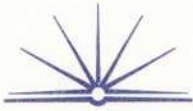


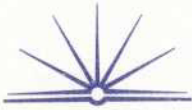
- a) Fragment 1 ~~uses~~ has been coded using the logic paradigm. This is evident because of the presence of a set of ~~rules~~ <sup>facts</sup> e.g. hair(sally, red) and then a query e.g. ?-hair(sally, X). Fragment 2 is of the functional paradigm. The features of the paradigm present are functions and recursion.

- b) The object orientated paradigm has emerged as a result of the development of Graphical User Interfaces (GUIs) and the need for greater productivity and ease of use of software. ~~The imperative paradigm~~ With the emergence of GUIs, tasks no longer needed to be executed in a set order. Users were able to select from many different tasks on screen so it became more efficient for an object orientated paradigm to emerge. Code was now activated when the user invoked it through objects on the screen, and thus it also ~~came~~ emerged through the increase in demand for highly interactive software and games, multimedia that could not follow ~~a~~ a set ~~path~~ sequence such as in the imperative paradigm.



The object oriented paradigm emerged also from the need for greater productivity as ~~well as~~ tested and secure objects could simply ~~be put together~~ put together to synthesise a much larger program. This meant less development time and easier maintenance of code.

c) i) ~~Then another~~ The input for the rectangle height is located within the loop that ~~uses~~ uses it as a condition. This prevents the program from running. To fix this you could ~~either~~ move the input to before the loop. Alternatively you could change the repetition to a post-test (or unguarded loop) either method would solve the problem.



ii) type

~~Rectangle~~ →

PTriangle = ^TTriangle

TTriangle = object(TObject)

private

height, ~~width~~<sup>base</sup>: integer;

public

function area: integer;

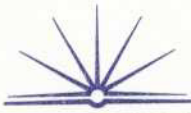
end;

function ~~TRectangle~~<sup>TTriangle</sup>. area: integer;

begin

area := (height \* width);

end;



d) The logic or declarative paradigm is best suited to this problem.

Once the barcode has been read on the bag the facts can be

passed through a set of rules which will determine which chute the

bag will be placed in: e.g. Rules:- chute1 (Firstclass, Sydney)

chute2 (Firstclass, Melbourne)

chute3 (Economy, Sydney)

and the query would be structured:

? (Class, Destination)

Informing the system of which chute to place the bag in.

These rules are extracted from the flight schedule each day. 'Compression rules' will show where the number of

active chutes can be reduced by adding to the rules. Eg.

Flight (Melbourne, Stopover)

Flight (Sydney, Terminate)

Flight (Adebride, Destination)

If a flight is merely stopping over or refuelling then the baggage need not be separated by destination in that case.