

## 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 بردارها (vec).

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، ورودی‌ها و خروجی آن 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 متغیرهایی محلی هر بلوک با استفاده از کلمه‌ی کلیدی «num» یا «vec» و به شکل زیر تعریف می‌شوند:

```
num n;          # n is a variable of type num
vec v;          # v is a variable of type vec
```

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.

```
num sum3(num a, num b, num c)
{
    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 بردار را به یک تابع فرستاد و با استفاده از حلقه‌ی for عددهای موجود در آن آرایه را بررسی کرد.

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

۱۳ مثالی از فراخوانی تابع vecsum در ادامه دیده می‌شود.

```
num main()
{
    vec A;
    vecadd(A, numread());
    vecadd(A, 0);
    vecadd(A, numread());
    vecadd(A, 0);
    A[1] = 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 word 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.

```
# Inefficient calculation of the Fibonacci sequence
num fib(num n)
{
    if (n < 2)
        return 1;
    return fib(n - 1) + fib(n - 2);
}
```

۱۸ جدول زیر توابع داخلی تسلنگ را نشان می‌دهد.

تابع	توضیح
num numread()	یک عدد را از ورودی استاندارد می‌خواند و بر می‌گرداند.
nil numprint(num n)	عدد ورودی را در خروجی استاندارد چاپ می‌کند.
num veclen(vec v)	اندازه‌ی یک بردار را بر می‌گرداند.
nil vecresize(vec v, num n)	اندازه‌ی یک بردار را تغییر می‌دهد.
nil vecput(vec v, num n)	طول بردار را یک واحد افزایش می‌دهد و عدد داده شده را به آخر آن اضافه می‌کند.
nil vecadd(vec dst, vec v1, vec v2)	عناصر متناظر بردارهای v1 و v2 را با هم جمع می‌کند و در بردار dst قرار می‌دهد. سه بردار ورودی باید با هم اندازه باشند.
nil vecmul(vec dst, vec v1, vec v2)	عناصر متناظر بردارهای v1 و v2 را با هم ضرب می‌کند و در بردار dst قرار می‌دهد. سه بردار ورودی باید هم اندازه باشند.
nil exit(num 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، علامت # و حروفی که بعد از آن آمده‌اند تا آخر خط توضیح (Comment) محسوب می‌شوند.

```
prog ::=      func |
              func prog
func ::=      type iden ( flist ) { body } |
body ::=     stmt |
              stmt body
stmt ::=     expr ; |
              defvar ; |
              if ( expr ) stmt |
              if ( expr ) stmt else stmt |
              for ( iden in expr ) stmt |
              while ( expr ) stmt |
              return expr ; |
              { body }
defvar ::=   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 or expr |
            expr and expr |
            ! expr |
            - expr |
            + expr |
            ( expr ) |
            iden |
            num
flist ::=   |
            type iden |
            type iden , flist
clist ::=  |
```

```
expr |
expr , clist
type ::= num |
vec |
nil
num ::= [0-9]+
iden ::= [a-zA-Z_][a-zA-Z_0-9]*
```