System Software and Operating System Lab-18CSL67

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.

Source Code

```
%{
#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 operator:"); ECHO;}
"(" {v++;}
")" {v--;}
";" {flag = 1;}
.|\n {;}
%%
int main()
{
printf("Enter the expression: \n");
yylex();
if((op+1)==id && v==0 && flag==0)
{
printf("\n Expression is valid\n");
printf("The no. of identifiers are : %d\n",id);
printf("The no. of operators are: %d\n",op);
}
else
printf("\n Expression is invalid\n");
}
```

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

Source Code:

LEX part:

```
%{
#include "y.tab.h"
extern yylval;
%}
%%
[0-9]+ {yylval=atoi(yytext); return num;}
[\+\-\*\/] {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();
}
```

```
%{
#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();
}
```

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 b (note: input n value).

Source Code

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);}
s: A s1 B|B
s1: ; | A s1
%%
int yyerror()
{
printf("Error\n");
exit(0);
}
main()
{
printf("Enter a string:\n");
yyparse();
}
```

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

```
#include<stdlib.h>
#include<string.h>
#include<stdio.h>
char prod[3][10]={"A->aBa","B->bB","B->@"}, input[10],stack[25];
int top=-1; int j=0,k,l;
void push(char item)
{
        stack[++top]=item;
}
void pop()
{
        top=top-1;
}
void display()
{
        int j;
        for(j=top;j>=0;j--)
                printf("%c",stack[j]);
}
void stackpush(char p)
{
        if(p=='A')
        {
                pop();
                for(j=strlen(prod[0])-1;j>=3;j--)
                        push(prod[0][j]);
        }
        else
        {
                pop();
                for(j=strlen(prod[1])-1;j>=3;j--)
                        push(prod[1][j]);
        }
}
void main()
{
        char c; int i;
        printf("first(A)={a}\t");
        printf("follow(A)={$}\n");
        printf("first(B)={b,@}\t");
        printf("follow(B)={a}\n\n");
        printf("\t a \t b \t $ \n");
        printf("A\t%s\n",prod[0]);
        printf("B\t%s\t%s\n",prod[2],prod[1]);
        printf("enter the input string terminated with $ to parse:-");
        scanf("%s",input);
        for(i=0;input[i]!='\0';i++)
```

```
{
        if((input[i]!='a')&&(input[i]!='b')&&(input[i]!='$'))
        {
                printf("invalid string");
                exit(0);
        }
}
if(input[i-1]!='$')
{
        printf("\n\nInput string entered without end marker $");
        exit(0);
}
push('$');
push('A');
i=0;
printf("\n\n");
printf("stack\tInput\taction");
printf("\n---------\n");
while(i!=strlen(input)&&stack[top]!='$')
{
        printf("\n");
        for(l=top;l>=0;l--)
                printf("%c",stack[l]);
        printf("\t");
        for(l=i;l<strlen(input);l++)
                printf("%c",input[l]);
        printf("\t");
        if(stack[top]=='A')
        {
                printf("A->aBa");
                stackpush('A');
        }
        else if(stack[top]=='B')
        {
                if(input[i]!='b')
                {
                        printf("B->@");
                        printf("\t matched @");
                        pop();
                }
                else
                {
                        printf("B->bB");
                        stackpush('B');
                }
        }
        else
        {
                if(stack[top]==input[i])
                {
                        printf("pop%c",input[i]);
```

```
                        printf("\tmatched %c",input[i]);
                        pop();
                        i++;
                }
                else
                        break;
        }
    }
    if(stack[top]=='$' && input[i]=='$')
    {
        printf("\n$\t$");
        printf("\nValid string Accepted\n");
    }
    else
        printf("\nInvalid string rejected\n");
}
```

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 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();
```

```
                }
        }
}
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;
                }
}
```

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>
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);
}
```

6a. 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=0;
%}
%%
```

```
[/][*][^"*/"]*[*][/] {c++;}
[//].*[^\n] {c++;}
[a-zA-Z0-9] {fprintf(yyout,"%s",yytext);}
%%
int main(int argc, char *argv[])
{
yyin=fopen(argv[1], "r");
yyout=fopen(argv[2], "w");
yylex();
printf("The number of commented lines are %d\n",c);
return 0;
}
```

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

```
LEX part
%{
#include <stdio.h>
#include "y.tab.h"
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 part
%{
#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("f2.c", "r");
	if (!myfile)
	{
		printf("I can't open f2.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);
}
```

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.

Source Code:

```
#include<stdio.h>
int main()
{
int count,j,n,time,flag=0,time_quantum,ch=0;
int wait_time=0,turnaround_time =0,at[10],bt[10],rt[10];
int endTime ,i,smallest;
int remain=0,sum_wait=0,sum_turnaround=0;
printf("1.Round Robin \n2.SRTF \n");
scanf("%d",&ch);
printf("Enter no. of processes: ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter arrival time for Process P%d: ",i+1);
scanf("%d",&at[i]);
printf("Enter burst time for Process P%d: ",i+1);
scanf("%d",&bt[i]);
rt[i]=bt[i];
}
switch(ch)
{
case 1:
```

```
printf("Enter Time Quantum:\t");
scanf("%d",&time_quantum);
remain=n;
printf("\nProcess Time|Turnaround Time|Waiting Time\n");
for(time=0,count=0;remain!=0;)
{
if(rt[count]<=time_quantum && rt[count]>0)
{
time+=rt[count];
rt[count]=0;
flag=1;
}
else if(rt[count]>0)
{
rt[count]-=time_quantum;
time+=time_quantum;
}
if(rt[count]==0 && flag==1)
{
remain--;
printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-at[count]-bt[count]);
wait_time+=time-at[count]-bt[count];
turnaround_time+=time-at[count];
flag=0;
}
if(count==n-1)
count=0;
else if(at[count+1]<=time)
count++;
else
count=0;
}
printf("\nAverage Waiting Time=%.2f\n",wait_time*1.0/n);
printf("Average Turnaround Time=%.2f\n",turnaround_time*1.0/n);
break;
case 2:
remain=0;
printf("\nProcess Time|Turnaround Time|Waiting Time\n");
rt[9]=9999;
for(time=0;remain!=n;time++)
{
smallest=9;
for(i=0;i<n;i++)
if(at[i]<=time && rt[i]<rt[smallest] && rt[i]>0)
smallest=i;
rt[smallest]--;
if(rt[smallest]==0)
{
remain++;
endTime=time+1;
printf("\nP[%d]\t|\t%d\t|\t%d",smallest+1,endTime-at[smallest],endTime-bt[smallest]-at[smallest]);
```

```
printf("\n");
sum_wait+=endTime-bt[smallest]-at[smallest];
sum_turnaround+=endTime-at[smallest];
}}
printf("\nAverage Waiting Time=%f\n",sum_wait*1.0/n);
printf("AVerage Turnaround Time =%f\n",sum_turnaround*1.0/n);
break;
default:
printf("Invalid\n");
}
return 0;
}
```

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

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
	int Max[10][10], need[10][10], alloc[10][10], avail[10], completed[10], safeSequence[10];
	int p, r, i, j, process, count;
	count = 0;
	printf("Enter the no of processes : ");
	scanf("%d", &p);
	for(i = 0; i< p; i++)
		completed[i] = 0;
	printf("Enter the no of resources : ");
	scanf("%d", &r);
	printf("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("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("Enter 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("Max matrix:\t\nAllocation 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("Process %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("The system is in a safe state!!\n");
        printf("Safe Sequence : < ");
        for( i = 0; i < p; i++)
                printf("%d ", safeSequence[i]);
        printf(">\n");
}
```

```
        else
                printf("The system is in an unsafe state!!");
}
```

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

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

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

int main()
{
   int ch,YN=1,i,l,f;
   char F[10],s[25];
   printf("\nEnter the no of empty frames: ");
   scanf("%d",&f);
   printf("\nEnter the length of the string: ");
   scanf("%d",&l);
   printf("\nEnter the string: ");
   scanf("%s",s);
   for(i=0;i<f;i++)
      F[i]=-1;

   do
   {
      printf("\n********** MENU **********");
      printf("\n1:FIFO\n2:LRU \n3:EXIT");
      printf("\nEnter 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 3: exit(0);
      }
      printf("\n\nDo u want to continue IF YES PRESS 1\nIF 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\t\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\t\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;
  }
}
```