National University of Computer and Emerging Sciences, Lahore Campus

	Course Name:	**Compiler Construction**	**Course Code:**	
	Degree Program:	**BS (CS)**	**Semester:**	**Fall 2019**
	Exam Duration:	**2 hr 30 min**	**Total Marks:**	**50**
	Paper Date:	**19-Dec-2019**	**Weight**	
	Section:	**CS**	**Page(s):**	**5**
	Exam Type:	**Final Exam**		

Student : Name:_____ Roll No._____
Section:_____

Question 1 (5+5+10 marks)

LaTeX is a document preparation system. Consider the following LaTeX code:

```
\begin{itemize}
\item TeX is a typesetting language and not a word processor
\item TeX is a program and and not an application
\item There is no meaning in comparing TeX to a word processor
\end{itemize}
```

When the above code is given as input, the LaTeX generates the following bullet-list:

- TeX is a typesetting language and not a word processor
- TeX is a program and and not an application
- There is no meaning in comparing TeX to a word processor

If we replace the keyword "itemize" with "enumerate" then the LaTeX generates a numbered list instead. Secondly, a list may be nested in another list. See the following code for example:

```
\begin{enumerate}
\item Prepare a source file with the extension "tex"
\item Compile it with LaTeX to produce a "dvi" file
      \begin{enumerate}
      \item Use a previewer to view the output
      \item Edit the source if needed
      \item Recompile
      \end{enumerate}
\item Print the document using a "dvi" driver
\end{enumerate}
```

This code may result in the following text:

1) Prepare a source file with the extension "tex"
2) Compile it with LaTeX to produce a "dvi" file
    a. Use a previewer to view the output
    b. Edit the source if needed
    c. Recompile
3) Print the document using a "dvi" driver

Now answer the following questions:

a) Identify all the tokens in the above LaTeX code.
b) Give regular definitions for all those tokens which have more-than-one lexemes.
c) Give a CFG to recognize such LaTeX code for lists.

SOL:

T -> enumerate | itemize
S -> \begin {enumerate} L \end{enumerate} | \begin {itemize} L \end{itemize}
L -> \ item STR L | S | ^

Question 2 (10 marks)
Give a translation scheme to convert a given C++ For loop into While loop. Consider the following loop for example:

```
for (int i = 1; i <= 100; ++i) {
     sum = sum + i;
}
```

The preceding loop shall be converted into the following:

```
int i = 1;
while (i <= 100) {
        sum = sum + i;
        ++i;
}
```

Your solution shall be generic; however you can assume that the loop body is enclosed within braces. Use the following CFG:

```
A -> for (S ; S ; S) { L }
L -> L S ; | ^
```

Here S is any statement. Assume you have an attribute S.ins (instance) that provides the actual statement in the form of a string. In the above example, S_1.ins shall give "int i = 1", S_2.ins shall yield "i <= 100", and so on.

SOL:

```
A -> for (S             {print(S1.ins), “;”}
          ; S           {print(“while ( ”,S2.ins, “) { \n”)}
          ; S) { L      {print(L.ins); print(S3.ins, “}”}
          }

     L -> L
          S {L.ins = L1.ins + S.ins}
          ;

     L -> ^ {L.ins = “”}
```

Question 3 (10 marks)

t

Remove left recursion from the following translation scheme. Do not use global variables.

```
S -> S1 ( S2 )    {S.c = S1.c + S2.c}
S -> #            {S.c = 1}

SOLUTION:
S -> # {R.c = 1}
      R {S.c = R.c}
R -> (S) {R1.c = S.c + R.c}
      R  {R.c = R1.c}
      | ^ {R.c = 0}

Other solution:
S -> #R {S.c = R.c + 1}
R -> (S)R   {R.c = S.c + R1.c}
R-> ^ {R.c = 0}
```

Question 4 (10 marks)

Consider a virtual machine that executes two-address code. All variables are global and are stored in a data section. The only data type available is Integer. Following is its code skeleton:

```
int *ds = new int[..]; // data section
int quad[..][3]; // three-address code stored in triplet
int pc = 0; // program counter
...
for (int pc = 0; quad[pc][0] != HALT; ++pc) {
     switch (quad[pc][0]) {
          case '+': ...
          case '-: ...
          case SWAP: // Add code here!
          case SKIP: // Add code here!
          ...
     }
}
```

The machine supports several instructions. Your task is to give C/C++ code for the following two instructions:

```
SWAP  X, Y
SKIP  N
```

The first instruction swaps the contents of the variables X and Y. While the second instruction is a jump instruction: it skips N machine-code instructions, and control is transferred to a later instruction.

SOLUTION:

```
for (int pc = 0; quad[pc][0] != HALT; ++pc) {

          switch (quad[pc][0]) {
               case SWAP:
                 int index1 = quad[pc][1];
                 int index2 = quad[pc][2];
                 swap(ds[index1],ds[index2]);

               case SKIP:
                    pc = pc + ds[quad[pc][1]] - 1

          }

     }
```