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 به کمک کلمه‌ی کلیدی local می‌توان متغیرهای محلی تعریف کرد (نوع متغیر بعد از علامت : قرار می‌گیرد):

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

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.

```
sum3(a: num, b: num, c: num): num
{
	local sum: num;
	sum = a + b + c;
	return sum;
}
```

۱۱ 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.

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

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

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

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

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

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

۱۶ مثال زیر استفاده از if را نمایش می‌دهد.

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

۱۷ جدول زیر توابع داخلی تسلنگ را نشان می‌دهد.

تابع	توضیح
`numread()`	یک عدد را از ورودی استاندارد می‌خواند و بر می‌گرداند.
`numprint(n)`	عدد ورودی را در خروجی استاندارد چاپ می‌کند.
`vec(n)`	یک آرایه با n عنصر بر می‌گرداند.
`veclen(v)`	اندازه‌ی یک لیست را بر می‌گرداند.
`exit(n)`	برنامه را با کد برگشتی داده شده خاتمه می‌دهد.

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 تسلنک، علامت # و حروفی که بعد از آن آمده‌اند تا آخر خط توضیح (Comment) محسوب می‌شوند.

```
prog ::=      func |
              func prog
func ::=      iden ( flist ) : type { 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 flist
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
flist ::=      |
              iden: type |
              iden: type , flist
clist ::=      |
              expr |
              expr , clist
type ::=      num |
              list
num ::=       [0-9]+
iden ::=      [a-zA-Z_][a-zA-Z_0-9]*
```