کامپایلر Teslang: گام دوم

در گام دوم از تمرین عملی درس طراحی کامپایلر، به تحلیل نحوی و مفهومی پرداخته می‌شود. در این گام باید برنامه‌ای نوشته شود که با خواندن یک فایل TesLang از ورودی استاندارد و تحلیل نحوی و مفهومی آن، پیغام‌های مناسبی چاپ کند. دقت کنید که تجزیهٔ برنامه ورودی الزامی است و منطق برنامه باید به کمک عملیات مفهومی نوشته شود. در این گام باید هر یک از خطاهای زیر را پیاده‌سازی نمایید:

۱. فراخوانی یک تابع با تعداد نادرست پارامترها
۲. فراخوانی یک تابع با نوع نادرست پارامترها
۳. برگرداندن مقداری با نوع نادرست از یک تابع
۴. دسترسی به متغیرهایی که تعریف نشده‌اند
۵. دسترسی به متغیرهایی که مقداردهی نشده‌اند
۶. مشخص کردن نادرست نوع داده ای متغیرها
۷. خطاهای مفهومی و نحوی دیگری که توسط استاد به شما در کلاس آموزش داده شدند

```
fn find(A as vector, n as int) <int>
{
    k :: int;
    j :: int;

    for (i = 0 to length(A))
    begin
        if [[ n == k ]]
        begin
            return j;
        end
        j = x + 1;
    end

    return -1;
}

fn main() <null>
{
    A :: int;
    a :: int;

    A = list(3);
    A[0] = 1;
    A[1] = 2;
    A[2] = 3;

    print(find(A, a));
    print(find(A));
    print(find(a, A));

    return A;
}
```

خطاهایی که در کد بالا وجود دارند و باید توسط کامپایلر شما نمایش داده شوند[1]:

```
Error:
function 'main': variable 'A' expected to be of type 'vector' but it
is 'int' instead.
Error:
function 'find': Variable 'k' is used before being assigned.
Error:
function 'find': expected 'A' to be of type 'vector', but got 'int'
instead.
Error:
function 'find': expects 2 arguments but got 1.
Error:
function 'find': expected 'n' to be of type 'int', but got 'null'
instead.
Error:
function 'find': wrong type 'float' found. types must be one of the
following 'int', 'string', 'vector’
Error:
function 'find': variable 'x' is not defined.
Error:
function 'find': wrong return type. expected 'vector’ but got 'int'.
```

[1] بدیهی است که تمامی خطاهای این گام در اینجا نمایش داده نشده‌اند. کامپایلر شما باید بیشترین خطاهای کاربردی را نمایش دهد.

- در خروجی کامپایلر شما، همانند گام‌های پیشین باید اطلاعات کافی از مکان رخ دادن خطا نمایش داده شود.
- برای پیاده‌سازی جدول نمادها باید از ساختار درختی و تو در توی آن استفاده کنید و توابع دسترسی مناسبی را برای آن طراحی نمایید. از لینک‌های زیر دربارهٔ این طراحی بخوانید:

  https://www.geeksforgeeks.org/symbol-table-compiler/

  https://www.tutorialspoint.com/compiler_design/compiler_design_symbol_table.htm

- تجزیه نحوی کامپایلر می‌تواند به صورت دستی و بدون استفاده از کتابخانه‌های آماده و با استفاده از الگوریتم‌های LL, LR, SLR, LALR انجام شود.
- در رابطه با الگوی visitor برای ساخت و پیمایش درخت تجزیه می‌توانید از لینک زیر استفاده کنید:

  https://refactoring.guru/design-patterns/visitor

- از شما انتظار می‌رود به کد نوشته مسلط باشید و توانایی ارائه حضوری آن را داشته باشید. **استفاده از کدهای ترم‌های پیشین و کپی مجاز نیست** و در صورتی که نیاز به راهنمایی دارید با دستیاران آموزشی بخش پروژه در ارتباط باشید.