SPECIMEN MATERIAL

AS
COMPUTER SCIENCE
(7516/1C)

Paper 1 Pascal

Skeleton Program

```
Program Monster;

{Skeleton Program code for the AQA AS Unit 1 SAM this code should be used in
conjunction with the Preliminary Material written by the AQA Programmer Team
developed in the Turbo Pascal v7 programming environment}

{Centres using Delphi should add the compiler directive that sets
the application type to Console (other centres can ignore this
comment).  Centres may also add the SysUtils library if their
version of Pascal uses this}

{Permission to make these changes to the Skeleton Program does not
need to be obtained from AQA/AQA Programmer - just remove the \ symbol from
the next line of code  and remove the braces around Uses SysUtils;}

{\$APPTYPE CONSOLE}

{Uses
  SysUtils;}

Const NoOfTraps = 2;
Const NSDistance = 5;
Const WEDistance = 7;

Type
  TCellReference = Record
                     NoOfCellsSouth : Integer;
                     NoOfCellsEast : Integer;
                   End;
  TCavern = Array[1..NSDistance, 1..WEDistance] Of Char;
  TTrapPositions = Array[1..NoOfTraps] Of TCellReference;
  TGameData = Record
                TrapPositions : TTrapPositions;
                MonsterPosition : TCellReference;
                PlayerPosition : TCellReference;
                FlaskPosition : TCellReference;
                MonsterAwake : Boolean;
              End;

Var
  Cavern : TCavern;
  Choice : Integer;
  FlaskPosition : TCellReference;
  MonsterAwake : Boolean;
  MonsterPosition : TCellReference;
  PlayerPosition : TCellReference;
  TrapPositions : TTrapPositions;

Procedure DisplayMenu;
  Begin
    Writeln('MAIN MENU');
    Writeln;
    Writeln('1.  Start new game');
```

```
    Writeln('2.  Load game');
    Writeln('3.  Save game');
    Writeln('4.  Play training game');
    Writeln('9.  Quit');
    Writeln;
    Write('Please enter your choice: ');
  End;

Function GetMainMenuChoice : Integer;
  Var
    Choice : Integer;
  Begin
    Readln(Choice);
    Writeln;
    GetMainMenuChoice := Choice;
  End;

Procedure ResetCavern(Var Cavern : TCavern);
  Var
    Count1 : Integer;
    Count2 : Integer;
  Begin
    For Count1 := 1 to NSDistance
      Do
        For Count2 := 1 to WEDistance
          Do Cavern[Count1, Count2] := ' ';
  End;

Procedure GetNewRandomPosition(Var NewPosition : TCellReference);
  Var
    Position : TCellReference;
  Begin
    Repeat
      Position.NoOfCellsSouth := Random(NSDistance) + 1;
      Position.NoOfCellsEast := Random(WEDistance) + 1;
    Until (Position.NoOfCellsSouth > 1) Or (Position.NoOfCellsEast > 1);
    {a random coordinate of (1,1) needs to be rejected as this is the starting
position of the player}
    NewPosition := Position;
  End;

Procedure SetPositionOfItem(Var Cavern : TCavern; Var ObjectPosition :
TCellReference; Item : Char; NewGame : Boolean);
  Var
    Position : TCellReference;
  Begin
    If NewGame And (Item <> '*')
      Then
        Begin
          Repeat
            GetNewRandomPosition(Position);
          Until Cavern[Position.NoOfCellsSouth, Position.NoOfCellsEast] = ' ';
          ObjectPosition := Position;
        End;
```

```
    Cavern[ObjectPosition.NoOfCellsSouth, ObjectPosition.NoOfCellsEast] := Item;
  End;

Procedure SetUpGame(Var Cavern : TCavern; Var TrapPositions : TTrapPositions; Var
MonsterPosition,
  PlayerPosition , FlaskPosition: TCellReference;  Var MonsterAwake : Boolean;
NewGame : Boolean);
  Var
    Count : Integer;
  Begin
    ResetCavern(Cavern);
    If NewGame
      Then
        Begin
          PlayerPosition.NoOfCellsSouth := 1;
          PlayerPosition.NoOfCellsEast := 1;
          MonsterAwake := False;
        End;
    For Count := 1 To NoOfTraps
      Do SetPositionOfItem(Cavern, TrapPositions[Count], 'T', NewGame);
    SetPositionOfItem(Cavern, MonsterPosition, 'M', NewGame);
    SetPositionOfItem(Cavern, FlaskPosition, 'F', NewGame);
    SetPositionOfItem(Cavern, PlayerPosition, '*', NewGame);
  End;

Procedure SetUpTrainingGame(Var Cavern : TCavern; Var TrapPositions :
TTrapPositions; Var MonsterPosition,
  PlayerPosition, FlaskPosition : TCellReference; Var MonsterAwake : Boolean);
  Begin
    ResetCavern(Cavern);
    PlayerPosition.NoOfCellsSouth := 3;
    PlayerPosition.NoOfCellsEast := 5;
    MonsterAwake := False;
    TrapPositions[1].NoOfCellsSouth := 2;
    TrapPositions[1].NoOfCellsEast := 7;
    TrapPositions[2].NoOfCellsSouth := 4;
    TrapPositions[2].NoOfCellsEast := 5;
    MonsterPosition.NoOfCellsSouth := 1;
    MonsterPosition.NoOfCellsEast := 4;
    FlaskPosition.NoOfCellsSouth := 5;
    FlaskPosition.NoOfCellsEast := 6;
    SetUpGame(Cavern, TrapPositions, MonsterPosition, PlayerPosition,
FlaskPosition, MonsterAwake, False);
  End;

Procedure LoadGame(Var TrapPositions : TTrapPositions; Var MonsterPosition,
PlayerPosition,
                   FlaskPosition : TCellReference; Var MonsterAwake : Boolean);
  Var
    CurrentFile : File Of TGameData;
    Filename : String;
    LoadedGameData : TGameData;
  Begin
```

```
    Write('Enter the name of the file to load: ');
    Readln(Filename);
    Writeln;
    Assign(CurrentFile, Filename);
    Reset(CurrentFile);
    Read(CurrentFile, LoadedGameData);
    Close(CurrentFile);
    TrapPositions := LoadedGameData.TrapPositions;
    MonsterPosition := LoadedGameData.MonsterPosition;
    PlayerPosition := LoadedGameData.PlayerPosition;
    FlaskPosition := LoadedGameData.FlaskPosition;
    MonsterAwake := LoadedGameData.MonsterAwake;
  End;

Procedure SaveGame(TrapPositions : TTrapPositions; MonsterPosition,
PlayerPosition,
                   FlaskPosition : TCellReference; MonsterAwake : Boolean);
  Var
    CurrentFile : File Of TGameData;
    Filename : String;
    CurrentGameData : TGameData;
  Begin
    CurrentGameData.TrapPositions := TrapPositions;
    CurrentGameData.MonsterPosition := MonsterPosition;
    CurrentGameData.PlayerPosition := PlayerPosition;
    CurrentGameData.FlaskPosition := FlaskPosition;
    CurrentGameData.MonsterAwake := MonsterAwake;
    Write('Enter new file name: ');
    Readln(Filename);
    Writeln;
    Assign(CurrentFile, Filename);
    Rewrite(CurrentFile);
    Write(CurrentFile, CurrentGameData);
    Close(CurrentFile);
  End;

Procedure DisplayCavern(Cavern : TCavern; MonsterAwake : Boolean);
  Var
    Count1 : Integer;
    Count2 : Integer;
  Begin
    For Count1 := 1 To NSDistance
      Do
        Begin
          Writeln(' ------------- ');
          For Count2 := 1 To WEDistance
            Do
              If (Cavern[Count1, Count2] In [' ','*']) Or ((Cavern[Count1,Count2]
= 'M') And MonsterAwake)
                Then Write('|',Cavern[Count1, Count2])
                Else Write('| ');
          Writeln('|');
        End;
    Writeln( ' ------------- ');
```

```
    Writeln;
  End;

Procedure DisplayMoveOptions;
  Begin
    Writeln;
    Writeln('Enter N to move NORTH');
    Writeln('Enter E to move EAST');
    Writeln('Enter S to move SOUTH');
    Writeln('Enter W to move WEST');
    Writeln('Enter M to return to the Main Menu');
    Writeln;
  End;

Function GetMove : Char;
  Var
    Move : Char;
  Begin
    Readln(Move);
    Writeln;
    GetMove := Move;
  End;

Procedure MakeMove(Var Cavern : TCavern; Direction : Char; Var PlayerPosition :
TCellReference);
  Begin
    Cavern[PlayerPosition.NoOfCellsSouth, PlayerPosition.NoOfCellsEast] := ' ';
    Case Direction Of
      'N' : PlayerPosition.NoOfCellsSouth := PlayerPosition.NoOfCellsSouth - 1;
      'S' : PlayerPosition.NoOfCellsSouth := PlayerPosition.NoOfCellsSouth + 1;
      'W' : PlayerPosition.NoOfCellsEast := PlayerPosition.NoOfCellsEast - 1;
      'E' : PlayerPosition.NoOfCellsEast := PlayerPosition.NoOfCellsEast + 1;
    End;
    Cavern[PlayerPosition.NoOfCellsSouth, PlayerPosition.NoOfCellsEast] := '*';
  End;

Function CheckValidMove(PlayerPosition : TCellReference; Direction : Char) :
Boolean;
  Var
    ValidMove : Boolean;
  Begin
    ValidMove := True;
    If Not (Direction In ['N','S','W','E','M'])
      Then ValidMove := False;
    CheckValidMove := ValidMove;
  End;

Function CheckIfSameCell(FirstCellPosition, SecondCellPosition : TCellReference) :
Boolean;
  Var
    InSameCell : Boolean;
  Begin
    InSameCell := False;
```

```
    If (FirstCellPosition.NoOfCellsSouth = SecondCellPosition.NoOfCellsSouth)
      And (FirstCellPosition.NoOfCellsEast = SecondCellPosition.NoOfCellsEast)
      Then InSameCell := True;
    CheckIfSameCell := InSameCell;
  End;

Procedure DisplayWonGameMessage;
  Begin
    Writeln('Well done!  You have found the flask containing the Styxian
potion.');
    Writeln('You have won the game of MONSTER!');
    Writeln;
  End;

Procedure DisplayTrapMessage;
  Begin
    Writeln('Oh no!  You have set off a trap.  Watch out, the monster is now
awake!');
    Writeln;
  End;

Procedure MoveFlask(Var Cavern : TCavern; NewCellForFlask : TCellReference; Var
FlaskPosition : TCellReference);
  Begin
    Cavern[NewCellForFlask.NoOfCellsSouth, NewCellForFlask.NoOfCellsEast] := 'F';
    Cavern[FlaskPosition.NoOfCellsSouth, FlaskPosition.NoOfCellsEast] := ' ';
    FlaskPosition := NewCellForFlask;
  End;

Procedure MakeMonsterMove(Var Cavern : TCavern; Var MonsterPosition, FlaskPosition
: TCellReference;
  PlayerPosition : TCellReference);
  Var
    OriginalMonsterPosition : TCellReference;
    MonsterMovedToSameCellAsFlask : Boolean;
  Begin
    OriginalMonsterPosition := MonsterPosition;
    Cavern[MonsterPosition.NoOfCellsSouth, MonsterPosition.NoOfCellsEast] := ' ';
    If MonsterPosition.NoOfCellsSouth < PlayerPosition.NoOfCellsSouth
      Then MonsterPosition.NoOfCellsSouth := MonsterPosition.NoOfCellsSouth + 1
      Else
        If MonsterPosition.NoOfCellsSouth > PlayerPosition.NoOfCellsSouth
          Then MonsterPosition.NoOfCellsSouth := MonsterPosition.NoOfCellsSouth -
1
          Else
            If MonsterPosition.NoOfCellsEast < PlayerPosition.NoOfCellsEast
              Then MonsterPosition.NoOfCellsEast := MonsterPosition.NoOfCellsEast
+ 1
              Else MonsterPosition.NoOfCellsEast := MonsterPosition.NoOfCellsEast
- 1;
    MonsterMovedToSameCellAsFlask := CheckIfSameCell(MonsterPosition,
FlaskPosition);
    If MonsterMovedToSameCellAsFlask
      Then MoveFlask(Cavern, OriginalMonsterPosition, FlaskPosition);
```

```
    Cavern[MonsterPosition.NoOfCellsSouth, MonsterPosition.NoOfCellsEast] := 'M';
  End;

Procedure DisplayLostGameMessage;
  Begin
    Writeln('ARGHHHHHH!  The monster has eaten you.  GAME OVER.');
    Writeln('Maybe you will have better luck next time you play MONSTER!');
    Writeln;
  End;

Procedure PlayGame(Var Cavern : TCavern; TrapPositions : TTrapPositions; Var
MonsterPosition, PlayerPosition,
  FlaskPosition : TCellReference; Var MonsterAwake : Boolean);
  Var
    Count : Integer;
    Eaten : Boolean;
    FlaskFound : Boolean;
    MoveDirection : Char;
    ValidMove : Boolean;
  Begin
    Eaten:= False;
    FlaskFound := False;
    DisplayCavern(Cavern, MonsterAwake);
    Repeat
      Repeat
        DisplayMoveOptions;
        MoveDirection := GetMove;
        ValidMove := CheckValidMove(PlayerPosition, MoveDirection);
      Until ValidMove;
      If MoveDirection <> 'M'
        Then
          Begin
            MakeMove(Cavern, MoveDirection, PlayerPosition);
            DisplayCavern(Cavern, MonsterAwake);
            FlaskFound := CheckIfSameCell(PlayerPosition, FlaskPosition);
            If FlaskFound
              Then DisplayWonGameMessage;
            Eaten := CheckIfSameCell(MonsterPosition, PlayerPosition);
            If Not MonsterAwake And Not FlaskFound And Not Eaten
              Then
                Begin
                  MonsterAwake := CheckIfSameCell(PlayerPosition,
TrapPositions[1]);
                  If Not MonsterAwake
                    Then MonsterAwake := CheckIfSameCell(PlayerPosition,
TrapPositions[2]);
                  If MonsterAwake
                    Then
                      Begin
                        DisplayTrapMessage;
                        DisplayCavern(Cavern, MonsterAwake);
                      End;
                End;
```

```
            If MonsterAwake And Not Eaten And Not FlaskFound
              Then
                Begin
                  Count := 0;
                  Repeat
                    MakeMonsterMove(Cavern, MonsterPosition, FlaskPosition,
PlayerPosition);
                    Eaten := CheckIfSameCell(MonsterPosition, PlayerPosition);
                    Writeln;
                    Writeln('Press Enter key to continue');
                    Readln;
                    DisplayCavern(Cavern, MonsterAwake);
                    Count := Count + 1;
                  Until (Count = 2) Or Eaten;
                End;
              If Eaten
                Then DisplayLostGameMessage;
          End;
    Until Eaten Or FlaskFound Or (MoveDirection = 'M');
  End;

  Begin
    Randomize;
    Repeat
      DisplayMenu;
      Choice := GetMainMenuChoice;
      Case Choice Of
        1 : Begin
              SetUpGame(Cavern, TrapPositions, MonsterPosition, PlayerPosition,
FlaskPosition, MonsterAwake, True);
              PlayGame(Cavern, TrapPositions, MonsterPosition, PlayerPosition,
FlaskPosition, MonsterAwake);
            End;
        2 : Begin
              LoadGame(TrapPositions, MonsterPosition, PlayerPosition,
FlaskPosition, MonsterAwake);
              SetUpGame(Cavern, TrapPositions, MonsterPosition, PlayerPosition,
FlaskPosition, MonsterAwake, False);
              PlayGame(Cavern, TrapPositions, MonsterPosition, PlayerPosition,
FlaskPosition, MonsterAwake);
            End;
        3 : SaveGame(TrapPositions, MonsterPosition, PlayerPosition,
FlaskPosition, MonsterAwake);
        4 : Begin
              SetUpTrainingGame(Cavern, TrapPositions, MonsterPosition,
PlayerPosition, FlaskPosition, MonsterAwake);
              PlayGame(Cavern, TrapPositions, MonsterPosition, PlayerPosition,
FlaskPosition, MonsterAwake);
            End;
      End;
    Until Choice = 9;
  End.
```