Описание языка L_2

С.Ю. Скоробогатов

1 сентября 2009

1 Введение

Язык L_2 представляет собой игрушечный императивный язык программирования, предназначенный для проведения лабораторных работ по курсу «Конструирование компиляторов».

2 Лексика

В языке L_2 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_2 игнорирует комментарии и расположенные между лексемами так называемые *невидимые символы*, а именно: пробелы, знаки табуляции и перевода строки. При этом требуется, чтобы смежные 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

Символы ## , расположенные в любом месте строки программы, означают, что после них до конца строки идёт комментарий.

Также имеется многострочный комментарий, начинающийся с #(и заканчивающийся на)#.

2.2 Идентификаторы

Идентификатор представляет собой последовательность букв, цифр, пробелов и знаков подчёркивания, заключённую в фигурные скобки. Реализации языка L_2, поддерживающие Unicode, должны разрешать использование в идентификаторах букв любых алфавитов, охваченных стандартом Unicode.

Примеры идентификаторов:

```
{alpha}  {sum of digits}  {x_1}
```

2.3 Ключевые слова

Ниже приведён список используемых в программах на языке L_2 ключевых слов:

bool	**int**	**return**	**void**
char	**loop**	**then**	
else	**null**	**while**	

2.4 Константы

Константы в языке L_2 представляют собой изображения значений, известных на момент компиляции, и бывают нескольких типов.

2.4.1 Целочисленные константы

Целочисленные константы в языке L_2 изображают неотрицательные целые числа в диапазоне от 0 до 2147483647.

Целочисленная константа записывается в виде последовательности цифр, после которой через знак доллара может указываться основание системы счисления. Например:

```
12000000        7FFFF$16
100111011$2     gh10fj$20
1202211$03      0
```

Код	Последовательность	Код	Последовательность	Код	Последовательность
7	%BEL%	10	%LF%	13	%CR%
8	%BS%	11	%VT%	34	%"%
9	%TAB%	12	%FF%	37	%%

Таблица 1: Escape-последовательности.

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.

2.4.2 Символьные константы

Символьные константы изображают символы Unicode или ASCII в зависимости от того, поддерживает компилятор языка L_2 стандарт Unicode или не поддерживает.

Символы записываются в кавычках и предваряются знаком доллара. При этом некоторые управляющие символы ASCII, символ «кавычка» и символ «процент» изображаются Escape-последовательностями, приведёнными в таблице 1. Например:

```
$"Q"    $"я"    $"%CR%"    $"%"%"
```

Кроме того, любой символ с кодом x может быть представлен как $\$x_{16}$, где x_{16} – шестнадцатеричная запись числа x. Примеры:

```
$"%CR%"   ## перевод каретки
$D        ## тоже перевод каретки
```

2.4.3 Строковые константы

Строковая константа в языке L_2 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 и Escape-последовательностей (см. таблицу 1), заключённая в кавычки.

Особая форма строковой секции позволяет представить одиночный символ с кодом x. Секция в особой форме выглядит как $\%x_{16}$, где x_{16} – шестнадцатеричная запись числа x.

Например, строковая константа

```
"We say %"%Hello, World!%"%" %A
```

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 null

Ключевое слово **null** обозначает нулевую ссылку.

2.5 Знаки операций и другие разделители

В языке L_2 используются следующие знаки операций и разделители:

```
~    ;    ,    [    ]    (    )
^    *    /    %    -    +
=    :=   <-   !    &    |    @
==   !=   <    >    <=   >=
```

3 Синтаксис и семантика

Программа на языке L_2 представляет собой набор *определений функций* (3.1). Порядок определений не важен, то есть из функ-

ции 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 записывается следующим образом:

тип имя_функции <– параметры

Здесь тип – это тип возвращаемого функцией значения. Причём в случае, если функция не возвращает значения, вместо типа указывается ключевое слово **void**.

Список формальных параметров записывается как

тип1 имя1, ... , типN имяN

При этом, если функция не имеет параметров, знак <– после имени функции не ставится.

После заголовка функции ставится знак «равно», за которым следует последовательность *операторов* (3.4), формирующая тело функции. Операторы разделяются точками с запятой, а после последнего оператора ставится точка.

Пример определения функции:

```
int[] {SumVectors}
  <– int[] A, int[] B
  = int {size}:={Length}<–{A};
    int[] {C}:=int {size};
    0 ~ {size}–1 loop {i}
      {C}{i}:={A}{i}+{B}{i}.;
    return {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 )#
    return 0.
```

3.2 Типы данных

Язык L_2 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_2 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 динамически и автоматически освобождается сборщиком мусора.

Тип-массив задаётся с помощью пары квадратных скобок [ ], которая записывается после изображения 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 **null**.

3.3 Выражения

Выражения в языке L_2 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

Уровень приоритета	Операция	Описание	Порядок выполнения
1	$x\ y$	Доступ к элементу массива x, имеющему индекс y.	$\longrightarrow$
	$T\ x$	Выделение памяти в куче для массива типа $T[\ ]$ размера x.	
2	$-x$	Изменение знака числа x.	$\longleftarrow$
	$!x$	Логическое НЕ.	
3	$x\ \hat{}\ y$	Возведение x в степень y.	$\longleftarrow$
4	$x\ *\ y$	Произведение x на y.	$\longrightarrow$
	$x\ /\ y$	Частное от деления x на y.	
	$x\ \%\ y$	Остаток от деления x на y.	
5	$x + y$	Сумма x и y.	$\longrightarrow$
	$x - y$	Разность x и y.	
6	$f <- x_1, \ldots, x_n$	Вызов функции f с передачей фактических параметров $x_1, \ldots, x_n$.	$\longrightarrow$
7	$x == y$	Сравнение x и y.	$\longrightarrow$
	$x\ !=\ y$		
	$x < y$		
	$x > y$		
	$x <= y$		
	$x >= y$		
8	$x\ \&\ y$	Логическое И.	$\longrightarrow$
9	$x\ \vert\ y$	Логическое ИЛИ.	$\longrightarrow$
	$x\ @\ y$	Логическое ИСКЛЮЧАЮЩЕЕ ИЛИ.	

Таблица 2: Операции, их приоритет и порядок выполнения.

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 $T\ x$ размер x должен иметь тип **int**.

Операции сравнения == и != позволяют сравнивать ссылки на массивы с константой **null**.

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

3.4 Операторы

В языке L_2 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), цикл с постусловием (3.4.6) и оператор завершения функции (3.4.7).

3.4.1 Оператор-объявление

Оператор-объявление служит для объявления и инициализации переменных. Он начинается с изображения типа, за которым

x	y	$x\ y$
$T[\,]$	**int**	T
$T[\,]$	**char**	T

(a) Индексирование массива.

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

(b) Унарный минус.

x	$!x$
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 ^, *, /, %.

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 ==, !=.

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

(h) Операции <, >, <=, >=.

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

(i) Операции &, |, @.

Таблица 3: Зависимости типа возвращаемого значения от типов операндов.

следует список объявляемых переменных. Переменные в списке разделяются запятыми. Кроме того, сразу после имени переменной может быть указан знак :=, за которым должно следовать выражение, значением которого переменная инициализируется.

Примеры операторов-объявлений:

```
char {c};
int[] {A} := int 5,
      {B} := int 10;
int {x}, {y} := {B}3, {z} := {y}
```

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;
```

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_2 самый простой вариант оператора выбора содержит только один участок кода, на который передаётся управление в случае истинности условия:

```
условие then
  операторы .
```

Условие является выражением, вычисляющим булевское значение. Операторы, расположенные после ключевого слова **then**, разделяются точками с запятой, а после последнего оператора ставится точка.

Во втором варианте оператора выбора указываются два участка кода:

```
условие then
  операторы_1
else
  операторы_2 .
```

Пример оператора выбора:

```
{a} < 0 then
  {sign} := -1
else {a} == 0 then
  {sign} := 0
else
  {sign} := 1..
```

В примере используются сразу два оператора выбора. При этом один из них вложен в false-ветку второго. Именно поэтому в конце стоят сразу две точки.

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_2 присутствуют два варианта цикла с предусловием.

Первый вариант выглядит следующим образом:

```
условие loop
  операторы .
```

Условие является выражением, вычисляющим булевское значение. Операторы в теле цикла разделяются точками с запятой, а после последнего оператора ставится точка.

Второй вариант цикла с предусловием является вариацией цикла for:

```
a ~ b loop i
  операторы .
```

Эта запись подразумевает, что некоторая переменная i инициализируется значением выражения a и затем на каждой итерации цикла к i прибавляется единица, пока i не примет значение b.

Переменная i может иметь тип **int** или **char**.

Пример:

```
char {letter};
$"A" ~ $"Z" loop {letter}
  {Print}<-{letter}.
```

3.4.6 Цикл с постусловием

Цикл с постусловием осуществляет повторное выполнение некоторого участка кода, называемого телом цикла, до тех пор, пока некоторое условие, которое первый раз проверяется после выполнения тела цикла, остаётся истинным.

В языке L_2 цикл с постусловием выглядит следующим образом:

```
loop
  операторы
while условие .
```

Условие является выражением, вычисляющим булевское значение. Операторы в теле цикла разделяются точкой с запятой.

3.4.7 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

```
return 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.