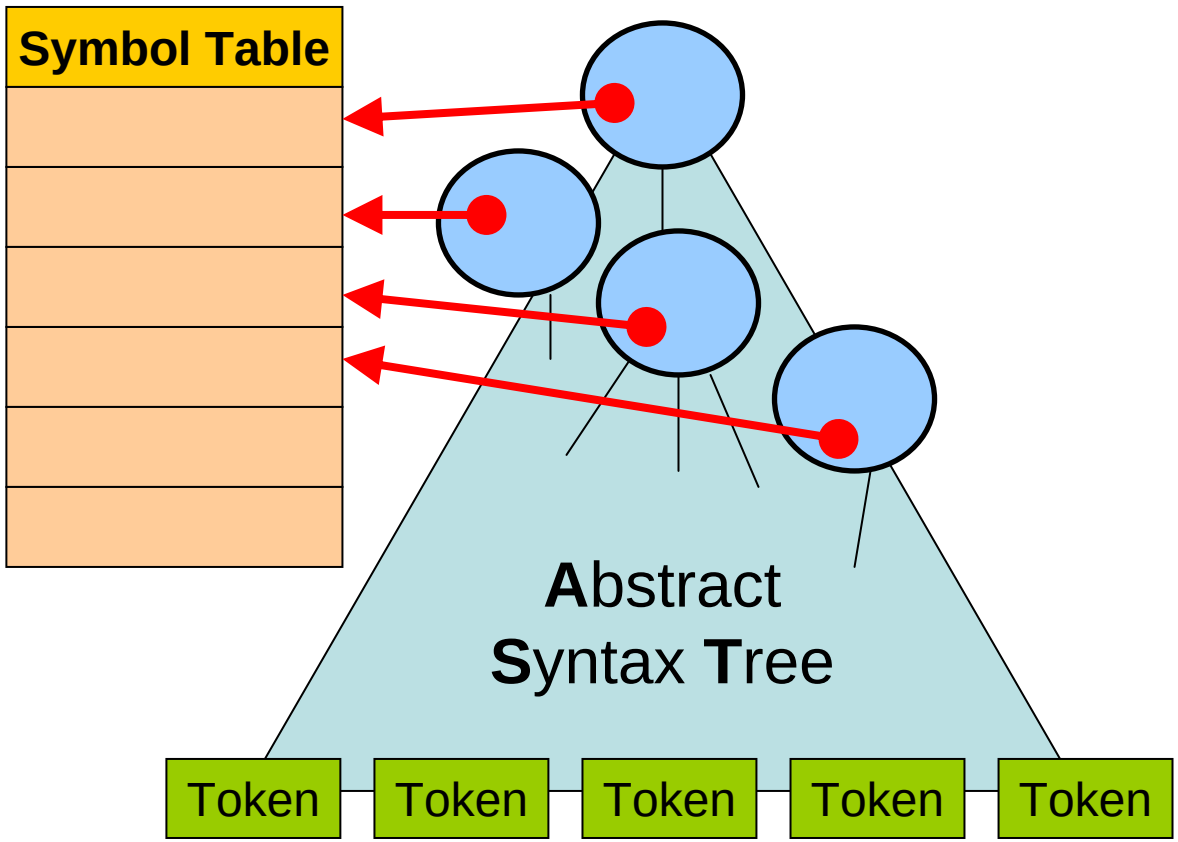
A Translation Example

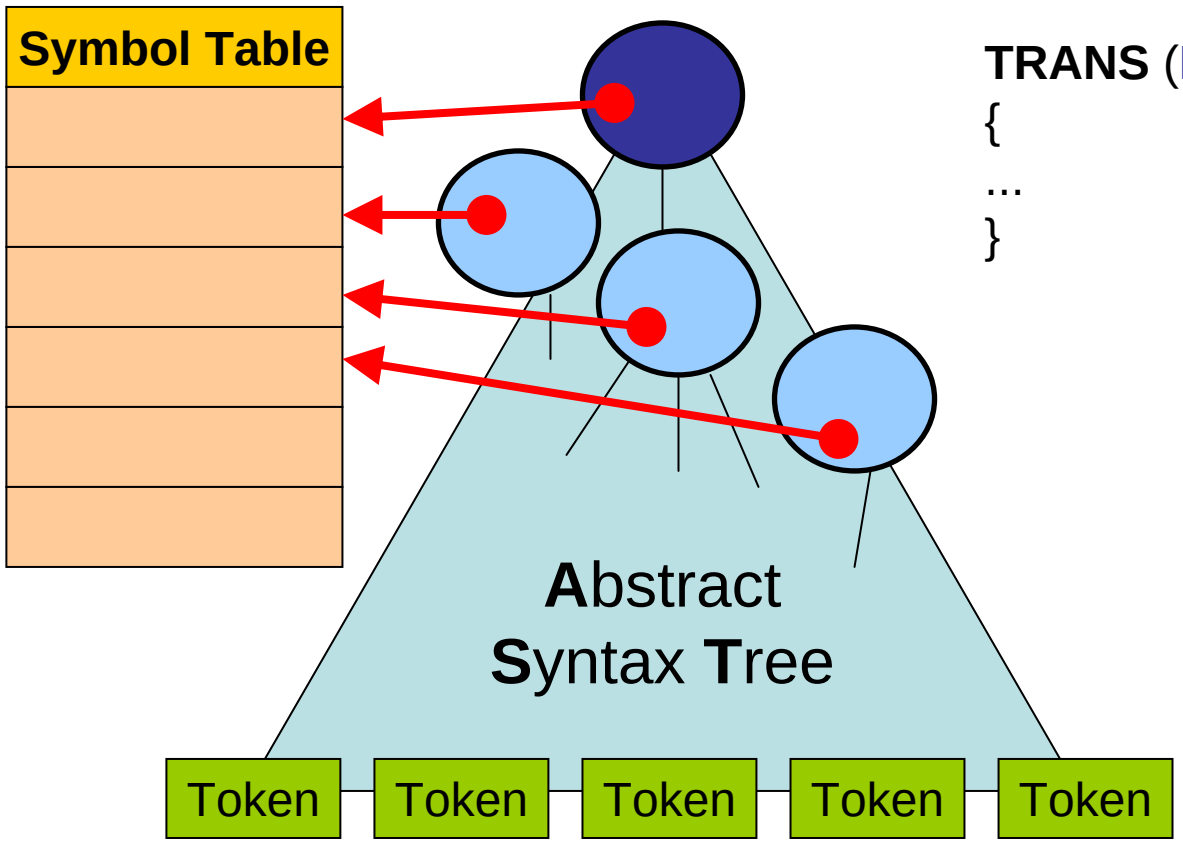
related to textbook **chapter #6**

Important Note:
You must study the Textbook Chapter #6 together with these slides!

General Principle: **AST**-based **recursive** Translation

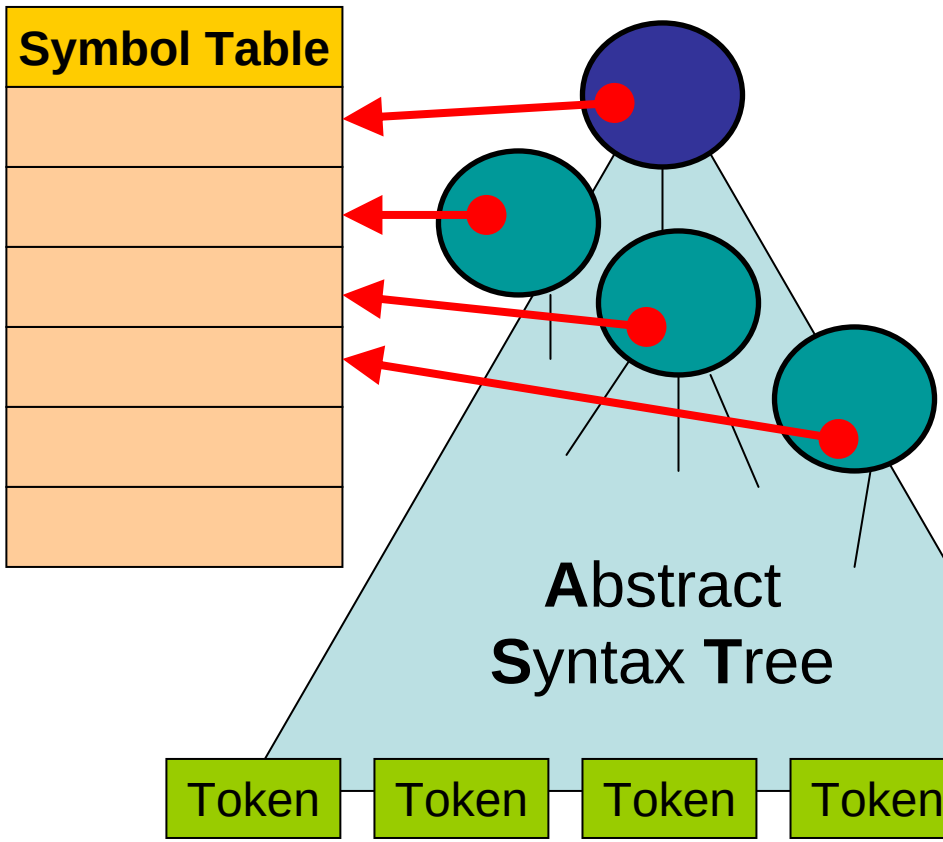


General Principle: **AST**-based **recursive** Translation



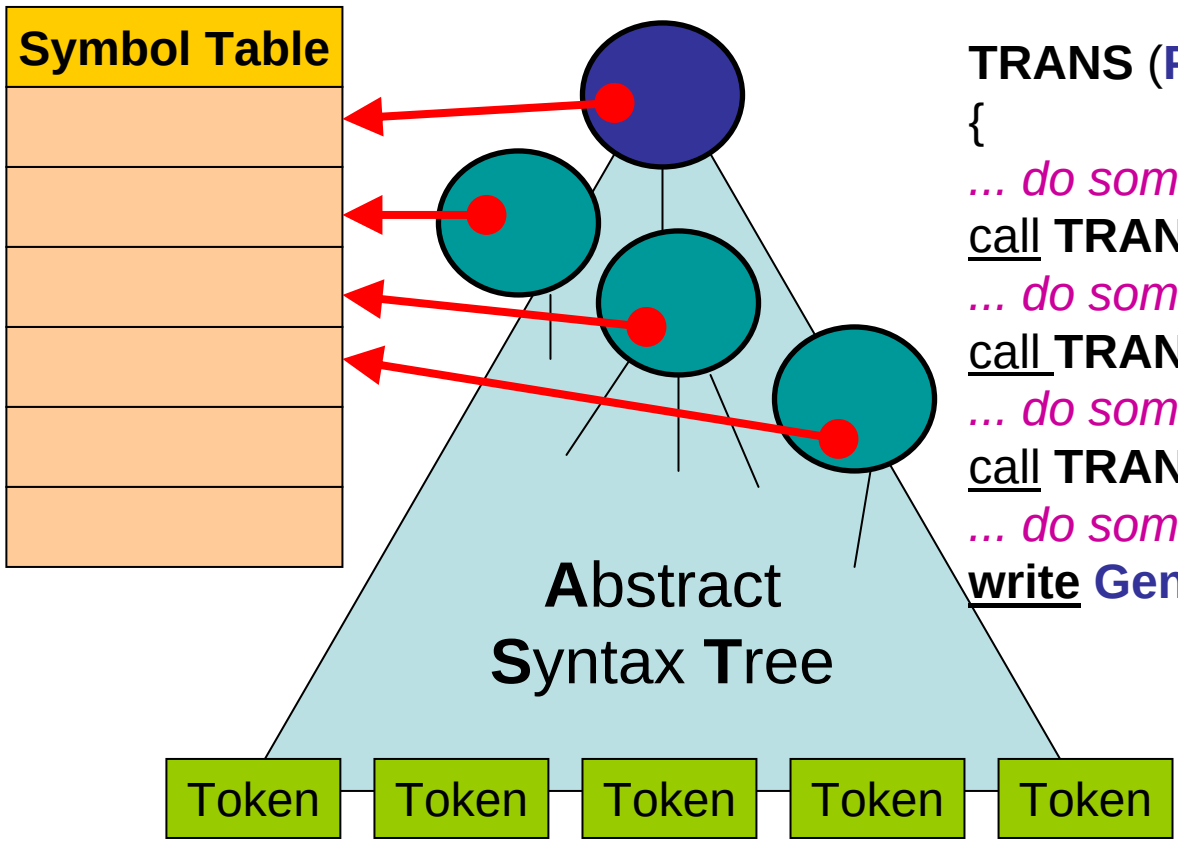
TRANS (**Parent_node**, parameters...)
{
...
}

General Principle: **AST**-based **recursive** Translation



TRANS (**Parent_node**, parameters...)
{
... do something ...
call **TRANS** (**Child_node**, parameters)
... do something ...
call **TRANS** (**Child_node**, parameters)
... do something ...
call **TRANS** (**Child_node**, parameters)
... do something ...
write **Generated_Code** }

General Principle: **AST**-based **recursive** Translation



TRANS (**Parent_node**, parameters...)
{
... do something ...
call **TRANS** (**Child_node**, parameters)
... do something ...
call **TRANS** (**Child_node**, parameters)
... do something ...
call **TRANS** (**Child_node**, parameters)
... do something ...
write **Generated_Code** }

Note that the "do something" lines are allowed to access the Symbol Table via the node-Pointers!

Example: **Source Code Snippet** (which we want to translate)

In this example, let us assume that **V** is some global variable, which can also be “seen” from inside the body of procedure **P**

```
while ( condition_on_V ) // comment: some Boolean Condition that uses V
{
 call P
}

... some more code ...

ProcDef P
{
 V := some_calculation...
}
```

Example: **Source Code Snippet** (which we want to translate)

TRANS ***// Comment: Remember: Recursion on Sub-Nodes!***

```
while ( condition_on_V )
{
 call P
}

... some more code ...

ProcDef P
{
 V := some_calculation...
}
```

Example: **Source Code Snippet** (which we want to translate)

TRANS

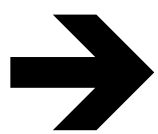
```
while ( condition_on_V )
{
 call P
}
```

TRANS

```
... some more code ...
```

TRANS

```
ProcDef P
{
 V := some_calculation...
}
```



Example: **Source Code Snippet** (which we want to translate)

TRANS

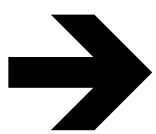
```
while ( condition_on_V )
{
 call P
}
```

TRANS

```
... some more code ...
```

TRANS

```
ProcDef P
{
 V := some_calculation...
}
```



- Label *L_entrance*
- **TRANS** (condition_on_V , *L_exit*)
- **TRANS** call P
- GOTO *L_entrance*
- Label *L_exit*

Example: **Source Code Snippet** (which we want to translate)

TRANS

```
while ( condition_on_V )
{
 call P
}
```

TRANS

```
... some more code ...
```

TRANS

```
ProcDef P
{
 V := some_calculation...
}
```

- Label *L_entrance*
- **TRANS** (condition_on_V , *L_exit*)
- **TRANS** call P
- GOTO *L_entrance*
- Label *L_exit*

Note:
To translate the Boolean Condition,
we need to pass the symbolic exit address
L_exit as additional Parameter to **TRANS**.
This call must finally generate a command
• **GOTO *L_exit***
similar to what is shown in Figure 6.8
on page 141 of our textbook.

Example: **Source Code Snippet** (which we want to translate)

- **Label *L_entrance***

  TRANS **(condition_on_V , *L_exit*)**

  TRANS **call P**

- **GOTO *L_entrance***
- **Label *L_exit***

TRANS

... some more code ...

TRANS

```
ProcDef P
{
 V := some_calculation...
}
```

Example: **Source Code Snippet** (which we want to translate)

- **Label *L_entrance***

TRANS **(condition_on_V , *L_exit*)**

TRANS **call P**

- **GOTO *L_entrance***
- **Label *L_exit***

TRANS

... some more code ...

TRANS

```
ProcDef P
{
 V := some_calculation...
}
```

Note:
The **call to P**
and the **definition of P**
are "far away" from each other!
To "link" them together via some symbolic address, **L_p**.
It does not really matter
if we first translate the call, or
if we first translate the definition.
In any case, however, we must make a newly generated label **L_p** available in the **symbol table** for later usage!

Example: **Source Code Snippet** (which we want to translate)

- **Label *L_entrance***

TRANS **(condition_on_V , *L_exit*)**

- **GOSUB *L_p***
- **GOTO *L_entrance***
- **Label *L_exit***

TRANS

... some more code ...

TRANS

```
ProcDef P
{
 V := some_calculation...
}
```

Note:
In the programming language BASIC, which we use as the target language in our practical, "**GOSUB**" is the command that causes a jump to a sub-program or procedure.
Into the **Symbol Table** we have now **written** the information that procedure-name **P** must be associated with the symbolic address label ***L_p***

Example: **Source Code Snippet** (which we want to translate)

- **Label *L_entrance***

TRANS **(condition_on_V , *L_exit*)**

- **GOSUB *L_p***
- **GOTO *L_entrance***
- **Label *L_exit***

TRANS

... some more code ...

TRANS

```
ProcDef P
{
 V := some_calculation...
}
```

- **Label *L_p***

TRANS **{ V := some_calculation... }**

- **Return**

Note:
From the **Symbol Table** we now **retrieve** the information that procedure-name **P** is associated with the symbolic address label ***L_p***

Example: **Source Code Snippet** (which we want to translate)

- **Label *L_entrance***

TRANS **(condition_on_V , *L_exit*)**

- **GOSUB *L_p***
- **GOTO *L_entrance***
- **Label *L_exit***

TRANS

... some more code ...

- **Label *L_p***

TRANS **{ V := some_calculation... }**

- **Return**

Note:
"**Return**" is a BASIC command which automatically **causes a jump back** to the program point immediately behind the GOSUB command from which the sub-procedure had been called.

Example: **Source Code Snippet** (which we want to translate)

- **Label *L_entrance***

TRANS **(condition_on_V , *L_exit*)**

- **GOSUB *L_p***
- **GOTO *L_entrance***
- **Label *L_exit***

TRANS **... some more code ...**

- **Label *L_p***

TRANS **{ V := some_calculation... }**

- **Return**

Example: **Source Code Snippet** (which we want to translate)

- **Label *L_entrance***

TRANS **(condition_on_V , *L_exit*)**

- **GOSUB *L_p***
- **GOTO *L_entrance***
- **Label *L_exit***

TRANS **... some more code ...**

- **Label *L_p***

TRANS **{ V := some_calculation... }**

- **Return**

And so the translation continues, with further recursive invocations of the TRANS function, until only

- **BASIC Commands**

remain standing there!

Example: **Source Code Snippet** (which we want to translate)

- **Label *L_entrance***
- ***more BASIC code...***
- ***more BASIC code...***
- **GOSUB *L_p***
- **GOTO *L_entrance***
- **Label *L_exit***
- ***more BASIC code...***
- ***more BASIC code...***
- **Label *L_p***
- ***more BASIC code...***
- ***more BASIC code...***
- ***more BASIC code***
- **Return**

Example: **Source Code Snippet** (which we want to translate)

- **Label *L_entrance***
- ***more BASIC code...***
- ***more BASIC code...***
- **GOSUB *L_p***
- **GOTO *L_entrance***
- **Label *L_exit***
- ***more BASIC code...***
- ***more BASIC code...***
- **Label *L_p***
- ***more BASIC code...***
- ***more BASIC code...***
- ***more BASIC code***
- **Return**

Note, however:
"**Label**" is **NOT** a proper BASIC command, and also the **Line Numbers** are still missing!

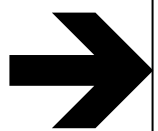
Thus we still need to do some post-processing after code-generation, in order to make everything nice!

Example: **Source Code Snippet** (which we want to translate)

- **Label *L_entrance***
- ***more BASIC code...***
- ***more BASIC code...***
- **GOSUB *L_p***
- **GOTO *L_entrance***
- **Label *L_exit***
- ***more BASIC code...***
- ***more BASIC code...***
- **Label *L_p***
- ***more BASIC code...***
- ***more BASIC code...***
- ***more BASIC code***
- **Return**

Note, however:
"**Label**" is **NOT** a proper BASIC command, and also the **Line Numbers** are still missing!

Thus we still need to do some post-processing after code-generation, in order to make everything nice!



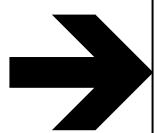
110 ***more BASIC code...***
120 ***more BASIC code...***
130 GOSUB 170
140 GOTO 110
150 ***more BASIC code...***
160 ***more BASIC code...***
170 ***more BASIC code...***
180 ***more BASIC code...***
190 ***more BASIC code***
200 Return

Example: **Source Code Snippet** (which we want to translate)

- **Label *L_entrance***
- ***more BASIC code...***
- ***more BASIC code...***
- **GOSUB *L_p***
- **GOTO *L_entrance***
- **Label *L_exit***
- ***more BASIC code...***
- ***more BASIC code...***
- **Label *L_p***
- ***more BASIC code...***
- ***more BASIC code...***
- ***more BASIC code***
- **Return**

Note, however:
"**Label**" is **NOT** a proper BASIC command, and also the **Line Numbers** are still missing!

Thus we still need to do some post-processing after code-generation, in order to make everything nice!



110 ***more BASIC code...***
120 ***more BASIC code...***
130 GOSUB 170
140 GOTO 110
150 ***more BASIC code...***
160 ***more BASIC code...***
170 ***more BASIC code...***
180 ***more BASIC code...***
190 ***more BASIC code***
200 Return

Attention!
The numbering must be done consistently, such that the **jump-targets** remain correct**!**

Conclusion

This slide concludes our translation example.

You should now be well prepared for your own implementation of the BASIC code generator in your practical project.

```
110 more BASIC code...
120 more BASIC code...
130 GOSUB 170
140 GOTO 110
150 more BASIC code...
160 more BASIC code...
170 more BASIC code...
180 more BASIC code...
190 more BASIC code
200 Return
```