پری ناز سلطان زاده – 961113020 – گزارش مستند سازی نحوه آماده سازی

پروژه ی کامپایلر – قسمت دوم – پیاده سازی با استفاده از lex – آپدیت شده:

در ابتدا نیاز بود که باتوجه به توضیحاتی که داده شده بود فایل های مربوطه از جمله ابزار **FLEX** را نصب کردم.

(Fast Lexical Analyzer) که به عنوان ابزار تحلیلگر لغوی استفاده می شود .

کد lex به سه قسمت تقسیم می شود:

1) بخش تعریف: (definitions)

در ابتدا از %{ و %} استفاده شده که در بین آن مواردی تعریف می شود که از آن ها در زبان c یا c++ نیز استفاده می شد. مثلا اگر بخواهیم از کتابخانه ای که در c موجود است استفاده کنیم ، در این جا باید تعریف شود . به طور مثال در اینجا من **<stdio.h>** نوشتم ، که شامل ورودی ها و خروجی ها می شود ، برای این است که بتوانم در طول کد از تابع printf() استفاده کنم .از کتابخانه ی **<stdlib.h>** برای انواع داده و توابع عمومی استفاده می شود. در صورتی که نیاز باشد متغیرهایی از نوع int,char,... نیاز باشد ، در این جا باید تعریف شود.

تابع **yyerror()** برای گزارش خطاهای ایجاد شده به کار می‌رود.

```
void yyerror(char const *s);
```

وقتی در parser ، error action اتفاق افتد ، تابع yyerror صدا زده می شود.

در واقع یک تابع کتابخانه ای برای lex و yacc هست که آرگومان رشته ای را به **stderr** در تابع fprintf() پاس می دهد.

Yacc پس از اجرا شدن فایلی به اسم **y.tab.h** را به صورت اتوماتیک تولید می کند ، که این باید حتما به **lex** داده شود . این فایل ارتباط بین دو فایل lex و yacc است که از این طریق فایل yacc به فایل lexبتواند بگوید که به طور مثال digit توکن معتبری است که تعریف شده است . برای همین در بالای کد این فایل را include کردیم.

```
#include "psmadar.tab.h"
```

بعد از آن یکسری **token** های معروف را تعریف کردم که بتوان سایر توکن های معرفی شده مورد نیاز پروژه که در قسمت yacc به آن ها پرداخته می شود را با استفاده از این token ها تعریف کرد .

برای این tokenها regular expression معادل آن ها نوشته شده است . به طور مثال برای white [\t] نوشته شده برای حروف letter انتخاب شده که : [a-zA-Z] برای آن نوشتم .(یعنی تمام حروف کوچک و بزرگ a تا z) برای رقم از digit استفاده کردم . [0-9] (ارقام صفر تا نه) . و برای معرفی سایر اعداد دورقمی ، سه رقمی تا ... یا در واقع همان اعداد integer (صحیح) که البته مثبت هستند ، integer را تعریف کردم که از روی token digit ساخته می شود و regx آن : {digit}+ است. و برای Operator هم که در هنگام مقدار دهی به کار می رود ، از : استفاده کردم.

در نهایت بخش تعریف تمام می شود .

2)بخش الگو یا قواعد (Rules) :

باید از دو علامت ٪٪ در ابتدا و انتها برای معرفی این بخش استفاده کرد . در اینجا الگوها ، قواعد ی که می خواهم در طول پروژه به کار ببرم را ذکر میکنم . این بخش در واقع به بخش گرامر نیز مربوط می شود.

الگوی مشخصات اجزای مدار تعریف شده است . ابتدا white را تعریف کردم و {} در جلوی آن قرار دادم . سپس integer را تعریف کردم . ابتدا با استفاده از تابع **atoi** ، رشته را به integer تبدیل می کنم. Yytext همان رشته ی مورد نظر ماست و yylval ، مقدار مرتبط با token مربوطه را در خود ذخیره می کند . و NUMBER بر می گرداند.

```
yylval=atoi(yytext);
```

مقاومت را تعریف کردم که token مربوط به Resistor را برمیگرداند . ("R"|"r") . سپس سلف را تعریف کردم که token مربوط به Inductor را برمیگرداند . ("l"|"L") . بعد از آن خازن تعریف می شود که token مربوط به Capacitor را برمیگرداند .("C"|"c")

سپس اپراتور مقدار دهی را تعریف کردم که OP برمیگردد. همچنین AND توکن مشخص کننده سری یا SeriOp برمیگردد و OR مشخص کننده موازی یا MovaziOp برمیگردد. "\n" که در واقع همان Enter هست ، به عنوان end برمیگردد.

در آخر هم با یک dot مشخص شده که اگر کاراکتری وارد شد که با الگوهای نوشته شده سازگار نبود ، به کاربر اعلام کند که از کاراکتر اشتباهی استفاده کرده و چاپ کند : Unrecognized character . این همان تابع **yyerror()** است که بالاتر هم توضیح دادم.

3)بخش توابع (Routines) :

این بخش از نظر FLEX اختیاری است اما برای کامپایل به زبان c جهت تولید تحلیلگر لغوی اجباری است.

تابع **yywrap()** ، هنگامی که کار تحلیل فایل به پایان برسد توسط **FLEX** فراخوانی می شود . این تابع همیشه مقدار یک را برمیگرداند . (اگر کار تحلیل تمام نشده باشد صفر برمیگردد.)

کامنت : (کار اضافی)

از "//.*" برای کامنت استفاده شده. معنی این خط هم به این صورت است که وقتی به // رسید بعد از آن هر کاراکتری بود به هر تعدادی برای آن هیچ عملیاتی انجام ندهد .

پروژه ی کامپایلر – قسمت چهارم و پنجم – پیاده سازی با استفاده از Yacc و نهایی سازی پروژه:

بخش تعریف: (definitions)

در ابتدا از %{ و %} استفاده شده است. مثلا اگر بخواهیم از کتابخانه ای که در c موجود است استفاده کنیم ، در این جا باید تعریف شود . به طور مثال در اینجا من **<stdio.h>** نوشتم ، که شامل ورودی ها و خروجی ها می شود ، برای این است که بتوانم در طول کد از تابع printf() استفاده کنم . از **< string.h>** برای رشته ها حساب می شود. از **<math.h>** برای محاسبات ریاضی استفاده می شود. که بتوانیم عملیات ریاضی را با استفاده از توابع این کتابخانه انجام دهیم.

تابع () **yyerror** که تکراری است.

تابع ()yyerror برای گزارش خطاهای ایجاد شده به کار می‌رود.

```
void yyerror(char const *s);
```

در واقع وقتی در parser ، error action اتفاق افتد ، تابع yyerror صدا زده می شود.

در واقع یک تابع کتابخانه ای برای lex و yacc هست که آرگومان رشته ای را به stderr در تابع fprintf() پاس می دهد.

خط بعدی صدا زده می شود برای این که lexer را فراخوانی کند و توکن را برگرداند.

```
extern int yylex(void);
```

از نوع **extern** تعریف شده چون به طور مستقیم از آن استفاده نمی شود .خروجی آن int است و ورودی آن هم voidهست .

خط بعدی w را از جنس float به طور ثابت تعریف کردم که در واقع همان مقدار امگا است که به طور ثابت در نظر گرفتم . می دانیم امگا یعنی : w=2pif که این را به صورت کامنت در کد هم نوشتم.

بعد از آن component را از جنس **enum** تعریف کردم . درواقع enum یک نوع داده است (مثل int,string,...) که برای تعریف نوع مولفه هایمان از آن استفاده کردم. Comres مولفه مقاومت ، comind مولفه سلف ، comcap مولفه خازن است.

پس از آن **توابع** را تعریف کردم . در زبان سی ، ابتدا تابع را در بالای برنامه باید تعریف کرد و بعد در پایین جزییات تابع را نوشت که چه کار میکند و در صورت void نبودن ، چه مقداری را برمی گرداند.

اولین تابع **()info** است که برای تعیین جنس مولفه ی ارسال شده و شماره اندیس آن استفاده می شود. این تابع void است و مقدار بازگشتی ندارد.

دومین تابع **()Quant** است که برای مقداردهی مولفه های مدار استفاده می شود . یعنی علاوه بر ذخیره سازی جنس مولفه و شماره اندیس ، مقدار آن مولفه را هم ذخیره می کند . در واقع این تابع ، تابع کامل تری از تابع info است و برای تعریف مشخصات اجزای مدار استفاده می شود . این تابع void است و مقدار بازگشتی ندارد.

سومین تابع ، **()ImpedanceCal** است که شماره مولفه هر اندیس ارسالی را می گیرد و بر اساس جنس آن مولفه تصمیم می گیرد . اگر مقاومت باشد ، خود مقدار مقاومت را به عنوان قسمت حقیقی امپدانس ذخیره می کند . اگر سلف باشد ، **xl** را برای آن حساب می کند و به عنوان قسمت موهومی امپدانس ذخیره می کند . اگر خازن باشد ، **xc** را برای آن حساب می کند و به عنوان قسمت موهومی امپدانس ذخیره

می کند. مقداری که بر میگرداند ، اخرین شماره اندیس ساختارداده ای است که برای محاسبه امپدانس استفاده می کنیم که از جنس float است.

چهارمین تابع **()MadarCal** است که ابتدا ورودیی که می گیرد برای این است که تشخیص دهد مدار سری است یا موازی. سپس بر اساس آن امپدانس کل را به صورت عددی مختلط حساب کرده و پرینت می کند. این تابع void است و مقدار بازگشتی ندارد.

پس از تعریف توابع به سراغ تعریف متغیرهای **global** خود می روم. S را برای مشخص کردن سری و موازی بودن (مقدار یک و دو) و x را برای محاسباتمان در تابع **MadarCal** تعریف کردم که مقدار اولیه آن 2 است. متغیر **lastimp** را با مقدار اولیه 1 تعریف کردم که قرار است اندیس آخرین خانه ی پر شده در ارایه imp را نگه می دارد.

متغیر های t و r را برای تابع MadarCal تعریف کردم که از جنس float است.

حال به تعریف **struct** ها می پردازم . ساختمان ها (Structures) نوع داده ای هستند که می توانند انواع مختلف داده های دیگر را در خود تحت یک نام ذخیره کند.

انواع داده هایی که(پارامتر های ساختمان) در ساختمان **Component** تعریف شده است : اولی **type** است که از نوع داده ای Component است که برای تعیین جنس مولفه مان (مقاومت ، سلف ، خازن) استفاده می شود. دومی **number** است که درون آن مقدار مولفه ی ارسال شده ذخیره می شود . سومی **id** است که اندیس مولفه را ذخیره می کند. سپس در انتها ی آرایه 100 تایی تعریف کردم که از جنس ساختمان ذکر شده است که در طول برنامه از آن استفاده می کنم.

انواع داده هایی که(پارامترهای ساختمان) در ساختمان **Impedance** تعریف شده است: اولی **value** است که مقدار حقیقی عدد مختلط امپدانس را نگه می دارد . دومی **imag** است که مقدار موهومی عدد مختلط امپدانس را نگه می دارد . سپس در انتها آرایه 100 تایی تعریف کردم که از جنس ساختمان ذکر شده است که در طول برنامه از آن استفاده می کنم.

توکن ها:

حال باید توکن هایی که lex برگردانده را در فایل yacc تعریفشان کنم.

توکن های برگردانده شده را با token% مشخص می کنم.

```
%token Resistor Inductor Capacitor
%token NUMBER

%token OP
%token SeriOp MovaziOp

%token END
```

گرامرها:

حال به قسمت گرامر می روم که باید بین دو تا درصد ها آن را بنویسم. :

برای ورودی دو حالت دارم . یا میتواند شامل هیچ چیز نباشد ، یا شامل عبارت مدنظرمان باشد که می تواند چند خط باشد بنابراین در input ، Line را اضافه می کنم تا چند بار بتوانم ورودی بگیرم:

```
Input: /* empty */;
Input: Input Line;
```

Line می تواند expression باشد که در انتهایش /n می گذاریم. که تنها finish چاپ می کند .

سپس گرامر های مربوط به تعریف مشحصات مقاومت خازن سلف تعریف می شود.خروجی یا $$ را برابر با $2 قرار می دهیم و درواقع شماره اندیس درون آن قرار می گیرد. همچنین عبارتی را چاپ می کند تا کاربر متوجه شود که اون مولفه (مقاومت ، خازن یا سلف) تشخیص داده شده است.همچنین تابع **info** فراخوانی می شود که جنس مولفه و شماره اندیس آن ورودی تابع هستند.

سپس گرامری تعریف کردم که expression به expression2 که مخصوص گرامرهای سری و موازی است و می رود و آنجا تابع **MadarCal** را فراخوانی می کند و $1 که خروجی گرامر expression2 است (شماره یک یا دو که تعیین کننده سری یا موازی بودن است.) را به عنوان ورودی به تابع می دهد. همچنین عبارتی را چاپ می کند تا کاربر متوجه شود که اون مولفه (مقاومت ، خازن یا سلف) تشخیص داده شده است. همچنین اندیس آن مولفه را جلویش چاپ می کند.بعد تابع **Quant** فراخوانی می شود که نوع جنس مولفه و شماره اندیس $2 و مقدار مولفه $4 به عنوان ورودی به تابع داده می شود.تابع **ImpeadnceCal** را هم فراخوانی می کنم که شماره اندیس مولفه $2 به عنوان ورودی به داده می شود . پس در واقع بعد مقدار دهی هر مولفه محاسبه امپدانس برای آن مولفه خاص انجام می شود و به ترتیب در خانه های آرایه imp ذخیره می شود.

دوخط بعدی گرامرهای مربوط به سری و موازی تعریف شده است که چاپ می کند سری یا موازی است . متغیر s درصورت سری بودن مقدار یک و درصورت موازی بودن مقدار دو را می گیرد. همچنین محتوای خروجی برابر با مقدار s است.

بخش گرامر ها تمام شد و حال توابع را توضیح می دهم

توابع:

تابع **Info()** :

ورودی ها: type که جنس مولفه (مقاومت یا خازن یا سلف) را مشخص می کند. Index که شماره اندیس مولفه را مشخص می کند.

عملکرد: جنس مولفه ارسال شده به همراه شماره اندیس آن را به ترتیب در خانه های آرایه value ذخیره می کند.همچنین باید به طرز استفاده کردن از انواع داده های موجود در ساختمان value توجه کرد که چگونه باید در زبان سی آن را نوشت . به طور مثال :

```
value[index].type = type;
```

خروجی : چون این تابع void است ، خروجیی ندارد و مقداری باز نمی گرداند.

تابع **Quant()**:

ورودی ها: type که جنس مولفه (مقاومت یا خازن یا سلف) را مشخص می کند. Index که شماره اندیس مولفه را مشخص می کند.num که مقدار آن مولفه را مشخص می کند.

عملکرد: جنس مولفه ارسال شده به همراه شماره اندیس آن و به همراه مقدار آن مولفه به ترتیب در خانه های آرایه value ذخیره می کند.همچنین باید به طرز استفاده کردن از انواع داده های موجود در ساختمان value توجه کرد که چگونه باید در زبان سی آن را نوشت .

(همچنین در کامنت تابع پرینت را نوشتم تا با آن بتوان امتحان کرد که شماره اندیس مولفه و مقدار آن درست است یا خیر.)

خروجی : چون این تابع void است ، خروجیی ندارد و مقداری باز نمی گرداند.

تابع **ImpedanceCal()**:

ورودی: num که اندیس آن مولفه را مشخص می کند.

عملکرد: در این تابع مقدار امپدانس هر مولفه محاسبه می شود. با استفاده از دستور switch case برای هر نوع جنس مولفه تصمیم جداگانه می گیریم. در داخل سوییچ جنس مولفه ی اندیس وارد شده قرار می گیرد .

اگر مقاومت باشد : مقدار مقاومت به عنوان مقدار حقیقی برای امپدانس موردنظر که یک عدد مختلط است ، در نظر گرفته می شود. مقدار موهومی آن صفر در نظر گرفته می شود.

اگر سلف باشد : باید xl را محاسبه کنم . XL=L*W پس مقدار سلف را ضبدر امگا (که در بالای برنامه به طور ثابت تعریف شده است) می کنم و به عنوان مقدار موهومی برای امپدانس موردنظر که یک عدد مختلط است ، در نظر گرفته می شود. مقدار حقیقی آن صفر در نظر گرفته می شود.

اگر خازن باشد : باید xc را محاسبه کنم . XC = 1/(C*W*(-1)) پس مقدار خازن را ضبدر امگا (که در بالای برنامه به طور ثابت تعریف شده است) می کنم ومعکوس می کنم همچنین در یک منفی ضرب می کنم (چون در فرمول اصلی امپدانس xc مقدار منفی می گیرد.) و به عنوان مقدار موهومی برای امپدانس موردنظر که یک عدد مختلط است ، در نظر گرفته می شود. مقدار حقیقی آن صفر در نظر گرفته می شود.

به متغیر **lastimp** که نمایان گر شماره آخرین خانه ی ارایه imp است یک واحد اضافه می کنیم اما مقداری که به عنوان خروجی بر میگردد ، همان شماره اخرین خانه ی ارایه imp است.

به طور کلی عدد مختلطی که برای محاسبه امپدانس هر مولفه حساب می شود با فرمول زیر است:

$$Z = R + jX \rightarrow Z = R + Z = R + j(X_L - X_C)$$

خروجی : تابع float است و شماره اخرین خانه ارایه imp را بر می گرداند.

تابع **MadarCal()**:

ورودی : s است اگر یک بگیرد مدار سری و اگر دو بگیرد مدار موازی است.

عملکرد: در این تابع مقدار امپدانس کل حساب می شود که به نوع مدار (سری یا موازی بودن) بستگی دارد. با استفاده از دستور switch case برای هر نوع مدار تصمیم جداگانه می گیریم. در داخل سوییچ مقدار s (یک یعنی سری و دو یعنی موازی) قرار می گیرد .

اگر سری باشد: باید مقادیر حقیقی را باهم جمع بزنیم . مقادیر موهومی رو هم به صورت جدا جمع می زنیم . در واقع در ارایه ی imp تک تک جلو می رویم و مقدار امپدانس هر خانه را با حاصل جمع خانه های قبلی جمع میزنیم و مقدار نهایی را در خانه ی بعد از اخرین خانه ی پر شده در ارایه imp قرار می دهیم. فرمولی که برای محاسبه امپدانس کل در مدار سری استفاده می شود به صورت زیر است:

$$series \rightarrow Z_T = Z_1 + Z_2 + \dots$$

اگر موازی باشد : باید مقادیر حقیقی و موهومی را با توجه به فرمول موجود جمع بزنیم. در واقع در ارایه ی imp تک تک جلو می رویم و مقدار امپدانس هر خانه را با حاصل جمع خانه های قبلی جمع میزنیم و مقدار نهایی را در خانه ی بعد از اخرین خانه ی پر شده در ارایه imp قرار می دهیم.

$$parallel \rightarrow \frac{1}{Z_T} = \frac{1}{Z_1} + \frac{1}{Z_2} + \dots$$

در موازی حتما باید چک کنیم که اگر هر دو مقدار خانه های کنارهم صفر باشند ، چون کسری است ، 1/0 مقدار بی نهایت تولید می شود پس باید صفر در نظر بگیریم.

اگر یکی از آن ها صفر و دیگری مخالف صفر باشد ، باید مقدار آن خانه ی صفر را در نظر نگیریم. اگر هر دو مخالف صفر باشند طبق فرمول پیش می رویم.

t و r هم برای این استفاده کردم که بتوان مقادیر را درست جا به جا کرد و بتوان حاصل جمع خانه های قبلی را در خانه ی قبلی آن خانه ی مورد نظرمان قرار دهیم.

هر مرحله که این تابع فرخوانی شود ، مقدار امپدانس کل تا آن مرحله را حساب می کند و چاپ می کند. آخرین عبارت نمایش داده شده ، **امپدانس کل** ما است که به صورت عددی مختلط نمایش داده می شود. و این مرحله خروجیی است که از کل برنامه ما انتظار داریم.

خروجی : چون این تابع void است ، خروجیی ندارد و مقداری باز نمی گرداند.

در آخر :

در نهایت ، با استفاده از تابع **yyerror()** رشته مربوط به ارور را چاپ می کند. و فقط عملیات چاپ را انجام می دهد و کار دیگه ای نمی کند.

پس از آن تابع **int main()** داریم که نتیجه تابع **yyparse()** در ret ریخته می شود. تابع **yyparse()** را صدا می زنیم تا عملیات parse اتفاق بیوفتد و همچنین در آن **yylex** صدا زده می شود. اما خروجی اش تعداد error ای است که پیدا کرده است . اگر صفر باشد که کاری نداریم اما یک باشد به **stderr** می رود و رشته error found چاپ می شود.

اجرا و تست کدها:

برای قسمت اجرا و تست کدها باید دستورات زیر زده شوند :

(اسم فایل های lex و yacc من psmadar هست.)

flex psmadar.l

bison –d psmadar.y

gcc psmadar.tab.c lex.yy.c

./a.exe

اولی برای فایل **lex** و دومی برای فایل **yacc** است و سومی کامپایلر زبان **c** است. آخری هم فایل اجرایی ما است که در آن کدمان را تست می کنیم .

خلاصه کل برنامه :

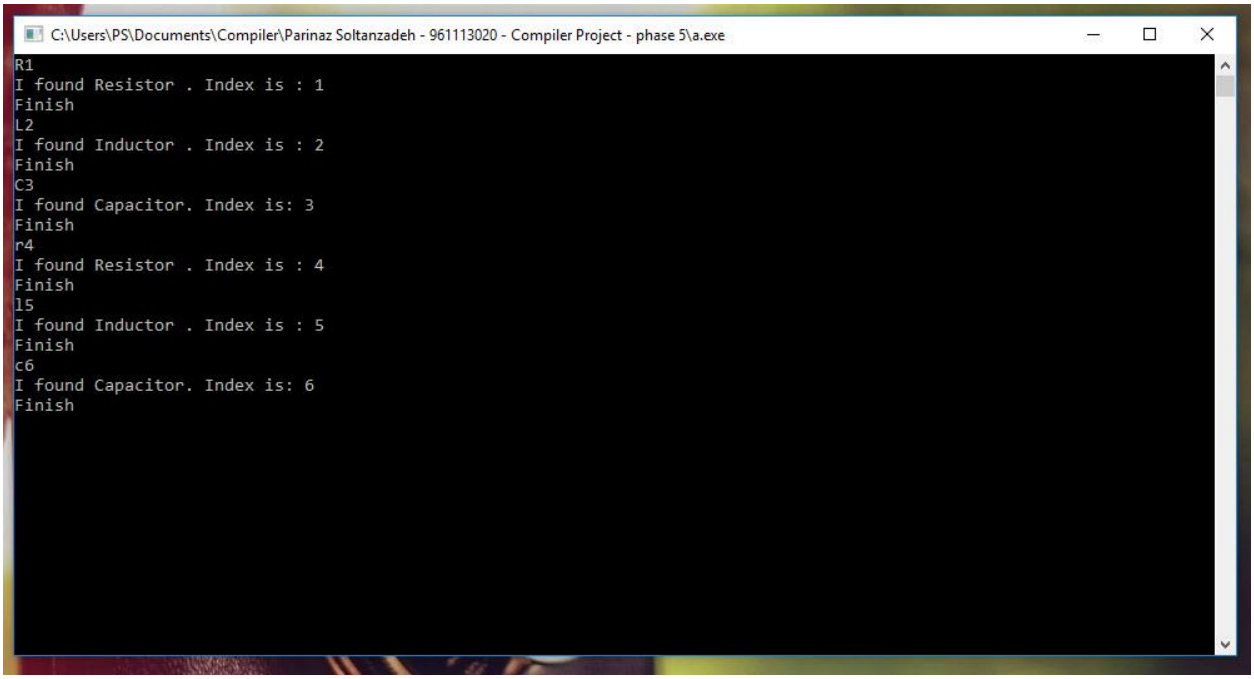
پس در کل برنامه ما به این صورت کار می کند که ابتدا مولفه های موجود در مدارمان را به ترتیب هم زمان تعریف و مقدار دهی می کنیم که با استفاده از تابع Quant نوع مولفه و شماره اندیس مولفه و مقدار مولفه وارد خانه ارایه value می شود. سپس تابع ImpedanceCal فراخوانی می شود و با توجه به اندیس آن مولفه ، امپدانس آن مولفه را محاسبه کرده و در ارایه imp ذخیره می کند .

سپس مولفه ها را به ترتیبی که وارد کردیم ، با AND به صورت سری و یا با OR به صورت موازی کنار هم می چینیم و امپدانس کل برای ما به صورت گام به گام محاسبه شده و در آخرین خط نشان می دهد.

خروجی ها :

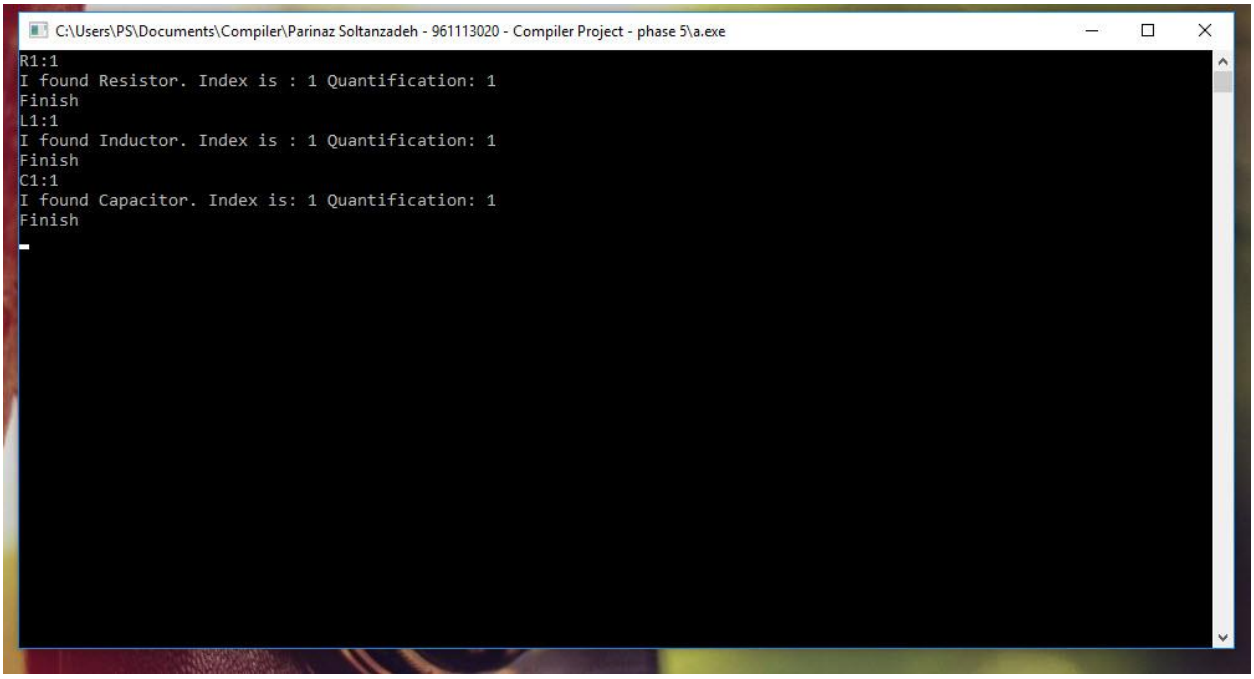
خروجی برنامه را به عنوان مثال می بینیم:

1) اگر فقط بخواهیم مولفه ها را تشخیص دهیم و هیچ گونه محاسبه ای انجام ندهیم (فقط تشخیص) به این صورت وارد می کنیم و این خروجی را می گیریم:



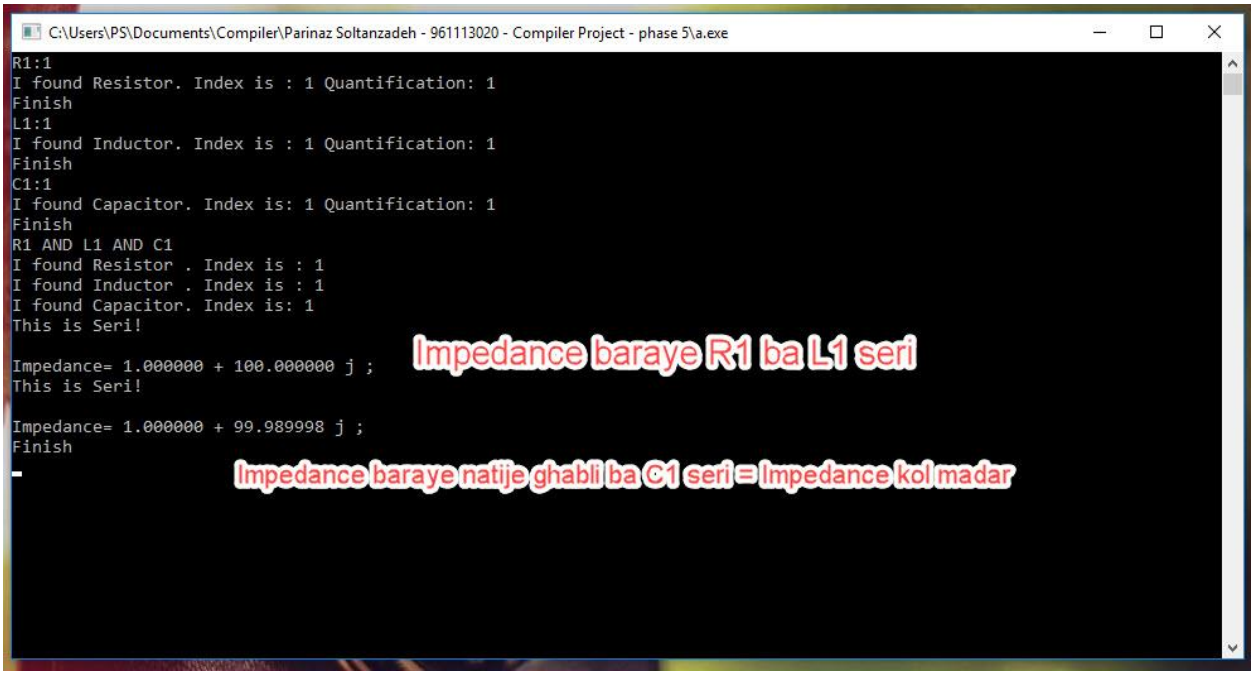
2)اگر بخواهیم علاوه بر تعریف مشخصات ، مولفه هایمان را مقدار دهی کنیم و هدفمان محاسبه امپدانس کل باشد باید به صورت زیر پیش برویم:

ابتدا مولفه هایی که داریم را به ترتیب وارد و مقدار دهی می کنیم:

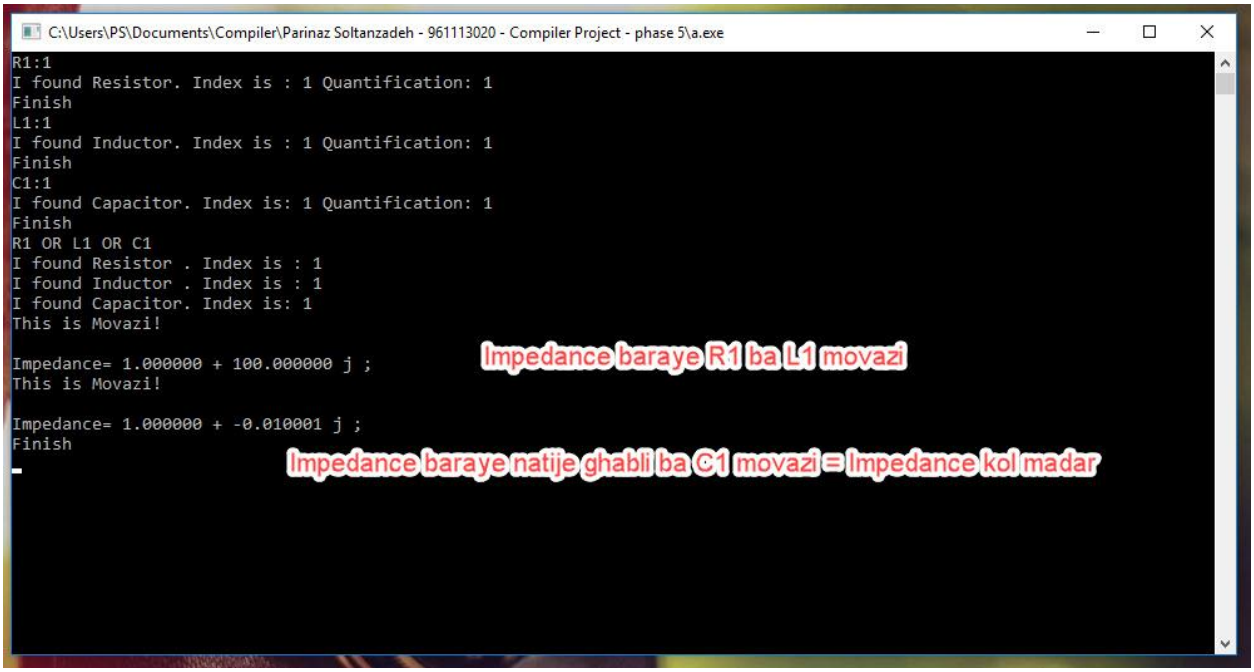


سپس سری بودن را با AND یا موازی بودن را با OR مشخص می کنیم . **خروجی که در آخرین خط است ، امپدانس کل ماست**:

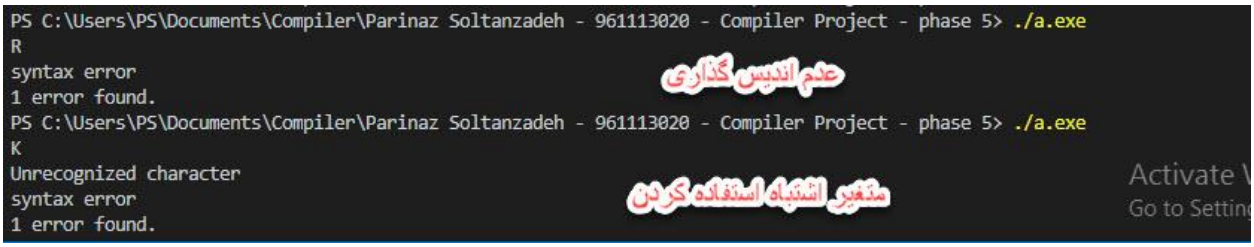
سری:



موازی:



در صورتی که اشتباه وارد کنیم این پیغام **خطا** را نشان می دهد:



علاوه بر توضیحات مستند فوق ، در فیلم نیز به صورت کامل و عملی توضیح خواهم داد. در آن جا توضیح خواهم داد که به چه ترتیبی باید ورودی دهیم تا خروجی مد نظر را دریافت کنیم.

منابع :

منابعی که در طول پروژه از آن ها کمک گرفتم :

https://silcnitc.github.io/ywl.html

https://tldp.org/HOWTO/Lex-YACC.html

https://cse.iitkgp.ac.in/~bivasm/notes/LexAndYaccTutorial.pdf

https://blog.faradars.org/%D8%A7%D9%85%D9%BE%D8%AF%D8%A7%D9%86%D8%B3/

https://www.youtube.com/watch?v=8MMzeeHNjIw

https://www.youtube.com/watch?v=2GvqQvohP2k&t=105s

پری ناز سلطان زاده – 961113020