

Trial Test2 Questions – all taken from "Test2" used over the last 2 years

A. Name an appropriate Lazarus control structure for each of the following situations:

- (a) a loop where we know the number of times it should cycle.
- (b) a simple binary decision.
- (c) a loop that needs to be done at least once, possibly many times.
- (d) a multiple decision.
- (e) a loop that may not need to be done at all

[5 marks KA]

B. Describe briefly the SYNTAX error(s) encoded in the following structures:

(a) for *index* := 0.1 to 0.9 do *showmessage*('hello')

[1 error = 1 mark KA]

(b) *thing* := *num* * 3.4;

```
case thing of
  0..4   : showmessage('low');
  3..8   : showmessage('middle');
  else   : showmessage('high');
end; {case}
```

[2 errors = 2 marks KA]

C. The following code, whilst syntactically correct, is written with MANY inefficiencies and a bunch of unnecessary stuff – re-write it and make it as efficient as you can (assume **ok:boolean** and **x:byte** are GLOBAL variables that already have been defined and are both needed elsewhere so MUST be assigned):

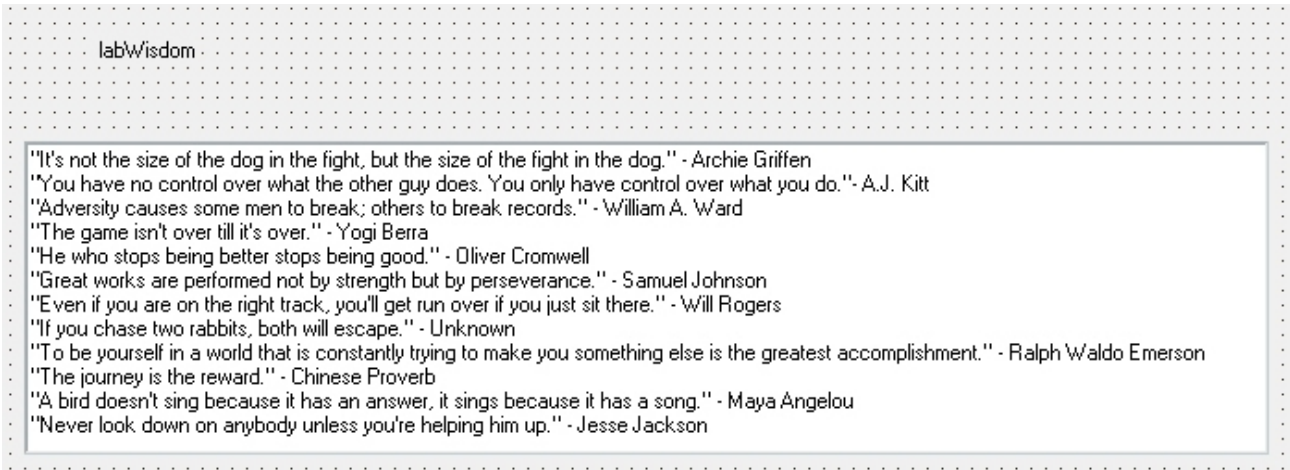
```
ok := false;
repeat
begin
  x := random(52);
  if x < 31
    then ok := true
    else ok := false
end
until not(ok = false)
```

[3 marks KA]

D. Write a simple ONCLICK process that asks the user for a string and then outputs that number of VOWELS (a,e,i,o,u) it contains. To gain marks for this question, you will make use of an INPUTBOX command for the user response and a SHOWMESSAGE for the output. You are free to invent a formname and sender and can safely assume the user enters the string in LOWERCASE.

[5 marks AS]

E. Given the following simple interface:



frmMain consists of a **label** (*labWisdom*) and a **visible:false memo box** (*memQuotes*).

Given the MOUSEDOWN event has the following form (and the x,y parameters give you the coordinates of the click):

```
FormMouseDown(Sender: TObject; Button: TMouseButton;  
              Shift: TShiftState; X, Y: Integer);
```

Write a simple form mousedown event that MOVES *labWisdom* to the location of the click and picks a RANDOM quote from the *memQuotes* to display there.

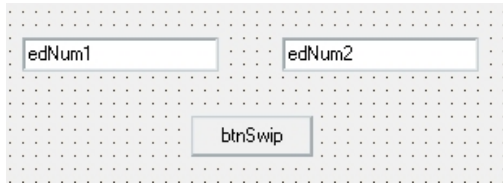
[move 2 + quote 3 = 5 marks AS]

F. Assuming stdctrls and extctrls are defined in the USES clause of a project, write the ONTIMER event (for a timer called rainmaker) so that each tick the following things happen:

- A NEW tshape (20x20, round, default colour) is spawned somewhere above the top edge of the form (randomly placed across the form)
- ALL tshapes on the form move down some random number of pixels between 5 and 10
- ANY tshapes that fall off the bottom of the form are permanently deleted

[3+5+2 marks AS]

- G. A *swip* is an imagined type of swapping algorithm where 2 numbers are conditionally swapped (only if they are not already smallest to biggest).



Suppose we design a simple Lazarus app with 2 edit boxes and a button that will be used for the *swip*. The user types a number in each of the boxes and then presses the button:



In the above example, the numbers were not in ascending order so they were swapped.



In the second example, the numbers were already in the right order so they were left alone (or *swipped*).

Code the `btnSwip` `onClick` event complete with event header. Be careful! Do not worry about error trapping the user-input (assume rather dangerously they will do the right thing)

[6 marks AS]

H. Many hotels and hospital wards provide a MINI-SAFE for the secure storage of valuables during the stay. The user interface is very simple (it needs to be for Americans to understand it). In a Lazarus-written simulation of the locking mechanism, we have a simple interface similar to the following:



To operate the lock, you start with it unlocked (...der), put your stuff in, close the door and then type a 5 DIGIT code (you make this up on the spot, presuming you can remember it again later) then press the LOCK button.



If there was a 5 digit number entered, the display turns to a "?" and says it is LOCKED, less than 5 digits nothing happens. The app should actively ignore anything after the 5th digit.

To open it again, the user re-types their 5 digit code and presses the UNLOCK button – if they got it right then the status changes to UNLOCKED



If the user types the wrong 5 digit sequence, after they press the UNLOCK button the interface tells the user it is incorrect and returns to LOCKED status

(a) Each of the numeric keypad buttons (btn0..btn9) use the SAME bit of code to conditionally add their digit to the **labCombo** as their **onclick** event – write it

[6 marks AS]

(b) The "oops" button (btnOops) click does one of 2 things, depending on your skill level as a programmer (use comments to indicate WHICH option you have chosen)

Either (i) clear the current combination entirely

[2 marks AS]

Or (ii) eat off the last entered digit (if there is one)

[6 marks AS]

(c) **Explain** clearly, with code fragments, what the LOCK and UNLOCK buttons must do to work as indicated – be sure to mention any GLOBAL variables you see necessary and explain the logic that will result in the described label displays

[EC criteria]