# DLang 语言设计

Kaiyuan Zhang, Zhe Qiu

DLang是一个类似于C语言的过程式语言，我们去除了C语言中一些不被经常用到的特性，使得DLang更加的简洁易懂。下面介绍一下DLang的基本语法。

###### *变量声明*

DLang的变量声明类似于C。格式为

int i, j;

float k；

DLang支持的数据类型有 int，char，float, string以及数组，数组的声明和使用与C语言相似。

需要注意的是，在一个函数当中，变量名应是唯一的，因此下面的代码是不合法的，因为变量名a被使用了两次

int $[ main ] {

int a;

int b;

a <- 10;

b <- 0;

if (b == 0) {

int a;

b <- b + a;

}

}

###### *赋值语句*

在DLang中，不采用‘=’作为赋值操作符，因为在C语言中‘=’和‘==’非常容易混淆，造成很多不必要的debug开销。我们采用的是‘<-’符号。它的好处在于非常的直观，同时与常用的伪代码一致，有助于程序员对于算法的描述。

###### *计算表达式*

DLang支持+， -， \*， /， %和 （）预算符，优先级的处理与C语言相同。逻辑运算符|| && !的优先级比算数运算符低。不提供异或等位运算操作。

###### *用户输入*

提供两个函数， hla\_stdin\_geti32 和 hla\_stdout\_puti32， 实现单个数字的输入输出。同时提供hla\_stdout\_puts和hla\_stdout\_putc 输出一个字符串或者单个字符。

###### *数组和复合类型*

DLang中的数组与C语言相同。

int arr[100];

arr[0] = 1;

复合类型方面，支持struct 语法与C语言略有差异

Struct的声明

struct point {

int x;

int y;

}

Struct的使用

struct point p1;

p1-> x <- 2;

p2-> y <- 6;

###### *复合语句*

DLang支持if、for、while语句，它们的语法与C语言相似。DLang中的foreach语句语法与for语句近似：

if (exp1) {

//do something;

} else {

//Do something else;

}

for (i = 0; i < 10; i <- i + 1) {

//do loop task;

}

while (loop not end) {

//Loop;

}

int i;

int sum;

sum <- 0;

int array[100];

foreach (i in array) {

sum <- sum+i;

}

###### *函数的声明与调用*

DLang语言的函数声明和调用与C语言有一些区别。函数声明的格式如下：

int $[ gcd : int i, int j] {

if ( !j )

return i;

return $[ gcd : j , i%j ];

}

这一语法参照了一些函数式语言的语法，因为我们希望DLang有一些函数式语言的特性。

###### *控制流*

DLang提供和C语言相同的break和continue语句来改变程序的控制流。考虑到程序的可读性，我们并不提供goto语句。

###### *注释*

DLang支持使用“//”进行注释。一行中所有在“//”之后的字符都会被忽略。多行注释用

/\* comment \*/

###### *try-throw-catch子句*

DLang支持try-throw-catch子句，可以throw任意32位整数。try-throw-catch示例如下：

try {

throw 1;

} catch 1: {

$[hla\_stdout\_puts: “woops”];

}

###### *系统调用*

DLang提供对于UNIX类型系统调用的支持。调用方法采用int 0x80 通过%eax储存系统调用的类型，%ebx, %ecx, %edx, %edi, %esi保存参数，实现系统调用。如write、open、fork、getpid、wait等常见系统调用都可以在DLang中使用。

###### *设置断点*

为方便程序员的调试，DLang支持程序员手工在程序中加入断点通过调用hla\_brkpt这样程序员在使用gdb等工具进行调试的时候可以更加方便地找到bug所在的位置。

###### *随机数生成*

DLang支持随机数的生成。在程序中，要首先使用hla\_rand\_randomize函数初始化随机数生成器，之后调用hla\_rand\_uniform生成随机数。

下面的代码可以输出一个0-99的随机整数

$[ hla\_rand\_randomize];

$[ hla\_stdout\_puti32 : $[ hla\_rand\_uniform ] % 100 ];

###### *十六进制输入输出*

DLang同时支持十六进制的输入和输出，使用hla\_stdin\_geth32和hla\_stdout\_puth32即可输入和输出十六进制的整数。

###### *QuickSort*

下面是由DLang描述的quicksort程序：

int numbers[100];

void $[ q\_sort : int left, int right]{ //quicksort

int pivot;

int l\_hold;

int r\_hold;

int tmp ;

l\_hold <- left;

r\_hold <- right;

pivot <- numbers[left];

while (l\_hold <= r\_hold) {

while (numbers[l\_hold] < pivot)

l\_hold <- l\_hold + 1;

while (numbers[r\_hold] > pivot)

r\_hold <- r\_hold - 1;

if (l\_hold <= r\_hold) {

tmp <- numbers[r\_hold];

numbers[r\_hold] <- numbers[l\_hold];

numbers[l\_hold] <- tmp;

r\_hold <- r\_hold - 1;

l\_hold <- l\_hold + 1;

}

}

if (left < r\_hold)

$[ q\_sort : left, r\_hold];

if (l\_hold < right)

$[ q\_sort : l\_hold, right];

}

###### *N皇后问题*

下面是使用DLang给出的N皇后问题的解：

void $[ solve : int depth ]{

int x ;

int y ;

int success ;

if ( found )

return ;

for ( x<- 1 ; x<= size ; x<-x+1 ) {

numbers[depth] <- x ;

success <- 1 ;

for ( y <- 0 ; y < depth ; y<-y+1 ) {

if ( (numbers[y]==numbers[depth] ) ||

((numbers[y]-x)==(y-depth) ) ||

((numbers[y]-x)==(depth-y) )

){

success <- 0 ;

break;

}

}

if ( success ){

if ( depth == size - 1 ){

$[output\_routine : size ] ;

found <- 1 ;

} else

$[ solve : depth+1 ];

}

numbers[depth] <- 0 ;

}

}

Reserved Keywords:

'break', 'char', 'const', 'continue', 'do', 'else', 'float', 'foreach', 'for', 'in', 'if', 'int', 'return', 'struct', 'void', 'while', ‘try’, ‘throw’, ‘catch’

%-----------LEX------------------

% Integer literal

t\_ICONST = '\d+'

% Floating literal

t\_FCONST = '((\d+)(\.\d+)(e(\+|-)?(\d+))? | (\d+)e(\+|-)?(\d+))'

% String literal

t\_SCONST = '\"([^\\\n]|(\\.))\*?\"'

% Character constant 'c' or L'c'

t\_CCONST = '\'([^\\\n]|(\\.))\*?\''

% Comments (ignored)

t\_comment='(/\\*(.|\n)\*?\\*/)|(//[^\n]\*)'

%Identifier

t\_ID='[A-Za-z\_][\w\_]\*'

%-------------YACC----------------------

%token int\_const char\_const float\_const id string

%%

external\_decl = function\_definition

| decl

;

function\_definition = type\_spec declarator compound\_stat

;

decl = type\_spec declarator ';'

;

type\_spec = 'void'

| 'char'

| 'int'

| 'float'

| struct\_spec

;

struct\_spec = 'struct' id '{' struct\_decl\_list '}'

| 'struct' '{' struct\_decl\_list '}'

| 'struct' id

;

struct\_decl\_list = struct\_decl

| struct\_decl\_list struct\_decl

;

struct\_decl = declarator\_list ';'

;

declarator\_list = declarator

| declarator\_list ',' declarator

;

declarator = '\*' direct\_declarator

| direct\_declarator

;

direct\_declarator = id

| '(' declarator ')'

| direct\_declarator '[' logical\_exp ']'

| '$[' direct\_declarator ':' param\_list ']'

| '$[' direct\_declarator ']'

;

param\_list = param\_decl

| param\_list ',' param\_decl

;

param\_decl = type\_spec declarator

;

stat = exp\_stat

| compound\_stat

| selection\_stat

| iteration\_stat

| jump\_stat

;

exp\_stat = exp ';'

| ';'

;

compound\_stat = '{' decl\_list stat\_list '}'

| '{' stat\_list '}'

| '{' decl\_list '}'

| '{' '}'

;

stat\_list = stat

| stat\_list stat

;

selection\_stat = 'if' '(' exp ')' stat

| 'if' '(' exp ')' stat 'else' stat

;

iteration\_stat = 'while' '(' exp ')' stat

| 'do' stat 'while' '(' exp ')' ';'

| 'for' '(' exp ';' exp ';' exp ')' stat

| 'foreach' '(' id 'in' stat ')' stat

;

jump\_stat = 'continue' ';'

| 'break' ';'

| 'return' exp ';'

| 'return' ';'

;

exp = assignment\_exp ;

assignment\_exp = logical\_exp

| unary\_exp '<--' assignment\_exp

;

logical\_exp = relational\_exp

| logical\_and\_exp '||' relational\_exp

| logical\_and\_exp '&&' relational\_exp

;

relational\_exp = additive\_exp

| relational\_exp '<' additive\_exp

| relational\_exp '>' additive\_exp

| relational\_exp '<=' additive\_exp

| relational\_exp '>=' additive\_exp

| relational\_exp '=' additive\_exp

| relational\_exp '!=' additive\_exp

;

additive\_exp = mult\_exp

| additive\_exp '+' mult\_exp

| additive\_exp '-' mult\_exp

;

mult\_exp = cast\_exp

| mult\_exp '\*' cast\_exp

| mult\_exp '/' cast\_exp

| mult\_exp '%' cast\_exp

;

cast\_exp = unary\_exp

| '(' type\_spec ')' cast\_exp

;

unary\_exp = postfix\_exp

| unary\_operator cast\_exp

;

unary\_operator = '\*' | '+' | '-' | '!'

;

postfix\_exp = primary\_exp

| id '[' exp ']'

| '$[' id ':' argument\_exp\_list ']'

| '$[' id ']'

| postfix\_exp '-->' id

;

primary\_exp = id

| const

| string

| '(' exp ')'

;

argument\_exp\_list = assignment\_exp

| argument\_exp\_list ',' assignment\_exp

;

const = int\_const

| char\_const

| float\_const

;