فاز اول پروژه

کامپایلر

دکتر دلدار

نحوه تحویل:

فایل های مورد نظر را در zip که با نام، شماره دانشجویی و شماره فاز نام گذاری شده است

(مانند Ph1_LastName_40018173) در سامانه یکتا بارگذاری کنید.

طراحی و ساخت کامپایلر Ciut

فاز یک (scanner)

این پروژه شامل سه بخش اصلی است که هر بخش به یکی از مراحل ساخت کامپایلر Ciut می‌پردازد:

۱. **scanner (تحلیلگر لغوی):** در این بخش، باید مجموعه‌ای از توکن‌ها را برای زبان Ciut تعریف کرده و یک DFA برای تشخیص هر توکن رسم کنید. سپس با ترکیب این DFA ها، برنامه‌ای به زبان پایتون بنویسید که یک فایل متنی شامل کد Ciut را به عنوان ورودی دریافت کرده و توکن‌های آن را به عنوان خروجی ارائه دهد.

۲. **تجزیه کننده(parser):** این بخش به تجزیه نحوی کد Ciut و ساخت درخت تجزیه می‌پردازد.

۳. **تولید کننده کد میانی:** در این بخش، کد Ciut به کد میانی تبدیل می‌شود.

نکات مهم

- هر بخش از پروژه یک مرحله از کامپایلر را پیاده سازی می‌کند و خروجی هر بخش به عنوان ورودی بخش بعدی استفاده می‌شود.
- شما می‌توانید از کدهای موجود در کتاب‌های درسی با ذکر منبع استفاده کنید، اما استفاده از کد از اینترنت یا سایر دانشجویان اکیدا ممنوع است و منجر به نمره مردودی خواهد شد.

مشخصات کامپایلر

کامپایلری که در این پروژه پیاده‌سازی می‌کنید باید دارای ویژگی‌های زیر باشد:

- کامپایلر شامل سه ماژول اصلی برای تحلیلگر لغوی، تحلیلگر نحوی و تولید کد میانی است.
- برنامه ورودی فقط یک بار خوانده می‌شود و سه بخش اصلی کامپایلر به صورت (pipeline) با یکدیگر کار می‌کنند.

- ورودی کامپایلر (از طریق فایلی به نام 'input.txt') یک فایل متنی شامل یک برنامه Ciut است که باید ترجمه شود.
- در هر یک از سه فاز پروژه ، شما یکی از سه ماژول اصلی کامپایلر را پیاده‌سازی می‌کنید. در نهایت، خروجی این فازهای جداگانه باید در یک کامپایلر واحد ترکیب شود.
- وابستگی شدیدی بین فازهای یک تا سه پروژه وجود دارد. به عنوان مثال، فاز دوم کاملاً وابسته به فاز اول (اسکنر) است. به عبارت دیگر، تجزیه‌کننده شما نمی‌تواند به درستی کار کند مگر اینکه یک اسکنر داشته باشید که بتواند برنامه ورودی را به درستی توکن‌بندی کند. به طور مشابه، تولید کننده کد میانی شما بدون یک اسکنر یا تجزیه‌کننده درست کار نخواهد کرد

نتایج اسکنر

همانطور که قبلاً ذکر شد، اسکنر یک فایل متنی شامل یک برنامه Ciutرا دریافت می‌کند و فرم توکن‌شده برنامه را خروجی می‌دهد. اسکنر باید تابعی به نام get_next_token داشته باشد که پس از هر فراخوانی، در برنامه ورودی (character by character) پیش می‌رود تا زمانی که یک توکن معتبر شناسایی شود. سپس تابع، توکن شناسایی شده را به صورت یک زوج و به شکل زیر برمی‌گرداند:

```
(Token Type, Token String)
```

برای این پروژه، شما همچنین باید یک حلقه while بنویسید که به طور مکرر get_next_token را فراخوانی می‌کند تا زمانی که همه توکن‌های برنامه ورودی Ciut شناسایی شوند. توکن‌ها باید در یک فایل متنی به نام tokens.txt ذخیره شوند، جایی که هر خط دارای یک شماره است که مربوط به شماره خط در فایل ورودی است، و یک دنباله از زوج‌های توکن که در آن خط شناسایی شده‌اند. لطفاً توجه داشته باشید که این حلقه while فقط برای این پروژه مورد نیاز است. در پروژه بعدی، تابع get_next_token توسط تجزیه‌کننده هر زمان که یک توکن لازم باشد فراخوانی می‌شود و این حلقه while دیگر لازم نخواهد بود.

شما می‌توانید فرض کنید که ورودی فقط شامل کاراکترهای ASCII (که در UTF-8 نشان داده می‌شوند) است. اسکنر شما باید برای هر ورودی درست کار کند. به عنوان مثال، نباید لازم باشد که همه توکن‌ها با یک کاراکتر فاصله از هم جدا شوند. برای مثال، باید بتواند مجموعه یکسانی از توکن‌ها را برای هر دو ورودی زیر تشخیص دهد:

```
if(b==3){a=3;}else{b=4;}
if ( b == 3 ) { a = 3 ; } else { b = 4 ; }
```

اسکنر شما همچنین باید خطاهایی مانند ظاهر شدن یک کاراکتر غیرقانونی در جریان ورودی (به جز کامنت ها، که ممکن است شامل هر کاراکتری باشند)، یا یک عدد اشتباه مانند "۱۲۵d" را مدیریت کند. شما همچنین باید مقدماتی را برای خاتمه در صورت بروز یک fatal error فراهم کنید. استثناهای مدیریت نشده قابل قبول نیستند.

همه انواع توکن که اسکنر شما باید قادر به تشخیص آنها باشد در جدول زیر نشان داده شده است:

Token Type	Description
NUM	Any string matching: [0-9]+
ID	Any string matching: [A-Za-z][A-Za-z0-9]*
KEYWORD	if, else, void, int, while, break, return
SYMBOL	; : , [ ] ( ) { } + - * = < == /
COMMENT	Any string between a /* and a */[1]
WHITESPACE	blank (ASCII 32), \n (ASCII 10), \r (ASCII 13), \t (ASCII 9), \v (ASCII 11), \f (ASCII 12)

مدیریت خطا

خطاهای لغوی احتمالی با روش Panic Mode بازیابی می‌شوند. در این روش، هنگامی که یک خطای لغوی شناسایی می‌شود، اسکنر به طور متوالی کاراکترهای ورودی باقی‌مانده را حذف می‌کند تا زمانی که یک توکن درست پیدا شود. کاراکترهایی که برای توکن فعلی پردازش شده‌اند تا نقطه‌ای که خطا شناسایی شده است نیز دور انداخته می‌شوند (به مثال‌های ورودی-خروجی مراجعه کنید). خطاهای لغوی باید در یک فایل متنی به نام lexical_errors.txt ثبت شوند. هر خطا در یک خط جداگانه به عنوان یک زوج شامل یک رشته (یعنی رشته‌ای از کاراکترهایی که توسط اسکنر دور انداخته شده‌اند) و یک پیام به همراه شماره خط مربوطه که خطا در آن رخ داده است، گزارش می‌شود. اگر برنامه ورودی از نظر لغوی صحیح باشد، جمله "هیچ خطای لغوی وجود ندارد." باید در lexical_errors.txt نوشته شود.

- هنگامی که یک کاراکتر نامعتبر (کاراکتری که نمی‌تواند هیچ توکنی را شروع کند) برخورد شود، یک رشته حاوی فقط آن کاراکتر و پیام "ورودی نامعتبر" باید در lexical_errors.txt ذخیره شود. اسکنر از کاراکتر بعدی از سر گرفته می‌شود.

- اگر یک کامنت هنگام رسیدن به انتهای فایل ورودی باز بماند، این خطا فقط با پیام "کامنت بسته نشده" ثبت شود. در این نوع خطاها، یک رشته طولانی (یعنی کامنت بسته نشده) ممکن است توسط اسکنر دور انداخته شود. با این حال، چاپ حداکثر هفت کاراکتر اول کامنت بسته نشده با سه نقطه (...) کافی است. (به آخرین مثال در شکل ۳ مراجعه کنید).
- اگر یک "*/" در خارج از یک کامنت دیده شود، اسکنر باید این خطا را به عنوان "کامنت نامطابق" گزارش کند، نه اینکه آن را به عنوان * و / توکن‌بندی کند.
- اگر "d۱۲۵" را ببینید، باید این خطا را به عنوان "عدد نامعتبر" گزارش کنید، نه اینکه آن را به عنوان یک NUM و یک ID توکن‌بندی کنید.
- اسکنر باید توکن‌ها را حداکثر با یک کاراکتر lookahead تشخیص دهد. برای مثال، برای تشخیص همه توکن‌های SYMBOL به جز "="، نیازی به خواندن هیچ کاراکتر lookahead نیست. در مورد "="، باید فقط یک کاراکتر lookahead را بخوانید تا بتوانید "=" را از "==" تشخیص دهید.

توجه داشته باشید که اسکنرها به طور کلی فقط کلاس بسیار محدودی از خطاها (یعنی خطاهای لغوی) را تشخیص می‌دهند. بنابراین، سعی نکنید خطاهایی را که فراتر از سطح لغوی هستند بررسی کنید. برای مثال، نباید بررسی کنید که آیا یک متغیر قبل از استفاده از آن تعریف شده است یا خیر، که به عنوان یک خطای معنایی در نظر گرفته می‌شود.

جدول نمادها

برنامه‌ها تمایل دارند که تعداد زیادی تکرار از یک شناسه (identifier) داشته باشند. اسکنرها معمولاً کلمات کلیدی و شناسه‌ها را در جدولی به نام "جدول نمادها" ذخیره می‌کنند. برای هر شناسه و کلمه کلیدی، یک سطر در جدول نمادها وجود خواهد داشت. به طور کلی، وظیفه اسکنر این است که یک سطر جدید برای هر شناسه جدید (یعنی زمانی که یک شناسه برای اولین بار دیده می‌شود) به جدول نمادها اضافه کند و lexeme شناسه را در آن سطر ذخیره کند. سایر اطلاعات مانند نوع و تعداد آرگومان‌های شناسه‌ها توسط ماژول‌های دیگر (به عنوان مثال، تولید کننده کد میانی) به جدول اضافه می‌شوند. جدول نمادها به طور معمول در ابتدای کامپایل با کلمات کلیدی مقداردهی اولیه می‌شود. اسکنر شما باید جدول نماد خود را در یک فایل متنی خروجی به نام symbol_table.txt ذخیره کند.

نکات دیگر

اسکنر شما باید متغیری (به عنوان مثال، linenumber) را نگه دارد که نشان می‌دهد کدام خط از متن ورودی در حال اسکن شدن است. این ویژگی به ماژول‌های مختلف کامپایلر در چاپ پیام‌های خطا کمک می‌کند. این متغیر هر زمان که اسکنر یک نماد خط جدید \n را می‌خواند تغییر می‌کند (توجه داشته باشید که \f مقدار linenumber را تغییر نمی‌دهد. همچنین کامنت‌های یک خطی را نیز پایان نمی‌دهد). اگر متن ورودی بخشی از یک کامنت باشد، ممکن است شامل هر کاراکتری باشد. علاوه بر این، نیازی به ثبت یا گزارش توکن‌های COMMENT و WHITESPACE نیست.

نمونه‌های ورودی-خروجی

توجه داشته باشید که اسکنر شما روی تعدادی از موارد آزمایشی که جنبه‌های مختلف اسکنر را بررسی می‌کنند، آزمایش خواهد شد. شکل‌های زیر یک نمونه ساده از چنین مورد آزمایشی و خروجی مورد انتظار را نشان می‌دهند. توجه داشته باشید که در خطوط ۹ و ۱۴ از شکل ۲ (یعنی tokens.txt)، 'e' و '۲' به ترتیب به عنوان نتیجه بازیابی خطای Panic Mode شناسایی شده‌اند.

lineno	code
1	void main (void) {
2	int a = 0;
3	/* comment1*/
4	a = 2 + 2;
5	a = a - 3;
6	cde = a;
7	if (b /* comment2 */ == 3d) {
8	a = 3;
9	cd!e = 7;
10	}
11	else */
12	{
13	b = a < cde;
14	{cde = @2;
15	}}
16	return; /* comment 3
17	}

شکل ۱

lineno	Recognized Tokens
1	(KEYWORD, void) (ID, main) (SYMBOL, () (KEYWORD, void) (SYMBOL,)) (SYMBOL, {)
2	(KEYWORD, int) (ID, a) (SYMBOL, =) (NUM, 0) (SYMBOL, ;)
4	(ID, a) (SYMBOL, =) (NUM, 2) (SYMBOL, +) (NUM, 2) (SYMBOL, ;)
5	(ID, a) (SYMBOL, =) (ID, a) (SYMBOL, -) (NUM, 3) (SYMBOL, ;)
6	(ID, cde) (SYMBOL, =) (ID, a) (SYMBOL, ;)
7	(KEYWORD, if) (SYMBOL, () (ID, b) (SYMBOL, ==) (SYMBOL,)) (SYMBOL, {)
8	(ID, a) (SYMBOL, =) (NUM, 3) (SYMBOL, ;)
9	(ID, e) (SYMBOL, =) (NUM, 7) (SYMBOL, ;)
10	(SYMBOL, })
11	(KEYWORD, else)
12	(SYMBOL, {)
13	(ID, b) (SYMBOL, =) (ID, a) (SYMBOL, <) (ID, cde) (SYMBOL, ;)
14	(SYMBOL, {) (ID, cde) (SYMBOL, =) (NUM, 2) (SYMBOL, ;)
15	(SYMBOL, }) (SYMBOL, })
16	(KEYWORD, return) (SYMBOL, ;)

شکل ۲

lineno	Error Message
7	(3d, Invalid number)
9	(cd!, Invalid input)
11	(*/, Unmatched comment)
14	(@, Invalid input)
16	(/* comm..., Unclosed comment)

شکل ۳

no	lexeme
1	break
2	else
3	if
4	int
5	while
6	return
7	void
8	main
9	a
10	cde
11	b
12	e

شکل ۴

مشخصات زبان برنامه‌نویسی Ciut

زبان Ciut یک زبان سطح پایین است که شباهت زیادی به C دارد اما ساده‌تر شده و برخی ویژگی‌های سطح بالای C را ندارد. هدف این زبان این است که دانشجویان بتوانند برای آن یک کامپایلر طراحی کنند و نحوه‌ی ترجمه‌ی کدهای سطح بالا به کدهای ماشین را درک کنند.

ویژگی‌های کلی زبان:

۱. **انواع داده‌ای :** زبان **Ciut** دارای انواع داده‌ای ساده‌ای است، از جمله:

- int عدد صحیح ۳۲ بیتی
- bool مقدار true یا false
- void برای توابع بدون مقدار بازگشتی

۲. **عملگرها:**

- +, -, *, / برای عملیات عددی
- &&, ||, ! برای عملیات منطقی
- ==, !=, <, >, <=, >= برای مقایسه

۳. **ساختارهای کنترلی:**

- if, else برای شرط‌ها
- while برای حلقه‌ها
- return برای خروج از توابع

۴. **تعریف توابع:**

توابع به صورت returnType functionName(parameters) {} تعریف می‌شوند، مثلا:

```
int add(int a, int b) {
   return a + b;
}
```

۵. **متغیرها و تخصیص مقدار:**

```
int x = 10;
bool y = true;
```

۶. **عدم پشتیبانی از ویژگی‌های پیچیده‌ترC :**

- اشاره‌گرها(& , *) در این زبان وجود ندارند.
- structو classنداریم، فقط متغیرهای ساده تعریف می‌شوند.
- تخصیص حافظه پویا (malloc, free) پشتیبانی نمی‌شود.

۷. **نمونه‌ای از یک برنامه در زبان Ciut :**

```
int factorial(int n) {
   int result = 1;
   while (n > 1) {
      result = result * n;
      n = n - 1;
   }
   return result;
}
```

```
int main() {
   int x = 5;
   int y = factorial(x);
   return y;
}
```

موارد تحویلی

قبل از ارسال، لطفاً اطمینان حاصل کنید که موارد زیر را انجام داده‌اید:

- مسئولیت شماست که اطمینان حاصل کنید نسخه نهایی ارسالی شما هیچ دستور چاپ برای debug ندارد و مشخصات لغوی شما کامل است.
- شما باید فایلی به نام compiler.py ارسال کنید که در این مرحله فقط شامل کد پایتون اسکنر شما باشد. لطفاً نام کامل و شماره دانشجویی خود و هر مرجعی که ممکن است استفاده کرده باشید را به عنوان کامنت در ابتدای برنامه خود بنویسید.
- مسئولیت نشان دادن اینکه مباحث پروژه را فهمیده‌اید بر عهده شماست. کد مبهم تأثیر منفی بر نمره شما خواهد داشت، بنابراین وقت اضافی بگذارید تا کد خود را خوانا کنید.
- اسکنر شما با اجرای خط فرمان python compiler.py در سیستم عامل اوبونتو با استفاده از مفسر پایتون نسخه ۳.۱۲ آزمایش خواهد شد. این یک نصب پیش‌فرض از مفسر بدون هیچ کتابخانه اضافی به جز anytree است که ممکن است برای فازهای بعدی مورد نیاز باشد. هیچ تابع کتابخانه پایتون اضافی دیگری را نمی‌توان برای این یا پروژه‌های فازهای بعدی استفاده کرد. لطفاً اطمینان حاصل کنید که اسکنر شما به درستی در محیط ذکر شده و با دستور داده شده قبل از ارسال کد شما کامپایل شده است. این مسئولیت شماست که اطمینان حاصل کنید که کد شما به درستی با استفاده از سیستم عامل و مفسر پایتون ذکر شده کار می‌کند.
- کدهای ارسالی با استفاده از چندین مورد آزمایشی مختلف (یعنی چندین فایل input.txt) با خطا و بدون خطاهای لغوی آزمایش و نمره‌دهی می‌شوند. اسکنر شما باید input.txt را از همان پوشه کاری که اسکنر شما (یعنی compiler.py) در آن قرار دارد، بخواند. لطفاً توجه داشته باشید که در صورت دریافت خطای کامپایل یا زمان اجرا برای یک مورد آزمایشی، نمره صفر برای اسکنر شما برای آن مورد آزمایشی اختصاص

داده می‌شود. به طور مشابه، اگر اسکنر نتواند خروجی‌های مورد انتظار (یعنی tokens.txt، symbol_table.txt و lexical_errors.txt) را برای یک مورد آزمایشی تولید کند، نمره صفر برای آن مورد آزمایشی به آن اختصاص داده می‌شود. بنابراین، توصیه می‌شود که اسکنر خود را روی چندین مورد آزمایشی تصادفی مختلف قبل از ارسال کد خود آزمایش کنید.

- تصمیم گیری در مورد اینکه آیا تابع اسکنر در compiler.py گنجانده شود یا به عنوان یک فایل جداگانه مانند مثلاً Scanner.py باشد، بر عهده شماست. با این حال، همه فایل‌های مورد نیاز باید در همان دایرکتوری compiler.py قرار داشته باشند. بنابراین، همه فایل‌های ارسالی شما را در همان دایرکتوری قرار داده می شود و python3 compiler.py اجرا می شود. شما باید فایل‌های برنامه خود (compiler.py و هر فایل متنی دیگری که برنامه شما ممکن است به آن نیاز داشته باشد) را ارسال کنید.

شایان ذکر است که مجدداً، همانطور که در بخش ۲ ذکر شد، همه فاز های بعدی پروژه به نتیجه این فاز وابسته خواهند بود. بنابراین، اجرای اسکنر در این مرحله اولیه بسیار مهم است. در غیر این صورت، باید این فاز را بعداً (بدون هیچ نمره ای) به خاطر فازهای بعدی انجام دهید :)

هرگونه سوال یا ابهامی داشتید می توانید به دو آیدی زیر در تلگرام پیام دهید

@falakian_ali

@M_s_m1984