Questions, Projects, and Labs Chapter Six

6.1 Questions

1) What is a *First Class Object*?
2) What is the difference between deferred and eager evaluation?
3) What is a thunk?
4) How does HLA implement thunk objects?
5) What is the purpose of the HLA THUNK statement?
6) What is the difference between a thunk and procedure variable?
7) What is the syntax for declaring a thunk as a formal parameter?
8) What is the syntax for passing a thunk constant as an actual parameter?
9) Explain how an activation record's lifetime can affect the correctness of a thunk invocation.
10) What is a trigger and how can you use a thunk to create a trigger?
11) The yield statement in an iterator isn't a true HLA statement. It's actually equivalent to something else. What is it equivalent to?
12) What is a resume frame?
13) What is the problem with breaking out of a FOREACH loop using the BREAK or BREAKIF statement?
14) What is the difference between a coroutine and a procedure?
15) What is the difference between a coroutine and a generator?
16) What is the purpose of the coret call in the coroutines class?
17) What is the limitation of a coret operation versus a standard RET instruction?
18) What is the lifetime of the automatic variables declared in a coroutine procedure?
19) Where is the easiest place to pass parameters between two coroutines?
20) Why is it difficult to pass parameters between coroutines on the stack?
21) State seven places you can pass parameters between two procedures.
22) State at least six different ways you can pass parameters.
23) Where is the most efficient place to pass parameters?
24) Where do most high level languages pass their parameters?
25) What some problems with passing parameters in global variables?
26) What is the difference between the Pascal/HLA and the CDECL parameter passing mechanisms?
27) What is the difference between the Pascal/HLA and the STDCALL parameter passing mechanisms?
28) What is the difference between the STDCALL and the CDECL parameter passing mechanisms?
29) Provide one reason why some assembly code might require the caller to remove the parameters from the stack.
30) What is the disadvantage of having the caller remove procedure parameters from the stack?
31) Explain how to pass parameters in the code stream.
32) Describe how you might pass a reference parameter in the code stream. What is the limitation on such reference parameters?

33) Explain how you might pass a “pass by value/result” or “pass by result” parameter in the code stream.

34) What is a parameter block?

35) What is the difference between pass by value/result and pass by result?

36) What is the difference between pass by name and pass by lazy evaluation?

37) What parameter passing mechanism does pass by name most closely resemble?

38) What parameter passing mechanism does pass by lazy evaluation most closely resemble?

39) When passing a parameter by name or lazy evaluation, what does HLA actually pass on the stack.

40) What is the difference in the calling sequence between pass by reference, pass by value/result, and pass by result (assuming the standard implementation)?

41) Give an example where pass by value/result produces different semantics than pass by reference.

42) What parameter passing mechanism(s) support(s) deferred execution?

For each of the following subquestions, assume that a parameter (in) is passed into one procedure and that procedure passes the parameter on to another procedure (out). Specify how to do this given the following in and out parameter passing mechanisms (if possible):

43) Parameter is passed into the first procedure by value and passed on to the second procedure by:

a. value

b. reference

c. result

d. result

e: name

f: lazy evaluation

44) Parameter is passed into the first procedure by reference and passed on to the second procedure by:

a. value

b. reference

c. result

d. result

e: name

f: lazy evaluation

45) Parameter is passed into the first procedure by value/result and passed on to the second procedure by:

a. value

b. reference

c. result

d. result

e: name

f: lazy evaluation

46) Parameter is passed into the first procedure by result and passed on to the second procedure by:

a. value

b. reference

c. result

d. result

e: name

f: lazy evaluation

47) Parameter is passed into the first procedure by name and passed on to the second procedure by:

a. value

b. reference

c. result

d. result

e: name

f: lazy evaluation

48) Parameter is passed into the first procedure by lazy evaluation and passed on to the second procedure by:

a. value

b. reference

c. result

d. result

e: name

f: lazy evaluation

49) Describe how to pass a variable number of parameters to some procedure. Describe at least two different ways to do this.

50) How can you return a function's result on the stack?

51) What's the best way to return a really large function result?

52) What is a lex level?

53) What is a static link?

54) What does the term "scope" mean?

55) What is a "display"?

56) What does the term "address binding" mean?

57) What is the "lifetime" of a variable?

58) What is an intermediate variable?

59) How do you access intermediate variables using static links? Give an example.

60) How do you access intermediate variables using a display? Give an example.

61) How do you nest procedures in HLA?

62) What does the @lex function return?

63) What is one major difference between the *_display_* array and standard arrays?

64) What does the ENTER instruction do? Provide an algorithm that describes its operation.

65) What does the LEAVE instruction do? Provide an equivalent machine code sequence.

66) Why does HLA not emit the ENTER and LEAVE instructions in those procedures that have a display?

67) Provide a short code example that demonstrates how to pass an intermediate variable by value to another procedure.

68) Provide a short code example that demonstrates how to pass an intermediate variable by reference to another procedure.

69) Provide a short code example that demonstrates how to pass an intermediate variable by value/result to another procedure.

70) Provide a short code example that demonstrates how to pass an intermediate variable by result to another procedure.

71) Provide a short code example that demonstrates how to pass an intermediate variable by name to another procedure.

72) Provide a short code example that demonstrates how to pass an intermediate variable by lazy evaluation to another procedure.

6.2 Programming Problems

1) Rewrite Program 1.1 in Chapter One (Fibonacci number generation) to use a pass by reference parameter rather than a thunk parameter.

2) Write a function ifx that has the following prototype:

```
procedure ifx( expr:boolean; lazy trueVal:dword; lazy falseVal:dword );
```

The function should test *expr's* value; if true, it should evaluate and return *trueVal*, else it should evaluate and return *falseVal*. Write a main program that tests the execution of this function.

3) Write an iterator that returns all "words" of a given length. The iterator should have the following prototype:

```
iterator wordOfLength( length:uns32 );  // returns( "eax" );
```

The iterator should allocate a string with *length* characters on the heap, initialize this string, and return a pointer to the string in the EAX register. On each call to this iterator, it should return the next string of alphabetic characters using a lexicographical ordering. E.g., for strings of length three, the iterator would return aaa, aab , aac, aad, ..., aaz, aba, abb, abc, ..., zzz. Write a main program to test this word generator. Don't forget to free the storage associated with each string in the main program when you're done with the string.

4) Modify the program in programming project (3) so that it only returns strings that have a maximum of two consenants in a row and a maximum of three vowels in a row.

5) Write a "Tic-Tac-Toe" game that uses coroutines to make each move. One coroutine should prompt the "X" player for a move, the second coroutine should prompt the "O" player for a move (note that the moves are made by players, not by the computer). The main program/coroutine should call the other two coroutines and determine if there was a win/loss/draw after each move.

6) Modify programming project (5) so that the computer makes the moves for the "O" player.

7) Write a factorial function (n!) that passes a real80 parameter on the FPU stack and returns the real80 result on the FPU stack. (note: n! = 1*2*3*...*n).

8) Write the equivalent of cs.difference that passes the two character sets to the function in the MMX registers MM0 / MM1 and MM2 / MM3. Return the character sets' difference in MM0 / MM1.

9) Write a "printstr" procedure that expects a pointer to a zero-terminated sequence of characters to follow the call to *printstr* in the code stream. This procedure should print the string to the standard output device. A typcial call will look like the following:


```
static
       staticStrVar: char; nostorage;
           byte “Hello world”, 0;
           .
           .
           .
       call printstr
       dword staticStrVar
```

Note that this function must work with any zero terminated string; don’t assume the string is an HLA string. Write a main program that makes several calls to *printstr* and tests this function.

