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

%{

#include<stdio.h>

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

%}

%%

[a-zA-Z]+[0-9A-Za-z]\* {id++;printf("\n Identifier:");ECHO;}

[\+\-\\*\/\=] {op++;printf("\n Operartor:");ECHO;}

"(" {v++;}

")" {v--;}

";" {flag=1;}

.|\n {return 0;}

%%

main()

{

printf("Enter the expression");

yylex();

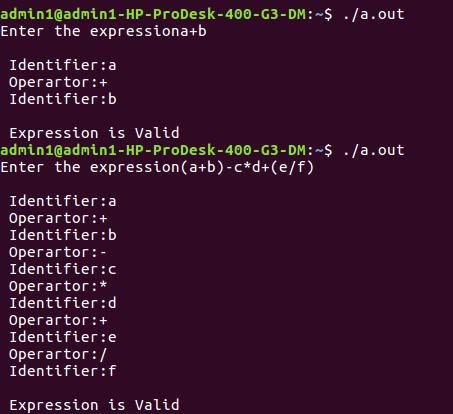
if((op+1) ==id && v==0 && flag==0)

printf("\n Expression is Valid\n");

else

printf("\n Expression is Invalid\n");

}

**Execution Steps:**

Lex <lexfilename.l>

cc lex.yy.c –ll

. /a.out <temp.txt>

**Output:**

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

%{

#include "y.tab.h"

extern yylval;

%}

%%

[0-9]+ {yylval=atoi(yytext);return num;} **/\* convert the string to number and send the value\*/**

[\+\-\\*\/] {return yytext[0];}

* {return yytext[0];}
* {return yytext[0];}

. {;}

\n {return 0;}

%%

**YACC Part**

%{

#include<stdio.h>

#include<stdlib.h>

%}

%token num

%left '+' '-'

%left '\*' '/'

%%

input:exp {printf("%d\n",$$);exit(0);}

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

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

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

|exp'/'exp { if($3==0){printf("Divide by Zero\n");exit(0);} else

$$=$1/$3;}

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

|num{$$=$1;};

%%

int yyerror()

{

printf("error");

exit(0);

}

int main()

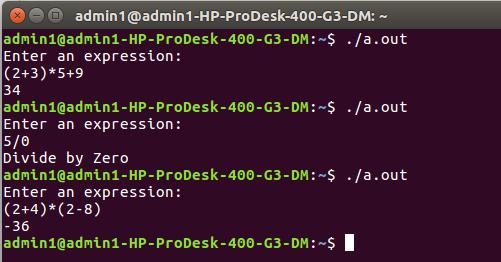
{

printf("Enter an expression:\n");

yyparse();

}

**Output:**



**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).**

**Lex Part**

%{

#include "y.tab.h"

%}

%%

a {return A;}

b {return B;}

[\n] return '\n';

%%

**YACC Part**

%{

#include<stdio.h>

#include<stdlib.h>

%}

%token A B

%%

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

s1: A s1 B| B

s1: ; | A s1

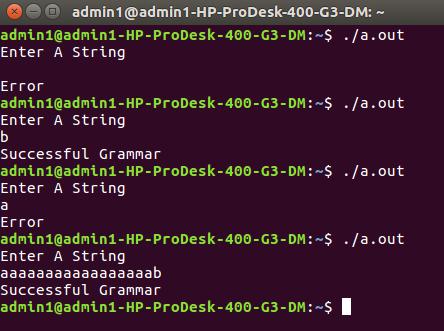
%%

main()

{

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

}



int yyerror()

{

printf("Error \n");

exit(0);

}

**Output:**

**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<conio.h>

#include<string.h>

void main()

{

char fin[10][20],st[10][20],ft[20][20],fol[20][20]; int a=0,e,i,t,b,c,n,k,l=0,j,s,m,p; printf("enter the no. of coordinates\n"); scanf("%d",&n);

printf("enter the productions in a grammar\n");

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

scanf("%s",st[i]);

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

fol[i][0]='\0';

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

{

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

{

j=3;

l=0;

a=0;

l1:if(!((st[i][j]>64)&&(st[i][j]<91)))

{

for(m=0;m<l;m++)

{

if(ft[i][m]==st[i][j])

goto s1;

}

ft[i][l]=st[i][j];

l=l+1;

s1:j=j+1;

}

else

{

if(s>0)

{

while(st[i][j]!=st[a][0])

{

a++;

}

b=0;

while(ft[a][b]!='\0')

{

for(m=0;m<l;m++)

{

if(ft[i][m]==ft[a][b])

goto s2;

}

ft[i][l]=ft[a][b];

l=l+1;

s2:b=b+1;

}

}

}

while(st[i][j]!='\0')

{

if(st[i][j]=='|')

{

j=j+1;

goto l1;

}

j=j+1;

}

ft[i][l]='\0';

}

}

printf("first pos\n");

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

printf("FIRS[%c]=%s\n",st[i][0],ft[i]);

fol[0][0]='$';

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

{

k=0;

j=3;

if(i==0)

l=1;

else

l=0;

k1:while((st[i][0]!=st[k][j])&&(k<n))

{

if(st[k][j]=='\0')

{

k++;

j=2;

}

j++;

}

j=j+1;

if(st[i][0]==st[k][j-1])

{

if((st[k][j]!='|')&&(st[k][j]!='\0'))

{

a=0;

if(!((st[k][j]>64)&&(st[k][j]<91)))

{

for(m=0;m<l;m++)

{

if(fol[i][m]==st[k][j])

goto q3;

}

q3:

fol[i][l]=st[k][j];

l++;

}

else

{

while(st[k][j]!=st[a][0])

{

a++;

}

p=0;

while(ft[a][p]!='\0')

{

if(ft[a][p]!='@')

{

for(m=0;m<l;m++)

{

if(fol[i][m]==ft[a][p])

goto q2;

}

fol[i][l]=ft[a][p];

l=l+1;

}

else

e=1;

q2:p++;

}

if(e==1)

{

e=0;

goto a1;

}

}

}

else

{

a1:c=0;

a=0;

while(st[k][0]!=st[a][0])

{

a++;

}

while((fol[a][c]!='\0')&&(st[a][0]!=st[i][0]))

{

for(m=0;m<l;m++)

{

if(fol[i][m]==fol[a][c])

goto q1;

}

fol[i][l]=fol[a][c];

l++;

q1:c++;

}

}

goto k1;

}

fol[i][l]='\0';

}

printf("follow pos\n");

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

printf("FOLLOW[%c]=%s\n",st[i][0],fol[i]);

printf("\n");

s=0;

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

{

j=3;

while(st[i][j]!='\0')

{

if((st[i][j-1]=='|')||(j==3))

{

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

{

fin[s][p]=st[i][p];

}

t=j;

for(p=3;((st[i][j]!='|')&&(st[i][j]!='\0'));p++)

{

fin[s][p]=st[i][j];

j++;

}

fin[s][p]='\0';

if(st[i][t]=='@')

{

b=0;

a=0;

while(st[a][0]!=st[i][0])

{

a++;

}

while(fol[a][b]!='\0')

{

printf("M[%c,%c]=%s\n",st[i][0],fol[a][b],fin[s]);

b++;

}

}

else if(!((st[i][t]>64)&&(st[i][t]<91)))

printf("M[%c,%c]=%s\n",st[i][0],st[i][t],fin[s]);

else

{

b=0;

a=0;

while(st[a][0]!=st[i][3])

{

a++;

}

while(ft[a][b]!='\0')

{

printf("M[%c,%c]=%s\n",st[i][0],ft[a][b],fin[s]);

b++;

}

}

s++;

}

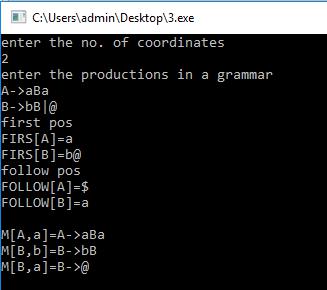
if(st[i][j]=='|')

j++;

}

}

getch();

}

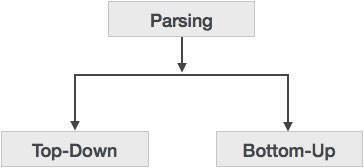
**Output:**

1. **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*.**

A parser is a compiler or interpreter component that breaks data into smaller elements for easy translation into another language. A parser takes input in the form of a sequence of tokens or program instructions and usually builds a data structure in the form of a parse tree or an abstract syntax tree.

A parser's main purpose is to determine if input data may be derived from the start symbol of the grammar.

Syntax analyzers follow production rules defined by means of context-free grammar. The way the production rules are implemented (derivation) divides parsing into two types: top-down parsing and bottom-up parsing.



**Top-down Parsing**

When the parser starts constructing the parse tree from the start symbol and then tries to transform the start symbol to the input, it is called top-down parsing.

* **Recursive descent parsing**: It is a common form of top-down parsing. It is called recursive as it usesrecursive procedures to process the input. Recursive descent parsing suffers from backtracking.
* **Backtracking**: It means, if one derivation of a production fails, the syntax analyzer restarts the processusing different rules of same production. This technique may process the input string more than once to determine the right production.

**Bottom-up Parsing**

Bottom-up parsing starts with the input symbols and tries to construct the parse tree up to the start

symbol.

**Shift-reduce Parsing (Bottom-up Parsing)**

Shift-reduce parsing attempts to construct a parse tree for an input string beginning at the leaves and working up towards the root. In other words, it is a process of “reducing” (opposite of deriving a symbol using a production rule) a string w to the start symbol of a grammar. At every (reduction) step, a particular substring matching the RHS of a production rule is replaced by the symbol on the LHS of the production.

A general form of shift-reduce parsing is **LR** (scanning from **L**eft to right and using **R**ight-most derivation in reverse) parsing, which is used in a number of automatic parser generators like Yacc, Bison, etc.

#include<stdio.h>

#include<conio.h>

#include<string.h>

int k=0,z=0,i=0,j=0,c=0;

char a[16],ac[20],stk[15],act[10];

void check();

void main()

{

puts("GRAMMAR is E->E+E \n E->E\*E \n E->(E) \n E->id");

puts("enter input string ");

gets(a);

c=strlen(a);

strcpy(act,"SHIFT->");

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

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

{

if(a[j]=='i' && a[j+1]=='d')

{

stk[i]=a[j];

stk[i+1]=a[j+1];

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

a[j]=' ';

a[j+1]=' ';

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

check();

}

else

{

stk[i]=a[j];

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

a[j]=' ';

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

check();

}

}

getch();

}

void check()

{

strcpy(ac,"REDUCE TO E");

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

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

{

stk[z]='E';

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

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

j++;

}

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

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

{

stk[z]='E';

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

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

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

i=i-2;

}

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

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

{

stk[z]='E';

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

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

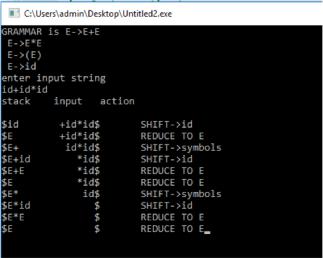
printf("\n$%s\t%s$\t%s",stk,a,ac);

i=i-2;

}

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

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

{

stk[z]='E';

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

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

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

i=i-2;

}

}

**Output:**

1. **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>

char op[2],arg1[5],arg2[5],result[5]; void main()

{

FILE \*fp1,\*fp2; fp1=fopen("input.txt","r"); fp2=fopen("output.txt","w");

while(!feof(fp1))

{

fscanf(fp1,"%s%s%s%s",result,arg1,op,arg2);

if(strcmp(op,"+")==0)

{

fprintf(fp2,"\nMOV R0,%s",arg1);

fprintf(fp2,"\nADD R0,%s",arg2);

fprintf(fp2,"\nMOV %s,R0",result);

}

if(strcmp(op,"\*")==0)

{

fprintf(fp2,"\nMOV R0,%s",arg1);

fprintf(fp2,"\nMUL R0,%s",arg2);

fprintf(fp2,"\nMOV %s,R0",result);

}

if(strcmp(op,"-")==0)

{

fprintf(fp2,"\nMOV R0,%s",arg1);

fprintf(fp2,"\nSUB R0,%s",arg2);

fprintf(fp2,"\nMOV %s,R0",result);

}

if(strcmp(op,"/")==0)

{

fprintf(fp2,"\nMOV R0,%s",arg1);

fprintf(fp2,"\nDIV R0,%s",arg2);

fprintf(fp2,"\nMOV %s,R0",result);

}

if(strcmp(op,"=")==0)

{

fprintf(fp2,"\nMOV R0,%s",arg1);

fprintf(fp2,"\nMOV %s,R0",result);

}

}

fclose(fp1);

fclose(fp2);

getch();

}

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Output:** | | |  |  |  |  |
| **input.txt** | |
|  | |
| T1 -B = ? |  |
| T2 C + D |  |
| T3 T1 \* T2 |  |
| A T3 = ? |  |
| **output.txt** | |
|  | |
| MOV R0,-B |  |
| MOV T1,R0 |  |
| MOV R0,C |  |
| ADD R0,D |  |
| MOV T2,R0 |  |
| MOV R0,T1 |  |
|
| MUL R0,T2 |  |
| MOV T3,R0 |  |
| MOV R0,T3 |  |
| MOV A,R0 |  |

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

%{

#include<stdio.h>

int c\_count=0;

%}

%%

"/\*"[^\*/]\*"\*/" {c\_count++;} **/\*for single and multiple line comments\*/**

"//".\* {c\_count++;} **/\*for single line comments\*/**

%%

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

{

FILE \*f1,\*f2;

if(argc>1) **/\*Pass two filenames for execution\*/**

{

f1=fopen(argv[1],"r"); **/\*open first file for reading\*/**

if(!f1) **/\*not able to open file\*/**

{

printf("file error \n");

exit(1);

}

yyin=f1;

f2=fopen(argv[2],"w"); **/\*open second file for writing\*/**

if(!f2) **/\*not able to open file\*/**

{

printf("Error");

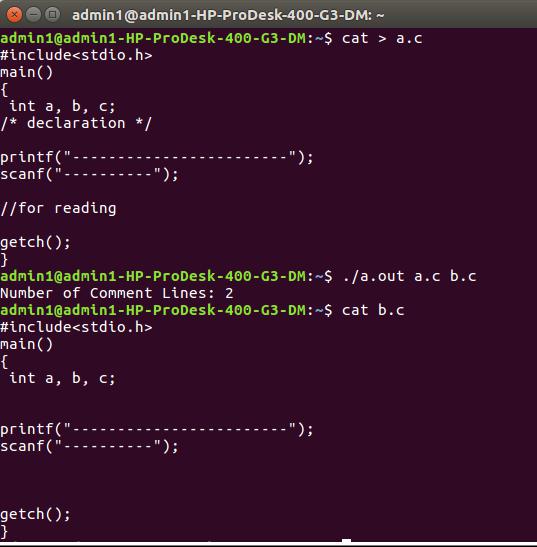
exit(1);

}

yyout=f2;

yylex();

printf("Number of Comment Lines: %d\n",c\_count);

}

return 0;

}

**OUTPUT:**

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

**Lex File**

%{

#include <stdio.h>

#include "y.tab.h"

extern yylval;

%}

%%

[ \t] ;

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

[0-9]+ {yylval = atoi(yytext); printf("numbers is %d\n",yylval); return DIGIT;}

int|char|bool|float|void|for|do|while|if|else|return|void {printf("keyword is

%s\n",yytext);return KEY;}

[a-zA-Z0-9]+ {printf("identifier is %s\n",yytext);return ID;}

. ;

%%

**Yacc File**

%{

#include <stdio.h>

#include <stdlib.h>

int id=0, dig=0, key=0, op=0;

%}

%token DIGIT ID KEY OP

%%

input:

DIGIT input { dig++; }

| ID input { id++; }

| KEY input { key++; }

| OP input {op++;}

| DIGIT { dig++; }

| ID { id++; }

| KEY { key++; }

| OP { op++;}

;

%%

#include <stdio.h>

extern int yylex();

extern int yyparse();

extern FILE \*yyin;

main() {

FILE \*myfile = fopen("sam\_input.c", "r");

if (!myfile) {

printf("I can't open sam\_input.c!");

return -1;

}

yyin = myfile;

do {

yyparse();

} while (!feof(yyin));

printf("numbers = %d\nKeywords = %d\nIdentifiers = %d\noperators = %d\n",

dig, key,id, op);

}

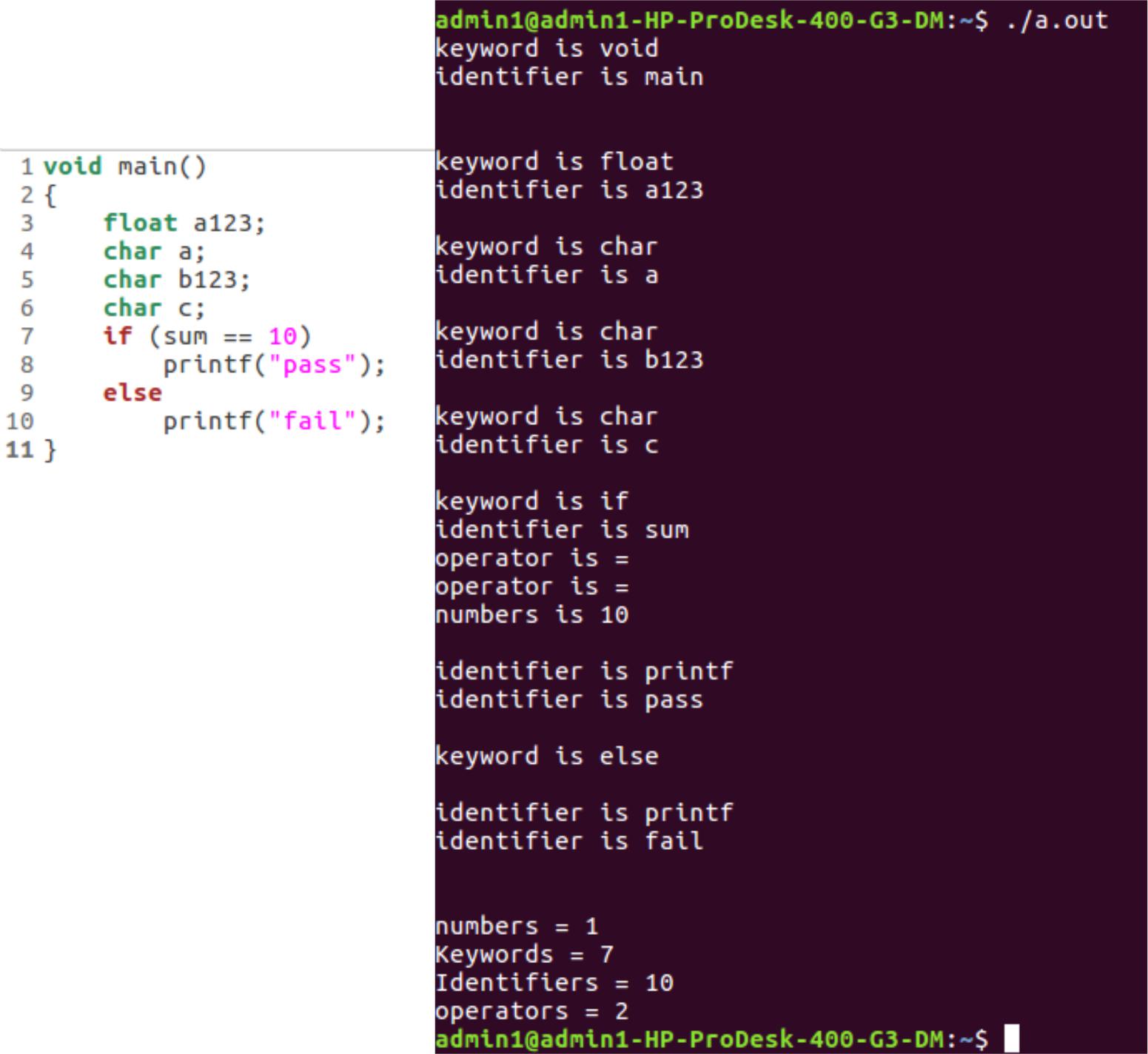
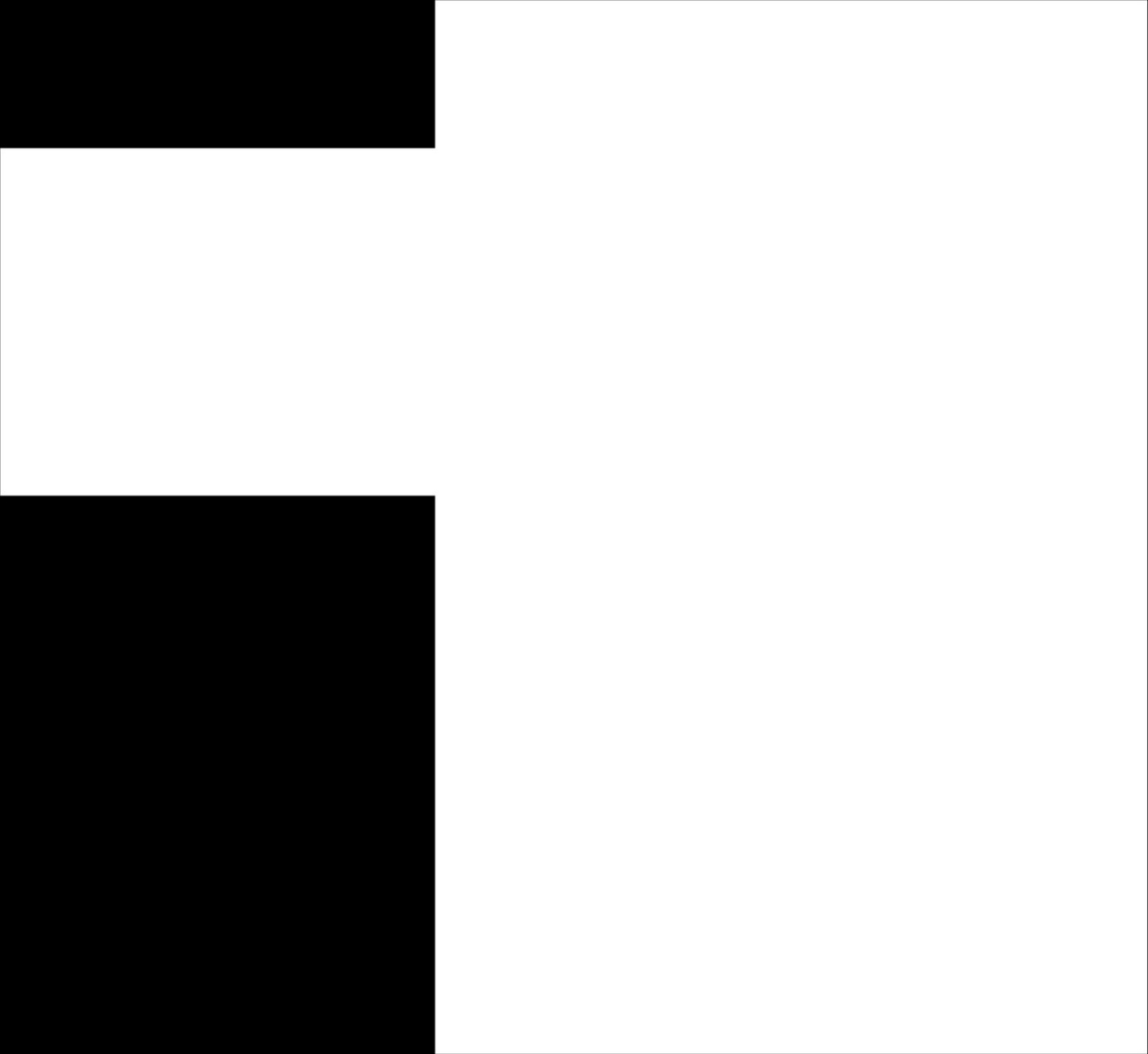
void yyerror() {

printf("EEK, parse error! Message: ");

exit(-1);

}

**Output :**

**Input file**

1. 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 RRalgorithm.

Round-robin (RR) is one of the algorithms employed by [process](https://en.wikipedia.org/wiki/Process_scheduler) and [network](https://en.wikipedia.org/wiki/Network_scheduler) [schedulers](https://en.wikipedia.org/wiki/Network_scheduler) in [computing.](https://en.wikipedia.org/wiki/Computing) As the term is generally used, [time slices](https://en.wikipedia.org/wiki/Preemption_(computing)#Time_slice) (also known as time quanta) are assigned to each process in equal portions and in circular order, handling all processes without [priority](https://en.wiktionary.org/wiki/priority) (also known as [cyclic executive)](https://en.wikipedia.org/wiki/Cyclic_executive). Round-robin scheduling is simple, easy to implement, and [starvation](https://en.wikipedia.org/wiki/Resource_starvation)-free. Round-robin scheduling can also be applied to other scheduling problems, such as data packet scheduling in computer networks. It is an [operating system](https://en.wikipedia.org/wiki/Operating_system) concept.

The name of the algorithm comes from the [round-robin](https://en.wikipedia.org/wiki/Round-robin_(disambiguation)) principle known from other fields, where each person takes an equal share of something in turn.

#include<stdio.h>

struct proc

{

int id;

int arrival;

int burst;

int rem;

int wait;

int finish;

int turnaround;

float ratio;

}process[10]; //structure to hold the process information

struct proc temp;

int no;

int chkprocess(int);

int nextprocess();

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

void srtf(int);

main()

{

int n,tq,choice;

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

for(; ;)

{

printf("Enter the choice \n");

printf(" 1. Round Robin\n 2. SRT\n 3. Exit \n");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("Round Robin scheduling algorithm\n"); printf("Enter number of processes:\n");

scanf("%d",&n);

printf("Enter burst time for sequences:");

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

{

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

st[i]=bt[i]; //service time

}

printf("Enter time quantum:");

scanf("%d",&tq);

roundrobin(n,tq,st,bt);

break;

case 2:

printf("\n \n ---SHORTEST REMAINING TIME NEXT---\n \n "); printf("\n \n Enter the number of processes: "); scanf("%d", &n);

srtf(n);

break;

case 3: exit(0);

}// end of switch

**}// end of for**

}**//end of main()**

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

{

int time=0;

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

float awt=0.0,atat=0.0;

while(1)

{

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

{

temp1=tq;

if(st[i]==0) **// when service time of a process equals zero then** **//count value is incremented**

{

count++;

continue;

}

if(st[i]>tq) **// when service time of a process greater than time** **//quantum then time**

st[i]=st[i]-tq**;** **//quantum value subtracted from** service time

else

if(st[i]>=0)

{

temp1=st[i]; **// temp1 stores the service time of a process**

st[i]=0; **// making service time equals 0**

}

sq=sq+temp1; **// utilizing temp1 value to calculate turnaround time**

tat[i]=sq; **// turn around time**

} //end of for

if(n==count) **// it indicates all processes have completed their task**

**because the count value**

break; **// incremented when service time equals 0**

} //end of while

for(i=0;i<n;i++) **// to calculate the wait time and turnaround time of each**

**process**

{

wt[i]=tat[i]-bt[i]; **// waiting time calculated from the turnaround time - burst**

**time**

swt=swt+wt[i]; **// summation of wait time**

stat=stat+tat[i]; **// summation of turnaround time**

}

awt=(float)swt/n; **// average wait time**

atat=(float)stat/n; **// average turnaround time**

printf("Process\_no Burst time Wait time Turn around time\n");

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

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

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

}**// end of Round Robin**

int chkprocess(int s) **// function to check process remaining time is zero or not**

{

int i;

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

{

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

return 1;

}

return 0;

} // end of chkprocess

int nextprocess() **// function to identify the next process to be executed**

{

int min, l, i;

min = 32000; **//any limit assumed**

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

{

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

{

min = process[i].rem;

l = i;

}

}

return l;

} // end of nextprocess

void srtf(int n)

{

int i,j,k,time=0;

float tavg,wavg;

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

{

process[i].id = i;

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

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

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

scanf("%d", &(process[i].burst)); process[i].rem = process[i].burst;

}

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

{

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

{

if(process[i].arrival > process[j].arrival) // sort arrival time of a

process

{

temp = process[i];

process[i] = process[j];

process[j] = temp;

}

}

}

no = 0;

j = 1;

while(chkprocess n) == 1)

{

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

{

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

no++;

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

process[j].finish=time;

j = nextprocess();

}

if(process[j].rem != 0) **// to calculate the waiting time of a process**

{

process[j].rem--;

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

{

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

process[i].wait++;

}

}

else

{

process[j].finish = time;

j=nextprocess();

time--;

k=j;

}

time++;

}

process[k].finish = time;

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

printf("\n\n Process Arrival Burst Waiting Finishing turnaround Tr/Tb \n"); printf("%5s %9s %7s %10s %8s %9s\n\n", "id", "time", "time", "time", "time", "time");

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

{

process[i].turnaround = process[i].wait + process[i].burst; // calc of turnaround process[i].ratio = (float)process[i].turnaround / (float)process[i].burst;

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

tavg=tavg+ process[i].turnaround; **//summation of turnaround time**

wavg=wavg+process[i].wait; **// summation of waiting time**

printf("\n\n");

}

tavg=tavg/n; // average turnaround time

wavg=wavg/n; // average wait time

printf("tavg=%f\t wavg=%f\n",tavg,wavg); }**// end of srtf**

**Output:**

Enter the choice

1. Round Robin 2) SRT
2. Exit

1

Round Robin scheduling algorithm

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter number of processes:3

Enter burst time for sequences:24

3

3

|  |  |  |  |
| --- | --- | --- | --- |
| Enter time quantum:4 | |  |  |
| Process\_no | Burst time | Wait time | Turnaround time |
| 1 | 24 | 6 | 30 |
| 2 | 3 | 4 | 7 |
| 3 | 3 | 7 | 10 |

Avg wait time is 5.666667

Avg turnaround time is 15.666667

Enter the choice

1. Round Robin 2) SRT
2. Exit

2

---SHORTEST REMAINING TIME NEXT---

Enter the number of processes: 4

Enter the arrival time for process 1: 0

Enter the burst time for process 1: 8

Enter the arrival time for process 2: 1

Enter the burst time for process 2: 4

Enter the arrival time for process 3: 2

Enter the burst time for process 3: 9

Enter the arrival time for process 4: 3

Enter the burst time for process 4: 5

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 24 | 6 | 30 |
| 2 | 3 | 4 | 7 |
| 3 | 3 | 7 | 10 |

---SHORTEST REMAINING TIME FIRST---

Enter the number of processes: 4

Enter the arrival time for process 1: 0

Enter the burst time for process 1: 8

Enter the arrival time for process 2: 1

Enter the burst time for process 2: 4

Enter the arrival time for process 3: 2

Enter the burst time for process 3: 9

Enter the arrival time for process 4: 3

Enter the burst time for process 4: 5

---SHORTEST REMAINING TIME NEXT---

Process Arrival Burst Waiting Finishing turnaround Tr/Tb

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| id | time | time | time | time | time | time |
| 1 | 0 | 8 | 9 | 17 | 17 | 2.1 |
| 2 | 1 | 4 | 0 | 5 | 4 | 1.0 |
| 3 | 2 | 9 | 15 | 26 | 24 | 2.7 |
| 4 | 3 | 5 | 2 | 10 | 7 | 1.4 |

tavg=13.000000

wavg=6.500000

Using OpenMP

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

The **Banker's algorithm**, sometimes referred to as the **detection algorithm**, is a [resource](https://en.wikipedia.org/wiki/Resource_allocation) [allocation](https://en.wikipedia.org/wiki/Resource_allocation) and [deadlock](https://en.wikipedia.org/wiki/Deadlock) avoidance [algorithm](https://en.wikipedia.org/wiki/Algorithm) developed by [Edsger Dijkstra](https://en.wikipedia.org/wiki/Edsger_Dijkstra) that tests for safety by simulating the allocation of predetermined maximum possible amounts of all [resources,](https://en.wikipedia.org/wiki/Resource_(computer_science)) and then makes an "s-state" check to test for possible deadlock conditions for all other pending activities, before deciding whether allocation should be allowed to continue.

The algorithm was developed in the design process for the [operating system](https://en.wikipedia.org/wiki/Operating_system) and originally described (in [Dutch)](https://en.wikipedia.org/wiki/Dutch_language) in EWD108. When a new process enters a system, it must declare the maximum number of instances of each resource type that it may ever claim; clearly, that number may not exceed the total number of resources in the system. Also, when a process gets all its requested resources it must return them in a finite amount of time.

#include <stdio.h>

#include <stdlib.h>

int main()

{

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

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

count = 0;

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

scanf("%d", &p);

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

completed[i] = 0;

printf("\n\nEnter the no of resources : ");

scanf("%d", &r);

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

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

{

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

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

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

}

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

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

{

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

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

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

}

printf("\n\nEnter the Available Resources : "); for(i = 0; i < r; i++)

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

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

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

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

do

{

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

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

{

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

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

printf("\t\t");

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

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

printf("\n");

}

process = -1;

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

{

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

{

process = i ;

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

{

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

{

process = -1;

break;

}

}

}

if(process != -1)

break;

}

if(process != -1)

{

printf("\nProcess %d runs to completion!", process + 1); safeSequence[count] = process + 1; count++;

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

{

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

alloc[process][j] = 0;

Max[process][j] = 0;

completed[process] = 1;

}

}

}

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

if(count == p)

{

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

printf("Safe Sequence : < ");

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

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

printf(">\n");

}

else

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

}

**Output:**

Enter the no of processes : 5

Enter the no of resources : 3

Enter the Max Matrix for each process :

For process 1 : 7

5

3

For process 2 : 3

2

2

For process 3 : 7

0

2

For process 4 : 2

2

2

For process 5 : 4

3

3

Enter the allocation for each process :

For process 1 : 0

1

0

For process 2 : 2

0

0

For process 3 : 3

0

2

For process 4 : 2

1

1

For process 5 : 0

0

2

Enter the Available Resources : 3

|  |  |  |  |
| --- | --- | --- | --- |
| 3 |  |  | |
| 2 |  |  | |
| Max matrix: | | Allocation matrix: | |
| 7 5 3 | 0 | 1 0 | |
| 3 2 2 | 2 | 0 0 | |
| 7 0 2 | 3 | 0 2 | |
| 2 2 2 | 2 | 1 1 | |
| 4 3 3 | 0 | 0 2 | |
| Process 2 runs to completion! | | | |
| Max matrix: | | Allocation matrix: | |
| 7 5 3 | 0 | 1 0 | |
| 0 0 0 | 0 | 0 0 | |
| 7 0 2 | 3 | 0 2 | |
| 2 2 2 | 2 | 1 1 | |
| 4 3 3 | 0 0 2 | | | |
| Process 3 runs to completion! | | | | |
| Max matrix: | | | Allocation matrix: | |
| 7 5 3 | 0 1 0 | | | |
| 0 0 0 | 0 0 0 | | | |
| 0 0 0 | 0 0 0 | | | |
| 2 2 2 | 2 1 1 | | | |
| 4 3 3 | 0 0 2 | | | |
| Process 4 runs to completion! | | | | |
| Max matrix: | | | Allocation matrix: | |
| 7 5 3 | 0 1 0 | | | |
| 0 0 0 | 0 0 0 | | | |
| 0 0 0 | 0 0 0 | | | |
| 0 0 0 | 0 0 0 | | | |
| 4 3 3 | 0 0 2 | | | |
| Process 1 runs to completion! | | | | |
| Max matrix: | | | Allocation matrix: | |
| 0 0 0 | 0 0 0 | | | |
| 0 0 0 | 0 0 0 | | | |
| 0 0 0 | 0 0 0 | | | |
| 0 0 0 | 0 0 0 | | | |
| 4 3 3 | 0 0 2 | | | |

Process 5 runs to completion!

The system is in a safe state!!

Safe Sequence: < 2 3 4 1 5 >

1. 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.

In a [computer](https://en.wikipedia.org/wiki/Computer) [operating system](https://en.wikipedia.org/wiki/Operating_system) that uses [paging](https://en.wikipedia.org/wiki/Paging) for [virtual memory](https://en.wikipedia.org/wiki/Virtual_memory) [management,](https://en.wikipedia.org/wiki/Memory_management) **page** **replacement algorithms** decide which memory pages to page out, sometimes called swap out, or writeto disk, when a [page](https://en.wikipedia.org/wiki/Page_(computer_memory)) of memory needs to be allocated. [Page replacement](https://en.wikipedia.org/wiki/Paging) happens when a requested page is not in memory [(page fault)](https://en.wikipedia.org/wiki/Page_fault) and a free page cannot be used to satisfy the allocation, either because there are none, or because the number of free pages is lower than some threshold.

When the page that was selected for replacement and paged out is referenced again it has to be paged in (read in from disk), and this involves waiting for I/O completion. This determines the *quality* of the page replacement algorithm: the less time waiting for page-ins, the better the algorithm. A page replacement algorithm looks at the limited information about accesses to the pages provided by hardware, and tries to guess which pages should be replaced to minimize the total number of page misses, while balancing this with the costs (primary storage and processor time) of the algorithm itself.

The page replacing problem is a typical [online problem](https://en.wikipedia.org/wiki/Online_problem) from the competitive analysis perspective in the sense that the optimal deterministic algorithm is known.

#include<stdio.h>

#include<stdlib.h>

void FIFO(char [ ],char [ ],int,int);

void lru(char [ ],char [ ],int,int);

void opt(char [ ],char [ ],int,int);

int main()

{

int ch,YN=1,i,l,f;

char F[10],s[25];

printf("\n\n\tEnter the no of empty frames: "); scanf("%d",&f);

printf("\n\n\tEnter the length of the string: ");

scanf("%d",&l);

printf("\n\n\tEnter the string: ");

scanf("%s",s);

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

F[i]=-1;

do

{

printf("\n\n\t\*\*\*\*\*\*\*\*\*\*\* MENU \*\*\*\*\*\*\*\*\*\*\*"); printf("\n\n\t1:FIFO\n\n\t2:LRU \n\n\t4:EXIT"); printf("\n\n\tEnter your choice: "); scanf("%d",&ch);

switch(ch)

{

case 1:

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

{

F[i]=-1;

}

FIFO(s,F,l,f);

break;

case 2:

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

{

F[i]=-1;

}

lru(s,F,l,f);

break;

case 4:

exit(0);

}

printf("\n\n\tDo u want to continue IF YES PRESS 1\n\n\tIF NO PRESS 0 : "); scanf("%d",&YN);

}while(YN==1);return(0);

}

//FIFO

void FIFO(char s[],char F[],int l,int f)

{

int i,j=0,k,flag=0,cnt=0;

printf("\n\tPAGE\t FRAMES\t FAULTS");

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

{

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

{

if(F[k]==s[i])

flag=1;

}

if(flag==0)

{

printf("\n\t%c\t",s[i]);

F[j]=s[i];

j++;

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

{

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

}

printf("\tPage-fault%d",cnt);

cnt++;

}

else

{

flag=0;

printf("\n\t%c\t",s[i]);

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

{

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

}

printf("\tNo page-fault");

}

if(j==f)

j=0;

}

}

//LRU

void lru(char s[],char F[],int l,int f)

{

int i,j=0,k,m,flag=0,cnt=0,top=0;

printf("\n\tPAGE\t FRAMES\t FAULTS");

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

{

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

{

if(F[k]==s[i])

{

flag=1;

break;

}

}

printf("\n\t%c\t",s[i]);

if(j!=f && flag!=1)

{

F[top]=s[i];

j++;

if(j!=f)

top++;

}

else

{

if(flag!=1)

{

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

{

F[k]=F[k+1];

}

F[top]=s[i];

}

if(flag==1)

{

for(m=k;m<top;m++)

{

F[m]=F[m+1];

}

F[top]=s[i];

}

}

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

{

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

}

if(flag==0)

{

printf("\tPage-fault%d",cnt);

cnt++;

}

else

printf("\tNo page fault");

flag=0;

}

}

**Output:**

Enter the no of empty frames: 3

Enter the length of the string: 5

Enter the string: hello

\*\*\*\*\*\*\*\*\*\*\* MENU \*\*\*\*\*\*\*\*\*\*\*

1:FIFO

2:LRU

4:EXIT

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Enter your choice: 1 | | | |  |
| PAGE | FRAMES | | | FAULTS |
| H | h |  |  | Page-fault 0 |
| E | h | e |  | Page-fault 1 |
| L | h | e | l | Page-fault 2 |
| L | h | e | l | No page-fault |
| O | o | e | l | Page-fault 3 |

Do u want to continue IF YES PRESS 1

IF NO PRESS 0 : 1

\*\*\*\*\*\*\*\*\*\*\* MENU \*\*\*\*\*\*\*\*\*\*\*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1:FIFO |  |  |  |  |
| 2:LRU |  |  |  |  |
| 4:EXIT |  |  |  |  |
| Enter your choice: 2 | | | |  |
| PAGE | FRAMES | | | FAULTS |
| h | h |  |  | Page-fault 0 |
| e | h | e |  | Page-fault 1 |
| l | h | e | l | Page-fault 2 |
| l | h | e | l | No page fault |
| o | e | l | o | Page-fault 3 |

Do u want to continue IF YES PRESS 1

IF NO PRESS 0 : 1

\*\*\*\*\*\*\*\*\*\*\* MENU \*\*\*\*\*\*\*\*\*\*\*

1:FIFO

2:LRU

4:EXIT

Enter your choice: 4

**Viva Questions**

* **Define system software**.

System software is computer software designed to operate the computer hardware and to provide a platform for running application software. Eg: operating system, assembler, and loader.

* **What is an Assembler?**

Assembler for an assembly language, a computer program to translate between lower-level representations of computer programs.

* **Explain lex and yacc tools**
  + Lex: - scanner that can identify those tokens
  + Yacc: - parser.yacc takes a concise description of a grammar and produces a C routine that can parse that grammar.
* **Explain yyleng?**

Yyleng-contains the length of the string our lexer recognizes.

* **What is a Parser?**

A Parser for a Grammar is a program which takes in the Language string as it's input and produces either a corresponding Parse tree or an Error.

* **What is the Syntax of a Language?**

The Rules which tells whether a string is a valid Program or not are called the Syntax.

* **What is the Semantics of a Language?**

The Rules which gives meaning to programs are called the Semantics of a Language.

* **What are tokens?**

When a string representing a program is broken into sequence of substrings, such that each substring represents a constant, identifier, operator, keyword etc of the language, these substrings are called the tokens of the Language.

* **What is the Lexical Analysis?**

The Function of a lexical Analyzer is to read the input stream representing the Source program, one character at a time and to translate it into valid tokens.

* **How can we represent a token in a language?**

The Tokens in a Language are represented by a set of Regular Expressions. A regular expression specifies a set of strings to be matched. It contains text characters and operator characters. The Advantage of using regular expression is that a recognizer can be automatically generated.

* **How are the tokens recognized?**

The tokens which are represented by an Regular Expressions are recognized in an input string by means of a state transition Diagram and Finite Automata.

* **Are Lexical Analysis and Parsing two different Passes?**

These two can form two different passes of a Parser. The Lexical analysis can store all the recognized tokens in an intermediate file and give it to the Parser as an input. However it is more convenient to have the lexical Analyzer as a co routine or a subroutine which the Parser calls whenever it requires a token.