فصل ۱

راهنمای سازنده تحلیلگر لغوی FLEX

۱ - ۱ مقدمه

flex، ابزاری جهت تولید اسکنرها می باشد. این برنامه فایلی که شامل توضیحات یک اسکنر است را به عنوان ورودی گرفته و اسکنر مربوطه را تولید می کند. توضیحات یک اسکنر شامل عبارات منظم و کدهای C (قوانین) می باشد. خروجی flex فایلی با نام lex.yy.c به زبان C است. این فایل می تواند کامپایل شده و به کتابخانه زمان اجرای flex لینک شود تا یک فایل قابل اجرا تولید کند. فایل اجرایی مورد نظر در زمان اجرا رخدادهای عبارات منظم در ورودی خود را آنالیز می کند. زمانی که یک عبارت منظم یافت شد، کد C مرتبط با آن را اجرا می کند.

۱ - ۲ نصب و اجرای یک مثال ساده

در این بخش مراحل نصب و ایجاد یک تحلیل گر لغوی ساده به شکل گام به گام بیان شده است.

- **نصب flex در لینوکس**: flex جزء نرم افزارهای موجود در مخازن پیش فرض لینوکس می باشد. بنابراین، این نرم افزار را می توان بدون هیچ نوع تغییراتی در مخازن لینوکس و با استفاده از دستور ساده زیر، نصب کرد:

```
sudo apt-get install flex
```

برای نصب این نرم افزار در ویندوز به لینک زیر مراجعه کنید:

https://sourceforge.net/projects/winflexbison/

- **ایجاد یک فایل با پسوند l. که توضیحات اسکنر در آن مشخص شده است**: بدین منظور یک فایل متنی به نام username.l باز کرده و دستورات زیر را در آن تایپ کنید:

```
%%
\username    printf("%s",getlogin());
```

این اسکنر، محتویات یک فایل را بررسی خواهد کرد و هرجایی که الگوی username را ببیند، به جای آن نام کاربری کامپیوتر را قرار خواهد داد.

- **کامپایل اسکنر با استفاده از flex**: فایل username.l را به صورت زیر کامپایل کنید:

```
flex username.l
```

خروجی این دستور، فایلی به نام lex.yy.c خواهد بود.

- **تولید تجلیلگر نهایی**: فایل lex.yy.c را با استفاده از کامپایلر C کامپایل کنید. برای این منظور از دستور زیر استفاده کنید:

```
gcc -o un lex.yy.c -ll
```

خروجی این دستور یک برنامه قابل اجرا به نام un خواهد بود که قابلیت اعمال بر روی ورودی های کاربر را دارد.

- **تست تحلیلگر تولید شده**: حال یک فایل متنی به نام myfile ایجاد کرده و متن زیر را در آن ذخیره کنید:

```
Hello username

this is your computer username
```

با استفاده از دستور زیر، فایل ایجاد شده را به اسکنر می دهیم:

```
./un  <  myfile
```

۳-۱ ساختار برنامه های flex

یک برنامه flex دارای سه بخش به صورت زیر می باشد:

```
definition section
%%
rules section
%%
user subroutines
```

در یک برنامه، دو بخش اول الزامی هستند. اگرچه هر یک از این دو بخش می توانند خالی باشند. بنابراین، در یک برنامه، حداقل به یک %% نیاز خواهد بود. بخش سوم به همراه %% متناظر می توانند حذف شوند. بخش اول (بخش تعاریف) می تواند شامل تعدادی شرط، تعریف و یا آپشن باشد. بخش دوم (بخش قوانین) حاوی تعدادی خط الگو و کدهای C متناظر با هر قانون خواهد بود. بخش سوم (بخش زیرروالهای کاربر) حاوی زیرروالهایی به زبان C است که اغلب توسط قوانین فراخوانی می شوند. مثال زیر نحوه چینش این سه بخش جهت شمارش تعداد کاراکترها، کلمات و خطوط در یک فایل را نشان می دهد:

```
/* just like Unix wc */
%{
int chars = 0;
int words = 0;
int lines = 0;
%}
%%
[a-zA-Z]+       { words++; chars += strlen(yytext); }
\n              { chars++; lines++; }
.               { chars++; }
%%
main(int argc, char **argv)
{
yylex();
printf("%8d%8d%8d\n", lines, words, chars);
```

```
16 }
```

۴ - ۱ عبارات منظم در flex

در این بخش، به شیوه های بیان عبارات منظم در flex می پردازیم. همانطور که گفته شد، دومین بخش از ساختارهای یک برنامه flex در برگیرنده بیان الگوها است. الگوهایی که در این بخش قابل استفاده هستند را می توان به صورت زیر لیست کرد:

- . (نقطه): هر چیزی به غیر از خط جدید (\n)
- []: هر کلاس کاراکتری که با هر یک از کاراکترهای داخل [] مطابقت داشته باشد. به عنوان مثال:
  - $[A-Z]$: کاراکترهای A تا Z
  - [0123456789]: ارقام ۰ تا ۹
  - $[0-9]$: ارقام ۰ تا ۹
  - $[\hat{}A-Z]$: هر کاراکتری غیر از A تا Z
- {}: اگر حاوی یک یا دو عدد باشد، نشان دهنده حداقل و حداکثر تعداد رخدادهای الگوی قبلی می باشد. ولی اگر حاوی یک نام باشد، الگوی همان نام است. به عنوان مثال:
  - A{1,3}: یک تا سه رخداد A
  - A{5} : ۵ رخداد حرف A
  - {Ali}: الگوی کلمه Ali
- *: صفر یا چند رخداد از الگوی قبلی.
- +: یک یا چند رخداد از عبارت منظم قبلی.
- ؟: صفر یا یک رخداد از عبارت منظم قبلی

مثال زیر، نحوه بهره گیری از عبارات منظم جهت تشخیص اعداد علمی در زبان فرترن را نشان می دهد:

```
1 %%
2 [-+]?([0-9]*\.?[0-9]+|[0-9]+\.)(E([-+])?[0-9]+)?        {printf("FN");}
3 %%
```

۱-۵ ورودی خروجی فایل ها در flex

اسکنرهای flex به شکل پیش فرض از ورودی_خروجی استاندارد می خوانند، مگر این که خود طراح اسکنر منبع را مشخص کند. در این بخش، برنامه مربوط به تعداد کلمات و خطوط را به گونه ای تغییر می دهیم که بتواند از فایل بخواند.

نکته: به شکل پیش فرض یک اسکنر ورودی خود را از طریق متغیری به نام yyin مشخص می کند. همان طور که گفته شد، مقدار پیش فرض این متغیر برابر stdin (ورودی استاندارد) می باشد. در مثال زیر، اسکنر به گونه ای بازتعریف شده است که بتواند اطلاعات خود را از یک فایل (که توسط آرگومان خط فرمان ارسال می شود)، دریافت می کند:

```
%option noyywrap
%{
int chars = 0;
int words = 0;
int lines = 0;
%}
%%
[a-zA-Z]+       { words++; chars += strlen(yytext); }
\n              { chars++; lines++; }
.               { chars++; }
%%
int main(int argc, char **argv)
{
        if(argc > 1) {
                if(!(yyin = fopen(argv[1], "r"))) {
                        perror(argv[1]);
                        return (1);
                }
        }
        yylex();
        printf("%8d%8d%8d\n", lines, words, chars);

        return 0;
}
```

در این برنامه، اگر کاربر نام یک فایل را ارسال کند (خط فرمان)، آنگاه مقدار yyin برابر نام فایل قرار خواهد گرفت و در غیر این صورت، مقدار پیش فرض (stdin) در yyin قرار خواهد گرفت.

در ادامه، مثالی از نحوه خواندن چندین فایل و محاسبه تعداد کلمات و خطوط هر یک را می بینیم. نکته ای که در این مثال باید بدان توجه کرد آن است که flex از رویه ای به نام yyrestart(f) جهت تعیین فایل f به عنوان ورودی اسکنر استفاده می شود.

```
%option noyywrap
%{
int chars = 0;
int words = 0;
int lines = 0;
```

```
int totchars = 0;
int totwords = 0;
int totlines = 0;
%}
%%
[a-zA-Z]+       { words++; chars += strlen(yytext); }
\n              { chars++; lines++; }
.               { chars++; }
%%
int main(int argc, char **argv)
{
        if(argc < 2){
                yylex();
                printf("%8d%8d%8d\n", lines, words, chars);
        }

        for(int i=1; i<argc ; i++){
                FILE *f=fopen(argv[i] , "r");

                if( !f ) {
                        perror(argv[i]);
                        return (1);
                }

                yyrestart(f);
                yylex();
                fclose(f);
                printf("%8d%8d%8d %s\n",lines, words, chars,argv[i]);
                totchars += chars;
                totlines += lines;
                totwords += words;

                chars = 0;
                words = 0;
                lines = 0;
        }

        if( argc >= 2 )
                printf("%8d%8d%8d TOTAL\n", totlines, totwords, totchars);

        return 0;
}
```

۱-۶ ساختار ورودی

flex، جهت مدیریت ورودی ها، از ساختمان داده ای تحت عنوان YY_BUFFER_STATE استفاده می کند. این ساختمان داده دارای فیلدهای متفاوتی است که یکی از این فیلدها، متغیری از نوع FILE می باشد. این متغیر، به فایل منبع ورودی اشاره می کند. البته، اسکنر به صورت پیش فرض یک متغیر از نوع YY_BUFFER_STATE ایجاد کرده

و مقدار FILE آن را برابر تهی قرار می دهد (زیرا به شکل پیش فرض از ورودی خروجی استاندارد استفاده می شود).