# گزارش

**اسم درس**

**۱۴۰۱/۱۱/۴**

- **نام و نام خانوادگی :**
- **نام استاد :**
- **گروه :**

# گزارش کار ۱

## توضیح اجرای برنامه:

برای اجرای برنامه میتوان از تست کیس های تعبیه شده استفاده کرد. سعی شد از مثال های متنوع استفاده شود.

برای اجرای برنامه ، کافیست فایل main را اجرا کنیم.

اجرای برنامه دو حالت مختلف دارد ، اگر get_input ، True باشد ، هر بار از کاربر یک ورودی جدید میگیرد. در غیر این صورت از لیست expressions ، یکی یکی عبارت ها را دریافت و انجام میدهد.

## کتابخانه های مورد نیاز:

۱. regex : این تابع شباهت به کتابخانه ی re از زبان خود پایتون دارد ولی با این تفاوت که قوی تر بوده و میتواند اعمال recursive را انجام دهد.

- این کتابخانه از طریق pip قابل نصب است. کتابخانه های دیگر ، برای خود زبان هستند و نیاز به نصب ندارند.

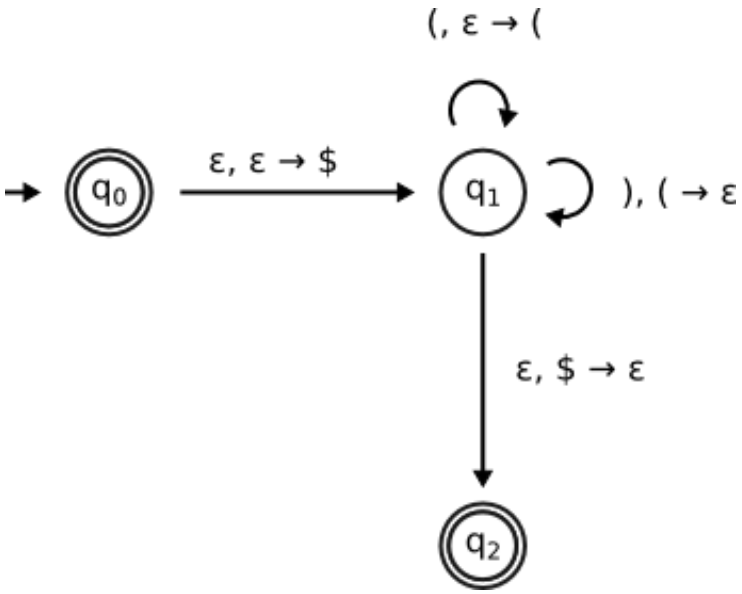
# گزارش کار

## روند انجام کار:

ابتدا برای انجام این پروژه ، زبان پایتون به دلیل راحتی ، دسترسی بیشتر و داکیومنت های فراوانی که وجود دارد انتخاب شد.

پس از بررسی فاز اول و مرور درس نظریه ی زبان ، یک کلاس برای ماشین پشته ای طراحی شد. با ماشین حالات متناهی قادر به بررسی درست بودن عبارات ریاضی نیستیم زیرا این ماشین برای پیدا کردن الگو هایی در یک رشته استفاده میشود نه چک کردن درستی یک عبارت.

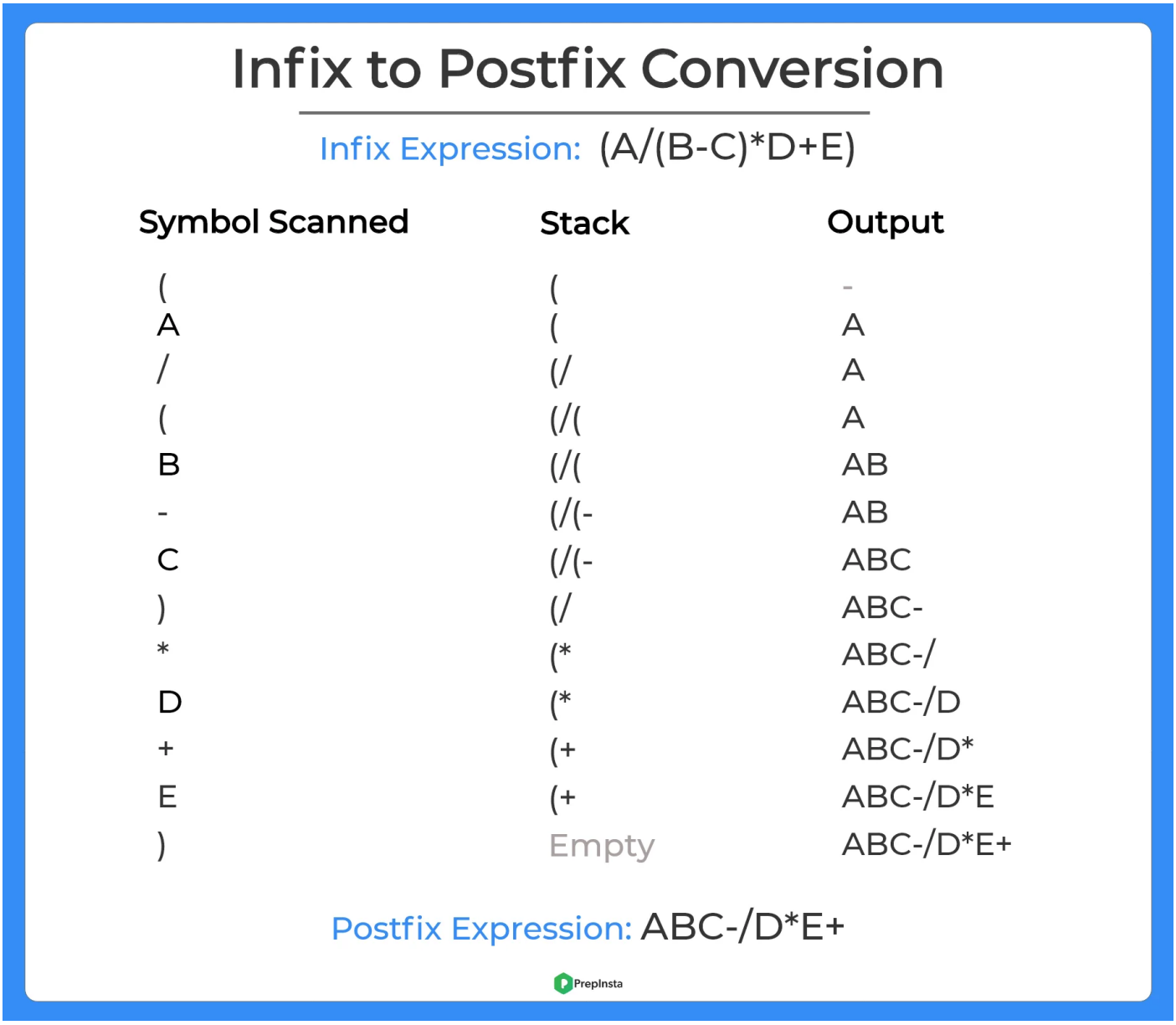
سپس بر اساس آن ، یک تابع برای انجام فاز اول در نظر گرفته شد. این تابع ابتدا درستی عبارت را به دو روش که در توابع checker1 و checker2 هستند چک میکرد و سپس تمامی توکن های مورد نیاز را با استفاده از regex پیدا میکرد. روش اول استفاده از یک ماشین پشته ای با حالات آن و روش دوم استفاده از regex است. در این روش ها ، بسته و باز شدن و مچ بودن پرانتز ها چک میشد.



## گزارش کار

برای فاز دوم ، باز از ماشین پشته ای استفاده شد ، که در این فاز ، عبارتی که ما به صورت infix مینویسیم را به صورت postfix با استفاده از stack انجام میداد.

همچنین از توابعی برای pop ، insert و replace برای کلاس ماشین استفاده کردیم که مقدار پشته را عوض میکردند.



## گزارش کار

سپس برای فاز سوم ، باز از ماشین پشته ای استفاده شد که عبارتی را به صورت postfix گرفته و به صورت infix در آورد. و در آخر نیز جواب حاصل از آن را چاپ کند.

Postfix expression : ABC/-AD/E-*

| Expression | Stack |
|---|---|
| ABC/-AD/E-* | A |
| BC/-AD/E-* | B, A |
| C/-AD/E-* | C, B, A |
| /-AD/E-* | (B/C), A |
| -AD/E-* | (A-(B/C)) |
| AD/E-* | A, (A-(B/C)) |
| D/E-* | D, A, (A-(B/C)) |
| /E-* | (A/D), (A-(B/C)) |
| E-* | E, (A/D), (A-(B/C)) |
| -* | ((A/D)-E), (A-(B/C)) |
| * | ((A-(B/C))*((A/D)-E)) |

*Therefore, Infix expression : ((A-(B/C))*((A/D)-E))*

# گزارش کار

## متغییر های مهم کلاس PushDownAutomata:

۱. PRECEDENCE:

این دیکشنری ، اولویت بندی عملیات ها را نگه داری میکند. هر چه مقدار عدد آن بیشتر باشد ، اولویت بیشتری دارد. مثلا ضرب از همه بالا تر است ، پس ابتدا آن بررسی میشود.

۲. stack:

پشته ای که برای این کلاس در نظر گرفته شده است. بین هر فاز ، باید پشته را خالی کرد.

۳. Postfix:

لیستی که برای قرار دادن عبارت های postfix ساخته شده است.

## توابع مهم کلاس PushDownAutomata:

۱. Is_accepted:

برای اجرای فاز اول و بررسی بالانس بودن پرانتز ها

۲. To_postfix و to_infix:

گرفتن postfix و تبدیل آن به infix یا برعکس با استفاده از ماشین pda

۳. Evaluate:

اجرای عملیات ریاضی روی رشته ی ورودی

# گزارش کار

## تست کیس ها:

برخی نتایج در فایل result.txt قابل مشاهده هستند.