Practical File

of

Intelligent Systems Lab

(PCC-CS-601)

submitted in partial fulfillment of the requirement for the award of degree of

Bachelor of Technology (B.Tech)

in

Computer Engineering

By

Name: Dilpreet Grover

(Roll No.: 21001003037)

Under the guidance of

Dr. Mamta Kathuria

Department of Computer Engineering

J. C. BOSE UNIVERSITY OF SCIENCE & TECHNOLOGY, YMCA

SECTOR-6 FARIDABAD

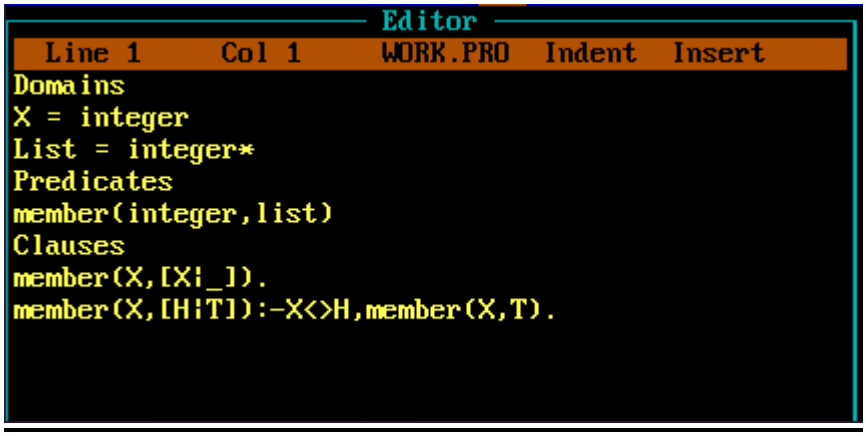
HARYANA-121006.

INDEX:-

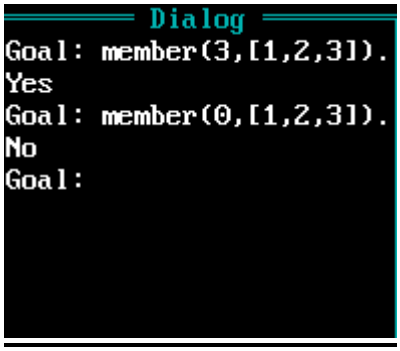
20.	**Program to implement Mergesort.**
21.	**Program to implement Bubblesort.**
22.	**Program to implement Selectionsort.**
23.	**Program to implement Insertionsort.**
24.	**Program to implement BFS.**
25.	**Program to implement DFS.**
26.	**Program to implement Family Relation Tree.**

Practical 1: Program to check whether an element is a member of a list or not.

Code:



Output:



Practical 2: Program to check whether a list is a subset of another list or not.

Code:

```
Domains
List=integer*
X=integer
Predicates
member(integer,List)
subset(List,List)
Clauses
member(X,[X|_]).
member(X,[H|T]):-X<>H, member(X,T).
subset([],List).
subset([H|T],List):-member(H, List), subset(T,List).
```

Output:

```
Dialog
Goal: subset([1,2],[1,3,
2]).
Yes
Goal:
```

Practical 3: Program to count number of elements in a list.

Code:

```
Domains
List=integer*
X=integer
Predicates
count(List,X)
Clauses
count([],0).
count([H|T],X):-count(T,M),X=M+1.
```

Output:

```
Goal: count([1,2,3,4],A)
.
A=4
1 Solution
```

Practical 4: Program to find sum of all elements of a list.

Code:

```
Domains
List=integer*
X=integer
Predicates
sum(List,X)
Clauses
sum([],0).
sum([H|T],X):-sum(T,M),X=M+H.
```

Output:

```
Goal: sum([1,2,3,4],A).
A=10
1 Solution
```

Practical 5: Program to implement Factorial of a number.

Code:

```
Domains
X=integer
Y=integer
Predicates
fac(X,Y)
Clauses
fac(0,1).
fac(X,Y):-H=X-1,fac(H,M),Y=X*M.
```

Output:

```
Goal: fac(4,A).
A=24
```

Practical 6: Program to append an element to a list.

Code:

```
Domains
X=integer*
Predicates
append(X,X,X)
Clauses
append([],X,X).
append([H|T],X,[H|T2]):-append(T,X,T2).
```

Output:

```
Goal: append([1,2],[3,4]
,A).
A=[1,2,3,4]
1 Solution
```

Practical 7 : Program to concatenate two lists.

Code:

```
Domains
X=integer*
Predicates
concaten(X,X,X)
Clauses
concaten([],X,X).
concaten([H|T],X,[H|T2]):-concaten(T,X,T2).
```

Output:

```
Goal: concaten([1,2],[3,
4],A).
A=[1,2,3,4]
1 Solution
```

Practical 8: Program to delete an element from a list.

Code:

```
Domains
X=integer*
Y=integer
Predicates
delete(integer,X,X)
Clauses
delete(Y,[Y|T],T).
delete(Y,[H|T],[H|T2]):-Y<>H,delete(Y,T,T2).
```

Output:

```
Goal: delete(3,[1,3,2],A
).
A=[1,2]
1 Solution
```

Practical 9: Program to reverse a list.

Code:

```
Domains
X=integer*
Y=integer
Predicates
append(X,X,X)
reverse(X,X)
Clauses
append([],X,X).
append([H|T],X,[H|T2]):-append(T,X,T2).
reverse([],[]).
reverse([H|T],L2):-reverse(T,L1),append(L1,[H],L2).
```

Output:

```
Goal: reverse([1,2,3,4],
A).
A=[4,3,2,1]
1 Solution
```

Practical 10: Program to delete all occurrences of an element from a list.

Code:

```
Domains
X=integer*
Y=integer
Predicates
delete(integer,X,X)
Clauses
delete(_,[],[]).
delete(Y,[Y|T],B):-delete(Y,T,B).
delete(Y,[H|T],[H|T2]):-Y<>H,delete(Y,T,T2).
```

Output:

```
Goal: delete(1,[1,3,1,1,
4,1,1,2],A).
A=[3,4,2]
1 Solution
```

Practical 11: Program to Replace an element by another element in a list.

Code:

```
Domains
X=integer*
Y=integer
Predicates
replace(integer,integer,X,X)
Clauses
replace(_,_,[],[]).
replace(N,M,[N|T],[M|T2]):-replace(N,M,T,T2).
replace(N,M,[H|T],[H|T2]):-N<>H,replace(N,M,T,T2).
```

Output:

```
Goal: replace(1,2,[1,3,4
,5],A).
A=[2,3,4,5]
1 Solution
```

Practical 12: Program to Replace all occurrences of an element by another element in a list.

Code:

```
Domains
X=integer*
Y=integer
Predicates
replace(integer,integer,X,X)
Clauses
replace(_,_,[],[]).
replace(N,M,[N|T],[M|T2]):-replace(N,M,T,T2).
replace(N,M,[H|T],[H|T2]):-N<>H,replace(N,M,T,T2).
```

Output:

```
Goal: replace(1,2,[1,3,1
,1,4,5],A).
A=[2,3,2,2,4,5]
1 Solution
```

Practical 13: Program to find union of two lists.

Code:

```
Domains
X=integer
List=integer*
Predicates
member(integer,List)
union(List,List,List)
Clauses
member(X,[X|_]).
member(X,[H|T]):-X<>H,member(X,T).
union([],List,List).
union(List,[],List).
union([H|T],List,[H|T2]):-not(member(H,List)),
union(T,List,T2).
union([H|T],List,T2):-member(H,List),
```

```
union(T,List,T2)._
```

Output:

```
Goal: union([1,2,3],[2,4
,5],A).
A=[1,3,2,4,5]
1 Solution
Goal:
```

Practical 14: Program to find intersection of two lists.

Code:

```
Domains
X=integer
List=integer*
Predicates
member(integer,List)
inters(List,List,List)
Clauses
member(X,[X|_]).
member(X,[H|T]):-X<>H,member(X,T).
inters([],List,[]).
inters(List,[],[]).
inters([H|T],List,T2):-not(member(H,List)),
inters(T,List,T2).
inters([H|T],List,[H|T2]):-member(H,List),
```

```
inters(T,List,T2).
```

Output:

```
Goal: inters([1,2,3],[2,
3,4,5],A).
A=[2,3]
1 Solution
```

Practical 15: Program to generate Fibonacci Series.

Code:

```
Domains
X=integer
X1=integer
X2=integer
A=integer
A1=integer
A2=integer
Predicates
FIB(integer,integer)
Clauses
FIB(0,1).
FIB(1,1).
FIB(A,X):-A1=A-1,A2=A-2,FIB(A1,X1),FIB(A2,X2),
X=X1+X2._
```

Output:

```
Goal: FIB(4,A).
A=5
Goal: FIB(3,A).
A=3
Goal: FIB(2,A).
A=2
Goal: FIB(1,A).
A=1
```

Practical 16: Program to find last element of a list.

Code:

```
Domains
X=integer
List=integer*
Predicates
last(List,integer)
Clauses
last([H],H).
last([H|T],R):-last(T,R).
```

Output:

```
Goal: last([1,3,4,2],A).
A=2
1 Solution
```

Practical 17: Program to check if two lists are equal or not if in same order.

Code:

```
Domains
X=integer
List=integer*
Predicates
equal(List,List)
Clauses
equal([],[]).
equal([H|T],[H|R]):-equal(T,R).
```

Output:

```
Goal: equal([1,2,3],[1,2
,3]).
Yes
```

Practical 18: Program to check if two lists are equal or not if in different order.

Code:

```
Domains
X=integer
L=integer*
Predicates
MEMBER(integer,L)
EQUAL(L,L)
DEL(integer,L,L)
Clauses
DEL(_,[],[]).
DEL(X,[X|T],T).
DEL(X,[H|T],[H|T2]):-DEL(X,T,T2),X<>H.
MEMBER(X,[X|_]).
MEMBER(X,[_|T]):-MEMBER(X,T).
EQUAL([],[]).
```

```
EQUAL([X],[X]).
EQUAL([H1|T],L):-MEMBER(H1,L),DEL(H1,L,L3),
EQUAL(T,L3)._
```

Output:

```
Goal: EQUAL([1,2,3,4],[3
,4,1,2]).
Yes
```

Practical 19: Program to implement Quicksort.

Code:

```
Domains
List=integer*
X=integer
Predicates
Qsort(List,List)
Partition(integer,List,List,List)
append(List,List,List)
Clauses
append([],List,List).
append([H|T],List,[H|T2]):-append(T,List,T2).
Qsort([],[]).
Qsort([H|T],S):-Partition(H,T,L,R),
Qsort(L,LS),Qsort(R,RS),append(LS,[H|RS],S).
Partition(X,[],[],[]).
```

```
Partition(X,[H|T],[H|L],R):-H<X,Partition(X,T,L,R).
Partition(X,[H|T],L,[H|R]):-H>=X,Partition(X,T,L,R).
```

Output:

```
Goal: Qsort([8,6,9,7,3],
A).
A=[3,6,7,8,9]
1 Solution
```

Practical 20: Program to implement Mergesort.

Code:

```
Domains
List=integer*
X=integer
Predicates
MS(List,List)
M(List,List,List)
D(LiST,List,List)
Clauses
D([],[],[]).
D([X],[X],[]).
D([A,B|T],[A|T1],[B|T2]):-D(T,T1,T2).
M([],List,List).
M(List,[],List).
M([H1|T1],[H2|T2],[H1|T3]):-H1<H2,M(T1,[H2|T2],T3).
```

```
M([H1|T1],[H2|T2],[H2|T3]):-H2<H1,M([H1|T1],T2,T3).
MS([],[]).
MS([X],[X]).
MS([A,B|T],S):-D([A,B|T],L1,L2),
MS(L1,S1),MS(L2,S2),M(S1,S2,S).
```

Output:

```
Goal: MS([8,6,9,7,2],A).
A=[2,6,7,8,9]
1 Solution
```

Practical 21: Program to implement Bubblesort.

Code:

```
Domains
List=integer*
Predicates
Bsort(List,List)
Swap(List,List)
Clauses
Bsort([],[]).
Bsort(List,SL):-Swap(List,List1),!,Bsort(List1,SL).
Bsort(List,List).
Swap([X,Y|T],[Y,X|T]):-X>=Y.
Swap([H|T],[H|T1]):-Swap(T,T1).
```

Output:

```
Goal: Bsort([8,6,9,7,2],
A).
A=[2,6,7,8,9]
1 Solution
```

Practical 22: Program to implement Selectionsort.

Code:

```
Domains
X=integer
L=integer*
A=integer
B=integer
M=integer
Predicates
MIN(integer,L,integer)
SSORT(L,L)
REMOVE(integer,L,L)
APPEND(L,L,L)
Clauses
SSORT([],[]).
SSORT([M1|S],[H|T]):-MIN(H,T,M1),REMOVE(M1,[H|T],N)
```

```
,SSORT(S,N).
MIN(M,[],M).
MIN(A,[H|T],M1):-A<H,MIN(A,T,M1).
MIN(A,[H|T],M1):-A>=H,MIN(H,T,M1).
APPEND([],B,B).
APPEND([H|A],B,[H|AB]):-APPEND(A,B,AB).
REMOVE(X,L,N):-APPEND(A,[X|B],L),APPEND(A,B,N).
```

Output:

```
Goal: SSORT(A,[8,6,9,7,3
]).
A=[3,6,7,8,9]
1 Solution
```

Practical 23: Program to implement Insertionsort.

Code:

```
Domains
X=integer
L=integer*
Predicates
I(integer,L,L)
IS(L,L)
Clauses
I(X,[],[X]).
I(X,[X1|L1],[X,X1|L1]):-X<X1.
I(X,[X1|L1],[X1|L]):-I(X,L1,L),X>=X1.
IS([],[]).
IS([X|L],S):-IS(L,S1),I(X,S1,S).
```

Output:

```
Goal: IS([8,6,7,9,2],A).
A=[2,6,7,8,9]
1 Solution
```

Practical 24: Program to implement BFS.

Code:

```
Domains
List=Symbol*
X=Symbol
Y=Symbol
Predicates
Children(Symbol,List)
BFS(List,Symbol,List)
append(List,List,List)
Clauses
append([],X,X).
append([H|T],X,[H|T2]):-append(T,X,T2).
Children(a,[b,c,d]).
Children(b,[e,f]).
Children(c,[g]).
```

```
Children(d,[h,i]).
Children(e,[]).
Children(f,[]).
Children(g,[]).
Children(h,[]).
Children(i,[]).
BFS([],_,[]).
BFS([H|_],H,[H]).
BFS([H|T],Y,P):-H<>Y,Children(H,L1),append(T,L1,N)
,BFS(N,Y,NP),append([H],NP,P),!.
```

Output:

```
Goal: BFS([a,b,c,d,e,f,g
,h,i],g,A).
A=["a","b","c","d","e","
f","g"]
1 Solution
```

Practical 25: Program to implement DFS.

Code:

```
Domains
X=Symbol
List=Symbol*
Predicates
Children(Symbol,List)
append(List,List,List)
DFS(List,Symbol,List)
Clauses
Children(a,[b,c,d]).
Children(b,[e,f]).
Children(c,[g]).
Children(d,[h,i]).
Children(e,[]).
Children(f,[]).
```

```
Children(g,[]).
Children(h,[]).
Children(i,[]).
append([],List,List).
append([H|T],List,[H|L2]):-append(T,List,L2).
DFS([H|_],H,[H]).
DFS([H|T],Y,P):-H<>Y,Children(H,L1),append(L1,T,NT)
,DFS(NT,Y,NP),append([H],NP,P).
```

Output:

```
Goal: DFS([a,b,c,d,e,f,g
,h,i],g,A).
A=["a","b","e","f","c","
g"]
1 Solution
```

Practical 26: Program to implement Family Relation Tree.

Code:

```
Domains
P=Symbol
Predicates
Husband(P,P)
Wife(P,P)
Father(P,P)
Mother(P,P)
Parent(P,P)
Son(P,P)
Daughter(P,P)
Female(P)
Male(P)
Grandfather(P,P)
Grandmother(P,P)
```

```
Clauses
Husband("VJ","BM").
Husband("VM","HM").
Father("VJ","DM").
Father("VM","VJ").
Wife(X,Y):-Husband(Y,X).
Mother(X,Y):-Wife(X,Z),Father(Z,Y).
Parent(X,Y):-Mother(X,Y);Father(X,Y).
Son(X,Y):-Male(X),Father(X,Y).
Daughter(X,Y):-Female(X),Father(X,Y).
Grandfather(X,Y):-Father(X,Z),Father(Z,Y).
Grandmother(X,Y):-Mother(X,Z),Mother(Z,Y).
Male(X):-Son(X,Y);Father(X,Y).
Female(X):-Daughter(X,Y);Mother(X,Y).
```

Output:

```
Dialog
A=VJ
1 Solution
Goal: Mother("HM",A).
A=VJ
1 Solution
Goal: Grandfather("VM",A
).
A=DM
1 Solution
Goal: Parent("DM",A).
No Solution
Goal: Parent("VJ",A).
A=DM
1 Solution
```