INSTRUCTION MANUAL

SYMBOLIC LOGIC & LOGIC PROCESSING (SL&LP)

(CS-416)

Prepared by

Er. Sukhjit Singh Sehra
Assistant Professor (CSE)

GURU NANAK DEV ENGINEERING COLLEGE
LUDHIANA – 141006

INDEX

DECLARATION

This Manual of Symbolic Logic & Logic Processing (SL&LP) (CS-416) has been prepared by me as per syllabus of Symbolic Logic & Logic Processing (SL&LP) (CS-416).

Signature

SYLLABUS
CS-416 (Software Lab XIII) SL&LP

External Marks: 20 **L T P**

Internal Marks: 30 **- - 2**

Total Marks: 50

- Study of Propositional Logic
- Study of First Order Predicate Logic
- Introduction to prolog programming by a simple prolog program
- Program to check whether input is alphabet or not
- Program to find if given number is positive or negative.
- Write a program to check whether a given person is a member of Club
- Program in prolog showing mapping that is constructing new structure similar to old one.
- Program illustrating the use of recursion that is finding sum of first Nintegers.
- Program to find the length of a list using ‘Recursion’ and then using“recursion and Accumulators’
- Program to find the factorial of a number using recursion and accumulators and cut.
- Program to calculate average tax illustrating cut-fail combination usage.
- Program showing use of cut in Terminating a ‘generate and test’.Program to play “Tic Tac Toe”
- Write a program to generate fibonacci series upto the given no.
- Write a program which accepts any number and checks whether it is prime or not.
- To describe some basic predicates that are useful for manipulating lists.
- Program for Bubble Sort
- Program for Insertion Sort

INDEX

PRACTICAL 1

TITLE: - STUDY OF PROPOSITIONAL LOGIC.

PROPOSTIONAL LOGIC:-

Valid statements or sentences in Propositional Logic are determined according to the rules of propositional syntax. This syntax governs the combination of basic building blocks such as propositions and logical connectives. Propositions are elementary atomic sentences. Propositions may be either true or false but may take no other value. Propositions are of two types:

1. Simple Propositions
2. Compound Propositions

Example of simple Propositions:

a) It is raining.
b) I have time
c) We will go for movie

Compound Propositions are formed from atomic formulas using the logical connectives not and or if…. then, and if and only if.

Example of Compound propositions:

a) It is raining and the wind is blowing.
b) If you study hard you will be rewarded.

The following symbols are used for logical connectives.

~	not / negation
&	and / conjunction
$\longleftrightarrow$	if and only if / double implication
V	Or / disjunction
$\longrightarrow$	if---then / implication

Syntax:-

The syntax for Propositional Logic is defined recursively as follows:

If P and Q are formulas, the following are formulas, then

(~P)

(~Q)

(P&Q)

(PVQ)

$(P \rightarrow Q)$

$(P \leftrightarrow Q)$

All formulas are generated from a finite number of the above operation.

Semantics:-

The semantics or meaning of a sentence is just the value true or false; that is, it is an assignment of a truth-value to the sentence. The values true and false should not be confused with the symbols T and F, which can appear within a sentence.

An interpretation for a sentence or group of sentences is an assignment of a truth value to each propositional symbol. Once an interpretation has been given to a statement, its truth value can be determined. This is done by repeated application of semantic rules to larger and larger parts of the statement until a single truth-value is determined.

Semantic rules for statements:

Rule number	**True statements**	**False statements**
1.	T	F
2.	~f	~t
3.	t & t’	f & f’
4.	t & a	a & f
5.	t V t’	f V f’
6.	$a \rightarrow t$	$f \rightarrow f$
7.	$f \rightarrow a$	$t \leftrightarrow f$
8.	$t \leftrightarrow t’$	$f \leftrightarrow t$
9.	$f \leftrightarrow f’$	

PRACTICAL 2

TITLE: - STUDY OF FIRST ORDER PREDICATE LOGIC

FIRST ORDER PREDICATE LOGIC:-

FOPL was developed by logicians to extend the expressiveness of Pl. It is a generalization of PL that permits reasoning about world objects as relational entities as well as classes or subclasses of objects. This generalization comes from the introduction of predicates in place of propositions, the use of functions and the use of variables together with variable quantifiers.

Syntax of FOPL:-

The symbols and rules of combination permitted in FOPL are defined as follows:

1. **Connectives**: There are five connective symbols:

~ (not / negation),

& (and / conjunction),

V (or / inclusive disjunction),

→ (implication),

↔ (equivalence / if and only if).

2. **Quantifiers**: The two quantifier symbols are V (universal quantifier) I (existential quantifier)
3. **Constants**: Constants are fixed value terms that belong to a given domain of discourse. They are denoted by numbers, words, and small letters near the beginning of the alphabet such as a, b, c, 5.3, -21, and john.
4. **Variables**: Variables are terms that can assume different values over a given domain. They are denoted by words and small letters near the end of the alphabet, such as aircraft-type, individuals, x, y, and, z.
5. **Functions**: Function symbols denote relations defined on a Domain D. They map *n* elements (n>=0) to a single element of the domain. Symbols f, g, h, and words such as father-of, or age-of, represent functions. An n place function is written as f (t1, t2… tn) where the ti are terms (constants, variables, or functions) defined over some domain.

6. **Predicates**: Predicate symbols denote relations or functional mappings from the elements of a domain D to the values true or false. Capital letter and capitalized words such as P, Q, R, EQUAL, and MARRIED are used to represent predicates.

Constants, variables and functions are referred to as *terms* and predicates are referred to as *atomic formulas* or *atoms* for short and, when we want to refer to an atom or its negation, we often use the word *literal*.

Semantics of FOPL:-

When considering specific wffs (well formed formulas), we always have in mind some domain D. If not stated explicitly, D will be understood from the context. D is the set of all elements or objects from which fixed assignments are made to constants and from which the domain and range of functions are defined. The arguments of predicates must be terms (constants, variables, or functions). Therefore, the domain of each n-place predicate is also defined over D.

When an assignment of values is given to each term and to each predicate symbol in a wff, we say an interpretation is given to the wff. Since literals always evaluate to either true or false under an interpretation, the value of any given wff can be determined by referring to a truth table, which gives truth-values for the subexpressions of the wff.

If the truth-values for two different wffs are the same under every interpretation, they are said to be *equivalent.* A predicate that has no variables is called a *ground atom.*

Properties of statements: -

- Satisfiable: **A statement is satisfiable if there is some interpretation for which it is true (PVQ).**

P	**Q**	**PVQ**
F	**F**	**F**
F	**T**	**T**
T	**F**	**T**
T	**T**	**T**

- Contradiction: **A sentence is contradictory if there is no interpretation for which it is true (PV~P).**

P	**~P**	**PV~P**
T	**F**	**T**
F	**T**	**T**
T	**F**	**T**

- Valid: **A sentence is valid if it is true for every interpretation (P&~P).**

P	**~P**	**P&~P**
T	**F**	**F**
F	**T**	**F**

- Equivalence: **Two sentences are equivalent if they have the same truth-value under every interpretation.**

- Logical consequence: **A sentence is a logical consequence of another if it is satisfied by all interpretations, which satisfy the first (P&Q→T if P→T).**

P	**Q**	**P&Q**
F	**T**	**F**
T	**F**	**F**
F	**F**	**F**
T	**T**	**T**

PRACTICAL 3

TITLE: - INTRODUCTION TO PROLOG PROGRAMMING.

INTRODUCTION TO PROLOG

Prolog is a computer programming language that is used for solving problems that involve objects and the relationships between objects.

Computer programming in Prolog consists of:

- Declaring some facts about objects and their relationships,
- Defining some rules about objects and their relationships, and
- Asking questions about objects and their relationships.

Features of Prolog Programming:-

1. Facts: Suppose we want to tell prolog the fact that "john likes Mary". This fact consists of two objects, called "Mary" and "john", and a relationship, called "likes". In prolog, we need to write facts in a standard form, like this:

   likes (john, Mary).

2. Questions: In prolog, a question looks just like a fact, except that we put a special symbol before it. The special symbol is written as a question mark and a hyphen.

   ? - owns (Mary, book).

3. Variables: In prolog we cannot only name particular objects, but we can also use names like X to stand for objects to be determined by prolog. Names of this second kind are called variables. The variable can be either instantiated or not instantiated. A variable is instantiated when there is an object that the variable stands for. A variable is not instantiated when what the variable stands for is not yet known.

   write (X).

Where, X is a variable.

4. Conjunctions: Suppose we have the following database:

```
likes(mary, food).

likes(mary, wine).

likes(john, wine).

likes(john, mary).
```

We ask "Does john like mary?" *and* "Does mary like john?" The *and* expresses the idea that we are interested in the conjunction of the two goals- we want to satisfy them both one after the other. We represent this by putting a comma between the goals:

? – likes(john, mary), likes(mary, john).

The comma is pronounced "*and*", and it serves to separate any number of different goals that have to be satisfied in order to answer a question.

5. Rules: In prolog, rules are used when you want to say that a fact depends on a group of other facts. A rule is a general statement about objects and their relationships.

In prolog, a rule consists of a *head* and a *body*. The head and body are connected by the symbol ':-', which is made up of a colon and a hyphen. The ':-' is pronounced if. For example:

likes(john, X) : - likes(X, wine).

A simple Prolog Program:-

```
male(albert).
male(edward)
female(alice)
female(victoria)
parents(edward,victoria,albert)
parents(alice,victoria,albert)
sister_of(X,Y) :- female(X), parents(X,M,F), parents(Y,M,F).
?- sister_of(alice,edward).
```

Output:

```
Yes
```

Here , facts are defined like male(edward) , male(albert), female(alice), female(victoria), parents(edward,victoria,albert),parents(alice,victoria,albert). And then we state a rule sister_of.The rule defines the predicate sister_of, having two arguments such as that sister_of(X,Y) is a fact if X is a sister of Y. When Prolog uses the rule, variables M and F will initially be uninstantiated,so that they will mmatch against anything when it becomes time to satisfy the goal parents(X,M,F).

The question ?- sister_of(alice,edward) searches the database and is found that alice is a sister of Edward so prolog answers 'yes' as output.

PRACTICAL 4

TITLE:-WRITE A PROGRAM TO CHECK WHETHER INPUT IS ALPHABET OR NOT .

Input Window:

```
% isalpha: succeeds if given symbol is alphabet

Isalpha:-  write \n (“enter any symbol”), get O(N),pass(N).
pass(N):-  64<N , N>91 , write \n (“given symbol is alphabet”).
pass(N):-  92<N , N>123 , write \n (“given symbol is an alphabet”).
pass(N):-  write \n (“given symbol is not an alphabet”).
```

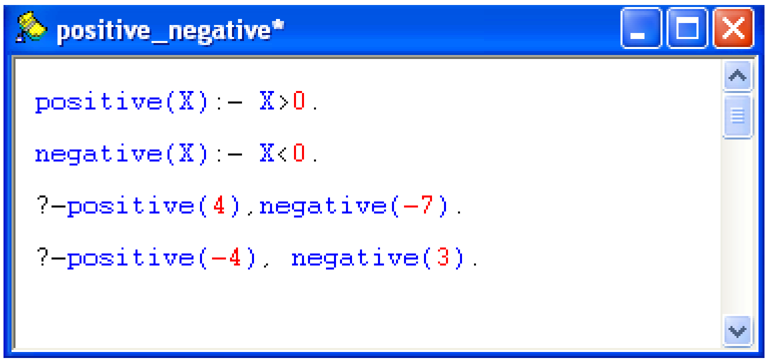
Output Window:

```
?- isalpha
Enter any symbol
a
Given symbol is an alphabet
```

PRACTICAL 5

TITLE :- WRITE A PROGRAM TO CHECK WHETHER A GIVEN NUMBER IS POSITIVE OR NEGATIVE.

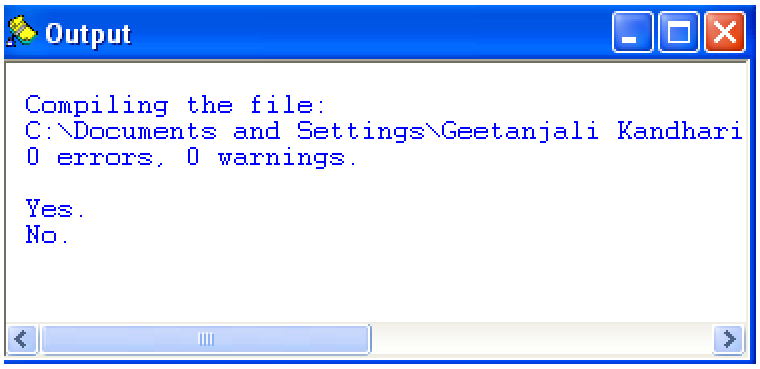
Input Window:



positive_negative*

```
positive(X):- X>0.

negative(X):- X<0.

?-positive(4),negative(-7).

?-positive(-4), negative(3).
```

Output Window:



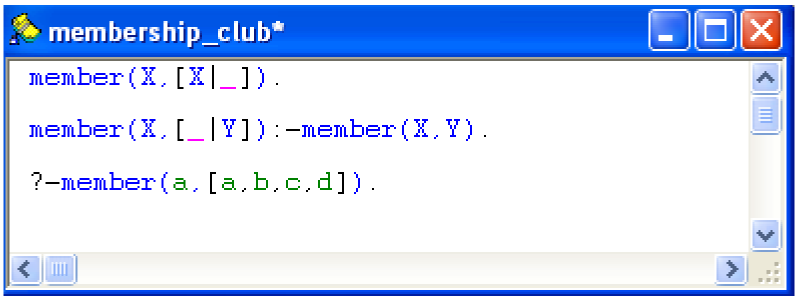
Output

```
Compiling the file:
C:\Documents and Settings\Geetanjali Kandhari
0 errors, 0 warnings.

Yes.
No.
```

PRACTICAL 6

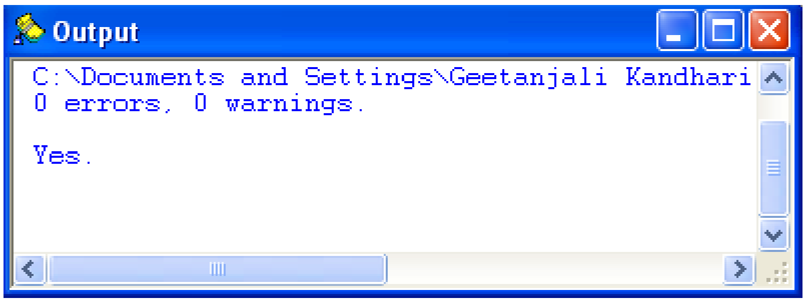
TITLE:- WRITE A PROGRAM TO FIND OUT THE MEMBERSHIP OF A GIVEN PERSON OF A CLUB.

Input Window:



```
member(X,[X|_]).

member(X,[_|Y]):-member(X,Y).

?-member(a,[a,b,c,d]).
```

Output Window:

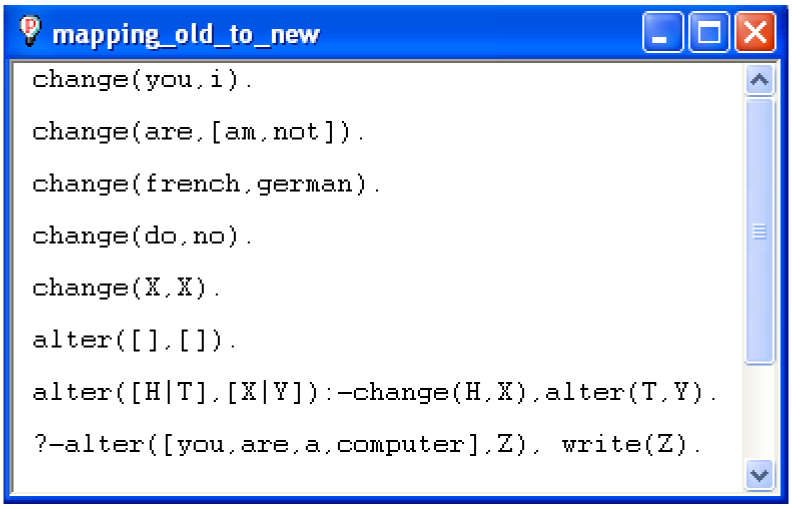


```
C:\Documents and Settings\Geetanjali Kandhari
0 errors, 0 warnings.

Yes.
```

PRACTICAL 7

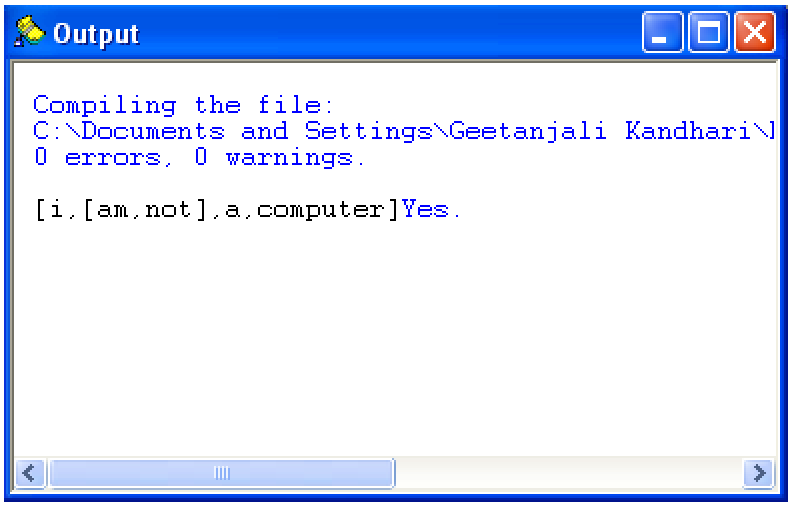
TITLE :- WRITE A PROGRAM SHOWING THE MAPPING FUNCTION CONSTRUCTING A NEW STRUCTURE SIMILAR TO OLD ONE

Input Window:



```
change(you,i).

change(are,[am,not]).

change(french,german).

change(do,no).

change(X,X).

alter([],[]).

alter([H|T],[X|Y]):-change(H,X),alter(T,Y).

?-alter([you,are,a,computer],Z), write(Z).
```

Output Window:



```
Compiling the file:
C:\Documents and Settings\Geetanjali Kandhari\]
0 errors, 0 warnings.

[i,[am,not],a,computer]Yes.
```

PRACTICAL 8

TITLE:- WRITE A PROGRAM TO FIND SUM OF FIRST N POSITIVE INTEGERS

Input Window:

```
Sum(0,0)
Sum(N,R): N1 is N-1,sum(N1,R1),R is R1+N
```

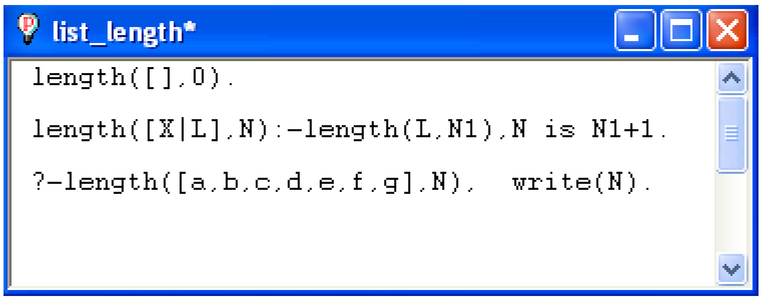
Output Window :

```
?- sum(3,R)
R=6
```

PRACTICAL 9

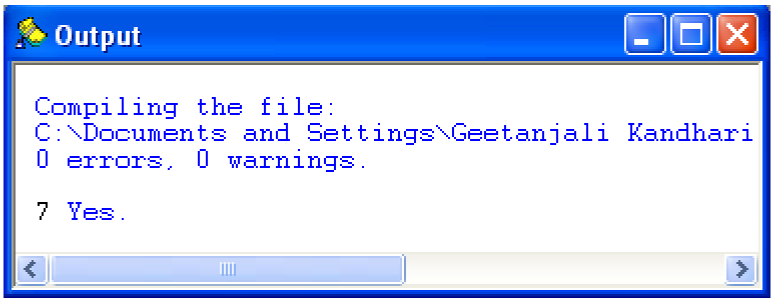
TITLE:-WRITE A PROGRAM TO FIND THE LENGTH OF A LIST.

Input Window:



```
length([],0).

length([X|L],N):-length(L,N1),N is N1+1.

?-length([a,b,c,d,e,f,g],N),  write(N).
```

Output Window:

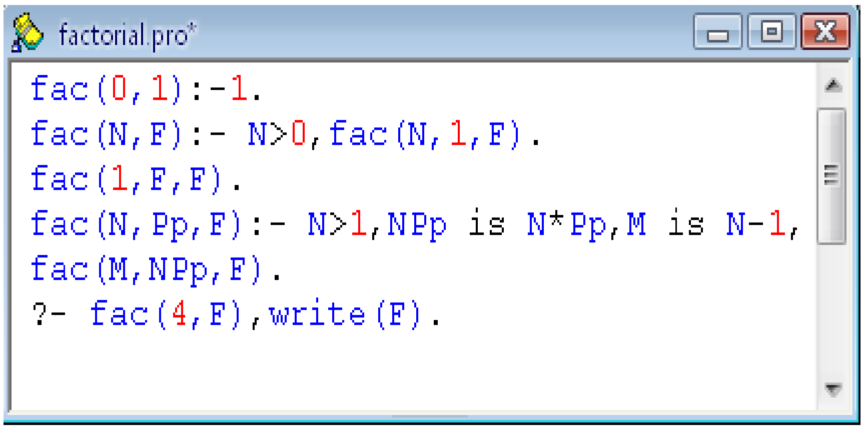


```
Compiling the file:
C:\Documents and Settings\Geetanjali Kandhari
0 errors, 0 warnings.

7 Yes.
```

PRACTICAL 10

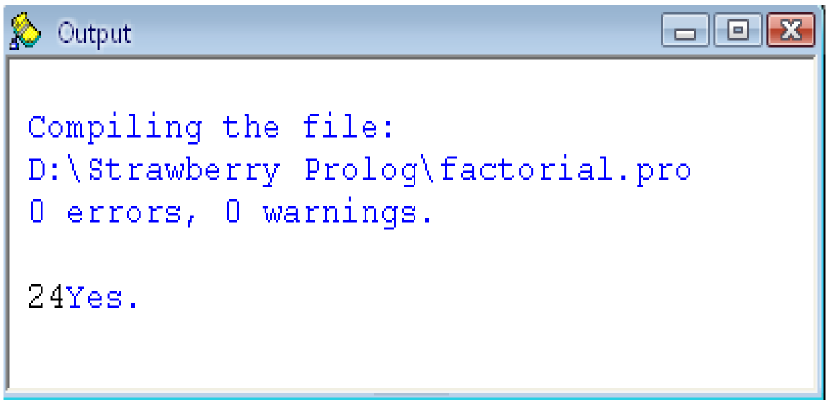
TITLE :- WRITE A PROGRAM TO FIND FACTORIAL OF A NUMBER USING CUT, ACCUMULATOR AND RECURSION

Input Window:



```
fac(0,1):-1.
fac(N,F):- N>0,fac(N,1,F).
fac(1,F,F).
fac(N,Pp,F):- N>1,NPp is N*Pp,M is N-1,
fac(M,NPp,F).
?- fac(4,F),write(F).
```

Output Window:

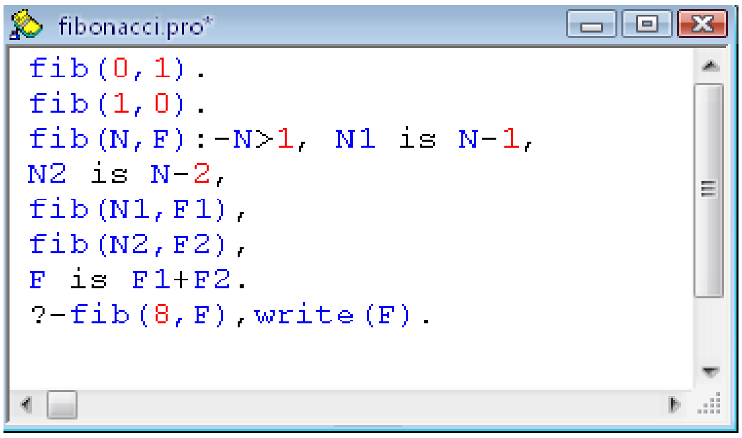


```
Compiling the file:
D:\Strawberry Prolog\factorial.pro
0 errors, 0 warnings.

24Yes.
```

PRACTICAL 11

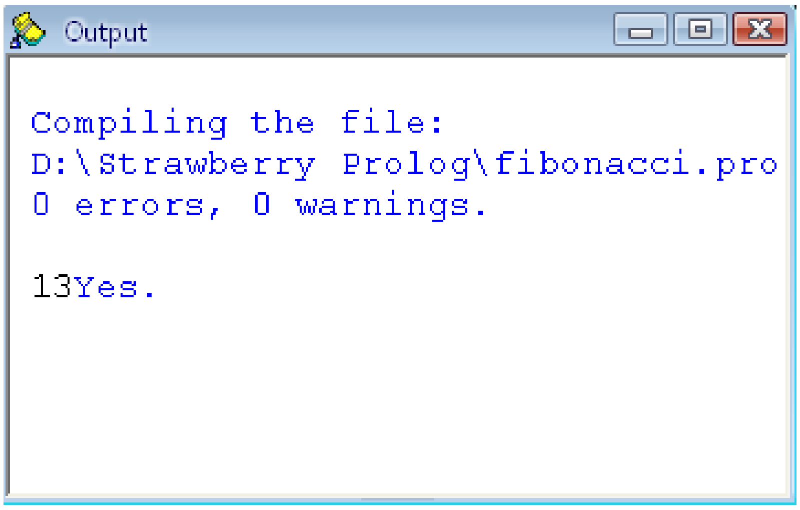
TITLE:- TO GENERATE FIBONACCI SERIES UP TO GIVEN NUMBER

Input Window:



```
fib(0,1).
fib(1,0).
fib(N,F):-N>1, N1 is N-1,
N2 is N-2,
fib(N1,F1),
fib(N2,F2),
F is F1+F2.
?-fib(8,F),write(F).
```

Output Window:



```
Compiling the file:
D:\Strawberry Prolog\fibonacci.pro
0 errors, 0 warnings.

13Yes.
```

PRACTICAL 12

TITLE:- WRITE A PROGRAM TO CHECK NUMBER IS PRIME OR NOT.

Input Window:

```
%is_prime(x) succeed if x is prime.
is_prime(2)
is_prime(3)
is_prime(P):integer (P),P>3,P mod 2 =\=0,\+ has factor (P,3)
has _factor(N,L): N mod =:=0
has_factor(N,L):L*L<N,L2 is L+2,
has_factor(N,L2)
```

Output Window:

```
? :- is_prime(11)
Yes
```

PRACTICAL 13

TITLE: - TO DESCRIBE THE BASIC BUILT IN PREDICATES FOR MANIPULATING A LIST.

BULIT IN PREDICATES:

By a built-in predicate we mean that the predicate's definition is provided in advance by the Prolog System, instead of your own clauses. Built-in predicates may provide facilities that cannot be obtained by definitions in pure Prolog. These predicates manipulate lists. They are bootstrapped predicates (i.e. written in Prolog) and no error cases are tested (for the moment). However, since they are written in Prolog using other built-in predicates, some errors can occur due to those built-in predicates.

Various Commands:-

Using a built-in predicates may cause changes apart from the instantiation of the arguments . Another important fact about built-in predicates is that they may expect particular sorts of arguments. There are two basic built-in predicates for reading in new clauses: *consult & reconsult.*

Consult(X): The built-in predicate consult is meant for those situations when you want the clauses in some file to augment those already in the database. The argument must be an atom giving the name of the file the clauses are to be taken from. Which atoms constitute a legal file name will depend on your computer.

Example:
```
?-consult(myfile).
?-connsult('/usr/john/pl/chat').
```

When it has to satisfy a consult goal, Prolog reads through the file, adding the clauses it finds at the end of the database. As a result new clauses will appear after any already existing clauses for the same predicates.

Reconsult(X): The predicate reconsult is just like consult, except that the clauses read in are taken to supersede all existing clauses for the same predicate. Because of this, reconsult is appropriate for correcting programming mistakes. If you read in several files of clauses and then discover that there is a mistake in one clause, you may be able to correct it without having to read in all the fileas again. You just have to reconsult a file containing a correct set of clauses for the predicate in question.

Example:
```
?-(reconsult(user)).
```

PRACTICAL 14

TITLE:- WRITE A PROGRAM FOR BUBBLE SORT

Input Window:

```
% bubble sort(list,sorted)- succeed if 'sorted' is sorted.
%list of given list 'list'
bubble sort(list,sorted):swap(list,list1),1,bubble sort(list1,sorted)
bubble sort(sorted ,sorted)
swap([X,Y/Rest],[Y/X/Rest]):X>Y
swap([Z/Reset],[Z/Reset1]):swap(Rest,Rest1)
```

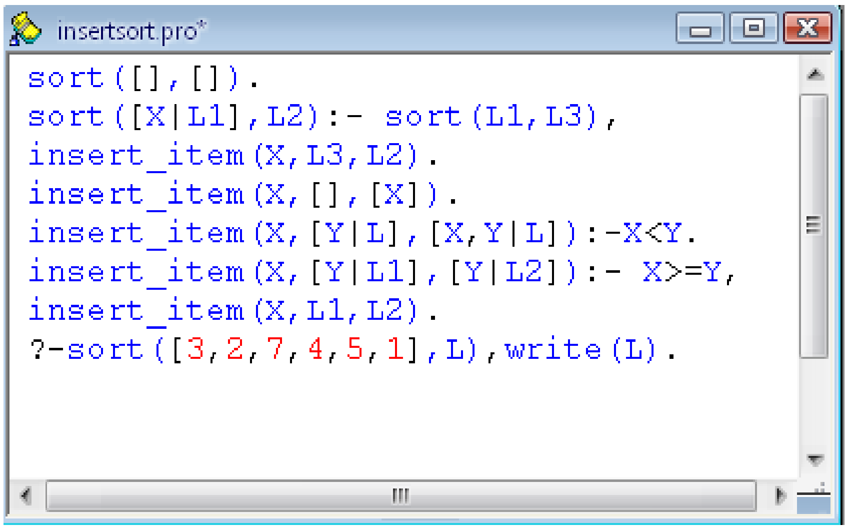
Output Window:

```
? bubble sort([10,4,5],sorted)
Sorted=[4,5,10]

.
```

PRACTICAL 15

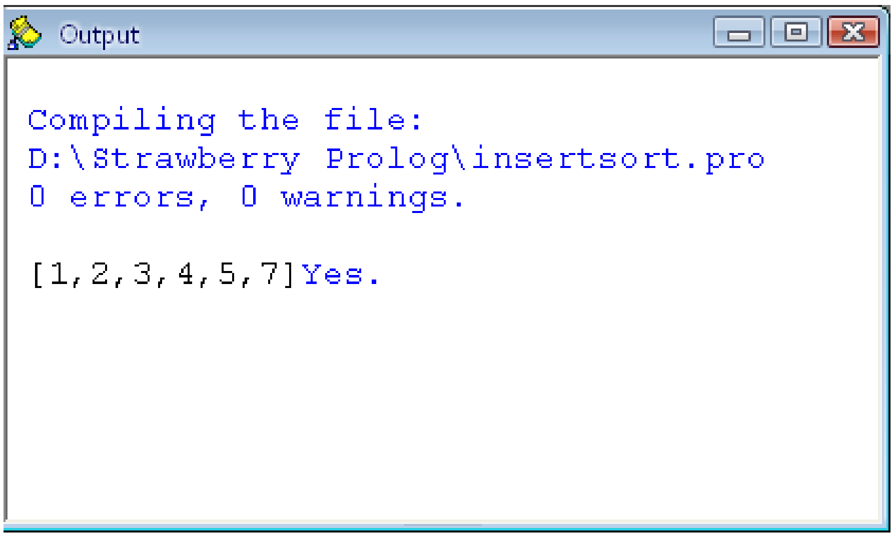
TITLE:- PROGRAM FOR INSERTION SORT

Input Window:



```
sort([],[]).
sort([X|L1],L2):- sort(L1,L3),
insert_item(X,L3,L2).
insert_item(X,[],[X]).
insert_item(X,[Y|L],[X,Y|L]):-X<Y.
insert_item(X,[Y|L1],[Y|L2]):- X>=Y,
insert_item(X,L1,L2).
?-sort([3,2,7,4,5,1],L),write(L).
```

Output Window:



```
Compiling the file:
D:\Strawberry Prolog\insertsort.pro
0 errors, 0 warnings.

[1,2,3,4,5,7]Yes.
```

PRACTICAL 16

TITLE:- WRITE A PROGRAM TO FIND LENGTH OF LIST USING RECURSION

Input Window:

```
%len_list(L,N)- succeeds if N is unified with the  length of  a list L.
len_list ( [],0 )
len_list ( [ H/T],N ) :len_list(T,N1),N is N1+1
```

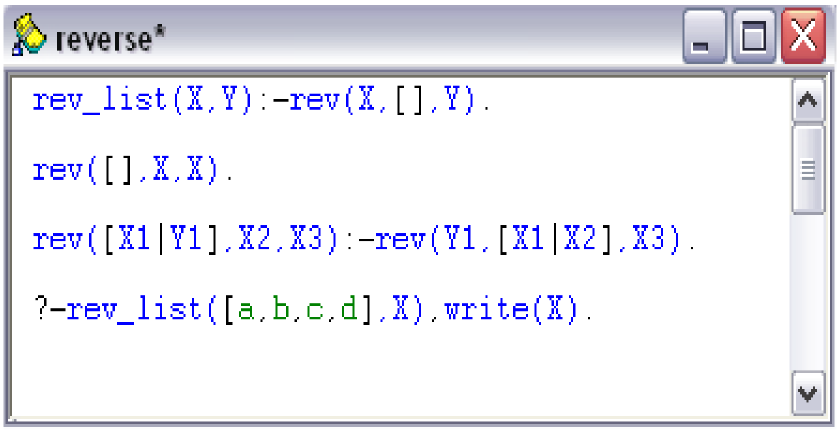
Output Window:

```
? len_list( [a,b,c],N)
N=3
```

PRACTICAL 17

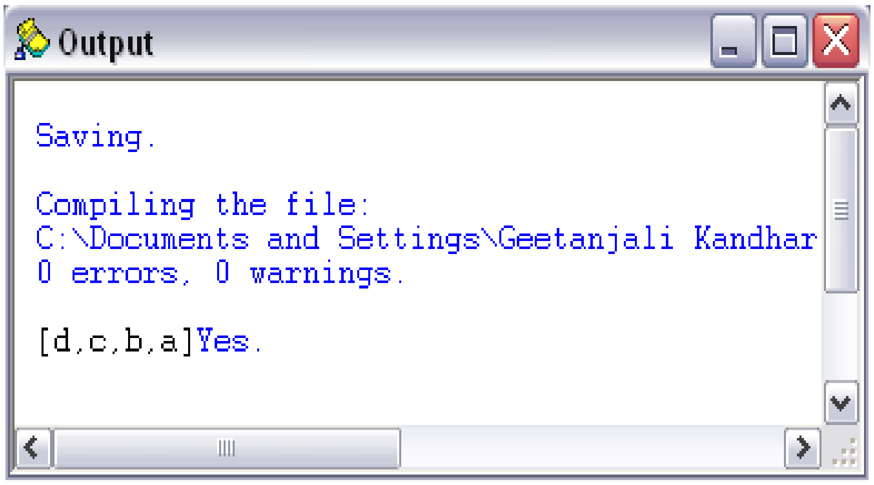
TITLE:- WRITE A PROGRAM TO FIND REVERSE OF LIST.

Input Window:



```
rev_list(X,Y):-rev(X,[],Y).

rev([],X,X).

rev([X1|Y1],X2,X3):-rev(Y1,[X1|X2],X3).

?-rev_list([a,b,c,d],X),write(X).
```

Output Window:



```
Saving.

Compiling the file:
C:\Documents and Settings\Geetanjali Kandhar
0 errors, 0 warnings.

[d,c,b,a]Yes.
```

PRACTICAL 18

TITLE:- WRITE A PROGRAM ILLUSTRING THE USE OF RECURSION THAT FINDING OF FIRST N INTEGERS

Input Window :

```
% Sum(N,S):succeeds if R is sum of first.
%N integers.
Sum(1,1)
Sum(N,S): N1 is N-1,Sum(N1,S1),
S is S1+N.
```

Output Window:

```
? : - Sum(3,S)
S=6
```

PRACTICAL 19

TITLE:- WRITE A PROGRAM FOR COMPUTING SQUARE OF A NUMBER

Input Window:

```
% square(X,Y)-suceeds if Y is a square of X
Square(X,Y): Y is X*X

Query:?-square(2,4)
Answer: Y=4
```

Output window :

```
?- square(3,Y)
Y=9
```