

معرفی زبان تسلنگ

مستند حاضر زبان ساده‌ی تسلنگ (TSLANG) را معرفی می‌کند. در گام‌های تمرین عملی درس طراحی کامپایلر، بخش‌هایی از یک مترجم برای این زبان نوشته می‌شوند. قواعد این زبان در ادامه‌ی این مستند بیان می‌شوند.

- ۱ زبان تسلنگ دارای دو نوع داده‌ی اصلی است: اعداد صحیح (num) و لیست‌ها (list).
- ۲ برنامه‌های این زبان در یک فایل نوشته می‌شوند که شامل تعدادی تابع است. در این زبان متغیرهای سراسری (Global) وجود ندارند.
- ۳ خط اول هر تابع، نام تابع، ورودی‌های آن و نوع خروجی تابع را مشخص می‌کند.
- ۴ بدنه‌ی هر تابع بین دو علامت «{» و «}» قرار می‌گیرد و شامل تعدادی عبارت (Statement) می‌باشد.
- ۵ شباهت زیادی بین ساختار عبارت‌ها و اولویت عملگرها در زبان تسلنگ و زبان C وجود دارد.
- ۶ هر بلوک (Block) در این زبان نیز بین دو علامت «{» و «}» قرار می‌گیرد.
- ۷ در هر بلوک می‌توان متغیر تعریف نمود و بلوک‌ها می‌توانند تو در تو (Nested) باشند. حوزه‌ی (Scope) هر متغیر مشابه زبان C تعریف می‌گردد.
- ۸ در هر بلوک می‌توان متغیرهای محلی تعریف کرد:

```
local num n;  
local list A;
```

- ۹ مقدار خروجی یک تابع با استفاده از کلمه‌ی کلیدی «return» مشخص می‌شود و با اجرای عبارتی که با این کلمه شروع می‌شود، اجرای تابع خاتمه می‌یابد.

۱۰ مثالی از تعریف یک تابع در ادامه نشان داده می‌شود. تابع sum3 سه عدد دریافت می‌کند و مجموع آنها را بر می‌گرداند.

```
function sum3:
  param num a;
  param num b;
  param num c;
  returns num;
{
  local num sum;
  sum = a + b + c;
  return sum;
}
```

۱۱ در صورتی که تابع چیزی را بر نگرداند، نوع برگشتی آن باید nil باشد.

۱۲ همان طور که در مثال بعدی دیده می‌شود، می‌توان یک لیست را به یک تابع فرستاد و با استفاده از حلقه‌ی for عدد‌های موجود در آن آرایه را بررسی کرد.

```
function vecsum:
  param list A;
  returns num;
{
  local num sum;
  sum = 0;
  for (a in A) {
    sum = sum + a;
  }
  return sum;
}
```

۱۳ مثالی از فراخوانی تابع vecsum در ادامه دیده می‌شود. تابع list(n) یک لیست با n عنصر بر می‌گرداند.

```

function main:
    returns num;
{
    local list A;
    A = list(4);
    A[0] = numread();
    A[1] = numread();
    A[2] = numread();
    A[3] = numread();
    numprint (vecsum(A));
    return 0;
}

```

۱۴ هر برنامه‌ی تسلنگ می‌تواند شامل یک تابع با نام main باشد که اجرای برنامه با فراخوانی آن آغاز می‌گردد.

۱۵ تابع main بدون ورودی است و یک عدد بر می‌گرداند که کد برگشتی برنامه را مشخص می‌نماید.

۱۶ در زبان تسلنگ از عبارت شرطی if و حلقه‌ی while با ساختاری مشابه زبان C می‌توان استفاده کرد.

۱۷ مثال زیر استفاده از if را نمایش می‌دهد.

```

# The Fibonacci sequence
function fib:
    param num n;
    returns num;
{
    if (n < 2)
        return 1;
    return fib(n - 1) + fib(n - 2);
}

```

تابع	توضیح
<code>numread ()</code>	یک عدد را از ورودی استاندارد می خواند و بر می گرداند.
<code>numprint (n)</code>	عدد ورودی را در خروجی استاندارد چاپ می کند.
<code>list (n)</code>	یک آرایه با <code>n</code> عنصر بر می گرداند.
<code>listlen (v)</code>	اندازه ی یک لیست را بر می گرداند.
<code>exit (n)</code>	برنامه را با کد برگشتی داده شده خاتمه می دهد.

قواعد تجزیه‌ی زبان تسلنگ

در ادامه ساختار BNF زبان تسلنگ نمایش داده شده است. در این ساختار اولویت‌های عملگرها (که مشابه عملگرهای زبان C هستند) در نظر گرفته نشده است. همچنین در برنامه‌های زبان تسلنگ، علامت # و حروفی که بعد از آن آمده‌اند تا آخر خط توضیح (Comment) محسوب می‌شوند.

```
prog ::=      func |
              func prog
func ::=      function iden : plist { body }
body ::=     stmt |
              stmt body
stmt ::=     expr ; |
              defvar ; |
              if ( expr ) stmt |
              if ( expr ) stmt else stmt |
              while ( expr ) stmt |
              for ( iden in expr ) stmt |
              return expr ; |
              { body }
defvar ::=   local type iden
expr ::=    iden ( clist ) |
            expr [ expr ] |
            expr = expr |
            expr + expr |
            expr - expr |
            expr * expr |
            expr / expr |
            expr % expr |
            expr < expr |
            expr > expr |
            expr == expr |
            expr <= expr |
            expr >= expr |
            expr || expr |
            expr && expr |
            ! expr |
            - expr |
            + expr |
            ( expr ) |
            iden |
            num
plist ::=   returns type ; |
           param type iden ; plist
clist ::=  |
           expr |
           expr , clist
type ::=  num |
          list
num ::=  [0-9]+
iden ::= [a-zA-Z_][a-zA-Z_0-9]*
```