MyLang 语言设计

Kaiyuan Zhang, Zhe Qiu

MyLang 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 MyLang 更加的简洁易懂。下面介绍一下 MyLang 的基本语法。

变量声明

MyLang 的变量声明类似于 C。格式为

```
int  i, j;
float  k;
```

MyLang 支持的数据类型有 `int, char, float` 以及指针，指针的生命与 C 语言类似。MyLang 不支持指向指针、函数和复合类型的指针，因为这样的特性经常让用户困惑。

赋值语句

在 MyLang 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

MyLang 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， read 和 write， 实现单个数字的输入输出。同时提供 print_str() 输出一个字符串。

数组和复合类型

MyLang 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

MyLang 支持 if、for、while 语句，它们的语法与 C 语言相似。MyLang 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=0;
    int array[100];
    foreach (i in array) {
        sum = sum+i;
    }
```

函数的声明与调用

MyLang 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 MyLang 有一些函数式语言的特性。

控制流

MyLang 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

MyLang 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 */

QuickSort

下面是由 MyLang 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;
            l_hold <- left;
            r_hold <- right;
            pivot <- numbers[left];
            while (left < right) {
                  while ((numbers[right] >= pivot) &&
(left < right))
                        right <- right - 1;
                  if (left != right){
                        numbers[left] <-
numbers[right];
                        left <- left + 1;
                  }
                  while ((numbers[left] <= pivot) &&
(left < right))
                        left <- left + 1;
                  if (left != right){
                        numbers[right] <-
numbers[left];
                        right <- right - 1;
                  }
            }
            numbers[left] <- pivot;
            pivot <- left;
            left <- l_hold;
            right <- r_hold;
```

```
            if (left < pivot)
                  $[ q_sort :  left, pivot-1];
            if (right > pivot)
                  $[ q_sort :  pivot+1, right];
}
```

Reserved Keywords:
'break', 'char', 'const', 'continue', 'do', 'else', 'float', 'foreach', 'for', 'in', 'if', 'int', 'return', 'struct', 'void', 'while'

```
%-----------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
                        ;
```