Amrutha Reddy Jayachandran
FINAL Exam : Advanced Programming Languages

Final Exam Solutions

WSU ID: 11233619

TOTAL TIME TAKEN FOR EXAM IS 7-8 Hours approximately

Problem 1

Time Taken : 20 MINUTES
Given: The multiplication operator(juxtaposition) is left associative and has lower precedence.
The exponentiation operator (carrat symbol) is right associative and has higher precedence

- The parse tree must grow down towards the left for a operator which is left associative . Let this operator be called ' left-op'
  $< non-terminal > - >< non-terminal >< left-op >< terminal > | < terminal >$
- The parse tree must grow down towards the right for a operator which is right associative. Let this operator be called 'right-op'
  $< non-terminal > - >< terminal >< right-op >< non-terminal > | < terminal >$
- Following the above rules, a grammar to recognize an expression having the above both the multiplication and exponentiation operator is as follows: (First write the rule for lower precedence and then hightest precedence)
  $< t1 > - >< t1 >< JuxtapositionOperator >< t2 > | < t2 >$
  $< t2 > - >< t >< ExponentiationOperator >< t2 > | < t >$

Amrutha Reddy Jayachandran
FINAL Exam : Advanced Programming Languages

Final Exam Solutions

WSU ID: 11233619

Problem 2

Time Taken : 40 MINUTES

- Unused objects/memory in the program must be deallocated. This can be done either manually by the programmer or in an automatic fashion using the garbage collector.
- A language supports first class functions if we can pass functions as arguments to other functions, return them as the values from other functions, store them in data structures and assign them to variables.
- When a function calls another function, the caller's local variables are pushed onto the stack (activation record) so that they can be used when the control returns to the caller after executing the called function. But in first class functions since functions can be passed as arguments to other functions and functions can be returned, there are nested definitions of functions possible. There must be a way to handle sitiuations where the function refers to non-local variables which are not defined in the environment in which we have the function call, but are defined in the environment in which the function is defined.
- This deep binding is implemented by creating closures wherein there is a reference to the function and also an explicit representation of the associated referencing environment.
- In a first-class function, a caller can refer to the callee's state after the callee has returned. This requires that that the callee's activation record should not be deallocated when it returns the control to the caller. This violates the stack-based function call paradigm. To overcome this the memory can be allocated for such objects on a heap.
- Most of the functional programming languages support first-class-functions and higher order functions. Therefore they require the allocation of some objects on the heap(as discussed above). Most of the imperative programs like C do not allow nested functions, so that each function has it's own referencing environment and it can use the stack based memory allocation. Functional programs generate a lot of objects on the heap. Therefore automatic garbage collectors can be used to deallocate the unused objects which is more efficient and less error prone than manually deallocating objects.
- Therefore first-class function values are much more likely to be found in languages with garbage collection than in languages without garbage collection.

  REFERENCES:

  - Link: `http://en.wikipedia.org/wiki/Funarg_problem`

Amrutha Reddy Jayachandran
FINAL Exam : Advanced Programming Languages

Final Exam Solutions

WSU ID: 11233619

Problem 3

Time Taken : 1.5 HOURS

- The Imperative Language used: 'C'
- Implementation that I tested : ' Microsoft C compiler' (part of visual studio 2010)
- Source Code : Kindly refer to the attached folder 'FE-APL-Question3'. The source code is present in Question3-SourceCode.C
- Output Screen shots: Kindly refer to the attached foolder 'FE-APL-Question3'. The output screenshots are present in Question3-Output.doc.
- PROGRAM OUPUT:

  Initial value of variable a=10
  Function call: function1(a++,a++,a++)

  The Program was run in two modes :

  - CASE 1: DEBUG mode. Output : (Parameter1=12, Parameter2 =11, Parameter3 =10)
    From the output it is clear that the arguments to the function call were evaluated from Right to Left.
  - CASE 2: RELEASE mode. Output : (Parameter1=10, Parameter2 =10, Parameter3 =10)
    From the output it is clear that the argument was evaluated only once and then the same value was used for all three arguments.

- WHY the Difference in output?

  In C and C++ the order of evaluation of function arguments are undefined. Because of this there is no guarantee that the arguments passed to the function will be evaluated in the same order. This may lead to different results based on the implementation of the function (as seen in the output for the above program).

  In the Debug mode the optimization was not performed and all three arguments to the function were evaluated from Right to left. The compiler performed some optimization in the RELEASE mode since the order of evaluation is not defined in the language itself and gave an answer which is completely different from when the program was run in the DEBUG mode.

- In newer imperative programs like *C#* the order of evaluation of arguments is defined in the implementation of the language as 'Strictly Left to Right'
- REFERENCES:
  - Link: `http://www2.research.att.com/~bs/bs_faq2.html#evaluation-order`
  - Link: `http://blogs.msdn.com/b/oldnewthing/archive/2007/08/14/4374222.aspx`

Amrutha Reddy Jayachandran
FINAL Exam : Advanced Programming Languages

Final Exam Solutions

WSU ID: 11233619

Problem 4

Time Taken : 5 HOURS
The submitted program has 3 classes:

- IteratorHelper - A helper class for creating True Iterators. It uses a producer-consumer model for synchronization between the thread which is performing the iteration (using the yield() method) and the thread which is returning the iterated value (using the get() method).
- Range - An example True Iterator implemented using IteratorHelper. It implements the Iterator interface but does not support the remove method. It also implements the Runnable interface so that the iteration operation can happen in a separate thread from the caller of the next() method.
- Main - Test program for using the Range Iterator

Brief control flow explaining how the program works:

1. Range Iterator is created by calling its constructor Range(1,10).
   (a) This will create the IteratorHelper class and starts the iteration operation in a separate thread (Range::run method).
   (b) IteratorHelper::yield(1) is called
   (c) The yield method will add 1 to the internal list and waits for it to be consumed
2. Range::hasNext() method is called to check if the iterator has reached the end
   (a) This will call the IteratorHelper::done() method which waits for the Range's thread to be terminated or the list to be populated by an element. In this case, the wait in done method is satisfied when the list gets an element.
   (b) The done method returns false.
3. Range::next() method is called to get the next element from the Range iterator
   (a) This will call the IteratorHelper::get() method which waits for the Range's thread to be terminated or the list to be populated by an element (by calling the done() method). In this case, the list already contains an element and the done method returns false.
   (b) The element in the list is removed and returned to the caller.
   (c) This will also satisfies the wait in IteratorHelper::yield method and returns control to the Range::run method.
   (d) The Range::run method now continues the iteration and calls InteratorHelper::yield(2).
   (e) The flow continues as explained above.

REFERENCES:

- Link: `http://javamex.com/tutorials/synchronization_producer_consumer.shtml`
- Link: `http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html`