

COMPILER DESIGN

LABORATORY MANUAL

**B.TECH
(III YEAR – I SEM)
(2016-17)**

**Department of
Computer Science and Engineering**



**MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY
(Autonomous Institution – UGC, Govt. of India)**

Recognized under 2(f) and 12 (B) of UGC ACT 1956

Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2008 Certified)
Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, India



MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous Institution – UGC, Govt. of India)

Recognized under 2(f) and 12 (B) of UGC ACT 1956

Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2008 Certified)
Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, India

Department of Computer Science and Engineering

Vision

- To acknowledge quality education and instill high patterns of discipline making the students technologically superior and ethically strong which involves the improvement in the quality of life in human race.

Mission

- To achieve and impart holistic technical education using the best of infrastructure, outstanding technical and teaching expertise to establish the students into competent and confident engineers.
- Evolving the center of excellence through creative and innovative teaching learning practices for promoting academic achievement to produce internationally accepted competitive and world class professionals.



MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous Institution – UGC, Govt. of India)

Recognized under 2(f) and 12 (B) of UGC ACT 1956

Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2008 Certified)
Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, India

Department of Computer Science and Engineering

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

PEO1 – ANALYTICAL SKILLS

1. To facilitate the graduates with the ability to visualize, gather information, articulate, analyze, solve complex problems, and make decisions. These are essential to address the challenges of complex and computation intensive problems increasing their productivity.

PEO2 – TECHNICAL SKILLS

2. To facilitate the graduates with the technical skills that prepare them for immediate employment and pursue certification providing a deeper understanding of the technology in advanced areas of computer science and related fields, thus encouraging to pursue higher education and research based on their interest.

PEO3 – SOFT SKILLS

3. To facilitate the graduates with the soft skills that include fulfilling the mission, setting goals, showing self-confidence by communicating effectively, having a positive attitude, get involved in team-work, being a leader, managing their career and their life.

PEO4 – PROFESSIONAL ETHICS

To facilitate the graduates with the knowledge of professional and ethical responsibilities by paying attention to grooming, being conservative with style, following dress codes, safety codes, and adapting themselves to technological advancements.



MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous Institution – UGC, Govt. of India)

Recognized under 2(f) and 12 (B) of UGC ACT 1956

Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2008 Certified)
Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, India

Department of Computer Science and Engineering

PROGRAM SPECIFIC OUTCOMES (PSOs)

After the completion of the course, B. Tech Computer Science and Engineering, the graduates will have the following Program Specific Outcomes:

1. **Fundamentals and critical knowledge of the Computer System:-** Able to Understand the working principles of the computer System and its components , Apply the knowledge to build, asses, and analyze the software and hardware aspects of it .
2. **The comprehensive and Applicative knowledge of Software Development:** Comprehensive skills of Programming Languages, Software process models, methodologies, and able to plan, develop, test, analyze, and manage the software and hardware intensive systems in heterogeneous platforms individually or working in teams.
3. **Applications of Computing Domain & Research:** Able to use the professional, managerial, interdisciplinary skill set, and domain specific tools in development processes, identify the research gaps, and provide innovative solutions to them.

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 multi disciplinary 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.

TABLE OF CONTENTS

Sl. No	Description of the item/title	Page No.	Remarks
1.	Importance/Rationale behind the CD Lab	7	
2.	General Laboratory Instructions	8	
3.	Objectives & Outcomes	9	
4.	Software / Hardware Requirements	9	
5.	Case Study: Description of the Syntax of the source Language(mini language) for which the compiler components are designed	10	
6.	Design a lexical analyzer for the given language. The lexical analyzer should ignore redundant spaces, tabs and new lines, comments etc.	13	
7.	Implement the lexical analyzer using JLex, flex or other lexical analyzer generating tools.	17	
8.	Design Predictive Parser for the given language	19	
9.	Design a LALR bottom up parser for the given language	24	
10	Convert the BNF rules into Yacc form and write code to generate abstract syntax tree.	26	
11	A program to generate machine code from the abstract syntax tree generated by the parser.	33	
Additional Tasks / Programs			
12	Implementation of Finite State machines DFA, NFAs etc.		
13	Write a lex program to find out total number of vowels, and consonants from the given input sting.		
14	Computation of Leading & Trailing Sets		
15	Recursive Descent, SLR Parsers		
16	Intermediate Code Generation		

Importance of Compiler Design Lab

Every software has to be translated to its equivalent machine instruction form so that it will be executed by the underlying machine. There are many different types of system softwares that help during this translation process. Compiler is one such an important **System Software** that converts High level language programs to its equivalent machine (low level) language. It is impossible to learn and use machine language in software development for the users. Therefore we use high level computer languages to write programs and then convert such programs into machine understandable form with the help of mediator softwares such as compilers, interpreters, assemblers etc. Thus compiler bridges the gap between the user and the machine, i.e., computer.

It's a very complicated piece of software which took 18 man years to build first compiler. To build this software we must understand the principles, tools, and techniques used in its working. The compiler goes through the following sequence of steps called phases of a compiler.

- 1) Lexical Analysis
- 2) Syntax Analysis
- 3) Semantic Analysis
- 4) Intermediate Code Generation
- 5) Code Optimization
- 6) Code Generation.

In performing its role, compiler also uses the services of two essential components namely, Symbol Table and Error Handler

The objective of the Compiler Design Laboratory is to understand and implement the principles, techniques, and also available tools used in compiler construction process. This will enable the students to work in the development phase of new computer languages in industry.



MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

Maisammaguda, Dhulapally Post, Via Hakimpet, Secunderabad – 500100

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GENERAL LABORATORY INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before (to the starting time), those who come after 5 minutes will not be allowed into the lab.
2. Plan your task properly much before to the commencement, come prepared to the lab with the synopsis / program / experiment details.
3. Student should enter into the laboratory with:
 - a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.
 - b. Laboratory Record updated up to the last session experiments and other utensils (if any) needed in the lab.
 - c. Proper Dress code and Identity card.
4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.
5. Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.
6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.
7. Computer labs are established with sophisticated and high end branded systems, which should be utilized properly.
8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severe punishment.
9. Students must take the permission of the faculty in case of any urgency to go out ; if anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.
10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

Head of the Department

Principal



OBJECTIVES AND OUTCOMES

OBJECTIVES:

- To provide an Understanding of the language translation peculiarities by designing complete translator for mini language.

OUTCOMES:

- By this laboratory, students will understand the practical approach of how a compiler works.
- This will enable him to work in the development phase of new computer languages in industry

RECOMMENDED SYSTEM / SOFTWARE REQUIREMENTS:

1. Intel based desktop PC with minimum of 166MHz or faster processor with at least 64 MB RAM and 100 MB free disk space.
2. C ++ Compiler and JDK kit, Lex or Flex and YACC tools (Unix/Linux utilities)

USEFUL TEXT BOOKS / REFERECES / WEBSITES :

1. Modern compiler implementation in C, Andrew w.Appel, Revised Edn, Cambridge University Press
2. Principles of Compiler Design. – A.V Aho, J.D Ullman ; Pearson Education.
3. **lex&yacc** , -John R Levine, Tony Mason, Doug Brown; O'reilly.
4. **Compiler Construction**, - LOUDEN, Thomson.
5. Engineering a compiler – Cooper& Linda, Elsevier
6. Modern Compiler Design – Dick Grune, Henry E.Bal, Caryl TH Jacobs, Wiley Dreatech

SOURCE LANGUAGE (A Case Study)

Consider the following mini language, a simple procedural High Level Language, operating on integer data with a syntax looking vaguely like a simple C crossed with Pascal. The syntax of the language is defined by the following BNF grammar:

```
<program> ::= <block>
<block> ::= { <variable definition> <slist> }
           | { <slist> }
<variable definition> ::= int <vardeflist> ;
<vardeflist> ::= <vardec> | <vardec>, <vardeflist>
<vardec> ::= <identifier> | <identifier> [<constant>]
<slist> ::= <statement> | <statement> ; <slist>
<statement> ::= <assignment> | <ifstatement> | <whilestatement> | <block>
               | <printstatement> | <empty>
<assignment> ::= < identifier> = <expression>
               | < identifier> [<expression>] = [<expression>]
<ifstatement> ::= if <bexpression> then <slist> else <slist> endif
               | if <bexpression> then <slist> endif
<whilestatement> ::= while <bexpression> do <slist> enddo
<printstatement> ::= print{ <expression> }
<expression> ::= <expression> <addingop> <term> | <term> | <addingop> <term>
<bexpression> ::= <expression> <relop> <expression>
<relop> ::= < | <= | = | >= | > | !=
<addingop> ::= + | -
<term> ::= <term> <multop> <factor> | <factor>
<multop> ::= * | /
<factor> ::= <constant> | <identifier> | <identifier> [<expression>]
           | (<expression>)
           <constant> ::= <digit> | <digit> <constant>
           <identifier> ::= <identifier> <letterordigit> | <letter>
           <letterordigit> ::= a|b|c|...|y|z
           <digit> ::= 0|1|2|3|...|8|9
           <empty> ::= has the obvious meaning
```

Comments : zero or more characters enclosed between the standard C/Java style comment brackets /*...*/. The language has the rudimentary support for 1-Dimensional arrays. Ex: int a[3] declares a as an array of 3 elements, referenced as a[0],a[1],a[2].

Sample Program written in this language is :

```
{
  int a[3],t1,t2;
  t1=2;
  a[0]=1; a[1]=2; a[t1]=3;
  t2= -(a[2]+t1*6) / a[2]-t1);
  if t2>5 then
    print(t2);
  else
```

```

{
  int t3;
  t3=99;
  t2=25;
  print(-11+t2*t3); /* this is not a comment on two lines */
}
endif
}

```

1. Design a Lexical analyzer for the above language. The lexical analyzer should ignore redundant spaces, tabs and newlines. It should also ignore comments. Although the syntax specification states that identifiers can be arbitrarily long, you may restrict the length to some reasonable value.
2. Implement the lexical analyzer using JLex, flex, flex or lex or other lexical analyzer generating tools.
3. Design Predictive parser for the given language
4. Design LALR bottom up parser for the above language.
5. Convert the BNF rules into Yacc form and write code to generate abstract syntax tree.
6. Write program to generate machine code from the abstract syntax tree generated by the parser. The following instruction set may be considered as target code.

The following is a simple register-based machine, supporting a total of 17 instructions. It has three distinct internal storage areas. The first is the set of 8 registers, used by the individual instructions as detailed below, the second is an area used for the storage of variables and the third is an area used for the storage of program. The instructions can be preceded by a label. This consists of an integer in the range 1 to 9999 and the label is followed by a colon to separate it from the rest of the instruction. The numerical label can be used as the argument to a jump instruction, as detailed below.

In the description of the individual instructions below, instruction argument types are specified as follows:

R specifies a register in the form R0, R1, R2, R3, R4, R5, R6 or R7 (or r0, r1, etc).

L specifies a numerical label (in the range 1 to 9999).

V specifies a "variable location" (a variable number, or a variable location pointed to by a register - see below).

A specifies a constant value, a variable location, a register or a variable location pointed to by a register (an indirect address). Constant values are specified as an integer value, optionally preceded by a minus sign, preceded by a # symbol. An indirect address is specified by an @ followed by a register.

So, for example an A-type argument could have the form 4 (variable number 4), #4 (the constant value 4), r4 (register 4) or @r4 (the contents of register 4 identifies the variable location to be accessed).

The instruction set is defined as follows:

LOAD A, R

loads the integer value specified by A into register R.

STORE R, V

stores the value in register R to variable V.

OUT R

outputs the value in register R.

NEG R

negates the value in register R.

ADD A, R

adds the value specified by A to register R, leaving the result in register R.

SUB A, R

subtracts the value specified by A from register R, leaving the result in register R.

MUL A, R

multiplies the value specified by A by register R, leaving the result in register R.

DIV A, R

divides register R by the value specified by A, leaving the result in register R.

JMP L

causes an unconditional jump to the instruction with the label L.

JEQ R, L

jumps to the instruction with the label L if the value in register R is zero.

JNE R, L

jumps to the instruction with the label L if the value in register R is not zero.

JGE R, L

jumps to the instruction with the label L if the value in register R is greater than or equal to zero.

JGT R, L

jumps to the instruction with the label L if the value in register R is greater than zero.

JLE R, L

jumps to the instruction with the label L if the value in register R is less than or equal to zero.

JLT R, L

jumps to the instruction with the label L if the value in register R is less than zero.

NOP

is an instruction with no effect. It can be tagged by a label.

STOP

stops execution of the machine. All programs should terminate by executing a STOP instruction.

1. Problem Statement: Design a Lexical analyzer. The lexical analyzer should ignore redundant spaces and tabs and new lines. It should also ignore comments. Although the syntax specification says those identifiers can be arbitrarily long, you may restrict the length to some reasonable Value.

AIM: Write a C/C++ program to implement the design of a Lexical analyzer to recognize the tokens defined by the given grammar.

ALGORITHM / PROCEDURE:

We make use of the following two functions in the process.

look up() – it takes string as argument and checks its presence in the symbol table. If the string is found then returns the address else it returns NULL.

insert() – it takes string as its argument and the same is inserted into the symbol table and the corresponding address is returned.

1. Start
2. Declare an array of characters, an input file to store the input;
3. Read the character from the input file and put it into character type of variable, say 'c'.
4. If 'c' is blank then do nothing.
5. If 'c' is new line character line=line+1.
6. If 'c' is digit, set token Val, the value assigned for a digit and return the 'NUMBER'.
7. If 'c' is proper token then assign the token value.
8. Print the complete table with
Token entered by the user,
Associated token value.
9. Stop

PROGRAM / SOURCE CODE :

```
#include<string.h>
#include<ctype.h>
#include<stdio.h>
void keyword(char str[10])
{
    if(strcmp("for",str)==0 || strcmp("while",str)==0 || strcmp("do",str)==0 || strcmp("int",str
    )==0 || strcmp("float",str)==0 || strcmp("char",str)==0 || strcmp("double",str)==0 || strcmp("st
    atic",str)==0 || strcmp("switch",str)==0 || strcmp("case",str)==0)
        printf("\n%s is a keyword",str);
    else
        printf("\n%s is an identifier",str);
}
main()
{
```

```

FILE *f1,*f2,*f3;
char c, str[10], st1[10];
int num[100], lineno=0, tokenvalue=0,i=0,j=0,k=0;
printf("\n Enter the c program : ");/*gets(st1);*/
f1=fopen("input","w");
while((c=getchar())!=EOF)
putc(c,f1);
fclose(f1);
f1=fopen("input","r");
f2=fopen("identifier","w");
f3=fopen("specialchar","w");
while((c=getc(f1))!=EOF)
{
    if(isdigit(c))
    {
        tokenvalue=c-'0';
        c=getc(f1);
        while(isdigit(c))
        {
            tokenvalue*=10+c-'0';
            c=getc(f1);
        }
        num[i++]=tokenvalue;
        ungetc(c,f1);
    }
    else
    if(isalpha(c))
    {
        putc(c,f2);
        c=getc(f1);
        while(isdigit(c) || isalpha(c) || c=='_' || c=='$')
        {
            putc(c,f2);
            c=getc(f1);
        }
        putc(' ',f2);
    }
}

```

```

        ungetc(c,f1);
    }
    else
        if(c==' ' | c=='\t')
            printf(" ");
        else
            if(c=='\n')
                lineno++;
        else
            putc(c,f3);
    }
    fclose(f2);
    fclose(f3);
    fclose(f1);
    printf("\n The no's in the program are :");
    for(j=0; j<i; j++)
        printf("%d", num[j]);
    printf("\n");
    f2=fopen("identifier", "r");
    k=0;
    printf("The keywords and identifiers are:");
    while((c=getc(f2))!=EOF)
    {
        if(c!=' ')
            str[k++]=c;
        else
        {
            str[k]='\0';
            keyword(str);
            k=0;
        }
    }
    fclose(f2);
    f3=fopen("specialchar","r");
    printf("\n Special characters are : ");
    while((c=getc(f3))!=EOF)

```

```
        printf("%c",c);  
    printf("\n");  
    fclose(f3);  
    printf("Total no. of lines are:%d", lineno);  
}
```

Output :

Enter the C program: a+b*c

Ctrl-D

The no's in the program are:

The keywords and identifiers are:

a is an identifier and terminal

b is an identifier and terminal

c is an identifier and terminal

Special characters are:

+ *

Total no. of lines are: 1

[Viva Questions]

1. What is lexical analyzer?
2. Which compiler is used for lexical analysis?
3. What is the output of Lexical analyzer?
4. Which Finite state machines are used in lexical analyzer design?
5. What is the role of regular expressions, grammars in Lexical Analyzer?

2. Problem Statement: Implement the lexical analyzer using JLex, flex or other lexical Analyzer generating tools.

AIM: To Implement the lexical analyzer using JLex, flex or lex other lexical analyzer generating Tools.

ALGORITHM / PROCEDURE:

Input : LEX specification files for the token

Output : Produces the source code for the Lexical Analyzer with the name lex.yy.c and displays the tokens from an input file.

1. Start
2. Open a file in text editor
3. Create a Lex specifications file to accept keywords, identifiers, constants, operators and relational operators in the following format.
 - a) %{
 Definition of constant /header files
 %}
 - b) Regular Expressions
 %%
 Transition rules
 %%
 - c) Auxiliary Procedure (main() function)
4. Save file with .l extension e.g. **mylex.l**
5. Call lex tool on the terminal e.g. [root@localhost]# lex mylex.l. This lex tool will convert
 “.l” file into “.c” language code file i.e., **lex.yy.c**
6. Compile the file lex.yy.c using C / C++ compiler. e.g. **gcc lex.yy.c**. After compilation the file lex.yy.c, the output file is in **a.out**
7. Run the file a.out giving an input(text/file) e.g. **./a.out**.
8. Upon processing, the sequence of tokens will be displayed as output.
9. Stop

LEX SPECIFICATION PROGRAM / SOURCE CODE (lexprog.l) :

```
// ***** LEX Tool program to identify C Tokens *****//
DIGIT      [0-9]
LETTER     [A-Z a-z]
DELIM      [\t\n]
WS         {DELIM}+ID{(LETTER)[LETTER/DIGIT]}+INTEGER{DIGIT}+
%%
{WS}       { printf("\n special characters\n"); }
{ID}       { printf("\n Identifiers\n"); }
{DIGIT}    { printf("\n Intgers\n"); }
If         { printf("\n Keywords\n"); }
```

```

else                { printf("\n keywords\n"); }
">"               { printf("\n Logical Operators\n"); }
"<"               { printf("\n Logical Operators\n"); }
"<="             { printf("\n Logical Operators\n"); }
">="             { printf("\n Logical Operators\n"); }
"="                { printf("\n Logical Operators\n"); }
"!="              { printf("\n Logical Operators\n"); }
"&&"             { printf("\n Relational \n"); }
"||"              { printf("\n Relational Operatos\n"); }
"!"               { printf("\n Relational Operators\n"); }
"+"               { printf("\n Arthmetic Operator\n"); }
"_"               { printf("\n Arthmetic Operator\n"); }
"*"               { printf("\n Arthmetic Operator\n"); }
"/"               { printf("\n Arthmetic Operator\n"); }
"%"               { printf("\n Arthmetic Operator\n"); }
%%
main()
{
    yylex();
}

```

OUTPUT

```

[root@localhost]# lex lexprog.l
[root@localhost]# cc lex.yy.c
[root@localhost]# ./a.out lexprog

```

TEST CASES:

INPUT	OUTPUT
If	Keyword
%	Arithmetic Operator
>=	Relational Operator
&&	Logical Operator

[Viva Questions]

1. What is are the functions of a Scanner?
2. What is Token?
3. What is lexeme, Pattern?
4. What is purpose of Lex?
5. What are the other tools used in Lexical Analysis?

3. Problem Statement : Design a Predictive Parser for the following grammar

G: { E-> TE' , E' -> +TE' | 0, T-> FT' , T'-> *FT'|0 , F-> (E) | id }

AIM: To write a 'C' Program to implement for the Predictive Parser (Non Recursive Descent parser) for the given grammar ,

Given the parse Table:

	id	+	*	()	\$
E	E-> TE'			E-> TE'		
E'		E' -> +TE'			E'->0	E'->0
T	T-> FT'			T-> FT'		
T'		T'->0	T'-> *FT'		T'->0	T'->0
F	F-> id			F->(E)		

ALGORITHM / PROCEDURE :

Input : string w\$, Predictive Parsing table M

Output : A Left Most Derivation of the input string if it is valid , error otherwise.

- Step1: Start
- Step2: Declare a character array w[10] and Z as an array
- Step3: Enter the string with \$ at the end
- Step4: if (A(w[z])) then increment z and check for (B(w[z])) and if satisfies increment z and check for 'd' if d is present then increment and check for (D(w[z]))
- Step5: if step 4 is satisfied then the string is accepted
Else string is not
- Step 6: Exit

SOURCE CODE :

```
// **IMPLEMENTATION OF PREDICTIVE / NON-RECURSIVE DESCENT PARSING ****//
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
char ch;
#define id 0
#define CONST 1
#define mulop 2
#define addop 3
#define op 4
#define cp 5
#define err 6
```

```

#define col 7
#define size 50
int token;
char lexbuff[size];
int lookahead=0;
int main()
{
    clrscr();
    printf(" Enter the string :");
    gets(lexbuff);
    parser();
    return 0;
}
parser()
{
    if(E())
        printf("valid string");
    else
        printf("invalid string");
    getch();
    return 0;
}
E()
{
    if(T())
    {
        if(EPRIME())
            return 1;
        else
            return 0;
    }
    else
        return 0;
}
T()
{
    if(F())
    {
        if(TPRIME())
            return 1;
        else
            return 0;
    }
    else
        return 0;
}
EPRIME()
{
    token=lexer();
    if(token==addop)
    {
        lookahead++;
        if(T())

```

```

    {
        if(EPRIME())
            return 1;
        else
            return 0;
    }
    else
        return 0;
}
else
    return 1;
}
TPRIME()
{
    token=lexer();
    if(token==mulop)
    {
        lookahead++;
        if(F())
        {
            if(TPRIME())
                return 1;
            else
                return 0;
        }
        else return 0;
    }
    else return 1;
}
F()
{
    token=lexer();
    if(token==id)
        return 1;
    else
    {
        if(token==4)
        {
            if(E())
            {
                if(token==5)
                    return 1;
                else return 0;
            }
            else return 0;
        }
        else return 0;
    }
}
lexer()
{
    if(lexbuff[lookahead]!='\n')
    {

```

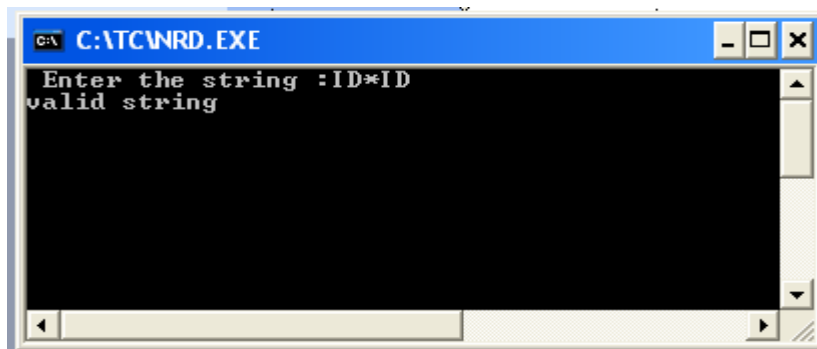
```

while(lexbuff[lookahead]=='\t')
lookahead++;
if(isalpha(lexbuff[lookahead]))
{
while(isalnum(lexbuff[lookahead]))
lookahead++;
return(id);
}
else
{
if(isdigit(lexbuff[lookahead]))
{
while(isdigit(lexbuff[lookahead]))
lookahead++;
return CONST;
}
else
{
if(lexbuff[lookahead]=='+')
{
return(addop);
}
else
{
if(lexbuff[lookahead]=='*')
{
return(mulop);
}
else
{
if(lexbuff[lookahead]=='(')
{
lookahead++;
return(op);
}
else
{
if(lexbuff[lookahead]==')')
{
return(op);
}
else
{
return(err);
}
}
}
}
}
}
else
return (col);

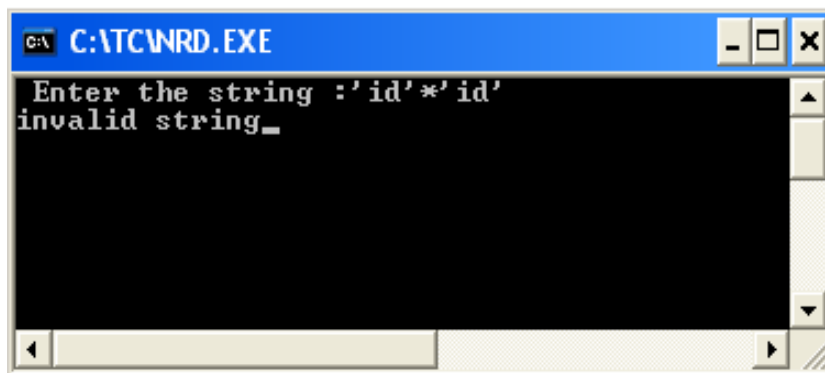
```

}

OUTPUT :



```
C:\VTC\NRD.EXE
Enter the string :ID*ID
valid string
```



```
C:\VTC\NRD.EXE
Enter the string :\'id\'*\'id\'
invalid string_
```

Viva Questions :

1. What is a parser and state the Role of it?
2. Types of parsers? Examples to each
3. What are the Tools available for implementation?
4. How do you calculate FIRST(),FOLLOW() sets used in Parsing Table construction?

4. Problem Statement: Design a LALR Bottom Up Parser for the given grammar

Aim: To Design and implement an LALR bottom up Parser for checking the syntax of the Statements in the given language.

ALGORITHM/PROCEDURE:

Source Code : LALR Bottom Up Parser

```
<parser.l>
%{
#include<stdio.h>
#include "y.tab.h"
%}
%%
[0-9]+ {yy1val.dval=atof(yytext);
return DIGIT;
}
\n|. return yytext[0];
%%
<parser.y>
%{
/*This YACC specification file generates the LALR parser for the program
considered in experiment 4.*/
#include<stdio.h>
%}
%union
{
double dval;
}
%token <dval> DIGIT
%type <dval> expr
%type <dval> term
%type <dval> factor
%%
line: expr '\n' {
;
printf("%g\n", $1);
}
expr: expr '+' term {$$=$1 + $3 ;}
| term
;
term: term '*' factor {$$=$1 * $3 ;}
| factor
;
;
```



```
factor: '(' expr ')' {$$=$2 ;}
| DIGIT
;
%%
int main()
{
  yyparse();
}
yyerror(char *s)
{
  printf("%s",s);
}
```

Output:

```
$lex parser.l
$yacc -d parser.y
$cc lex.yy.c y.tab.c -ll -lm
$./a.out
2+3
5.0000
```

Viva Questions?

1. What is yacc? Are there any other tools available for parser generation?
2. How do you use it?
3. Structure of parser specification program
- 4.

5. Problem statement: Convert The BNF rules into Yacc form and write code to generate abstract syntax tree.

Aim: To Implement the process of conversion from BNF rules to Yacc form and generate Abstract Syntax Tree.

ALGORITHM/PROCEDURE

```
<int.l>
%{
#include"y.tab.h"
#include<stdio.h>
#include<string.h>
int LineNo=1;
% }
identifier [a-zA-Z][_a-zA-Z0-9]*
number [0-9]+|([0-9]*\.[0-9]+)
%%
main\(\) return MAIN;
if return IF;
else return ELSE;
while return WHILE;
int |
char |
float return TYPE;
{identifier} {strcpy(yylval.var,yytext);
return VAR;}
{number} {strcpy(yylval.var,yytext);
return NUM;}
\< |
\> |
\>= |
\<= |
== {strcpy(yylval.var,yytext);
return RELOP;}
[ \t ] ;
\n LineNo++;
. return yytext[0];
%%
<int.y>
%{
#include<string.h>
#include<stdio.h>
struct quad
```

```

{
    char op[5];
    char arg1[10];
    char arg2[10];
    char result[10];
}QUAD[30];
struct stack
{
    int items[100];
    int top;
}stk;
int Index=0,tIndex=0,StNo,Ind,tInd;
extern int LineNo;
%}
%union
{
    char var[10];
}
%token <var> NUM VAR RELOP
%token MAIN IF ELSE WHILE TYPE
%type <var> EXPR ASSIGNMENT CONDITION IFST ELSEST WHILELOOP
%left '-' '+'
%left '*' '/'
%%
PROGRAM : MAIN BLOCK
;
BLOCK: '{' CODE '}'
;
CODE: BLOCK
| STATEMENT CODE
| STATEMENT
;
STATEMENT: DESCT ';'
| ASSIGNMENT ';'
| CONDST
| WHILEST
;
DESCT: TYPE VARLIST
;
VARLIST: VAR ',' VARLIST
| VAR
;
ASSIGNMENT: VAR '=' EXPR{
strcpy(QUAD[Index].op,"=");

```

```

strcpy(QUAD[Index].arg1,$3);
strcpy(QUAD[Index].arg2,"");
strcpy(QUAD[Index].result,$1);
strcpy($$,QUAD[Index++].result);
}
;
EXPR: EXPR '+' EXPR {AddQuadruple("+",$1,$3,$$);}
| EXPR '-' EXPR {AddQuadruple("-", $1,$3,$$);}
| EXPR '*' EXPR { AddQuadruple("*",$1,$3,$$);}
| EXPR '/' EXPR { AddQuadruple("/", $1,$3,$$);}
| '-' EXPR { AddQuadruple("UMIN",$2,"",$$);}
| '(' EXPR ')' {strcpy($$, $2);}
| VAR
| NUM
;
CONDST: IFST{
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
}
| IFST ELSEST
;
IFST: IF '(' CONDITION ')' {
strcpy(QUAD[Index].op,"==");
strcpy(QUAD[Index].arg1,$3);
strcpy(QUAD[Index].arg2,"FALSE");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
}
BLOCK {
strcpy(QUAD[Index].op,"GOTO");
strcpy(QUAD[Index].arg1,"");
strcpy(QUAD[Index].arg2,"");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
};
ELSEST: ELSE{
tInd=pop();
Ind=pop();
push(tInd);
sprintf(QUAD[Ind].result,"%d",Index);

```

```

}
BLOCK{
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
};
CONDITION: VAR RELOP VAR {AddQuadruple($2,$1,$3,$$);
StNo=Index-1;
}
| VAR
| NUM
;
WHILEST: WHILELOOP{
Ind=pop();
sprintf(QUAD[Ind].result,"%d",StNo);
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
}
;
WHILELOOP: WHILE '(' CONDITION ')' {
strcpy(QUAD[Index].op,"=");
strcpy(QUAD[Index].arg1,$3);
strcpy(QUAD[Index].arg2,"FALSE");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
}
BLOCK {
strcpy(QUAD[Index].op,"GOTO");
strcpy(QUAD[Index].arg1,"");
strcpy(QUAD[Index].arg2,"");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
}
;
%%
extern FILE *yyin;
int main(int argc,char *argv[])
{
FILE *fp;
int i;
if(argc>1)
{
fp=fopen(argv[1],"r");

```

```

if(!fp)
{
printf("\n File not found");
exit(0);
}
yyin=fp;
}
yyparse();
printf("\n\n\t\t -----""\n\t\t Pos Operator Arg1 Arg2 Result" "\n\t\t
-----");
for(i=0;i<Index;i++)
{
printf("\n\t\t %d\t %s\t %s\t %s\t
%s",i,QUAD[i].op,QUAD[i].arg1,QUAD[i].arg2,QUAD[i].result);
}
printf("\n\t\t -----");
printf("\n\n");
return 0;
}
void push(int data)
{
stk.top++;
if(stk.top==100)
{
printf("\n Stack overflow\n");
exit(0);
}
stk.items[stk.top]=data;
}
int pop()
{
int data;
if(stk.top==-1)
{
printf("\n Stack underflow\n");
exit(0);
}
data=stk.items[stk.top--];
return data;
}
void AddQuadruple(char op[5],char arg1[10],char arg2[10],char result[10])
{
strcpy(QUAD[Index].op,op);
strcpy(QUAD[Index].arg1,arg1);

```

```

strcpy(QUAD[Index].arg2,arg2);
sprintf(QUAD[Index].result,"t%d",tIndex++);
strcpy(result,QUAD[Index++].result);
}
yyerror()
{
printf("\n Error on line no:%d",LineNo);
}
Input:
$vi test.c
main()
{
int a,b,c;
if(a<b)
{
a=a+b;
}
while(a<b)
{
a=a+b;
}
if(a<=b)
{
c=a-b;
}
else
{
c=a+b;
}
}

```

Output:

```

$ lex int.l
$ yacc -d int.y
$ gcc lex.yy.c y.tab.c -ll -lm
$ ./a.out test.c

```

OUTPUT:

Pos	Operator	Arg1	Arg2	Result
0	<	a	b	to
1	==	to	FALSE	5
2	+	a	b	t1
3	=	t1		a
4	GOTO			5
5	<	a	b	t2
6	==	t2	FALSE	10
7	+	a	b	t3
8	=	t3		a
9	GOTO			5
10	<=	a	b	t4
11	==	t4	FALSE	15
12	-	a	b	t5
13	=	t5		c
14	GOTO		17	
15	+	a	b	t3
16	=	t6		c

Viva Questions:

1. What is abstract syntax tree?
- 2.

6. Problem Statement: A program to generate machine code from the abstract syntax tree generated by the parser.

Aim: To write a C Program to Generate Machine Code from the Abstract Syntax Tree using the specified machine instruction formats.

ALGORITHM / PROCEDURE / SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int label[20];
int no=0;
int main()
{
FILE *fp1,*fp2;
char fname[10],op[10],ch;
char operand1[8],operand2[8],result[8];
int i=0,j=0;
printf("\n Enter filename of the intermediate code");
scanf("%s",&fname);
fp1=fopen(fname,"r");
fp2=fopen("target.txt","w");
if(fp1==NULL || fp2==NULL)
{
printf("\n Error opening the file");
exit(0);
}
while(!feof(fp1))
{
fprintf(fp2,"\n");
fscanf(fp1,"%s",op);
i++;
if(check_label(i))
fprintf(fp2,"\nlabel#%d",i);
if(strcmp(op,"print")==0)
{
fscanf(fp1,"%s",result);
fprintf(fp2,"\n\t OUT %s",result);
}
if(strcmp(op,"goto")==0)
{
fscanf(fp1,"%s %s",operand1,operand2);
fprintf(fp2,"\n\t JMP %s,label#%s",operand1,operand2);
label[no++]=atoi(operand2);
}
```

```

}
if(strcmp(op,"[]")==0) 18
{
fscanf(fp1,"%s %s %s",operand1,operand2,result);
fprintf(fp2,"\n\t STORE %s[%s],%s",operand1,operand2,result);
}
if(strcmp(op,"uminus")==0)
{
fscanf(fp1,"%s %s",operand1,result);
fprintf(fp2,"\n\t LOAD -%s,R1",operand1);
fprintf(fp2,"\n\t STORE R1,%s",result);
}s
witch(op[0])
{
case '*': fscanf(fp1,"%s %s %s",operand1,operand2,result);
fprintf(fp2,"\n\t LOAD",operand1);
fprintf(fp2,"\n\t LOAD %s,R1",operand2);
fprintf(fp2,"\n\t MUL R1,R0");
fprintf(fp2,"\n\t STORE R0,%s",result);
break;
case '+': fscanf(fp1,"%s %s %s",operand1,operand2,result);
fprintf(fp2,"\n\t LOAD %s,R0",operand1);
fprintf(fp2,"\n\t LOAD %s,R1",operand2);
fprintf(fp2,"\n\t ADD R1,R0");
fprintf(fp2,"\n\t STORE R0,%s",result);
break;
case '-': fscanf(fp1,"%s %s %s",operand1,operand2,result);
fprintf(fp2,"\n\t LOAD %s,R0",operand1);
fprintf(fp2,"\n\t LOAD %s,R1",operand2);
fprintf(fp2,"\n\t SUB R1,R0");
fprintf(fp2,"\n\t STORE R0,%s",result);
break;
case '/': fscanf(fp1,"%s %s %s",operand1,operand2,result);
fprintf(fp2,"\n\t LOAD %s,R0",operand1);
fprintf(fp2,"\n\t LOAD %s,R1",operand2);
fprintf(fp2,"\n\t DIV R1,R0");
fprintf(fp2,"\n\t STORE R0,%s",result);
break;
case '%': fscanf(fp1,"%s %s %s",operand1,operand2,result);
fprintf(fp2,"\n\t LOAD %s,R0",operand1);
fprintf(fp2,"\n\t LOAD %s,R1",operand2);
fprintf(fp2,"\n\t DIV R1,R0");
fprintf(fp2,"\n\t STORE R0,%s",result); 19

```

```

break;
case '=': fscanf(fp1,"%s %s",operand1,result);
fprintf(fp2,"\n \t STORE %s %s",operand1,result);
break;
case '>': j++;
fscanf(fp1,"%s %s %s",operand1,operand2,result);
fprintf(fp2,"\n \t LOAD %s,R0",operand1);
fprintf(fp2,"\n \t JGT %s,label#%s",operand2,result);
label[no++]=atoi(result);
break;
case '<': fscanf(fp1,"%s %s %s",operand1,operand2,result);
fprintf(fp2,"\n \t LOAD %s,R0",operand1); fprintf(fp2,"\n \t
JLT %s,label#%d",operand2,result);
label[no++]=atoi(result);
break;
}
}
fclose(fp2); fclose(fp1);
fp2=fopen("target.txt","r");
if(fp2==NULL)
{
printf("Error opening the file\n");
exit(0);
}
do
{
ch=fgetc(fp2);
printf("%c",ch);
}while(ch!=EOF);
fclose(fp1);
return 0;
}
int check_label(int k)
{
int i;
for(i=0;i<no;i++)
{
if(k==label[i])
return 1;
}
return 0; 20

```

```

}
Input:
$vi int.txt
=t1 2
[]=a 0 1
[]=a 1 2
[]=a 2 3
*t1 6 t2
+a[2] t2 t3
-a[2] t1 t2
/t3 t2 t2
uminus t2 t2
print t2
goto t2 t3
=t3 99
uminus 25 t2
*t2 t3 t3
uminus t1 t1
+t1 t3 t4
print t4

```

Output:

Enter filename of the intermediate code: int.txt

```

STORE t1,2
STORE a[0],1
STORE a[1],2
STORE a[2],3
LOAD t1,R0
LOAD 6,R1
ADD R1,R0
STORE R0,t3
LOAD a[2],R0
LOAD t2,R1
ADD R1,R0
STORE R0,t3
LOAD a[t2],R0
LOAD t1,R1
SUB R1,R0
STORE R0,t2
LOAD t3,R0
LOAD t2,R1 21

```

```
DIV R1,R0
STORE R0,t2
LOAD t2,R1
STORE R1,t2
LOAD t2,R0
JGT 5,label#11
Label#11: OUT t2
JMP t2,label#13
Label#13: STORE t3,99
LOAD 25,R1
STORE R1,t2
LOAD t2,R0
LOAD t3,R1
MUL R1,R0
STORE R0,t3
LOAD t1,R1
STORE R1,t1
LOAD t1,R0
LOAD t3,R1
ADD R1,R0
STORE R0,t4
OUT t4
```

Additional Programs/Tasks:

7. Problem Statement: Implement RECURSIVE DESCENT PARSER

AIM: To Implement the RECURSIVE DESCENT PARSER for the given grammar / language

ALGORITHM/ PROCEDURE:

Input : A string $w\$$, where w is a string of terminals

Output : w is accepted if it can be derived by the given grammar, else not accepted.

- Step1: Start
- Step2: declare $w[10]$ as char and Z as an array
- Step3: enter the string with $\$$ at the end
- Step4: if ($A(w[z])$) then increment z and check for ($B(w[z])$) and if satisfies increment z and check for 'd' if d is present then increment and check for ($D(w[z])$)
- Step5: if step 4 is satisfied then the string is accepted
Else string is not
- Step6: give for the grammar $A \rightarrow bc/ab$ in the loop $A(int\ k)$
- Step7S: Describe the grammar $b \rightarrow c/d$ in the loop $B(int\ k)$
- Step8: Similarly describe the grammar $D \rightarrow d/abcd$
- Step9: if steps7,8,9 are satisfied accordingly string is accepted
Else string is not accepted
- Step10: Stop

. Name of the Experiment: Design SLR

AIM: Design SLR bottom up parser for the above language

ALGORITHM

- SStep1: Start
- Step2: Initially the parser has s_0 on the stack where s_0 is the initial state and $w\$$ is in buffer
- Step3: Set ip point to the first symbol of $w\$$
- Step4: repeat forever, begin
- Step5: Let S be the state on top of the stack and a symbol pointed to by ip
- Step6: If action [S, a] = shift S then begin
Push S_1 on to the top of the stack
Advance ip to next input symbol
- Step7: Else if action [S, a], reduce $A \rightarrow B$ then begin
Pop $2 * |B|$ symbols of the stack
Let S_1 be the state now on the top of the stack
- Step8: Output the production $A \rightarrow B$
End
- Step9: else if action [S, a] = accepted, then return
Else

```

Error()
End
Step10: Stop

```

SOURCE CODE

```

// ***** IMPLEMENTATION OF LALR PARSING PROGRAM *****//

#include<stdio.h>
#include<conio.h>
#include<string.h>
#define MAX 50
void push(char item);
char pop(void);
int top=-1;
int call(char c);
char stack[MAX],input[10],str2[15],str1[8]="",c;
void prn(int j)
{
    int i;
    for(i=j;input[i]!='\0';i++)
        printf("%c",input[i]);
}
void prnstack(int top)
{
    int i;
    for(i=0;i<top;i++)
        printf("%c",stack[i]);
}
void main()
{
    char str1[6],*cmp="",c[8]="";
    int i=0,cn=0, k,j;
    FILE *ptr, *gptr;
    clrscr();
    printf("\n\n enter the expression : \n");
    scanf("%s",input);
    push('0');
    printf("\n\n\t STACK \t\t\t COMPARISION \t\t\t OUTPUT \n\n");
    do
    {
        printf("");
        prnstack(top);
        printf("\t\t\t");
        prn(i);
        if(strcmp(cmp,"1$")==0)
        {
            strcpy(str2,"accepted");
            printf("\n\nthe input is accepted");
            getch();
            exit(0);
        }
    }
}

```

```

else
{
    cmp[0]=stack[top];
    cmp[1]=input[i];
    cmp[2]='\0';
    if((ptr=fopen("d:\ltable.doc","r"))==NULL)
        printf("\n\n FILE CAN NOT BE OPEN");
    else
    {
        while(!feof(ptr))
        {
            fscanf(ptr, "%s%s",str1,str2);
            if(strcmp(str1,cmp)==0)
            {
                if(str2[0]=='s')
                {
                    push(input[i]);
                    push(str2[1]);
                    i++;
                    break;
                }
            }
            else if(str2[0]=='r')
            {
                cn=call(str2[1]);
                for(k=0;k<(cn*2);k++)
                    pop();
                c[0]=stack[top];
                push(str2[0]);
                c[1]=stack[top];
                c[2]='\0';
                if(strcmp(c,"0E")==0)
                    push('1');
                else if(strcmp(c,"0T")==0)
                    push('2');
                else if(strcmp(c,"0F")==0)
                    push('3');
                else if(strcmp(c,"0E")==0)
                    push('8');
                else if(strcmp(c,"0T")==0)
                    push('2');
                else if(strcmp(c,"0F")==0)
                    push('3');
                else if(strcmp(c,"0T")==0)
                    push('9');
                else if(strcmp(c,"0F")==0)
                    push('3');
                else if(strcmp(c,"0F")==0)
                    push('t');
            }
        }
        else if(strcmp(str2,"0")==0)
        {
            printf("\n\n the string is not accepted");
            break;
        }
    }
}

```



```

        }
    }
}

fclose(ptr);
}
printf("\t\t%s\t\t\n",cmp,str2);
}
while(input[i]!='\0');
getch();
}
int call(char c)
{
    int count =0;
    switch(c)
    {
        case 1: strcpy(str2,"E->E+T");
                count=3;
                break ;

        case 2: strcpy(str2,"E->T");
                count=1;
                break;

        case 3: strcpy(str2,"T->T*F");
                count=3;
                break;

        case 4: strcpy(str2,"T->F");
                count=1;
                break;

        case 5: strcpy(str2,"F->(E)");
                count=3;
                break;

        case 6: strcpy(str2,"F->id");
                count=1;
                break;

    }
    return count ;
}
void push(char item)
{
    if(top==MAX)
        printf("\n\n stack overflow");
    else
    {
        top=top+1;
        stack[top]=item;
    }
}
char pop(void)

```

```

{
    char item;
    if(top==1)
        printf("\n\n stack underlow");
    else
    {
        item=stack[top];
        top--;
    }
    return item;
}

```

We have to down load from drive “D:\\ lrtable.txt”

Lrtable.txt

SLR PARSER TABLE

States	Action						GOTO		
	Id	+	*	()	\$	E	T	F
0	S5			S4			1	2	3
1						accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6				S11			
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

--	--	--	--	--	--	--	--	--	--

OUTPUT:

Id*(id+id)\$

Grammar accepted

Aim: To calculate trailing for all the non- terminals of the given grammar

ALGORITHM :

1. Start
2. For each non terminal A and terminal a do $L(A,a) := \text{false}$;
3. For each production of the form $A \rightarrow a(\alpha)$ or $A \rightarrow B(\alpha)$ do $\text{INSTALL}(A,a)$
4. While STACK not empty repeat 5 and 6
5. Pop top pair from stack
6. For each production of the form $A \rightarrow B(\alpha)$ do
 $\text{INSTALL}(A,a)$
7. Stop

Algorithm For $\text{INSTALL}(A,a)$

1. Start
2. If $L[A,a]$ not present repeat step 3 and 4
3. Make $L(A,a) = \text{True}$
4. Push (A,a) onto stack
5. Stop

8) Implementation of Predictive Parser.

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
#include<stdlib.h>
#define SIZE 128
#define NONE -1
#define EOS '\0'
#define NUM 257
#define KEYWORD 258
#define ID 259
#define DONE 260
#define MAX 999
char lexemes[MAX];
char buffer[SIZE];
int lastchar=-1;
int lastentry=0;
int tokenval=DONE;
int lineno=1;
int lookahead;
struct entry
{
char *lexptr;
int token;
}symtable[100]; struct entry
keywords[]={{"if",KEYWORD,"else",KEYWORD,"for",KEYWORD,"int",KEYWORD,
"float",KEYWORD,"double",KEYWORD,"char",KEYWORD,"struct",KEYWORD,"ret
urn",KEYWORD,0,0};
void Error_Message(char *m)
{
fprintf(stderr,"line %d, %s \n",lineno,m);
exit(1);
}
int look_up(char s[ ])
{
int k; for(k=lastentry;k>0;k--)
if(strcmp(symtable[k].lexptr,s)==0)
return k;
return 0;
} 10
```

```

int insert(char s[ ],int tok)
{
int len;
len=strlen(s); if(lastentry+1>=MAX)
Error_Message("Symbpl table is full");
if(lastchar+len+1>=MAX)
Error_Message("Lexemes array is full");
lastentry=lastentry+1;
symtable[lastentry].token=tok;
symtable[lastentry].lexptr=&lexemes[lastchar+1];
lastchar=lastchar+len+1;
strcpy(symtable[lastentry].lexptr,s);
return lastentry;
}
/*void Initialize()
{
struct entry *ptr;
for(ptr=keywords;ptr->token;ptr+1)
insert(ptr->lexptr,ptr->token);
}*/
int lexer()
{
int t;
int val,i=0;
while(1)
{
t=getchar();
if(t==' '||t=='\t');
else if(t=='\n')
lineno=lineno+1;
else if(isdigit(t))
{
ungetc(t,stdin);
scanf("%d",&tokenval);
return NUM;
}
else if(isalpha(t))
{
while(isalnum(t))
{
buffer[i]=t; i++;

```

```

t=getchar();
i=i+1;
if(i>=SIZE)
Error_Message("Compiler error");
}
buffer[i]=EOS;if(t!=EOF)
ungetc(t,stdin);
val=look_up(buffer);
if(val==0)
val=insert(buffer,ID);
tokenval=val;
return symtable[val].token;
}
else if(t==EOF)
return DONE;
else
{
tokenval=NONE;
return t;
}
}
}
void Match(int t)
{
if(lookahead==t)
lookahead=lexer();
else
Error_Message("Syntax error");
}
void display(int t,int tval)
{
if(t=='+'||t=='-'||t=='*'||t=='/')
printf("\nArithmetic Operator: %c",t);
else if(t==NUM)
printf("\n Number: %d",tval);
else if(t==ID)
printf("\n Identifier: %s",symtable[tval].lexptr);
else
printf("\n Token %d tokenval %d",t,tokenval);
}
void F() 12

```

```

{
//void E();
switch(lookahead)
{
case '(' : Match('(');
E();
Match(')');
break;
case NUM : display(NUM,tokenval);
Match(NUM);
break;
case ID : display(ID,tokenval);
Match(ID);
break;
default : Error_Message("Syntax error");
}
}
void T()
{
int t;
F();
while(1)
{
switch(lookahead)
{
case '*' : t=lookahead;
Match(lookahead);
F();
display(t,NONE);
continue;
case '/' : t=lookahead;
Match(lookahead);
display(t,NONE);
continue;
default : return;
}
}
}
void E()
{
int t; 13

```

```

T();
while(1)
{
switch(lookahead)
{
case '+': t=lookahead;
Match(lookahead);
T();
display(t,NONE);
continue;
case '-': t=lookahead;
Match(lookahead);
T();
display(t,NONE);
continue;
default : return;
}
}
}
void parser()
{
lookahead=lexer();
while(lookahead!=DONE)
{
E();
Match(';');
}
}
main()
{
char ans[10];
printf("\n Program for recursive decent parsing ");
printf("\n Enter the expression ");
printf("And place ; at the end\n");
printf("Press Ctrl-Z to terminate\n");
parser();
}

```

Output:

```

Program for recursive decent parsing
Enter the expression And place ; at the end
Press Ctrl-Z to terminate 14

```


a+b*c;
Identifier: a
Identifier: b
Identifier: c
Arithmetic Operator: *
Arithmetic Operator: +
2*3;
Number: 2
Number: 3
Arithmetic Operator: *
+3;
line 5,Syntax error
Ctrl-Z

9) A Program to Generate Machine Code.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int label[20];
int no=0;
int main()
{
FILE *fp1,*fp2;
char fname[10],op[10],ch;
char operand1[8],operand2[8],result[8];
int i=0,j=0;
printf("\n Enter filename of the intermediate code");
scanf("%s",&fname);
fp1=fopen(fname,"r");
fp2=fopen("target.txt","w");
if(fp1==NULL || fp2==NULL)
{
printf("\n Error opening the file");
exit(0);
}
while(!feof(fp1))
{
fprintf(fp2,"\n");
fscanf(fp1,"%s",op);
i++;
if(check_label(i))
fprintf(fp2,"\nlabel#%d",i);
if(strcmp(op,"print")==0)
{
fscanf(fp1,"%s",result);
fprintf(fp2,"\n\t OUT %s",result);
}
if(strcmp(op,"goto")==0)
{
fscanf(fp1,"%s %s",operand1,operand2);
fprintf(fp2,"\n\t JMP %s,label#%s",operand1,operand2);
label[no++]=atoi(operand2);
}
if(strcmp(op,"[]")==0) 18
```

```

{
fscanf(fp1,"%s %s %s",operand1,operand2,result);
fprintf(fp2,"\n\t STORE %s[%s],%s",operand1,operand2,result);
}
if(strcmp(op,"uminus")==0)
{
fscanf(fp1,"%s %s",operand1,result);
fprintf(fp2,"\n\t LOAD -%s,R1",operand1);
fprintf(fp2,"\n\t STORE R1,%s",result);
}s
witch(op[0])
{
case '*': fscanf(fp1,"%s %s %s",operand1,operand2,result);
fprintf(fp2,"\n \t LOAD",operand1);
fprintf(fp2,"\n \t LOAD %s,R1",operand2);
fprintf(fp2,"\n \t MUL R1,R0");
fprintf(fp2,"\n \t STORE R0,%s",result);
break;
case '+': fscanf(fp1,"%s %s %s",operand1,operand2,result);
fprintf(fp2,"\n \t LOAD %s,R0",operand1);
fprintf(fp2,"\n \t LOAD %s,R1",operand2);
fprintf(fp2,"\n \t ADD R1,R0");
fprintf(fp2,"\n \t STORE R0,%s",result);
break;
case '-': fscanf(fp1,"%s %s %s",operand1,operand2,result);
fprintf(fp2,"\n \t LOAD %s,R0",operand1);
fprintf(fp2,"\n \t LOAD %s,R1",operand2);
fprintf(fp2,"\n \t SUB R1,R0");
fprintf(fp2,"\n \t STORE R0,%s",result);
break;
case '/': fscanf(fp1,"%s %s %s",operand1,operand2,result);
fprintf(fp2,"\n \t LOAD %s,R0",operand1);
fprintf(fp2,"\n \t LOAD %s,R1",operand2);
fprintf(fp2,"\n \t DIV R1,R0");
fprintf(fp2,"\n \t STORE R0,%s",result);
break;
case '%': fscanf(fp1,"%s %s %s",operand1,operand2,result);
fprintf(fp2,"\n \t LOAD %s,R0",operand1);
fprintf(fp2,"\n \t LOAD %s,R1",operand2);
fprintf(fp2,"\n \t DIV R1,R0");
fprintf(fp2,"\n \t STORE R0,%s",result); 19

```

```

break;
case '=': fscanf(fp1,"%s %s",operand1,result);
fprintf(fp2,"\n\t STORE %s %s",operand1,result);
break;
case '>': j++;
fscanf(fp1,"%s %s %s",operand1,operand2,result);
fprintf(fp2,"\n\t LOAD %s,R0",operand1);
fprintf(fp2,"\n\t JGT %s,label#%s",operand2,result);
label[no++]=atoi(result);
break;
case '<': fscanf(fp1,"%s %s %s",operand1,operand2,result);
fprintf(fp2,"\n\t LOAD %s,R0",operand1); fprintf(fp2,"\n\t
JLT %s,label#%d",operand2,result);
label[no++]=atoi(result);
break;
}
}
fclose(fp2); fclose(fp1);
fp2=fopen("target.txt","r");
if(fp2==NULL)
{
printf("Error opening the file\n");
exit(0);
}
do
{
ch=fgetc(fp2);
printf("%c",ch);
}while(ch!=EOF);
fclose(fp1);
return 0;
}
int check_label(int k)
{
int i;
for(i=0;i<no;i++)
{
if(k==label[i])
return 1;
}
return 0;
}

```

}

Input:

\$vi int.txt

=t1 2

[]=a 0 1

[]=a 1 2

[]=a 2 3

*t1 6 t2

+a[2] t2 t3

-a[2] t1 t2

/t3 t2 t2

uminus t2 t2

print t2

goto t2 t3

=t3 99

uminus 25 t2

*t2 t3 t3

uminus t1 t1

+t1 t3 t4

print t4

Output:

Enter filename of the intermediate code: int.txt

STORE t1,2

STORE a[0],1

STORE a[1],2

STORE a[2],3

LOAD t1,R0

LOAD 6,R1

ADD R1,R0

STORE R0,t3

LOAD a[2],R0

LOAD t2,R1

ADD R1,R0

STORE R0,t3

LOAD a[t2],R0

LOAD t1,R1

SUB R1,R0

STORE R0,t2

LOAD t3,R0

LOAD t2,R1 21

```
DIV R1,R0
STORE R0,t2
LOAD t2,R1
STORE R1,t2
LOAD t2, R0
JGT 5, label#11
Label#11: OUT t2
JMP t2, label#13
Label#13: STORE t3,99
LOAD 25,R1
STORE R1,t2
LOAD t2,R0
LOAD t3,R1
MUL R1,R0
STORE R0,t3
LOAD t1,R1
STORE R1,t1
LOAD t1,R0
LOAD t3,R1
ADD R1,R0
STORE R0,t4
OUT t4 22
```

10). Write a LEX Program to convert the substring abc to ABC from the given input string.

Lex Program

```
%{
/*The Program replaces the substring abc by ABC from
the given input string*/
#include<stdio.h>
#include<string.h>
int i;
%}
%%
[a-zA-Z]* {
for(i=0;i<=yyleng;i++)
{
If((yytext[i]=='a')&&( yytext[i+1]=='b')&&( yytext[i+2]=='c'))
{
yytext[i]='A';
yytext[i+1]='B';
yytext[i+2]='C';
}
}
Printf("%s", yytext);
}
[\t]*return;
.*{ECHO;}
\n {printf("%s", yytext);}
%%
main()
{
yytext();
}
int yywrap()
{
return 1;
} 23
```

11). Write a lex program to find out total number of vowels and consonants from the given input sting.

Lex Program

```
%{
/*Definition section*/
int vowel_cnt=0,consonant_cnt=0;
%}
vowel [aeiou]+
consonant [^aeiou]
eol \n
%%
{eol} return 0;
[\\t]+ ;
{vowel} {vowel_cnt++;}
%%
int main()
{
printf("\\n Enter some input string\\n");
yylex();
printf("vowels=%d and consonant=%d\\n",vowel_cnt,consonant_cnt);
return 0;
}
Int yywrap()
{
return 1;
}
```

```
%{
/*Definition section*/
int vowel_cnt=0,consonant_cnt=0;
%}
vowel [aeiou]+
consonant [^aeiou]
eol \n
%%
{eol} return 0;
[\\t]+ ;
{vowel} {vowel_cnt++;}
%%
int main()
{
printf("\\n Enter some input string\\n");
yylex();
```



```
printf("vowels=%d and consonant=%d\n",vowel_cnt,consonant_cnt);  
return 0;  
}  
Int yywrap()  
{  
return 1;  
}
```

Operating Systems

LABORATORY MANUAL

**B.TECH
(III YEAR – I SEM)
(2016-17)**

**Department of
Computer Science and Engineering**



**MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY
(Autonomous Institution – UGC, Govt. of India)**

Recognized under 2(f) and 12 (B) of UGC ACT 1956

(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2008 Certified)
Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, India



MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous Institution – UGC, Govt. of India)

Recognized under 2(f) and 12 (B) of UGC ACT 1956

(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2008 Certified)

Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, India

Department of Computer Science and Engineering

Vision

- To acknowledge quality education and instill high patterns of discipline making the students technologically superior and ethically strong which involves the improvement in the quality of life in human race.

Mission

- To achieve and impart holistic technical education using the best of infrastructure, outstanding technical and teaching expertise to establish the students into competent and confident engineers.
- Evolving the center of excellence through creative and innovative teaching learning practices for promoting academic achievement to produce internationally accepted competitive and world class professionals.



MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous Institution – UGC, Govt. of India)

Recognized under 2(f) and 12 (B) of UGC ACT 1956

(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2008 Certified)
Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, India

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

PEO1 – ANALYTICAL SKILLS

1. To facilitate the graduates with the ability to visualize, gather information, articulate, analyze, solve complex problems, and make decisions. These are essential to address the challenges of complex and computation intensive problems increasing their productivity.

PEO2 – TECHNICAL SKILLS

2. To facilitate the graduates with the technical skills that prepare them for immediate employment and pursue certification providing a deeper understanding of the technology in advanced areas of computer science and related fields, thus encouraging to pursue higher education and research based on their interest.

PEO3 – SOFT SKILLS

3. To facilitate the graduates with the soft skills that include fulfilling the mission, setting goals, showing self-confidence by communicating effectively, having a positive attitude, get involved in team-work, being a leader, managing their career and their life.

PEO4 – PROFESSIONAL ETHICS

To facilitate the graduates with the knowledge of professional and ethical responsibilities by paying attention to grooming, being conservative with style, following dress codes, safety codes, and adapting themselves to technological advancements.



MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous Institution – UGC, Govt. of India)

Recognized under 2(f) and 12 (B) of UGC ACT 1956

(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2008 Certified)
Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, India

PROGRAM SPECIFIC OUTCOMES (PSOs)

After the completion of the course, B. Tech Computer Science and Engineering, the graduates will have the following Program Specific Outcomes:

1. **Fundamentals and critical knowledge of the Computer System:-** Able to Understand the working principles of the computer System and its components , Apply the knowledge to build, asses, and analyze the software and hardware aspects of it .
2. **The comprehensive and Applicative knowledge of Software Development:** Comprehensive skills of Programming Languages, Software process models, methodologies, and able to plan, develop, test, analyze, and manage the software and hardware intensive systems in heterogeneous platforms individually or working in teams.
3. **Applications of Computing Domain & Research:** Able to use the professional, managerial, interdisciplinary skill set, and domain specific tools in development processes, identify the research gaps, and provide innovative solutions to them.



MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous Institution – UGC, Govt. of India)

Recognized under 2(f) and 12 (B) of UGC ACT 1956

(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2008 Certified)

Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, India

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 multi disciplinary 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.



MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(Permanently Affiliated to JNTUH, Approved by AICTE-Accredited by NBA & NAAC- A-Grade; ISO 9001:2008 Certified)

Maisammaguda, Dhulapally Post, Via Hakimpet, Secunderabad – 500100

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Operating Systems Lab (A50589) Manual

TABLE OF CONTENTS

S.No	TITLE	PAGE NO
1.	General Laboratory Instructions	3
2.	Objectives & Outcomes	4
3.	Software / Hardware Requirements	4
4	CPU Scheduling Algorithms a)FCFS b)SJF c)ROUND ROBIN d)PRIORITY	5-15
5.	File Allocation Strategies: a) SEQUENTIAL b) INDEXED c) LINKED	16-24
6.	Memory Variable Type And Fixed Type(MVT,MFT)	25-31
7.	File Organization Techniques a) SINGLE LEVEL DIRECTORY b) TWO LEVEL c) HIERARCHICAL d) DAG	32-37
8.	Bankers Algorithm for Dead Lock Avoidance	38-43
9.	Bankers Algorithm for Dead Lock Prevention	44-50

10.	Page Replacement Algorithm a) FIFO b) LRU c) LFU d) OPTIMAL	51-63
11.	Paging Technique	64-65
ADDITIONAL PROGRAMS		
12.	Write and execute a c program to simulate Placement Technique Best fit Algorithm	66-67
13.	Write and execute a c program to implement Disk Scheduling Algorithms i) FCFS ii) SSTF iii) SCAN	68-76

Faculty-In charge

Head of the Department

GENERAL LABORATORY INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before (to the starting time), those who come after 5 minutes will not be allowed into the lab.
2. Plan your task properly much before to the commencement, come prepared to the lab with the synopsis / program / experiment details.
3. Student should enter into the laboratory with:
 - a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.
 - b. Laboratory Record updated up to the last session experiments and other utensils (if any) needed in the lab.
 - c. Proper Dress code and Identity card.
4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.
5. Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.
6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.
7. Computer labs are established with sophisticated and high end branded systems, which should be utilized properly.
8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions .Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severe punishment.
9. Students must take the permission of the faculty in case of any urgency to go out ; if anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.
10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

Head of the Department

Principal

OBJECTIVES AND OUTCOMES

Objectives:

- To use linux operating system for study of operating system concepts.
- To write the code to implement and modify various concepts in operating systems using Linux.

Outcomes:

- The course objectives ensure the development of students applied skills in operating systems related areas.
- Students will gain knowledge in writing software routines modules or implementing various concepts of operating system.

RECOMMENDED SYSTEM / SOFTWARE REQUIREMENTS:

1. Intel based desktop PC of 166MHz or faster processor with at least 64 MB RAM and 100 MB free disk space.
2. C, C++ compiler.

USEFUL TEXT BOOKS / REFERENCES :

1. Operating System Concepts, Abraham Silberchatz, Peter B. Galvin, Greg Gagne, Eighth edition, John Wiley.
2. Operating Systems: Internals and Design Principles, Stallings, Sixth Edition–2009, Pearson Education
3. Modern Operating Systems, Andrew S Tanenbaum, Second Edition, PHI.
4. Operating Systems, S.Haldar, A.A.Aravind, Pearson Education.
5. Principles of Operating Systems, B.L.Stuart, Cengage learning, India Edition.
6. Operating Systems, A.S.Godbole, Second Edition, TMH.
7. An Introduction to Operating Systems, P.C.P. Bhatt, PHI.

EXPT NO.1**CPU SCHEDULING ALGORITHMS****A). FIRST COME FIRST SERVE:**

AIM: To write a c program to simulate the cpu scheduling algorithm First Come First Serve(FCFS)

DESCRIPTION:

To calculate the average waiting time using the FCFS algorithm first the waiting time of the first process is kept zero and the waiting time of the second process is the burst time of the first process and the waiting time of the third process is the sum of the burst times of the first and the second process and so on . after calculating all the waiting times the average waiting time is calculated as the average of all the waiting times. FCFS mainly says first come first serve the algorithm which came first will be served first.

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 name and the 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). $\text{Waiting time}(n) = \text{waiting time}(n-1) + \text{Burst time}(n-1)$

b). $\text{Turnaround time}(n) = \text{waiting time}(n) + \text{Burst time}(n)$

Step 6: Calculate

a) $\text{Average waiting time} = \text{Total waiting Time} / \text{Number of process}$

b) $\text{Average Turnaround time} = \text{Total Turnaround Time} / \text{Number of process}$

Step 7: Stop the process

SOURCE CODE :

// First Come First Serve (FCFS Program in C).

```
#include<stdio.h>
int arr[10], ser[10], pro[10];
float avgtime=0,avgwait;
void main()
{
int n=0,i=0,j=0,sum;
printf("enter number of process ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter arrival time and service time of %d process",i+1);
scanf("%d%d",&arr[i],&ser[i]);
}
sum=0;
avgtime=0;
for(i=0;i<n;i++)
{
pro[i]=sum;
sum=sum+ser[i];
avgtime=avgtime+sum-i;
}
avgwait=avgtime/n;
printf("average waiting time for fcfs is: %f",avgwait);
}
```

OUTPUT:

```
enter number of process: 3
enter arrival time and service time of 1 process 15
enter arrival time and service time of 2 process 21
enter arrival time and service time of 3 process 6
average waiting time for fcfs is: 17
```

VIVA QUESTIONS

- 1) Define CPU scheduling.
- 2) What is the difference between turnaround time and throughput?
- 3) What is a Dispatcher?
- 4) What is preemptive and no preemptive scheduling?
- 5) What is dispatch latency ?

B). SHORTEST JOB FIRST:

AIM: To write a program to stimulate the CPU scheduling algorithm Shortest job first (Non-Preemption)

DESCRIPTION:

To calculate the average waiting time in the shortest job first algorithm the sorting of the process based on their burst time in ascending order then calculate the waiting time of each process as the sum of the bursting times of all the process previous or before to that process.

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: Sort the processes names based on their Burt time

Step 7: For each process in the ready queue, calculate

a) $\text{Waiting time}(n) = \text{waiting time}(n-1) + \text{Burst time}(n-1)$

b) $\text{Turnaround time}(n) = \text{waiting time}(n) + \text{Burst time}(n)$

Step 8: Calculate

c) $\text{Average waiting time} = \text{Total waiting Time} / \text{Number of process}$

d) $\text{Average Turnaround time} = \text{Total Turnaround Time} / \text{Number of process}$

Step 9: Stop the process

SOURCE CODE :

```

#include<stdio.h>
void main()
{
int serv[10][2],tempo[1][2];
int i,j,n,totoal;
float average_time,average_wait;
printf("shortest job first sjf scheduling program in c tn");
printf("enter number of processn");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("nenter serv time of %d processn",i+1);
serv[i][0]=i;
scanf("%d",&serv[i][1]);
}
serv[0][1]=serv[0][1]-1;
for(i=0;i<n;i++)
{
for(j=0;j<n-1;j++) { tempo[0][0]=0; tempo[0][1]=0; if(serv[j][1]>=serv[j+1][1])
{
tempo[0][1]=serv[j][1];
tempo[0][0]=serv[j][0];
serv[j][1]=serv[j+1][1];
serv[j][0]=serv[j+1][0];
serv[j+1][1]=tempo[0][1];
serv[j+1][0]=tempo[0][0];
}
}
}
totoal=1;
average_time=0;
for(i=0;i<n;i++)
{
totoal=totoal+serv[i][1];
average_time=average_time+(totoal-serv[i][0]);
}
average_wait=average_time/n;
printf("average waiting time for sjf is: %f",average_wait);
}
enter number of process 4
enter serv time of 1 process 8
enter serv time of 2 process 4
enter serv time of 3 process 9
enter serv time of 4 process 5
average waiting time for sjf is: 13.00

```

C). ROUND ROBIN:

AIM: To simulate the CPU scheduling algorithm round-robin.

DESCRIPTION:

To aim is to calculate the average waiting time. There will be a time slice, each process should be executed within that time-slice and if not it will go to the waiting state so first check whether the burst time is less than the time-slice. If it is less than it assign the waiting time to the sum of the total times. If it is greater than the burst-time then subtract the time slot from the actual burst time and increment it by time-slot and the loop continues until all the processes are completed.

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

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

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

Step 8: Stop the process

SOURCE CODE :

```

#include<stdio.h>
int ttime,i,j,temp;
main()
{
    int pname[10],btime[10],pname2[10],btime2[10];
    int n,x,z;
    printf("Enter the no. of process:");
    scanf("%d",&n);
    printf("Enter the process name and burst time for the process\n");
    for(i=0;i<n;i++)
    {
        printf("Enter the process name:");
        scanf("%d",&pname2[i]);
        printf("Enter burst time for the process %d:",pname2[i]);
        scanf("%d",&btime2[i]);
    }
    printf("PROCESS NAME \t\t BURST TIME\n");
    for(i=0;i<n;i++)
        printf("%d \t \t %d\n",pname2[i],btime2[i]);
    z=1;
    while(z==1)
    {
        ttime=0;
        for(i=0;i<n;i++)
        {
            pname[i]=pname2[i];
            btime[i]=btime2[i];
        }
        printf ("PRESS 1.ROUND ROBIN 2.EXIT\n");
        scanf("%d",&x);
        switch(x)
        {
            case 1:
                rrobin(pname,btime,n);
                break;
            case 2:
                exit(0); break;
            default:
                printf("Invalid option");
                break;
        }
    }
    printf("\n\n If you want to continue press 1:");
}

```

```

scanf("%d",&z);
}
}
rrobin(int pname[],int btime[],int n)
{
    int tslice;
    j=0;
    printf("\n\t ROUND ROBIN SCHEDULING \n\n");
    printf("Enter the time slice: \n");
    scanf("%d",&tslice);
    printf("PROCESS NAME \t REMAINING TIME \t TOTAL TIME");
while(j<n)
    {
    for(i=0;i<n;i++)
    {
        if(btime[i]>0)
        {
            if(btime[i]>=tslice)
            {
                ttime+=tslice;
                btime[i]=btime[i]-tslice;
                printf("\n%d\t\t %d \t\t %d",pname[i],btime[i],ttime);
                if(btime[i]==0)
                    j++;
            }
            else
            {
                ttime+=btime[i];
                btime[i]=0;
                printf("\n%d\t\t %d \t\t %d",pname[i],btime[i],ttime);
            }
        }
    }
}
}
}
}
}

```

OUTPUT:

Enter the no. of process: 4

Enter the process name and burst time for the process

Enter the process name: 1

Enter burst time for the process 1: 8

Enter the process name: 2

Enter burst time for the process 2: 3

Enter the process name: 3

Enter burst time for the process 3: 6

Enter the process name: 4

Enter burst time for the process 4: 1

PROCESS NAME	BURST TIME
1	8
2	3
3	6
4	1

PRESS 1.ROUND ROBIN 2.EXIT

1

ROUND ROBIN SCHEDULING

Enter the time slice:2

PROCESS NAME	REMAINING TIME	TOTAL TIME
1	6	2
2	1	4
3	4	6
4	0	7
1	4	9
2	0	10
3	2	12
1	2	14
3	0	16

VIVA QUESTIONS:

1) Define race condition.

2)What is the difference between Round Robin and SJF scheduling ?

3)What are the advantages of Round Robin scheduling ?

4) What is known as Priority inversion?

5) What is known as Resource Reservation in Real time Scheduling?

D). PRIORITY:

AIM: To write a c program to simulate the cpu scheduling priority algorithm.

DESCRIPTION:

To calculate the average waiting time in the priority algorithm, sort the burst times according to their priorities and then calculate the average waiting time of the processes. The waiting time of each process is obtained by summing up the burst times of all the previous processes.

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: Sort the ready queue according to the priority number.

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

Step 6: Arrange the processes based on process priority

Step 7: For each process in the Ready Q calculate

Step 8: for each process in the Ready Q calculate

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

b) Turnaround time (n)= waiting time(n)+Burst time(n)

Step 9: Calculate

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

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

Print the results in an order.

Step 10: Stop the process

SOURCE CODE :

```

#include<stdio.h>
#include<conio.h>
void main()
{
    char p[10][5],temp[5];
    int i,j,pt[10],wt[10],totwt=0,pr[10],temp1,n;
    float avgwt;
    clrscr();
    printf("enter no of processes:");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("enter process%d name:",i+1);
        scanf("%s",&p[i]);
        printf("enter process time:");
        scanf("%d",&pt[i]);
        printf("enter priority:");
        scanf("%d",&pr[i]);
    }
    for(i=0;i<n-1;i++)
    {
        for(j=i+1;j<n;j++)
        {
            if(pr[i]>pr[j])
            {
                temp1=pr[i];
                pr[i]=pr[j];
                pr[j]=temp1;
                temp1=pt[i];
                pt[i]=pt[j];
                pt[j]=temp1;
                strcpy(temp,p[i]);
                strcpy(p[i],p[j]);
                strcpy(p[j],temp);
            }
        }
    }
    wt[0]=0;
    for(i=1;i<n;i++)
    {
        wt[i]=wt[i-1]+et[i-1];
        totwt=totwt+wt[i];
    }
}

```

```

}
avgwt=(float)totwt/n;
printf("p_name\t p_time\t priority\t w_time\n");
for(i=0;i<n;i++)
{
printf(" %s\t %d\t %d\t %d\n",p[i],pt[i],pr[i],wt[i]);
}
printf("total waiting time=%d\n avg waiting time=%f",tot,avg);
getch();
}

```

OUTPUT:

```

enter no of processes: 5
enter process1 name: aaa
enter process time: 4
enter priority:5
enter process2 name: bbb
enter process time: 3
enter priority:4
enter process3 name: ccc
enter process time: 2
enter priority:3
enter process4 name: ddd
enter process time: 5
enter priority:2
enter process5 name: eee
enter process time: 1
enter priority:1
p_name P_time priority w_time
eee      1      1      0
ddd      5      2      1
ccc      2      3      6
bbb      3      4      8
aaa      4      5     11
total waiting time=26
avg waiting time=5.20

```

EXPT NO.2**FILE ALLOCATION STRATEGIES****A) SEQUENTIAL:**

The most common form of file structure is the sequential file in this type of file, a fixed format is used for records. All records (of the system) have the same length, consisting of the same number of fixed length fields in a particular order because the length and position of each field are known, only the values of fields need to be stored, the field name and length for each field are attributes of the file structure.

ALGORITHM:

Step 1: Start the program.

Step 2: Get the number of files.

Step 3: Get the memory requirement of each file.

Step 4: Allocate the required locations to each in sequential order a). Randomly select a location from available location $s1 = \text{random}(100)$;

a) Check whether the required locations are free from the selected location.

```

if(b[s1].flag==0)
{
    for(j=s1;j<s1+p[i];j++)
    {
        if((b[j].flag)==0)
            count++;
    }
    if(count==p[i])

```

```

        break;
    }

    b) Allocate and set flag=1 to the allocated locations. for(s=s1;s<(s1+p[i]);s++)
    {

        k[i][j]=s;

        j=j+1;

        b[s].bno=s;

        b[s].flag=1;

    }

```

Step 5: Print the results file no, length ,Blocks allocated.

Step 6: Stop the program

SOURCE CODE :

```

#include<stdio.h>
main()
{
int f[50],i,st,j,len,c,k;
clrscr();
for(i=0;i<50;i++)
f[i]=0;
X:
printf("\n Enter the starting block & length of file");
scanf("%d%d",&st,&len);
for(j=st;j<(st+len);j++)
if(f[j]==0)
{
f[j]=1;
printf("\n%d->%d",j,f[j]);
}
else
{
printf("Block already allocated");
break;
}
if(j==(st+len))
printf("\n the file is allocated to disk");
printf("\n if u want to enter more files?(y-1/n-0)");
scanf("%d",&c);
if(c==1)
goto X;
else
exit();
getch();
}

```



```
}
```

OUTPUT:

Enter the starting block & length of file 4 10

4->1

5->1

6->1

7->1

8->1

9->1

10->1

11->1

12->1

13->1

The file is allocated to disk.

VIVA QUESTIONS

1)What is a file?

2)What are the various file operations?

3)List the various file attributes.

4)What are the information associated with an open file?

5)What are the different accessing methods of a file?

B) INDEXED:

AIM: To implement allocation using chained method

DESCRIPTION:

In the chained method file allocation table contains a field which points to starting block of memory. From it for each bloc a pointer is kept to next successive block. Hence, there is no external fragmentation.

ALGORITHM:

Step 1: Start the program.

Step 2: Get the number of files.

Step 3: Get the memory requirement of each file.

Step 4: Allocate the required locations by selecting a location randomly $q = \text{random}(100)$;

a) Check whether the selected location is free .

b) If the location is free allocate and set flag=1 to the allocated locations.

```

        q=random(100);
        {
            if(b[q].flag==0)
                b[q].flag=1;
                b[q].fno=j;
                r[i][j]=q;
        }
    
```

Step 5: Print the results file no, length ,Blocks allocated.

Step 6: Stop the program

SOURCE CODE :

```

#include<stdio.h>
int f[50],i,k,j,inde[50],n,c,count=0,p;
main()
{
clrscr();
for(i=0;i<50;i++)
f[i]=0;
x:
printf("enter index block \t");
scanf("%d",&p);
if(f[p]==0)
{
f[p]=1;
printf("enter no of files on index \t");
scanf("%d",&n);
}
else
{
printf("Block already allocated \n");
goto x;
}
for(i=0;i<n;i++)
scanf("%d",&inde[i]);
for(i=0;i<n;i++)
if(f[inde[i]]==1)
{
printf("Block already allocated");
goto x;
}
for(j=0;j<n;j++)
f[inde[j]]=1;
printf("\n allocated");
printf("\n file indexed");
for(k=0;k<n;k++)
printf("\n %d->%d:%d",p,inde[k],f[inde[k]]);
printf(" Enter 1 to enter more files and 0 to exit \t");
scanf("%d",&c);
if(c==1)
goto x;

```

```
else  
exit();  
getch();  
}
```

OUTPUT:

```
enter index block 9  
Enter no of files on index 3  
1 2 3  
Allocated  
File indexed  
9->1:1  
9->2;1  
9->3:1 enter 1 to enter more files and 0 to exit
```

VIVA QUESTIONS

- 1)What is indexed file allocation ?
- 2)What are the advantages of indexed file allocation ?
- 3)Explain how data is stored in indexed file ?
- 4)What is the difference between indexed file allocation and directory ?
- 5) What is meant by Executable file?

C) LINKED:

AIM: To implement linked file allocation technique.

DESCRIPTION:

In the chained method file allocation table contains a field which points to starting block of memory. From it for each bloc a pointer is kept to next successive block. Hence, there is no external fragmentation

ALGORITHM:

Step 1: Start the program.

Step 2: Get the number of files.

Step 3: Get the memory requirement of each file.

Step 4: Allocate the required locations by selecting a location randomly $q = \text{random}(100)$;

a) Check whether the selected location is free .

b) If the location is free allocate and set flag=1 to the allocated locations.

While allocating next location address to attach it to previous location

```

for(i=0;i<n;i++)
{
    for(j=0;j<s[i];j++)
    {
        q=random(100);
        if(b[q].flag==0)
            b[q].flag=1;
        b[q].fno=j;
    }
}

```

```

        r[i][j]=q;
    if(j>0)
    {
        p=r[i][j-1];
        b[p].next=q;
    }
}
}

```

Step 5: Print the results file no, length ,Blocks allocated.

Step 6: Stop the program

SOURCE CODE :

```

#include<stdio.h>
main()
{
    int f[50],p,i,j,k,a,st,len,n,c;
    clrscr();
    for(i=0;i<50;i++)
        f[i]=0;
    printf("Enter how many blocks that are already allocated");
    scanf("%d",&p);
    printf("\nEnter the blocks no.s that are already allocated");
    for(i=0;i<p;i++)
    {
        scanf("%d",&a);
        f[a]=1;
    }
    X:
    printf("Enter the starting index block & length");
    scanf("%d%d",&st,&len);
    k=len;
    for(j=st;j<(k+st);j++)
    {
        if(f[j]==0)
        {
            f[j]=1;

```

```

printf("\n%d->%d",j,f[j]);
}
else
{
printf("\n %d->file is already allocated",j);
k++;
}
}
printf("\n If u want to enter one more file? (yes-1/no-0)");
scanf("%d",&c);
if(c==1)
goto X;
else
exit();
getch( );
}

```

OUTPUT:

```

Enter how many blocks that are already allocated 3
Enter the blocks no.s that are already allocated 4 7 9
Enter the starting index block & length 3
7
3->1
4->1 file is already allocated
5->1
6->1
7->1 file is already allocated
8->1
9->1file is already allocated
10->1
11->1
12->1

```

VIVA QUESTIONS

- 1)What is linked file allocation ?
- 2) What are the advantages of linked file allocation ?
- 3)What is the difference between linked file allocation and indexed file allocation ?
- 4) What is meant by Source File?
- 5) What is meant by Object File?

EXPT NO. 3**MEMORY MANAGEMAENT****A). MEMORY MANAGEMENT WITH FIXED PARTITIONING
TECHNIQUE(MFT)**

AIM: To implement and simulate the MFT algorithm.

DESCRIPTION:

In this the memory is divided in two parts and process is fit into it. The process which is best suited in to it is placed in the particular memory where it suits. We have to check memory partition. If it suits, its status should be changed.

ALGORITHM:

Step1: Start the process.

Step2: Declare variables.

Step3: Enter total memory size

ms. Step4: Allocate memory for

os.

$Ms = ms - os$

Step5: Read the no partition to be divided n Partition size = ms/n .

Step6: Read the process no and process size.

Step 7: If process size is less than partition size allot else block the process. While allocating update memory wastage-external fragmentation.

$if(pn[i] == pn[j])$

$f = 1;$

$if(f == 0)\{$

$if(ps[i] \leq siz)$

$\{$

$extft = extft + size - ps[i];$


```
        avail[i]=1;  
        count++;  
    }  
}
```

Step 8: Print the results

Step 9 :Stop the
process.

SOURCE CODE :

```

#include<stdio.h>
#include<conio.h>
int main()
{
int m,p,s,p1;
int m1[4],i,f,f1=0,f2=0,fra1,fra2,s1,pos;
clrscr();
printf("Enter the memory size:");
scanf("%d",&m);
printf("Enter the no of partitions:");
scanf("%d",&p);
s=m/p;
printf("Each partn size is:%d",s);
printf("\nEnter the no of processes:");
scanf("%d",&p1);
pos=m;
for(i=0;i<p1;i++)
{
if(pos<s)
{
printf("\nThere is no further memory for process%d",i+1);
m1[i]=0;
break;
}
else
{
printf("\nEnter the memory req for process%d:",i+1);
scanf("%d",&m1[i]);
if(m1[i]<=s)
{
printf("\nProcess is allocated in partition%d",i+1);
fra1=s-m1[i];
printf("\nInternal fragmentation for process is:%d",fra1);
f1=f1+fra1;
pos=pos-s;
}
else
{
printf("\nProcess not allocated in partition%d",i+1);
s1=m1[i];

```

```

while(s1>s)
{
s1=s1-s;
pos=pos-s;
}
pos=pos-s;
fra2=s-s1;
f2=f2+fra2;
printf("\nExternal Fragmentation for this process is:%d",fra2);
}
}
}
printf("\nProcess \t allocated memory");
for(i=0;i<p1;i++)
printf("\n%5d \t %5d",i+1,m1[i]);
f=f1+f2;
printf("\nThe tot no of fragmentation is:%d",f);
getch();
return 0;
}

```

OUTPUT:

Enter the memory size: 80
 Enter the no of partitions: 4
 Each partition size: 20
 Enter the number of processes: 2
 Enter the memory req for process1: 18
 Process1 is allocated in partn1
 Internal fragmentation for process1 is: 2
 Enter the memory req for process2: 22
 Process2 is not allocated in partn2
 External fragmentation for process2 is: 18

Process	memory	allocated
1	20	18
2	20	22

The tot no of fragmentation is: 20

VIVA QUESTIONS

- 1)what is memory management ?
- 2)Define cache memory.
- 3)What is meant by Memory Compaction?
- 4)why RAM is called Main Memory ?
- 5)Define External Fragmentation .

MEMORY VARIABLE PARTIONING TYPE (MVT)

AIM: To write a program to simulate the MVT algorithm

ALGORITHM:

Step1: start the process.

Step2: Declare
variables.

Step3: Enter total memory size
ms. Step4: Allocate memory for
os.

$Ms=ms-os$

Step5: Read the no partition to be divided
n Partition size= ms/n .

Step6: Read the process no and process size.

Step 7: If process size is less than partition size allot else block the process. While
allocating update memory wastage-external fragmentation.

```

if(pn[i]==pn[j])
f=1;
if(f==0){
if(ps[i]<=size
)
{
extft=extft+size-ps[i];
avail[i]=1;
count++;
}
}

```

Step 8: Print the results

Step 9: Stop the process.

SOURCE CODE :

```

#include<stdio.h>
#include<conio.h>
void main()
{
int m=0,m1=0,m2=0,p,count=0,i;
clrscr();
printf("enter the memory capacity:");
scanf("%d",&m);
printf("enter the no of processes:");
scanf("%d",&p);
for(i=0;i<p;i++)
{
printf("\nenter memory req for process%d: ",i+1);
scanf("%d",&m1);
count=count+m1;
if(m1<=m)
{
if(count==m)
printf("there is no further memory remaining:");
printf("the memory allocated for process%d is: %d ",i+1,m);
m2=m-m1;
printf("\nremaining memory is: %d",m2);
m=m2;
}
}
else
{
printf("memory is not allocated for process%d",i+1);
}
printf("\nexternal fragmentation for this process is:%d",m2);
}
getch();
}

```

OUTPUT:**Input:**

Enter the memory capacity: 80

Enter no of processes: 2

Enter memory req for process1: 23

Output:

The memory allocated for process1 is: 80

Remaining memory is: 57

External fragmentation for this process is: 57

Enter memory req for process2: 52

The memory allocated for process2 is: 57

Remaining memory is: 5

External fragmentation for this process is: 5

VIVA QUESTIONS

- 1) What is Double Buffering?
- 2) What is meant by Free Space List?
- 3) What is meant by Page Table?
- 4) What is meant by Working Set?
- 5) What is meant by Global Replacement and Local Replacement?

EXPT NO. 4**FILE ORGANIZATION TECHNIQUES****A) SINGLE LEVEL DIRECTORY:**

AIM: Program to simulate Single level directory file organization technique.

ALGORITHM:

Step 1: Start

Step 2: Initialize values gd=DETECT, gm, count, i, j, mid, cir_x; Initialize character array
fname[10][20];

Step 3: Initialize graph function as Initgraph(& gd, &gm, "c:/tc/bgi"); Clear device();

Step 4: set back ground color with setbkcolor();

Step 5: read number of files in variable
count.

Step 6: if check i < count

Step 7: for i=0 & i < count

```

    i      increment;
    Cleardevice();
    setbkcolor(GREEN);
    read file name;
    setfillstyle(1,MAGENTA);
  
```

Step 8: mid=640/count;

```

    cir_x=mid/3;
    bar3d(270,100,370,150,0,0);
    settextstyle(2,0,4);
    settextstyle(1,1);
    outtextxy(320,125,"rootdirectory");
    setcolor(BLUE);
    i++;
  
```

Step 9: for j=0 & j <= i & cir_x += mid j

```

    increment;
    line(320,150,cir_x,250);
  
```

```

    fillellipse(cir_x,250,30,30);

    outtextxy(cir_x,250,fname[i]);

```

Step 10: End

SOURCE CODE :

```

#include<stdio.h>
#include<conio.h>
main()
{
int master,s[20];
char f[20][20][20];
char d[20][20];
int i,j;
clrscr();
printf("enter number of directorios:");
scanf("%d",&master);
printf("enter names of directories:");
for(i=0;i<master;i++)
scanf("%s",&d[i]);
printf("enter size of directories:");
for(i=0;i<master;i++)
scanf("%d",&s[i]);
printf("enter the file names :");
for(i=0;i<master;i++)
for(j=0;j<s[i];j++)
scanf("%s",&f[i][j]);
printf("\n");
printf(" directory \ tsize \ tfilenames \n");
printf("***** \n");
for(i=0;i<master;i++)
{
printf("%s \ t \ t%2d \ t",d[i],s[i]);
for(j=0;j<s[i];j++)
printf("%s \ n \ t \ t",f[i][j]);
printf("\n");
}
printf("\ t \n");
getch();
}

```


B) TWO LEVEL DIRECTORY

AIM: Program to simulate Two level file organization technique

ALGORITHM:

Step 1:Start

Step 2: Initialize structure elements

```
struct tree_element char name[20];
```

```
Initialize integer variables x, y, ftype, lx, rx, nc, level; struct tree_element
```

```
*link[5];}typedef structure tree_element
```

```
node;
```

Step 3:start main function

Step 4: Step variables gd=DETECT,gm; node *root;

```
root=NULL;
```

Step 5:create structure using create(&root,0,"null",0,639,320);

Step 6:initgraph(&gd, &gm,"c:\tc\bgi"); display(root); closegraph();

Step 7: end main function

Step 8: Initialize variables i,gap;

Step 9:if check *root==NULL (*root)=(node*)malloc(sizeof(node)); enter name of ir file

```
name in dname; fflush(stdin);
```

```
gets((*root)->name);
```

Step 10 if check lev==0||lev==1 (*root)->ftype=1; else(*root)->ftype=2; (*root)-

```
>level=lev; (*root)->y=50+lev*5; (*root)->x=x;
```

```
(*root)->lx=lx;
```

```
(*root)->rx=rx;
```

Step 11:for i=0&&i<5

```
increment i
```

```
(*root)->link[i]=NULL;
```

```
if check (*root)->ftype==1
```

Step 12: if check (lev==0||lev==1) if check(*root)->level==0 print "how many users" else

```
print "how many files" print (*root)->name
```

```
read (*root)->nc
```

Step 13: Then $(*root)->nc=0$; if $check(*root)->nc==0$ $gap=rx-lx$;

```
else gap=(rx-lx)/(*root)->nc;
```

Step 14: for $i=0 \&\& i < (*root)->nc$

```
    increment i;
```

```
    create(&((*root)->link[i]), lev+1, (*root)->name,
```

```
    lx+gap*i, lx+gap*i+gap, lx+gap*i+gap/2);
```

```
    then
```

```
    (*root)->nc=0;
```

Step 15: Initialize e display function Initialize i

```
set textstyle(2,0,4);
```

```
set textjustify(1,1);
```

```
set fillstyle(1,BLUE);
```

```
setcolor(14); step 13: if check root!=NULL
```

Step 16: for $i=0 \&\& i < root->nc$ increment i

```
    line(root->x, root->y, root->link[i]->x, root->link[i]->y);
```

Step 17: if $check\ root->ftype==1$

```
    bar3d(root->x-20, root->y-10, root->x+20, root->y+10, 0, 0);
```

```
    else fill ellipse(root->x, root->y, 20, 20);
```

```
    out textxy(root->x, root->y, root->name);
```

Step 18: for $i=0 \&\& i < root->nc$ increment i display($root->link[i]$);

Step 19: End

SOURCE CODE :

```

#include<stdio.h>
#include<conio.h>
struct st
{
char dname[10];
char sdname[10][10];
char fname[10][10][10];
int ds,sds[10];
}dir[10];
void main()
{
int i,j,k,n;
clrscr();
printf("enter number of directories:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter directory %d names:",i+1);
scanf("%s",&dir[i].dname);
printf("enter size of directories:");
scanf("%d",&dir[i].ds);
for(j=0;j<dir[i].ds;j++)
{
printf("enter subdirectory name and size:");
scanf("%s",&dir[i].sdname[j]);
scanf("%d",&dir[i].sds[j]);
for(k=0;k<dir[i].sds[j];k++)
{
printf("enter file name:");
scanf("%s",&dir[i].fname[j][k]);
}
}
}
printf("\ndirname\t\tsize\tsubdirname\tsize\tfiles");
printf("\n*****\n");
for(i=0;i<n;i++)
{
printf("%s\t\t%d",dir[i].dname,dir[i].ds);
for(j=0;j<dir[i].ds;j++)
{
printf("\t\t%s\t\t%d\t",dir[i].sdname[j],dir[i].sds[j]);
for(k=0;k<dir[i].sds[j];k++)

```

```
printf("%s\t",dir[i].fname[j][k]);  
printf("\n\t\t");  
}  
printf("\n");  
}  
getch();  
  
}
```

C)Write a C program for Hierachial directory.
D)Write a C program for DAG.

VIVA QUESTIONS

- 1) What are the different directory structures available?
- 2) Define UFD and MFD.
- 3) What are the allocation methods of a disk space?
- 4) What are the advantages of Contiguous allocation?
- 5) What are the advantages of Linked allocation?

EXPT NO. 5

DEADLOCK AVOIDANCE

AIM: Simulate bankers algorithm for DeadLock Avoidance (Banker's Algorithm)

DESCRIPTION:

Deadlock is a situation where in two or more competing actions are waiting for the other to finish, and thus neither ever does. When a new process enters a system, it must declare the maximum number of instances of each resource type it needed. This number may exceed the total number of resources in the system. When the user request a set of resources, the system must determine whether the allocation of each resources will leave the system in safe state. If it will the resources are allocation; otherwise the process must wait until some other process release the resources.

Data structures

-
- n-Number of process, m-number of resource types.
- Available: $Available[j]=k$, k – instance of resource type R_j is
- available. Max: If $max[i, j]=k$, P_i may request at most k instances resource R_j .

Allocation: If $Allocation[i, j]=k$, P_i allocated to k instances of resource

R_j Need: If $Need[I, j]=k$, P_i may need k more instances of resource type

R_j , $Need[I, j]=Max[I, j]-Allocation[I, j]$;

Safety Algorithm

1. Work and Finish be the vector of length m and n respectively,
Work=Available and Finish[i] =False.
2. Find an i such that both
 -
 - Finish[i] =False
 Need<=Work
If no such I exists go to step 4.
3. work= work + Allocation, Finish[i] =True;

4. if $\text{Finish}[1]=\text{True}$ for all I , then the system is in safe state. Resource request algorithm

Let Request i be request vector for the process P_i , If request $i[j]=k$, then process P_i wants k instances of resource type R_j .

1. if $\text{Request} \leq \text{Need } I$ go to step 2. Otherwise raise an error condition.
2. if $\text{Request} \leq \text{Available}$ go to step 3. Otherwise P_i must since the resources are available.
3. Have the system pretend to have allocated the requested resources to process P_i by modifying the state as follows;

$\text{Available} = \text{Available} - \text{Request } I;$

$\text{Allocation } I = \text{Allocation } + \text{Request}$

$I; \text{Need } i = \text{Need } i - \text{Request } I;$

If the resulting resource allocation state is safe, the transaction is completed and process P_i is allocated its resources. However if the state is unsafe, the P_i must wait for Request i and the old resource-allocation state is restored.

ALGORITHM:

1. Start the program.
2. Get the values of resources and processes.
3. Get the avail value.
4. After allocation find the need value.
5. Check whether its possible to allocate.
6. If it is possible then the system is in safe state.
7. Else system is not in safety state.
8. If the new request comes then check that the system is in safety.
9. or not if we allow the request.
10. stop the program.
11. end

SOURCE CODE :

```

#include<stdio.h>
#include<conio.h>
int max[100][100];
int alloc[100][100];
int need[100][100];
int avail[100];
int n,r;
void input();
void show();
void cal();
int main()
{
    int i,j;
    printf("***** Banker's Algo *****\n");
    input();
    show();
    cal();
    getch();
    return 0;
}
void input()
{
    int i,j;
    printf("Enter the no of Processes \t");
    scanf("%d",&n);
    printf("Enter the no of resources instances \t");
    scanf("%d",&r);
    printf("Enter the Max Matrix\n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<r;j++)
        {
            scanf("%d",&max[i][j]);
        }
    }
    printf("Enter the Allocation Matrix\n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<r;j++)
        {
            scanf("%d",&alloc[i][j]);
        }
    }
}

```

```

}
printf("Enter the available Resources\n");
for(j=0;j<r;j++)
{
    scanf("%d",&avail[j]);
}
}
void show()
{
    int i,j;
    printf("Process\t Allocation\t Max\t Available\t");
    for(i=0;i<n;i++)
    {
        printf("\nP%d\t ",i+1);
        for(j=0;j<r;j++)
        {
            printf("%d ",alloc[i][j]);
        }
        printf("\t");
        for(j=0;j<r;j++)
        {
            printf("%d ",max[i][j]);
        }
        printf("\t");
        if(i==0)
        {
            for(j=0;j<r;j++)
            printf("%d ",avail[j]);
        }
    }
}
void cal()
{
    int finish[100],temp,need[100][100],flag=1,k,c1=0;
    int safe[100];
    int i,j;
    for(i=0;i<n;i++)
    {
        finish[i]=0;
    }
    //find need matrix
    for(i=0;i<n;i++)
    {
        for(j=0;j<r;j++)

```



```

{
    need[i][j]=max[i][j]-alloc[i][j];
}
}
printf("\n");
while(flag)
{
    flag=0;
    for(i=0;i<n;i++)
    {
        int c=0;
        for(j=0;j<r;j++)
        {
            if((finish[i]==0)&&(need[i][j]<=avail[j]))
            {
                c++;
                if(c==r)
                {
                    for(k=0;k<r;k++)
                    {
                        avail[k]+=alloc[i][j];
                        finish[i]=1;
                        flag=1;
                    }
                    printf("P%d->",i);
                    if(finish[i]==1)
                    {
                        i=n;
                    }
                }
            }
        }
    }
    for(i=0;i<n;i++)
    {
        if(finish[i]==1)
        {
            c1++;
        }
        else
        {
            printf("P%d->",i);
        }
    }
}

```

```

}
if(c1==n)
{
printf("\n The system is in safe state");
}
else
{
printf("\n Process are in dead lock");
printf("\n System is in unsafe state");
}
}

```

OUTPUT:

Enter the no of processes 5

Enter the no of resources instances 3

Enter the max matrix

7 5 3

3 2 2

9 0 2

2 2 2

4 3 3

Enter the allocation matrix

0 1 0

2 0 0

3 0 2

2 1 1

0 0 2

Enter available resources 3 2 2

P1->p3->p4->p2->p0->

The system is in safe state.

VIVA QUESTIONS

- 1) What is Banker's algorithm?
- 2) Define deadlock avoidance?
- 3) What are conditions under which a deadlock situation may arise?
- 4) What are the methods for handling deadlocks?
- 5) What is a resource-allocation graph?

EXPT NO.6**DEADLOCK PREVENTION**

AIM: To implement deadlock prevention technique

Banker's Algorithm:

When a new process enters a system, it must declare the maximum number of instances of each resource type it needed. This number may exceed the total number of resources in the system. When the user request a set of resources, the system must determine whether the allocation of each resources will leave the system in safe state. If it will the resources are allocation; otherwise the process must wait until some other process release the resources.

DESCRIPTION:

Data structures

-
- n-Number of process, m-number of resource types.
-
- Available: $Available[j]=k$, k – instance of resource type R_j is available. Max: If $max[i, j]=k$, P_i may request at most k instances resource R_j .
-

Allocation: If $Allocation[i, j]=k$, P_i allocated to k instances of resource

R_j Need: If $Need[i, j]=k$, P_i may need k more instances of resource type R_j ,

$Need[i, j]=Max[i, j]-Allocation[i, j];$

Safety Algorithm

5. Work and Finish be the vector of length m and n respectively,

Work=Available and Finish[i] =False.

6. Find an i such that both

-
- Finish[i] =False

Need<=Work

If no such I exists go to step 4.

7. work=work+Allocation, Finish[i] =True;

if Finish[1]=True for all I, then the system is in safe state

ALGORITHM:

1. Start the program.
2. Get the values of resources and processes.
3. Get the avail value.
4. After allocation find the need value.
5. Check whether its possible to allocate.
6. If it is possible then the system is in safe state.
7. Else system is not in safety state
8. Stop the process.

SOURCE CODE :

```

#include< stdio.h>
#include< conio.h>
void main()
{
    int allocated[15][15],max[15][15],need[15][15],avail[15],tres[15],work[15],flag[15];
    int pno,rno,i,j,prc,count,t,total;
    count=0;
    clrscr();

    printf("\n Enter number of process:");
    scanf("%d",&pno);
    printf("\n Enter number of resources:");
    scanf("%d",&rno);
    for(i=1;i<=pno;i++)
    {
        flag[i]=0;
    }
    printf("\n Enter total numbers of each resources:");
    for(i=1;i<= rno;i++)
        scanf("%d",&tres[i]);

    printf("\n Enter Max resources for each process:");
    for(i=1;i<= pno;i++)
    {
        printf("\n for process %d:",i);
        for(j=1;j<= rno;j++)
            scanf("%d",&max[i][j]);
    }

    printf("\n Enter allocated resources for each process:");
    for(i=1;i<= pno;i++)
    {
        printf("\n for process %d:",i);
        for(j=1;j<= rno;j++)
            scanf("%d",&allocated[i][j]);
    }
}

```

```

printf("\n available resources:\n");
for(j=1;j<= rno;j++)
{
    avail[j]=0;
    total=0;
    for(i=1;i<= pno;i++)
    {
        total+=allocated[i][j];
    }
    avail[j]=tres[j]-total;
    work[j]=avail[j];
    printf("    %d \t",work[j]);
}

do
{

for(i=1;i<= pno;i++)
{
    for(j=1;j<= rno;j++)
    {
        need[i][j]=max[i][j]-allocated[i][j];
    }
}

printf("\n Allocated matrix    Max    need");
for(i=1;i<= pno;i++)
{
    printf("\n");
    for(j=1;j<= rno;j++)
    {
        printf("%4d",allocated[i][j]);
    }
    printf(" | ");
    for(j=1;j<= rno;j++)
    {
        printf("%4d",max[i][j]);
    }
    printf(" | ");
    for(j=1;j<= rno;j++)
    {
        printf("%4d",need[i][j]);
    }
}

```

```

}

prc=0;

for(i=1;i<= pno;i++)
{
    if(flag[i]==0)
    {
        prc=i;

        for(j=1;j<= rno;j++)
        {
            if(work[j]< need[i][j])
            {
                prc=0;
                break;
            }
        }
    }

    if(prc!=0)
    break;
}

if(prc!=0)
{
    printf("\n Process %d completed",i);
    count++;
    printf("\n Available matrix:");
    for(j=1;j<= rno;j++)
    {
        work[j]+=allocated[prc][j];
        allocated[prc][j]=0;
        max[prc][j]=0;
        flag[prc]=1;
        printf("  %d",work[j]);
    }
}

}while(count!=pno&&prc!=0);

if(count==pno)
    printf("\nThe system is in a safe state!!");
else

```

```
printf("\nThe system is in an unsafe state!!");

getch();
}
```

OUTPUT:

Enter number of process:5

Enter number of resources:3

Enter total numbers of each resources:10 5 7

Enter Max resources for each process:
for process 1:7 5 3

for process 2:3 2 2

for process 3:9 0 2

for process 4:2 2 2

for process 5:4 3 3

Enter allocated resources for each process:
for process 1:0 1 0

for process 2:3 0 2

for process 3:3 0 2

for process 4:2 1 1

for process 5:0 0 2
available resources:
2 3 0

Allocated matrix	Max	need
0 1 0 7 5 3	7 4 3	
3 0 2 3 2 2	0 2 0	
3 0 2 9 0 2	6 0 0	
2 1 1 2 2 2	0 1 1	
0 0 2 4 3 3	4 3 1	

Process 2 completed

Available matrix: 5 3 2

Allocated matrix Max need

0 1 0 | 7 5 3 | 7 4 3

0 0 0 | 0 0 0 | 0 0 0

3 0 2 | 9 0 2 | 6 0 0

2 1 1 | 2 2 2 | 0 1 1

0 0 2 | 4 3 3 | 4 3 1

Process 4 completed

Available matrix: 7 4 3

Allocated matrix Max need

0 1 0 | 7 5 3 | 7 4 3

0 0 0 | 0 0 0 | 0 0 0

3 0 2 | 9 0 2 | 6 0 0

0 0 0 | 0 0 0 | 0 0 0

0 0 2 | 4 3 3 | 4 3 1

Process 1 completed

Available matrix: 7 5 3

Allocated matrix Max need

0 0 0 | 0 0 0 | 0 0 0

0 0 0 | 0 0 0 | 0 0 0

3 0 2 | 9 0 2 | 6 0 0

0 0 0 | 0 0 0 | 0 0 0

0 0 2 | 4 3 3 | 4 3 1

Process 3 completed

Available matrix: 10 5 5

Allocated matrix Max need

0 0 0 | 0 0 0 | 0 0 0

0 0 0 | 0 0 0 | 0 0 0

0 0 0 | 0 0 0 | 0 0 0

0 0 0 | 0 0 0 | 0 0 0

0 0 2 | 4 3 3 | 4 3 1

Process 5 completed

Available matrix: 10 5 7

The system is in a safe state!!

EXPT NO.7**PAGE REPLACEMENT ALGORITHMS****A) FIRST IN FIRST OUT:**

AIM: To implement FIFO page replacement technique.

DESCRIPTION:

- The FIFO page-replacement algorithm is easy to understand and program. However, its performance is not always good.
- On the one hand, the page replaced may be an initialization module that was used a long time ago and is no longer needed.
- On the other hand, it could contain a heavily used variable that was initialized early and is in constant use.

ALGORITHM:

1. Start the process
2. Read number of pages n
3. Read number of pages no
4. Read page numbers into an array a[i]
5. Initialize avail[i]=0 .to check page hit
6. Replace the page with circular queue, while re-placing check page availability in the frame

Place avail[i]=1 if page is placed in the frame Count
page faults
7. Print the results.
8. Stop the process.

SOURCE CODE :

```
#include<stdio.h>
#include<conio.h>
int fr[3];
void main()
{
void display();
int i,j,page[12]={2,3,2,1,5,2,4,5,3,2,5,2};
int flag1=0,flag2=0,pf=0,frsize=3,top=0;
clrscr();
for(i=0;i<3;i++)
{
fr[i]=-1;
}
for(j=0;j<12;j++)
{
flag1=0;
flag2=0;
for(i=0;i<12;i++)
{
if(fr[i]==page[j])
{
flag1=1;
flag2=1;
break;
}
}
if(flag1==0)
{
for(i=0;i<frsize;i++)
{
if(fr[i]==-1)
{
fr[i]=page[j];
flag2=1;
break;
}
}
}
if(flag2==0)
{
fr[top]=page[j];
top++;
}
```

```

pf++;
if(top>=frsize)
top=0;
}
display();
}
printf("Number of page faults : %d ",pf);
getch();
}
void display()
{
int i;
printf("\n");
for(i=0;i<3;i++)
printf("%d\t",fr[i]);
}

```

OUTPUT :

2 -1 -1

2 3 -1

2 3 -1

2 3 1

5 3 1

5 2 1

5 2 4

5 2 4

3 2 4

3 2 4

3 5 4

3 5 2

Number of page faults : 6

VIVA QUESTIONS

- 1) What is a pure demand paging?
- 2) What is the basic approach of page replacement? If no frame is free is available, find ?
- 3) What are the various page replacement algorithms used for page replacement?
- 4) What are the major problems to implement demand paging?
- 5) Define effective access time.

B) LEAST RECENTLY USED

AIM: To implement LRU page replacement technique.

ALGORITHM :

1. Start the process
2. Declare the size
3. Get the number of pages to be inserted
4. Get the value
5. Declare counter and stack
6. Select the least recently used page by counter value
7. Stack them according the selection.
8. Display the values
9. Stop the process

SOURCE CODE :

```
#include<stdio.h>
#include<conio.h>
int fr[3];
void main()
{
void display();
int p[12]={2,3,2,1,5,2,4,5,3,2,5,2},i,j,fs[3];
int index,k,l,flag1=0,flag2=0,pf=0,frsize=3;
clrscr();
for(i=0;i<3;i++)
{
fr[i]=-1;
}
for(j=0;j<12;j++)
{
flag1=0,flag2=0;
for(i=0;i<3;i++)
{
if(fr[i]==p[j])
{
flag1=1;
flag2=1;
break;
}
}
if(flag1==0)
{
for(i=0;i<3;i++)
{
if(fr[i]==-1)
{
fr[i]=p[j];
flag2=1;
break;
}
}
}

if(flag2==0)
{
for(i=0;i<3;i++)
fs[i]=0;
```

```

for(k=j-1,l=1;l<=frsize-1;l++,k--)
{
for(i=0;i<3;i++)
{
if(fr[i]==p[k])
fs[i]=1;
}
}
for(i=0;i<3;i++)
{
if(fs[i]==0)
index=i;
}
fr[index]=p[j];
pf++;
}
display();
}
printf("\n no of page faults :%d",pf);
getch();
}

void display()
{
int i;
printf("\n");
for(i=0;i<3;i++)
printf("\t%d",fr[i]);
}

```

OUTPUT :

```

2 -1 -1
2 3 -1
2 3 -1
2 3 1
2 5 1
2 5 1
2 5 4
2 5 4
3 5 4
3 5 2
3 5 2
3 5 2
no of page faults : 4

```

C) LFU: LEAST FREQUENTLY USED

AIM: To implement LFU page replacement technique.

ALGORITHM:

- 1.Start Program
- 2.Read Number Of Pages And Frames
- 3.Read Each Page Value
- 4.Search For Page In The Frames
- 5.If Not Available Allocate Free Frame
- 6.If No Frames Is Free Repalce The Page With The Page That Is Leastly Used
- 7.Print Page Number Of Page Faults
- 8.Stop process.

SOURCE CODE :

```

#include<stdio.h>
int main()
{
    int f,p;
    int pages[50],frame[10],hit=0,count[50],time[50];
    int i,j,page,flag,least,minTime,temp;

    printf("Enter no of frames : ");
    scanf("%d",&f);
    printf("Enter no of pages : ");
    scanf("%d",&p);

    for(i=0;i<f;i++)
    {
        frame[i]=-1;
    }
    for(i=0;i<50;i++)
    {
        count[i]=0;
    }
    printf("Enter page no : \n");
    for(i=0;i<p;i++)
    {
        scanf("%d",&pages[i]);
    }
    printf("\n");
    for(i=0;i<p;i++)
    {
        count[pages[i]]++;
        time[pages[i]]=i;
        flag=1;
        least=frame[0];
        for(j=0;j<f;j++)
        {
            if(frame[j]==-1 || frame[j]==pages[i])
            {
                if(frame[j]!=-1)
                {
                    hit++;
                }
            }
            flag=0;
        }
    }
}

```

```

frame[j]=pages[i];
    break;
}
if(count[least]>count[frame[j]])
{
    least=frame[j];
}
}
if(flag)
{
    minTime=50;
    for(j=0;j<f;j++)
    {
        if(count[frame[j]]==count[least] && time[frame[j]]<minTime)
        {
            temp=j;
            minTime=time[frame[j]];
        }
    }
    count[frame[temp]]=0;
    frame[temp]=pages[i];
}
for(j=0;j<f;j++)
{
    printf("%d ",frame[j]);
}
printf("\n");
}
printf("Page hit = %d",hit);
return 0;
}

```

VIVA QUESTIONS:

- 1) Define lazy swapper.
- 2) What is meant by Page Fault?
- 3) What is meant by Thrashing?
- 4) What is meant by Locality of reference?
- 5) What is meant by Page Table?

D). OPTIMAL

AIM: To implement Optimal page replacement technique.

ALGORITHM:

Here we select the page that will not be used for the longest period of time.

1. Start Program
2. Read Number Of Pages And Frames
3. Read Each Page Value
4. Search For Page In The Frames
5. If Not Available Allocate Free Frame
6. If No Frames Is Free Replace The Page With The Page That Is Not Used In Next
N Pages(N Is Number Of Frames)
7. Print Page Number Of Page Faults
8. Stop process.

SOURCE CODE :

```

#include<stdio.h>
#include<conio.h>
int fr[3];
void main()
{
void display();
int p[12]={2,3,2,1,5,2,4,5,3,2,5,2},i,j,fs[3];
int max,found=0,lg[3],index,k,l,flag1=0,flag2=0,pf=0,frsize=3;
clrscr();
for(i=0;i<3;i++)
{
fr[i]=-1;
}
for(j=0;j<12;j++)
{
flag1=0;
flag2=0;
for(i=0;i<3;i++)
{
if(fr[i]==p[j])
{
flag1=1;
flag2=1;
break;
}
}
if(flag1==0)
{
for(i=0;i<3;i++)
{
if(fr[i]==-1)
{
fr[i]=p[j];
flag2=1;
break;
}}}
if(flag2==0)
{
for(i=0;i<3;i++)
lg[i]=0;
for(i=0;i<frsize;i++)
{
for(k=j+1;k<12;k++)

```

```

{
if(fr[i]==p[k])
{
lg[i]=k-j;
break;
}}}
found=0;
for(i=0;i<frsize;i++)
{
if(lg[i]==0)
{
index=i;
found=1;
break;
}
}
if(found==0)
{
max=lg[0];
index=0;
for(i=1;i<frsize;i++)
{
if(max<lg[i])
{
max=lg[i];
index=i;
}
}
}
fr[index]=p[j];
pf++;
}
display();
}
printf("\n no of page faults:%d",pf);
getch();
}

void display()
{
int i;
printf("\n");
for(i=0;i<3;i++)
printf("\t%d",fr[i]);
}

```

OUTPUT :

2 -1 -1

2 3 -1

2 3 -1

2 3 1

2 3 5

2 3 5

4 3 5

4 3 5

4 3 5

2 3 5

2 3 5

2 3 5

no of page faults : 3

VIVA QUESTIONS

- 1)How is memory protected in a paged environment?
- 2)What are pages and frames?
- 3)What is the use of valid-invalid bits in paging?
- 4)How does page replacement takes place in optimal algorithm?
- 5)What is the disadvantage of optimal page replacement algorithm?

EXPT NO.8**PAGING**

AIM: TO implement simple paging technique.

ALGORITHM:

Step 1: Read all the necessary input from the keyboard.

Step 2: Pages - Logical memory is broken into fixed - sized blocks.

Step 3: Frames – Physical memory is broken into fixed – sized blocks.

Step 4: Calculate the physical address using the following

$$\text{Physical address} = (\text{Frame number} * \text{Frame size}) + \text{offset}$$

Step 5: Display the physical address.

Step 6: Stop the process.

SOURCE CODE :

```
#include<stdio.h>
```

```
#include<conio.h>
main()
{
    int np,ps,i;
    int *sa;
    clrscr();
    printf("enter how many pages\n");
    scanf("%d",&np);
    printf("enter the page size \n");
    scanf("%d",&ps);
    sa=(int*)malloc(2*np);
    for(i=0;i<np;i++)
    {
        sa[i]=(int)malloc(ps);
        printf("page%d \t address %u \n",i+1,sa[i]);
    }
    getch();
}
```

OUTPUT:

Enter how many pages: 5

Enter the page size: 4

o/p:

Page1 Address: 1894

Page2 Address: 1902

Page3 Address: 1910

Page4 Address: 1918

Page5 Address: 1926

VIVA QUESTIONS

- 1) Define Paging ?
- 2) Why OS Using Paging Concept ?
- 3) What is the basic method of segmentation?
- 4) How is thrashing prevented using Page-Fault Frequency strategy?
- 5) What is the difference between paging and Segmentation ?

ADDITIONAL PROGRAMS

EXPT NO.9

AIM: - Program to implement Best-Fit algorithm.

Description: -

Allocate the smallest hole that is big enough. We must search the entire list, unless the list is ordered by size. This strategy produces the smallest left over hole.

SOURCE CODE :

```
// * Best- Fit * //
#include<stdio.h>
#include<conio.h>
void main()
{
int m[5],n,mr[10],i,j,t,rm=0,pr[5]={1,2,3,4,5},q;
char p[10][10];
clrscr();
printf("Enter the no.of memory partitions \n");
scanf("%d",&q);
for(i=0;i<q;i++)
{
printf("the memory for partition %d \t",i++);
scanf("%d",&m[i]);
}
printf("Enter no.of process \n:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf(" process name \t ");
scanf("%s",&p[i]);
printf("Enter memory required \t");
scanf("%d",&mr[i]);
}
for(i=0;i<q;i++)
{
for(j=i+1;j<q;j++)
{
if(m[i]>m[j])
{
t=m[i];
m[i]=m[j];
m[j]=t;
}
```

```

t=pr[i];
pr[i]=pr[j];
pr[j]=t;
}
}
}
for(i=0;i<n;i++)
{
for(j=0;j<q;j++)
if(m[j]>mr[i])
{
printf(" Process %s has been allocated partitio p[i],pr[j],m[j]);
m[j]=m[j]-mr[i];
break;
}
}
for(i=0;i<q;i++)
rm=rm+m[i];
printf("remaining
free space is %d kb \n", rm);
}

```

OUTPUT

Process p1 has been allocated partition 4 with space 300kb
 Process p2 has been allocated partition 2 with space 500kb
 Process p3 has been allocated partition 3 with space 200kb
 Process p4 has been allocated partition 5 with space 600kb
 Remaining free space is 533kb.

VIVA QUESTIONS

- 1)What are the common strategies to select a free hole from a set of available holes?
- 2)What do you mean by best fit?
- 3)What do you mean by first fit?
- 4)Define Fragmentation.
- 5) What do you mean by worst fit?

EXPT NO.10**Disk Scheduling**

- The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth.
- Access time has two major components
 - *Seek time* is the time for the disk are to move the heads to the cylinder containing the desired sector.
 - *Rotational latency* is the additional time waiting for the disk to rotate the desired sector to the disk head.
- Minimize seek time
- $\text{Seek time} \approx \text{seek distance}$
- Disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.

AIM:

To write a C program to implement the Disk Scheduling algorithm for First Come

First Served (FCFS), Shortest Seek Time First (SSTF), and SCAN.

PROBLEM DESCRIPTION:

Disk Scheduling is the process of deciding which of the cylinder request is in the ready queue is to be accessed next. The access time and the bandwidth can be improved by scheduling the servicing of disk I/O requests in good order.

Access Time:

The access time has two major components: Seek time and Rotational Latency.

Seek Time:

Seek time is the time for disk arm to move the heads to the cylinder containing the desired sector.

Rotational Latency:

Rotational latency is the additional time waiting for the disk to rotate the desired sector to the disk head.

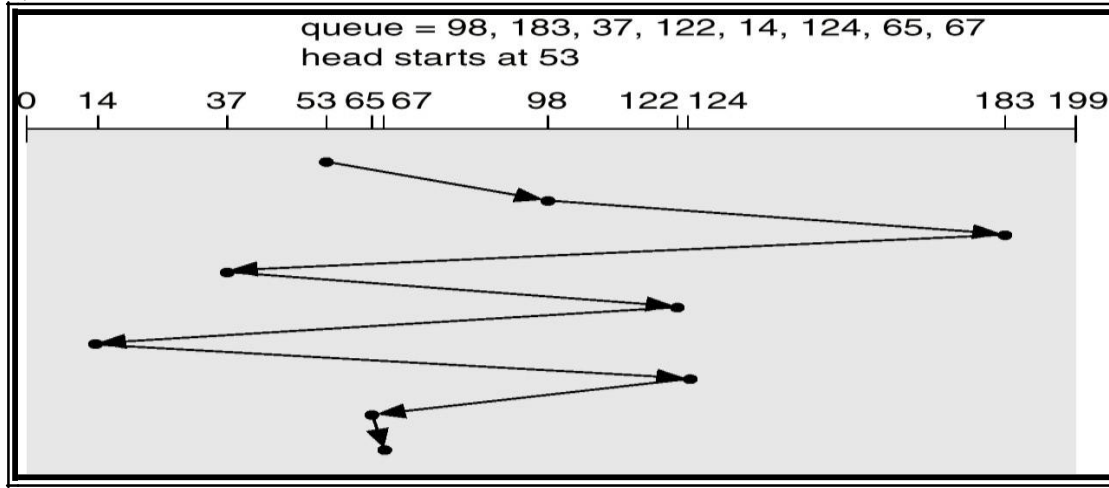
Bandwidth:

The disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.

ALGORITHM:

1. Input the maximum number of cylinders and work queue and its head starting position.
2. First Come First Serve Scheduling (FCFS) algorithm – The operations are performed in order requested.
3. There is no reordering of work queue.
4. Every request is serviced, so there is no starvation.
5. The seek time is calculated.
6. Shortest Seek Time First Scheduling (SSTF) algorithm – This algorithm selects the request with the minimum seek time from the current head position.
7. Since seek time increases with the number of cylinders traversed by the head, SSTF chooses the pending request closest to the current head position.
8. The seek time is calculated.
9. SCAN Scheduling algorithm – The disk arm starts at one end of the disk, and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk.
10. At the other end, the direction of head movement is reversed, and servicing continues.
11. The head continuously scans back and forth across the disk.
12. The seek time is calculated.
13. Display the seek time and terminate the program

a) FCFS:



SOURCE CODE :

```
#include<conio.h>
#include<stdio.h>

int main()
{
    int i,j,sum=0,n;
    int ar[20],tm[20];
    int disk;
    clrscr();
    printf("enter number of location \t");
    scanf("%d",&n);
    printf("enter position of head \t");
    scanf("%d",&disk);
    printf("enter elements of disk queue \n");
    for(i=0;i<n;i++)
    {
        scanf("%d",&ar[i]);
        tm[i]=disk-ar[i];
    }
```

```

if(tm[i]<0)
{
tm[i]=ar[i]-disk;
}
disk=ar[i];
sum=sum+tm[i];
}
/*for(i=0;i<n;i++)
{
printf("\n%d",tm[i]);
} */

printf("\nmovement of total cylinders %d",sum);
getch();
return 0;
}

```

Output:

```

enter number of location  8
enter position of head  53
enter elements of disk queue
98
183
37
122
14
124
65
67
movement of total cylinders 640

```

VIVA QUESTIONS

- 1) Define rotational latency?
- 2) What are the various Disk-Scheduling algorithms?
- 3) What is LOOK Scheduling ?
- 4) What is meant by Seek Time?
- 5) What is meant by Low-level formatting?

b) SSTF :

- It seems reasonable to service all the requests close to the current head position before moving the head far away to service other requests. This assumption is the basis for the shortest-seek-time-first (SSTF) algorithm.
- The SSTF algorithm selects the request with the minimum seek time from the current head position.
- Since seek time increases with the number of cylinders traversed by the head, SSTF chooses the pending request closest to the current head position.

SOURCE CODE :

```
#include<conio.h>
#include<stdio.h>
struct di
{
int num
;
int flag;
};
int main()
{
int i,j,sum=0,n,min,loc,x,y;
struct di d[20];
int disk;
int ar[20],a[20];
clrscr();
printf("enter number of location \t");
scanf("%d",&n);
printf("enter position of head \t");
scanf("%d",&disk);
printf("enter elements of disk queue \n");
for(i=0;i<n;i++)
{
```

```

scanf("%d",&d[i].num); d[i].flag=0;
}
for(i=0;i<n;i++)
{
    x=0; min=0;loc=0;
    for(j=0;j<n;j++)
    {
        if(d[j].flag==0)
        {
            if(x==0)
            {
                ar[j]=disk-d[j].num;
                if(ar[j]<0){ ar[j]=d[j].num-disk;}
                min=ar[j];loc=j;x++; }
            else
            {
                ar[j]=disk-d[j].num;
                if(ar[j]<0){ ar[j]=d[j].num-disk;}
            }
            if(min>ar[j]){ min=ar[j]; loc=j;}
        }
    }
    d[loc].flag=1;
    a[i]=d[loc].num-disk;
    if(a[i]<0){a[i]=disk-d[loc].num;}

    disk=d[loc].num;
}

```

```

for(i=0;i<n;i++)
{
    sum=sum+a[i];
}

printf("\nmovement of total cylinders %d",sum);
getch();

```



```

return 0;
}

```

Output:

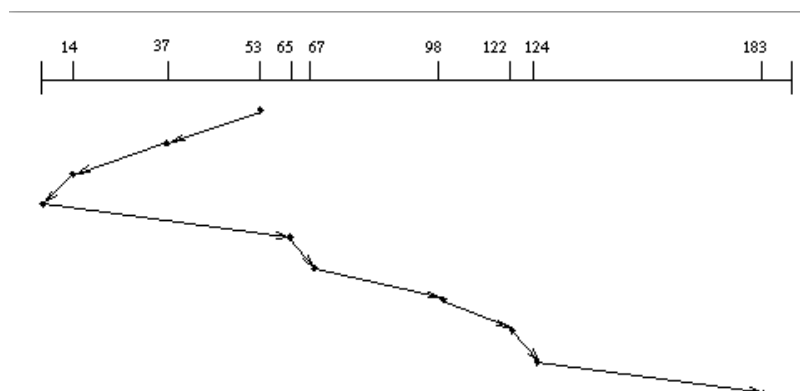
```

enter number of location 8
enter position of head 53
enter elements of disk queue
98
183
37
122
14
124
65
67
movement of total cylinders 236

```

c) SCAN:

- In the SCAN algorithm, the disk arm starts at one end of the disk and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk.
- At the other end, the direction of head movement is reversed, and servicing continues. The head continuously scans back and forth across the disk.
- The SCAN algorithm is sometimes called the elevator algorithm, since the disk arms behaves just like an elevator in a building, first servicing all the requests going up and then reversing to service requests the other way.



SOURCE CODE :

```
#include<conio.h>

#include<stdio.h>
int main()
{
    int i,j,sum=0,n;
    int d[20];
    int disk; //loc of head
    int temp,max;
    int dloc; //loc of disk in array
    clrscr();
    printf("enter number of location \t");
    scanf("%d",&n);
    printf("enter position of head \t");
    scanf("%d",&disk);
    printf("enter elements of disk queue \n");
    for(i=0;i<n;i++)
    {
        scanf("%d",&d[i]);
    }
    d[n]=disk;
    n=n+1;
    for(i=0;i<n;i++) // sorting disk locations
    {
        for(j=i;j<n;j++)
        {
            if(d[i]>d[j])
            {
                temp=d[i];
                d[i]=d[j];
                d[j]=temp;
            }
        }
    }
}
```

```

}
max=d[n];
for(i=0;i<n;i++) // to find loc of disc in array
{
if(disk==d[i]) { dloc=i; break; }
}
for(i=dloc;i>=0;i--)
{
printf("%d -->",d[i]);
}
printf("0 -->");
for(i=dloc+1;i<n;i++)
{
printf("%d-->",d[i]);
}
sum=disk+max;
printf("\nmovement of total cylinders %d",sum);
getch();
return 0;
}

```

Output:

Enter no of location 8
Enter position of head 53
Enter elements of disk queue
98
183
37
122
14
124
65
67
53->37->14->0->65->67->98->122->124->183->
Movement of total cylinders 236.