

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 (Int) و آرایه‌ها (Array).
- 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 توکن «:» و «end» 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 توکن «:» و «end» 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 متغیرهایی محلی هر بلوک با استفاده از کلمه‌ی کلیدی «Int» یا «Array» و به شکل زیر تعریف می‌شوند:

```
val Int n;           -- n is a variable of type Int
val Array v;         -- v is a variable of type Array
```

- 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(Int a, Int b, Int c) returns Int:
    val Int sum;
    sum = a + b + c;
    return sum;
end
```

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 عددی‌های موجود در آن آرایه را بررسی کرد.

```
function vecsum(Array A) returns Int:
    val Int sum;
    sum = 0;
    foreach (a of A):
        sum = sum + a;
    end
    return sum;
end
```

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

```
function main() returns Int:
    val Array A;
    A = createArray(4);
    A[0] = getInt();
    A[1] = getInt();
    A[2] = getInt();
    A[3] = getInt();
    printInt(vecsum(A));
    return 0;
end
```

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(Int n) returns Int:
    if (n < 2):
        return 1;
    end
    return fib(n - 1) + fib(n - 2);
end
```

۱۸ جدول زیر توابع داخلی تسلنگ را نشان می‌دهد.

تابع	توضیح
getInt ()	یک عدد را از ورودی استاندارد می‌خواند و بر می‌گرداند.
printInt (n)	عدد ورودی را در خروجی استاندارد چاپ می‌کند.
createArray (n)	یک آرایه با n عنصر بر می‌گرداند.
arrayLength (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 تسلنگ، دو علامت - پشت سر هم و حروفی که بعد از آن آمده‌اند تا آخر خط توضیح (Comment) محسوب می‌شوند.

```
prog ::=      func |
              func prog
func ::=      function iden ( flist ) returns type : body end |
body ::=      stmt |
              stmt body
stmt ::=      expr ; |
              defvar ; |
              if ( expr ) stmt |
              if ( expr ) stmt else stmt |
              while ( expr ) do stmt |
              foreach ( iden of expr ) stmt |
              return expr ; |
              : body end
defvar ::=    val 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 && expr |
              ! expr |
              - expr |
              + expr |
              ( expr ) |
              iden |
              num
flist ::=      |
              type iden |
              type iden , flist
clist ::=      |
              expr |
              expr , clist
type ::=      Int |
              Array |
              Nil
num ::=      [0-9]+
iden ::=      [a-zA-Z_][a-zA-Z_0-9]*
```