Program Code:

```
grade(english, 3).
grade(math, 3).
grade(history, 2).

gpa() :-
    findall(Grade, grade(_, Grade), Grades),
    sum_list(Grades, TotalGrade),
    length(Grades, NumberOfCourses),
    GPA is TotalGrade / NumberOfCourses,
    format("GPA is: ~2f~n", [GPA]).
```

Output:

```
?- gpa().
GPA is: 2.67
true.

?-
```

Program Code:

```
fibonacci(0, 0).
fibonacci(1, 1).

fibonacci(N, Result) :-
    N > 1,
    N1 is N-1,
    N2 is N-2,
    fibonacci(N1, Result1),
    fibonacci(N2, Result2),
    Result is Result1 + Result2.
```

Output:

```
?- fibonacci(9, Ans).
Ans = 34 ;
false.

?-
```

Program Code:

```
factorial(0, 1).

factorial(N, Result) :-
    N > 0,
    N1 is N-1,
    factorial(N1, Result1),
    Result is N * Result1.
```

Output:

```
?- factorial(7, Ans).
Ans = 5040 ;
false.

?-
```

Program Code:

```
gcd(A,0,A).

gcd(A,B,Result) :-
   B > 0,
   A1 is B,
   B1 is A mod B,
   gcd(A1, B1, Result).

lcm(A,B,Result) :-
   gcd(A,B,GCD),
   Result is (A*B)/GCD.
```

Output:

```
?- gcd(21, 35, GCD).
GCD = 7 ;

?- lcm(21, 35, LCM).
LCM = 105 .
```

Program Code:

```
edge(a,b).
edge(a,c).
edge(b,d).
edge(d,e).
edge(e,b).
path(A,B):-nextRoute(A,B,[]), write(B).
nextRoute(A,B,V) :-
    edge(A,X), write(A), write('->'),
    not(member(X,V)) ,
    ( B = X;
    nextRoute(X,B,[A|V]);
    write(X)).
```

Output:

```
?- path(a,d).
a->b->d
true .

?-
```

Program Code:

```
writeFile(File) :-
   tell(File),
   write("This is a demo."),
   told,
   writeln("Written in File.").

readFile(File) :-
   open(File,read,Str),
   readWords(Str,Words),
   close(Str),
   write(Words),  nl.
readWords(Stream,[]):-
   at_end_of_stream(Stream).
readWords(Stream,[X|L]):-
   \+ at_end_of_stream(Stream),
   readWord(Stream,X),
   readWords(Stream,L).
readWord(InStream,W):-
         get_code(InStream,Char),
         checkCharAndReadRest(Char,Chars,InStream),
         atom_codes(W,Chars).

   checkCharAndReadRest(10,[],_):-  !.
   checkCharAndReadRest(32,[],_):-  !.
   checkCharAndReadRest(-1,[],_):-  !.
   checkCharAndReadRest(end_of_file,[],_):-  !.
   checkCharAndReadRest(Char,[Char|Chars],InStream):-
         get_code(InStream,NextChar),
         checkCharAndReadRest(NextChar,Chars,InStream).
```

Output:

```
?- writeFile('in.txt').
Written in File.
true.

?- readFile('in.txt').
[This,is,a,demo.]
true
```

Program Code:

```
edges(a, b).
edges(a, c).
edges(b, c).
edges(b, d).
edges(c, d).
edges(d, e).
edges(e, f).

bfs(Start, Goal, Path) :-
    bfs_queue([[Start]], Goal, RevPath),
    reverse(RevPath, Path).

bfs_queue([[Node|Path]|_], Node, [Node|Path]).
bfs_queue([[Node|Path]|Rest], Goal, FinalPath) :-
    findall([Next, Node|Path],
            (edges(Node, Next), \+ member(Next, [Node|Path])),
            NextPaths),
    append(Rest, NextPaths, UpdatedQueue),
    bfs_queue(UpdatedQueue, Goal, FinalPath).
bfs_queue([_|Rest], Goal, Path) :-
    bfs_queue(Rest, Goal, Path).
```

Output:

```
?- bfs(a,f,Path).
Path = [a, b, d, e, f] .

?-
```

Program Code:

```
edges(a, b).
edges(a, c).
edges(b, c).
edges(b, d).
edges(c, d).
edges(d, e).
edges(e, f).

dfs(Start, Goal, Path) :-
    dfs_helper(Start, Goal, [Start], Path).

dfs_helper(Goal, Goal, AccPath, Path) :- reverse(AccPath, Path).
dfs_helper(Node, Goal, AccPath, Path) :-
    edges(Node, Next),
    \+ member(Next, AccPath),
    dfs_helper(Next, Goal, [Next|AccPath], Path).
```

Output:

```
?- dfs(a,f,Path).
Path = [a, b, c, d, e, f] .

?-
```

Program Code:

```
edges(a, b, 1).
edges(a, c, 2).
edges(b, c, 3).
edges(b, d, 4).
edges(c, d, 5).
edges(d, e, 6).
edges(e, f, 7).

heuristic(a, f, 10).
a_star(Start, Goal, Path) :-
   astar_search([[(0, Start)]], Goal, Path).
astar_search([(_, Node) | _], Node, Path) :- !.
astar_search([Path | Rest], Goal, FinalPath) :-
   Path = [(_, Node) | _],
   findall((HeuristicCost + NewCost, Next),
         (edges(Node, Next, StepCost),
         NewCost is Cost + StepCost,
         heuristic(Next, Goal, HeuristicCost),
         \+ member(Next, Path)),
         NextPaths),
   append(NextPaths, Rest, UpdatedQueue),
   keysort(UpdatedQueue, SortedQueue),
   astar_search(SortedQueue, Goal, FinalPath).
main :-
   Start = a,
   Goal = f,
   a_star(Start, Goal, Path),
   reverse(Path, PathInOrder),
   format("A* Path from ~w to ~w: ~w~n", [Start, Goal, PathInOrder]).
:- initialization(main).
```

Output:

```
?- main.
Path = [a, b, d, e, f] .

?-
```