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
                ;
```