Faculty of Information Technology

Spring 2025

Concepts of Programming Languages

CS 211

Lecture (2)

Outline

- Introduction
- The General Problem of Describing Syntax
- Formal Methods of Describing Syntax
- Derivation

Outline

- Introduction
- The General Problem of Describing Syntax
- Formal Methods of Describing Syntax
- Derivation

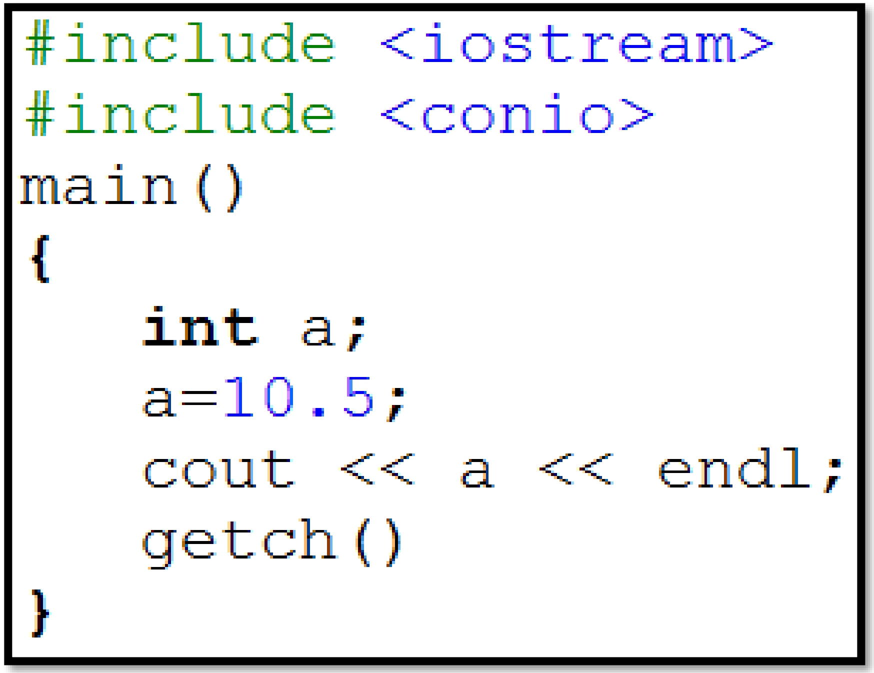
Introduction

- The task of providing a concise yet understandable description of a programming language is difficult but essential to the language’s success.
- One of the problems in describing a language is the diversity of the people who must understand the description. Among these are initial evaluators, implementors, and users.
- Most new programming languages are subjected to a period of scrutiny by potential users, often people within the organization that employs the language’s designer, before their designs are completed.
- These are the initial evaluators. The success of this feedback cycle depends heavily on the clarity of the description.

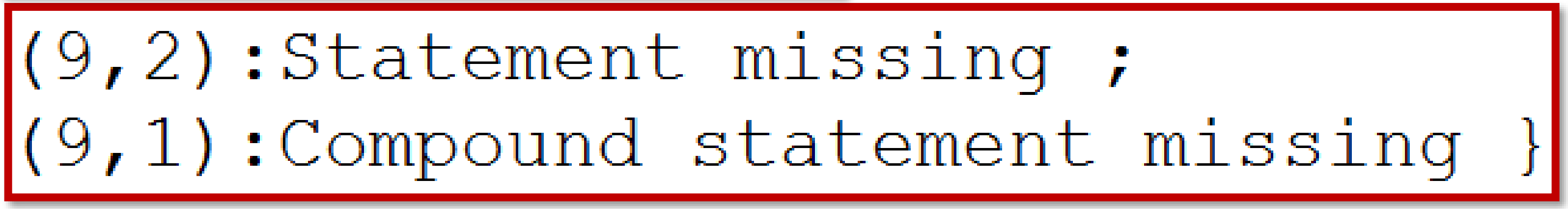
Introduction (Cont.)

- The study of programming languages, like the study of natural languages, can be divided into examinations of syntax and semantics.
  - Syntax: the form or structure of the expressions, statements, and program units.
  - Semantics: the meaning of the expressions, statements, and program units.

Introduction (Cont.)



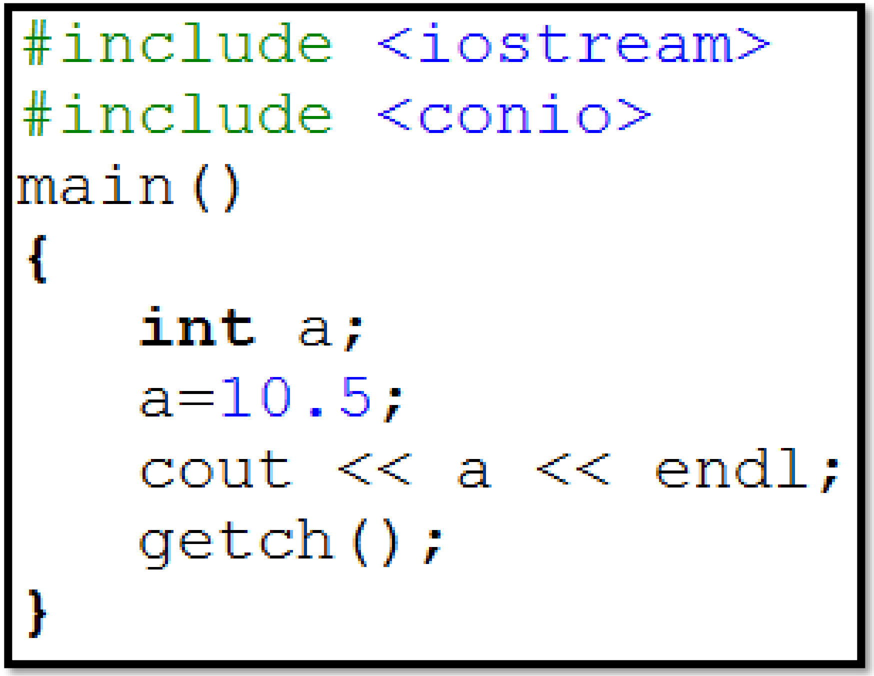
```
#include <iostream>
#include <conio>
main()
{
    int a;
    a=10.5;
    cout << a << endl;
    getch()
}
```



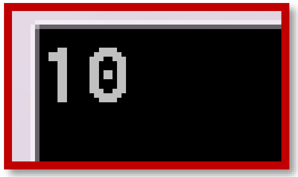
```
(9,2):Statement missing ;
(9,1):Compound statement missing }
```

- Syntax Error

Introduction (Cont.)



```
#include <iostream>
#include <conio>
main()
{
    int a;
    a=10.5;
    cout << a << endl;
    getch();
}
```



```
10
```

- Semantic error

Outline

- Introduction
- The General Problem of Describing Syntax
- Formal Methods of Describing Syntax
- Derivation

The General Problem of Describing Syntax: Terminology

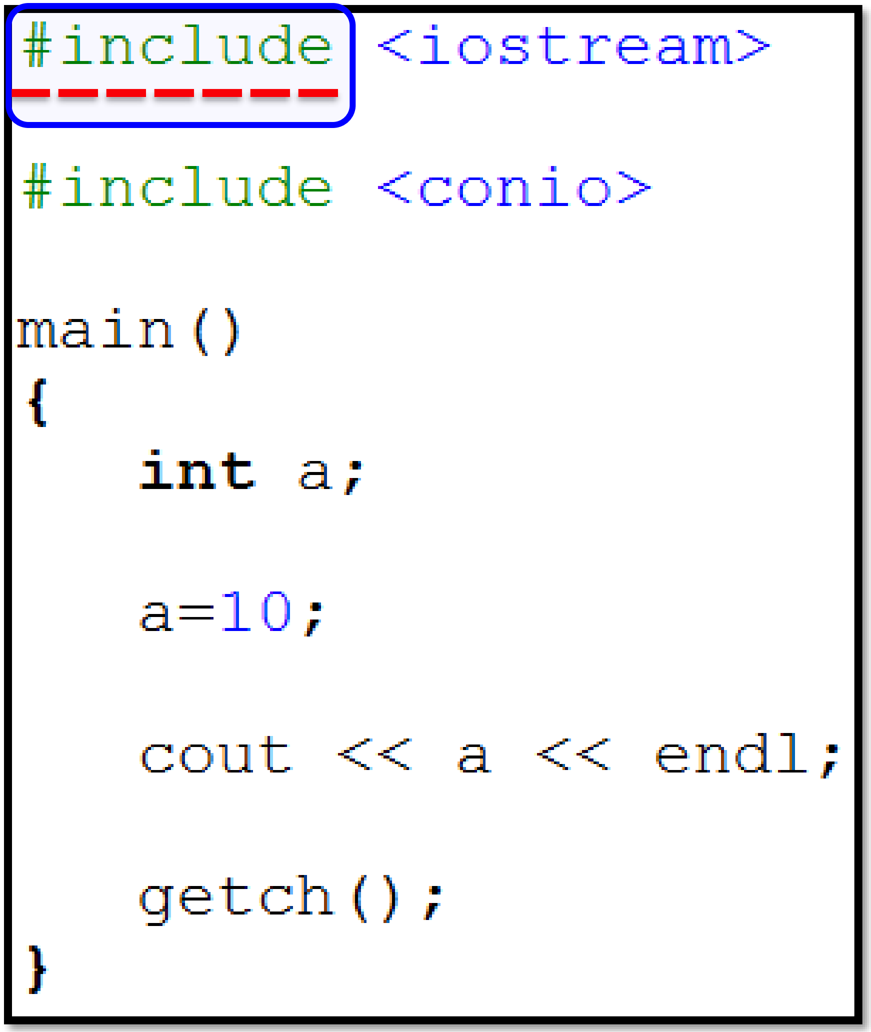
Character(s)

Lexeme

Token

pound sign

preprocessor directive



```
#include <iostream>

#include <conio>

main()
{
    int a;

    a=10;

    cout << a << endl;

    getch();
}
```

The General Problem of Describing Syntax: Terminology (Cont.)

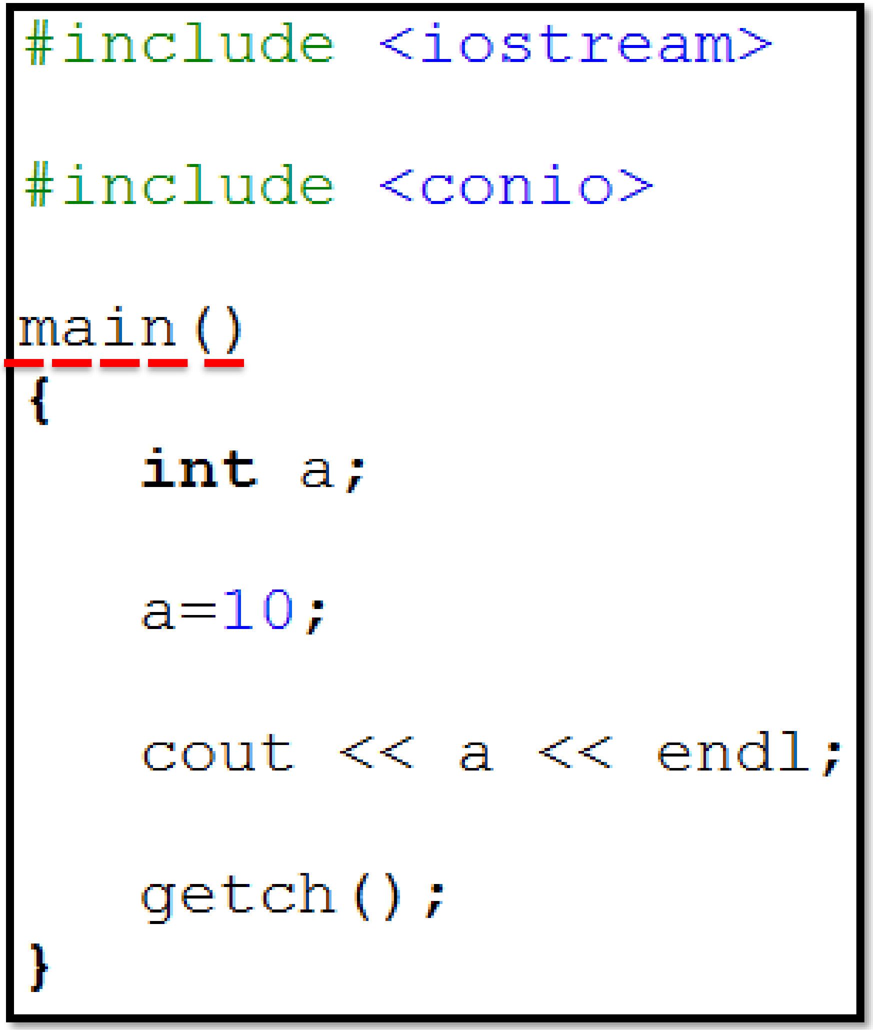
Character(s)

Lexeme

Token

reserved word

LPAREN and RPAREN



```
#include <iostream>

#include <conio>

main()
{
    int a;

    a=10;

    cout << a << endl;

    getch();
}
```

The General Problem of Describing Syntax: Terminology (Cont.)

Character(s)

Lexeme

Token

LBRACE

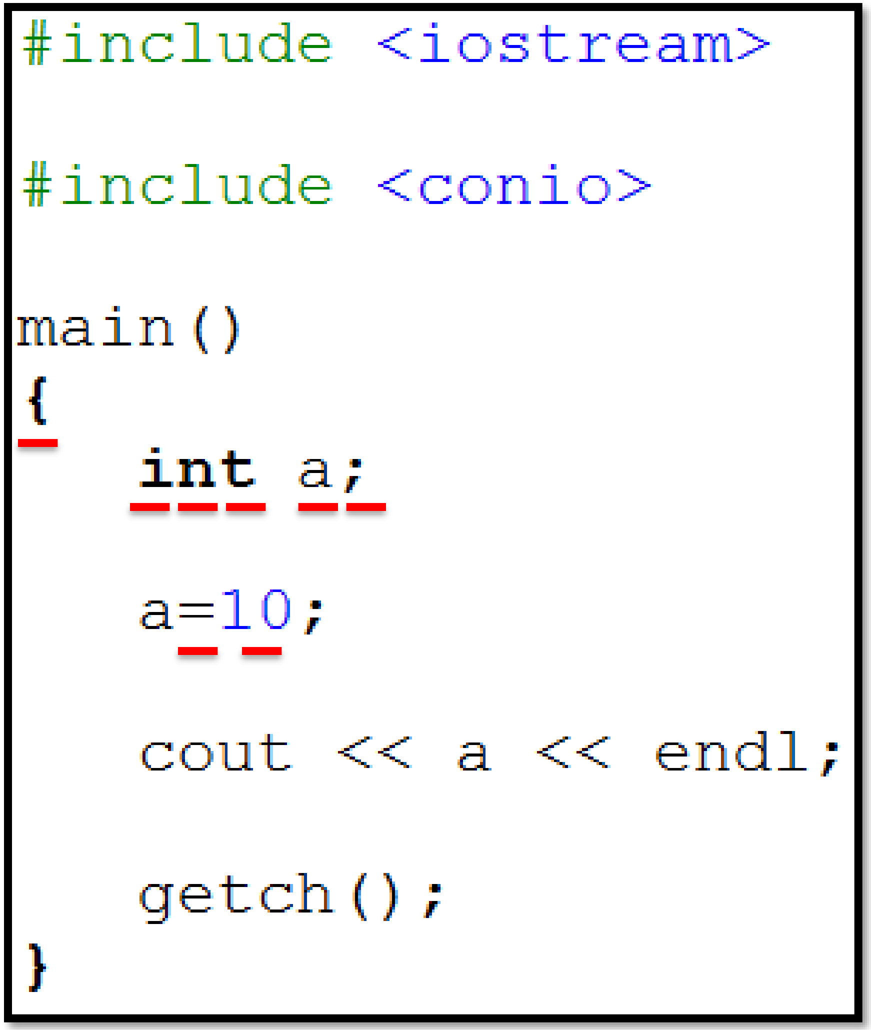
reserved word

identifier

semicolon

assign_op

int_literal



```
#include <iostream>

#include <conio>

main()
{
    int a;

    a=10;

    cout << a << endl;

    getch();
}
```

The General Problem of Describing Syntax: Terminology (Cont.)

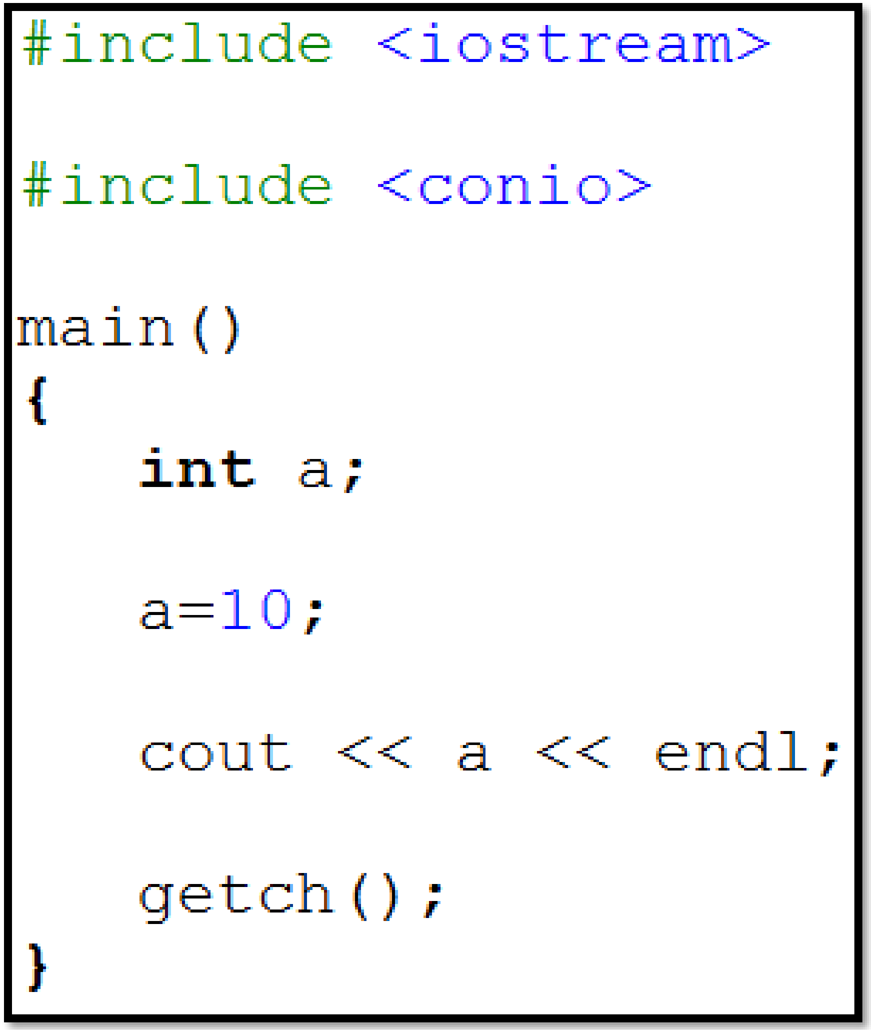
Character(s)

Lexeme

Token

Sentence

language



```
#include <iostream>

#include <conio>

main()
{
     int a;

     a=10;

     cout << a << endl;

     getch();
}
```

The General Problem of Describing Syntax: Terminology (Cont.)

- A lexeme is the lowest level syntactic unit of a language (e.g., *, sum, begin).
- A token is a category of lexemes (e.g., identifier).
- A sentence is a string of characters over some alphabet. The sentences of a language are called strings or statements.
- A language is a set of sentences of characters from some alphabet.

Formal Definition of Languages

- Recognizers
  - A recognition device reads input strings over the alphabet of the language and decides whether the input strings belong to the language.
  - Example: syntax analysis part of a compiler.
    - Detailed discussion of syntax analysis appears in Chapter 4.
- Generators
  - A device that generates sentences of a language.
  - One can determine if the syntax of a particular sentence is syntactically correct by comparing it to the structure of the generator.

Outline

- Introduction
- The General Problem of Describing Syntax
- Formal Methods of Describing Syntax
- Derivation

Formal Definition of Languages

- Context-Free Grammars
  - Developed by Noam Chomsky in the mid-1950s
  - Language generators, meant to describe the syntax of natural languages.
  - Define a class of languages called context-free languages.
- Backus-Naur Form (1959)
  - Invented by John Backus to describe the syntax of Algol 58.
  - BNF is equivalent to context-free grammars.

BNF Fundamentals

- In BNF, abstractions are used to represent classes of syntactic structures--they act like syntactic variables (also called non-terminal symbols, or just terminals).

- Terminals are lexemes or tokens.

- A rule has a left-hand side (LHS), which is a non-terminal, and a right-hand side (RHS), which is a string of terminals and/or non-terminals.

BNF Fundamentals (Cont.)

- Non-terminals are often enclosed in angle brackets.
    - Examples of BNF rules:-

      <ident_list> → identifier | identifier, <ident_list>

      <if_stmt> → if <logic_expr> then <stmt>

- Grammar: a finite non-empty set of rules.

- A start symbol is a special element of the non-terminals of a grammar.

BNF Rules

- An abstraction (or nonterminal symbol) can have more than one RHS.

```
<stmt> → <single_stmt>
       | begin <stmt_list> end
```

Describing Lists

- Syntactic lists are described using recursion.

<ident_list> → ident

| ident, <ident_list>

Outline

- Introduction
- The General Problem of Describing Syntax
- Formal Methods of Describing Syntax
- Derivation

Derivation

- A derivation is a repeated application of rules, starting with the start symbol and ending with a sentence (all terminal symbols).

- Every string of symbols in a derivation is a sentential form.

- A sentence is a sentential form that has *only* terminal symbols.

Derivation (Cont.)

- A derivation may be neither leftmost nor rightmost.

- A leftmost derivation is one in which the leftmost non-terminal in each sentential form is the one that is expanded.

- A rightmost derivation is one in which the rightmost non-terminal in each sentential form is the one that is expanded.

Example (1)

<program> → <stmts>

<stmts> → <stmt> | <stmt> ; <stmts>

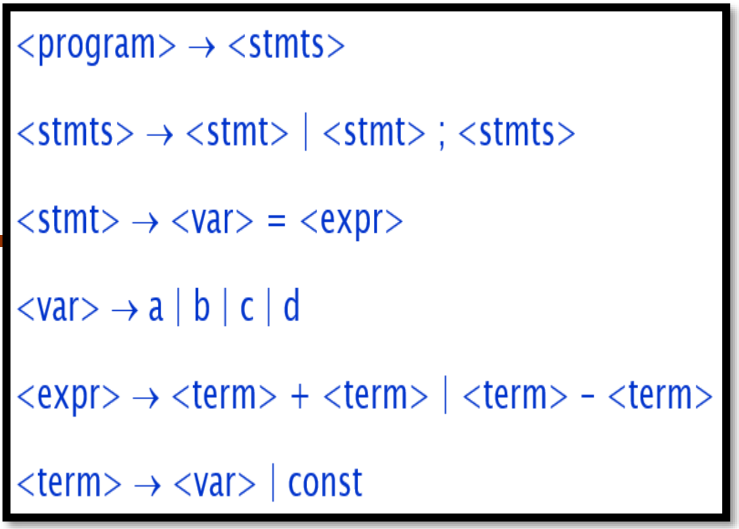
<stmt> → <var> = <expr>

<var> → a | b | c | d

<expr> → <term> + <term> | <term> – <term>

<term> → <var> | const

Example (1) (Cont.)



<program> → <stmts>

<stmts> → <stmt> | <stmt> ; <stmts>

<stmt> → <var> = <expr>

<var> → a | b | c | d

<expr> → <term> + <term> | <term> - <term>

<term> → <var> | const

<program> => <stmts>

=> <stmt>

=> <var> = <expr>

=> a = <expr>

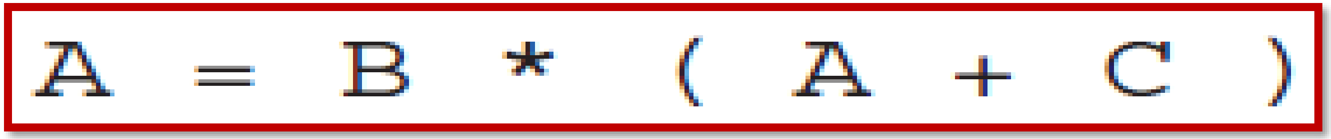
=> a = <term> + <term>

=> a = <var> + <term>

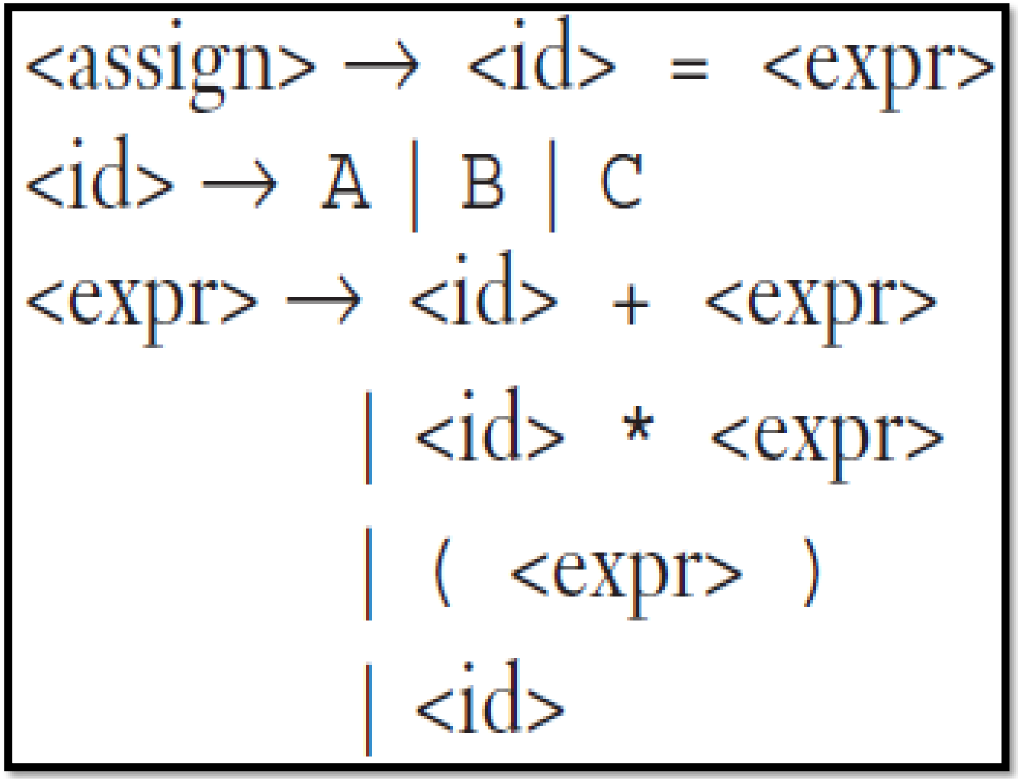
=> a = b + <term>

=> a = b + const

Example (2)



A = B * (A + C)



```
<assign> → <id> = <expr>
<id> → A | B | C
<expr> → <id> + <expr>
       | <id> * <expr>
       | ( <expr> )
       | <id>
```

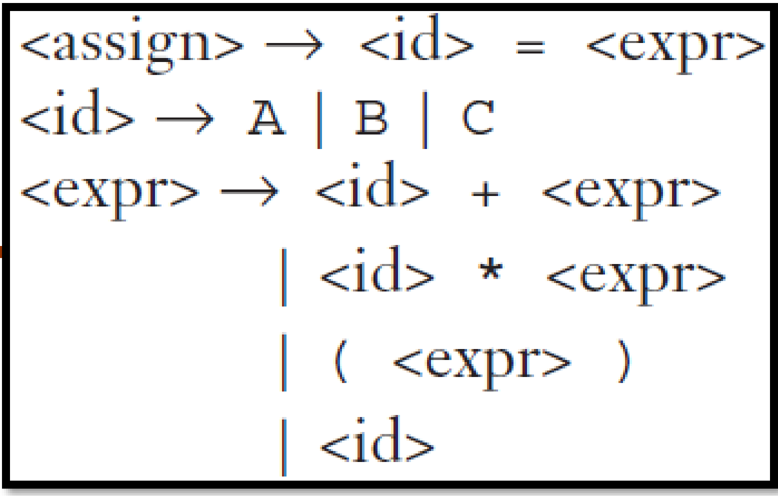
Example (2) (Cont.)

```
<assign> → <id> = <expr>
<id> → A | B | C
<expr> → <id> + <expr>
       | <id> * <expr>
       | ( <expr> )
       | <id>
```

A = B * (A + C)

<assign> => <id> = <expr>

Example (2) (Cont.)

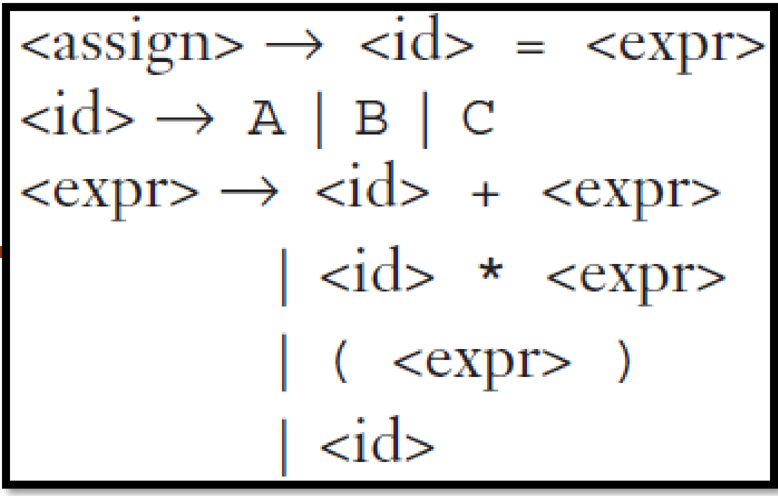


```
<assign> → <id> = <expr>
<id> → A | B | C
<expr> → <id> + <expr>
       | <id> * <expr>
       | ( <expr> )
       | <id>
```

A = B * (A + C)

```
<assign> => <id> = <expr>
         => A = <expr>
```

Example (2) (Cont.)



```
<assign> → <id> = <expr>
<id> → A | B | C
<expr> → <id> + <expr>
       | <id> * <expr>
       | ( <expr> )
       | <id>
```

```
A = B * ( A + C )
```

```
<assign> => <id> = <expr>
         => A = <expr>
         => A = <id> * <expr>
```

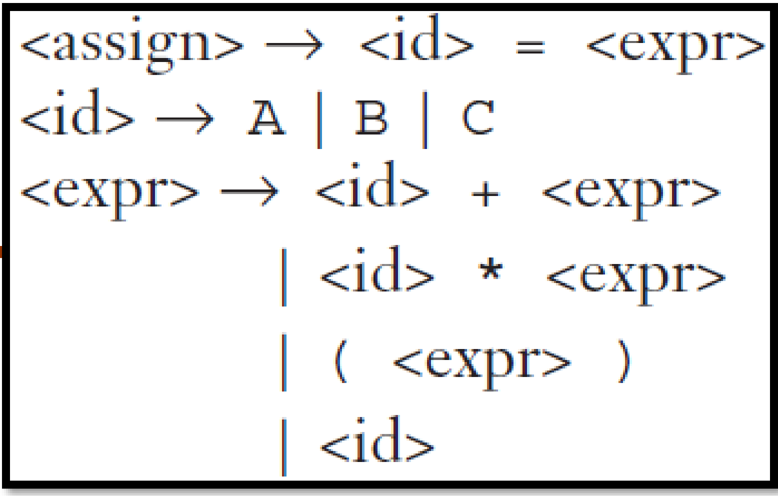
Example (2) (Cont.)

```
<assign> → <id> = <expr>
<id> → A | B | C
<expr> → <id> + <expr>
       | <id> * <expr>
       | ( <expr> )
       | <id>
```

```
A = B * ( A + C )
```

```
<assign> => <id> = <expr>
         => A = <expr>
         => A = <id> * <expr>
         => A = B * <expr>
```

Example (2) (Cont.)



```
<assign> → <id> = <expr>
<id> → A | B | C
<expr> → <id> + <expr>
       | <id> * <expr>
       | ( <expr> )
       | <id>
```

A = B * (A + C)

```
<assign> => <id> = <expr>
         => A = <expr>
         => A = <id> * <expr>
         => A = B * <expr>
         => A = B * ( <expr> )
```

Example (2) (Cont.)

```
<assign> → <id> = <expr>
<id> → A | B | C
<expr> → <id> + <expr>
       | <id> * <expr>
       | ( <expr> )
       | <id>
```

`A = B * ( A + C )`

```
<assign> => <id> = <expr>
         => A = <expr>
         => A = <id> * <expr>
         => A = B * <expr>
         => A = B * ( <expr> )
         => A = B * ( <id> + <expr> )
```

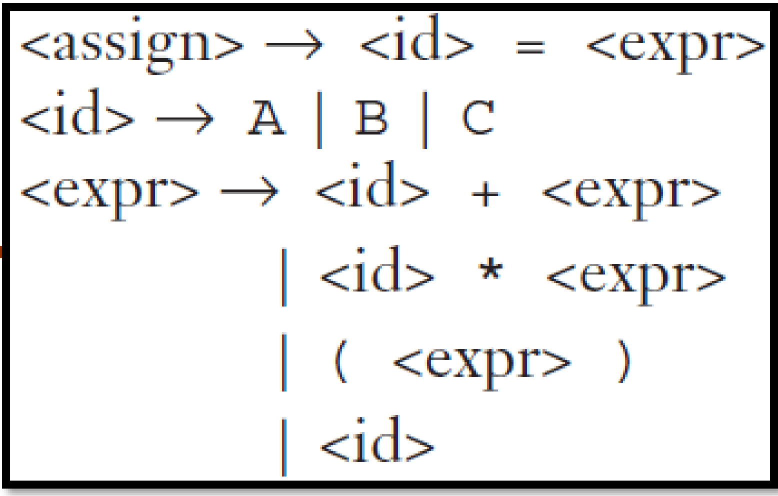
Example (2) (Cont.)

```
<assign> → <id> = <expr>
<id> → A | B | C
<expr> → <id> + <expr>
       | <id> * <expr>
       | ( <expr> )
       | <id>
```

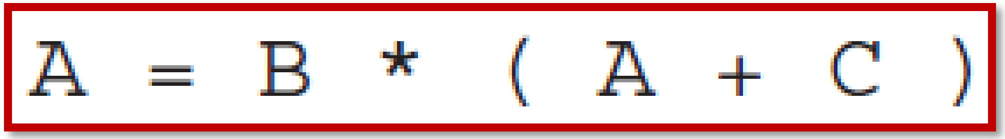
A = B * (A + C)

```
<assign> => <id> = <expr>
         => A = <expr>
         => A = <id> * <expr>
         => A = B * <expr>
         => A = B * ( <expr> )
         => A = B * ( <id> + <expr> )
         => A = B * ( A + <expr> )
```

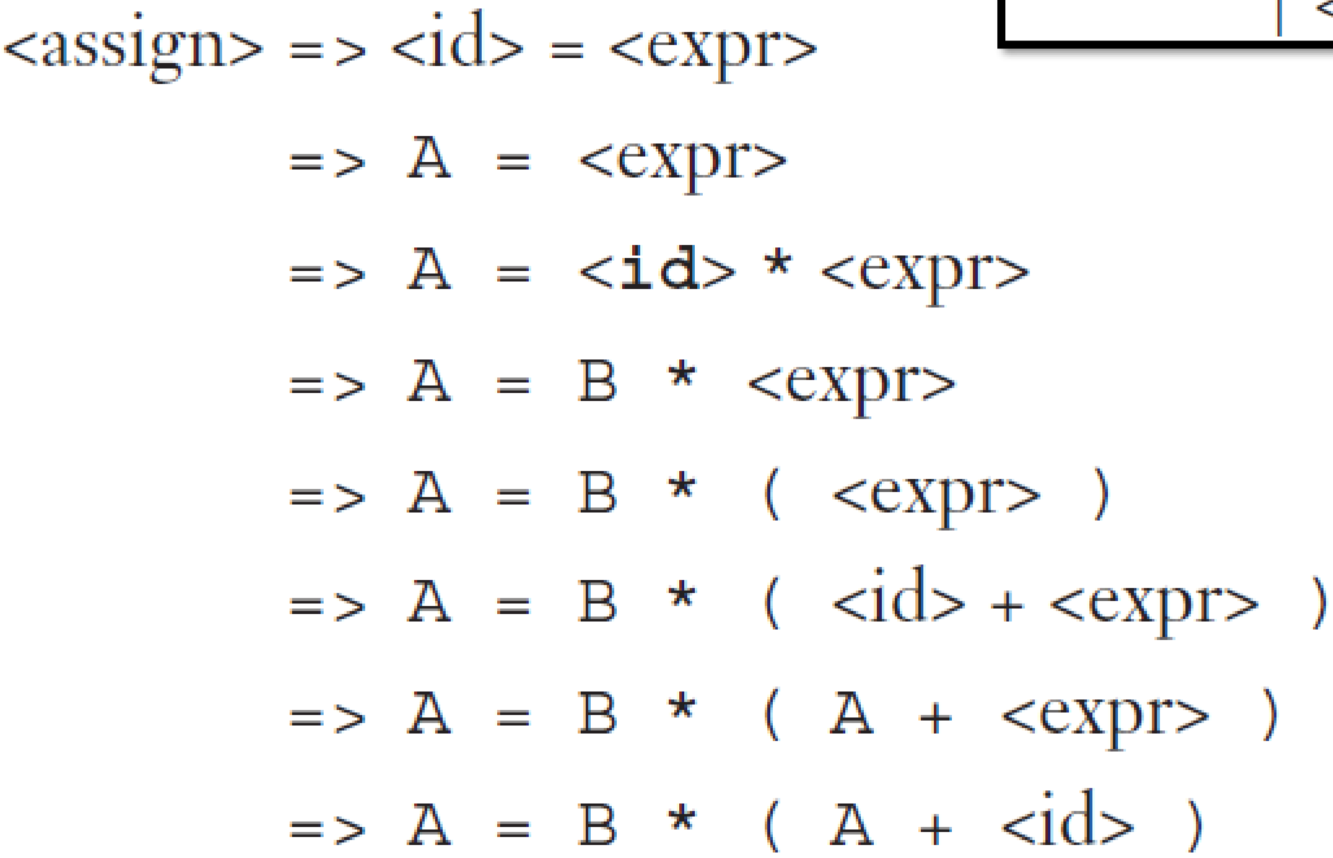
Example (2) (Cont.)



```
<assign> → <id> = <expr>
<id> → A | B | C
<expr> → <id> + <expr>
       | <id> * <expr>
       | ( <expr> )
       | <id>
```

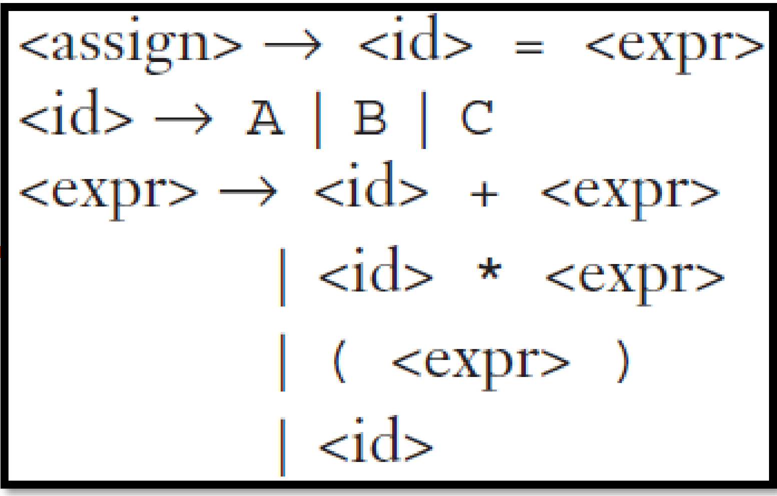


A = B * (A + C)



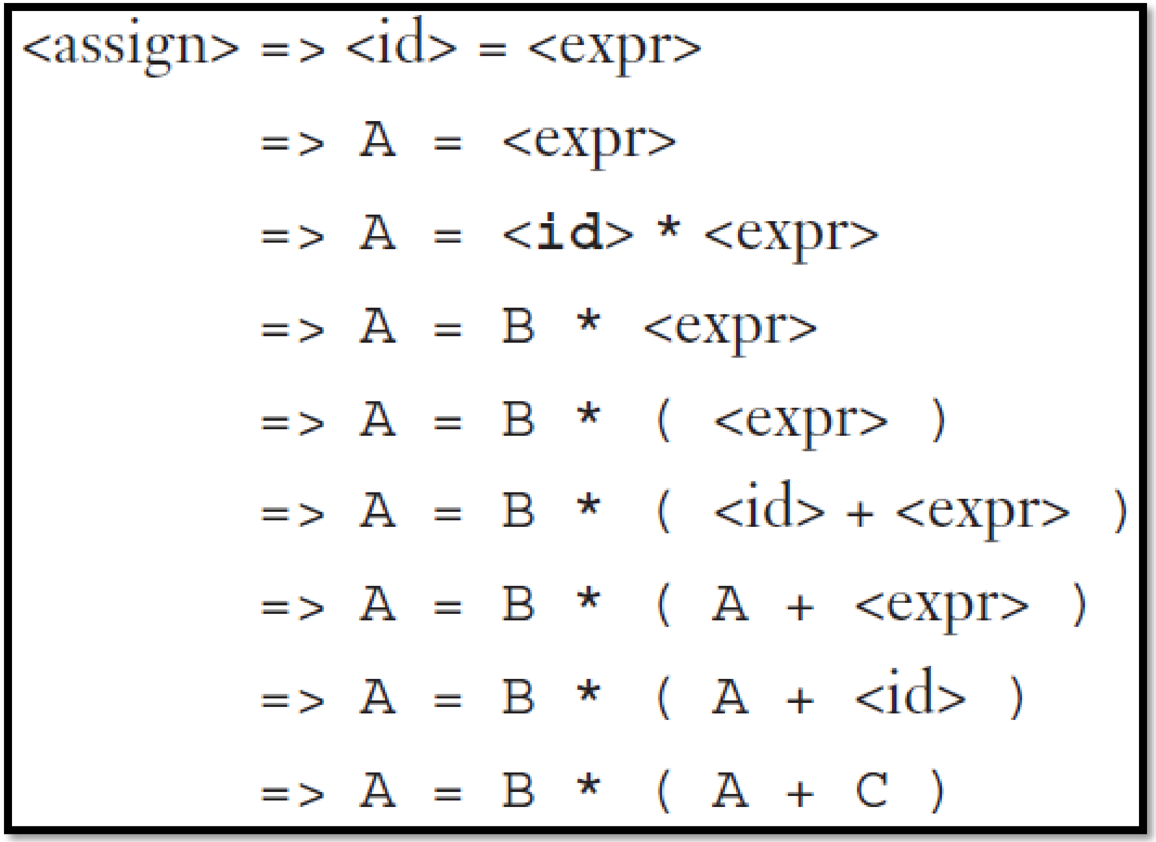
```
<assign> => <id> = <expr>
         => A = <expr>
         => A = <id> * <expr>
         => A = B * <expr>
         => A = B * ( <expr> )
         => A = B * ( <id> + <expr> )
         => A = B * ( A + <expr> )
         => A = B * ( A + <id> )
```

Example (2) (Cont.)



```
<assign> → <id> = <expr>
<id> → A | B | C
<expr> → <id> + <expr>
       | <id> * <expr>
       | ( <expr> )
       | <id>
```

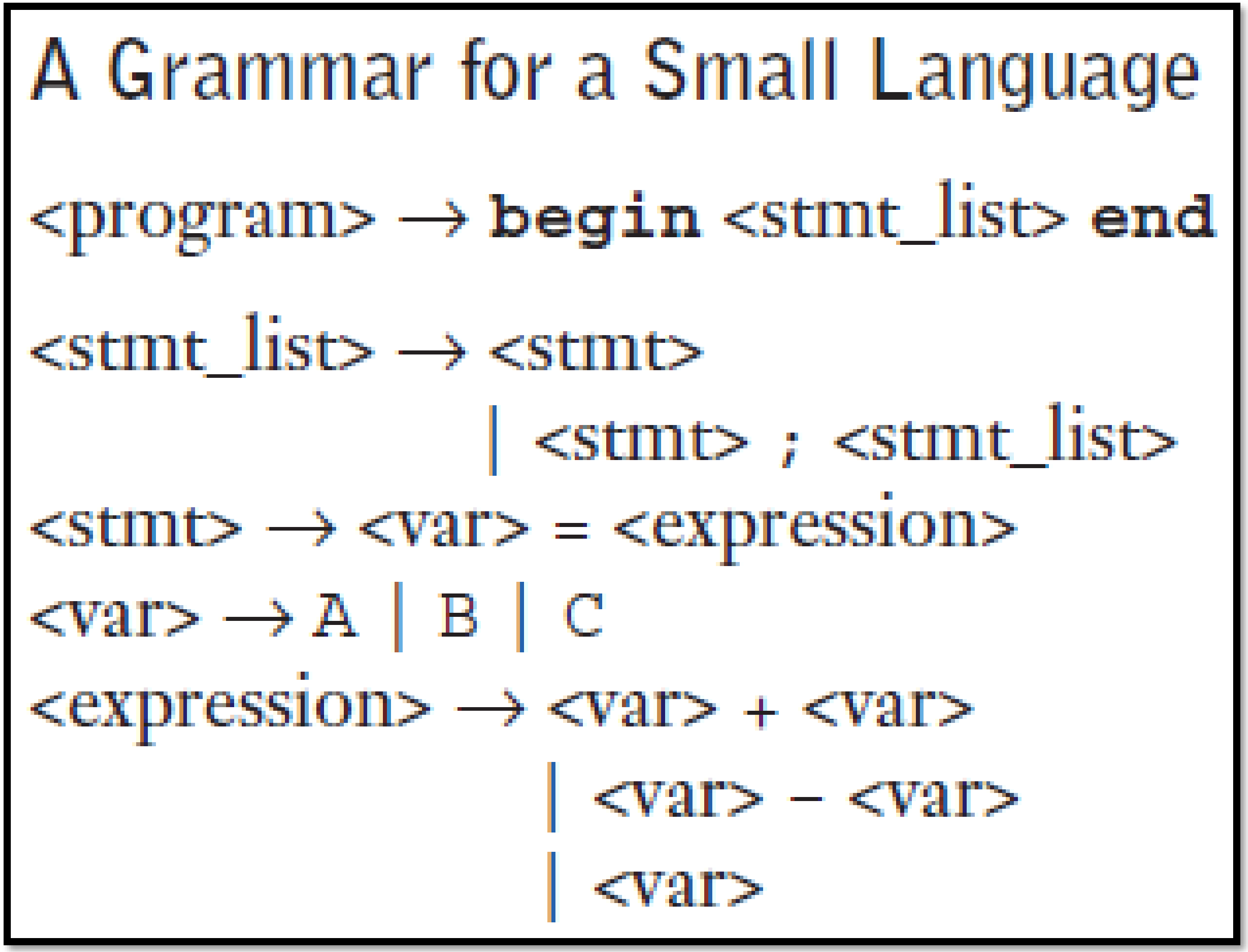
```
A = B * ( A + C )
```



```
<assign> => <id> = <expr>
         => A = <expr>
         => A = <id> * <expr>
         => A = B * <expr>
         => A = B * ( <expr> )
         => A = B * ( <id> + <expr> )
         => A = B * ( A + <expr> )
         => A = B * ( A + <id> )
         => A = B * ( A + C )
```

Accepted

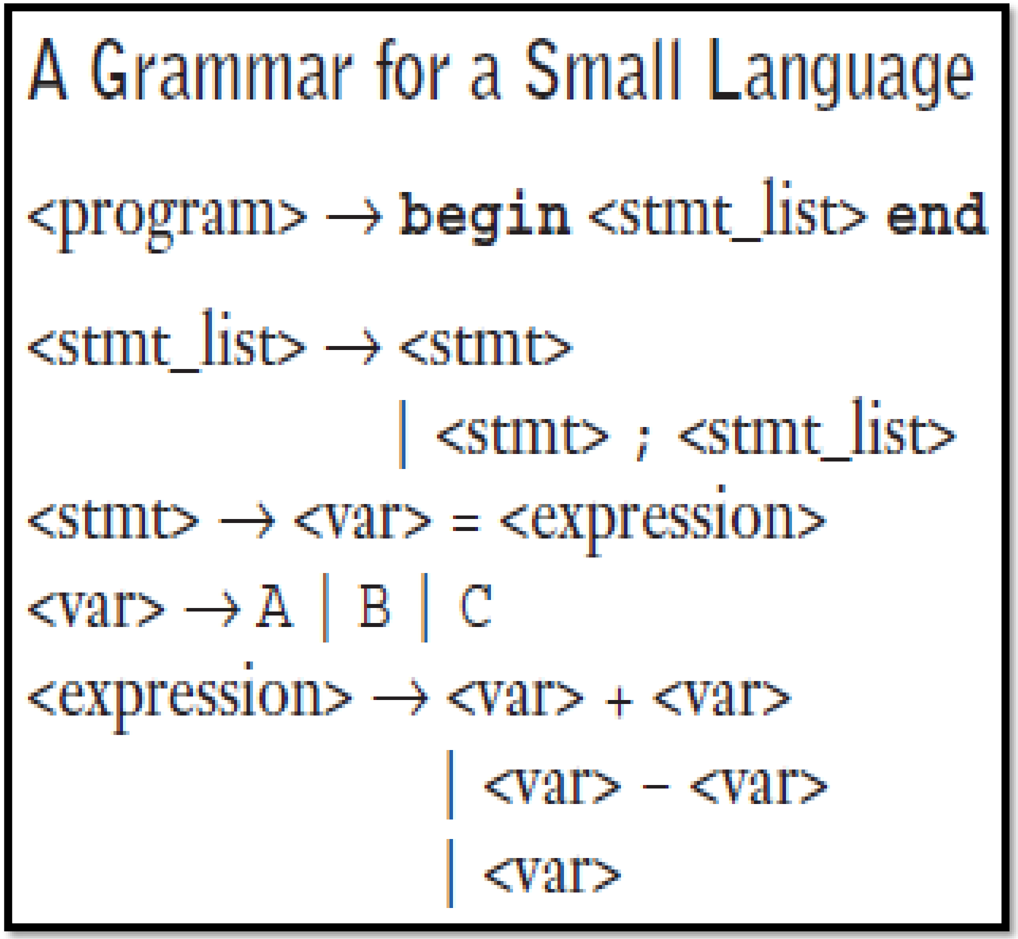
Example (3)



A Grammar for a Small Language

```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> - <var>
             | <var>
```

Example (3) (Cont.)



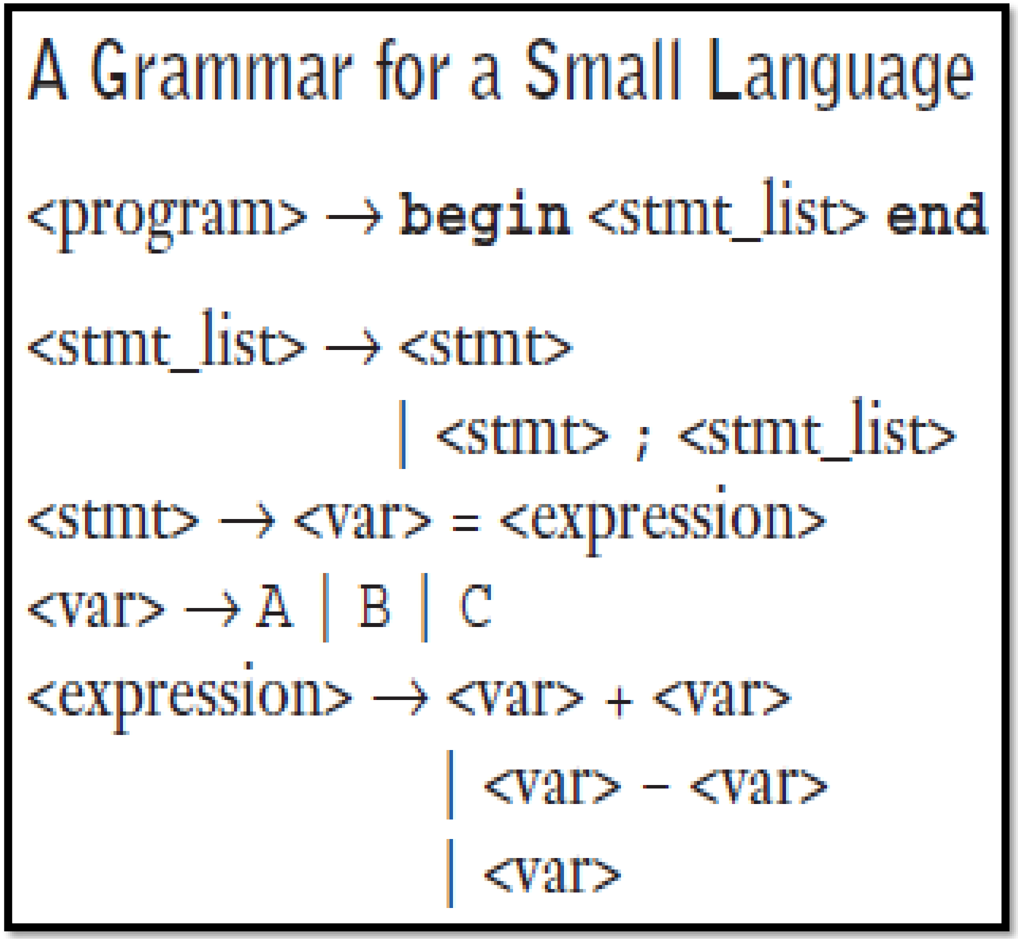
A Grammar for a Small Language

```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> - <var>
             | <var>
```

How many roles?

10

Example (3) (Cont.)



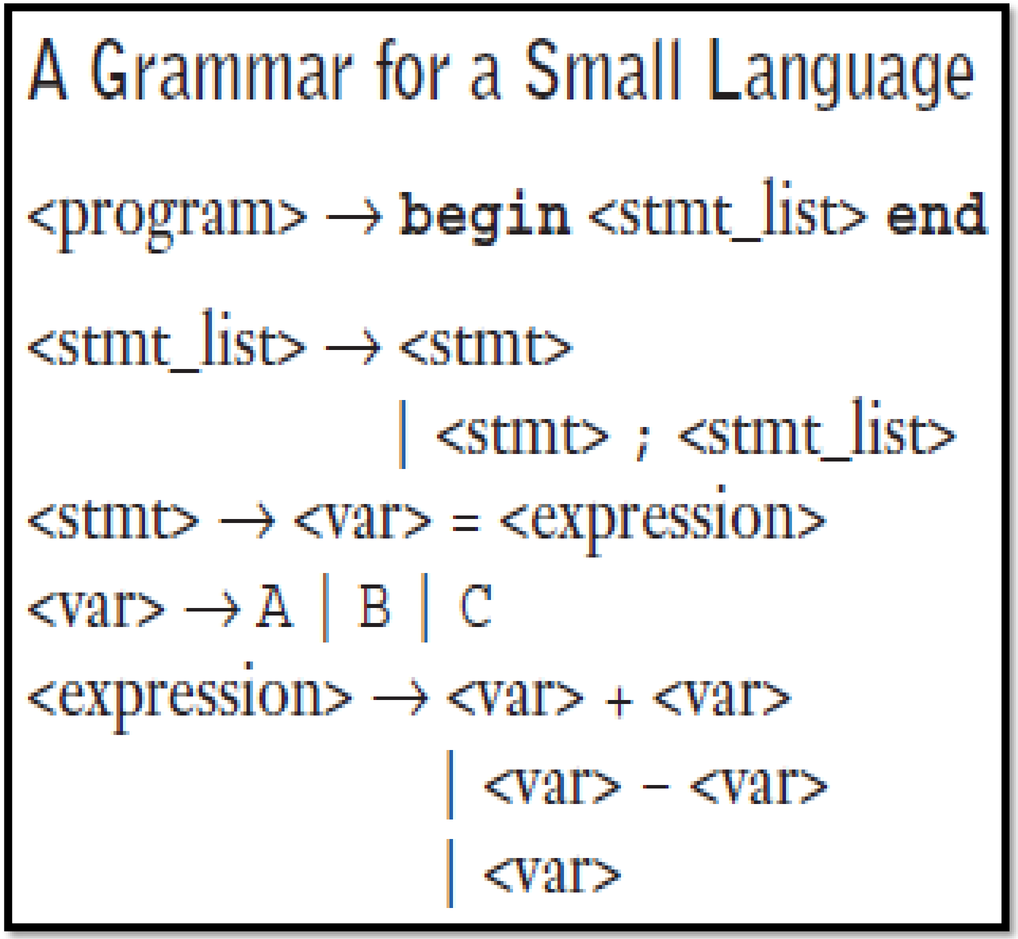
A Grammar for a Small Language

```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> - <var>
             | <var>
```

What are non-terminal(s) of the grammar?

<program>
<stmt_list>
<stmt>
<var>
<expression>

Example (3) (Cont.)



A Grammar for a Small Language

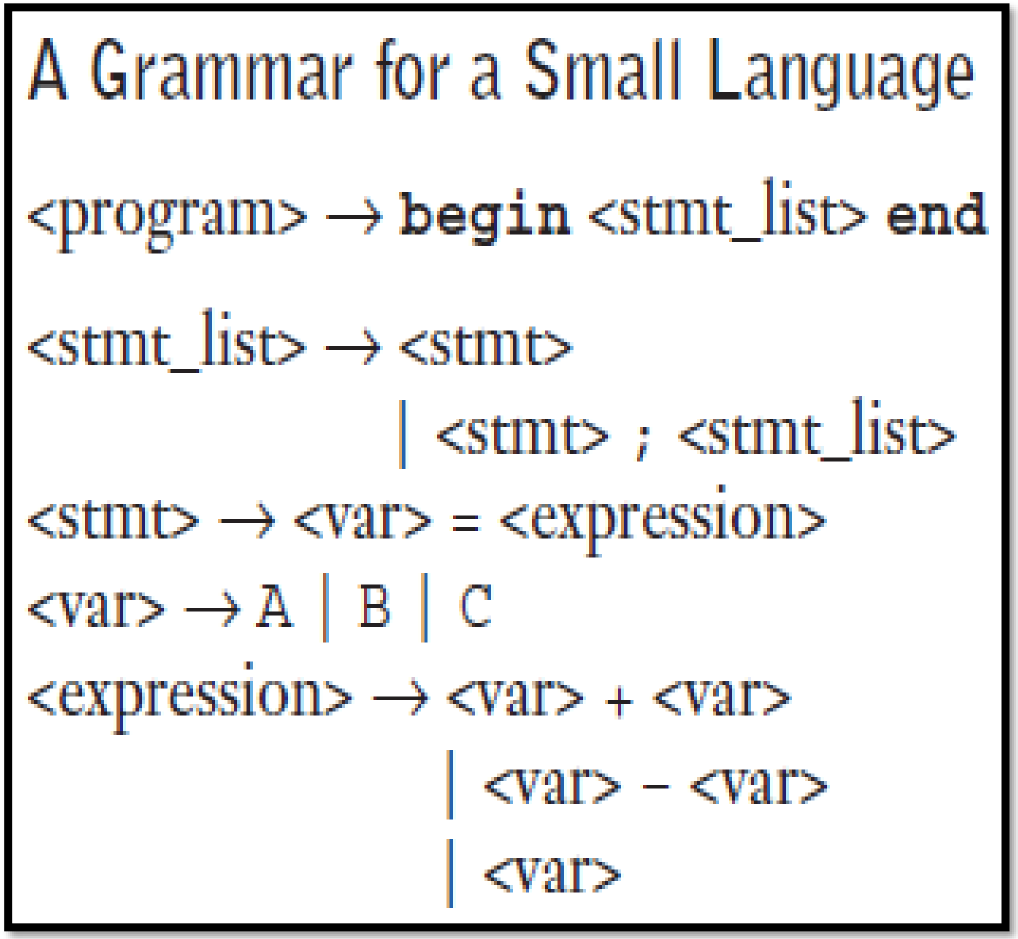
```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> - <var>
             | <var>
```

What is special non-terminal of the grammar?

<program>

called the **start symbol**

Example (3) (Cont.)



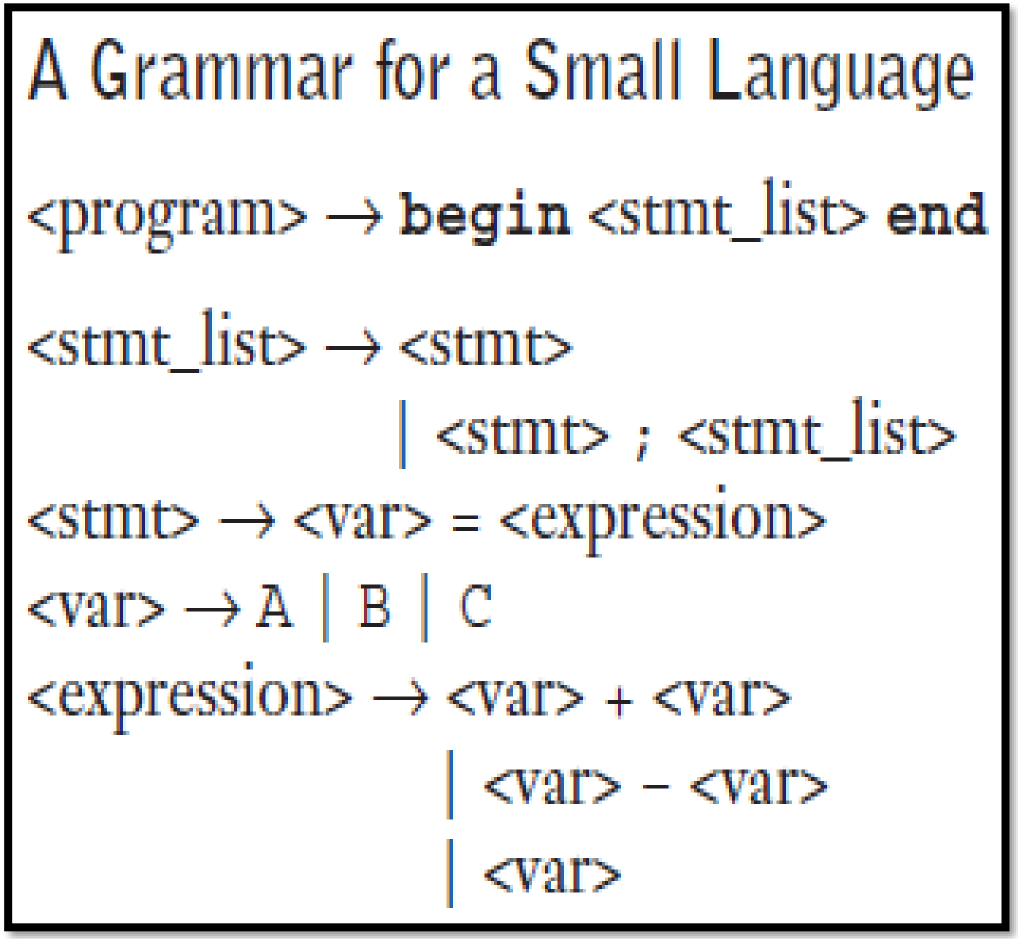
A Grammar for a Small Language

```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> - <var>
             | <var>
```

The grammar has only one statement form assignment.

<var> = <expression>

Example (3) (Cont.)

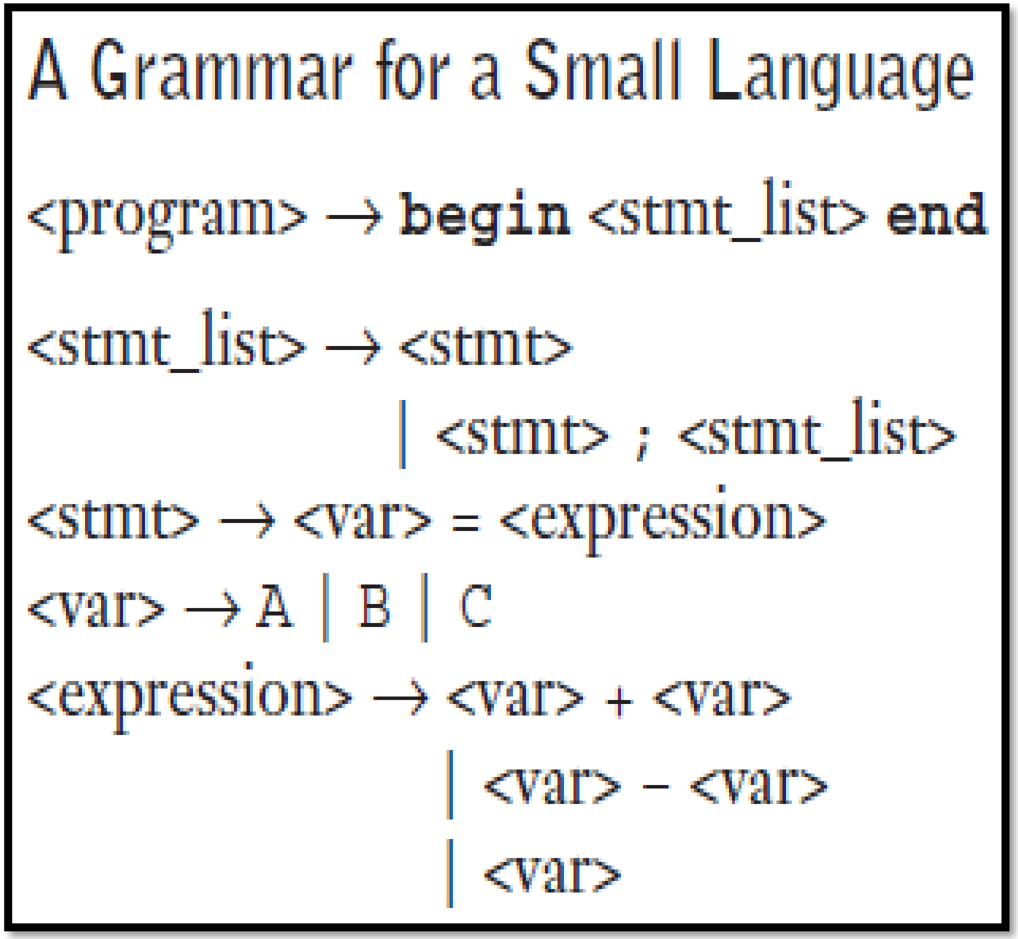


A Grammar for a Small Language

```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> - <var>
             | <var>
```

A program consists of the special word **begin**, followed by a list of statements separated by semicolons, followed by the special word **end**.

Example (3) (Cont.)

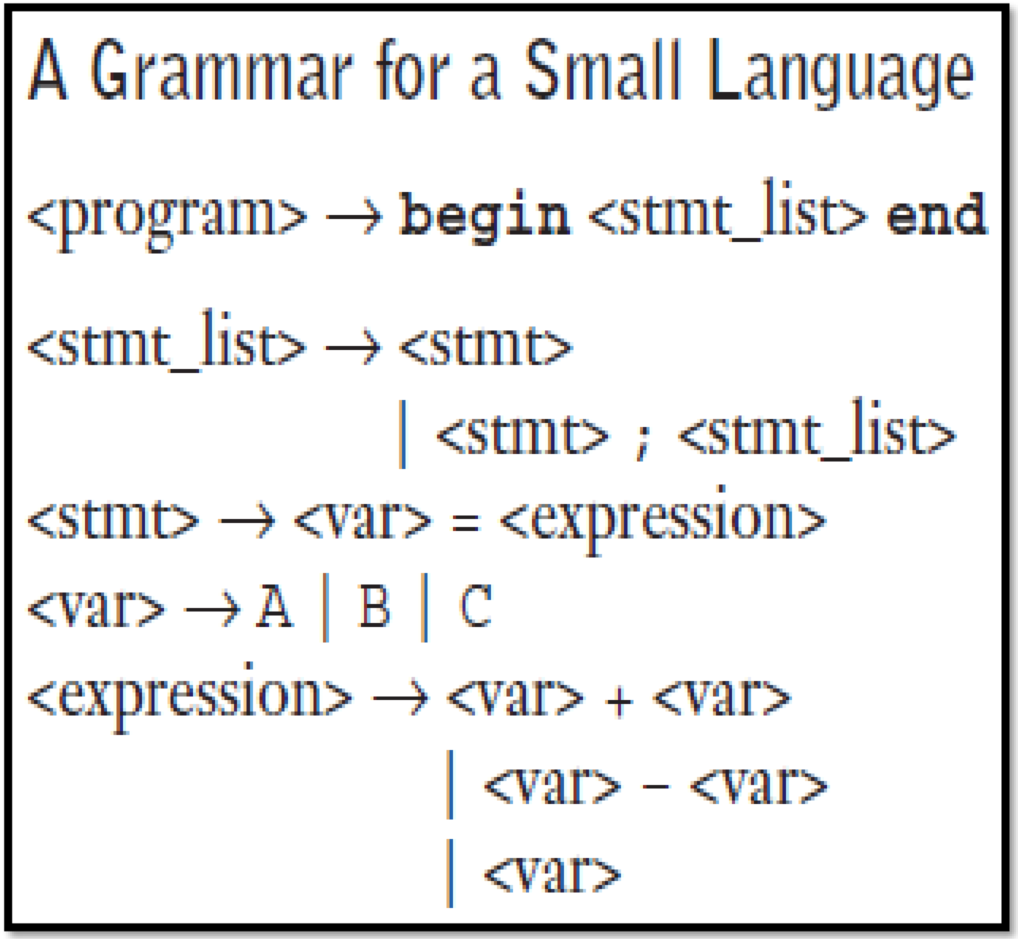


A Grammar for a Small Language

```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> - <var>
             | <var>
```

An expression is either a single variable or two variables separated by either + or − operator.

Example (3) (Cont.)

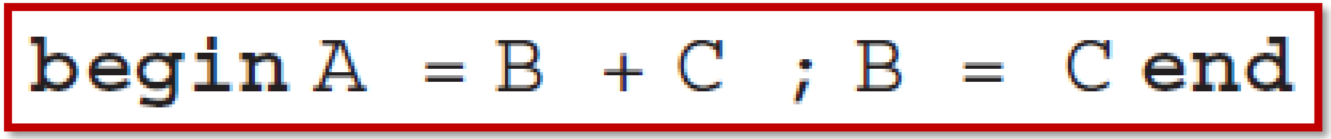


A Grammar for a Small Language

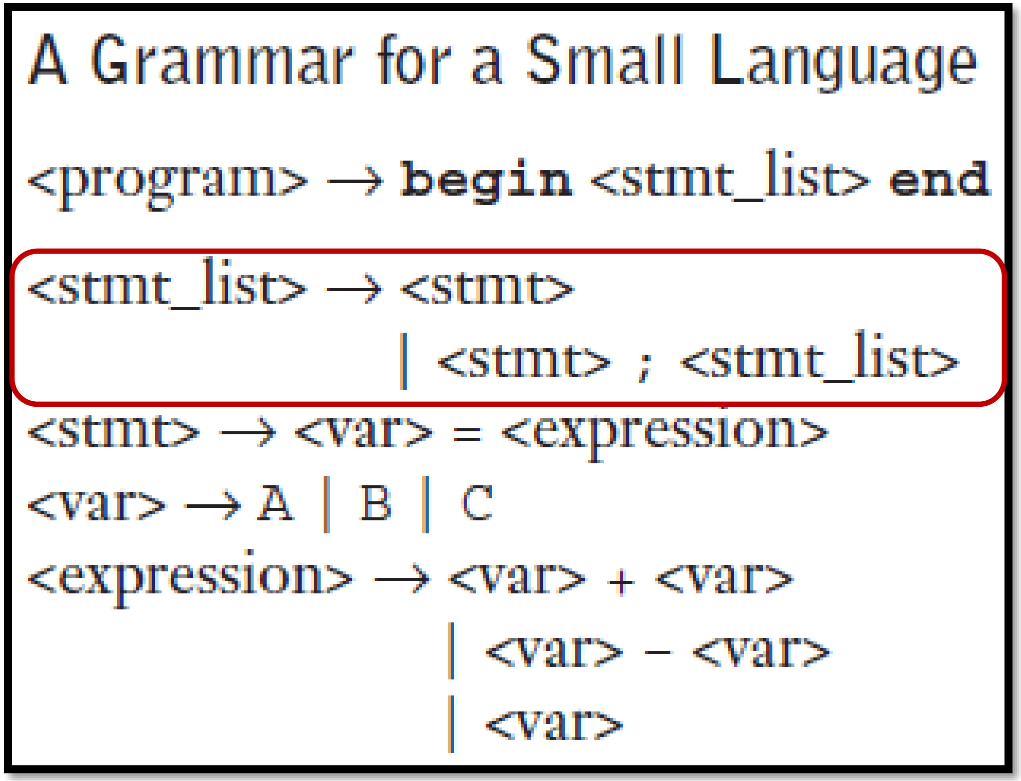
```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> - <var>
             | <var>
```

The only variable names in this language are A, B, and C.

Example (3) (Cont.)



```
begin A = B + C ; B = C end
```

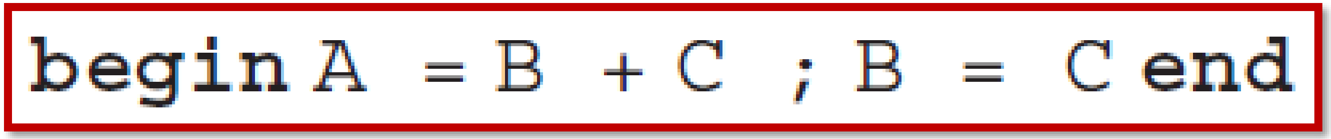


A Grammar for a Small Language

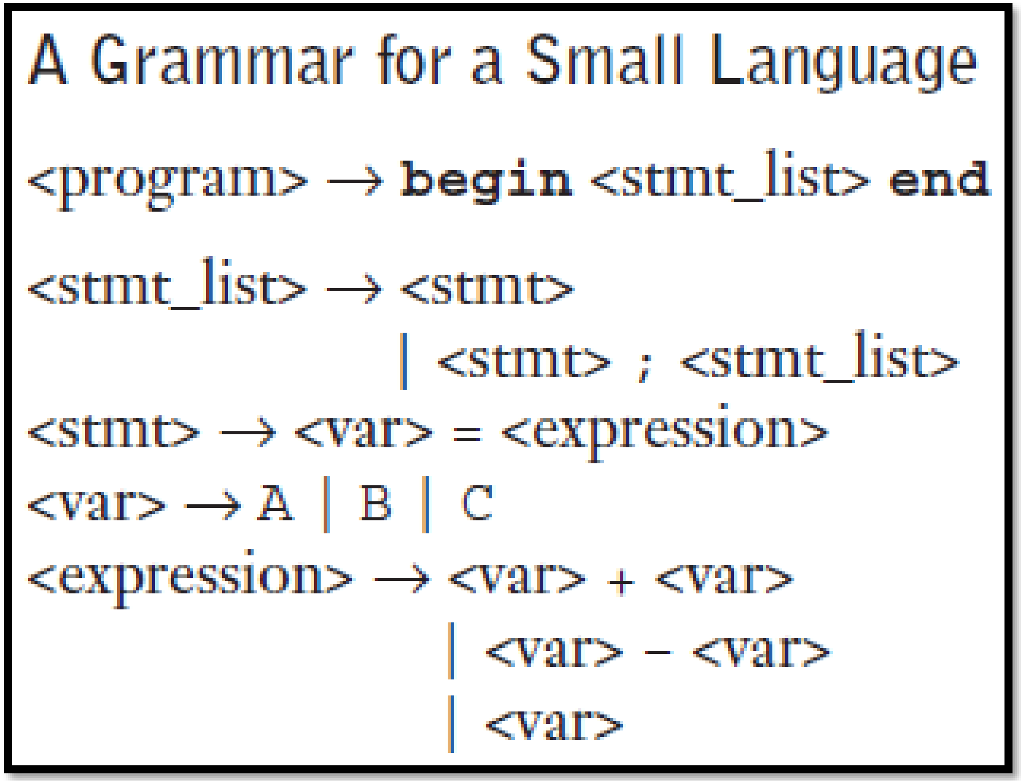
```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> – <var>
             | <var>
```

A **rule** is **recursive** if its LHS appears in its RHS.

Example (3) (Cont.)



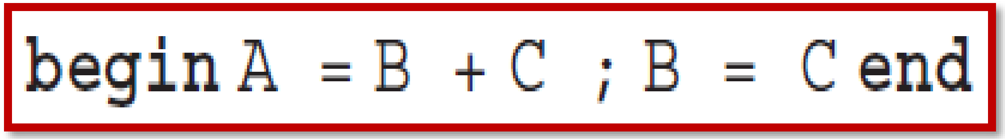
begin A = B + C ; B = C **end**



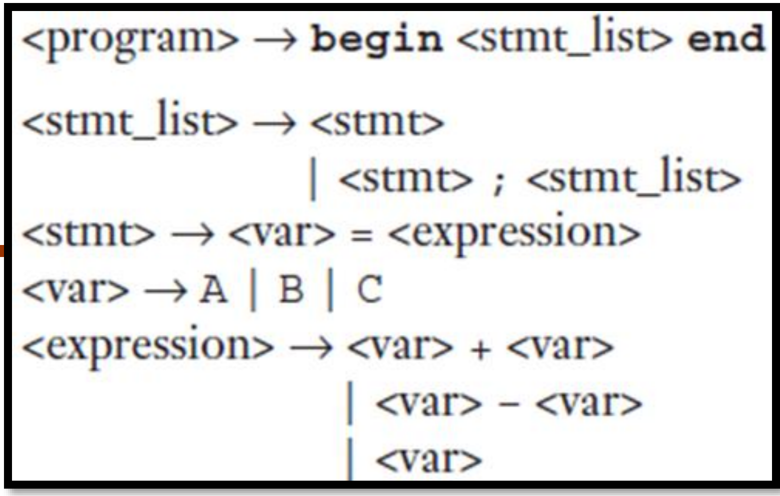
A Grammar for a Small Language

```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> – <var>
             | <var>
```

Example (3) (Cont.)

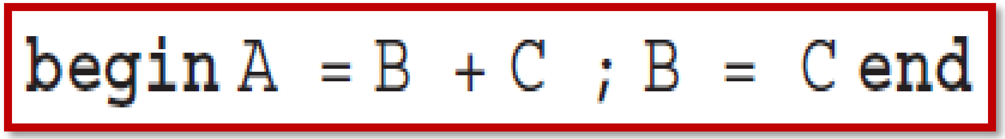


begin A = B + C ; B = C end

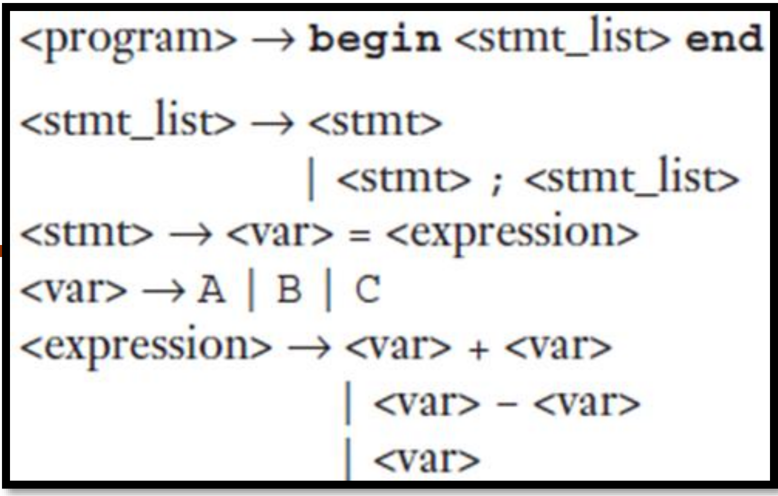


```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> - <var>
             | <var>
```

Example (2) (Cont.)



begin A = B + C ; B = C end

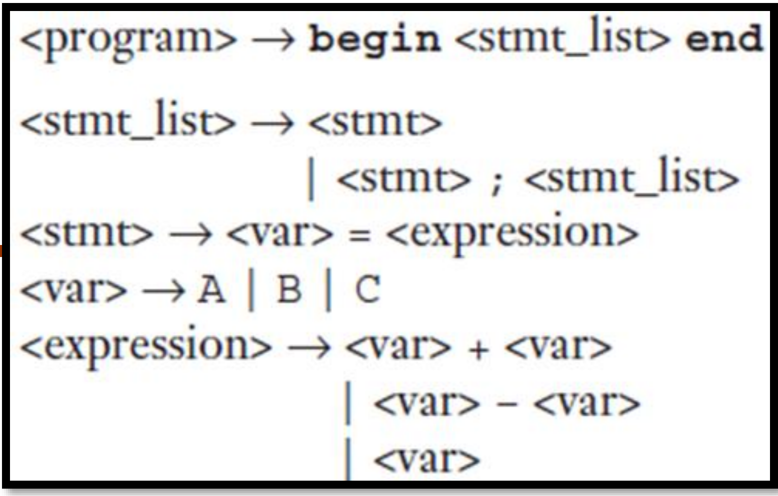


```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> – <var>
             | <var>
```

<program> => begin <stmt_list> end

Example (2) (Cont.)

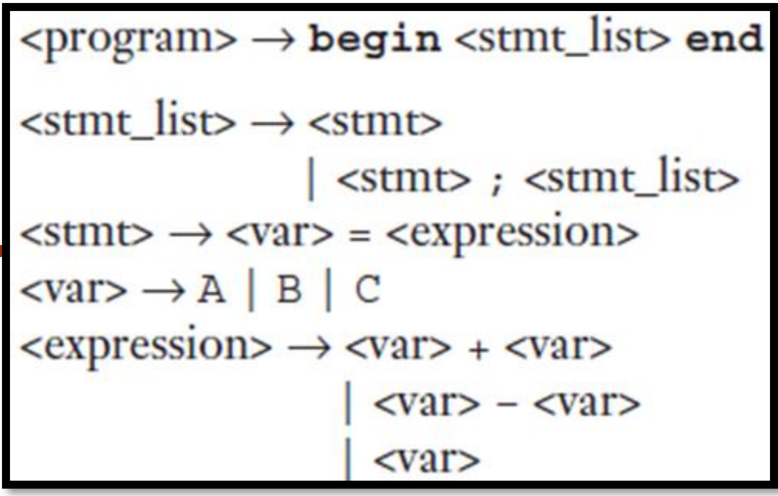
begin A = B + C ; B = C **end**



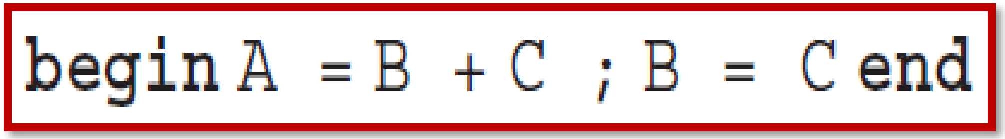
```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> - <var>
             | <var>
```

```
<program> => begin <stmt_list> end
          => begin <stmt> ; <stmt_list> end
```

Example (2) (Cont.)



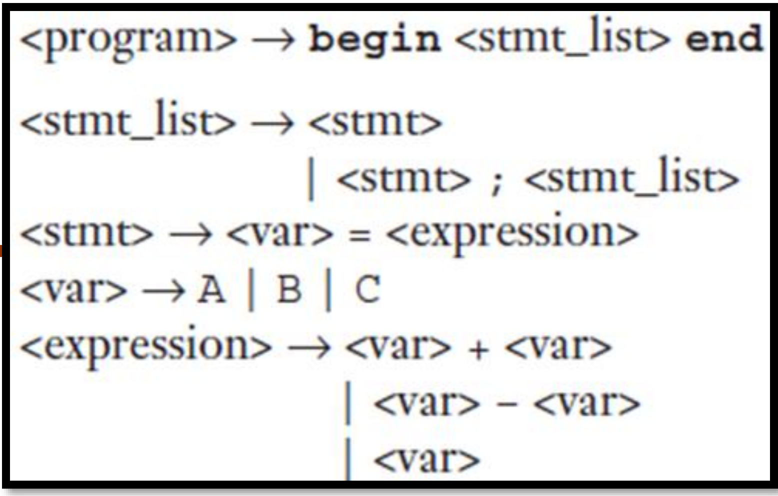
```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> - <var>
             | <var>
```



begin A = B + C ; B = C end

```
<program> => begin <stmt_list> end
          => begin <stmt> ; <stmt_list> end
          => begin <var> = <expression> ; <stmt_list> end
```

Example (2) (Cont.)

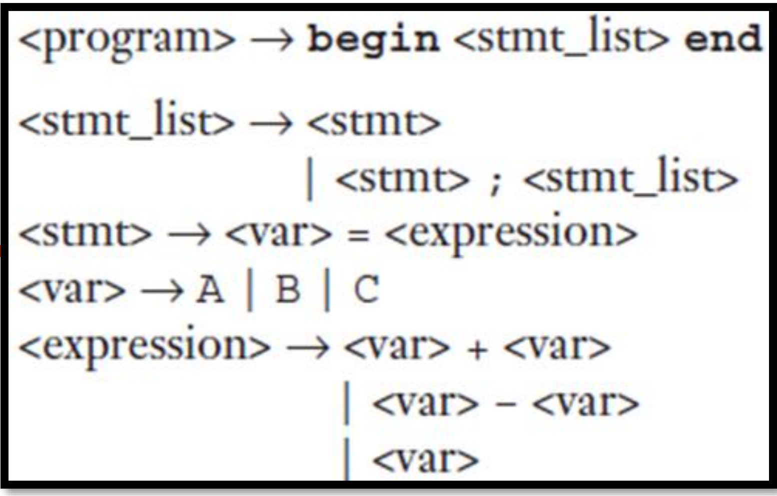


```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> - <var>
             | <var>
```

```
begin A = B + C ; B = C end
```

```
<program> => begin <stmt_list> end
          => begin <stmt> ; <stmt_list> end
          => begin <var> = <expression> ; <stmt_list> end
          => begin A = <expression> ; <stmt_list> end
```

Example (2) (Cont.)



```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> - <var>
             | <var>
```

begin A = B + C ; B = C end

```
<program> => begin <stmt_list> end
          => begin <stmt> ; <stmt_list> end
          => begin <var> = <expression> ; <stmt_list> end
          => begin A = <expression> ; <stmt_list> end
          => begin A = <var> + <var> ; <stmt_list> end
```

Example (2) (Cont.)

<program> → **begin** <stmt_list> **end**

<stmt_list> → <stmt>
             | <stmt> ; <stmt_list>

<stmt> → <var> = <expression>

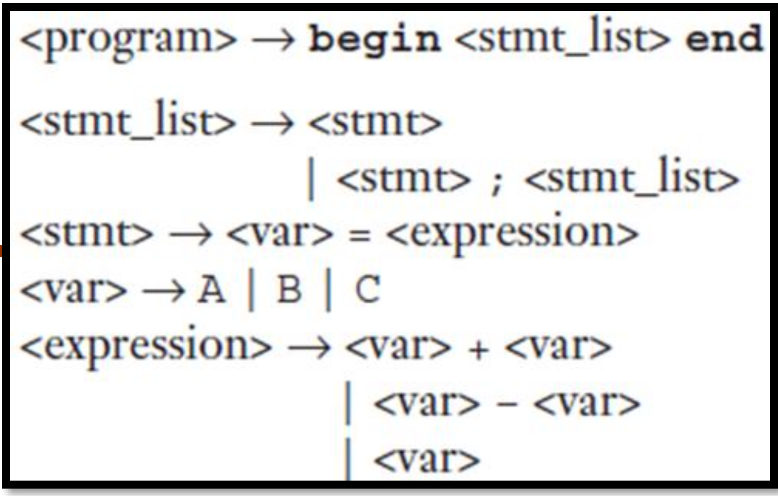
<var> → A | B | C

<expression> → <var> + <var>
               | <var> − <var>
               | <var>

begin A = B + C ; B = C **end**

<program> => **begin** <stmt_list> **end**
          => **begin** <stmt> ; <stmt_list> **end**
          => **begin** <var> = <expression> ; <stmt_list> **end**
          => **begin** A = <expression> ; <stmt_list> **end**
          => **begin** A = <var> + <var> ; <stmt_list> **end**
          => **begin** A = B + <var> ; <stmt_list> **end**

Example (2) (Cont.)

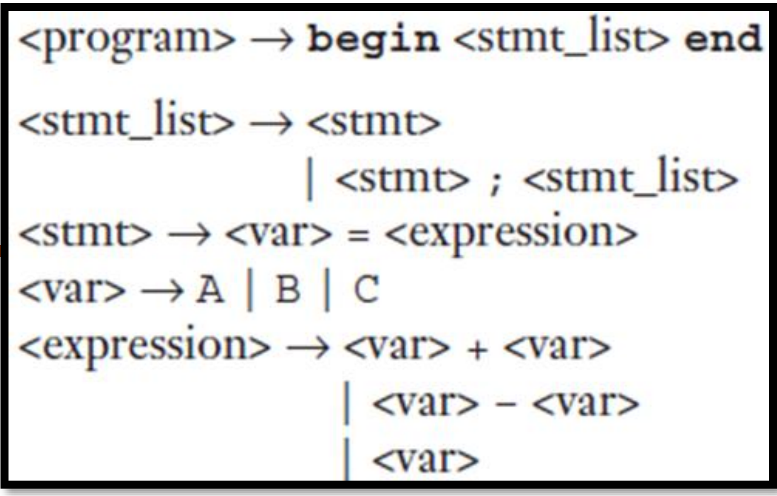


```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> - <var>
             | <var>
```

`begin A = B + C ; B = C end`

```
<program> => begin <stmt_list> end
          => begin <stmt> ; <stmt_list> end
          => begin <var> = <expression> ; <stmt_list> end
          => begin A = <expression> ; <stmt_list> end
          => begin A = <var> + <var> ; <stmt_list> end
          => begin A = B + <var> ; <stmt_list> end
          => begin A = B + C ; <stmt_list> end
```

Example (2) (Cont.)



<program> → **begin** <stmt_list> **end**

<stmt_list> → <stmt>
| <stmt> ; <stmt_list>

<stmt> → <var> = <expression>

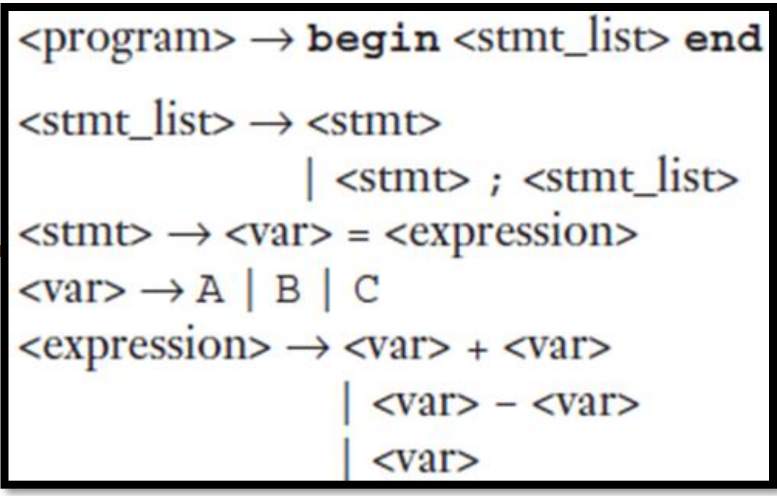
<var> → A | B | C

<expression> → <var> + <var>
| <var> – <var>
| <var>

begin A = B + C ; B = C end

<program> => **begin** <stmt_list> **end**
=> **begin** <stmt> ; <stmt_list> **end**
=> **begin** <var> = <expression> ; <stmt_list> **end**
=> **begin** A = <expression> ; <stmt_list> **end**
=> **begin** A = <var> + <var> ; <stmt_list> **end**
=> **begin** A = B + <var> ; <stmt_list> **end**
=> **begin** A = B + C ; <stmt_list> **end**
=> **begin** A = B + C ; <stmt> **end**

Example (2) (Cont.)

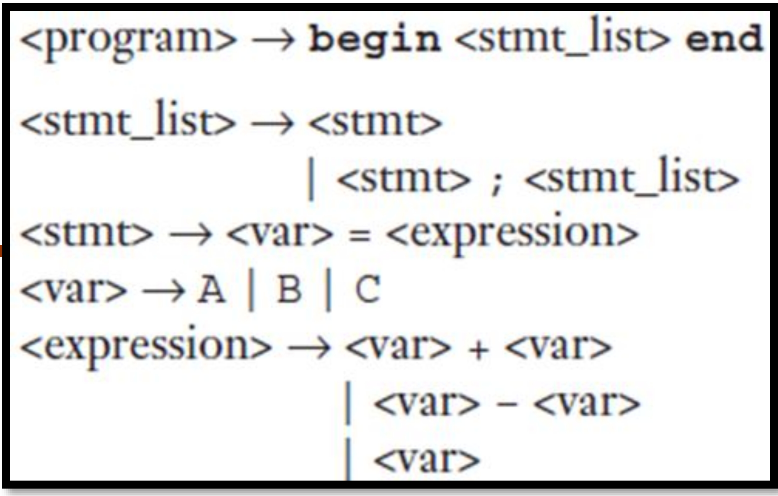


```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> - <var>
             | <var>
```

begin A = B + C ; B = C end

```
<program> => begin <stmt_list> end
          => begin <stmt> ; <stmt_list> end
          => begin <var> = <expression> ; <stmt_list> end
          => begin A = <expression> ; <stmt_list> end
          => begin A = <var> + <var> ; <stmt_list> end
          => begin A = B + <var> ; <stmt_list> end
          => begin A = B + C ; <stmt_list> end
          => begin A = B + C ; <stmt> end
          => begin A = B + C ; <var> = <expression> end
```

Example (2) (Cont.)

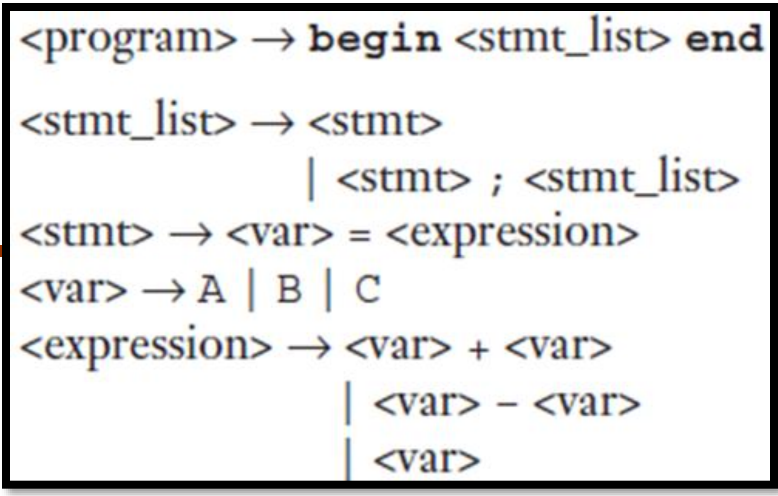


```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> - <var>
             | <var>
```

begin A = B + C ; B = C end

```
<program> => begin <stmt_list> end
          => begin <stmt> ; <stmt_list> end
          => begin <var> = <expression> ; <stmt_list> end
          => begin A = <expression> ; <stmt_list> end
          => begin A = <var> + <var> ; <stmt_list> end
          => begin A = B + <var> ; <stmt_list> end
          => begin A = B + C ; <stmt_list> end
          => begin A = B + C ; <stmt> end
          => begin A = B + C ; <var> = <expression> end
          => begin A = B + C ; B = <expression> end
```

Example (2) (Cont.)



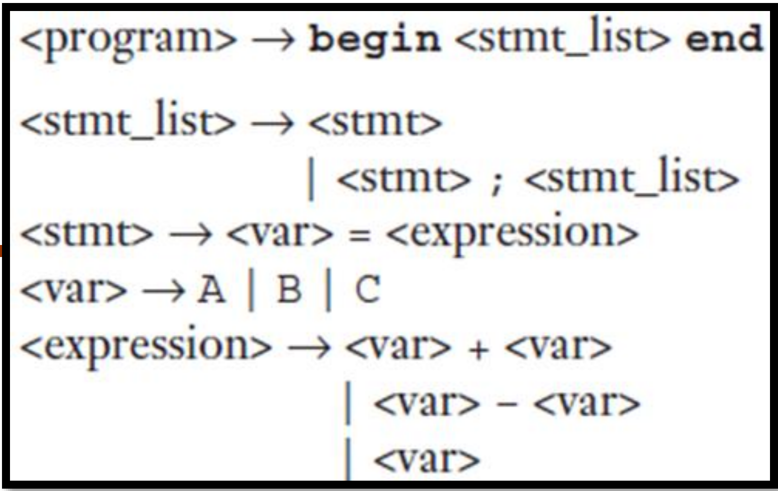
```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> – <var>
             | <var>
```

```
begin A = B + C ; B = C end
```

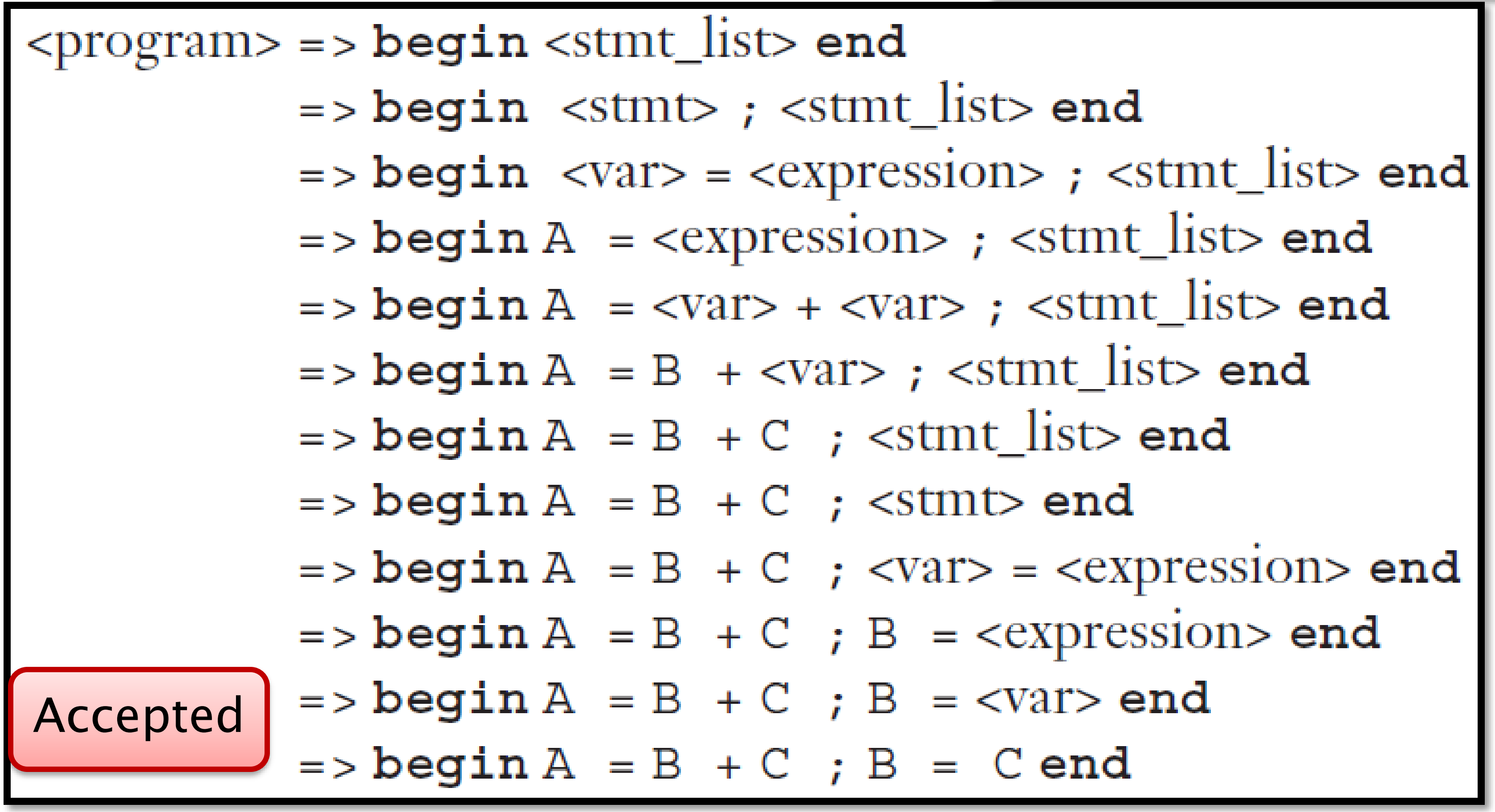
```
<program> => begin <stmt_list> end
          => begin <stmt> ; <stmt_list> end
          => begin <var> = <expression> ; <stmt_list> end
          => begin A = <expression> ; <stmt_list> end
          => begin A = <var> + <var> ; <stmt_list> end
          => begin A = B + <var> ; <stmt_list> end
          => begin A = B + C ; <stmt_list> end
          => begin A = B + C ; <stmt> end
          => begin A = B + C ; <var> = <expression> end
          => begin A = B + C ; B = <expression> end
          => begin A = B + C ; B = <var> end
```

Example (2) (Cont.)

begin A = B + C ; B = C **end**



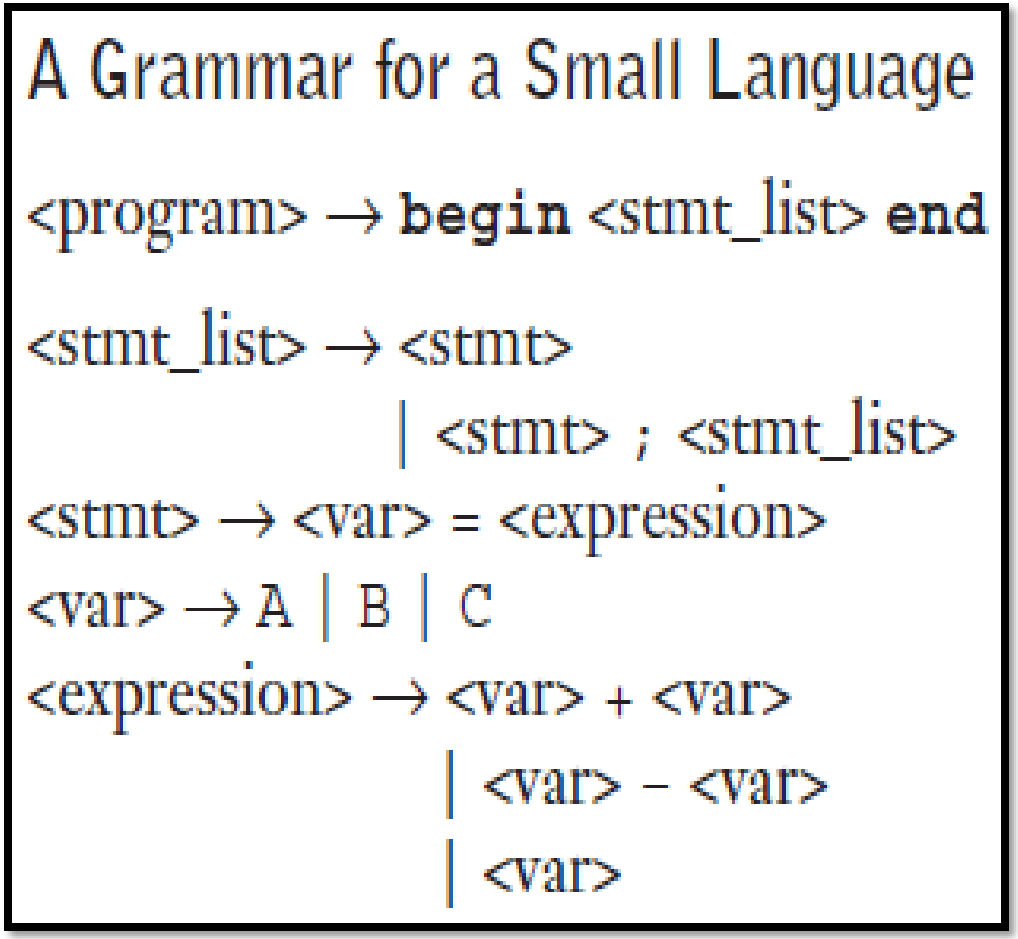
```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> – <var>
             | <var>
```



```
<program> => begin <stmt_list> end
          => begin <stmt> ; <stmt_list> end
          => begin <var> = <expression> ; <stmt_list> end
          => begin A = <expression> ; <stmt_list> end
          => begin A = <var> + <var> ; <stmt_list> end
          => begin A = B + <var> ; <stmt_list> end
          => begin A = B + C ; <stmt_list> end
          => begin A = B + C ; <stmt> end
          => begin A = B + C ; <var> = <expression> end
          => begin A = B + C ; B = <expression> end
          => begin A = B + C ; B = <var> end
          => begin A = B + C ; B = C end
```

Accepted

Example (3) (Cont.)

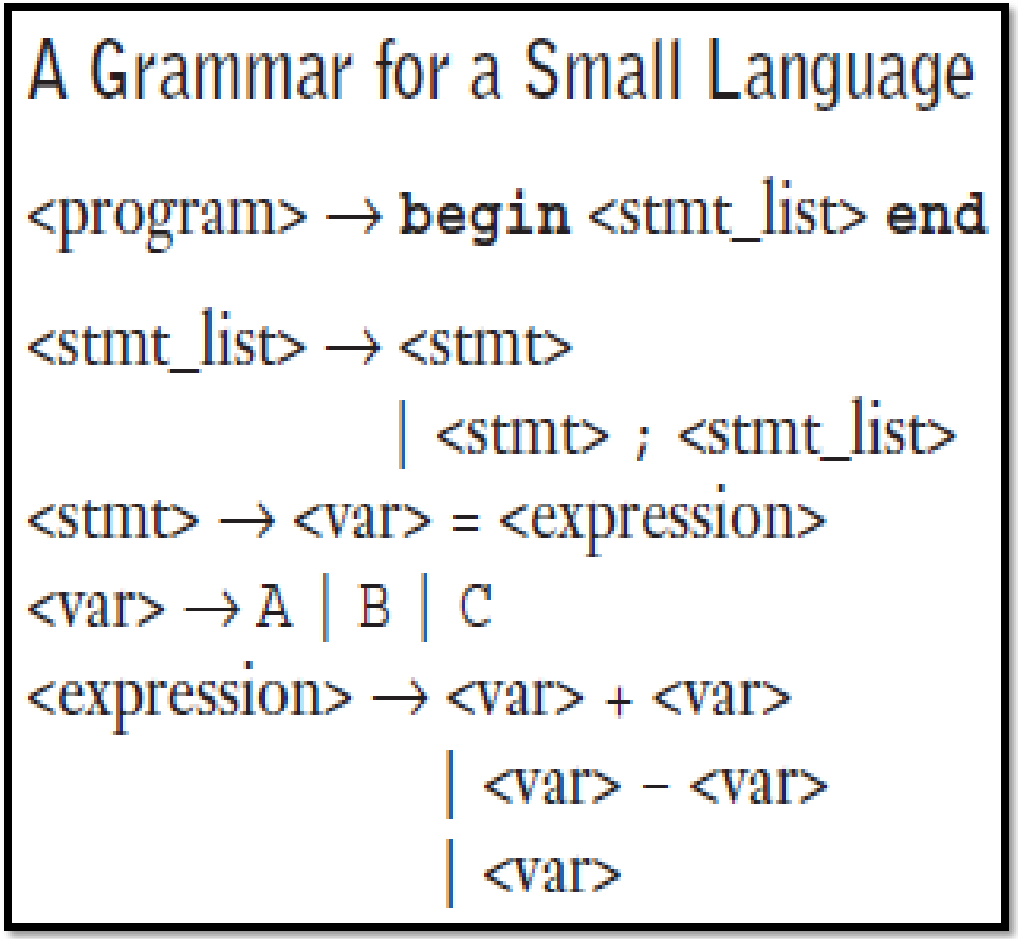


A Grammar for a Small Language

```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> - <var>
             | <var>
```

Derivation that use the leftmost non-terminal in order of replacement are called **leftmost derivations**.

Example (3) (Cont.)



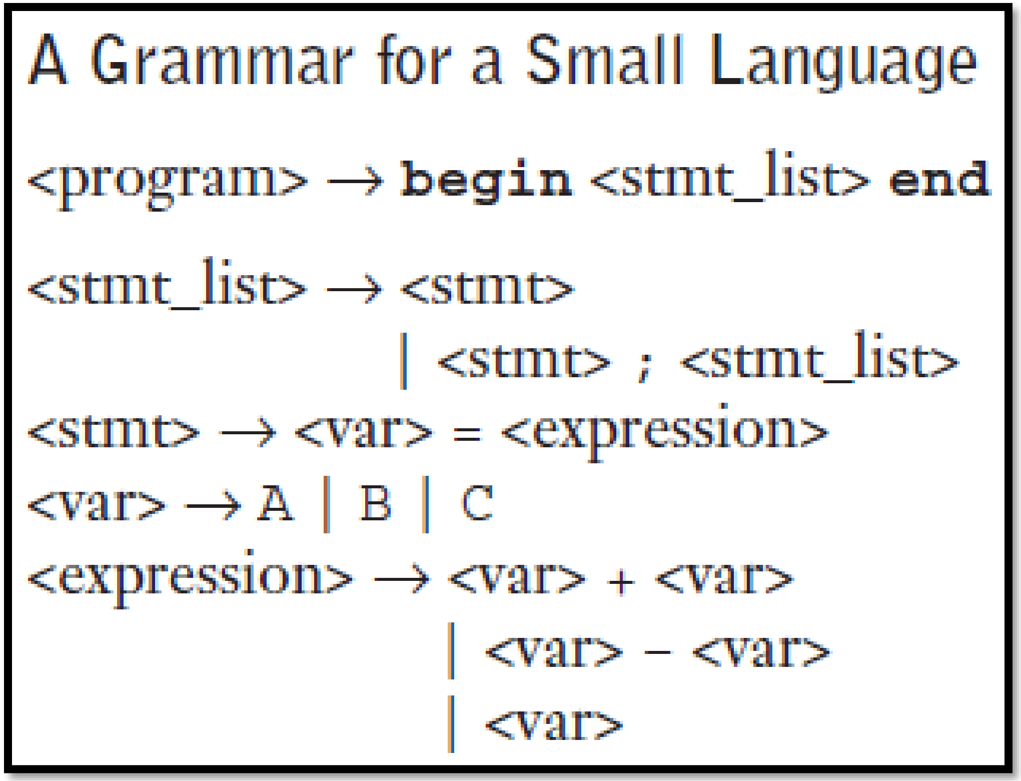
A Grammar for a Small Language

```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> - <var>
             | <var>
```

Derivation that use the rightmost non-terminal in order of replacement are called rightmost derivations.

Example (3) (Cont.)

begin B = C ; A = B + C end

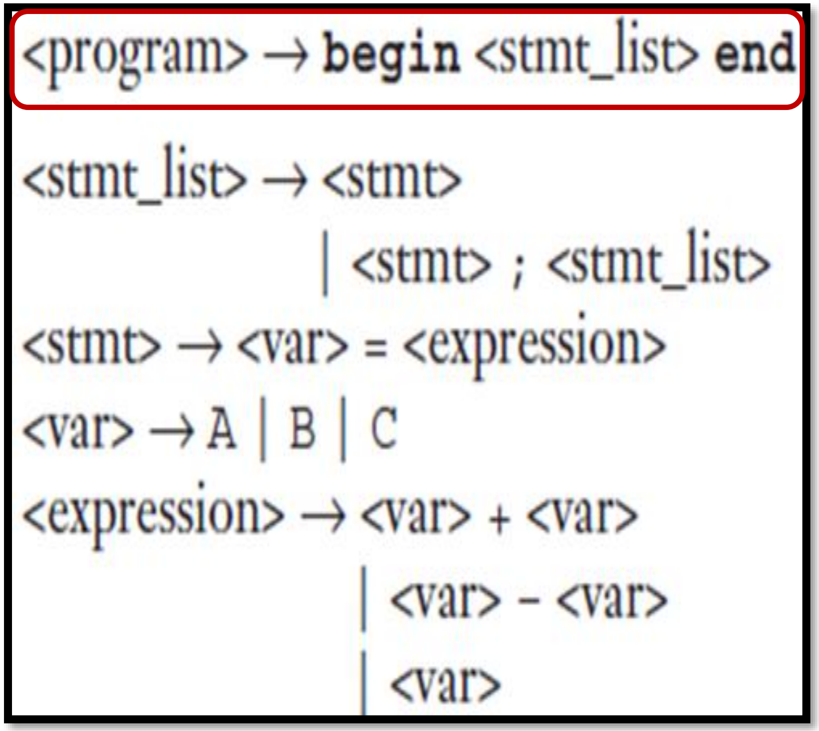


A Grammar for a Small Language

```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> - <var>
             | <var>
```

Derivation use leftmost derivation

Example (3) (Cont.)



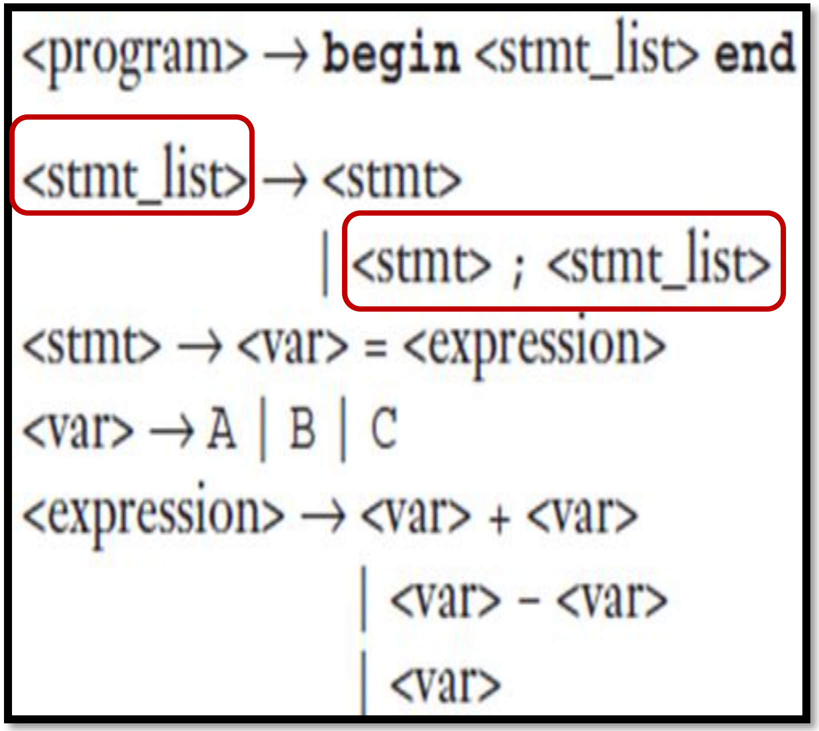
```
<program> → begin <stmt_list> end

<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> - <var>
             | <var>
```

begin B = C ; A = B + C end

<program> → **begin** <stmt_list> **end**

Example (3) (Cont.)



<program> → **begin** <stmt_list> **end**

<stmt_list> → <stmt>
| <stmt> ; <stmt_list>

<stmt> → <var> = <expression>

<var> → A | B | C

<expression> → <var> + <var>
| <var> - <var>
| <var>

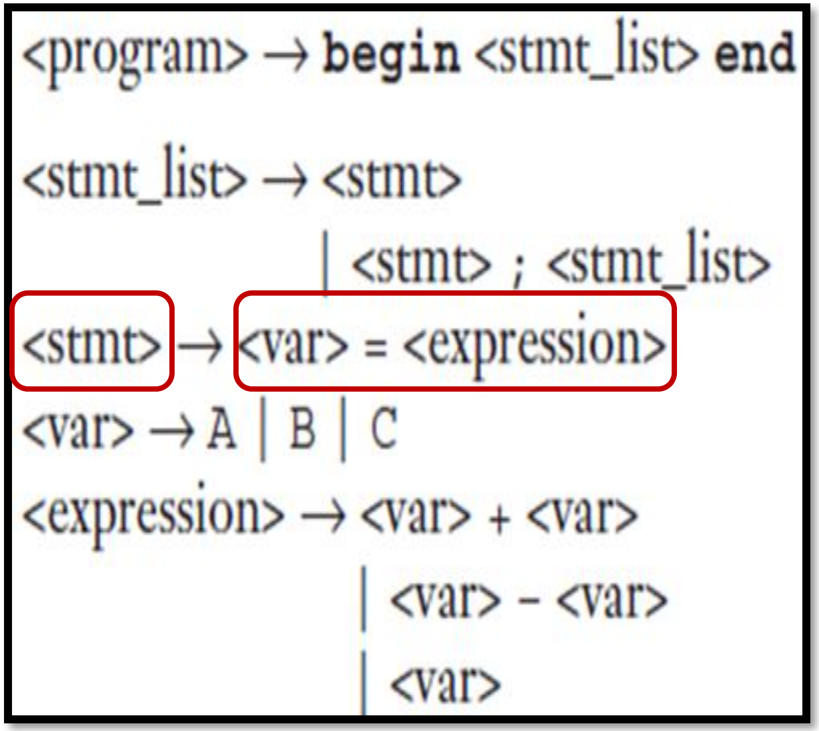
begin B = C ; A = B + C end

<program> → **begin** <stmt_list> **end**

<program> → **begin** <stmt_list> **end**

<program> → **begin** <stmt> ; <stmt_list> **end**

Example (3) (Cont.)



```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> - <var>
             | <var>
```

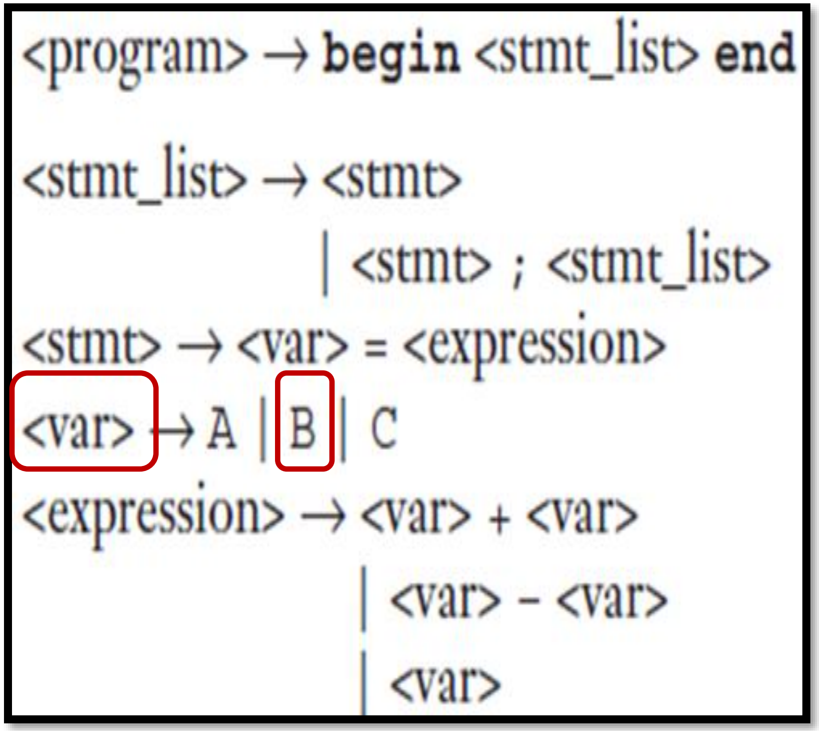
begin B = C ; A = B + C end

<program> → **begin** <stmt_list> **end**

<program> ⇒ **begin** <stmt> ; <stmt_list> **end**

<program> → **begin** <var> = <expression> ; <stmt_list> **end**

Example (3) (Cont.)



```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> - <var>
             | <var>
```

begin B = C ; A = B + C end

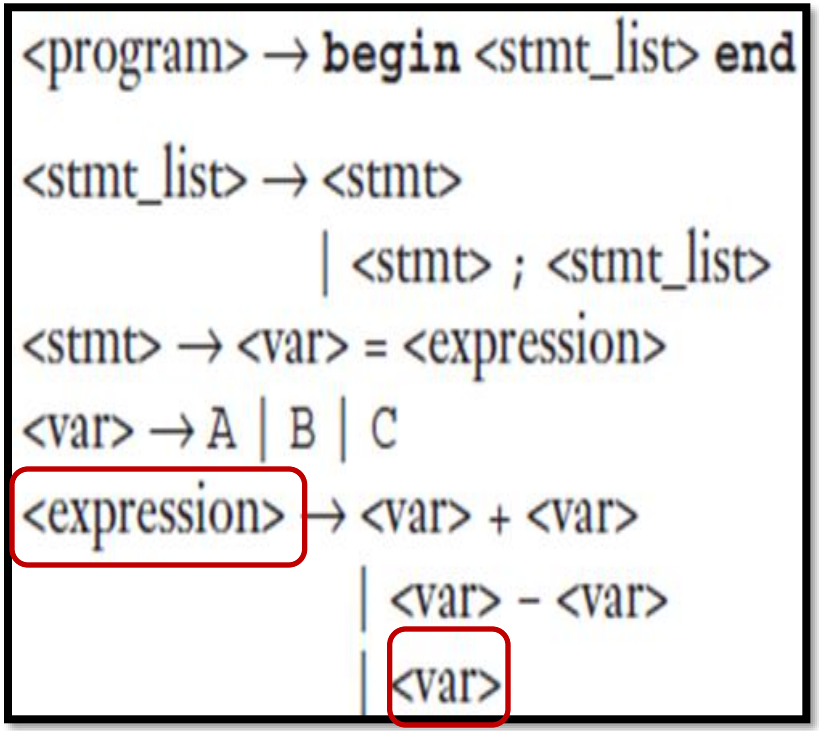
<program> → **begin** <stmt_list> **end**

<program> → **begin** <stmt> ; <stmt_list> **end**

<program> → **begin** <var> = <expression> ; <stmt_list> **end**

<program> → **begin** B = <expression> ; <stmt_list> **end**

Example (3) (Cont.)



<program> → **begin** <stmt_list> **end**

<stmt_list> → <stmt>
　　| <stmt> ; <stmt_list>

<stmt> → <var> = <expression>

<var> → A | B | C

<expression> → <var> + <var>
　　| <var> - <var>
　　| <var>

begin B = C ; A = B + C end

<program> → **begin** <stmt_list> **end**

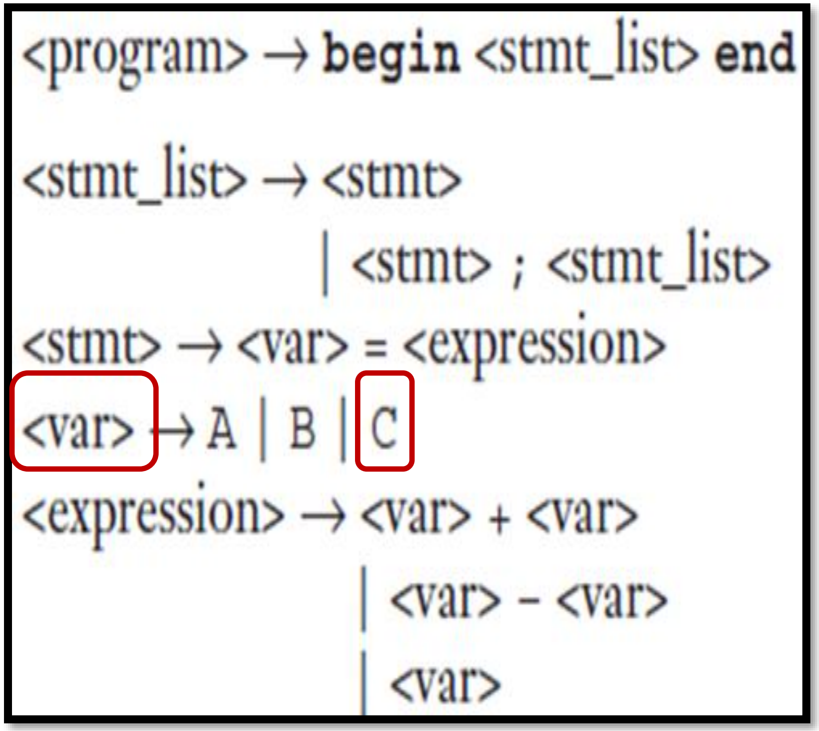
<program> → **begin** <stmt> ; <stmt_list> **end**

<program> → **begin** <var> = <expression> ; <stmt_list> **end**

<program> → **begin** B = <expression> ; <stmt_list> **end**

<program> → **begin** B = <var> ; <stmt_list> **end**

Example (3) (Cont.)



```
<program> → begin <stmt_list> end

<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> - <var>
             | <var>
```

begin B = C ; A = B + C end

<program> → **begin** <stmt_list> **end**

<program> → **begin** <stmt> ; <stmt_list> **end**

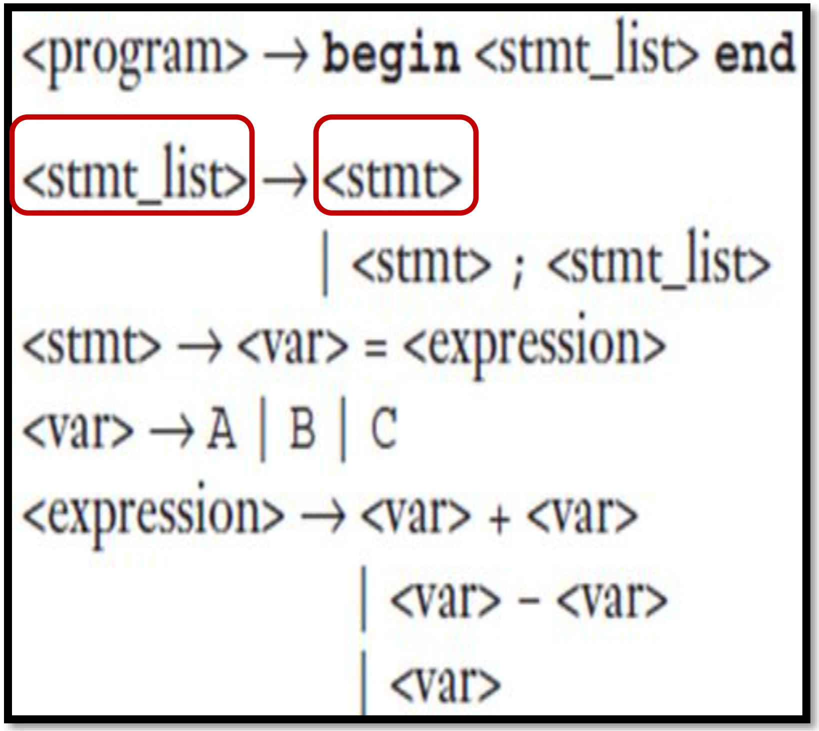
<program> → **begin** <var> = <expression> ; <stmt_list> **end**

<program> → **begin** B = <expression> ; <stmt_list> **end**

<program> → **begin** B = <var> ; <stmt_list> **end**

<program> → **begin** B = C ; <stmt_list> **end**

Example (3) (Cont.)



```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> - <var>
             | <var>
```

begin B = C ; A = B + C end

<program> → **begin** <stmt_list> **end**

<program> → **begin** <stmt> ; <stmt_list> **end**

<program> → **begin** <var> = <expression> ; <stmt_list> **end**

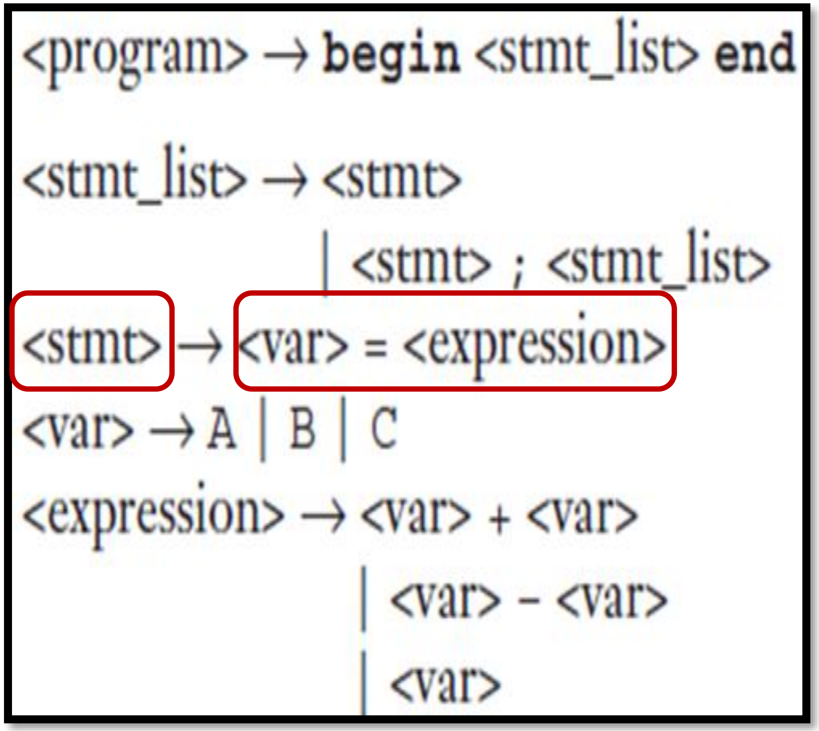
<program> → **begin** B = <expression> ; <stmt_list> **end**

<program> → **begin** B = <var> ; <stmt_list> **end**

<program> → **begin** B = C ; <stmt_list> **end**

<program> → **begin** B = C ; <stmt> **end**

Example (3) (Cont.)



<program> → **begin** <stmt_list> **end**

<stmt_list> → <stmt>

| <stmt> ; <stmt_list>

<stmt> → <var> = <expression>

<var> → A | B | C

<expression> → <var> + <var>

| <var> - <var>

| <var>

begin B = C ; A = B + C end

<program> → **begin** <stmt_list> **end**

<program> → **begin** <stmt> ; <stmt_list> **end**

<program> → **begin** <var> = <expression> ; <stmt_list> **end**

<program> → **begin** B = <expression> ; <stmt_list> **end**

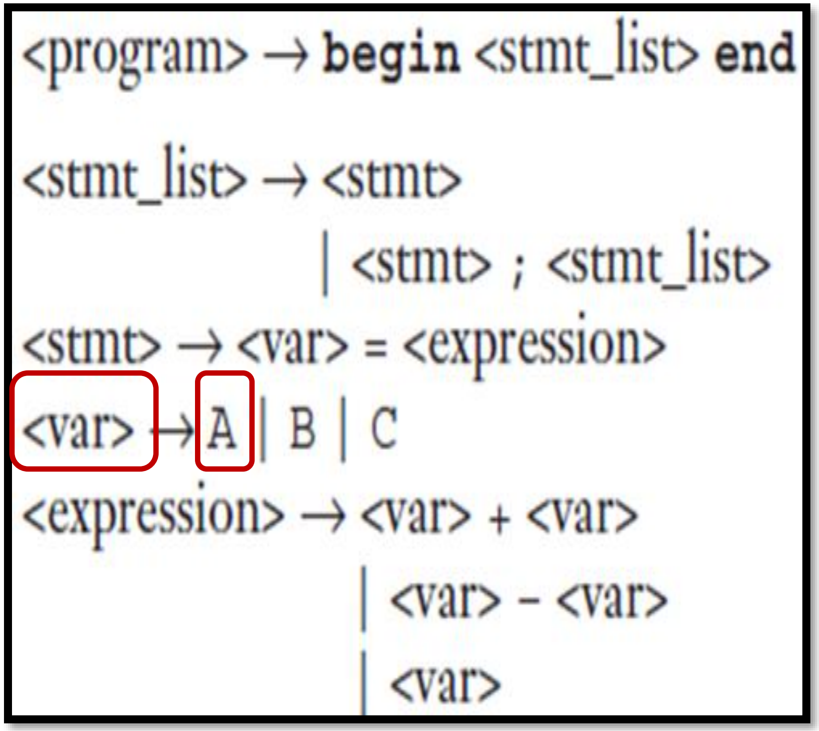
<program> → **begin** B = <var> ; <stmt_list> **end**

<program> → **begin** B = C ; <stmt_list> **end**

<program> → **begin** B = C ; <stmt> **end**

<program> → **begin** B = C ; <var> = <expression> **end**

Example (3) (Cont.)



```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> - <var>
             | <var>
```

begin B = C ; A = B + C end

<program> → **begin** <stmt_list> **end**

<program> → **begin** <stmt> ; <stmt_list> **end**

<program> → **begin** <var> = <expression> ; <stmt_list> **end**

<program> → **begin** B = <expression> ; <stmt_list> **end**

<program> → **begin** B = <var> ; <stmt_list> **end**

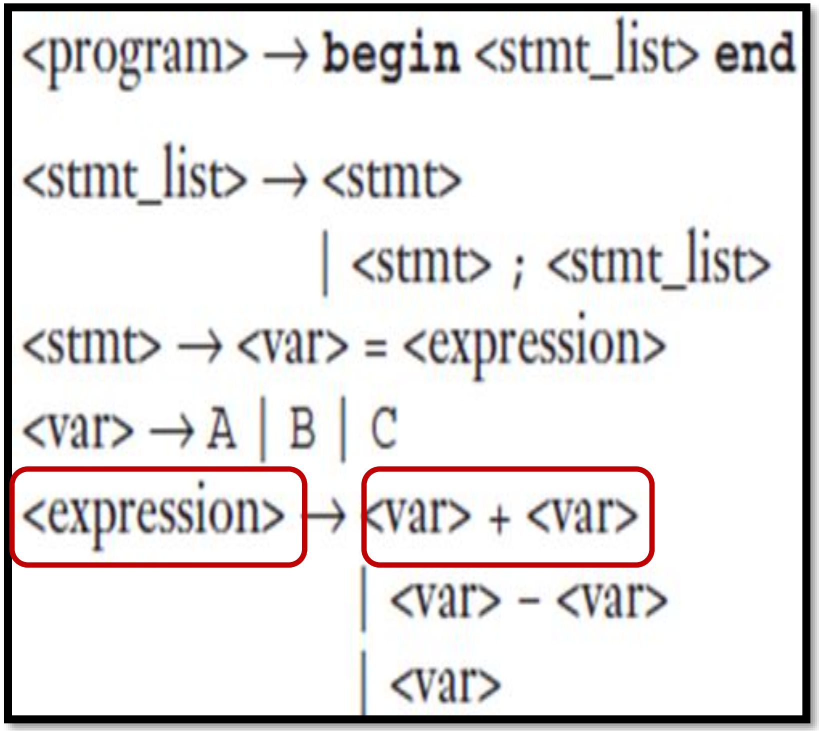
<program> → **begin** B = C ; <stmt_list> **end**

<program> → **begin** B = C ; <stmt> **end**

<program> → **begin** B = C ; <var> = <expression> **end**

<program> → **begin** B = C ; A = <expression> **end**

Example (3) (Cont.)



```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> - <var>
             | <var>
```

begin B = C ; A = B + C end

<program> → **begin** <stmt_list> **end**

<program> → **begin** <stmt> ; <stmt_list> **end**

<program> → **begin** <var> = <expression> ; <stmt_list> **end**

<program> → **begin** B = <expression> ; <stmt_list> **end**

<program> → **begin** B = <var> ; <stmt_list> **end**

<program> → **begin** B = C ; <stmt_list> **end**

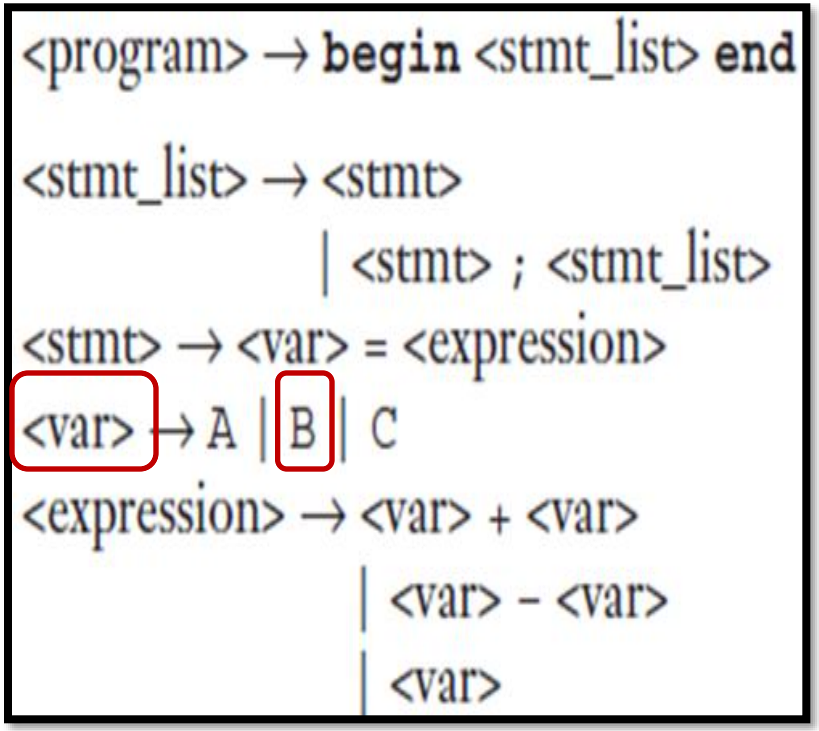
<program> → **begin** B = C ; <stmt> **end**

<program> → **begin** B = C ; <var> = <expression> **end**

<program> → **begin** B = C ; A = <expression> **end**

<program> → **begin** B = C ; A = <var> + <var> **end**

Example (3) (Cont.)



<program> → **begin** <stmt_list> **end**

<stmt_list> → <stmt>
          | <stmt> ; <stmt_list>

<stmt> → <var> = <expression>

<var> → A | B | C

<expression> → <var> + <var>
             | <var> - <var>
             | <var>

begin B = C ; A = B + C end

<program> → **begin** <stmt_list> **end**

<program> → **begin** <stmt> ; <stmt_list> **end**

<program> → **begin** <var> = <expression> ; <stmt_list> **end**

<program> → **begin** B = <expression> ; <stmt_list> **end**

<program> → **begin** B = <var> ; <stmt_list> **end**

<program> → **begin** B = C ; <stmt_list> **end**

<program> → **begin** B = C ; <stmt> **end**

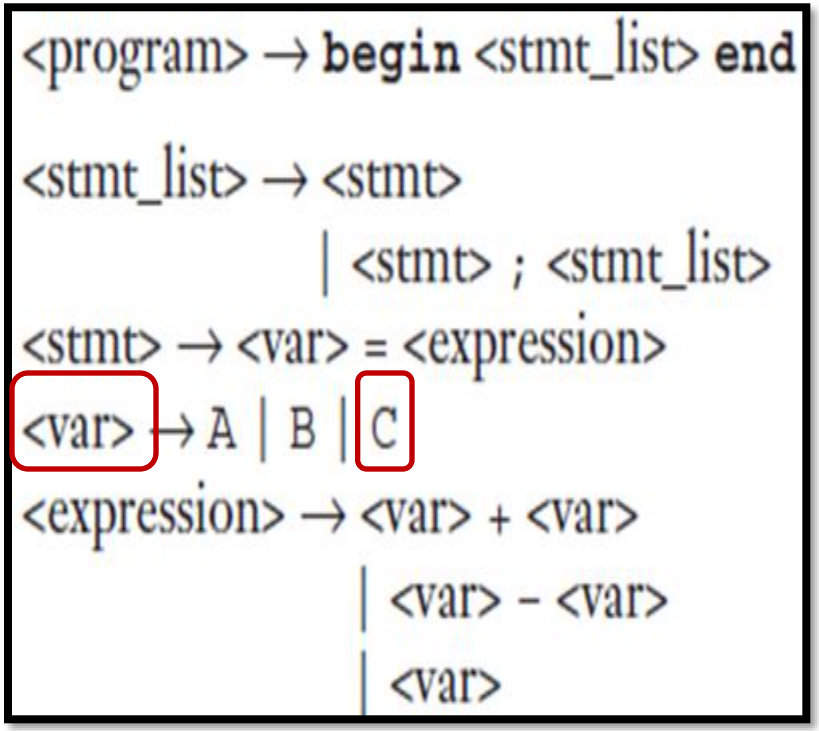
<program> → **begin** B = C ; <var> = <expression> **end**

<program> → **begin** B = C ; A = <expression> **end**

<program> → **begin** B = C ; A = <var> + <var> **end**

<program> → **begin** B = C ; A = B + <var> **end**

Example (3) (Cont.)



```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> - <var>
             | <var>
```

begin B = C ; A = B + C end

<program> → **begin** <stmt_list> **end**

<program> → **begin** <stmt> ; <stmt_list> **end**

<program> → **begin** <var> = <expression> ; <stmt_list> **end**

<program> → **begin** B = <expression> ; <stmt_list> **end**

<program> → **begin** B = <var> ; <stmt_list> **end**

<program> → **begin** B = C ; <stmt_list> **end**

<program> → **begin** B = C ; <stmt> **end**

<program> → **begin** B = C ; <var> = <expression> **end**

<program> → **begin** B = C ; A = <expression> **end**

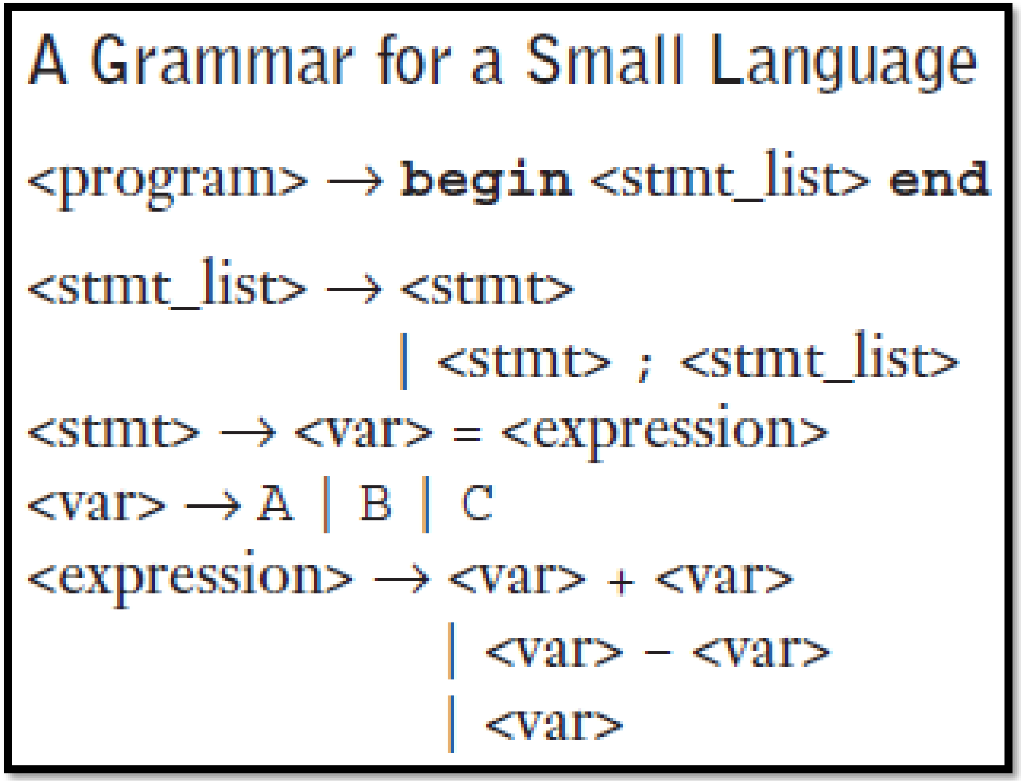
<program> → **begin** B = C ; A = <var> + <var> **end**

<program> → **begin** B = C ; A = B + <var> **end**

<program> → **begin** B = C ; A = B + C **end**

Example (3) (Cont.)

begin B = C ; A = B + C end

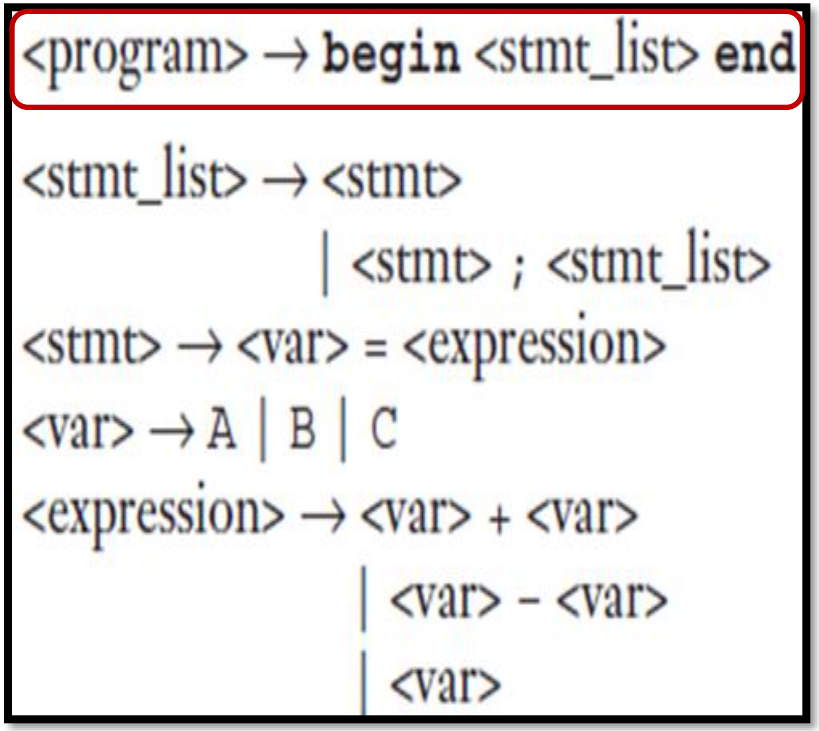


A Grammar for a Small Language

```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> - <var>
             | <var>
```

Derivation use rightmost derivation

Example (3) (Cont.)



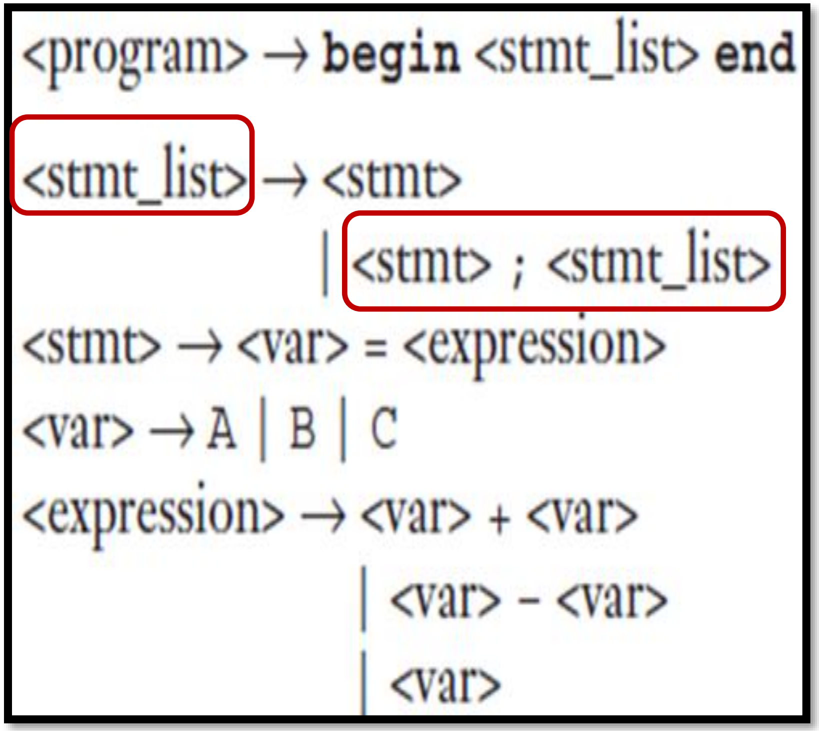
<program> → **begin** <stmt_list> **end**

<stmt_list> → <stmt>
  | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
  | <var> - <var>
  | <var>

begin B = C ; A = B + C end

<program> → **begin** <stmt_list> **end**

Example (3) (Cont.)



<program> → **begin** <stmt_list> **end**

<stmt_list> → <stmt>
| <stmt> ; <stmt_list>

<stmt> → <var> = <expression>

<var> → A | B | C

<expression> → <var> + <var>
| <var> - <var>
| <var>

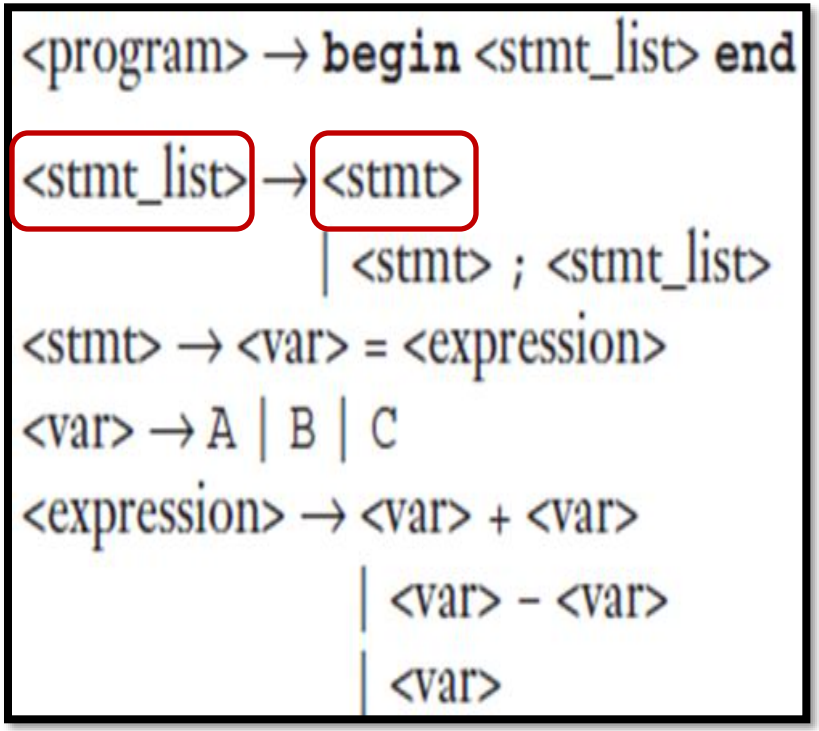
begin B = C ; A = B + C end

<program> → **begin** <stmt_list> **end**

<program> → **begin** <stmt_list> **end**

<program> → **begin** <stmt> ; <stmt_list> **end**

Example (3) (Cont.)



```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> - <var>
             | <var>
```

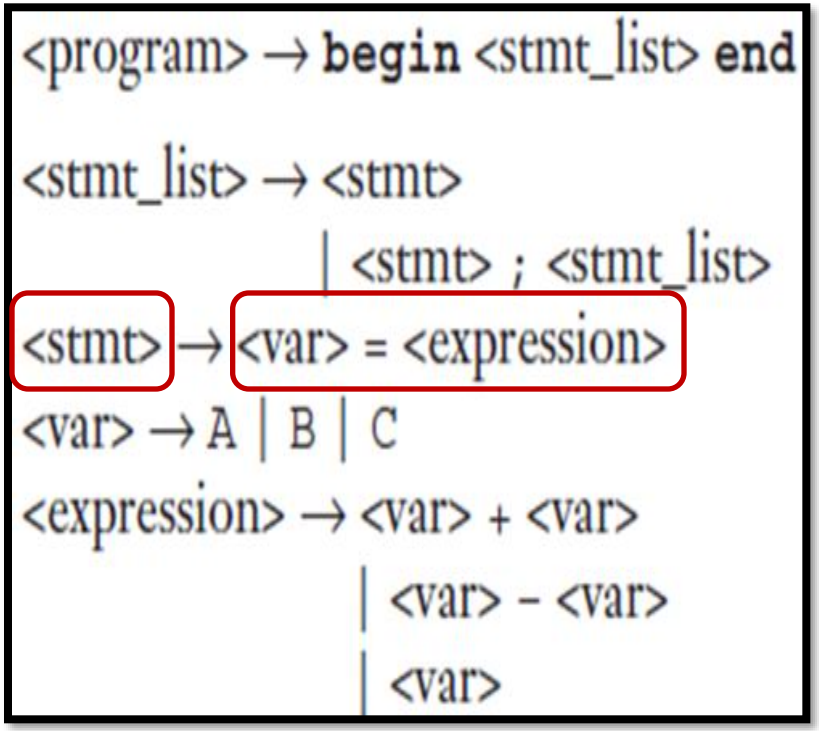
begin B = C ; A = B + C end

<program> → **begin** <stmt_list> **end**

<program> → **begin** <stmt> ; <stmt_list> **end**

<program> → **begin** <stmt> ; <stmt> **end**

Example (3) (Cont.)



<program> → **begin** <stmt_list> **end**

<stmt_list> → <stmt>
        | <stmt> ; <stmt_list>

<stmt> → <var> = <expression>

<var> → A | B | C

<expression> → <var> + <var>
        | <var> - <var>
        | <var>

begin B = C ; A = B + C end

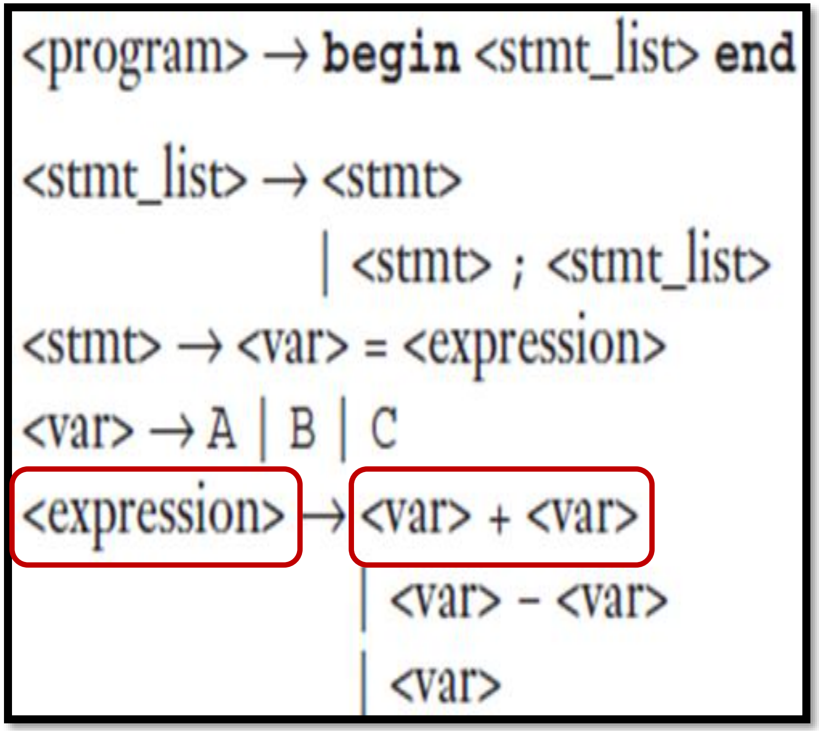
<program> → **begin** <stmt_list> **end**

<program> → **begin** <stmt> ; <stmt_list> **end**

<program> → **begin** <stmt> ; <stmt> **end**

<program> → **begin** <stmt> ; <var> = <expression> **end**

Example (3) (Cont.)



```
<program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
             | <var> - <var>
             | <var>
```

begin B = C ; A = B + C end

<program> → **begin** <stmt_list> **end**

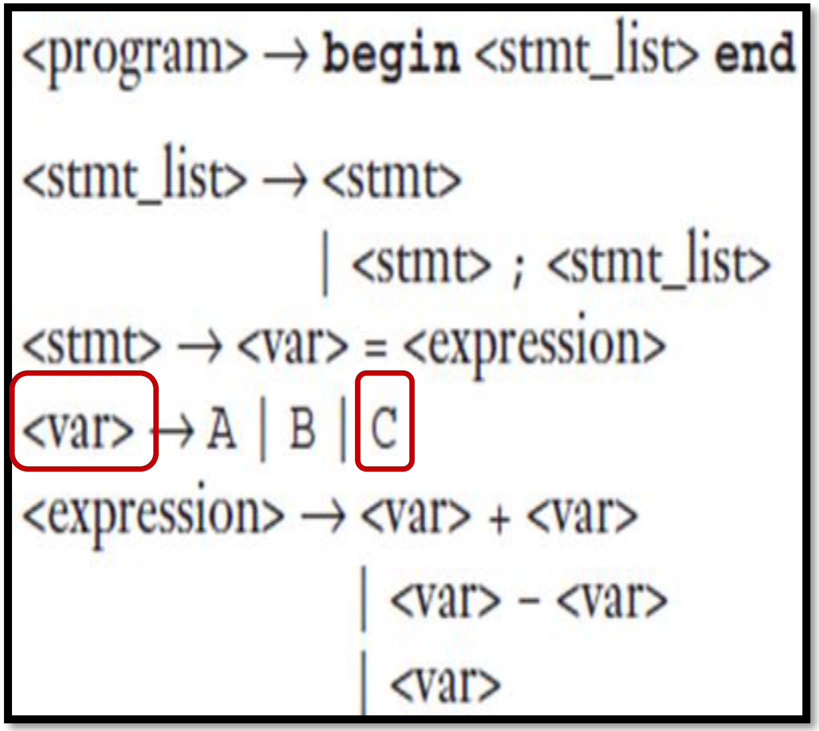
<program> → **begin** <stmt> ; <stmt_list> **end**

<program> → **begin** <stmt> ; <stmt> **end**

<program> → **begin** <stmt> ; <var> = <expression> **end**

<program> → **begin** <stmt> ; <var> = <var> + <var> **end**

Example (3) (Cont.)



<program> → **begin** <stmt_list> **end**

<stmt_list> → <stmt>
                     | <stmt> ; <stmt_list>

<stmt> → <var> = <expression>

<var> → A | B | C

<expression> → <var> + <var>
                     | <var> - <var>
                     | <var>

begin B = C ; A = B + C end

<program> → **begin** <stmt_list> **end**

<program> → **begin** <stmt> ; <stmt_list> **end**

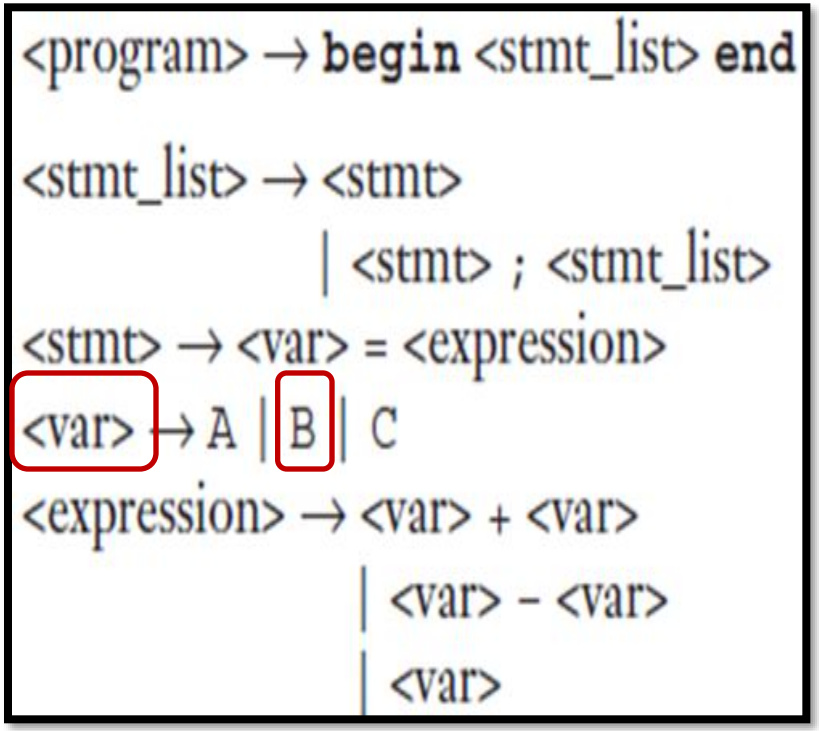
<program> → **begin** <stmt> ; <stmt> **end**

<program> → **begin** <stmt> ; <var> = <expression> **end**

<program> → **begin** <stmt> ; <var> = <var> + <var> **end**

<program> → **begin** <stmt> ; <var> = <var> + C **end**

Example (3) (Cont.)



<program> → **begin** <stmt_list> **end**

<stmt_list> → <stmt>
| <stmt> ; <stmt_list>

<stmt> → <var> = <expression>

<var> → A | B | C

<expression> → <var> + <var>
| <var> - <var>
| <var>

begin B = C ; A = B + C end

<program> → **begin** <stmt_list> **end**

<program> → **begin** <stmt> ; <stmt_list> **end**

<program> → **begin** <stmt> ; <stmt> **end**

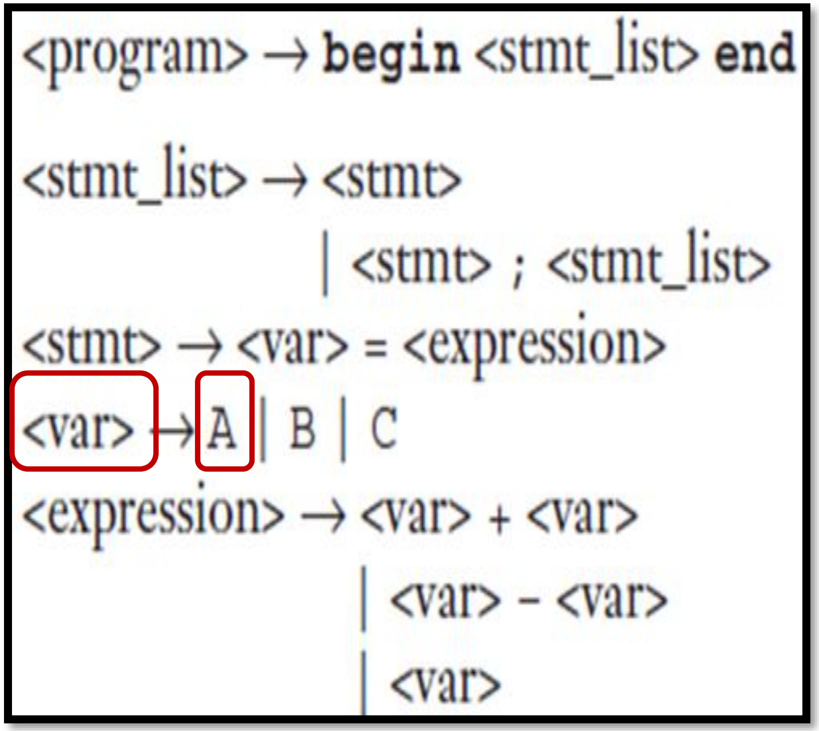
<program> → **begin** <stmt> ; <var> = <expression> **end**

<program> → **begin** <stmt> ; <var> = <var> + <var> **end**

<program> → **begin** <stmt> ; <var> = <var> + C **end**

<program> → **begin** <stmt> ; <var> = B + C **end**

Example (3) (Cont.)



<program> → **begin** <stmt_list> **end**

<stmt_list> → <stmt>
| <stmt> ; <stmt_list>

<stmt> → <var> = <expression>

<var> → A | B | C

<expression> → <var> + <var>
| <var> - <var>
| <var>

begin B = C ; A = B + C end

<program> → **begin** <stmt_list> **end**

<program> → **begin** <stmt> ; <stmt_list> **end**

<program> → **begin** <stmt> ; <stmt> **end**

<program> → **begin** <stmt> ; <var> = <expression> **end**

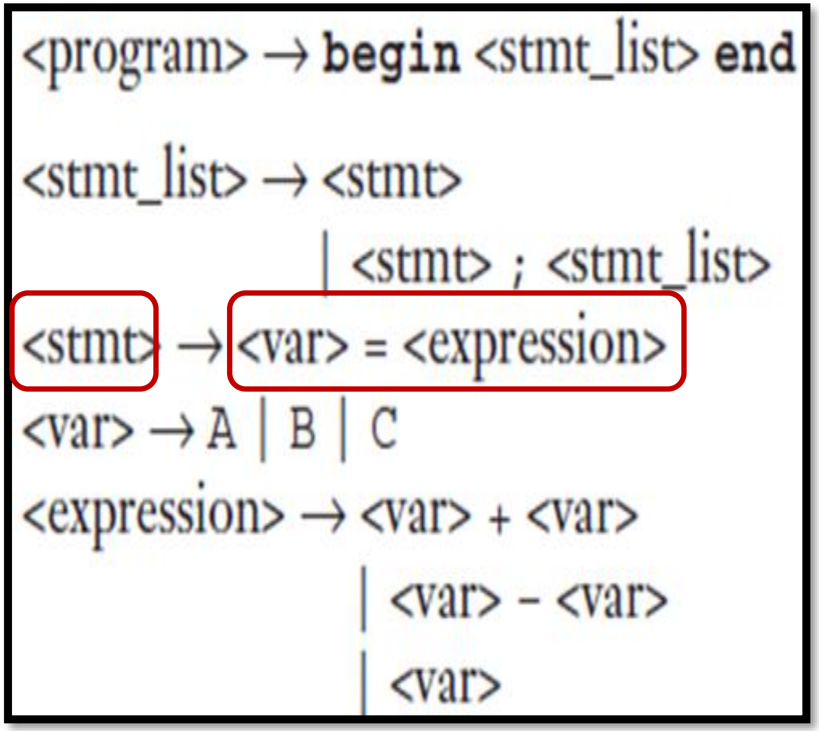
<program> → **begin** <stmt> ; <var> = <var> + <var> **end**

<program> → **begin** <stmt> ; <var> = <var> + C **end**

<program> → **begin** <stmt> ; <var> = B + C **end**

<program> → **begin** <stmt> ; A = B + C **end**

Example (3) (Cont.)



<program> → **begin** <stmt_list> **end**

<stmt_list> → <stmt>
| <stmt> ; <stmt_list>

<stmt> → <var> = <expression>

<var> → A | B | C

<expression> → <var> + <var>
| <var> - <var>
| <var>

begin B = C ; A = B + C end

<program> → **begin** <stmt_list> **end**

<program> → **begin** <stmt> ; <stmt_list> **end**

<program> → **begin** <stmt> ; <stmt> **end**

<program> → **begin** <stmt> ; <var> = <expression> **end**

<program> → **begin** <stmt> ; <var> = <var> + <var> **end**

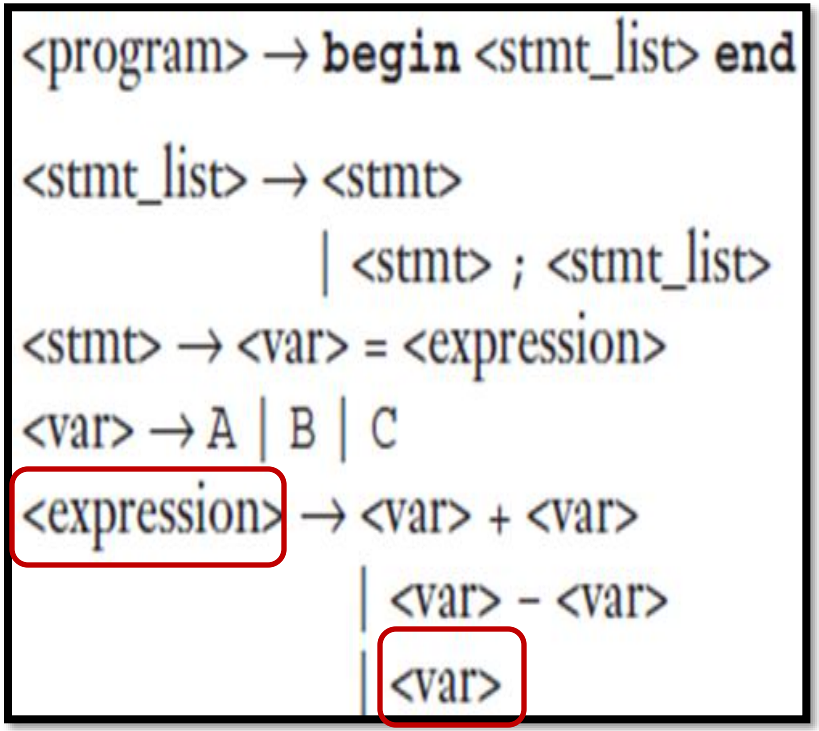
<program> → **begin** <stmt> ; <var> = <var> + C **end**

<program> → **begin** <stmt> ; <var> = B + C **end**

<program> → **begin** <stmt> ; A = B + C **end**

<program> → **begin** <var> = <expression> ; A = B + C **end**

Example (3) (Cont.)



<program> → **begin** <stmt_list> **end**

<stmt_list> → <stmt>
| <stmt> ; <stmt_list>

<stmt> → <var> = <expression>

<var> → A | B | C

<expression> → <var> + <var>
| <var> – <var>
| <var>

begin B = C ; A = B + C end

<program> → **begin** <stmt_list> **end**

<program> → **begin** <stmt> ; <stmt_list> **end**

<program> → **begin** <stmt> ; <stmt> **end**

<program> → **begin** <stmt> ; <var> = <expression> **end**

<program> → **begin** <stmt> ; <var> = <var> + <var> **end**

<program> → **begin** <stmt> ; <var> = <var> + C **end**

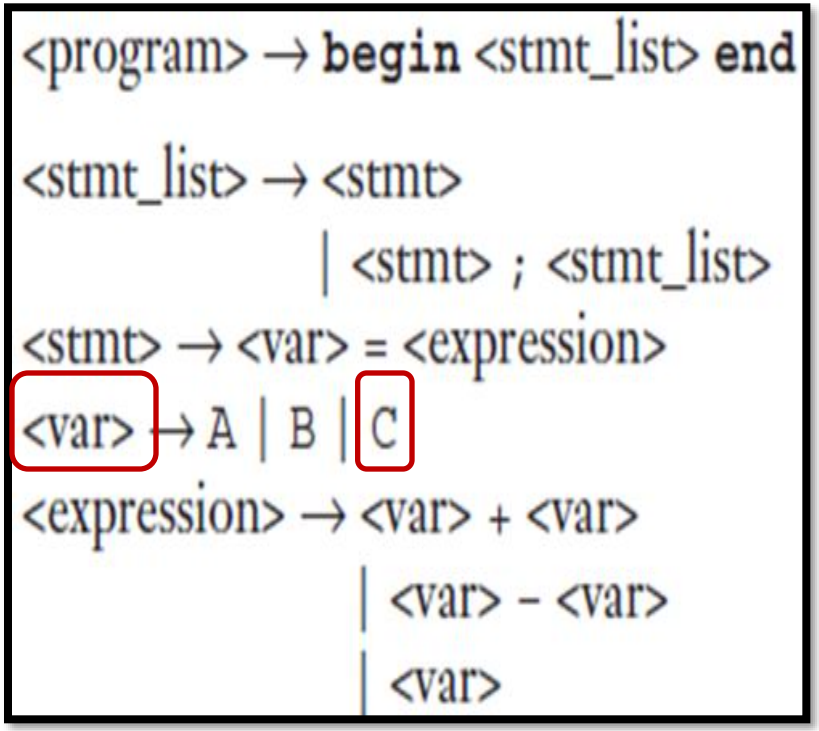
<program> → **begin** <stmt> ; <var> = B + C **end**

<program> → **begin** <stmt> ; A = B + C **end**

<program> → **begin** <var> = <expression> ; A = B + C **end**

<program> → **begin** <var> = < var > ; A = B + C **end**

Example (3) (Cont.)



<program> → **begin** <stmt_list> **end**

<stmt_list> → <stmt>
| <stmt> ; <stmt_list>

<stmt> → <var> = <expression>

<var> → A | B | C

<expression> → <var> + <var>
| <var> - <var>
| <var>

begin B = C ; A = B + C end

<program> → **begin** <stmt_list> **end**

<program> → **begin** <stmt> ; <stmt_list> **end**

<program> → **begin** <stmt> ; <stmt> **end**

<program> → **begin** <stmt> ; <var> = <expression> **end**

<program> → **begin** <stmt> ; <var> = <var> + <var> **end**

<program> → **begin** <stmt> ; <var> = <var> + C **end**

<program> → **begin** <stmt> ; <var> = B + C **end**

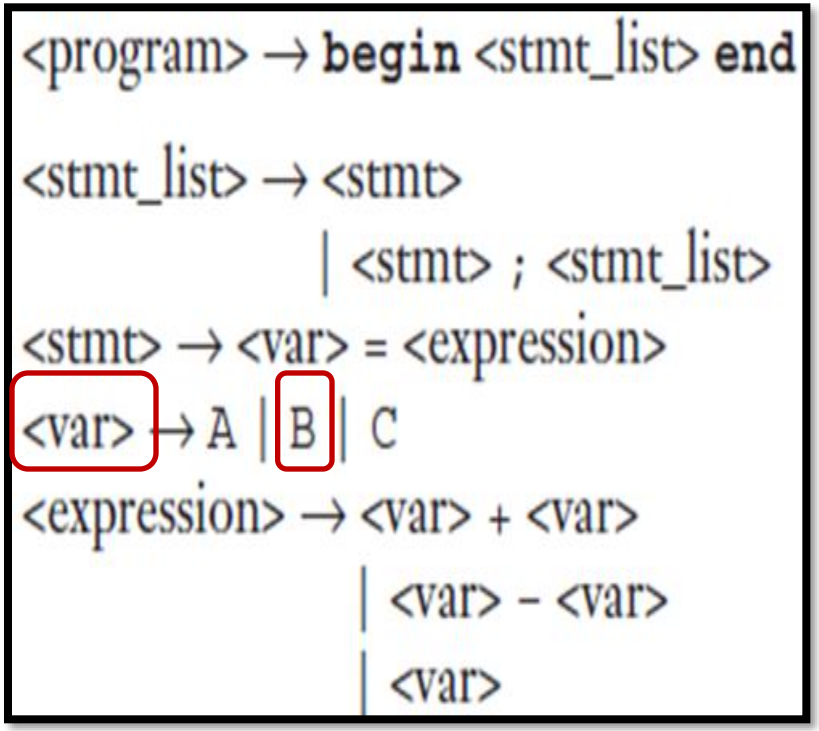
<program> → **begin** <stmt> ; A = B + C **end**

<program> → **begin** <var> = <expression> ; A = B + C **end**

<program> → **begin** <var> = < var > ; A = B + C **end**

<program> → **begin** <var> = C ; A = B + C **end**

Example (3) (Cont.)



<program> → **begin** <stmt_list> **end**

<stmt_list> → <stmt>
| <stmt> ; <stmt_list>

<stmt> → <var> = <expression>

<var> → A | B | C

<expression> → <var> + <var>
| <var> - <var>
| <var>

begin B = C ; A = B + C end

<program> → **begin** <stmt_list> **end**

<program> → **begin** <stmt> ; <stmt_list> **end**

<program> → **begin** <stmt> ; <stmt> **end**

<program> → **begin** <stmt> ; <var> = <expression> **end**

<program> → **begin** <stmt> ; <var> = <var> + <var> **end**

<program> → **begin** <stmt> ; <var> = <var> + C **end**

<program> → **begin** <stmt> ; <var> = B + C **end**

<program> → **begin** <stmt> ; A = B + C **end**

<program> → **begin** <var> = <expression> ; A = B + C **end**

<program> → **begin** <var> = < var > ; A = B + C **end**

<program> → **begin** <var> = C ; A = B + C **end**

<program> → **begin** B = C ; A = B + C **end**

ANY QUESTIONS

Thank You!