Sona College of Technology

Department of CSE

Compiler Design CIE questions

|  |  |  |
| --- | --- | --- |
| Would it be better if the phases of the compiler are combined into a single phase? | | |
| Differentiate compilers from interpreters. | | |
| An arithmetic expression with unbalanced parenthesis is lexical or syntax error. Comment on it. | | |
| A compiler that translates a high-level language into another high-level language is called a source- to-source translator. What advantages are there to use C as a target language for a compiler? | | |
| Point out why is buffering used in lexical analysis? What are the commonly used buffering methods? | | |
| Obtain the regular expressions for the following sets:   1. The set of all strings over {a, b} beginning and ending with ‘a’. 2. The set of all strings over {b} such that string belong to {b2 , b5 , b8 , …}. | | |
| Comment on the efficiency of the compiler if the number of passes in compilation is increased | | |
| What are the various representations of Intermediate languages? | | |
| Construct the syntax tree for the following assignment statement: a:=b\*- c+b\*-c. | | |
| Identify the type of error.  y = 3 + \* 5;  int a = "hello";  Justify this with the suitable points.. | | |
| Point out why is buffering used in lexical analysis? What are the commonly used buffering methods? | | |
| Describe the language denoted by the regular expressions a\*ba\*ba\*ba\*. | | |
| How can a lexical analyzer be constructed using Lex tool? | | |
| Draw the syntax tree for the augmented regular expression (ε | ab)\*(εa)bba# | | |
| Mention the basic issues in parsing. | | |
| What is the role of the parser in a compiler model? | | |
| Consider the grammar g=(V,T,P,S). Here V={S,N,Np,ADJ} and T={and, eggs, ham, pencilgreen, cold, tasty}. The set contains the following rules:  Np🡪Np and Np N 🡪eggs | ham | pencil|  Np🡪ADJ Np ADJ 🡪green |cold |tasty|  Np🡪N  Show that the grammar is ambiguous by constructing two different parse trees for the sentence “green eggs and ham”. | | |
| Eliminate Left recursion for the following grammar. | | |
| Write short notes on LEX. | | |
| Draw the syntax tree for the augmented expression (ab)ab\*( ε bb)# | | |
| Write a context free grammar that generates the set of all strings of a’s and b’s which is a palindrome. | | |
| How will you eliminate left factor in a grammar? | | |
| Draw transition diagrams for predictive parsers for the grammar  E 🡪E + T | T  T 🡪T \* F | F  F 🡪 (E) | id | | |
| Consider the grammar g=(V,T,P,S). Here V={S,N,Np,ADJ} and T={and, eggs, ham, pencil green, cold, tasty}. The set contains the following rules:  Np🡪Np and Np N 🡪eggs | bread| pencil|  Np🡪ADJ Np ADJ 🡪boiled |cold |tasty|  Np🡪N  Is this grammar ambiguous for the sentence “boiled eggs and bread”. | | |
| What is a front end and back end of a compiler? What are the advantages of breaking up the compiler functionality into these two distinct stages? | | |
| Obtain the regular expressions for the following having Σ={0,1}   1. strings of 0’s and 1’s beginning with 0 and ending with 1. 2. {x | x contains an even number of 0’s or an odd number of 1’s} | | |
| Depict diagrammatically and explain how a language is processed? | | |
|  | | |
| To verify the syntax of programming language constructs, compilers use CFG.Why not Regular expression?. Justify with examples. | | |
| Compute nullable(n), FIRSTPOS, LASTPOS and FOLLOWPOS for the regular expression  (ε | a)\*(ab)+ | | |
| Write the rules for computing the FIRST and FOLLOW functions for constructing predictive parsing table. | | |
| (i) | A compiler can choose one of the two options.   * + 1. Translate the input source into intermediate code and then convert it to final machine code.     2. Directly generate the final machine code from the input source.   What is the preferred option and why? Justify this with the suitable claims. |
| (ii) | Generate the target code for the following code fragment. Assume a=(b+c) \* (b+c) \* as input to the lexical analyzer and try to show the intermediate outputs of all the phases from lexical analyzer to code generator. |
| How do you speed up the reading the source program? What are the specialized algorithms used by the compiler? Did all of them read the source program with same time or they differ? Justify with suitable claims and figures. | |
| Can you identify the lexemes that make up the tokens in the following program segment? Write the corresponding tokens, lexemes and pattern.  **void swap(int i, int j)**  **{ int t; t=i; i=j; j=t;** | |
| [Do compiled languages beat interpreted languages in performance?](https://www.quora.com/Do-compiled-languages-beat-interpreted-languages-in-performance) [What is the difference between those languages?](https://www.quora.com/What-is-the-difference-between-a-high-level-and-low-level-computer-programming-language) Justify this with appropriate examples. | |
| Construct minimized Deterministic Finite Automata to accept the regular expression : (0+1)\* (00+11) (0+1)\*. | |
| Illustrate the output of each phase of compilation while translating the following assignment statement i = i\*70+j+2;. | |
| How to solve the source program to target machine code by using language processing system? | |
| Can you identify and list the systems/tools to help with the compiler-writing process are often been referred to as compiler-compilers, compiler-generators or translator-writing systems? Why are they called compiler-compilers? Write the representation and implementation techniques? What flexibility do they provide? | |
| Discuss the working of various phases of Compiler. Interpret the output for each phases for the expression a: = b + c \* 50. | |
| A compiler can choose one of the two options.   * + 1. Translate the input source into intermediate code and then convert it to final machine code.     2. Directly generate the final machine code from the input source.   What is the preferred option and why? Justify this with the suitable claims. | |
| int fact;  int factorial (int n)  {  int val;  if(n>1){  val=n\*factorial (n-1);  return (val);  }  else  {  return(1);  }  }  int main()  { printf (“factorial program”);  Fact 5=factorial(5);  Printf(“fact=5=%d \n”, fact5);  }  Consider the above program, What are the data structures used to store the information, how the symbol table management is handled? | |
| Describe the need of separating the analysis phase into lexical phase and parsing? | |
| Solve the given regular expression (a+b) abb (a/b)\* into NFA using Thompson construction and then to minimized DFA. . Describe the sequence of moves made by each in processing the input string ababba.  Explain in detail the structure of the compilers. Compare language dependent and independent phases | |
| Compare tokens, patterns and lexemes | |
| Count the number of tokens in the given snippet.  int main()  {  a=10,b=20;  printf(“ sum is : %d”, a+b;  return 0;  } | |
| Explain various Errors encountered in different phases of compiler. | |
| Obtain the required DFA from the regular expression (a/b)\* a (a/b) using Minimization of DFA using Π new constructions. Write down the suitable algorithm | |
| Explain in detail the structure of the compilers. Compare language dependent and independent phases. | |
| Construct transition diagram for keyword, identifiers and relational operators | |
| Is it possible to directly obtain a DFA from a regular expression? How is it different from obtaining DFA from an intermediate NFA? Obtain DFA directly for the regular expression (a/b)+c(c/d). | |
| How do we eliminate ambiguity from the dangling else grammar? | |
| Explain the structure of Lex program. Write Lex specifications for a Desk calculator application. | |
| Describe the different strategies that a parser can employ to recover from a syntactic error. | |
| Consider the following CFG G = (N={S, A, B, C, D}, T={a,b,c,d}, P, S) where the set of productions P is given below:  S → A  A → BC | DBC  B → Bb | ε  C → c | ε  D → a | d  Is this grammar suitable to be parsed using the recursive descendent parsing method? Justify and modify the grammar if needed | |
| Consider the grammar  bexpr -> bexpr or bterm | bterm  bterm -> bterm and bfactor | bfactor  bfactor -> not bfactor | ( bexpr ) | true | false   1. Construct a parse tree for the input string **not (true or false)**   Show that this grammar generates all boolean expressions. | |
| Illustrate the heuristic techniques for error-recovery in predictive parsing | |
| Consider the grammar    i) Construct FIRST and FOLLOW sets  ii)Construct the predictive parsing table  iii)Show the moves of the predictive parser for input (a,(a,a)) | |
| Left factoring is a technique to remove non-determinism. How left factoring does avoid backtracking in grammars? Apply left factoring in the following grammar to make it deterministic.  A→aBcC | aBb | aB | a  B →ε  C →ε | |
| How do SLR(1), LR(1) and LALR(1) methods compare against each other in the process of constructing the parsing table from the grammar? | |
| Consider the grammar  S→S(S)S | ε  Construct the predictive parsing table for the input (() () ) | |
| Consider the following (subset of a) CFG grammar  stmt 🡪 NIL | stmt ';' stmt | ifstmt | whilestmt | stmt  ifstmt 🡪 IF bexpr THEN stmt ELSE stmt  🡪 IF bexpr THEN stmt  whilestmt 🡪 WHILE bexpr DO stmt  where NIL, ';', IF, THEN, WHILE and DO are terminals, and "stmt", "ifstmt", "whilestmt" and "bexpr" are non-terminals.  For this grammar answer the following questions:  a) Is it ambiguous? Why? Is that a problem?  b) Rewrite the grammar so as to eliminate ambiguity | |
| Draw the precedence function graph for the following table. | |
| For the following three address code, give the quadruple representation  t1 := -r  t2 := q\*t1  t3 := -r  t4 := s \* t3  t5 := t2 + t4  p := t5 | |
| Construct LALR(1) parsers for the following grammar and parse the sentence id=id  S → L = R  S → R  L → \* R  L → id  R → L | |
| Only one occurrence of each object is allowable at a given moment during program execution. Justify your answer with respect to static allocation | |
| Design an activation tree for the Fibonacci sequence 1,1,2,3,5,8, ... defined by f(1)=f(2)=1 and, for n>2, f(n)=f(n-1)+f(n-2). Consider the function calls that result from a main program calling f(5).  system starts main  enter f(5)  enter f(4)  enter f(3)  enter f(2)  exit f(2)  enter f(1)  exit f(1)  exit f(3)  enter f(2)  exit f(2)  exit f(4)  enter f(3)  enter f(2)  exit f(2) s  enter f(1)  exit f(1)  exit f(3)  exit f(5)  main ends | |
| Consider the following grammar  S --> E  E --> E1 + T  E --> T  T --> T1 \* F  T --> F  F --> digit  Draw the annotated parse tree to compute s-attribute for the input string 4 \* 5 + 6. | |
| Write the program for dot product of two vectors. Optimize this code by  Eliminating the common sub expressions. | |
|  | |
| For the following three address code, give the triple and indirect triple implementation  t1 := -r  t2 := q\*t1  t3 := -r  t4 := s \* t3  t5 := t2 + t4  p := t5 | |
| Consider the grammar.  (1) S → A a  (2) S → b A c  (3) S → d c  (4) S → b d a  (5) A → d  Parse input string "bdc" constructing LALR Parsing Table. Is the grammar LALR(1)? | |
| Consider the following grammar:  E-> E+T|T  T-> TF|F  F-> F\*|a|b  Construct the SLR parsing table and also parse the input “a\*b+a” | |
| Design an activation tree for the for the following function calls:  enter main()  enter readarray()  leave readarry()  enter quicksort(1,9)  enter partition(1,9)  leave partition(1,9)  enter quicksort(1,3)  …..  leave quicksort(1,3)  enter quicksort(5,9)  …..  leave quicksort(5,9)  leave quicksort(1,9)  leave main() | |
| Evaluate the execution state of the following C program  Main()  {  int i;  int a[10];  i = 1;  While(i <= 10)  {  a[i] = 0;  i = i + 1;  }  }  into     1. Syntax tree. B.Postfix notation. C.3 address code. | |
| Consider the following grammar  S --> E  E --> E1 + T  E --> T  T --> T1 \* F  T --> F  F --> digit  Draw the annotated parse tree to compute s-attribute for the input string 3 \* 5 +5. | |
| Efficient code generation requires the Remember of internal architecture of the target machine. Justify your answer with an Example | |

.

.