1. Consider the following grammar:

   \[ \begin{align*}
   &<\text{expr}> \rightarrow <\text{expr}> + <\text{id}> \\
   &\quad \mid <\text{id}> * <\text{expr}> \\
   &\quad \mid (<\text{expr}> ) \\
   &\quad \mid <\text{id}> \\
   &<\text{id}> \rightarrow X | Y | Z
   \end{align*} \]

   Draw a Parse Tree for the expression

   \((X \ast Y + Z) + Z\)

2. Consider the grammar:

   \[ \begin{align*}
   &<\text{L}> \rightarrow s<\text{A}>t<\text{B}> | <\text{B}> | u \\
   &<\text{A}> \rightarrow s<\text{A}>t | <\text{L}> \\
   &<\text{B}> \rightarrow u<\text{B}> | u
   \end{align*} \]

   By left-most derivation show that the expression \(s\ u\ t\ u\ u\) can be derived from the grammar.
3. Consider the following C++-like program

```c++
int A;

void F1()
{
    A = A + 1;
    cout << A << endl;
}

void F2()
{
    int A = 10;
    F1();
    cout << A << endl;
}

void main()
{
    A = 30;
    F2();
    cout << A << endl;
}
```

If we assume static scooping, what are the values of A that are output (beginning with the cout statement in F1?)

If we assume dynamic scooping, what are the values of A that are output (beginning with the cout statement in F1?)

4. Consider the grammar

```
<assign> -> <id> = <expr>

<id> -> A|B|C

<expr> -> <id> + <expr>
        | <id> * <expr>
        | ( <expr> )
        | <id>
```

Explain why cannot A = (B+C) * A be derived?
5. Translate the following C++ code into primitive, virtual machine, C++ code: (using only goto statements, if statements, and labels. -- don’t use for, do-while, or while loops)

```cpp
do {
    A = A - B;
    while ( A > 100 );
    A = A + 10;
}
```

6. Suppose that C/C++ did ignored short cuts in Boolean evaluation. Then the code might have some problems when executed:

```cpp
cin >> a >> b;
if ( a == 0 || b/a > 0 )
    cout << b;
```

Rewrite the code to protect against the lack of short cuts.