# 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

;