**Date:30.04.2021**

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

#include<stdio.h>

int n,nf,in[100],p[50],hit=0,i,j,k,pgfaultcnt=0;

void getData()

{

printf("\nEnter length of page references sequences:");

scanf("%d",&n);

printf("\nEnter length of page references sequences:");

for(i=0;i<n;i++)

scanf("%d",&in[i]);

printf("\nEnter no of frames:");

scanf("%d",&nf);

}

void initialize()

{

pgfaultcnt=0;

for(i=0;i<nf;i++)

p[i]=9999;

}

int isHit(int data)

{

hit=0;

for(j=0;j<nf;j++)

{

if(p[j]==data)

{

hit=1;

break;

}

}

return hit;

}

void dispPages()

{

for(k=0;k<nf;k++)

{

if(p[k]!=9999)

printf("%d",p[k]);

}

}

void fifo()

{

initialize();

int m=0;

for(i=0;i<n;i++)

{

printf("\nFor%d:",in[i]);

if(isHit(in[i])==0)

{

p[m]=in[i];

m=(m+1)%nf;

pgfaultcnt++;

dispPages();

}

else

printf("No page fault");

}

printf("\nTotal no of pages faults:%d",pgfaultcnt);

}

void lru()

{

initialize();

int least[50];

for(i=0;i<n;i++)

{

printf("\nFor%d:",in[i]);

if(isHit(in[i])==0)

{

for(j=0;j<nf;j++)

{

int pg=p[j];

int found=0;

for(k=i-1;k>=0;k--)

{

if(pg==in[k])

{

least[j]=k;

found=1;

break.;

}

else

found=0;

}

if(!found)

least[j]=-9999;

}

int min=9999;

int repindex;

for(j=0;j<nf;j++)

{

if(least[j]<min)

{

min=least[j];

repindex=j;

}

}

p[repindex]=in[i];

pgfaultcnt++;

dispPages();

}

else

printf("No pages fault!");

}

printf("\nTotal no of pages faults:%d",pgfaultcnt);

}

int main()

{

int choice.

while (1)

{

printf("\nPages Replacement Algorithm\n1.enter data\n2.FIFO\n3.LRU\n4.Exit\nEnter your choice:");

Scanf("%d”, &choice);

switch(choice)

{

case 1: getData();

break;

case 2: fifo();

break;

case 3:lru();

break;

default: return 0;

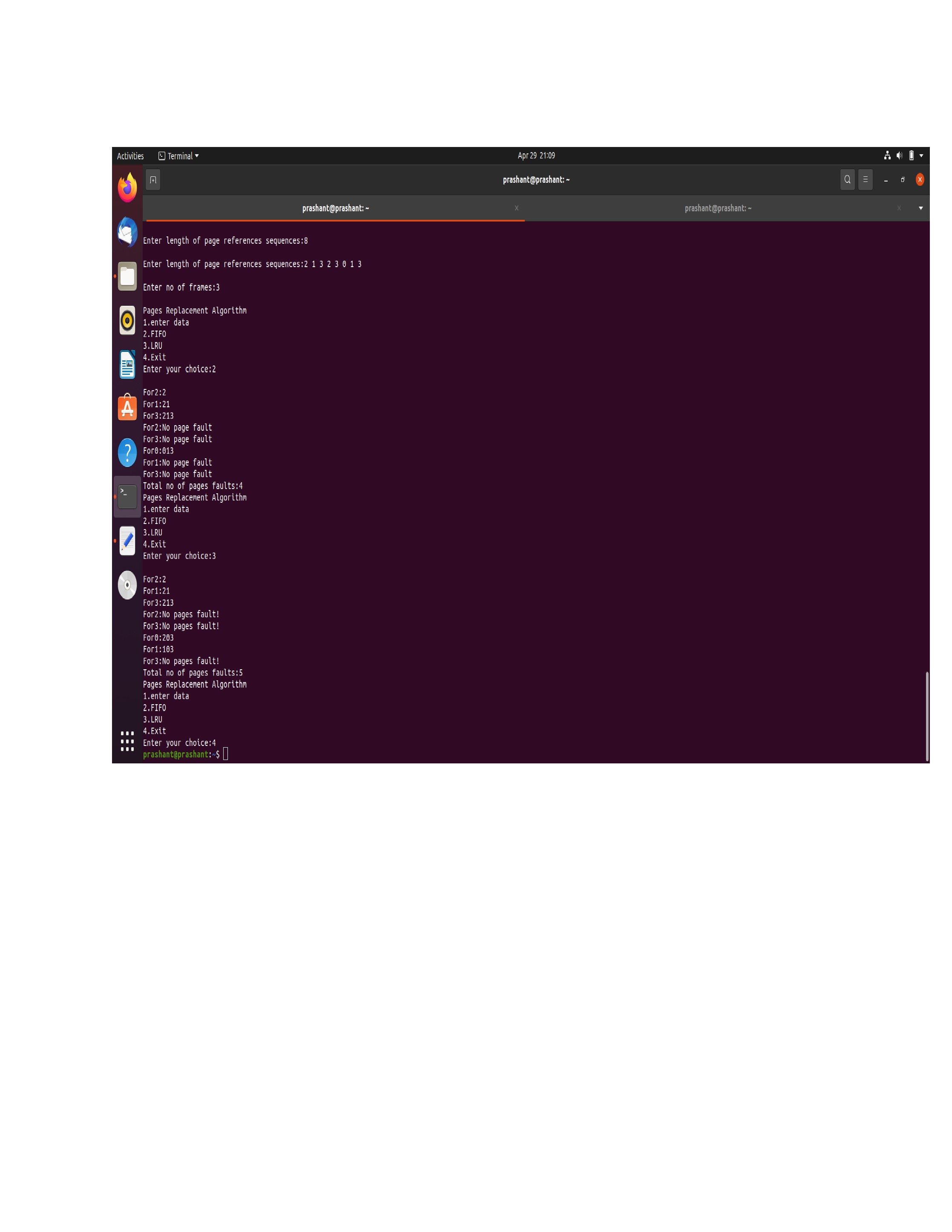
break;

}

}

}

OUTPUT

 Graphical user interface, text

Description automatically generated

**Date:08.05.2021**

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

#include<stdio.h>

#include<stdlib.h>

int main()

{

int Max[10][10],need[10][10],alloc[10][10],avail[10],completed[10],safeSequence[10];

int p,r,i,j,process,count;

count=0;

printf("Enter the no of processes:");

scanf("%d",&p);

for(i=0;i<p;i++)

{

completed[i]=0;

}

printf("\nEnter the no of resource:");

scanf("%d",&r);

printf("\n\n Enter the Max Matrix for each process:");

for(i=0;i<p;i++)

{

printf("\nFor process %d:",i+1);

for(j=0;j<r;j++)

{

scanf("%d",&Max[i][j]);

}

}

printf("\n\n Enter the allocation for each process:");

for(i=0;i<p;i++)

{

printf("\nFor process %d:",i+1);

for(j=0;j<r;j++){

scanf("%d",&alloc[i][j]);

}

}

printf("\n\nEnter the Available Resources:");

for(i=0;i<r;i++){

scanf("%d",&avail[i]);

}

for(i=0;i<p;i++)

{

for(j=0;j<r;j++)

{

need[i][j]=Max[i][j]-alloc[i][j];

}

}

do

{

printf("\n Max matrix:\tAllocation matrix:\n");

for(i=0;i<p;i++)

{

for(j=0;j<r;j++)

{

printf("%d",Max[i][j]);

}

printf("\t\t");

for(j=0;j<r;j++)

{

printf("%d",alloc[i][j]);

}

printf("\n");

}

process=-1;

for(i=0;i<p;i++)

{

if(completed[i]==0) //if not complete

{

process=i;

for(j=0;j<r;j++)

{

if(avail[j]< need[i][j])

{

process=-1;

break;

}

}

}

if(process!=-1){

break;

}

}

if(process!=-1)

{

printf("\nProcess%d runs to completion!",process+1);

safeSequence[count]=process+1;

count++;

for(j=0;j<r;j++)

{

avail[j]+=alloc[process][j];

alloc[process][j]=0;

Max[process][j]=0;

completed[process]=1;

}

}

} while(count!=p && process!=-1);

if (count == p)

{

printf("\nThe system is in a safe state! \n");

printf("Safe sequence : < ");

for (i = 0; i < p; i++)

{

printf("P %d", safeSequence[i]);

printf(" >\n");

}

}

else

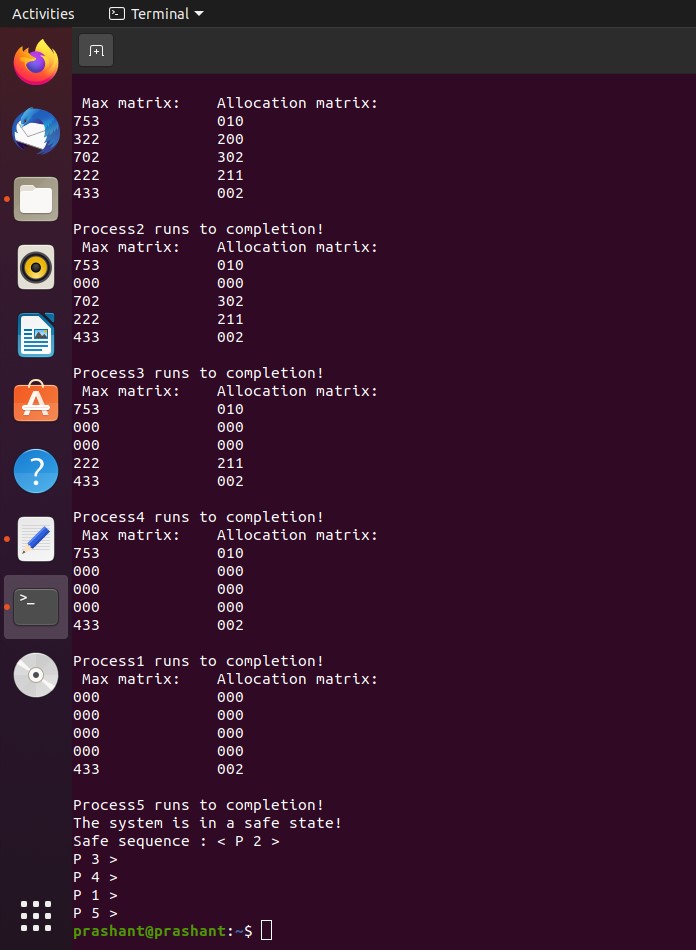
{

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

}

}

**OUTPUT**

****Text

Description automatically generated

Graphical user interface, text, application, Teams

Description automatically generated

**Date:22.05.2021**

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

#include<stdio.h>

#include<string.h>

int z=0,i=0,j=0,c=0; //c for length of string, i for stack, j for

//input and z for check function

char a[16],ac[20],stk[15],act[10]; //act[ ] for shift and ac[ ]for //reduce

void check(); // to check top of stack

void main()

{

printf("GRAMMER is E->E+T|T \n T->T\*F|F \n F->(E) \n F->id");

printf("enter input string\n");

scanf("%s",a);

c=strlen(a);

strcpy(act,"SHIFT->"); //initial configuration

printf("stack \t input \t action");

//printf("\n---------------------------------------\nâ€ );

for(i=0; j<c; i++,j++)

{

if(a[j]=='i' && a[j+1]=='d') //checks the input for"id"

{

stk[i]=a[j];

stk[i+1]=a[j+1]; //shift id to stack

stk[i+2]='\0';

a[j]=' '; //delete "id" from input array

a[j+1]=' ';

printf("\n$%s\t%s$\t%sid",stk,a,act);

//displays the first line SHIFT->id

check();

}

else

{

stk[i]=a[j];

stk[i+1]='\0';

a[j]=' ';

printf("\n$%s\t%s$\t%sSYMBOLS",stk,a,act);

//print SHIFT->symbols for inputs like +,\*

check();

}

}

if((strcmp(stk,"E")==0)) //at the end of input if stack holds "E"

printf("\n---------\n SUCCESS!!!!!!!!!!\n");

else

printf("\n-----------------\nERROR!!!!!!\n");

}

void check()

{

strcpy(ac,"REDUCE "); //dispaly REDUCE

for(z=0;z<c;z++)

if(stk[z]=='(' && stk[z+1]=='E' && stk[z+2]==')')

//if stack holds (E)

{

stk[z]='F'; //reduce to F

stk[z+1]='\0';

printf("\n$%s\t%s$\t%s",stk,a,ac);

i=i-2; //top of stack holds F

}

for(z=0;z<c;z++) // if stack holds id

if(stk[z]=='i' && stk[z+1]=='d')

{

stk[z]='F';

stk[z+1]='\0';

printf("\n$%s\t%s$\t%s",stk,a,ac);

j++; //move input pointer

}

for(z=0;z<c;z++) // if stack holds T\*F

{

if(stk[z]=='T' && stk[z+1]=='\*' && stk[z+2]=='F')

{

stk[z]='T';

stk[z+1]='\0';

printf("\n$%s\t%s$\t%s",stk,a,ac);

i=i-2;

}

else if(stk[z]=='F')

{

stk[z]='T';

printf("\n$%s\t%s$\t%s",stk,a,ac);

}

}

for(z=0;z<c;z++) //checks for stack E+T\*

{

if(stk[z]=='E' && stk[z+1]=='+' && stk[z+2]=='T'&&

stk[z+3]=='\*')

break;

if(stk[z]=='E' && stk[z+1]=='+' && stk[z+2]=='T'&&

a[j+1]=='\*')

//stack is E+T and ip is \*

break;

else if(stk[z]=='E' && stk[z+1]=='+' && stk[z+2]=='T')

//stack is E+T

{

stk[z]='E';

stk[z+1]='\0';

printf("\n$%s\t%s$\t%s",stk,a,ac);

i=i-2;

return;

}

}

for(z=0;z<c;z++)

{

if(stk[z]=='T')

if (a[j+1]=='\*')

break;

else if(stk[z+1]=='\*')

break;

else

{

stk[z]='E';

printf("\n$%s\t%s$\t%s",stk,a,ac);

}

}

}

**OUTPUT**

Text

Description automatically generated

**Date:02.06.2021**

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

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

char ip[20],stack[20];

int main()

{

char m[2][3][3]={ {"aBa","E","E"},

{"n","bB","E"} };

int size[2][3]= {3,1,1,1,2,1};

int i,j,k,n,row,col,flag=0;

int p,q,r;

printf("\nEnter the input string: ");

scanf("%s",ip);

strcat(ip,"$");

n=strlen(ip);

stack[0]='$';

stack[1]='A';

i=1;

j=0;

printf("PARSING TABLE:\n");

for(p=0;p<2;p++)

{

for(q=0;q<3;q++)

{

for(r=0;r<3;r++)

printf("%c",m[p][q][r]);

printf("\t");

}

printf("\n");

}

printf("\nStack\t\tInput\n");

printf("\_\_\_\_\_\_\_\_\t\_\_\_\_\_\_\_\_\_\n");

for(k=0;k<=i;k++)

printf("%c",stack[k]); /\* Initial stack\*/

printf("\t\t");

for(k=j;k<=n;k++)

printf("%c",ip[k]); /\* input\*/

printf("\n");

while((stack[i]!='$')&&(ip[j]!='$'))

{

if(stack[i]==ip[j])

{

i--;

j++;

for(k=0;k<=i;k++)

printf("%c",stack[k]); /\*stack content \*/

printf("\t\t");

for(k=j;k<=n;k++)

printf("%c",ip[k]); /\* input\*/

printf("\n");

}

switch(stack[i])

{

case 'A': row=0;

break;

case 'B': row=1;

break;

default:if ((stack[i]=='$')&&(ip[j]=='$'))

printf("\nSUCCESSFULL PARSING\n");

else

{

printf("\nUNSUCCESSFULL PARSING\n");

printf("ERROR-NO VALID MATCH\n");

}

exit(0);

}

switch(ip[j])

{

case 'a': col=0;

break;

case 'b': col=1;

break;

case '$': col=2;

break;

}

if(m[row][col][0]==ip[j])

{

for(k=size[row][col]-1;k>=0;k--)

{

stack[i]=m[row][col][k];

i++;

}

i--;

}

if(m[row][col][0]=='E')

{

if(i>0)

printf("\nERROR....\n");

else

flag=1;

exit(0);

}

if(m[row][col][0]=='n')

i--;

for(k=0;k<=i;k++)

printf("%c",stack[k]);

printf("\t\t");

for(k=j;k<=n;k++)

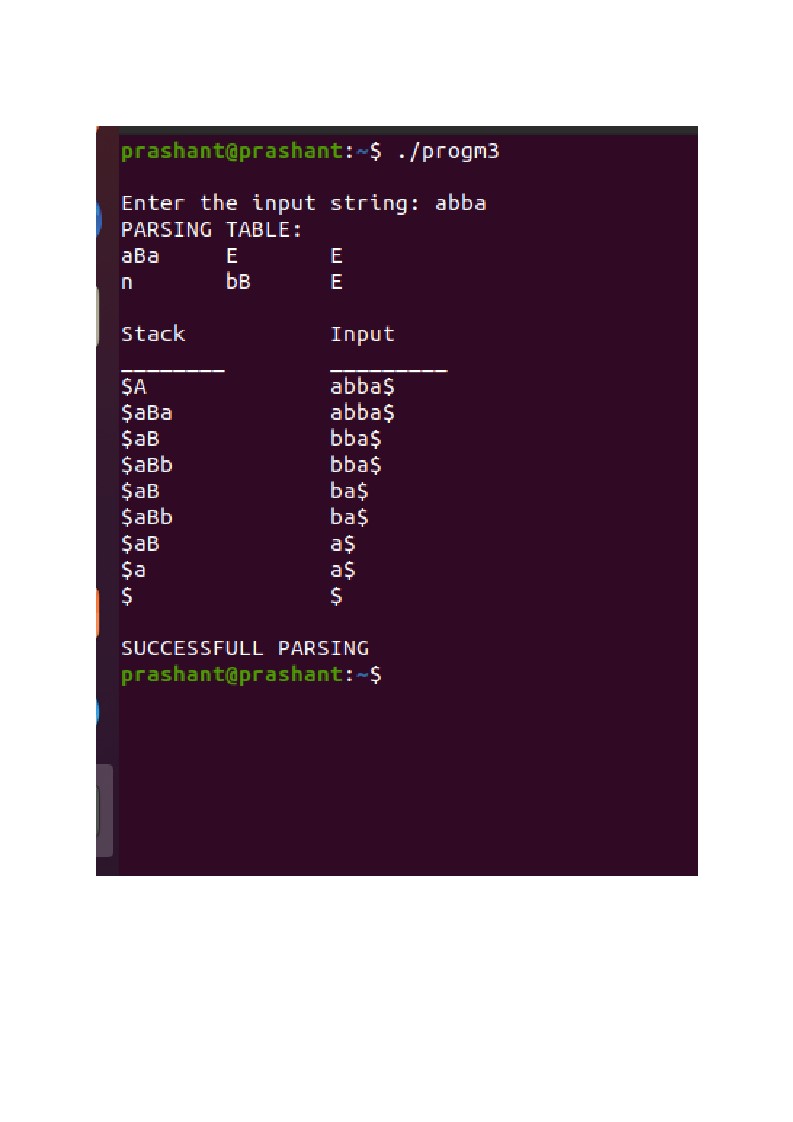
printf("%c",ip[k]);

printf("\n");

}

return 0;

}

**OUTPUT**

**Date: 05.06.2021**

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

#include<stdio.h>

#include<stdlib.h>

struct proc{

int id;

int arrival;

int brust;

int rem;

int wait;

int finish;

int turnaround;

float ratio;

}process[10];

struct proc temp;

int no;

int chkprocess(int);

int nextprocess();

void roundrobin(int,int,int[],int[]);

void srtf(int);

int main()

{

int n,tq,choice;

int bt[10],st[10],i,j,k;

for(;;)

{

printf("Enter the choice\n");

printf("1.RoundRobin\n2.SRT\n3.Exit\n");

scanf("%d",&choice);

switch(choice)

{

case 1:printf("RoundRobin scheduling algorithm\n");

printf("Enter Number of processes:\n");

scanf("%d",&n);

printf("Enter the brust time for sequences:\n");

for(i=0;i<n;i++)

{

scanf("%d",&bt[i]);

st[i]=bt[i];

}

printf("Enter the time quantum:");

scanf("%d",&tq);

roundrobin(n,tq,st,bt);

break;

case 2:printf("\n\n-------SHOREST REMAINING TIME NEXT----\n\n");

printf("\n\nEnter the number of processes:");

scanf("%d",&n);

srtf(n);

break;

case 3:exit(0);

}

}

}

void roundrobin(int n,int tq,int st[],int bt[])

{

int time=0;

int tat[10],wt[10],i,count=0,swt=0,stat=0,temp1,sq=0,jk;

float awt=0.0,atat=0.0;

while(1)

{

for(i=0,count=0;i<n;i++)

{

temp1=tq;

if(st[i]==0)

{

count++; //countvalue is incremented

continue;

}

if(st[i]>tq)

st[i]=st[i]-tq;

else

if(st[i]>=0)

{

temp1=st[i];

st[i]=0;

}

sq=sq+temp1;

tat[i]=sq;

}

if(n==count)

break;

}// end of while

for(i=0;i<n;i++)

{

wt[i]=tat[i]-bt[i];

swt=swt+wt[i];

stat=stat+tat[i];

}

awt=(float)swt/n;

atat=(float)stat/n;

printf("Process no brust time Wait time Turn around time \n");

for(i=0;i<n;i++)

printf("%d\t\t%d\t\t%d\t%d\n",i+1,bt[i],wt[i],tat[i]);

printf("Avg wait time is %f\nAvg turn around time is %f\n",awt,atat);

}

int chkprocess(int s)

{

int i;

for(i=1; i<=s; i++)

{

if(process[i].rem!=0)

return 1;

}

return 0;

}

int nextprocess()

{

int min,l,i;

min=32000;

for(i=1;i<=no;i++)

{

if(process[i].rem!=0 && process[i].rem<min)

{

min=process[i].rem;

l=i;

}

}

return l;

}

void srtf(int n)

{

int i,j,k,time=0;

float tavg,wavg;

for(i=1; i<=n; i++)

{

process[i].id = i;

printf("\n\n Enter the arrival time for process %d :",i);

scanf("%d",&(process[i].arrival));

printf("Enter the brust time for process %d:",i);

scanf("%d",&(process[i].brust));

process[i].rem = process[i].brust;

}

for(i=1;i<=n;i++)

{

for(j=i+1;j<=n;j++)

{

if(process[i].arrival > process[j].arrival)

{

temp = process[i];

process[i] = process[j];

process[j]= temp;

}

}

}

no=0;

j=1;

while(chkprocess(n) ==1)

{

if(process[no + 1].arrival==time)

{

while(process[no+1].arrival==time)

no++;

if(process[j].rem==0)

process[j].finish=time;

j=nextprocess();

}

if(process[j].rem!=0)

{

process[j].rem--;

for(i=1;i<=no;i++)

{

if(i!=j && process[i].rem!=0)

process[i].wait++;

}

}

else

{

process[j].finish=time;

j=nextprocess();

time--;

k=j;

}

time++;

}

process[k].finish = time;

printf("\n\n\t\t\t---SHORTEST REMAINING TIME FIRST---");

printf("\n\n Process Arrival Brust Waiting Finishing turnaround Tr/Tb\n\n");

printf("%5s%9s%7s%10s%8s%9s\n\n","id","time","time","time","time","time");

for(i=1;i<=n;i++)

{

process[i].turnaround=process[i].wait + process[i].brust;

process[i].ratio=(float)process[i].turnaround/(float)process[i].brust;

printf("%5d%8d%7d%8d%10d%9d%10.1f",process[i].id,process[i].arrival,process[i].brust,process[i].wait,process[i].finish,process[i].turnaround,process[i].ratio);

tavg=tavg+process[i].turnaround;

wavg=wavg+process[i].wait;

printf("\n\n");

}

tavg=tavg / n;

wavg=wavg / n;

printf("tavg=%f\n wavg=%f\n",tavg,wavg);

}

**OUTPUT**

Graphical user interface, text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Date:12.06.2021**

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

**T1 = -B**

**T2 = C + D**

**T3 = T1 + T2**

**A = T3**

#include<stdio.h>

#include<stdlib.h>

#include<ctype.h>

#include<string.h>

char tset[4][3][3]= { {"-","B","?"},

{"+","C","D"},

{"\*","0","1"},

{"=","A","2"}

};

int main()

{

int row,col;

printf("Input statement is:A = -B \* (C+D)\n");

printf("Intermediate code in three address form is:\nT1 = -B\nT2 = C + D\nT3 = T1 \* T2\nA=T3\n");

printf("Triples are:\n");

for(row=0;row<4;row++)

{

printf("%d\t", row);

for(col=0; col<3; col++)

printf("%c\t", tset[row][col][0]);

printf("\n");

}

printf("Machine code generated is:\n");

for(row=0;row<4;row++)

{

if (tset[row][2][0]=='?')

{

printf("\nLD R0,%s%s",tset[row][0],tset[row][1]);

}

else

{

if(tset[row][0][0]=='+')

{

printf("\nLD R1,%s",tset[row][1]);

printf("\nLD R2,%s",tset[row][2]);

printf("\nADD R1,R1,R2");

}

else

{

if(tset[row][0][0]=='\*')

printf("\nMUL R0,R0,R1");

else

printf("\nST %s,R0",tset[row][1]);

}

}

}

printf("\n");

return 0;

}

**OUTPUT**

Text

Description automatically generated

**Date:21.06.2021**

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

%{ /\* LEX program to recognize a valid arithmetic expression and count the identifiers and

operators \*/

int id=0, op=0, flag=0;

%}

%%

[0-9]+ { id++; printf("%s is an identifier\n", yytext); }

[+ \*] { op++; printf("%s is an operator\n", yytext); }

. { flag=1; }

\n { return; }

%%

main()

{

printf("Enter an arithmetic expression: ");

yylex();

if ( flag==0 && id==op+1 )

{

printf("Valid expression\n");

printf("The no of identifiers are %d\n", id);

printf("The no of operators are %d\n", op);

}

else

printf("Invalid expression\n");

}

**Date:27.06.2021**

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

|  |
| --- |
| %{ |
|  |  |
|  | %} |
|  |  |
|  | %% |
|  | "/\*"[a-zA-Z0-9' '\t\n]+"\*/" {} |
|  | "//".\* {} |
|  | %% |
|  |  |
|  | int main() |
|  | { |
|  | yyin=fopen("input.c","r"); |
|  | yyout=fopen("out.c","w"); |
|  | yylex(); |
|  | fclose(yyin); |
|  | fclose(yyout); |
|  | return 0; |
|  | } |

**Date:08.07.2021**

**1b. Write YACC program to evaluate arithmetic expression involving operators: +, -, \*,**

**and /**

%{ /\* YACC program to evaluate arithmetic expression involving operators +,-,\*,/\*/

#include "lex.yy.c"

%}

%token NUM

%left '+' '-'

%left '\*' '/'

%%

stmt: exp { printf("Value of expression = %d\n",$$); }

;

exp: exp '+' exp { $$=$1+$3; } |

exp '-' exp { $$=$1-$3; }

| exp '\*' exp { $$=$1\*$3; }

| exp '/' exp { if( $3==0 )

{ printf("Divide by zero error!\n"); yyerror();

}

else

$$=$1/$3;

}

| '(' exp ')' { $$=$2; }

| NUM { $$=$1; }

;

%%

main()

{

printf("Enter expression: ");

yyparse(); return;

}

int yyerror()

{

printf("Invalid expression\n");

exit(0);

}

%{ /\*Lexpgm\*/ #include

"y.tab.h" extern int

yylval;

%}

%%

[0-9]+ { yylval=atoi(yytext); return NUM; }

[\+\-\\*\/()] return( yytext[0] );

\n return(0);

. yyerror();

%%

**Date:20.07.2021**

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

%{

#include<stdio.h>

#include<stdlib.h>

%}

%token A B

%%

input:s'\n' {printf("Successful Grammar\n");exit(0);} s:

A s1 B| B s1: ; | A s1

%%

main()

{

printf("Enter A String\n"); yyparse();

}

int yyerror()

{

printf("Error \n"); exit(0);

}

%{

#include "y.tab.h"

%}

%%

a {return A;} b

{return B;}

[\n] return '\n';

%%

**Date:20.07.2021**

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

%{ /\*YACC program\*/ #include

"lex.yy.c"

int id=0, dig=0, key=0, op=0, lit=0, par=0, inv=0;

%}

%token DIGIT ID KEY OP LIT PAR INV

%%

input:

DIGIT input { dig++; }

| ID input { id++; }

| KEY input { key++; }

| OP input { op++; }

| LIT input { lit++; }

| PAR input { par++; }

| INV input { inv++; }

| DIGIT { dig++; }

| ID { id++; }

| KEY { key++; }

| OP { op++;}

| LIT { lit++; }

| PAR { par++; }

| INV { inv++; }

;

%%

main(int argc, char \*\*argv)

{

FILE \*f1 = fopen(argv[1], "r");

if(!f1) {

printf("File cannot be opened\n"); exit(0);

}

yyin = f1; do

{

yyparse();

} while(!feof(yyin)); printf("Numbers = %d\n Keywords = %d\n Identifiers = %d\n

Operators= %d\n Literals =

%d\n Parenthesis = %d\n", dig, key, id, op, lit, par);

}

void yyerror()

{

printf("Parse error! Message: "); exit(0);

}

%{

#include "y.tab.h" extern

yylval;

%}

%%

[ \t] ;

[(|)] {printf("%s -> parenthesis\n",yytext); return PAR;}

"\"".\*"\"" {printf("%s -> literal\n",yytext); return LIT;}

[+|-|\*|/|=|<|>] {printf("%s -> operator\n",yytext); return OP;}

[0-9]+|[0-9]\*[.][0-9]+ {printf("%s -> number\n",yytext); yylval =atoi(yytext); return

DIGIT;}

int|char|bool|float|void|for|do|while|if|else|return|void {printf("%s ->

keyword\n",yytext); return KEY;}

[a-zA-Z][a-zA-Z0-9\_]\* {printf("%s -> identifier\n",yytext); return ID;}

[0-9]+[a-zA-Z]\* {printf("%s -> invalid identifier\n",yytext); return INV;}

. ;

%%

#include<stdio.h> main()

{

int a1, 1b, sum=1.2;

printf("Enter values for a and b\n");

scanf("%d %d", a1, 1b); sum =

a1+1b;

printf("Sum = %d\n", sum);

}