**CD CAT1 QB – Answers**

|  |
| --- |
| **1. a)** Explain different types of translators with example. |
| **Ans 1. a)** A translator is a program that takes a program in high level language as input and produces a program in machine language. Beside program translation, the translator performs another very important role, the error-detection.  Types of Translators   1. Interpreter –  * An interpreter is a program that appears to execute a source program as if it were machine language. * An interpreter translates the entire source code line by line. * Examples of interpreted languages are **Perl, Python and MATLAB**  1. Compiler –  * A compiler is a program that translates a high-level language program into a functionally equivalent low-level language program. * A compiler translates the entire source code in a single run. * Examples of compiled languages are **C, C++, C#, Java** |
| 1. b) Explain various phases of compilers comes under front end. |
| **Ans 1. b) Phases of Compilers - Front End**   1. Lexical Analyzer (Scanner)  * Left to Right scanning * Then separation of continuous expressions into single element which we will call ‘Token’. * Example   d = a + b \* c     * Steps we performed are: -  1. Scanning   Parse Tree   1. Tokenization 2. Naming of each token 3. Syntax Analyzer (Parser)  * Syntax Analyzer 🡪 checks syntax * Semantic Analyzer 🡪 checks logic * Generates a Parse Tree  1. Intermediate Code Generation  * Three Address Code * c = a + b   }   * Example   id = id + id \* id  id  id = id + id ∵ (id \* id = id) |
|  |
| 2. a) Explain the following terms. |
| **Ans 2. a)**  i) Cross Compilers   * Cross compiler is a compiler that runs on one machine & produce the object code on another machine. * The Cross Compiler is used to implement the compiler which is characterized by 3 languages  1. The Source Language 2. The Object Language 3. The Language in which it is written |
| ii) Bootstrap Compilers   * The compiler which is written in its own language is called bootstrap compiler. * The technique for producing a self-compiling compiler |
| iii) Just in time Compiler   * A way of executing computer code that involves compilation during execution of a program (at run time) rather than before execution. * This reduces overall time taken for compilation of code to machine language. |
| 2. b) Explain various phases of compilers comes under back end. |
| **Ans 2. b) Phases of Compilers - Back End**   1. Code optimization  * Attempt to improve the intermediate code * necessary to have a faster executing code or less consumption of memory.  1. Code generation  * Generates code for the target machine * Assembly code |
| 3. a) Explain Top-Down Parser with example. |
| **Ans 3. a)** Top-Down Parser:   * A parsing technique that involves starting with the highest-level nonterminal symbol of the grammar and working downward to derive the input string. * Example:   S 🡪 Start Symbol  ↓  ω 🡪 String of terminal symbol  S 🡪 aAb   |  | | --- | | A 🡪 cd | | A 🡪 c |   A 🡪 cd/c }  ω 🡪 abc } – Required string |
| 3. b) Find the FIRST () and FOLLOW () for the following grammar.   * S→ aIJh * I → IbSe / c * J→ KLKr / Є * K→ d / Є * L→ p / Є |
| **Ans 3. b)**   * S→ aIJh * I → cI’ * I’ → bSeI’ / Є * J→ KLKr / Є * K→ d / Є * L→ p / Є   FIRST(S)🡪{a}  FIRST(I)🡪{c}  FIRST(I’)🡪{b,Є}  FIRST(J)🡪{d,p,r,Є}  FIRST(K)🡪{d, Є}  FIRST(L)🡪{p, Є}  FOLLOW(S)🡪{$}  FOLLOW(I)🡪 {p,d,r,h}  FOLLOW(I’)🡪{p,d,r,h}  FOLLOW(J)🡪{h}  FOLLOW(K)🡪{p,d,r}  FOLLOW(L)🡪{d,r} |
| 4. a) Show whether given grammar is LL(1) or not.   * S → AaAb/BbBa * A→ Є * B→ Є |
| **Ans 4. a)** Condition for LL(1) - **∵ FIRST( α ) ∩ FIRST( β ) = { Φ }**  FIRST (AaAb) 🡪 FIRST (A) – {ε} U FIRST (aAb)  FIRST (BbBa) 🡪 FIRST (b) – {ε} U FIRST (bBa)  Predictive Parsing Table:  **FIRST (AaAb) ∩ FIRST (BbBa)) = { Φ }**   |  |  |  |  | | --- | --- | --- | --- | |  | a | b | $ | | S | S → AaAb | S → BbBa |  | | A | A→ Є | A→ Є |  | | B | B→ Є | B→ Є |  |   ∴ The given grammar is **LL(1)**  FIRST (A) 🡪 {ε}  FIRST (B) 🡪 {ε}  FIRST (S) 🡪 FIRST (AaAb) U FIRST (BbBa)  🡪 {a} U {b}  🡪 {a,b}  FOLLOW (S) 🡪 {$}  FOLLOW (A) 🡪 FIRST (aAb) | FIRST (b)  🡪 {a} | {b}  🡪 {a,b}  FOLLOW (B) 🡪 FIRST (Bba) | FIRST (a)  🡪 {b} | {a}  🡪 {b,a} |
| 4. b) Explain why Top-Down Parser is called left most derivative Parser.  **Ans 4. b)**  The reason that top-down parsing follows the left-most derivation for an input string ω and not the right-most derivation is that **the input string ω is scanned by the parser from left to right, one symbol/token at a time**. |
| 5. a) Find the reduced grammar equivalent to CFG   * G {S A B C } { a b d} S P where P contains = ( , , , , , , , , ) * S → AC / SB * A→ bASC / a * B → aSB / bbC * C → Bc / ad |
| 5. b) Consider the following Grammar   * E → TA * A → +TA/ϵ * T → FB * B → \*FB/ϵ * F → (E)/id   Find FIRST () and FOLLOW () for each and every non terminal |
| **Ans 5. b)**  FIRST(E) → { ( , i }  FIRST(A) → { + , ϵ }  FIRST(T) → { ( , ϵ }  FIRST(B) → { \* , ϵ }  FIRST(F) → { ( , i }  FOLLOW(E) → { $, ) }  FOLLOW(A) → { $, ) }  FOLLOW(T) → { +, $, ) }  FOLLOW(B) → { +, $, ) }  FOLLOW(F) → { \*, +, $, ) } |
| 6. a) Compare SLR C LR and LALR Parser |
| **Ans 6. a)**   |  |  |  | | --- | --- | --- | | **SLR Parser** | **LALR Parser** | **CLR Parser** | | It is very easy and cheap to implement. | It is also easy and cheap to implement. | It is expensive and difficult to implement. | | SLR Parser is the smallest in size. | LALR and SLR have the same size. As they have a smaller number of states. | CLR Parser is the largest. As the number of states is very large. | | Error detection is not immediate in SLR. | Error detection is not immediate in LALR. | Error detection can be done immediately in CLR Parser. | | SLR fails to produce a parsing table for a certain class of grammars. | It is intermediate in power between SLR and CLR i.e., SLR ≤ LALR ≤ CLR. | It is very powerful and works on a large class of grammar. | | It requires less time and space complexity. | It requires more time and space complexity. | It also requires more time and space complexity. | |
| 6. b) Consider the following grammar:   * S → / / aSbS bSaS є   a) Show that this grammar is ambiguous by constructing two different leftmost derivation for the sentence abab. |
|  |
| 7. a) Consider the following grammar   * S → ABC * A→ a/ϵ * B→ r/ϵ * C→ b/ϵ   Construct parsing table with LL Parser |
| **Ans 7. a)**  FIRST (A) 🡪 FIRST (a) U FIRST (ε)  🡪 {a} U {ε}  🡪 {a, ε}  FIRST (B) 🡪 FIRST (r) U FIRST (ε)  🡪 {r} U {ε}  🡪 {r, ε}  FIRST (C) 🡪 FIRST (b) U FIRST (ε)  🡪 {b} U {ε}  🡪 {b, ε}  FIRST (S) 🡪 FIRST (ABC)  🡪 FIRST (A) - {ε} U FIRST (BC)  🡪 {a, ε} - {ε} U {r, ε} U {b, ε}  🡪 {a, ε} - {ε} U {r, ε} U {b, ε}  🡪 {a} U {r, b, ε}  🡪 {a, r, b, ε}  FOLLOW (S) 🡪 {$}  Parsing Table (Not Sure)  FOLLOW (A) 🡪 FIRST (BC) - {ε} U FOLLOW (S)   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | a | b | r | $ | | S | S → ABC | S → ABC | S → ABC |  | | A | A→ a | A→ ϵ | A→ ϵ | A→ ϵ | | B |  | B→ ϵ | B→ r | B→ ϵ | | C |  | C→ b |  | C→ ϵ |   🡪 {r, b, ε} - {ε} U {$}  🡪 {r, b, $}  FOLLOW (B) 🡪 FIRST (C) - {ε} U FOLLOW (S)  🡪 {b, ε} - {ε} U {$}  🡪 {b, $}  FOLLOW (C) 🡪 {b, $} |
| 7. b) Consider the following grammar   * E → CC * C→ cC * C→ d   Construct LR(0) Parser. |
|  |
| 8. a) Show, Quadruple Triple and Indirect triples for the following  expression.  - (a + b) \* (c + d) + (a + b + c) |
| **Ans 8. a)**   1. Quadruple Representation  |  |  |  |  |  | | --- | --- | --- | --- | --- | | Location | Operator | Operand 1 | Operand 2 | Result | | (1) | + | a | b | t1 | | (2) | + | c | d | t2 | | (3) | \* | t1 | t2 | t3 | | (4) | + | t1 | c | t4 | | (5) | + | t3 | t4 | t5 | | (6) | - | t5 |  |  |  1. Triple Representation  |  |  |  |  | | --- | --- | --- | --- | | Location | Operator | Operand 1 | Operand 2 | | (1) | + | a | b | | (2) | + | c | d | | (3) | \* | (1) | (2) | | (4) | + | (1) | c | | (5) | + | (3) | (4) | | (6) | - | (5) |  |  1. Indirect Triple Representation  |  | | --- | | Location | | (1) | | (2) | | (3) | | (4) | | (5) | | (6) | |
| 8. b) Find the TAC for following code: - if (B >) D and A < C then P an else Q b = +1 = +1; |
|  |
| 9. a) For the given grammar:   * E-> E + T/T * T-> T \* F/F * F-> (E)/id   Construct parse tree and syntax tree for string  w = id + id \* id |
| **Ans 9. a)**  Parse Tree  Syntax Tree |
|  |
| 9. b) Differentiate between Synthesized Attributes and Inherited Attributes.  **Ans 9. a)**   |  |  |  | | --- | --- | --- | | **S.NO** | **Synthesized Attributes** | **Inherited Attributes** | | 1. | An attribute is said to be Synthesized attribute if its parse tree node value is determined by the attribute value at child nodes. | An attribute is said to be Inherited attribute if its parse tree node value is determined by the attribute value at parent and/or siblings’ node. | | 2. | The production must have non-terminal as its head. | The production must have non-terminal as a symbol in its body. | | 3. | A synthesized attribute at node n is defined only in terms of attribute values at the children of n itself. | An Inherited attribute at node n is defined only in terms of attribute values of n’s parent, n itself, and n’s siblings. | | 4. | It can be evaluated during a single bottom-up traversal of parse tree. | It can be evaluated during a single top-down and sideways traversal of parse tree. | | 5. | Synthesized attributes can be contained by both the terminals or non-terminals. | Inherited attributes can’t be contained by both, it is only contained by non-terminals. | | 6. | Synthesized attribute is used by both S-attributed SDT and L-attributed SDT. | Inherited attribute is used by only L-attributed SDT. | | 7. |  |  | |