Graphic Era
HILL UNIVERSITY
Established by an Act of the State Legislature of Uttarakhand (Adhiniyam Sankhya 12 of 2011)
University under section 2(f) of UGC Act, 1956

Placement and Year Back Examination Oct.-2025

Roll no. 2261514

Name of the Course: **B. Tech., CSE**
Semester: **6th**
Name of the Paper: **Compiler Design**
Paper Code: **TCS-601**
Time: **3 Hours** **Maximum Marks: 100**

Note:
i. All Questions are compulsory
ii. Answer any two sub questions among a, b and c in each main question.
iii. Total marks in each main question are twenty
iv. Each question carries 10 marks.

Q1. **(10×2=20 Marks) CO1**

a. Differentiate between compiler, interpreter, and assembler with an example. Also discuss about the various types of tokens available in compiler design with suitable explanation.

b. Why most parsers do not allow a grammar to be left recursive, non-deterministic, or ambiguous due to fundamental limitations in parsing techniques? Justify your answer. Consider the given grammar productions:

E→E+E/E-E/E*E/id

For the given string **"id+id-id*id"**, verify whether the above grammar is ambiguous or not.

c. Provide a clear illustration with appropriate C code showing how the lexical analysis phase generates tokens and how syntax analysis uses these tokens to create a parse tree. Additionally, consider the below C program and design a lex code that counts the total number of left parentheses, right parentheses, left curly braces, and right curly braces.

```
int main()
{ int x=2,y=10,z;
z=x*y;
printf("%d",z); return 0; }
```

Q2. **(10×2=20 Marks) CO2**

a. What do you mean by operator grammar? Consider the following grammar productions:

S → S + S
S → S * S
S → id

Using the operator relation table, create a parse tree for the input string 3 + 4 + 5 * 6, where the precedence order is defined as "id > * > + > $" and both + and * are left associative.

b. What do you mean by a CFG? Using Leftmost Derivation (LMD) and Rightmost Derivation (RMD), generate the string "aaabbb" using the given grammar productions.

A→aAb/ab/ε

c. Define canonical collections of LR(0) items. Consider the given grammar productions:

E→ E+T|T
T→ T*F|F
F→ (E)|id

Verify whether the given grammar is LR(0) or not by constructing canonical collection of LR(0) items.

Q3.

(10×2=20 Marks) CO3

a. Consider the following Syntax-Directed Translation schemes.

Productions	**Semantic rules**
E→E+T	{ printf("+"); }
E→T	{ }
T→T*F	{ printf("*"); }
T→F	{ }
F→id	{ printf("number.lexvalue"); }

Using the above SDT, convert the given infix expression **3+2+3*5** into postfix expression using both top-down and bottom-up approach.

b. Provide a clear example to illustrate type checking during the compilation of a program. Also, demonstrate how the analysis phase of the compiler creates a symbol table and how the synthesis phase utilizes this symbol table. Justify your explanation with proper example.

c. What are the various applications of SDT? Also differentiate between L-attributed and S-attributed SDT using a suitable example.

Q4.

(10×2=20 Marks) CO4

a. What do you understand by back patching? Consider the given high level code fragment based on switch-case statement:

```
switch(i+j)
{
case 1:
   { A=(X+Y*Z) + (L+M*K)
   break; }
case 2:
   { Z=(X+Y+K) + (A+B*C)
   break; }
default:
   { x=(y+z+a+b)
   break; }
}
```

For the above code, generate 3-address code by using back patching technique.

b. Write short note on the following:
i. Loop jamming **ii.** Peephole optimization **iii.** Frequency reduction **iv.** Constant Propagation

c. Illustrate common sub-expression. Consider the given high-level expression:
X=(A+B*C) + (A+B+D) – (Y+Z*C)
Covert the given expression into 3-address code and also eliminate the common sub-expression if any.

Q5. **(10×2=20 Marks) CO5**

a. Consider the following 3-address instructions:
1. START
2. T1 = a + b
3. T2 = t1 * c
4. if (T2 > f) goto 6
5. e = T1 + f
6. goto 7
7. L1: e = T2 + f
8. L2: g = e * 2
9. EXIT/END
Using the above instructions, construct program flow graph using control flow analysis and verify whether there is any loop present or not? Justify your answer.

b. Is it possible to build a compiler for new machine without having an intermediate code? Briefly explain various types of intermediate codes available in compiler with example.

c. What are the supports needed from the operating system during program runtime. Also, differentiate between heap allocation and stack allocation with appropriate examples.