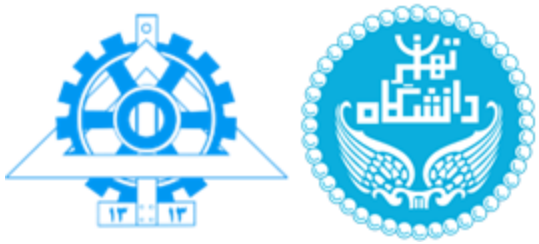

به نام خدا

تمرین کامپیوتری اول درس طراحی کامپایلر

بهار ۱۴۰۴

فهرست مطالب

فهرست مطالب **1**
ساخت AST **2**
تعریف گره‌های AST 2
ایجاد گره‌ها در گرامر ANTLR 3
نحوه ارزیابی **3**
تعداد statement ها برای scope ها 3
عمق expression-ها 5
تبدیل C به CPY (ویژه مهندسی کامپیوتر) **7**
کد نمونه 7
نودهای خروجی **8**
نکات مهم **8**

ساخت AST

در این بخش لازم است درخت AST را برای کد ورودی تشکیل دهید. برای این کار لازم است موارد زیر پیاده‌سازی شوند:

- گره‌های درخت (شامل کلاس‌ها، روابط بین آنها و ارث‌بری)
- ایجاد گره‌ها در گرامر antlr
- اضافه کردن متدهای مربوط به بخش ارزیابی

در این پروژه، لازم است درخت AST (Abstract Syntax Tree) برای کد ورودی تولید شود. AST یک نمایش درختی از کد منبع است که ساختار نحوی آن را به‌صورت سلسله‌مراتبی نمایش می‌دهد و نقش مهمی در پردازش و تجزیه کد دارد. برای پیاده‌سازی AST، مراحل زیر باید انجام شوند:

در این فاز، لازم است درخت AST (Abstract Syntax Tree) برای کد ورودی تولید شود. در کلاس درباره‌ی چگونگی ساخت AST توضیح داده شده است. توجه کنید که طراحی و ساختار درست AST بسیار مهم است زیرا در تمامی فازهای بعدی نیاز است در این درخت پیمایش کنید و از آن استفاده کنید. توصیه می‌شود برای درک بهتر ساختار آن و نحوه پیاده‌سازی آن به کد SimpleLang مراجعه کنید. همچنین برای پیاده‌سازی آن می‌توانید گام های زیر را طی کنید:

تعریف گره‌های AST

هر عنصر زبانی (مانند دستورات، عبارات، متغیرها و عملگرها) باید به‌عنوان یک گره (Node) در AST نمایش داده شوند. این گره‌ها باید به‌صورت کلاس‌های مجزا تعریف شوند که روابط بین آن‌ها نیز مشخص باشد.

- **ساختار کلاس‌ها**
  - یک کلاس پایه (مثلاً Node) برای تمامی گره‌های AST تعریف شود که گره‌های مخصوص به هر بخش از زبان از این کلاس پایه ارث‌بری کنند (مانند Expression و Statement).
  - گره‌های درخت تعریف شود. این گره‌ها می‌توانند چندین مرحله ارث‌بری داشته باشند. به عنوان مثال گره BinaryExpression از گره Expression ارث‌بری می‌کند و گره Expression از Node ارث‌بری می‌کند (ممکن است بتوان نحوه‌ی ارث‌بری کلاس‌ها را به

انواع دیگری نیز تعریف کرد). همچنین توجه کنید که در زبان جاوا، یک کلاس نمی‌تواند از بیش از یک کلاس به طور همزمان ارث‌بری کند.

- هر گره شامل اطلاعات مربوط به خود (مثلاً نوع عملیات، مقدار متغیرها یا فرزندان) است.

- **روابط بین گره‌های AST**

- هر گره‌ای که شامل زیرگره‌ها (فرزندان) است، باید فیلدهای مشخصی برای ذخیره‌سازی آنها داشته باشد. به‌عنوان مثال، گره‌های مربوط به عملگرهای دوتایی (باینری) باید دو فیلد به نام‌های فرزند چپ و فرزند راست داشته باشند که عملوندهای مربوطه را در خود نگه‌دارند (در این حالت در زبان جاوا، این فیلدها به صورت متغیرهایی از نوع رفرنس هستند و آدرس instance-هایی از کلاس‌های فرزندان در آن‌ها ذخیره می‌شود).

ایجاد گره‌ها در گرامر ANTLR

در این مرحله لازم است در کد ANTLR، کدهای جاوا را جهت ساخته شدن درخت AST اضافه کنید. برای این کار مانند کد simpleLang و کارگاه عمل کنید. در هر قانون گرامر لازم است گره مربوط به آن را بسازید و اطلاعات آن را پر کنید. این اطلاعات می‌توانند به طور مستقیم از توکن‌ها گرفته شوند یا اینکه سایر گره‌های AST باشند که زیر درخت آن هستند. در نهایت گره ساخته شده را بر می‌گردانید. لازم به ذکر است که در بعضی قوانین گرامر تنها کافی است گره زیر درخت را برگردانید.

نحوه ارزیابی

برای این که بتوان درخت AST ساخته شده را ارزیابی کرد لازم است موارد زیر را پیاده‌سازی کنید.

تعداد statement ها برای scope ها

در گره‌هایی که در ادامه گفته می‌شود، تعداد statement-ها را چاپ کنید.

گره‌هایی که لازم است این کار را انجام دهید:

- گره توابع
- گره حلقه for و while
- گره if

به عنوان مثال برای for در مثال زیر:

```
for (int i = 0; i < 10; i++) {
    printf("%d\n", i);
}
```

مقدار ۱ باید چاپ شود که یعنی یک فرزند دارد که گزاره‌ی مربوط به printf است.

توجه کنید که statement-های درون پرانتز که شرایط حلقه را مشخص می‌کنند در نظر گرفته نمی‌شوند و تنها بدنه‌ی for مد نظر است.

برای if به این صورت عمل می‌کنیم که به ترتیب برای بدنه خود if و سپس else if ها و در نهایت برای else چاپ می‌کنیم. مثلا:

```
if (x > 3) {
    printf("x is greater than 3\n");
} else if (x == 3) {
    printf("x is equal to 3\n");
} else {
    printf("x is less than 3\n");
}
```

در خروجی برای هر نود مقدار یک چاپ می‌شود.

توجه کنید که وقتی scope های تو در تو داریم تعداد statement های آن نیز جمع زده می‌شود به عنوان مثال:

```
for (int i = 0; i < 5; i++) {
    int a;
    if (i == 2) {
        int b;
        int c;
    } else {
        int d;
    }
}
```

در این مثال برای حلقه for مقدار ۲ چاپ می‌شود.

- یکی برای تعریف متغیر
- یکی برای خود عبارت if
- توجه کنید که خود statements ممکن است بلوک و بدنه داشته باشند که از شمارش آنها صرف نظر می‌کنیم (یعنی مثلا از تعریف متغیرهای b,c,d در این مثال، زیرا این موارد در زمان محاسبه‌ی statement-های if و else محاسبه می‌شوند).

در نهایت برای شمارش تعداد statement-ها توجه کنید که برای حلقه for (و به طرز مشابه برای if، توابع و while) عبارت های درون پرانتز که شرایط را مشخص می‌کنند، شمارش نمی‌شوند.

~~عمق expression-ها~~

~~گره‌های زیر باید عمق خود را در درخت چاپ کنند:~~

~~گره عبارت‌های دودویی~~

~~گره عبارت‌های unery~~

~~برای محاسبه عمق باید به این صورت عمل کنید:~~

~~برای عبارت‌های دودویی بیشترین عمق (بین عملوند چپ و راست) را یکی اضافه می‌کنیم.~~

~~برای عبارت‌های unery عمق عملوند را به ازای هر عملگر یکی اضافه می‌کنیم.~~

~~برای باقی عبارت‌ها (مثلا متغیر یا مقدار) عمق صفر را در نظر بگیرید.~~

~~در نهایت در گره‌های گفته شده عمق را چاپ کنید.~~

~~به عنوان نمونه داریم:~~

```
a + b++ * 8

// a -> 0
// b -> 0
// 8 -> 0
// b++ -> 0 + 1 = 1
// b++ * 8 -> max(0, 1) + 1 = 2
// a + b++ * 8 -> max(0, 2) + 1 = 3
```

~~بنابراین در خروجی مقدار ۱ برای b++ و مقدار ۲ برای عبارت ضرب و مقدار ۳ برای عبارت جمع را چاپ می‌کند.~~

~~مشخص است که برای عبارت‌هایی که چند عملگر با اولویت یکسان داشته باشند، با توجه به شرکت پذیری عملگر ها ارزیابی می‌شود. مثلا:~~

```
a + 2 - b

// a + 2 -> max(0, 0) + 1 = 1
// a + 2 - b -> max(1, 0) + 1 = 2
// + -> 1
// - -> 2
```

تبدیل C به CPY (ویژه مهندسی کامپیوتر)

در این پروژه، هدف ما تبدیل ساختار نحوی زبان C به یک سبک مشابه پایتون است. در زبان C، محدوده‌ی scope با {} مشخص می‌شود و هر دستور با ; پایان می‌یابد. اما در این پروژه، قصد داریم این ساختار را تغییر دهیم تا خوانایی و سادگی مشابه پایتون را فراهم کنیم. ویژگی‌های تغییر یافته برای ایجاد این تغییر باید موارد زیر را پیاده‌سازی کنید:

- بلوک‌ها و scope-های کد مانند پایتون، با استفاده از فاصله (space) مشخص شوند، نه {}.
- پایان هر خط کد با newline (یعنی \n) تعیین شود، نه ;.
- ابتدای هر بلوک با : شروع شود.
- پایان توابع به جای علامت {} از کلمه‌ی کلیدی end استفاده کند، بنابراین end به کلمات کلیدی زبان اضافه خواهد شد. مانند باقی بلوک ها، بلوک توابع هم با : شروع می شود.

همچنین به نکات زیر توجه کنید:

- در نوشتن شرایط حلقه for همچنان از ; استفاده می‌شود. (به مثال توجه کنید)
- جهت کاهش پیچیدگی پروژه دستورات چند خطی نداریم و هر دستور در یک خط نوشته می‌شود.
- در تست های پروژه Indentation طبق استاندارد PEP8 و برابر ۴ فاصله (space) در نظر گرفته می‌شود.
- پیشنهاد می‌شود این قسمت را با استفاده از یک کلاس مجزا و با استفاده از visitor pattern پیاده‌سازی کنید (در این مورد در کارگاه اول توضیح داده می‌شود).

کد نمونه

برای درک بهتر این موضوع، به مثال زیر توجه کنید که کدی را در C و CPY نمایش می‌دهد.

sample - cpy

```
void printNumbers(int limit):
    for (int i = 1; i <= limit; i++):
        if (i == 1):
            printf("%d is the first number\n", i)
        else if (i == limit):
            printf("%d is the last number\n", i)
        else if (i % 2 == 0):
            printf("%d is even\n", i)
        else:
            printf("%d is odd\n", i)
end

int main():
    int count = 5
    int i = 0

    while (i < count):
        printf("Iteration %d\n", i + 1)
        i++

    printNumbers(count)
    return 0
end
```

sample - c

```
void printNumbers(int limit) {
    for (int i = 1; i <= limit; i++) {
        if (i == 1) {
            printf("%d is the first number\n", i);
        } else if (i == limit) {
            printf("%d is the last number\n", i);
        } else if (i % 2 == 0) {
            printf("%d is even\n", i);
        } else {
            printf("%d is odd\n", i);
        }
    }
}

int main() {
    int count = 5;
    int i = 0;

    while (i < count) {
        printf("Iteration %d\n", i + 1);
        i++;
    }

    printNumbers(count);
    return 0;
}
```

نودهای خروجی

خط به همراه تعداد استیتمنت <- Function

خط به همراه تعداد استیتمنت <- If

خط به همراه تعداد استیتمنت <- Else

خط به همراه تعداد استیتمنت <- Else if

خط به همراه عملگر یا عملوند روت. توجه کنید اگر هیچ عملگر و عملوندی وجود نداشت <- Expression و فقط اکسپرشن داشتیم، باید اولین اکسپرشن ویزیت شود

نکات مهم

- تمامی فایل‌ها و کدهای خود را در GitHub repository مربوط به پروژه قرار دهید.
- بخش تبدیل C به CPY تنها ویژه مهندسی کامپیوتر است. باقی بخش‌ها مشترک است.
- در صورت کشف هرگونه تقلب، نمره صفر لحاظ می‌شود.

- دقت کنید که خروجی‌ها به صورت خودکار تست می‌شوند؛ پس نحوه چاپ خروجی باید عیناً مطابق موارد ذکر شده در تست‌ها باشد. **پیشنهاد می‌شود قبل از پیاده‌سازی بخش ارزیابی تست‌ها را بررسی کنید. برای مشاهده تست ها باید یکبار کد خود را آپلود کنید.**
- توجه کنید که علاوه بر تست های قرار داده شده، تعدادی تست دیگر نیز بعد از پایان مهلت ارسال تکلیف قرار می گیرد و نمره نهایی ترکیبی از نمره ی دو نوع تست است.