درس طراحی زبان‌های برنامه‌سازی

فاز دوم پروژه

دانشکده مهندسی کامپیوتر

دانشگاه صنعتی شریف

نیم سال دوم ۹۹ـ۹۸

استاد:
شیرین بقولی‌زاده

مهلت ارسال:
۳۱ تیر
ساعت ۲۳:۵۹

به موارد زیر توجه کنید:

* مهلت ارسال تمرین ساعت ۲۳:۵۹ روز ۳۱ تیر است.
* پروژه تحویل حضوری خواهد داشت.
* در نهایت تمام فایل‌های خود را در یک فایل زیپ قرار داده و با نام $P2\_StudentID$ آپلود کنید.
* در مجموعِ تمامی تمارین و پروژه ۱۴ روز مهلت تاخیر مجاز دارید و پس از تمام شدن این تاخیرهای مجاز به ازای هر روز ۲۵ درصد از کل نمره‌ی پروژه‌ی شما کم می‌شود.
* پروژه را با یکی از زبان‌های eopl یا racket پیاده‌سازی کنید.
* لطفا هیچ کدی را از یکدیگر کپی نکنید. در صورت وقوع چنین مواردی مطابق با سیاست درس رفتار می‌شود.

۱ تغییر گرامر

در این پروژه ما قصد داریم یک مفسر برای یک زبان ساده طراحی کنیم. گرامر این زبان به شکل زیر است:

$$COMMAND \rightarrow UNITCOM \mid COMMAND; UNITCOM$$
$$UNITCOM \rightarrow WHILECOM \mid IFCOM \mid ASSIGN \mid RETURN$$
$$WHILECOM \rightarrow while\ EXP\ do\ COMMAND\ end$$
$$IFCOM \rightarrow if\ EXP\ then\ COMMAND\ else\ COMMAND\ endif$$
$$ASSIGN \rightarrow variable = EXP \mid variable = FUNCTION \mid variable = CALL$$
$$RETURN \rightarrow return\ EXP$$
$$EXP \rightarrow AEXP \mid AEXP > AEXP \mid AEXP < AEXP \mid AEXP == AEXP \mid AEXP != AEXP$$
$$AEXP \rightarrow BEXP \mid BEXP - AEXP \mid BEXP + AEXP$$
$$BEXP \rightarrow CEXP \mid CEXP * BEXP \mid CEXP / BEXP$$
$$CEXP \rightarrow -CEXP \mid (EXP) \mid posNumber \mid null \mid variable \mid true \mid false \mid string \mid LIST \mid variable\ LISTMEM$$
$$LIST \rightarrow [LISTVALUES] \mid [\,]$$
$$LISTVALUES \rightarrow EXP \mid EXP, LISTVALUES$$
$$LISTMEM \rightarrow [EXP] \mid [EXP]\ LISTMEM$$
$$FUNCTION \rightarrow func(VARS)\ \{COMMAND\}$$
$$VARS \rightarrow variable \mid variable, VARS$$
$$CALL \rightarrow variable(ARGS)$$
$$ARGS \rightarrow EXP \mid EXP, ARGS$$

توجه کنید که func ترمینال است و خود این کلمه برای تعریف تابع استفاده می‌شود. همچنین برای راحتی گرامر تابع بدون ورودی نداریم.

۲ فاز ۲

۱.۲ ورودی و خروجی

در این فاز همانند فاز قبل باید یک تابع داشته باشید که با گرفتن آدرس برنامه ورودی، آن را اجرا کرده و خروجی مطلوب را نمایش دهد.

۲.۲ پیاده‌سازی تابع و فراخوانی آن (۴۵ نمره)

در این بخش شما باید بتوانید یک تابع را در یک متغیر ذخیره کنید. همانطور که از گرامر مشخص است، تابع تعدادی ورودی می‌گیرد. در بدنه‌ی تابع تعدادی UNITCOM اجرا خواهد شد. متغیرهای درون تابع محلی هستند. بنابراین با تعریف یک متغیر جدید، در خارج تابع به آن مقدار دسترسی نداریم. اما در خود تابع به متغیرهای جهانی (گلوبال) دسترسی داریم. دقت کنید که متغیرهای جهانی هر تابع هنگام تعریف تابع مشخص می‌شوند. به مثال‌های زیر توجه کنید:

ورودی نمونه

```
a = 2;
f = func(b) {
 return a
};
a = 3;
b = f(2);
return b
```

خروجی نمونه

```
2
```

ورودی نمونه

```
f = func(b) {
 return a
};
a = 3;
b = f(2);
return b
```

خروجی نمونه

```
error
```

دلیل آن است که هنگام تعریف تابع، متغیر a وجود نداشته است.

ورودی نمونه

```
f = func(b) {
 a = 3;
 return a
};
b = f(2);
return a
```

خروجی نمونه

```
error
```

دلیل آن است که متغیر a وجود ندارد.
حال هنگام اجرای دستورات درون بدنه‌ی تابع هرگاه به return برسیم، اجرای برنامه متوقف شده و مقدار مورد نظر برگردانده می‌شود.

ورودی نمونه

```
f = func(b) {
 a = b;
 return a;
 while true do a = a + 1 end
};
b = f(3);
return b
```

خروجی نمونه

```
3
```

۳.۲ خاصیت بازگشتی توابع (۲۰ نمره)

در این بخش شما باید ذخیره‌ی تابع را به نحوی انجام دهید که بتوان درون یک تابع خودش را صدا زد. به مثال زیر توجه کنید.

ورودی نمونه

```
listmaker = func(a, b) {
 if a == 0 then return [] else a = listmaker(a-1, b); return a + [b] endif
};
b = listmaker(3, 5);
return b
```

خروجی نمونه

```
[5, 5, 5]
```

خروجی می‌تواند نمایشی دیگر از همین لیست باشد.

۴.۲ Lazy Evaluation (۳۵ نمره)

در این بخش شما باید Lazy Evaluation را پیاده‌سازی کنید که در زبان ما شامل موارد زیر می‌شود:

- در هنگام * کردن، در صورتی که سمت چپ ضرب ۰ بود، سمت راست محاسبه نشده و مقدار ۰ برگردانده می‌شود. همچنین اگر سمت چپ ضرب false بود، بدون محاسبه‌ی سمت راست، مقدار false برگردانده شود.
- یک متغیر که در assign مقدار دهی می‌شود، تا وقتی که از آن استفاده نشود، محاسبه نشود.
- تا وقتی از یک ورودی تابع استفاده نشده است، آن مقدار محاسبه نمی‌شود.

ورودی نمونه

```
f = func(b) {
 return 8
};
a = f(2/0)
b = 2 / 0;
return a
```

خروجی نمونه

```
5
```

۵.۲ کتابخانه‌ی اولیه (نمره‌ی اضافی)

در این بخش شما باید یک کتابخانه‌ی اولیه برای استفاده‌ی راحت از این زبان پیاده‌سازی کنید. توابع مورد نظر باید در ابتدای اجرای برنامه به شکل توابع گلوبال وارد Environment شوند و بتوان در طول برنامه از آن‌ها استفاده کرد. این توابع عبارتند از:

- $pow(a, b)$: دو عدد a و b را گرفته و a را به توان b می‌رساند.
- $make\_list(a, b)$: یک لیست به طول a با مقادیر برابر b خروجی می‌دهد. در صورتی که a کوچکتر مساوی ۰ بود، لیست خالی برگردانده شود.
- $reverse(a)$: یک لیست می‌گیرد و تنها یک لایه از آن را برعکس می‌کند.

ورودی نمونه

```
return reverse([1, [2, 3], [4, [5, 6]]])
```

خروجی نمونه

```
[[4, [5, 6]], [2, 3], 1]
```

- $reverse\_all(a)$: یک لیست می‌گیرد و خود آن و تمام لیست‌های فرزند، نوه و ... را به صورت بازگشتی برعکس می‌کند.

ورودی نمونه

```
return reverse_all([1, [2, 3], [4, [5, 6]]])
```

خروجی نمونه

```
[[[6, 5], 4], [3, 2], 1]
```

- $set(a, index, value)$: یک لیست a را می‌گیرد و مقدار خانه index ام آن را برابر value می‌کند. در صورتی که طول a کوچکتر از مقدار مورد نظر بود، خود خطا برگردانده شده و برنامه متوقف شود. لیست‌ها را صفرمبنا (zero-based) بگیرید.
- $merge(a, b)$: دو لیست مرتب a و b را گرفته و مرج شده آن‌ها را برگرداند.
- $merge\_sort(a)$: لیست a را به روش mergeSort مرتب کند و برگرداند.
- $eval(s)$: با گرفتن رشته s آن را به عنوان یک برنامه‌ی جدید درنظر گرفته و اجرا می‌کند. دقت کنید که در هنگام اجرای این رشته نباید متغیری در Environment به جز همین توابع کمکی اولیه وجود داشته باشد. درواقع آن رشته یک برنامه‌ی جدا درنظر گرفته می‌شود.

این توابع را می‌توانید به زبان رکت یا زبان مفسر خودتان پیاده کنید.