Описание языка L_4

С.Ю. Скоробогатов

1 сентября 2009

1 Введение

Язык L_4 представляет собой игрушечный императивный язык программирования, предназначенный для проведения лабораторных работ по курсу «Конструирование компиляторов».

2 Лексика

В языке L_4 мы будем различать пять типов лексем: идентификаторы (2.2), ключевые слова (2.3), константы (2.4), знаки операций и другие разделители (2.5). Компилятор языка L_4 игнорирует комментарии и расположенные между лексемами так называемые *невидимые символы*, а именно: пробелы, знаки табуляции и перевода строки. При этом требуется, чтобы смежные идентификаторы, ключевые слова и константы разделялись невидимыми символами или комментариями.

При разбиении входного потока на лексемы должно учитываться *правило самой длинной лексемы*, а именно: если входной поток был разбит на лексемы вплоть до некоторой позиции, то начиная с этой позиции из входного потока выбирается самая длинная последовательность символов, которая может составлять лексему.

2.1 Комментарии

В языке L_4 используются многострочные вложенные комментарии, которые записываются внутри фигурных скобок. Например:

```
{ это комментарий {
в него вложен ещё один
комментарий } }
```

2.2 Идентификаторы

Идентификатор представляет собой последовательность букв и цифр, предваряемую одним из следующих символов:

```
_   !   @   .   #
```

Реализации языка L_4, поддерживающие Unicode, должны разрешать использование в идентификаторах букв любых алфавитов, охваченных стандартом Unicode.

2.3 Ключевые слова

Ниже приведён список используемых в программах на языке L_4 ключевых слов:

and	**_ne_**	**int**
_eq_	**_or_**	**new_**
_ge_	**_pow_**	**not_**
_gt_	**_xor_**	**nothing**
_le_	**bool**	**true**
_lt_	**char**	
mod	**false**	

2.4 Константы

Константы в языке L_4 представляют собой изображения значений, известных на момент компиляции, и бывают нескольких типов.

2.4.1 Целочисленные константы

Целочисленные константы в языке L_4 изображают неотрицательные целые числа произвольной разрядности. Они записываются в виде последовательностей цифр, после которых в фигурных скобках могут указываться основания систем счисления.

Код	Аббревиатура	Код	Аббревиатура	Код	Аббревиатура	Код	Аббревиатура
0	NUL	8	BS	16	DLE	24	CAN
1	SOH	9	TAB	17	DC1	25	EM
2	STX	10	LF	18	DC2	26	SUB
3	ETX	11	VT	19	DC3	27	ESC
4	EOT	12	FF	20	DC4	28	FS
5	ENQ	13	CR	21	NAK	29	GS
6	ACK	14	SO	22	SYN	30	RS
7	BEL	15	SI	23	ETB	31	US

Таблица 1: Аббревиатуры управляющих символов в символьных и строковых константах.

Цифрами считаются арабские цифры от 0 до 9, а также латинские буквы. При этом A обозначает 10, B обозначает 11, и т.д.

Основание системы счисления записывается в виде десятичного числа в диапазоне от 2 до 36. Если основание не указано, то целочисленная константа считается десятичной.

Целочисленная константа не может содержать пробелов, знаков табуляции и перевода строки.

Примеры целочисленных констант:

```
120000000000    7FFFF{16}
100111011{2}    gh10fj{20}
1202211{03}     0
```

2.4.2 Символьные константы

Символьные константы изображают символы Unicode или ASCII в зависимости от того, поддерживает компилятор языка L_4 стандарт Unicode или не поддерживает.

Символы, которые не являются управляющими, записываются в кавычках, причём для изображения символа «кавычка» используется пара кавычек:

```
"Q"   "я"   "1"   "&"   ""
```

Некоторые управляющие символы ASCII записываются с помощью аббревиатур символов (см. таблицу 1), обрамлённых знаками доллара. Кроме того, любой символ с кодом x может быть представлен как $\$x_{10}\$$, где x_{10} – десятичная запись числа x. Примеры:

```
$CR$   { перевод каретки }
$13$   { тоже перевод каретки }
```

2.4.3 Строковые константы

Строковая константа в языке L_4 представляет собой последовательность так называемых *строковых секций*, каждая из которых описывает цепочку символов Unicode или ASCII. Строковые секции в программе должны быть разделены невидимыми символами или комментариями.

Основная форма строковой секции выглядит как последовательность символов, заключённая в апострофы. При этом в эту последовательность не могут входить управляющие символы и апострофы.

Для представления апострофов используется специальная строковая секция, изображаемая как %AP%.

Каждый управляющий символ ASCII, входящий в строку, оформляется в виде отдельной строковой секции, которая выглядит как аббревиатура символа (см. таблицу 1), обрамлённая знаками процента. Кроме того, любой символ с кодом x может быть представлен как отдельная секция $\%x_{10}\%$, где x_{10} – десятичная запись числа x.

Например, строковая константа

```
'We say' %AP% 'Hello, World!'
    %AP% %LF%
```

в языке C могла бы быть записана как:

```
"We say 'Hello, World!'\n"
```

2.4.4 Булевские константы

Для представления «истины» используется константа **true**, а для представления «лжи» – константа **false**.

2.4.5 Ссылочная константа nothing

Ключевое слово **nothing** обозначает нулевую ссылку.

2.5 Знаки операций и другие разделители

В языке L_4 используются следующие знаки операций и разделители:

```
<    >    (    )    [    ]
,    :    ?    &    \    ^
+    −    *    /
%    %%   :=   +++
```

3 Синтаксис и семантика

Программа на языке L_4 представляет собой набор *определений функций* (3.1). Порядок определений не важен, то есть из функции A может быть вызвана функция B, даже если определение B расположено в тексте программы после определения A.

3.1 Определения функций

Определение функции состоит из *заголовка* и *тела*.

Заголовок функции, возвращающей значение, записывается следующим образом:

```
(тип [имя параметры])
```

Здесь тип – это тип возвращаемого функцией значения.

Если функция не возвращает значения, её заголовок имеет вид

```
[имя параметры]
```

Список формальных параметров представляет собой последовательность *объявлений параметров*. Каждый параметр объявляется следующим образом:

```
(тип имя)
```

Например, если некоторая функция F не возвращает значения и имеет пять параметров !a, !b, @c, #d и _e, и при этом !a и !b имеют тип **int**, @c имеет тип **char**, а #d и _e имеют тип <**int**>, то заголовок этой функции выглядит как

```
[F
  (int !a) (int !b)
  (char @c) (<int> #d)
  (<int> _e)
]
```

Тело функции непосредственно следует за заголовком. Оно представляет собой последовательность *операторов* (3.4), разделённых запятыми, и завершается знаком %%.

Пример определения функции:

```
(<int>
  [SumVectors
    (<int> !A) (<int> !B)
  ]
)
  (int #size) := [length !A],
  \ #size _eq_ [length !B],
  (<int> #C) := new_ <int> #size ,
  (<int> #i : 0, #size−1)
    <#C #i> := <!A #i> + <!B #i>
  %,
  ^ C
%%
```

Главная функция, на которую передаётся управление при запуске программы, всегда называется Main, получает массив строк в качестве параметра и возвращает целое число:

```
(int [Main (<<char>> !args)])
  { какие−то действия }
  ^ 0
%%
```

3.2 Типы данных

Язык L_4 является языком со строгой преимущественно статической проверкой. Это означает, что во время компиляции известен тип каждой локальной переменной, тип каждого параметра функции, а также типы возвращаемых функциями значений.

В языке определены три примитивных типа **int**, **char** и **bool**, обозначающие целые числа, символы Unicode (или ASCII) и булевские значения, соответственно. Значения примитивных типов могут располагаться во фреймах функций и в элементах массивов.

Переменным типа **int** можно присваивать значения типа **int** или **char**. Переменным

типа **char** можно присваивать только значения типа **char**, а переменным типа **bool** можно присваивать только значения типа **bool**.

Кроме примитивных типов в программах на языке L_4 можно использовать ссылочные типы-массивы. Значения этих типов могут располагаться во фреймах функций и в элементах массивов. Они являются указателями на массивы, расположенные куче. Память под массивы выделяется с помощью операции **new_** и автоматически освобождается сборщиком мусора.

Тип-массив задаётся с помощью пары угловых скобок < >, внутри которых помещается изображение типа элементов массива.

Например, тип-массив с целыми элементами записывается как <**int**>, а тип-массив, элементами которого являются ссылки на массивы символов, записывается как <<**char**>>.

Отметим, что переменные типа <**char**> играют роль строк, и им можно присваивать строковые константы.

Переменной типа <T>, где T – некоторый тип, можно присваивать либо ссылки на массивы, элементы которых имеют тип T, либо константу **nothing**.

3.3 Выражения

Выражения в языке L_4 формируются из символов операций, констант, имён переменных, параметров и функций, изображений типов и круглых скобок.

Все операции перечислены в таблице 2. Во второй колонке таблицы метапеременные x и y обозначают подвыражения, метапеременная f обозначает имя функции, а метапеременная T – изображение типа. Порядок выполнения операций может быть изменён с помощью круглых скобок.

В операции вызова функции $[f\ x_1\ \ldots\ x_n]$ типы фактических параметров должны совпадать с типами формальных параметров, заданными при объявлении функции. Кроме того, функция f должна возвращать значение.

В операции выделения памяти **new_** $T\ x$ размер x должен иметь тип **int**.

Операции сравнения **_eq_** и **_ne_** позволяют сравнивать ссылки на массивы с константой **nothing**.

Для остальных операций в таблице 3 приведены зависимости типов возвращаемых значений от допустимых типов операндов.

3.4 Операторы

В языке L_4 операторы задают действия, выполняемые программой. Всего предусмотрено восемь видов операторов: оператор-объявление (3.4.1), оператор присваивания (3.4.2), оператор вызова функции (3.4.3), оператор выбора (3.4.4), два цикла с предусловием (3.4.5), оператор завершения функции (3.4.6) и оператор-предупреждение (3.4.7).

3.4.1 Оператор-объявление

Оператор-объявление служит для объявления и инициализации переменных. Он имеет две формы. Первая форма совпадает с объявлением параметра функции:

```
(тип имя)
```

Вторая форма оператора-объявления позволяет присвоить объявляемой переменной начальное значение:

```
(тип имя) := выражение
```

3.4.2 Оператор присваивания

Оператор присваивания служит для присваивания значений *ячейкам*. Под ячейками мы будем понимать локальные переменные, параметры функций и элементы массивов.

Синтаксически оператор присваивания выглядит как два выражения, между которыми стоит знак :=. Подразумевается, что значение правого выражения присваивается ячейке, определяемой левым выражением.

Разумеется, не всякое выражение может стоять в левой части оператора присваивания. Например, следующие операторы присваивания записаны неправильно:

```
5 := <!a 1>,
[Func #x] := 10
```

Уровень приоритета	Операция	Описание	Порядок выполнения
1	$<x\ y>$	Доступ к элементу массива x, имеющему индекс y.	$\longrightarrow$
	$[f\ x_1\ \ldots\ x_n]$	Вызов функции f с передачей фактических параметров $x_1, \ldots, x_n$.	
	new_ $T\ x$	Выделение памяти в куче для массива типа $<T>$ размера x.	
2	$-x$	Изменение знака числа x.	$\longleftarrow$
	not_ x	Логическое НЕ.	
3	x * y	Произведение x на y.	$\longrightarrow$
	x / y	Частное от деления x на y.	
	x **_mod_** y	Остаток от деления x на y.	
4	x **_pow_** y	Возведение x в степень y.	$\longleftarrow$
5	$x + y$	Сумма x и y.	$\longrightarrow$
	$x - y$	Разность x и y.	
6	x **_eq_** y	Сравнение x и y.	$\longrightarrow$
	x **_ne_** y		
	x **_lt_** y		
	x **_gt_** y		
	x **_le_** y		
	x **_ge_** y		
7	x **_and_** y	Логическое И.	$\longrightarrow$
8	x **_or_** y	Логическое ИЛИ.	$\longrightarrow$
	x **_xor_** y	Логическое ИСКЛЮЧАЮЩЕЕ ИЛИ.	

Таблица 2: Операции, их приоритет и порядок выполнения.

Приведём примеры правильных операторов присваивания:

```
#x := #a*2 + 10,
<!A 5> := <#B 6> * 10,
<[Func #x #y] !i+1> := 0
```

Здесь мы считаем, что в последнем операторе присваивания функция Func возвращает ссылку на массив целых чисел.

3.4.3 Оператор вызова функции

Оператор вызова функции позволяет вызывать функции, не возвращающие значения.

Он выглядит как

```
[f x1 ... xn]
```

Здесь f – имя функции, а $x_1 \ldots x_n$ – выражения, вычисляющие значения фактических параметров функции. Типы этих выражений должны совпадать с типами формальных параметров, указанными в заголовке функции.

3.4.4 Оператор выбора

Оператор выбора осуществляет передачу управления на разные участки кода в зависимости от того, истинно или ложно некоторое условие.

В языке L_4 самый простой вариант оператора выбора содержит только один участок кода, на который передаётся управление в случае истинности условия:

```
(? условие)
  операторы
%
```

x	y	$<x\ y>$
$<T>$	**int**	T
$<T>$	**char**	T

(a) Индексирование массива.

x	$-x$
int	**int**
char	**int**

(b) Унарный минус.

x	**not_** x
bool	**bool**

(c) Логическое НЕ.

x	y	$x + y$
int	**int**	**int**
int	**char**	**char**
char	**int**	**char**

(d) Операция +.

x	y	$x - y$
int	**int**	**int**
char	**char**	**int**
char	**int**	**char**

(e) Операция –.

x	y	$x\ op\ y$
int	**int**	**int**

(f) Операции *, /, **_mod_**, **_pow_**.

x	y	$x\ op\ y$
int	**int**	**bool**
int	**char**	**bool**
char	**int**	**bool**
char	**char**	**bool**
bool	**bool**	**bool**
$<T>$	$<T>$	**bool**

(g) Операции **_eq_**, **_ne_**.

x	y	$x\ op\ y$
int	**int**	**bool**
int	**char**	**bool**
char	**int**	**bool**
char	**char**	**bool**

(h) Операции **_lt_**, **_gt_**, **_le_**, **_ge_**.

x	y	$x\ op\ y$
bool	**bool**	**bool**

(i) Операции **_and_**, **_or_**, **_xor_**.

Таблица 3: Зависимости типа возвращаемого значения от типов операндов.

Условие является выражением, вычисляющим булевское значение. Операторы, формирующие «тело» оператора выбора, разделяются запятыми, а после последнего оператора ставится знак %.

Во втором варианте оператора выбора указываются два участка кода:

```
(? условие)
  операторы_1
+++
  операторы_2
%
```

Пример оператора выбора:

```
(#a _lt_ 0)
  #sign := -1
+++
  (#a _eq_ 0)
    #sign := 0
  +++
    #sign := 1
  %
%
```

3.4.5 Циклы с предусловием

Цикл с предусловием осуществляет повторное выполнение некоторого участка кода, называемого телом цикла, до тех пор, пока некоторое условие, проверяемое до выполнения тела цикла, остаётся истинным.

В языке L_4 присутствуют два варианта цикла с предусловием.

Первый вариант выглядит следующим образом:

```
(& условие)
  операторы
%
```

Условие является выражением, вычисляющим булевское значение. Операторы в теле цикла разделяются запятыми.

Второй вариант цикла с предусловием является вариацией цикла for:

```
(i : a , b , s)
  операторы
%
```

Эта запись подразумевает, что некоторая переменная i инициализируется значением выражения a и затем на каждой итерации

цикла к i прибавляется значение выражения s до тех пор, пока i не примет значение b.

Шаг цикла s должен иметь тип **int**. Если шаг равен 1, то его можно не указывать.

Переменная i может иметь тип **int** или **char**. При этом предусмотрена возможность объявления переменной прямо в операторе цикла. Например:

```
(<char> #letter : "A", "Z")
   [Print #letter]
%
```

3.4.6 Оператор завершения функции

Оператор завершения функции прекращает выполнение функции.

Оператор завершения может вызываться в любом месте тела функции. При этом для функции, возвращающей значение, он имеет вид

^ выражение

Здесь тип выражения должен совпадать с типом возвращаемого значения, указанном в заголовке функции.

Для функции, не возвращающей значения, оператор завершения не содержит выражения. Кроме того, такая функция вообще может обойтись без оператора завершения, потому что её выполнение будет автоматически прекращено после выполнения последнего оператора в её теле.

3.4.7 Оператор-предупреждение

Оператор-предупреждение обеспечивает аварийное завершение программы при невыполнении некоторого условия:

\ условие

Например, оператор

\ !x **_gt_** 0

вызовет завершение программы, если значение $!x$ меньше или равно 0.