|  |  |  |  |
| --- | --- | --- | --- |
| **Compiler Construction (CS4031)** |  | **Sessional-I Exam** | |
| Date: February 25th 2025 |  | **Total Time (Hrs.):** | **1** |
| **Course Instructor(s)** |  | **Total Marks:** | **50** |
| Dr. Adeel Ashraf Cheema, Ms. Faryal Saud,  Ms. Mahzaib Younas |  | **Total Questions:** | **4** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Roll no |  | Section |  | Student Signature |
| **Vetted by: Signature: .** | | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CLO#** | **1** | **1** | **1** | **2** |
| **Q#** | **1** | **2** | **3** | **4** |
| **Total**  **Marks** | **10** | **13** | **12** | **15** |
| **Obtained**  **Marks** |  |  |  |  |

**Attempt all the questions.**

**Course Learning Outcomes (CLOs):**

CLO 1: Describe the architecture of compiler, and formal notations to define a programming language.

CLO 2: Implement syntax analyzer by using various top down and bottom up algorithms.

***CLO1***

**Question 01: Fill in the blanks with the correct terms to complete the statement. Cutting and Overwriting will lead to ZERO marks. [Marks: 10]**

1. A compiler that runs on one machine and produces code for another machine is called \_\_Cross compiler\_\_.
2. Intermediate Code Generator can be represented in a tree form as \_\_\_\_\_DAG\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and \_\_\_\_\_Syntax Tree\_\_\_\_\_
3. A preprocessor processes the \_ source code before the compilation\_ and produces a code that can be more efficiently used by the compiler.
4. \_\_\_\_\_main\_\_\_\_\_\_\_\_\_ and \_\_\_\_\_\_\_\_yywrap()\_\_\_\_\_\_\_\_\_\_ are the routines compulsory in the subroutine section.
5. \_\_\_\_\_lex.yy.c\_\_\_\_ is the default ‘c’ fi le name Lex produces after compiling the Lex program to c program.
6. To simplify the grammar S → ST | ε, T → abT | ab we apply elimination of \_\_\_\_left recursion.
7. The advantage of eliminating left recursion is \_Avoids parser going into infinite loop\_\_\_.
8. For A → A α | β, equivalent grammar without left recursion is \_\_\_\_A → βA′, A′ → αA′ | ε\_\_\_\_.
9. The parser groups tokens into a hierarchical structure called the \_\_\_Parse Tree\_\_\_\_\_\_\_\_\_.
10. The error handler is used to report and recover from errors encountered in the source. It is required in the \_\_\_all Compiler\_\_\_ phase.

***CLO1***

**Question 02: Answer the following questions. [Marks: 3+4+6=13]**

1. **What is the front end and backend of a compiler? (just name them)**

|  |
| --- |
| Front End Compiler  • Lexical Analysis • Syntax Analysis • Semantic Analysis • Intermediate code generation  **Backend Compiler:**  • Code Optimization • Target Code Generation |

1. **Differentiate between token and pattern with the help of example.**

|  |
| --- |
| **Token:** In programming languages, a token is the smallest unit of a program that is meaningful to the compiler. Tokens can be keywords, identifiers, constants, operators, or punctuation symbols.  **Pattern**: A pattern is a sequence of characters or symbols that describes a set of tokens in a programming language. It defines the structure or format that tokens must follow to be recognized by the compiler. |

1. **Give the output of each phase of compiler for the following source text:**

final\_grade = assignment\_score \* weight\_assignment + exam\_score \* weight\_exam;

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Lexical Analyzer** | id1= final\_grade  id2 = assignment\_score  id3 = weight\_ assignment  id4 = exam\_score  id5 = weight\_exam  id1 = id2 \* id3 + id4 \*id5 | | **Syntax Analyzer** |  | | **Semantic Analyzer** |  | | **Intermediate code generator** | t1 = id2 \* id3  t2 = id4 \* id5  id1 = t1 + t2 | | **Machine Code** | LOAD R1, assignment\_score  MUL R1, weight\_assignment  STORE R1, t1  LOAD R2, exam\_score  MUL R2, weight\_exam  STORE R2, t2  LOAD R3, t1  ADD R3, t2  STORE R3, final\_grade | |

***CLO1***

**Question 03:** Write a lex program that scans the given input extract its lexemes along with their corresponding token types. Write the rule for each token in the form of Regular expression.

**[Note: int, float, double etc. consider as data type not keyword.] [Marks: 10 +2=12]**

|  |
| --- |
| main( )  {  int a[3], t1t2;  t1=2;  a[0]=1; a[1]=2; a[t1]=3;  t2 = -(a[2]+t1\*6)/(a[2]-t1);  if (t2>5)  print(t2);  else  {  int t3;  t3=99;  t2=-25;  print(-t1 +t2\*t3);  }  endif  } |

1. **Provide the rule section.**

|  |
| --- |
| **%%**  **[main]| [print]|[if]|[else]|[endif] printf("%s is a keyword\n",yytext);**  **int printf("%s is a Datatype\n",yytext);**  **[[a-z]+[0-9]\* ] printf("%s is a identifiers\n",yytext);**  **[a-z]+[ [ ][0-9]\*[ ] ] printf("%s is a 1D array\n",yytext);**  **[0-9]+ printf("%s is integer constant \n",yytext);**  **+|-|\*|/ printf("%s is arithmetic operators\n",yytext);**  **= printf("%s is assignment operators\n",yytext);**  **> | < printf("%s is logical operator\n",yytext);**  **( | ) | { | } | ; printf("%s is a punctuator\n",yytext);**  **%%** |

1. **Describe the method for providing input to the above code in Flex.**

**[Note: Provide User Code section]**

|  |
| --- |
| **int main()**  **{**  **freopen("test.c", "r", stdin);**  **yylex();**  **return 0;**  **}** |

***CLO2***

**Question 04: Consider the grammar [Marks: 15]**

**S → aBCd | dCBe**

**B → bB | ε**

**C → ca | ac | ε**

1. **Compute the First sets and Follow sets for each of the non-terminals in the grammar. Note: must provide the answer in the form of table.**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  | **First** | **Follow** | | **S** | {a, d} | {$} | | **B** | {b, **ε** }, | {a, c, d, e} | | **C** | {a, c, **ε** }, | {b, d, e}, | |

1. **Construct an LL(1) parsing table for the above grammar.**

|  |
| --- |
|  |

1. **Construct the stack table for the following expression dcabe$.**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | **MATCHED** | **STACK** | **INPUT** | **ACTION** | |  | **S$** | **dcabe$** | **S → dCBe** | |  | **dCBe$** | **dcabe$** | **Match d** | | **d** | **CBe$** | **cabe$** | **C → ca** | | **dc** | **caBe$** | **cabe$** | **Match c** | | **dca** | **Be$** | **be$** | **Match a** | | **dca** | **bBe$** | **be$** | **B → bB** | | **dca** | **Be$** | **e$** | **Match b** | | **dcab** | **e$** | **e$** | **B → ℇ** | | **dcabe** | **$** | **$** | **Match e** | |

**Best of Luck! 😊**