|  |  |  |
| --- | --- | --- |
| **P.V.P SIDDHARTHA INSTITUTE OF TECHNOLOGY** | | |
| **BRANCH : Computer Science & Engineering** | | **REGULATION : PVP20** |
| **Course: B.Tech** | **SUBJECT :Compiler Design** | |
| **SubjectCode:20CS3601** | | **Year and Semester: III-II** |
| **Sample Questions** | | |

**UNIT I**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Q. NO.** | **QUESTION** | **CO** | **LEVEL** | **Marks** |
|  | Differentiate compiler and interpreter. Explain the need for dividing the compilation process into various phases and explain its functions. | CO1 | L2 | 14M |
|  | a) Explain the diagrammatic representation of a language processing system. | CO1 | L2 | 7M |
| b) Describe the Structure of LEX program | CO1 | L2 | 7M |
|  | Draw the block diagram of phases of a compiler and indicate the main functions of each phase. | CO1 | L2 | 14M |
|  | Demonstrate the output of each phase of compiler for the expression “position: = initial + rate \* 60” | CO1 | L2 | 14M |
|  | a) Explain the role and functions of lexical analyzer with the possible error recovery actions. | CO1 | L2 | 7M |
| b) Illustrate Input Buffering in detail | CO1 | L2 | 7M |
|  | Build a LEX specification to read a C program and calculate the number of new line characters, tabs and white spaces in the program. | CO1 | L2 | 14M |
|  | a) Write about lexical analyzer generator in detail. | CO1 | L2 | 7M |
| b) Discuss various phases of compiler and trace the program  segment c=a\*b+4 for all phases. | CO1 | L2 | 7M |
|  | Define lexeme, token and pattern. Identify the lexemes, that make up the tokens in the following program segment. Indicate corresponding token and pattern.   void swap(int i, int j)  {  int t;  t=i; i=j; j=t;  } | CO1 | L2 | 14M |
|  | a) Differentiate analysis and synthesis phases of the compiler | CO1 | L2 | 7M |
| b) Write down the output of each phase for the expression  a:=b+ c\*50. | CO1 | L2 | 7M |
|  | a) Explain the input buffer scheme for scanning the source program. How the use of sentinels can improve its performance? Describe in detail. | CO1 | L2 | 7M |
| b) Write a LEX program for identifying the keywords and identifiers from the file? | CO1 | L2 | 7M |

**UNIT II**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Q. NO.** | **QUESTION** | **CO** | **LEVEL** | **Marks** |
| 1 | a) Calculate FIRST and FOLLOW sets for the following grammar:  E→TE’ E’→+TE’ | ε T→FT’ T’→\*FT’ | ε F→(E) | id | CO2 | L3 | 7M |
| b) Explain error-recovery techniques in predictive parsing. | CO2 | L2 | 7M |
| 2 | a) Construct predictive parsing table for the grammar  E→E+T T→ T\*F F→ (E) |id | CO2 | L3 | 7M |
| b) Eliminate ambiguities in the following grammar.  S → iEtS|iEtSeS|a E → b|c|d | CO2 | L3 | 7M |
| 3 | a) Find the FIRST and FOLLOW sets for each of the non-  terminals of the following grammar  S→aAB|bA|εA→aAb| ε B→bB|c | CO2 | L3 | 7M |
| b) What are the differences between LL(1) Parsing and LL(k) Parsing? | CO2 | L2 | 7M |
| 4 | Justify whether the given grammar is LL(l) or not  S→aBDh, B→cG, G→bc|ϵ, D→ EF, E→g|ϵ, F→ f |c | CO5 | L4 | 14M |
| 5 | a) Explain the model of a non-recursive predictive parser with an example. | CO2 | L2 | 7M |
| b) Construct LL(1) parse table for the following grammar.  S→Aa|bAc|Bc|bBa A→d B→d | CO2 | L3 | 7M |
| 6 | Construct predictive parsing table for the following grammar:  S → (L) | a L →L,S |S  and check whether the following sentences belong to that grammar or not.  (a,a) , (a, (a , a)) , (a, ((a , a)) | CO2 | L3 | 14M |
| 7 | a) Find FIRST and FOLLOW for the following grammar  S → aAbB | bAaB | ε A → S B → S | CO2 | L2 | 7M |
| b) Compare top-down parsing and bottom-up parsing. | CO5 | L4 | 7M |
| 8 | a) Check whether the following grammar is a LL(1) or not  S → iEtS|iEtSeS|a E->b | CO5 | L4 | 7M |
| b) Eliminate left recursion for the following grammar  E-->E+T/T T-->T\*F/F F-->(E)/id | CO2 | L3 | 7M |
| 9 | a) Write down the necessary algorithms for FIRST and FOLLOW | CO2 | L2 | 7M |
| b) Construct predictive parser for the following grammar  S-->(L)/a L-->L, S/S and parse any input string. | CO2 | L3 | 7M |
| 10 | a) Verify whether string “id+(id+id)" is accepted by following grammar or not by using predictive parsing:  E -> TE’ E’ -> +TE’/ ϵ T -> FT’ T’ -> \*FT’/ ϵ F -> (E)/id. | CO5 | L4 | 7M |
| b) What are the merits & demerits of recursive descent parsing. | CO2 | L2 | 7M |

**UNIT III**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Q. NO.** | **QUESTION** | **CO** | **LEVEL** | **Marks** |
|  | 1. Obtain the stack implementation of shift reduce parser for the input string abbcde using the following grammar   S→aABe A→Abc| b B→d | CO3 | L3 | 7M |
| 1. What is Handle Pruning? Perform handle pruning for the string id\*id\*id by considering the grammar E→E\*E | id | CO3 | L3 | 7M |
|  | a) Explain about the LR parsing algorithm | CO3 | L2 | 7M |
| b) Compute LR (0) items for the following grammar:  S→AB A→a | ε B→b | ε | CO3 | L3 | 7M |
|  | 1. Explain the procedure for constructing set of LR(0) items | CO3 | L2 | 7M |
| b) Compute LR(0) items for the following grammar:  S→L=R | R L→\*R | id R→L | CO3 | L3 | 7M |
|  | 1. Obtain the stack implementation of shift reduce parser for the input string id1+ id2\* id3 for the following grammar   E→E+E E→E\*E E→ (E) E→id | CO3 | L3 | 7M |
| b) Explain the way to implement a shift-reduce parser using a stack by taking an input string for a grammar. | CO3 | L2 | 7M |
|  | a) Construct SLR parsing table for  S→abS|AAab|b A→baAb|b | CO3 | L3 | 7M |
| b) Discuss about shift reduce parsing conflicts that arise during parsing. | CO3 | L2 | 7M |
|  | 1. Construct SLR parsing table for   S→abS|AAab|b A→baAb|b | CO3 | L3 | 7M |
| 1. Discuss operations performed in the construction of SLR parsing table. | CO3 | L2 | 7M |
|  | 1. Check whether the following grammar is SLR   S → L = R | R L → \* R | id R→ L | CO5 | L4 | 7M |
| 1. What is a shift reduce parser? Mention the conflicts that occur during shift-reduce parsing. | CO3 | L2 | 7M |
|  | 1. Explain about the steps for constructing SLR parsing table with an example? | CO3 | L2 | 7M |
| 1. Consider the following grammar   S->TL; T->int/float L->L, id/id  parse the input string int id,id; using SR parser. | CO3 | L3 | 7M |
|  | a) Consider the following grammar  S->(L)/a L->L,S/S  parse the input string (a,(a,a)) using SR parser | CO3 | L3 | 7M |
| b) Write the properties of LR parser with its structure. Also explain the techniques of LR parser. | CO3 | L2 | 7M |
|  | 1. Consider the grammar:   E-> E+E E-> E\*E E->id  Perform shift reduce parsing of the input string “id1+id2+id3”. | CO3 | L3 | 7M |
| 1. Perform handle pruning for the string "kwwrgz” by considering the grammar   A->kXYz X->Xwr | w Y->g | CO3 | L3 | 7M |

**UNIT IV**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Q. NO.** | **QUESTION** | **CO** | **LEVEL** | **Marks** |
|  | 1. Construct the CLR parsing table for the following grammar:   S→L=R | R L→\*R | id R→L | CO3 | L3 | 7M |
| 1. What are the limitations of stack allocation? | CO4 | L2 | 7M |
|  | 1. Check whether the grammar is LALR or not   S-Aa/bAc/dc/bda A->d | CO5 | L4 | 7M |
| 1. write the quadruple, triple and indirect triple for the following expression   (x+y)\*(y+z)+(x+y+z) | CO4 | L3 | 7M |
|  | a) Explain the procedure for constructing LALR parsing tables. | CO3 | L2 | 7M |
| b) What is activation record? Write the various fields of Activation  Record. | CO4 | L2 | 7M |
|  | a) Construct LALR parsing table for the following grammar  S→A S|b A→S A|a | CO3 | L3 | 7M |
| b) Write an algorithm for constructing LALR Parser table. | CO3 | L2 | 7M |
|  | 1. Show the following grammar   S->Aa|bAc|Bc|bBa  A->d  B->d  Is LR(1) but not LALR(1). | CO5 | L4 | 7M |
| 1. Compare the different storage allocation strategies. | CO4 | L2 | 7M |
|  | 1. Construct syntax tree and postfix notation for the following expression: (a+(b\*c)^d-e/(f+g) | CO4 | L3 | 7M |
| 1. Construct LALR parsing table for the following grammar   S→Aa|Ac|c|Ba  A→d  B→d | CO3 | L3 | 7M |
|  | a) Write quadruples, triples and indirect triples for the expression:  -(a\*b)+(c+d)-(a+b+c+d) | CO4 | L3 | 7M |
| 1. Compare SLR, CLR and LALR parsers. | CO5 | L4 | 7M |
|  | 1. Compare Static allocation, Stack allocation and Heap Allocation with their merits and limitations. | CO4 | L2 | 7M |
| 1. Translate the expression - (a+b)\*(c+d) + (a+b+c) into quadruple, triple and indirect triple. | CO4 | L3 | 7M |
|  | a) What is intermediate code and write the two benefits of intermediate code generation. | CO4 | L2 | 7M |
| b) Construct the CLR parsing table for the following grammar:  S→L=R | R L→\*R | id R→L | CO3 | L3 | 7M |
|  | a) Write the quadruple, triple, indirect triple for the following expression.  (x + y)∗(y + z) + (x + y + z) | CO4 | L3 | 7M |
| b) Construct CLR parsing table for the given grammar  S->CC C->aC/d | CO3 | L3 | 7M |

**UNIT V**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Q. NO.** | **QUESTION** | **CO** | **LEVEL** | **MARKS** |
|  | 1. Describe in detail about DAG representation with example. | CO4 | L2 | 7M |
| 1. Construct a DAG for the following program code:   x=y\*z  w=p+y  y=y\*z  p=w-x | CO4 | L3 | 7M |
|  | a) Consider following basic blocks:  i) T1=b+c  T2=d-c  T3=b+c  T4=T2\*T3  T5=T4\*f  x=T1-T5  Which of the local optimization techniques are possible to be carried out with above basic block. | CO4 | L3 | 7M |
| 1. Describe in detail about local optimization. | CO4 | L2 | 7M |
|  | 1. Explain in detail about induction variable elements and live variable analysis. | CO4 | L2 | 7M |
| 1. What is DAG? Construct the DAG for the following basic block   D := B+C  E :=A+B  B := B+C  A := E-D | CO4 | L3 | 7M |
|  | 1. Explain the following with suitable examples.   1.Constant Propagation  2. Strength Reduction  3. Induction Variables  4. Code Motion | CO4 | L3 | 7M |
| 1. What is peephole optimization? Mention the transformations that are characteristic of peephole optimizations. | CO4 | L2 | 7M |
|  | 1. What is meant by copy propagation? Explain in detail. | CO4 | L2 | 7M |
| b) Construct DAG for given expressions  i.(a+b) \* (a+b+c)  ii. ((a+a)+(a+a))+((a+a)+(a+a)) | CO4 | L3 | 7M |
|  | 1. Construct DAG for given expression   a=b\*c  d=b  e=d\*c  b=e  f=b+c  g=f+d | CO4 | L3 | 7M |
| 1. What are basic blocks? Write the algorithm for partitioning into Blocks. | CO4 | L2 | 7M |
|  | 1. Consider the following program code:   Prod=0;  I=1;  Do{  Prod=prod+a[i]\*b[i];  I=i+1;  }while (i<=10);  i. Partition in into blocks  ii. Construct the flow graph | CO4 | L3 | 7M |
| 1. Explain Dead-code elimination with an example. | CO4 | L2 | 7M |
|  | 1. Write a short note with example to optimize the code:   a. Dead code elimination  b. Variable elimination  c. Code motion  d. Reduction in strength | CO4 | L3 | 7M |
| b) Explain the machine dependent code optimization techniques | CO4 | L2 | 7M |
|  | 1. Explain the machine independent code optimization techniques? | CO4 | L2 | 7M |
| 1. What is common sub-expression and how to eliminate it? Explain with example. | CO4 | L2 | 7M |
|  | 1. What is register allocation and assignment problem? | CO4 | L2 | 7M |
| 1. Discuss Global optimization with an example | CO4 | L2 | 7M |

Course coordinators  HOD