Assignment 1-2

尹嘉权 **3120000419** 计科**1205**

Problem Description

Errors in a computer program can be classified according to when they are detected and, if they are detected at compile time, what part of the compiler detects them. Using your favorite imperative language, give an example of each of the following:

1. A lexical error, detected by the scanner
2. A syntax error, detected by the parser
3. A static semantic error, detected by semantic analysis
4. A dynamic semantic error, detected by code generated by the compiler
5. An error that the compiler can neither catch nor easily generate code to catch (this should be a violation of the language definition, not just a program bug)

You need describe an error in each category listed above. You do not need to write specific code.

Are programs (A) and (B) equivalent? Which one is better in performance and in what conditions? Justify your answer.

A

```
if (Condition){
  Case A;
} else {
  Case B;
}
```

B

```
Case B;
if (Condition) {
  Undo Case B;
  Case A;
}
```

Answer 0.

In Java:

1. lexical error:
   The scanner will pick up things like invalid syntactic identifier character in Java.
   i.e. private Scanner &sin;

2. syntax error:
   missing a ‘ ; ’ at the end of a statement.
   i.e. int i = 0

3. static semantic error:
   Access a method which is not declared in the class.

   Static semantic analysis will detect things like assignment type compatibility, operator type compatibility, scope rules, variables being declared before being used, and do the actual and formal parameters match.
   i.e. It’s invalid to assignment an Object Type variable to a File Type variable.

4. dynamic semantic error:
   Variables being used in an expression without being given a value.
   Pointers being dereferenced that refer to invalid objects.
   Array subscripts that lie out of the bounds of the array.
   Arithmetic operations over-flowing.
   i.e. When you access an array index it has auto-generated code to do bounds checking, and it will throw an exception if you are out of bounds.

5. An error that the compiler can neither catch nor easily generate code to catch:
   Use method name as a variable.
   i.e. In implementation of HashMap, you cannot use itself as one of its own keys or values, because it will cause the HashCode and equal methods to have undefined behaviour.

Answer 1.

The equivalency between A and B depends on whether in program B, will it change to variable of the condition.

1) If in the program B, all operations in program B will not change the result of the condition, then program A and program B are equivalent. And in this time, it’s hard to say which one is better in performance. We need more hypothesis.

   A. If the probability of condition becoming true is very very low, then we can say A and B almost performance quite better, because in this case, when accessing program A, the total running time is the sum of if-statement and case B. While accessing program B, it’s the same.

   B. If the probability of condition becoming true is not very low, then program A is better than B in performance. Because at this time,
      when accessing program A, the total running time Ta is
      $$T_{\text{if-statement}} + T_{\text{case A}} * P_{\text{condition='T'}} + T_{\text{case B}} * (1 - P_{\text{condition='T'}})$$
      when accessing program B, the total running time Tb is
      $$T_{\text{case B}} + T_{\text{if-statement}} + P_{\text{condition='T'}} * (T_{\text{undo case B}} + T_{\text{case A}})$$

   we can easily subtract it, $Ta - Tb = - P_{\text{condition='T'}} * (T_{\text{case B}} + T_{\text{undo case B}}) < 0$
   It shows that A is better than B.

2) If in the program B, some operation(s) in the program will change the result of the condition. It's obvious that A and B are not equivalent. Because when we have done the case B in program B, we may change the condition result, which shows sometimes we may need to go to the if-statement scope to undo case B and do case A, but we haven't. And on the other hand, we may go to if-statement scope to do something which we don't need before. So in this case, program A and B are not equivalent.