Aufgabe 11: Syntaxfehlerbehandlung mit speziellen Fangsymbolen

Schreiben Sie für folgende Grammatik Parsermethoden und implementieren Sie dabei eine Fehlerbehandlung mit speziellen Fangsymbolen (Abschnitt 3.4.3).

```
Program     = "program" ident {Declaration} "{" {Statement} "}".
Declaration = ("public" | "private") Type ident ";".
Type        = ident ["[" "]"].
```

Da public und private Schlüsselwörter sind, die nur am Anfang einer Deklaration vorkommen, ist der Deklarationsanfang ein guter Synchronisationspunkt. Die Parsermethode für Statement brauchen Sie nicht zu implementieren.

Lösung

Wichtig ist, dafür zu sorgen, dass Declaration auch betreten bzw. nicht verlassen wird, wenn kein public oder private vorliegt, damit die Synchronisationsstelle erreicht wird. Es wird hier angenommen, dass es auch eine Synchronisationsstelle in Statement gibt, daher wird die Statement-Folge auf die gleiche Art implementiert. Die Schleifen werden nur verlassen, wenn ein gültiger Nachfolger der Iteration vorliegt oder eof, damit man nicht in eine Endlosschleife gelangt.

```
static void Program() {
    check(program);
    check(ident);
    while (sym != lbrace && sym != eof) {
        Declaration();
    }
    check(lbrace);
    while (sym != rbrace && sym != eof) {
        Statement();
    }
    check(rbrace);
}

static void Declaration() {
    if (sym != public_ && sym != private_ && sym != lbrace) {  // synchronization point
        error("Declaration expected");
        do scan(); while (sym != public_ && sym != private_ && sym != lbrace && sym != eof);
        errDist = 0;
    }
    if (sym == lbrace || sym == eof) break;
    if (sym == public_) scan();
    else if (sym == private_) scan();
    // no error branch, because synchronization ended with public, private, lbrace or eof
    Type();
    check(ident);
    check(semicolon);
}

static void Type() {
    check(ident);
    if (sym == lbrack) {
        scan();
        check(rbrack);
    }
}
```