|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **QUESTION 4**   1. Programming design methodologies can have a large influence on language design. Which of the otpions below is an example of a programming design methodology?  |  |  |  | | --- | --- | --- | |  |  | Orthogonality | |  |  | Step-wise refinement | |  |  | von Neumann architecture |   **QUESTION 5**   1. Some criteria for evaluating programming languange design are contraditory, requiring the designer to balance them. Which pair of criteria below are contradictory?  |  |  |  | | --- | --- | --- | |  |  | Cost and Reliability | |  |  | Readability and Writability | |  |  | Simplicity and Orthogonality |   **QUESTION 6**   1. A programmer is learning a new language. The programmer was very quickly able to pick up the language and quick write programs in the language. However, the programmer struggles with the long name and syntax, making many mistakes. Which criteria below is this language likely to have a poor evaluation on?  |  |  |  | | --- | --- | --- | |  |  | Cost | |  |  | Readablity | |  |  | Writabiliy |   **QUESTION 7**   1. Which of the following grammar rules make '+' right associative?  |  |  |  | | --- | --- | --- | |  |  | E -> E + E | |  |  | E -> E + I | |  |  | E -> I + E | |  |  |  | |  |  |

1. Consider the following grammar:   
     
   S -> aAa | bBb   
   A -> bBA | aAB | bBb   
   B -> b | bB   
     
   Which of the folloing is a string in the language generated by this grammar?

|  |  |  |
| --- | --- | --- |
|  |  | aabbbbbbbbaa |
|  |  | bbbabbbbb |
|  |  | aabbbbbbbba |

**QUESTION 9**

1. Consider the following BNF:   
   1. S -> aA
   2. S -> bB
   3. S -> cC
   4. A -> a
   5. A -> aA
   6. A -> bB
   7. A -> cC
   8. B -> b
   9. B -> bB
   10. B -> cC
   11. C -> c
   12. C -> aA
   13. C -> cC

And the following static semantics: Except for the first letter of a string, 'c' cannot appear until at least 2 'b' appear.   
  
Which of the following use **inherited** attributes to implement the static semantics?

|  |  |  |
| --- | --- | --- |
|  |  | * 1. A.bCount = 0   2. B.bCount = 1   3. C.bCount = 0   4. A[2].bCount = A[1].bCount   5. B.bCount = A.bCount + 1   6. C.bCount = A.bCount **Predicate:** A.bCount > 1   7. B[2].bCount = B[1].bCount + 1   8. C.bCount = B.bCount **Predicate:** B.bCount > 1   9. **Predicate:** C.bCount > 1   10. A.bCount = C.bCount   11. c[2].bCount = C[1].bCount **Predicate:** C[1].bCount > 1 |
|  |  | * 1. **Predicate:** if A.hasC then A.bCount > 1   2. **Predicate:** if B.hasC then B.bCOunt > 0   3. **Predicate:** if C.hasC then C.bCount > 1   4. A.bCount = 0, A.hasC = false   5. A[1].bCOunt = A[2].bCount, A[1].hasC = A[2].hasC   6. A.bCount = B.bCount + 1, A.hasC = B.hasC   7. A.bCount = 0, A.hasC = true   8. B.bCount = 1, A.hasC = false   9. B[1].bCount = B[2].bCOunt+1, B[1].hasC = B[2].hasC   10. B.bCount = 0, B.hasC = true   11. C.bCount = 0, C.hasC = true   12. C.bCount = A.bCount, C.hasC = A.hasC   13. C.bCount = 0, C.hasC = true |
|  |  | * 1. A.bCount = 0   2. B.bCount = 1   3. C.bCount = 0   4. A.bCount = 0   5. A[2].bCount = A[1].bCount, A[1].bCount2 = A[2].bCount2 **Predicate:** A[1].bCount == A[2].bCount   6. B.bCount = A.bCount+1, A.bCount2 = B.bCount2+1   7. C.bCount = A.bCount, A.bCount2 = C.bCount2   8. B.bCount = 1   9. B[2].bCount = B[1].bCount+1, B[1].bCount2 = B[2].bCount2+1   10. C.bCount = B.bCount, B.bCount2 = C.bCount2   11. C.bCount = 0   12. A.bCount = C.bCount, C.bCount2 = A.bCount2 **Predicate:** C.bCount == A.bCount   13. C[2].bCount = C[1].bCount, C[2].bCount2 = C[1].bCount2 |

**QUESTION 10**

1. Consider the following grammar:   
   * S -> aA | bB
   * A -> b | aA
   * B -> a | abB

the following transfomation: 

* + replace all substring of 'a' with the number of 'a's in that substring, always taking the maximum number of 'a's you can
  + replace any 'b' with 1

and the following denotational semantics:

* + M(a)=1
  + M(b)=1
  + M(aA)= 10 + M(A)
  + M(bB) = 10 + M(B)
  + M(abB)= 110 + M(B)

If the "meaning" of a string in this language is the above transformation regarded as a decimal number, which statement about the denotational semantics is true?

|  |  |  |
| --- | --- | --- |
|  |  | The denotational semantics are almost completely wrong, and most rules need to be rewritten. |
|  |  | The denotational semantics perfectly specify the dynamic semantics of this language. |
|  |  | The denotational semantics require one of its rules to be modified before it is correct. |
|  |  | The denotational semantics specify the dynamic semantics, but is considerably more complicated than needed. |

**QUESTION 11**

1. Consider the following grammar:   
   * S -> aA | bB
   * A -> b | aA
   * B -> a | abB

the following transfomation: 

* + replace all substring of 'a' with the number of 'a's in that substring, always taking the maximum number of 'a's you can
  + replace any 'b' with 1

and the following operational semantics:

* + Start each program with: VAL num = 0
  + For each symbok in the string from left to right:
    - If the symbol is an 'a' add: VAL num = num +1
    - If the symbol is an 'b' and is followed by the end of the string add: VAL num = 10\*num+1
    - If the symbol is an 'b' and is folowed by an 'a': VAL num = 10\*(10\*num+1)
  + End each program with DISPLAY num

If the "meaning" of a string in this language is the above transformation regarded as a decimal number, which statement about the operational semantics is true?

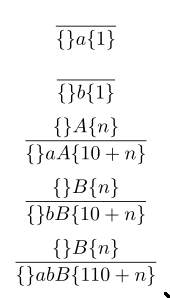
|  |  |  |
| --- | --- | --- |
|  |  | The operational semantics are almost completely wrong, and most rules need to be rewritten. |
|  |  | The operational semantics perfectly specify the dynamic semantics of this language. |
|  |  | The operational semantics require one of its rules to be modified before it is correct. |
|  |  | The operational semantics specify the dynamic semantics, but is considerably more complicated than needed. |

**QUESTION 12**

1. Consider the following grammar:   
   * S -> aA | bB
   * A -> b | aA
   * B -> a | abB

the following transfomation: 

* + replace all substring of 'a' with the number of 'a's in that substring, always taking the maximum number of 'a's you can
  + replace any 'b' with 1

and the following axiomatic semantics:   
  
   
  
If the "meaning" of a string in this language is the above transformation regarded as a decimal number, which statement about the axiomatic semantics is true?

|  |  |  |
| --- | --- | --- |
|  |  | The axiomatic semantics are almost completely wrong, and most rules need to be rewritten. |
|  |  | The axiomatic semantics perfectly specify the dynamic semantics of this language. |
|  |  | The axiomatic semantics require one of its rules to be modified before it is correct. |
|  |  | The axiomatic semantics specify the dynamic semantics, but is considerably more complicated than needed. |

**QUESTION 13**

1. Consider the following grammar:
   * S -> AB | CD
   * A -> a | bA | DB
   * B -> b | aA
   * C -> c | BD
   * D -> d | baC

According to the pairwise disjointness test is this grammar disjoint? Why?

|  |  |  |
| --- | --- | --- |
|  |  | No. The intersection of the first sets of both rules for S is not empty. |
|  |  | No. The second rule for A and the first rule for B start with the same symbol. |
|  |  | Yes. No rule for the same non terminal starts with the same symbol. |
|  |  | Yes. The intersection of the first sets for each rule for some nonterminal is empty. |

**QUESTION 14**

1. Consider the following left-recursive grammar:
   * L -> a | b | La | Lb

Which of the following grammars uses the algorithm from the book to eliminate the left-recursion?

|  |  |  |
| --- | --- | --- |
|  |  | * + L -> aM | bM   + M -> aM | bM | ε |
|  |  | * + L -> a | b | aL | bL |
|  |  | * + L -> aM | bM   + M -> aM | bM | a | b |

**QUESTION 15**

1. Consider the following rucursive descent parser:   
     
   void A() {   
      if( NEXT\_TOKEN == 'a' ) {   
         lex();   
         if( NEXT\_TOKEN == 'a') {   
            lex();   
            A();   
            lex();   
            C();   
         }   
         else   
           error();   
      }   
      else if(NEXT\_TOKEN == 'b') {   
            lex();   
            B();   
            lex();   
            C();   
      }   
      else   
         error();   
   }   
     
   void B() {   
      if( NEXT\_TOKEN == 'a' ) {   
         lex();   
         if( NEXT\_TOKEN == 'b' ) {   
            lex();   
            B();   
         }   
         else   
            error();   
      }   
      else if ( NEXT\_TOKEN != 'b' ) {   
          error();   
      }   
   }   
     
   void C() {   
      if( NEXT\_TOKEN == 'c' ) {   
         lex();   
         if( NEXT\_TOKEN == 'a' ) {   
            lex();   
            if( NEXT\_TOKEN != 'b' ) {   
                error();   
            }   
         }   
         else   
            error();   
      }   
      else   
         error();   
   }   
     
   Which of the following strings will parse without an error?

|  |  |  |
| --- | --- | --- |
|  |  | aabbcabcab |
|  |  | aacabbbcab |
|  |  | aaaabbbbcab |

**QUESTION 16**

1. Consider the following grammar:
   * S -> AA | BB
   * A -> a | aSa
   * B -> b | bSb

and the string: abbaabba   
  
Which of the following lists the phrases( using [] to denote the phrase) while labeling the simple phrases and handle?

|  |  |  |
| --- | --- | --- |
|  |  | [abbaabba], (simple) a[b]baabba, (simple) ab[b]aabba, a[bb]aabba, [abba]abba, (simple) abbaa[b]ba, (simple, handle) abbaab[b]a, abbaa[bb]a, abba[abba] |
|  |  | [abbaabba], (simple) a[b]baabba, (simple) ab[b]aabba, (simple) a[bb]aabba, [abba]abba, (simple) abbaa[b]ba, (simple, handle) abbaab[b]a, abbaa[bb]a, abba[abba] |
|  |  | [abbaabba], (simple, handle) a[b]baabba, (simple) ab[b]aabba, a[bb]aabba, [abba]abba, (simple) abbaa[b]ba, (simple) abbaab[b]a, abbaa[bb]a, abba[abba] |
|  |  | [abbaabba], (simple, handle) a[b]baabba, (simple) ab[b]aabba, (simple) a[bb]aabba, [abba]abba, (simple) abbaa[b]ba, (simple) abbaab[b]a, abbaa[bb]a, abba[abba] |

**QUESTION 17**

1. Recall the grammar and Shift-Reduce parser table from the book:   
     
   1. E -> E + T
   2. E -> T
   3. T -> T \* F
   4. T -> F
   5. F -> ( E )
   6. F -> id

**QUESTION 19**

1. Which of the following situations describe when a history-sensitive variable is useful?

|  |  |  |
| --- | --- | --- |
|  |  | Calling a function with large local variables |
|  |  | Implementing a random number generator function |
|  |  | To keep a log of changes to a state variable |

**QUESTION 20**

1. Recall that javascript does not have types, and the type of a variable is bound on assignment. What type of binding is this?

|  |  |  |
| --- | --- | --- |
|  |  | Explicit Heap-Dynamic |
|  |  | Implicit Heap-Dynamic |
|  |  | Stack-Dynamic |
|  |  | Static |

**QUESTION 21**

1. Consider the following program:   
     
   var x = 0;   
   function sub1() {   
      var x = 1;   
      function sub2() {   
         function sub3() {   
            print x;   
         }   
         sub3();   
      }   
      function sub4() {   
         var x = 2;   
         sub2();    
      }   
      sub4();   
   }   
   sub1();   
   If this code uses static scoping, when it is executed what is printed?

|  |  |  |
| --- | --- | --- |
|  |  | 0 |
|  |  | 1 |
|  |  | 2 |

**QUESTION 22**

1. Consider the following program:   
     
   var x = 0;   
   function sub1() {   
      var x = 1;   
      function sub2() {   
         function sub3() {   
            print x;   
         }   
         sub3();   
      }   
      function sub4() {   
         var x = 2;   
         sub2();    
      }   
      sub4();   
   }   
   sub1();   
   If this code uses dynamic scoping, when it is executed what is printed?

|  |  |  |
| --- | --- | --- |
|  |  | 0 |
|  |  | 1 |
|  |  | 2 |

**QUESTION 23**

1. Consider the following program:   
     
   void fun1 (void);   
   void fun2 (void);   
   void fun2 (void);   
     
   void main() {   
      int a, b, c;   
   }   
     
   void fun1() {   
      int b, c, d;   
   }   
     
   void fun2() {   
      int c, d, e;   
   }   
     
   void fun3() {   
      int d, e, f;   
   }   
   and the following call sequence: main calls fun2; fun2 calls fun1; fun1 calls fun3.   
     
   Assuming dynamic scoping, what are the variables visible within fun3?

|  |  |  |
| --- | --- | --- |
|  |  | d, e, and f from fun3 |
|  |  | d, e, and f from fun3; a, b, anc c from main |
|  |  | d, e, and f from fun3; b and c from fun 1; a from main |

**QUESTION 24**

1. Consider the following program:   
     
   void fun1 (void);   
   void fun2 (void);   
   void fun2 (void);   
     
   void main() {   
      int a, b, c;   
   }   
     
   void fun1() {   
      int b, c, d;   
   }   
     
   void fun2() {   
      int c, d, e;   
   }   
     
   void fun3() {   
      int d, e, f;   
   }   
   and the following call sequence: main calls fun1; fun1 calls fun3; fun3 calls fun2.   
     
   Assuming dynamic scoping, what are the variables visible within fun2?

|  |  |  |
| --- | --- | --- |
|  |  | a, b , and c from main; d from fun 1; e and f from fun3 |
|  |  | a from main; b from fun1; f from fun 3; c, d, and e from fun2 |
|  |  | c, d, and e from fun2 |

**QUESTION 25**

1. Why is a boolean usually not implemented as a single bit?

|  |  |  |
| --- | --- | --- |
|  |  | Bitwise operations would be needed to extract the bit |
|  |  | Storing a single bit is more expensive than an integer |
|  |  | The CPU cannot fetch a single bit from main memory |

**QUESTION 26**

1. In ALGOL, matrices are stored as a single dimensional array of pointers to the rows. Which statement about ALGOL matrices below is true?

|  |  |  |
| --- | --- | --- |
|  |  | Allows for simpler row manipulation in exchange for more complicated element access |
|  |  | Faster element access in exchange for more complicated row access |
|  |  | More compact representation in exchange for less efficient access of elements |

**QUESTION 27**

1. Given a 3-by-3, 2 dimensional array of bytes (size 1), A, stored in row-major order:   
   If the row indices start at 5, column indices start at 2, and the address of A[5,2] is 516, what is the address for A[7,3]?

|  |  |  |
| --- | --- | --- |
|  |  | 523 |
|  |  | 526 |
|  |  | 549 |

**QUESTION 28**

1. Why might we want to use references over pointers?

|  |  |  |
| --- | --- | --- |
|  |  | Increased expressively. References can do more than pointers. |
|  |  | Increased readability. References do not need to be passed by reference. |
|  |  | Increase reliability. References limit what can be pointed to. |

**QUESTION 29**

1. What is the difference between enumeration types in java and c++?

|  |  |  |
| --- | --- | --- |
|  |  | C++ enumerators can be coerced to an integer |
|  |  | Java enumerators are more efficient than c++ |
|  |  | Java enumerators can be String objects |

**QUESTION 30**

1. How does a decimal value waste memory?

|  |  |  |
| --- | --- | --- |
|  |  | It does not use the "exponent" part of the floating point |
|  |  | It takes less than a byte to represent a digit |
|  |  | Memory must be allotted for the decimal point |

**QUESTION 31**

1. Assume the following rules for associativity and precedence:

|  |  |  |
| --- | --- | --- |
| Precedence | Highest | \*,/,not |
|  |  | +, -, &, mod |
|  |  | - (unary) |
|  |  | =,/=,<,<=,>=,> |
|  |  | and |
|  | Lowest | xor, or |
| Associativity | Left to Right |  |

1. Given the string: a + b or c mod d = not e   
   Which parenthesised expression below represents the order of evaluation?

|  |  |  |
| --- | --- | --- |
|  |  | ((a + b) or (c mod d)) = (not e) |
|  |  | (((((a + b) or c) mod d) = (not e))) |
|  |  | ((a + b) or ((c mod d) = (not e))) |

**QUESTION 32**

1. Consider the following program: int f(int\* k) {   
     \*k -= 2;   
     return 2\*(\*k) + 1;   
   }   
     
   void main() {   
     int i = 5, j = 5, sum1, sum2;   
     sum1 = (i/2) + f(&i);   
     sum2 = f(&j) + (j/2);   
   }   
   What is the value of sum1 and sum2 if operands are evaluated right to left?

|  |  |  |
| --- | --- | --- |
|  |  | sum1 == 8, sum2 == 8 |
|  |  | sum1 == 8, sum2 == 9 |
|  |  | sum1 == 9, sum2 == 8 |
|  |  | sum1 == 9, sum2 == 9 |

**QUESTION 33**

1. Assume the following rules for associativity and precedence:

|  |  |  |
| --- | --- | --- |
| Precedence | Highest | \*,/,not |
|  |  | +, -, &, mod |
|  |  | - (unary) |
|  |  | =,/=,<,<=,>=,> |
|  |  | and |
|  | Lowest | xor, or |
| Associativity | Left to Right |  |

1. Given the string: 3 + 6 mod 7 - 5 & 6   
   Which parenthesised expression below represents the order of evaluation?

|  |  |  |
| --- | --- | --- |
|  |  | ((3 + (6 mod 7)) - (5 & 6)) |
|  |  | ((3 + (6 mod 7) - 5) & 6) |
|  |  | ((((3 + 6) mod 7) - 5) & 6) |

**QUESTION 34**

1. Consider the follow program:   
     
   int f(int \*i) {   
     \*i = (\*i>=0) ? \*i + 1 : 0;   
     return 4;   
   }   
     
   void main() {   
      int x = -5, y;   
     
      y = f(&x) + x + f(&x) - x - f(&x);   
   }   
   What is the value of y if operands are evaluated left to right? What is the value of y if operands are evaluated right to left?

|  |  |  |
| --- | --- | --- |
|  |  | left to right: 3, right to left: 3 |
|  |  | left to right: 3, right to left: 4 |
|  |  | left to right: 3, right to left: 5 |
|  |  | left to right: 4, right to left: 3 |
|  |  | left to right: 4, right to left: 4 |
|  |  | left to right: 4, right to left: 5 |
|  |  | left to right: 5, right to left: 3 |
|  |  | left to right: 5, right to left: 4 |
|  |  | left to right: 5, right to left: 5 |

**QUESTION 35**

1. Assume the following rules for associativity and precedence:

|  |  |  |
| --- | --- | --- |
| Precedence | Highest | \*,/,not |
|  |  | +, -, &, mod |
|  |  | - (unary) |
|  |  | =,/=,<,<=,>=,> |
|  |  | and |
|  | Lowest | xor, or |
| Associativity | Left to Right |  |

1. Given the string: -a or c = d and c   
   Which parenthesised expression below represents the order of evaluation?

|  |  |  |
| --- | --- | --- |
|  |  | ((-a) or ((c = d) and c)) |
|  |  | (((-a) or c) = (d and c)) |
|  |  | (((-a) or (c = d)) and c) |

**QUESTION 36**

1. COnsider the follow program: bool f(bool \*i) {   
       \*i = !(\*i);   
       return true;   
   }   
     
   void main() {   
      bool b = true, b2;   
     
      b2 = b && f(&b) && b && f(&b);   
   }   
   In the expression in the assignemnt to b2 what is actually evaluated?

|  |  |  |
| --- | --- | --- |
|  |  | b |
|  |  | b && f(&b) |
|  |  | b && f(&b) && b |
|  |  | b && f(&b) && b && f(&b) |