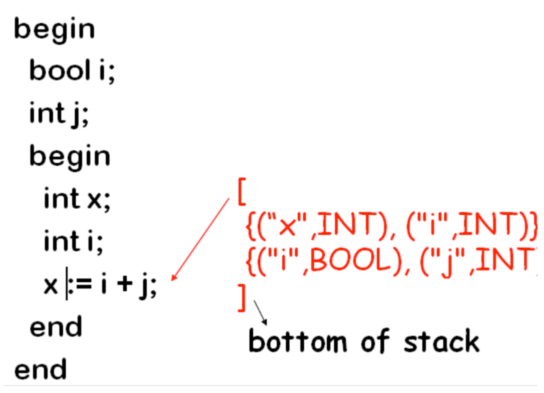
Attribute Grammar – Part 2

```
<prog>      ::= <block>
                <block>.alltbl  := emptystack
<block>     ::= begin <declist>; <stmtlist> end
                <stmtlist>.alltbl  := push(<declist>.tbl, <block>.alltbl)
<declist>_1 ::= <decl>
                <declist>_1.tbl  := <decl>.tbl
              | <decl> ; <declist>_2
              <declist>_1.tbl  := <decl>.tbl ∪ <declist>_2.tbl
              Cond: ids(<decl>.tbl)∩ids(<declist>_2.tbl)={}
<decl>     ::= int <id>
               <decl>.tbl  := { (id.lexval, INT) }
             | bool <id>
               <decl>.tbl  := { (id.lexval, BOOL) }
<stmtlist>_1 ::= <stmt>
                 <stmt>.alltbl  := <stmtlist>_1.alltbl
               | <stmt> ; <stmtlist>_2
                 <stmt>.alltbl  := <stmtlist>_1.alltbl
                <stmtlist>_2.alltbl  := <stmtlist>_1.alltbl
<stmt>        ::= <assign>
                   <assign>.alltbl  := <stmt>.alltbl
                | <block>
                  <block>.alltbl  := <stmt>.alltbl
<assign>    ::= <id> := <intexp>
                  <intexp>.alltbl  := <assign>.alltbl
                  Cond: typeof(id.lexval,<assign>.alltbl) = INT
              | <id> := <boolexp>
                <boolexp>.alltbl  := <assign>.alltbl
                Cond: typeof(id.lexval,<assign>.alltbl) = BOOL
<boolexp> ::= true | false | <id>
                Cond: typeof(id.lexval,<boolexp>.alltbl) = BOOL
<intexp>_1  ::= <number>
              | <id>
               Cond: typeof(id.lexval,<intexp>_1.alltbl) = INT
              | <intexp>_2 + <intexp>_3
                <intexp>_2.alltbl  := <intexp>_1.alltbl
                <intexp>_3.alltbl  := <intexp>_1.alltbl
```



```
begin
 bool i;
 int j;
 begin
  int x;
  int i;
  x := i + j;
 end
end
```