
بسم الله الرحمن الرحيم

دانشگاه صنعتی اصفهان

دانشکده مهندسی برق و کامپیوتر

پروژه درس کامپایلر

طراحی کامپایلر برای زبان برنامه نویسی Xlang

استاد درس

دکتر زینب زالی

ترم ۴۰۰۲

فهرست مطالب

تعریف پروژه

در این پروژه قصد داریم یک کامپایلر برای زبانی به نام $Xlang$ طراحی و پیاده سازی کنیم. $Xlang$ یک زبان دستوری ساده شبیه به C یا $Pascal$ است. پیاده سازی این کامپایلر با استفاده از دو ابزار $Bison$ و $Flex$ انجام می شود و کامپایلر هدف باید بتواند یک فایل حاوی کد نوشته شده به زبان $Xlang$ را دریافت کرده و با در نظر گرفتن سه بخش تحلیل لغوی[1]، نحوی[2] و معنایی [3] و دیگر مفاهیم لازم که در درس کامپایلر مطالعه خواهید کرد یک کد خروجی به زبان اسمبلی تولید نماید.

هدف از طراحی این پروژه این است که کامپایلر مذکور را قدم به قدم و همگام با مطالبی که در درس می‌آموزید پیاده سازی کنیم تا با جنبه‌های عملی نوشتن یک کامپایلر ابتدایی، آشنا شوید.

[1] Lexical Analysis
[2] Syntax Analysis
[3] Semantic Analysis

فصل اول

تحلیلگر لغوی

۱ـ۱ هدف

در این فاز از پروژه می‌بایست یک تحلیلگر لغوی را با استفاده از ابزار $flex$ نوشته و در محل مشخص شده درون سامانه آپلود کنید.

جهت ارزیابی، یک فایل حاوی قطعه کدی به زبان $Xlang$ که در ادامه توصیف خواهد شد به تحلیلگر لغوی شما داده می‌شود، در صورتی که کد برنامه قواعد لغوی زبان برنامه نویسی $Xlang$ را رعایت کرده باشد، شما باید در خروجی، توکن‌های آن برنامه را چاپ کنید و در غیر این صورت ، بعد از مواجه شدن با خطا، بدون تولید هرگونه توکنی، می‌بایست خطای مناسب چاپ گردد.

۱ـ۲ حساسیت به حروف کوچک و بزرگ

تمام کلمات کلیدی در زبان $Xlang$ با حروف کوچک نوشته می‌شوند. کلمات کلیدی و شناسه‌ها [1] حساس به حروف کوچک و بزرگ هستند [2] مثلاً `if` یک کلمه کلیدی هست ولی `IF` نام یک متغیر است یا به طور مثال `foo` و `Foo` دو نام متفاوت برای اشاره به دو متغیر متفاوت هستند.

[1] Identifiers
[2] Case-Sensitive

۱ـ۳ کلمات کلیدی

در زبان $Xlang$ کلمات کلیدی شامل موارد زیر است :

```
boolean    break    callout    class    continue    else    for    if
false      return   true       void     int
```

۱ـ۴ متغیرها

در زبان $Xlang$ متغیرها [۱] ترکیبی از حروف، اعداد انگلیسی و خط تیره [۲] هستند که حتما باید با یک حرف و یا خط تیره آغاز شوند و هیچ متغیری نمی‌تواند با عدد آغاز شود.

۱ـ۵ کامنت

کامنت‌ها با // شروع می‌شوند و با پایان خط [۳] خاتمه می‌یابند.
توجه ! اصولاً کامنت ها به وسیله preprocessor پردازش می‌شوند و کامپایلر وظیفه پردازش آنها را ندارد، اما چون در این پروژه ، preprocessor وجود ندارد باید به وسیله‌ی تحلیلگر لغوی پردازش شوند.

۱ـ۶ مقادیر ثابت

۱ـ۶ـ۱ رشته و کاراکتر

رشته‌ها ترکیبی از ⟨char⟩ها هستند که در داخل "" قرار می‌گیرد.
یک کاراکتر شامل یک ⟨char⟩ است که در داخل '' قرار می‌گیرد.
منظور از ⟨char⟩ هر کاراکتر اسکی قابل چاپ (کاراکترهایی که کد اسکی نظیر آنها از ۳۲ تا ۱۲۶ است به جز کاراکترهای single quote (') ، backslash (\) و double quote (")) به علاوه پنج دنباله کاراکتری شامل ('\') برای نمایش single quote ، (\\) برای نمایش backslash ، ("\") برای نمایش double quote ، (\n) برای نمایش newline و (\t) برای نمایش tab می‌باشد.

۱ـ۶ـ۲ اعداد

اعداد در زبان $Xlang$ ۳۲ بیتی و علامت‌دار هستند. همچنین در زبان $Xlang$ فقط با اعداد صحیح [۴] کار می‌کنیم. اعداد صحیح به یکی از دو فرم زیر بیان می‌شوند :

---
[1] Variables
[2] Underscore OR _
[3] Newline OR $\backslash n$
[4] Integer

- دسیمال[1] : مقادیر دسیمال از 2147483648− تا 2147483647 است.
- هگزا دسیمال[2] : اگر یک دنباله با 0x آغاز شود و بعد از آن دنباله‌ای از کاراکترهای نشأت گرفته شده از [a-fA-F0-9] بیاید آنگاه دنباله مذکور بیانگر یک عدد هگزا دسیمال است.

۱ـ۷ عملگرها

عملگرهایی که در زبان ورودی مجاز هستند شامل عملگرهای محاسباتی، منطقی و شرطی می‌شوند که لیست آنها در زیر آمده است :

```
+   -   *   /   %   <   >   !=   <=   >=   ==   &&   ||   !   =   -=   +=
```

۱ـ۸ توکن‌های خاص

توکن‌های خاص به توکن‌هایی گفته می‌شود که نه متغیر هستند نه کلمه کلیدی و نه عملگر که لیست آنها در زیر آمده است :

```
(  )    {  }    [  ]    ;    ,
```

۱ـ۹ تشخیص توکن‌ها

توکن‌ها از طریق فاصله [3] و یا از طریق توکن‌های خاص از هم جدا می‌شوند.
توجه ! هر تعداد فاصله که بین دو توکن وارد شود بی‌تاثیر است و باید نادیده گرفته شود.

۱ـ۱۰ خروجی تحلیلگر لغوی

همانگونه که پیش تر اشاره شد، چنانچه یک برنامه‌ی صحیح به تحلیلگر شما داده شود باید بتواند توکن‌های آن را استخراج کند. به منظور استخراج توکن‌ها از برنامه ورودی، تنها نام آن توکن و مقدار آن را در خروجی چاپ نمایید (ابتدا نام توکن و سپس مقدار آن).

---
[1] Decimal
[2] Hexadecimal
[3] Whitespace: Tab, Space, ···

خروجی تحلیلگر لغوی شما برای یک نمونه کد صحیح به صورت زیر خواهد بود :

```
Input Code :
    int x;
    x = 5;

Analyzer Output :
    TOKEN_INTTYPE int
    TOKEN_WHITESPACE [space]
    TOKEN_ID x
    TOKEN_SEMICOLON ;
    TOKEN_WHITESPACE [newline]
    TOKEN_ID x
    TOKEN_WHITESPACE [space]
    TOKEN_ASSIGNOP =
    TOKEN_WHITESPACE [space]
    TOKEN_DECIMALCONST 5
    TOKEN_SEMICOLON ;
```

خروجی تحلیلگر لغوی شما برای یک نمونه کد حاوی خطا به صورت زیر خواهد بود :

```
Input Code :
    int 9comp;
    9comp = 3;

Analyzer Output :
    TOKEN_INTTYPE int
    TOKEN_WHITESPACE [space]
    error in line 1 : wrong id definition
```

در پیوست ۱ لیست کلیه توکن‌های موجود در زبان $Xlang$ همراه با نام هر توکن آورده شده است.

توجه ! دو توکن `main` و `Program` در فاز بعدی به تفصیل شرح داده خواهند شد.

۱ـ۱۱ بخش امتیازی

درصورتی که تحلیلگر لغوی شما بتواند بعد از مواجه شدن با خطا ضمن چاپ پیغام مناسب ، از خطای موجود عبور کرده و مابقی توکن‌ها را نیز استخراج کند، نمره امتیازی کسب خواهید کرد.

۱ـ۱ پیوست ۱

Token	*Token Name*
boolean	TOKEN_BOOLEANTYPE
break	TOKEN_BREAKSTMT
callout	TOKEN_CALLOUT
class	TOKEN_CLASS
continue	TOKEN_CONTINUESTMT
else	TOKEN_ELSECONDITION
false	TOKEN_BOOLEANCONST
for	TOKEN_LOOP
if	TOKEN_IFCONDITION
int	TOKEN_INTTYPE
return	TOKEN_RETURN
true	TOKEN_BOOLEANCONST
void	TOKEN_VOIDTYPE
Program	TOKEN_PROGRAMCLASS
main	TOKEN_MAINFUNC
⟨variables⟩	TOKEN_ID
+	TOKEN_ARITHMATICOP
-	TOKEN_ARITHMATICOP
*	TOKEN_ARITHMATICOP
/	TOKEN_ARITHMATICOP
%	TOKEN_ARITHMATICOP
&&	TOKEN_CONDITIONOP
||	TOKEN_CONDITIONOP
<=	TOKEN_RELATIONOP
<	TOKEN_RELATIONOP
>	TOKEN_RELATIONOP
>=	TOKEN_RELATIONOP
!=	TOKEN_EQUALITYOP
==	TOKEN_EQUALITYOP
=	TOKEN_ASSIGNOP
+=	TOKEN_ASSIGNOP
-=	TOKEN_ASSIGNOP
!	TOKEN_LOGICOP
(	TOKEN_LP
)	TOKEN_RP
{	TOKEN_LCB
}	TOKEN_RCB
[	TOKEN_LB
]	TOKEN_RB
;	TOKEN_SEMICOLON
,	TOKEN_COMMA
\n [newline]	TOKEN_WHITESPACE
\t [tab]	TOKEN_WHITESPACE
[space]	TOKEN_WHITESPACE
//[some string until \n]	TOKEN_COMMENT
3 [or other decimal integers]	TOKEN_DECIMALCONST
0xFF [or other hexadecimal integers]	TOKEN_HEXADECIMALCONST
"[some string]"	TOKEN_STRINGCONST
'a'[or other characters]	TOKEN_CHARCONST