1 Switch Statement Code Generation

Recall that C and C++ also have switch statements. Suppose we wished to add a similar statement with a slightly different syntax to WLP4:

```plaintext
switch(expr) {
    case(expr) {
        statements
    }
    case(expr) {
        statements
    }
    ...
    default {
        statements
    }
}
```

Here, case statements don’t fall through, and the default case is mandatory (meaning it will always be there, but not always run). Furthermore, each case can contain an arbitrary expression rather than a constant. Write pseudocode to generate MIPS assembly code for the following production rules, once again assuming that all of scanning, parsing and semantic analysis are already handled.

```plaintext
statement → SWITCH LPAREN expr RPAREN LBRACE cases default RBRACE
            | cases
            | cases → ε
            | case → CASE LPAREN expr RPAREN LBRACE statements RBRACE
            | default → DEFAULT LBRACE statements RBRACE
```

2 Extending WLP4: Pre- and Post-Increment

Recall from the last tutorial that we added pre- and post-increment operators to WLP4 which operated on ints and used the grammar rules below. We will extend these operators to work for pointers as well.

```plaintext
factor → PLUS PLUS lvalue
factor → lvalue PLUS PLUS
```

1. Give type rules for the above grammar rules.
2. Show how to modify the code generation for these rules in the last tutorial so that the operators can either increment int or int* variables.

3  Extending WLP4: Pointers as Conditions

Recall that in C, pointers can be used as conditions, such as in the following example:

```c
int *c = NULL;
if (c) {
    //c is not null
} else {
    //c is null
}
```

This is most commonly used to check whether pointers are null.

1. Modify the WLP4 grammar so that pointers can be used in if and while tests.
2. Describe how to modify the WLP4 type checker to account for these new rules.
3. Write pseudocode to generate code for these new rules.