with the CPU multiplexed among them. By switching the CPU between the processes, the OS can

make the computer more productive. There is Process Scheduler which selects the process among many processes that are ready, for program execution on the CPU. Switching the CPU to another process requires performing a state save of the current process and a state restore of new process, this is Context Switch.

## Scheduling Algorithms

CPU Scheduler can select processes from ready queue based on various scheduling algorithms. Different scheduling algorithms have different properties, and the choice of a particular algorithm may favor one class of processes over another.

### The scheduling criteria include

*CPU utilization:*

*Throughput:* The number of processes that are completed per unit time.

*Waiting time:* The sum of periods spent waiting in ready queue.

*Turnaround time:* The interval between the time of submission of process to the time of completion.

*Response time:* The time from submission of a request until the first response is produced.

### The different scheduling algorithms are

FCFS: First Come First Served Scheduling

SJF: Shortest Job First Scheduling

SRTF: Shortest Remaining Time First

Scheduling Priority Scheduling

Round Robin Scheduling

Multilevel Queue Scheduling

Multilevel Feedback Queue Scheduling

### *i. Deadlocks*

A process requests resources; and if the resource is not available at that time, the process enters a waiting state. Sometimes, a waiting process is never able to change state, because the resource it has requested is held by another process which is also waiting. This situation is called Deadlock. Deadlock is characterized by four necessary conditions

- Mutual Exclusion
- Hold and Wait



- No Preemption
- Circular Wait

Deadlock can be handled in one of these ways,

- Deadlock Avoidance
- Deadlock Detection and
- Recovery

## Sample Lex Programs

### 1. Lex program to count number of vowels and consonants.

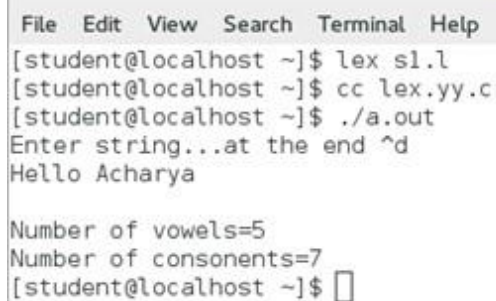
```
%{
#include<stdio.h>
int vowels=0;
int cons=0;
%}

%%
[aeiouAEIOU] {vowels++;}
[a-zA-Z] {cons++;}
%%

int yywrap()
{return 1;}

Int main()
{
printf("Enter string...at the end ^d\n");
yylex();
printf("Number of vowels=%d\nNumber of consonents=%d\n",vowels,cons);
return 0;
}
```

### Output:



```
File Edit View Search Terminal Help
[student@localhost ~]$ lex s1.l
[student@localhost ~]$ cc lex.yy.c
[student@localhost ~]$ ./a.out
Enter string...at the end ^d
Hello Acharya

Number of vowels=5
Number of consonents=7
[student@localhost ~]$
```

**2. LEX program to count character, word, line and space.**

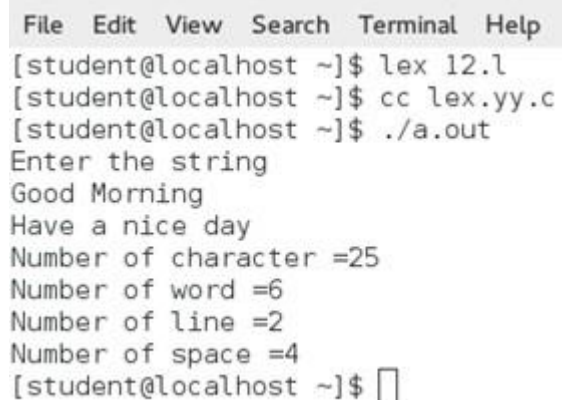
```
%{
    int c=0,l=0,w=0,s=0;
}%

space [ ]
character [a-zA-Z]
digit [0-9]
word ({character}|{digit})+
line \n

%%
{line} {c++; l++;}
{word} {w++; c+=strlen(yytext);}
{space} {s++;}
. {c++;}
%%

int yywrap()
{return 1;}

main()
{
    printf("Enter the string\n");
    yylex();
    printf("Number of character =%d\n",c);
    printf("Number of word =%d\n",w);
    printf("Number of line =%d\n",l);
    printf("Number of space =%d\n",s);
}
```

**Output:**

```
File Edit View Search Terminal Help
[student@localhost ~]$ lex 12.l
[student@localhost ~]$ cc lex.yy.c
[student@localhost ~]$ ./a.out
Enter the string
Good Morning
Have a nice day
Number of character =25
Number of word =6
Number of line =2
Number of space =4
[student@localhost ~]$
```

**3. LEX program to count number of a's in the given string.**

```
%{
    #include<stdio.h>
    int c=0;
}%

%%
[a]* {c++;}
. ;
%%

int yywrap()
{ return 1;}

main()
{
    printf("Enter a string\n");
    yylex();
    printf("Number of a's:%d\n",c);
}
```

**Output:**

```
[student@localhost ~]$ lex p3.l
[student@localhost ~]$ cc lex.yy.c
[student@localhost ~]$ ./a.out
enter a string
aa

no of a: 2
[student@localhost ~]$ ./a.out
enter a string
bcd

no of a: 0
[student@localhost ~]$
```

**4. LEX program to count number of a) +ve and -ve integers b) +ve and -ve fractions.**

```
%{
    #include<stdio.h>
    int posint=0,negint=0,posfraction=0,negfraction=0;
}%

%%
[-] [0-9]+ {negint++;}
[+]?[0-9]+ {posint++;}
[+]?[0-9]*\.[0-9]+ {posfraction++;}
[-][0-9]*\.[0-9]+ {negfraction++;}
%%

int yywrap() {return 1;}

main(int argc, char *argv[])
{
    if(argc!=2)
    {
        printf("Usage:./a.out<sourcefile>\n");
        exit(0);
    }
    yyin=fopen(argv[1],"r");
    yylex();
    printf("Number of +ve integers=%d\n",posint);
    printf("Number of -ve integers=%d\n",negint);
    printf("Number of +ve fractions=%d\n",posfraction);
    printf("Number of -ve fractions=%d\n",negfraction);
}
```

**pat.txt**

+13 52 -18 +32 -8.2 33.25 +5.83

**Output:**

```
student@iselab302-24 ~]$ lex p3.l
student@iselab302-24 ~]$ cc lex.yy.c
student@iselab302-24 ~]$ ./a.out pat.txt
no of +ve integers=3
no of -ve integer=1
No of +ve fractions=2
No of -ve fractions=1
student@iselab302-24 ~]$ █
```

## 5. Lex program to include line number for the contents of input files.

```
%{
    int lineno=1;
}%

line .*\n

%%
{line} {printf("%10d %s",lineno++,yytext);}
%%

int yywrap() {return 1;}

int main(int argc,char **argv )
{
    extern FILE **yyin;
    yyin=fopen("test.c","r");
    yylex();
    return 0;
}

/* test.c */
#include<stdio.h>
#include<conio.h>
main()
{
    int a,b,c;
    printf("enter 2 numbers\n:");
    scanf("%d%d",a,b);
    c=a+b;
    printf("result=%d",c);
}
```

### Output:

```
student@iselab302-24 ~]$ lex r99.l
student@iselab302-24 ~]$ cc lex.yy.c
student@iselab302-24 ~]$ ./a.out
1 #include<stdio.h>
2 #include<conio.h>
3 main()
4 {
5 int a,b,c;
6 printf("enter 2 numbers\n:");
7 scanf( "%d%d",a,b);
8 c=a+b;
9 printf("result=%d",c);
10 }
11
12
```

**6. Implement a program using YACC tool to recognize all strings having pattern anb where  $n \geq 10$**

```
/*LEX part*/

%{
#include "y.tab.h"
%}

%%

a {return A;}
b {return B;}
[.|\n] {return 0;}
%%

int yywrap()
{return 1;}

/*YACC part*/

%{
#include<stdio.h>
#include<stdlib.h>
int yylex();
int yyerror();
%}
%token A B
%%
start:A A A A A A A A S B
S: A S
  |
  ;

%%

int main()
{
printf("enter expression:\n");
yyparse();
printf("valid\n");
}

int yyerror()
{
printf("invalid\n");
exit(0);
}
```

**Output:**

```
File Edit View Search Terminal Help
[student@localhost ~]$ lex jp2.l
[student@localhost ~]$ yacc -d js2.y
[student@localhost ~]$ cc lex.yy.c y.tab.c
[student@localhost ~]$ ./a.out
enter expression:
aaaaaaaaaaaab
valid
[student@localhost ~]$
```



## Laboratory Programs

1a) Write a LEX program to recognize valid 17 arithmetic expression. Identifiers in the expression could be only integers and operators could be + and \*. Count the identifiers & operators present and print them separately.

```
%{
#include<stdio.h>
#include<stdlib.h>
int cid=0,cop=0,cbr=0;
}%
%%
[+*] {cop++; printf("\noperator is %s",yytext);}
[0-9]*[.]?[0-9]+ {cid++; printf("\noperand is %s",yytext);}
[(] {cbr++;}
[)] {cbr--;}
[.|\n] {return cop,cid,cbr;}
%%
int yywrap()
{
    return 1;
}
main()
{
    printf("enter expression\n");
    yylex();
    if(cbr!=0||cop>=cid||cid==0)
    {
        printf("invalid expression\n");
        exit(0);
    }
    else
        printf("\n valid expression");
    printf("\n no of operators=%d\n",cop);
    printf("\n no of identifiers=%d\n",cid);
    return 0;
}
```

## Output

```
[student@localhost ~]$ lex g1.l
[student@localhost ~]$ cc lex.yy.c
[student@localhost ~]$ ./a.out
enter expression
(2+3)

operand is 2
operator is +
operand is 3
valid expression
no of operators=1

no of identifiers=2
[student@localhost ~]$ ./a.out
enter expression
(3+

operand is 3
operator is +invalid expression
[student@localhost ~]$ □
```

**1 b) Write YACC program to evaluate arithmetic expression involving operators: +, -, \*, and /.****Lex part**

```
%{
#include"y.tab.h"
int yylval;
}%
%%
[0-9]+ {yylval=atoi(yytext);return NUM;}
[\n] {return 0;}
. {return yytext[0];}
%%
```

**Yacc Part**

```
%{
#include<stdio.h>
#include<stdlib.h>
int yylex();
int yyerror();
}%
%token NUM
%left '+' '-'
%left '*' '/'
%%
Start:E {printf("Valid expression\n");
printf("Result=%d", $$);}
E:E '+' E {$$=$1+$3;}
|E '-' E {$$=$1-$3;}
|E '*' E {$$=$1*$3;}
|E '/' E {if($3==0)
{printf("Divide by zero error\n");
exit(0);}
}
else
{$$=$1/$3;}
}
| '(' E ')' {$$=$2;}
| NUM {$$=$1;}
;
%%
void main()
{
printf("Enter Expression:");
yyparse();
}
int yyerror()
{
printf("Invalid expression\n");
exit(0);
}
```

**Output**

Enter Expression

1 + 2

Valid Expression

Result=3

**2. Develop, Implement and Execute a program using YACC tool to recognize all strings ending with b preceded by na's using the grammar  $a^n b$  (note: input n value)****Lex part**

```
%{
#include"y.tab.h"
}%
%%
[a] {return A;}
[b] {return B;}
[\\n] {return 0;}
. { return yytext[0];}
%%
```

**Yacc Part**

```
%{
#include<stdio.h>
#include<stdlib.h>
int yylex();
int yyerror();
}%

%token A B
%%
S:A S|B;
%%
int main()
{
printf("Enter expression: ");
yyparse();
printf("Expression is valid!!!");
}
int yyerror()
{
printf(" Expression is invalid");
exit(0);
}
```

**Output**

```
Run 1:
Enter expression
aaab
Expression is valid

Run 1:
Enter expression
aabbb
Expression is invalid
```

**3. Design, develop and implement YACC/C program to construct Predictive / LL (1) Parsing Table for the grammar rules:  $A \rightarrow aBa$ ,  $B \rightarrow bB \mid \epsilon$ . Use this table to parse the sentence: abba\$**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
char s[20],stack[20];
int main()
{
char m[2][3][10]={
{"aBa","Error","Error"},
{"n","bB","Error"}
};
int size[2][3]={3,5,5,1,2,5};
int i,j,k,n,row,col;
printf("\n Enter the input string: ");
scanf("%s",s);
strcat(s,"$");
n=strlen(s);
stack[0]='$';
stack[1]='A';
i=1;
j=0;
printf("\nStack\t\tInput\n");
printf("_____\t\t_____\n");
if(s[n-2]!='a')
{
printf("ERROR\n");
exit(0);
}
while((stack[i]!='$') && (s[j]!='$'))
{
if(stack[i]==s[j])
```

```
{
i--;
j++;
}
switch(stack[i])
{
case 'A': row=0;
break;
case 'B': row=1;
break;
}
switch(s[j])
{
case 'a': col=0;
break;
case 'b': col=1;
break;
case '$': col=2;
break;
default:printf("ERROR\n");exit(0);
}
if(row==1&&col==2)
i--;
else if(m[row][col][0]=='E')
{
printf("\nERROR");
exit(0);
}
else if(m[row][col][0]=='n')
i--;
else if(m[row][col][0]==s[j])
{
```

```
for(k=size[row][col]-1;k>=0;k--)  
{  
    stack[i]=m[row][col][k];  
    i++;  
}  
i--;  
  
for(k=0;k<=i;k++)  
    printf("%c",stack[k]);  
printf("\t\t");  
for(k=j;k<=n;k++)  
    printf("%c",s[k]);  
printf("\n");  
  
if (i <= 0)  
    printf("\n SUCCESS");  
else  
    printf("\n Error");  
return 0;  
getch();  
}
```



**4. Design, develop and implement YACC/C program to demonstrate Shift Reduce Parsing technique for the grammar rules:  $E \rightarrow E+T \mid T$ ,  $T \rightarrow T * F \mid F$ ,  $F \rightarrow (E) \mid id$  and parse the sentence:  $id + id * id$ .**

```
#include<stdio.h>
#include<string.h>
int k=0,z=0,i=0,j=0,c=0;
char a[16],ac[20],stk[15],act[10];
void check();
void main()
{
puts("grammar is E-E+E\nE-E*E\nE-(E)\nE-id");
puts("enter input string");
scanf("%s",a);
c=strlen(a);
strcpy(act,"SHIFT->");
puts("stack \t input \t action");
for(k=0,i=0;j<c;k++,i++,j++)
{
if(a[j]=='i' && a[j+1]=='d')
{
stk[i]=a[j];
stk[i+1]=a[j+1];
stk[i+2]='\0';
a[j]=' ';
a[j+1]=' ';
printf("\n%s\t%s$\t%sid",stk,a,act);
check();
}
else
{
stk[i]=a[j];
stk[i+1]='\0';
a[j]=' ';
printf("\n%s\t%s$\t%sid",stk,a,act);
check();
}
}
if (stk[i-1]=='E' && a[0]==' ')
printf("\nAccepted\n");
else
printf("\n Rejected\n");
}

void check()
{
strcpy(ac,"reduce to E");
for(z=0;z<c;z++)
if(stk[z]=='i' && stk[z+1]=='d')
{
stk[z]='E';
stk[z+1]='\0';
printf("\n%s\t%s$\t%sid",stk,a,ac);
```

```
j++;
}
for(z=0;z<c;z++)
    if(stk[z]=='E'&&stk[z+1]=='+'&&stk[z+2]=='E')
    {
        stk[z]='E';
        stk[z+1]='\0';
        stk[z+2]='\0';
        printf("\n$%s\t%s$\t%sid",stk,a,ac);
        i=i-2;
    }
for(z=0;z<c;z++)
    if(stk[z]=='E'&&stk[z+1]=='*'&&stk[z+2]=='E')
    {
        stk[z]='E';
        stk[z+1]='\0';
        stk[z+2]='\0';
        printf("\n$%s\t%s$\t%sid",stk,a,ac);
        i=i-2;
    }
for(z=0;z<c;z++)
    if(stk[z]=='('&&stk[z+1]=='E'&&stk[z+2]==')')
    {
        stk[z]='E';
        stk[z+1]='\0';
        stk[z+2]='\0';
        printf("\n$%s\t%s$\t%sid",stk,a,ac);
        i=i-2;
    }
}
```

**5) Design, develop and implement a C/Java program to generate the machine code using Triples for the statement  $A = -B * (C + D)$  whose intermediate code in three-address form:**

**T1 = -B**

**T2 = C + D**

**T3 = T1 + T2**

**A = T3**

```
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<string.h>
char op[2],arg1[5],arg2[5],result[5];
void main()
{
FILE *fp1,*fp2;
fp1=fopen("input.txt","r");
fp2=fopen("output.txt","w");
while(!feof(fp1))
{
fscanf(fp1,"%s%s%s",result,arg1,op,arg2);
if(strcmp(op,"+")==0)
{
fprintf(fp2,"\nmov r0,%s",arg1);
fprintf(fp2,"\nadd r0,%s",arg2);
fprintf(fp2,"\nmov %s,r0",result);
}
if(strcmp(op,"*")==0)
{
fprintf(fp2,"\nmov r0,%s",arg1);
fprintf(fp2,"\nmul r0,%s",arg2);
fprintf(fp2,"\nmov %s,r0",result);
}
if(strcmp(op,"-")==0)
{
fprintf(fp2,"\nmov r0,%s",arg1);
fprintf(fp2,"\nsub r0,%s",arg2);
fprintf(fp2,"\nmov %s,r0",result);
}
if(strcmp(op,"/")==0)
{
fprintf(fp2,"\nmov r0,%s",arg1);
fprintf(fp2,"\ndiv r0,%s",arg2);
fprintf(fp2,"\nmov %s,r0",result);
}
if(strcmp(op,"=")==0)
{
fprintf(fp2,"\nmov r0,%s",arg1);
fprintf(fp2,"\nmov %s,r0",result);
} }
fclose(fp1);
fclose(fp2); }
```

**6 a) Write a LEX program to eliminate comment lines in a C program and copy the resulting program into a separate file.**

**Lex part**

```
%{
#include<stdio.h>
int c1=0;
}%
%%
"/*" [a-zA-Z0-9' '\t\n] "*" /" {c1++;}
"//" [a-zA-Z0-9' '\t] * {c1++;}
%%
int main()
{
yyin=fopen("a.c","r");
yyout=fopen("b.c","w");
yylex();
printf("\n Number of comment lines = %d",c1);
fclose(yyin);
fclose(yyout);
}
int yywrap()
{
return 1;
}
```

**6 b) Write YACC program to recognize valid identifier, operators and keywords in the given text (C program) file.**

### Lex part

```
%{
#include<stdio.h>
#include "y.tab.h"
int yylval;
}%
%%
[\\t];
"(.*)";
#include<stdio.h>
[+|-|*|/|=|<|>|] { printf("Operator is %s\\n",yytext); return OP;}
[0-9]+ {yylval=atoi(yytext); printf("Number is %d\\n",yylval); return DIGIT;}
int|char|bool|float|void|for|do|while|if|else|return {printf("keyword is
%s\\n",yytext); return KEY;}
[a-zA-Z0-9]+ {printf("identifier is %s\\n",yytext); return ID;}
. ;
%%
```

### Yacc Part

```
%{
#include<stdio.h>
#include<stdlib.h>
int id=0,dig=0,key=0,op=0;
int yylex();
int yyerror();

extern FILE *yyin;

%}
%token DIGIT ID KEY OP
%%
input:
DIGIT input{dig++;}
|ID input{id++;}
|KEY input{key++;}
|OP input{op++;}
|DIGIT{dig++;}
|ID{id++;}
|KEY{key++;}
|OP{op++;}
;
%%
void main(){
yyin=fopen("6c.c","r");
yyparse();

printf("numbers=%d\\n keyword=%d\\n identifier=%d\\n opeator=%d\\n",dig,key,id,op);
```

```
}  
int yyerror()  
{  
printf("EEK ;parse error! Message:");  
exit(-1);  
}
```

**7. Design, develop and implement a C/C++/Java program to simulate the working of Shortest remaining time and Round Robin (RR) scheduling algorithms. Experiment with different quantum sizes for RR algorithm.**

```
#include<stdio.h>
struct proc
{
int id;
int arrival;
int burst;
int rem;
int wait;
int finish;
int turnaround;
float ratio;
}process[10]; //structure to hold the process information
struct proc temp;
int no;
int chkprocess(int);
int nextprocess();
void roundrobin(int, int, int[], int[]);
void srtf(int);
main()
{
int n,tq,choice;
int bt[10],st[10],i,j,k;
for(; ; )
{
printf("Enter the choice \n");
printf(" 1. Round Robin\n 2. SRT\n 3. Exit \n");
scanf("%d",&choice);
switch(choice)
{
case 1:
printf("Round Robin scheduling algorithm\n");
printf("Enter number of processes:\n");
scanf("%d",&n);
printf("Enter burst time for sequences:");
for(i=0;i<n;i++)
{
scanf("%d",&bt[i]);
st[i]=bt[i]; //service time
}
printf("Enter time quantum:");
scanf("%d",&tq);
roundrobin(n,tq,st,bt);
break;
case 2: printf("\n \n ---SHORTEST REMAINING TIME
NEXT---\n \n "); printf("\n \n Enter the number of
```

```
processes: "); scanf("%d", &n);
srtf(n);
break;
case 3: exit(0);
} // end of switch
// end of for
//end of main()
void roundrobin(int n,int tq,int st[],int bt[])
{
int time=0;
int tat[10],wt[10],i,count=0,swt=0,stat=0,temp1,sq=0,j,k;
float awt=0.0,atat=0.0;
while(1)
{
for(i=0,count=0;i<n;i++)
{
temp1=tq;
if(st[i]==0) // when service time of a process equals zero then
//count value is incremented
{
count++;
continue;
}
if(st[i]>tq) // when service time of a process greater than time
//quantum then time
st[i]=st[i]-tq; //quantum value subtracted from service time
else
if(st[i]>=0)
{
temp1=st[i]; // temp1 stores the service time of a process
st[i]=0; // making service time equals 0
}
sq=sq+temp1; // utilizing temp1 value to calculate turnaround
time
tat[i]=sq; // turn around time
} //end of for
if(n==count) // it indicates all processes have completed their
task
because the count value
break; // incremented when service time equals 0
} //end of while
for(i=0;i<n;i++) // to calculate the wait time and turnaround time of
each process
{wt[i]=tat[i]-bt[i]; // waiting time calculated from the turnaround time
burst time
swt=swt+wt[i]; // summation of wait time
stat=stat+tat[i]; // summation of turnaround time
}
```



```

awt=(float)swt/n; // average wait time
atat=(float)stat/n; // average turnaround time
printf("Process_no Burst time Wait time Turn around time\n");
for(i=0;i<n;i++)
printf("%d\t\t%d\t\t%d\t\t%d\n",i+1,bt[i],wt[i],tat[i]);
printf("Avg wait time is %f\n Avg turn around time is %f\n",awt,atat);
} // end of Round Robin
int chkprocess(int s) // function to check process remaining time is zero
or not
{
int i;
for(i = 1; i <= s; i++)
{
if(process[i].rem != 0)
return 1;
}
return 0;
} // end of chkprocess
int nextprocess() // function to identify the next process to be executed
{
int min, l, i;
min = 32000; //any limit assumed
for(i = 1; i <= no; i++)
{
if( process[i].rem!=0 && process[i].rem < min)
{
min = process[i].rem;
l = i;
}
}
return l;
} // end of nextprocess
void srtf(int n)
{
int i,j,k,time=0;
float tavg,wavg;
for(i = 1; i <= n; i++)
{
process[i].id = i;
printf("\n\nEnter the arrival time for process %d: ", i);
scanf("%d", &(process[i].arrival));
printf("Enter the burst time for process %d: ", i);
scanf("%d", &(process[i].burst));
process[i].rem = process[i].burst;
}
for(i = 1; i <= n; i++)
{
for(j = i + 1; j <= n; j++)

```

```

{
if(process[i].arrival > process[j].arrival) // sort arrival time of a
process
{
temp = process[i];
process[i] = process[j];
process[j] = temp;
}
}
}
no = 0;
j = 1;
while(chkprocess(n) == 1)
{
if(process[no + 1].arrival == time)
{
while(process[no+1].arrival==time)
no++;
if(process[j].rem==0)
process[j].finish=time;
j = nextprocess();
}
if(process[j].rem != 0) // to calculate the waiting time of a process
{
process[j].rem--;
for(i = 1; i <= no; i++)
{
if(i != j && process[i].rem != 0)
process[i].wait++;
}
}
else
{
process[j].finish = time;
j=nextprocess();
time--;
k=j;
}
time++;
}
    process[k].finish = time;
printf("\n\n\t\t\t\t\t---SHORTEST REMAINING TIME FIRST---");
printf("\n\n Process Arrival Burst Waiting Finishing turnaround
Tr/Tb\n");
printf("%5s %9s %7s %10s %8s %9s\n\n", "id", "time", "time", "time",
"time", "time");
for(i = 1; i <= n; i++)
{

```

```

process[i].turnaround = process[i].wait + process[i].burst; // calc of
turnaround process[i].ratio = (float)process[i].turnaround /
(float)process[i].burst;
printf("%5d %8d %7d %8d %10d %9d %10.1f ", process[i].id,
process[i].arrival, process[i].burst, process[i].wait,
process[i].finish,
process[i].turnaround, process[i].ratio);
tavg=tavg+ process[i].turnaround; //summation of turnaround time
wavg=wavg+process[i].wait; // summation of waiting time
printf("\n\n");
}
tavg=tavg/n; // average turnaround time
wavg=wavg/n; // average wait time
printf("tavg=%f\t wavg=%f\n",tavg,wavg); }// end of srtf

```

**Output:**

```

Enter the choice
1) Round Robin 2) SRT
3) Exit
1
Round Robin scheduling algorithm
*****
Enter number of processes:3
Enter burst time for sequences:24
3
3
Enter time quantum:4
Process_no Burst time Wait time Turnaround time
1 24 6 30
2 3 4 7
3 3 7 10
Avg wait time is 5.666667
Avg turnaround time is 15.666667
Enter the choice
1) Round Robin 2) SRT
3) Exit
2
---SHORTEST REMAINING TIME NEXT---
Enter the number of processes: 4
Enter the arrival time for process 1: 0
    Enter the burst time for process 1: 8
Enter the arrival time for process 2: 1
Enter the burst time for process 2: 4
Enter the arrival time for process 3: 2
Enter the burst time for process 3: 9
Enter the arrival time for process 4: 3
Enter the burst time for process 4: 5
1 24 6 30

```

```
2 3 4 7
3 3 7 10
---SHORTEST REMAINING TIME FIRST---
Enter the number of processes: 4
Enter the arrival time for process 1: 0
Enter the burst time for process 1: 8
Enter the arrival time for process 2: 1
Enter the burst time for process 2: 4
Enter the arrival time for process 3: 2
Enter the burst time for process 3: 9
Enter the arrival time for process 4: 3
Enter the burst time for process 4: 5
---SHORTEST REMAINING TIME NEXT---
Process Arrival Burst Waiting Finishing turnaround Tr/Tb
id time time time time time time
1      0   8      9 17      17  2.1
2      1   4      0   5      4   1.0
3      2   9     15 26     24   2.7
4      3   5      2  10      7   1.4
tavg=13.000000
wavg=6.500000
```

**Using OpenMP**

**8. Design, develop and implement a C/C++/Java program to implement Banker's algorithm. Assume suitable input required to demonstrate the results.**

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int Max[10][10], need[10][10], alloc[10][10], avail[10], completed[10],
    safeSequence[10];
    int p, r, i, j, process, count;
    count = 0;
    printf("Enter the no of processes : ");
    scanf("%d", &p);
    for(i = 0; i < p; i++)
        completed[i] = 0;
    printf("\n\nEnter the no of resources : ");
    scanf("%d", &r);
    printf("\n\nEnter the Max Matrix for each process : ");
    for(i = 0; i < p; i++)
    {
        printf("\nFor process %d : ", i + 1);
        for(j = 0; j < r; j++)
            scanf("%d", &Max[i][j]);
    }
    printf("\n\nEnter the allocation for each process : ");
    for(i = 0; i < p; i++)
    {
        printf("\nFor process %d : ", i + 1);
        for(j = 0; j < r; j++)
            scanf("%d", &alloc[i][j]);
    }
    printf("\n\nEnter the Available Resources : ");
    for(i = 0; i < r; i++)
        scanf("%d", &avail[i]);
    for(i = 0; i < p; i++)
        for(j = 0; j < r; j++)
            need[i][j] = Max[i][j] - alloc[i][j];
    do
    {
        printf("\n Max matrix:\tAllocation matrix:\n");
        for(i = 0; i < p; i++)
        {
            for(j = 0; j < r; j++)
                printf("%d ", Max[i][j]);
            printf("\t\t");
            for(j = 0; j < r; j++)
                printf("%d ", alloc[i][j]);
            printf("\n");
        }
    }
```

```
}
process = -1;
for(i = 0; i < p; i++)
{
    if(completed[i] == 0)//if not completed
    {
        process = i ;
        for(j = 0; j < r; j++)
        {
            if(avail[j] < need[i][j])
            {
                process = -1;
                break;
            }
        }
        if(process != -1)
            break;
    }
    if(process != -1)
    {
        printf("\nProcess %d runs to completion!", process + 1);
        safeSequence[count] = process + 1;
        count++;
        for(j = 0; j < r; j++)
        {
            avail[j] += alloc[process][j];
            alloc[process][j] = 0;
            Max[process][j] = 0;
            completed[process] = 1;
        }
    }
}
while(count != p && process != -1);
if(count == p)
{
    printf("\nThe system is in a safe state!!\n");
    printf("Safe Sequence : < ");
    for( i = 0; i < p; i++)
        printf("%d ", safeSequence[i]);
    printf(">\n");
}
else
    printf("\nThe system is in an unsafe state!!");
}
```

**Output:**

```
Enter the no of processes : 5
Enter the no of resources : 3
```

Enter the Max Matrix for each process :

For process 1 : 7

5

3

For process 2 : 3

2

2

For process 3 : 7

0

2

For process 4 : 2

2

2

For process 5 : 4

3

3

Enter the allocation for each process :

For process 1 : 0

1

0

For process 2 : 2

0

0

For process 3 : 3

0

2

For process 4 : 2

1

1

For process 5 : 0

0

2

Enter the Available Resources : 3

3

2

Max matrix: Allocation matrix:

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 7 | 5 | 3 | 0 | 1 | 0 |
|---|---|---|---|---|---|

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 3 | 2 | 2 | 2 | 0 | 0 |
|---|---|---|---|---|---|

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 7 | 0 | 2 | 3 | 0 | 2 |
|---|---|---|---|---|---|

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 2 | 2 | 2 | 2 | 1 | 1 |
|---|---|---|---|---|---|

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 4 | 3 | 3 | 0 | 0 | 2 |
|---|---|---|---|---|---|

Process 2 runs to completion!

Max matrix: Allocation matrix:

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 7 | 5 | 3 | 0 | 1 | 0 |
|---|---|---|---|---|---|

|          |          |          |          |          |          |
|----------|----------|----------|----------|----------|----------|
| <b>0</b> | <b>0</b> | <b>0</b> | <b>0</b> | <b>0</b> | <b>0</b> |
|----------|----------|----------|----------|----------|----------|

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 7 | 0 | 2 | 3 | 0 | 2 |
|---|---|---|---|---|---|

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 2 | 2 | 2 | 2 | 1 | 1 |
|---|---|---|---|---|---|

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 4 | 3 | 3 | 0 | 0 | 2 |
|---|---|---|---|---|---|

```
Process 3 runs to completion!
Max matrix: Allocation matrix:
7 5 3          0 1 0
0 0 0          0 0 0
0 0 0          0 0 0
2 2 2          2 1 1
4 3 3          0 0 2
Process 4 runs to completion!
Max matrix: Allocation matrix:
7 5 3          0 1 0
0 0 0          0 0 0
0 0 0          0 0 0
0 0 0          0 0 0
4 3 3          0 0 2
Process 1 runs to completion!
Max matrix: Allocation matrix:
0 0 0          0 0 0
0 0 0          0 0 0
0 0 0          0 0 0
0 0 0          0 0 0
4 3 3          0 0 2
Process 5 runs to completion!
The system is in a safe state!!
Safe Sequence: < 2 3 4 1 5 >
```



**9. Design, develop and implement a C/C++/Java program to implement page replacement algorithms LRU and FIFO. Assume suitable input required to demonstrate the results.**

```
#include<stdio.h>
#include<stdlib.h>
void FIFO(char [ ],char [ ],int,int);
void lru(char [ ],char [ ],int,int);
void opt(char [ ],char [ ],int,int);
int main()
{
    int ch,YN=1,i,l,f;
    char F[10],s[25];
    printf("\n\n\tEnter the no of
    empty frames: "); scanf("%d",&f);
    printf("\n\n\tEnter the length of the string: ");
    scanf("%d",&l);
    printf("\n\n\tEnter the string: ");
    scanf("%s",s);
    for(i=0;i<f;i++)
        F[i]=-1;
    do
    {
        printf("\n\n\t***** MENU *****");
        printf("\n\n\t1:FIFO\n\n\t2:LRU \n\n\t4:EXIT");
        printf("\n\n\tEnter your choice: ");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:
                for(i=0;i<f;i++)
                {
                    F[i]=-1;
                }
                FIFO(s,F,l,f);
                break;
            case 2:
                for(i=0;i<f;i++)
                {
                    F[i]=-1;
                }
                lru(s,F,l,f);
                break;
            case 4:
                exit(0);
        }
        printf("\n\n\tDo u want to continue IF YES PRESS 1\n\n\tIF NO
        PRESS 0 : "); scanf("%d",&YN);
    }while(YN==1);return(0);
```

```
}
//FIFO
void FIFO(char s[],char F[],int l,int f)
{
    int i,j=0,k,flag=0,cnt=0;
    printf("\n\tPAGE\t FRAMES\t FAULTS");
    for(i=0;i<l;i++)
    {
        for(k=0;k<f;k++)
        {
            if(F[k]==s[i])
            flag=1;
        }
        if(flag==0)
        {
            printf("\n\t%c\t",s[i]);
            F[j]=s[i];
            j++;
            for(k=0;k<f;k++)
            {
                printf(" %c",F[k]);
            }
            printf("\tPage-fault%d",cnt);
            cnt++;
        }
        else
        {
            flag=0;
            printf("\n\t%c\t",s[i]);
            for(k=0;k<f;k++)
            {
                printf(" %c",F[k]);
            }
            printf("\tNo page-fault");
        }
        if(j==f)
        j=0;
    }
}
//LRU
void lru(char s[],char F[],int l,int f)
{
    int i,j=0,k,m,flag=0,cnt=0,top=0;
    printf("\n\tPAGE\t FRAMES\t FAULTS");
    for(i=0;i<l;i++)
    {
        for(k=0;k<f;k++)
        {
```

```
if (F[k]==s[i])
{
flag=1;
break;
}
}
printf("\n\t%c\t",s[i]);
if(j!=f && flag!=1)
{
F[top]=s[i];
j++;
if(j!=f)
top++;
}
else
{
if(flag!=1)
{
for(k=0;k<top;k++)
{
F[k]=F[k+1];
}
F[top]=s[i];
}
if(flag==1)
{
for(m=k;m<top;m++)
{
F[m]=F[m+1];
}
F[top]=s[i];
}
}
for(k=0;k<f;k++)
{
printf(" %c",F[k]);
}
if(flag==0)
{
printf("\tPage-fault%d",cnt);
cnt++;
}
else
printf("\tNo page fault");
flag=0;
}
```

**Output:**

```
Enter the no of empty frames: 3
Enter the length of the string: 5
Enter the string: hello
***** MENU *****
1:FIFO
2:LRU
4:EXIT
Enter your choice: 1
PAGE FRAMES FAULTS
H h Page-fault 0
E h e Page-fault 1
L h e l Page-fault 2
L h e l No page-fault
O o e l Page-fault 3
Do u want to continue IF YES PRESS 1
IF NO PRESS 0 : 1
***** MENU *****
1:FIFO
2:LRU
4:EXIT
Enter your choice: 2
PAGE FRAMES FAULTS
H h Page-fault 0
E h e Page-fault 1
L h e l Page-fault 2
L h e l No page fault
O e l o Page-fault 3
Do u want to continue IF YES PRESS 1
IF NO PRESS 0 : 1
***** MENU *****
1:FIFO
2:LRU
4:EXIT
Enter your choice: 4
```