Script **generated by TTT**

Title:	Petter: Compiler Construction (11.06.2020) - 31: ANSI C Example
Date:	Wed Jun 10 16:20:44 CEST 2020
Duration:	18:14 min
Pages:	9

A Practial Example: Type Definitions in ANSI C

A type definition is a *synonym* for a type expression.
In C they are introduced using the **`typedef`** keyword.
Type definitions are useful

- as abbreviation:

```
typedef struct { int x; int y; } point_t;
```

- to construct *recursive* types:

Possible declaration in C:

```
struct list {
  int info;
  struct list* next;
}
struct list* head;
```

more readable:

```
typedef struct list list_t;
struct list {
  int info;
  list_t* next;
}
list_t* head;
```

A Practial Example: Type Definitions in ANSI C

The C grammar distinguishes `typename` and `identifier`.
Consider the following declarations:

```
typedef struct { int x,y } point_t;
point_t origin;
```

A Practial Example: Type Definitions in ANSI C

The C grammar distinguishes `typename` and `identifier`.
Consider the following declarations:

```
typedef struct { int x,y } point_t;
point_t origin;
```

Idea: in a *parser action* maintain a shared list between parser and scanner to communicate identifiers to report as typenames

A Practial Example: Type Definitions in ANSI C

The C grammar distinguishes `typename` and `identifier`.
Consider the following declarations:

```
typedef struct { int x,y } point_t;
point_t origin;
```

Idea: in a *parser action* maintain a shared list between parser and scanner to communicate identifiers to report as typenames
Relevant C grammar:

$$\begin{array}{lcl} \mathit{declaration} & \rightarrow & \boxed{(\mathit{declarationspecifier})^{+}}\mathit{declarator}\ ; \\ \mathit{declarationspecifier} & \rightarrow & \texttt{static} \mid \texttt{volatile} \cdots \texttt{typedef} \\ & & \mid \texttt{void} \mid \texttt{char} \mid \texttt{char} \cdots \texttt{typename} \\ \mathit{declarator} & \rightarrow & \texttt{identifier} \mid \cdots \end{array}$$

Problem:
During reduction of the declaration, the scanner eagerly provides a new lookahead token, thus has already interpreted `point_t` in line 2 as `identifier`

A Practial Example: Type Definitions in ANSI C: Solutions

Relevant C grammar:

$$\begin{array}{lcl} \mathit{declaration} & \rightarrow & (\mathit{declarationspecifier})^{+}\mathit{declarator}\ ; \\ \mathit{declarationspecifier} & \rightarrow & \texttt{static} \mid \texttt{volatile} \cdots \texttt{typedef} \\ & & \mid \texttt{void} \mid \texttt{char} \mid \texttt{char} \cdots \texttt{typename} \\ \mathit{declarator} & \rightarrow & \texttt{identifier} \mid \cdots \end{array}$$

Solution is difficult:

A Practial Example: Type Definitions in ANSI C: Solutions

Relevant C grammar:

$$\begin{array}{lcl} \mathit{declaration} & \rightarrow & (\mathit{declarationspecifier})^{+}\mathit{declarator}\ ; \\ \mathit{declarationspecifier} & \rightarrow & \texttt{static} \mid \texttt{volatile} \cdots \texttt{typedef} \\ & & \mid \texttt{void} \mid \texttt{char} \mid \texttt{char} \cdots \texttt{typename} \\ \mathit{declarator} & \rightarrow & \texttt{identifier} \mid \cdots \end{array}$$

Solution is difficult:

1. try to fix the lookahead token class within the scanner-parser-channel ⚠ *a mess*
2. add a rule to the grammar, to make it context-free:

$$\mathit{typename} \rightarrow \texttt{identifier}$$

A Practial Example: Type Definitions in ANSI C: Solutions

Relevant C grammar:

$$\begin{array}{lcl} \mathit{declaration} & \rightarrow & (\mathit{declarationspecifier})^{+}\mathit{declarator}\ ; \\ \mathit{declarationspecifier} & \rightarrow & \texttt{static} \mid \texttt{volatile} \cdots \texttt{typedef} \\ & & \mid \texttt{void} \mid \texttt{char} \mid \texttt{char} \cdots \texttt{typename} \\ \mathit{declarator} & \rightarrow & \texttt{identifier} \mid \cdots \end{array}$$

Solution is difficult:

1. try to fix the lookahead token class within the scanner-parser-channel ⚠ *a mess*
2. add a rule to the grammar, to make it context-free:

$$\mathit{typename} \rightarrow \texttt{identifier}$$

Example input: `(mytype1)(mytype2);`

A Practial Example: Type Definitions in ANSI C: Solutions

Relevant C grammar:

$$\begin{array}{lcl} \textit{declaration} & \rightarrow & (\textit{declarationspecifier})^{+}\textit{declarator}\ ; \\ \textit{declarationspecifier} & \rightarrow & \texttt{static} \mid \texttt{volatile} \cdots \texttt{typedef} \\ & & \mid \texttt{void} \mid \texttt{char} \mid \texttt{char} \cdots \texttt{typename} \\ \textit{declarator} & \rightarrow & \texttt{identifier} \mid \cdots \end{array}$$

Solution is difficult:

1. try to fix the lookahead token class within the scanner-parser-channel ⚠ *a mess*
2. add a rule to the grammar, to make it context-free:

$$\boxed{\textit{typename} \rightarrow \texttt{identifier}}$$ ambiguous

Example input: `(mytype1)(mytype2);`

$$\begin{array}{lcl} \textit{castexpr} & \rightarrow & (\ \texttt{typename}\ )\ \textit{castexpr} \\ \textit{postfixexpr} & \rightarrow & \textit{postfixexpr}\ (\ \textit{expression}\ ) \end{array}$$

A Practial Example: Type Definitions in ANSI C: Solutions

Relevant C grammar:

$$\begin{array}{lcl} \textit{declaration} & \rightarrow & (\textit{declarationspecifier})^{+}\textit{declarator}\ \boxed{;} \\ \textit{declarationspecifier} & \rightarrow & \texttt{static} \mid \texttt{volatile} \cdots \texttt{typedef} \\ & & \mid \texttt{void} \mid \texttt{char} \mid \texttt{char} \cdots \texttt{typename} \\ \textit{declarator} & \rightarrow & \texttt{identifier} \mid \cdots \end{array}$$

Solution is difficult:

1. try to fix the lookahead token class within the scanner-parser-channel ⚠ *a mess*
2. add a rule to the grammar, to make it context-free:

$$\textit{typename} \rightarrow \texttt{identifier}$$ ambiguous

$N \rightarrow \varepsilon$ {:act();}

Example input: `(mytype1)(mytype2);`

$$\begin{array}{lcl} \textit{castexpr} & \rightarrow & (\ \texttt{typename}\ )\ \textit{castexpr} \\ \textit{postfixexpr} & \rightarrow & \textit{postfixexpr}\ (\ \textit{expression}\ ) \end{array}$$

3. register identifier as typename before lookahead is harmful

N

$$\textit{declaration} \rightarrow (\textit{declarationspecifier})^{+}\textit{declarator}\ \boxed{\{\texttt{: act(); :}\}}\ \boxed{;}$$