MÔ TẢ NGÔN NGỮ SIMPLECODE

SimpleCode là một ngôn ngữ mệnh lệnh đơn giản tương tự với ngôn ngữ C.

Từ vựng

Các từ khoá và định danh phân biệt hoa thường. Chẳng hạn, if là từ khoá nhưng IF có thể đặt tên cho biến, foo và Foo là 2 tên khác nhau tham chiếu đến 2 biến khác nhau. Tất cả các từ khoá đều viết thường.

Các từ khoá bao gồm: **boolean, break, callout, class, continue, else, false, for, if, int, return, true, void**

Program không phải là từ khoá, nhưng là một định danh có ý nghĩa đặc biệt trong ngữ cảnh cố định

Các ghi chú (**comment**) bắt đầu bởi dấu '//' và kết thúc bằng kí tự xuống dòng. Hoặc ghi chú có thể nằm giữa '/*' và '*/'

Khoảng trắng có thể xuất hiện giữa các token từ vựng. Khoảng trắng được định nghĩa là một hoặc nhiều khoảng trắng, tab, kí tự xuống dòng hoặc ghi chú.

Từ khoá và định danh phải phân cách nhau bởi khoảng trắng, hoặc một token mà không phải là từ khoá hoặc định danh. Chẳng hạn, *thisfortrue* là một định danh, không phải là 3 từ khoá riêng biệt. Nếu một dãy bắt đầu bằng một chữ cái hoặc dấu gạch dưới thì nó và chuỗi dài nhất các kí tự theo sau tạo thành một token.

Kiểu chuỗi kí tự (**string**) bao gồm các kí tự đặt trong dấu nháy đôi. Trong khi kiểu kí tự (**char**) thì chỉ gồm một kí tự đặt trong dấu nháy đơn.

Kiểu số có độ dài 32-bit, có dấu, nghĩa là giá trị thập phân nằm trong đoạn $[-2^{31}, 2^{31} - 1]$. Nếu một chuỗi bắt đầu với 0x thì 2 kí tự này cùng với chuỗi dài nhất các kí tự thuộc tập [0-9 a-f A-F] tạo thành số nguyên ở hệ thập lục . Nếu chuỗi bắt đầu với chữ số thập phân (không phải là 0x) và chuỗi dài nhất các chữ số thập phân theo sau tạo thành số nguyên thập phân. Chú ý, việc kiểm tra miền giá trị được thực hiện sau. Một chuỗi dài các chữ số, ví dụ: 123456789123456789, vẫn được xem là một token đơn.

Một kí tự (**char**) là một kí tự ASCII có thể in được (giá trị mã nằm trong đoạn [32, 126] (ở hệ 8 tương ứng là 40 và 176). Các kí tự đặc biệt, gồm: nháy đôi (“), nháy đơn (‘) hoặc dấu gạch chéo (\) thì được viết thành chuỗi gồm 2 kí tự bắt đầu bằng dấu gạch chéo, cụ thể: nháy đôi ‘\””, nháy đơn ‘\’’, gạch chéo ‘\\’, tab ‘\t’, xuống dòng ‘\n’.

Văn phạm tham chiếu

Các qui ước trong mô tả văn phạm bên dưới

$< foo >$	foo là một non-terminal
$\boldsymbol{foo}$	foo (in hoa) là một terminal
$[x]$	x không xuất hiện hoặc xuất hiện 1 lần. Lưu ý, dấu ngoặc đặt trong dấu nháy đơn, ‘[‘ ‘]’ là các terminal.
x^*	x không xuất hiện hoặc xuất hiện một hoặc nhiều lần.
x^+,	Danh sách một hoặc nhiều x, phân tách nhau bởi dấu phẩy.
{ }	Dùng để nhóm các thành phần. Lưu ý rằng, các dấu ngoặc được đặt trong dấu nháy đơn thì là các terminal.
|	Phân tách các lựa chọn

<program> → **class Program** ‘{‘ <field_decl>* <method_decl>* ‘}’

<field_decl> → <type> {<id>| <id> ‘[‘ <int_literal> ‘]’ }+, ;

<method_decl> → {<type> | **void** } <id> ([{<type> <id>}+,]) <block>

<block> → ‘{‘ <var_decl>* <statement>* ‘}’

<var_decl> → <type> <id>+, ;

<type> → **int** | **boolean**

<statement> → <location> <assign_op> <expr> ;

| <method_call> ;

| **if** (<expr>) <block> [**else** <block>]

| **for** <id> = <expr> , <expr> <block>

| **return** [<expr>] ;

	|	**break** ;
	|	**continue** ;
	|	<block>
<assign_op>	→	=
	|	+=
	|	-=
<method_call>	→	<method_nam> ([<expr>+,])
	|	**callout** (<string_literal> [, <callout_arg>+,])
<method_name>	→	<id>
<location>	→	<id>
	|	<id> '[' <expr> ']'
<expr>	→	<location>
	|	<method_call>
	|	<literal>
	|	<expr> <bin_op> <expr>
	|	- <expr>
	|	! <expr>
	|	(<expr>)
<callout_arg>	→	<expr> | <string_literal>
<bin_op>	→	<arith_op> | <rel_op> | <eq_op> | <cond_op>
<arith_op>	→	+ | - | * | / | %
<rel_op>	→	< | > | <= | >=
<eq_op>	→	== | !=
<cond_op>	→	&& | ||
<literal>	→	<int_literal> | <char_literal> | <bool_literal>
<id>	→	<alpha> <alpha_num>*
<alpha_num>	→	<alpha> | <digit>
<alpha>	→	a | b | … | z | A | B | … | Z | _
<digit>	→	0 | 1 | 2 | … | 9

<hex_digit> → <digit> | a | b | c | d | e | f | A | B | C | D | E | F

<int_literal> → <decimal_literal> | <hex_literal>

<decimal_literal> → <digit> <digit>*

<hex_literal> → 0x <hex_digit> <hex_digit>*

<bool_literal> → **true | false**

<char_literal> → ‘ <char> ‘

<string_literal> → “<char>*”

Ngữ nghĩa

Một chương trình SimpleCode bao gồm một khai báo lớp với tên lớp là **Program.** Khai báo lớp này bao gồm các khai báo *trường dữ liệu* và *phương thức*. Khai báo trường dữ liệu cho biết các biến này có thể được truy cập ở mức toàn cục bởi tất cả các phương thức bên trong chương trình. Khai báo phương thức cho biết các hàm hoặc thủ tục. Chương trình phải chứa một khai báo cho một phương thức có tên là **main** và không có tham số. Thực thi chương trình SimpleCode bắt đầu từ phương thức **main** này.

Các kiểu dữ liệu

Gồm 2 kiểu cơ bản là **int** và **boolean.** Ngoài ra, có mảng các số nguyên (int [N]) hoặc mảng các phần tử boolean (boolean [N]).

Mảng chỉ có thể được khai báo trong phạm vi toàn cục (trong khai báo lớp). Tất cả các mảng chỉ có một chiều và kích thước được xác định tại thời điểm biên dịch (mảng tĩnh). Chỉ số của mảng có N phần tử bắt đầu từ 0 và kết thúc ở N – 1. Dấu ngoặc vuông được sử dụng để truy xuất đến từng phần tử trong mảng. Vì mảng có kích thước cố định và không thể khai báo như tham số (hoặc biến cục bộ), nên không có chức năng hỗ trợ lấy chiều dài của các biến mảng.

Các qui ước về phạm vi

Tất cả các định danh phải được định nghĩa trước khi sử dụng. chẳng hạn:

- Một biến phải được khai báo trước khi sử dụng

- Phương thức chỉ được gọi phía sau phần header khai báo của phương thức đó. Lưu ý các phương thức gọi đệ qui thì được cho phép)

Có ít nhất hai phạm vi hợp lệ trong một chương trình SimpleCode: **toàn cục** và **phạm vi phương thức**. Phạm vi toàn cục bao gồm các định danh của trường dữ liệu và phương thức được khai báo trong phần khai báo lớp **Program.** Phạm vi phương thức báo gồm các định danh của biến và các tham số được định nghĩa trong phần khai báo của phương thức.

Ngoài ra còn có phạm vi cục bộ, tương ứng với mỗi <block> trong chương trình, thường xuất hiện sau câu lệnh **if** hoặc **for** hoặc có thể được chèn bất kì chỗ nào mà một câu lệnh được cho phép.

Một định danh có phạm vi phương thức có thể che định danh có cùng tên ở phạm vi toàn cục. Tương tự, những định danh có phạm vi cục bộ có thể che các định danh cùng tên ở phạm vi lồng có độ sâu thấp hơn, phạm vi phương thức và phạm vi toàn cục.

Tên biến được định nghĩa bên trong phạm vi phương thức hoặc phạm vi cục bộ có thể che tên các phương thức ở phạm vi toàn cục. Trong trường hợp này, định danh có thể được dùng đặt tên cho biến cho đến khi rời khỏi phạm vi của phương thức.

Không định danh nào có thể được định nghĩa nhiều hơn một lần với cùng một phạm vi. Do đó, tên trường dữ liệu và phương thức, tất cả phải phân biệt ở mức toàn cục; tên của các biến cục bộ và các tham số phương thức, tất cả phải phân biệt ở mỗi phạm vi cục bộ.

Vị trí (locations)

Bao gồm 2 loại: các biến lưu một giá trị đơn cục bộ / toàn cục và các phần tử của mảng (mức toàn cục).

Các vị trí có kiểu **int** và **boolean** lần lượt chứa các giá trị số nguyên hoặc boolean. Vì các mảng có kích thước tĩnh trong SimpleCode nên chúng được cấp phát trong không gian dữ liệu tĩnh của một chương trình và không cần phải cấp phát trong heap.

Mỗi vị trí được khởi tạo một giá trị mặc định khi nó được khai báo. Các số nguyên có giá trị mặc định là 0 và boolean có giá trị mặc định là false. Các biến cục bộ phải được

khởi tạo khi phạm vi được xác định. Các phần tử của mảng được khởi tạo khi chương trình bắt đầu.

Phép gán

Phép gán chỉ cho phép đối với các giá trị đơn. Đối với kiểu **int** và **boolean,** SimpleCode sử dụng ngữ nghĩa sao chép giá trị, cụ thể:

- Phép gán <location> = <expr> sao chép giá trị kết quả của biểu thức <expr> vào <location>.
- Phép gán <location> += <expr> tăng giá trị được lưu trong <location> một giá trị bằng giá trị của <expr>, lưu ý rằng phép gán này chỉ hợp lệ khi cả <location> và <expr> đều có kiểu là **int**.
- Phép gán <location> -= <expr> giảm giá trị được lưu trong <location> một giá trị bằng giá trị của <expr> và cũng chỉ hợp lệ khi cả <location> và <expr> đều có kiểu là **int**.

<location> và <expr> trong một phép gán phải có cùng kiểu. Đối với kiểu mảng thì <location> và <expr> phải tham chiếu đến một phần tử trong mảng (có giá trị đơn)

Việc gán giá trị cho tham số bên trong phương thức là hợp lệ. Lưu ý rằng phép gán này chỉ có phạm vi cục bộ.

Gọi phương thức và trả về kết quả

Việc gọi phương thức liên quan đến:

(1) truyền giá trị tham số từ chỗ gọi vào phương thức được gọi.

(2) Thực hiện mã trong thân phương thức được gọi

(3) Trả về giá trị của phương thức

Việc truyền tham số được định nghĩa theo phép gán: Các tham số của phương thức được xem như là các biến cục bộ bên trong phương thức và được khởi tạo bằng phép gán bằng giá trị được ước lượng từ biểu thức tham số. Các tham số được ước lượng từ trái sang phải.

Thân phương thức được gọi tiếp theo sẽ thực thi bằng cách gọi thực thi các câu lệnh bên trong phương thức một cách tuần tự.

Một phương thức không có khai báo kiểu trả về chỉ có thể được gọi như một câu lệnh, nghĩa là không được sử dụng trong các biểu thức. Phương thức dạng này trả điều khiển lại cho nơi gọi nó bằng cách gọi lệnh **return** hoặc sau khi thực hiện hết các câu lệnh trong phương thúc.

Một phương thức trả về giá trị thì có thể được sử dụng trong các biểu thức. Kết quả trả về của phương thức là kết quả ước lượng biểu thức trong câu lệnh **return.** Là không hợp lệ nếu phương thức dạng này không trả về kết quả.

Một phương thức trả về kết quả cũng có thể được gọi như một câu lệnh và kết quả trả về của nó được lờ đi.

Các câu lệnh điều khiển

Câu lệnh if

Câu lệnh if có ngữ nghĩa như sau:

- Đầu tiên, <expr> được ước lượng.
- Nếu kết quả ước lượng là **true**, thì nhánh lệnh tương ứng **true** thực thi. Ngược lại, nhánh **else** được thực thi (nếu có).

Lưu ý rằng, SimpleCode yêu cầu các nhánh **true** và **else** đều phải được đặt giới hạn trong dấu ngoặc nhọn nên sẽ không có nhập nhằng khi xác định câu lệnh nào thuộc về nhánh nào.

Câu lệnh for

<id> là biến chỉ số cho vòng lặp và nó có thể che các biến có cùng tên được khai báo bên ngoài phạm vi của lệnh **for** (nếu có). Biến chỉ số cho vòng lặp là một biến kiểu số nguyên và phạm vi của nó giới hạn trong thân vòng lặp.

<expr> đầu tiên là giá trị khởi tạo cho biến chỉ số vòng lặp và <expr> thứ hai là giá trị kết thúc của bên chỉ số vòng lặp. Mỗi biểu thức chỉ được ước lượng một lần, trước khi đi vào vòng lặp lần đầu tiên. Mỗi biểu thức phải ước lượng thành giá trị số nguyên.

Thân vòng lặp được thực thi nếu giá trị hiện thời của chỉ số vòng lặp nhỏ hơn giá trị kết thúc tương ứng. Sau một thực thi của vòng lặp, biến chỉ số được tăng lên 1 và giá trị mới được so sánh với giá trị kết thúc để quyết định có thực thi vòng lặp lần nữa không.

Biểu thức

Các biểu thức tuân theo các luật ước lượng thông thường. Các toán tử có cùng độ ưu tiên sẽ được ước lượng theo thứ tự từ phải trái sang phải. Dấu ngoặc đơn có thể được sử dụng để ghi đè thứ tự ưu tiên thông thường.

Biểu thức <location> được ước lượng bằng giá trị được lưu tại vị trí đó.

Các biểu thức gọi phương thức tuân theo các qui ước trong phần "Gọi phương thức và trả về kết quả". Các thao tác trên mảng được ghi rõ trong phần "Các kiểu dữ liệu". Các biểu thức liên quan đến nhập/xuất được đề cập trong phần "Gọi thư viện"

Các số nguyên được ước lượng bằng giá trị nguyên của chúng. Các kí tự được ước lượng bằng giá trị mã ASCII, ví dụ: 'A' có giá trị là 65.

Các toán tử số học (<arith_op> và phép trừ đảo dấu) có độ ưu tiên và ý nghĩa như thông thường, tương tự cho các toán tử quan hệ (<rel_op>). Toán tử '%' tính số dư của một phép chia các toán hạng.

Các toán tử quan hệ được sử dụng để so sánh các biểu thức số nguyên. Các toán tử bằng,== và != chỉ được định nghĩa cho các kiểu **int** và **boolean,** có thể được sử dụng để so sánh hai biểu thức bất kì có cùng kiểu (== nghĩa là "bằng nhau" và != nghĩa là "không bằng nhau")

Kết quả ước lượng của toán tử quan hệ và toán tử bằng nhau là có kiểu là **boolean.**

Các toán tử liên kết logic && và || được thông dịch tương tự như ngôn ngữ C. Toán hạng thứ 2 không cần thực thi (ước lượng) nếu kết quả của toán hạng thứ nhất xác định luôn giá trị của cả biểu thức (nghĩa là nếu kết quả là false cho trường hợp && và kết quả là true cho ||)

Dưới đây là độ ưu tiên các toán tử (từ cao nhất đến thấp nhất):

Toán tử	Ghi
-	Trừ (đổi dấu)
!	Phủ định
*, / , %	Nhân, chia , lấy dư
+, -	Cộng, trừ
<, <=, >, >=	Quan hệ
==, !=	So sánh bằng
&&	Điều kiện và
||	Điều kiện hoặc

Lưu ý rằng độ ưu tiên không được phản ánh trong văn phạm tham chiếu.

Gọi thư viện

SimpleCode bao gồm một phương thức cơ sở (main) để gọi hàm được cung cấp bởi hệ thống run-time, ví dụ thư viện chuẩn của C và các hàm người dùng định nghĩa.

Phương pháp gọi hàm như sau:

int callout (<string_literal>, [<callout_arg>+,])

Hàm có tên là <string_literal> được gọi và các tham số kèm theo sẽ được truyền vào phía sau. Biểu thức kiểu boolean hoặc số nguyên sẽ được truyền vào dưới dạng số nguyên, chuỗi và biểu thức mảng được truyền như một con trỏ. Giá trị trả về của hàm sẽ được truyền trở ra như một số nguyên.

Người sử dụng callout có trách nhiệm đảm bảo các tham số truyền vào phải khớp với khai báo hàm và chỉ trả về giá trị nếu hàm thư viện bên dưới thực sự trả về một gía trị của kiểu thích hợp. Các tham số được truyền vào hàm tuần theo các qui định gọi hàm chuẩn của hệ thống.

Ngoài việc truy xuất thư viện chuẩn C sử dụng callout, một hàm nhập/xuất có thể được viết bằng C (hay bất kì ngôn ngữ khác) đươc biên dịch sử dụng các công cụ chuẩn, liên kết với hệ thống runtime và được truy cập theo cơ chế **callout**.

Các qui tắc ngữ nghĩa

Những luật này này đưa thêm các ràng buộc cho tính hợp lệ của chương trình SimpleCode bên cạnh các ràng buộc liên quan đến văn phạm. Một chương trình tuân theo văn phạm và không vi phạm một trong các luật sau đây được gọi là hợp lệ. Trình biên dịch mạnh sẽ kiểm tra một cách tường minh các luật này và phát sinh các thông điệp lỗi mô tả từng vi phạm có thể tìm thấy. Trình biên dịch sẽ phát sinh ít nhất một thông điệp lỗi cho một chương trình lỗi, nhưng không phát sinh lỗi nào cho chương trình hợp lệ.

1. Không định danh nào được khai báo 2 lần với cùng một phạm vi
2. Không định danh nào được sử dụng trước khi nó được khai báo
3. Chương trình chứa một định nghĩa cho phương thức **main** không có tham số (chú ý rằng các phương thức nào định nghĩa sau **main** sẽ không bao giờ được thực thi)
4. <int_literal> trong một khai báo mảng lớn hơn 0
5. Số lượng tham số và kiểu tương ứng của chúng khi gọi phương thức phải trùng khớp với số lượng và kiểu của các tham số trong phần định nghĩa của phương thức đó.
6. Nếu phương thức được sử dụng như một biểu thức, phương thức đó phải trả về một giá trị.
7. Câu lệnh **return** không phải có giá trị trả về trừ khi nó xuất hiện trong thân của phương thức được khai báo là trả về một giá trị
8. Biểu thức trong câu lệnh **return** phải có cùng kiểu với kiểu kết quả trả về được khai báo trong định nghĩa của phương thức.
9. <id> được sử dụng như một <location> phải là một biến toàn cục hoặc cục bộ hoặc tham số phương thức.
10. Đối với với tất cả các <location> có dạng <id> [<expr>]
    a) <id> phải là một biến mảng , và
    b) Kiểu của <expr> phải là **int.**
11. <expr> trong một câu lệnh **if** phải có kiểu **boolean**
12. Toán hạng của <arith_op> và <rel_op> phải có kiểu **int**
13. Toán hạng của <eq_op> phải có cùng kiểu hoặc là **int** hoặc là **boolean**

14. Toán hạng của <cond_op> và toán hạn của phép phủ định (!) phải có kiểu là **boolean.**
15. <location> và <expr> trong một phép gán, <location> = <expr> phải có cùng kiểu
16. <location> và <expr> trong phép gán tăng/giảm, <location> += <expr> và <location> -= <expr> phải có kiểu là **int**
17. <expr> khởi tạo và kết thúc trong câu lệnh **for** phải có kiểu là số nguyên.
18. Tất cả các câu lệnh **break** và **continue** phải nằm trong phần thân của **for**.

Kiểm tra run-time

Bên cạnh các ràng buộc được mô tả ở trên là bắt buộc(kiểm tra ngữ nghĩa tĩnh bởi trình biên dịch) những ràng buộc sau đây cần được kiểm tra động: bộ phát sinh code của trình biên dịch phải sinh ra code để thực hiện các kiểm tra này, các vi phạm được phát hiện ở thời điểm run-time.

- Chỉ số các phần tử trong mảng phải nằm trong giới hạn khai báo
- Đối với phương thức có giá trị trả về thì control không được rơi vào cuối của hàm (nghĩa là không có trường hợp thoát khỏi phương thức mà không qua lệnh return).

Khi lỗi run-time xảy ra, một thông điệp lỗi thích hợp cần được in ra màn hình nhằm giúp người lập trình tìm ra lỗi trong chương trình nguồn.