بسمه تعالی

دانشکده ی مهندسی برق و کامپیوتر
دانشگاه صنعتی اصفهان

طراحی کامپایلرها – نیمسال دوم ۹۹-۱۳۹۸
تکلیف شماره یک – تحویل شنبه ۱۳۹۹/۱/۱۶

مریم سعیدمهر – ش.د. : ۹۶۲۹۳۷۳

۱- با تحقیق و بررسی تفاوت‌های کد میانی و کد نهایی را مشخص کنید و مزیت ترجمه به کد میانی پیش از کد نهایی را توضیح دهید.

پاسخ :

مهم‌ترین تفاوت های intermediate code با target code در وابستگی به ماشین است; به این صورت که کد میانی مستقل از سخت‌افزار ماشین است و کد نهایی کاملاً وابسته به این قضیه میباشد. به علاوه کد میانی قابل اجرا روی هیچ سخت افزاری نیست و نیاز به یک سری مراحل دیگر برای این محقق شدن این امر دارد.

اگر کامپایلر source code را بدون گزینه‌ای برای تولید کد میانی به کد ماشین مقصد ترجمه کند، در این صورت برای هر ماشین جدید می‌بایست یک کامپایلر بومی جدید استفاده شود .(مثلاً اگر ۵ زبان متفاوت و ۳ نوع ماشین مختلف داشتیم باید در کل ۱۵ کامپایلر بنویسیم در حالی که با وجود کد میانی ،فقط به ۵ کامپایلر برای تبدیل زبان‌ها به کد میانی و ۳ کامپایلر برای ترجمه ی کد میانی به زبان ماشین اختصاصی هر دستگاه نیاز بود و مزیت این کد میانی وقتی بیشتر احساس می‌شود که یک زبان جدید اضافه شود. همچنین مزیت دیگرش موقع بهینه سازی کد ها بیشتر دیده میشود) کد میانی نیاز به کامپایلر جدید را برای هر ماشین منحصر به فرد جدید حذف می‌کند و بخش تحلیلی کد برای همه کامپایلرها یکسان خواهد بود. بخش دوم کامپایلر که بخش سنتز است بر اساس ماشین مقصد تغییر می‌یابد.

بدین ترتیب اعمال تغییرات source code برای بهینه‌سازی عملکرد از طریق به‌کارگیری تکنیک‌های بهینه‌سازی کد روی کد میانی آسان‌تر خواهد بود.

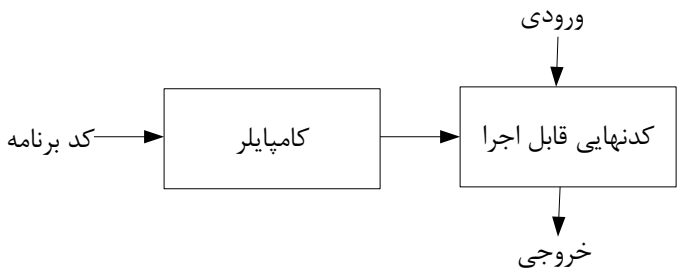
کدهای میانی را می‌توان به چندین روش نمایش داد که هر کدام مزایای خاص خود را دارند :

- نمایش سطح بالا (High Level IR) – نمایش سطح بالای کد میانی بسیار به خود زبان نزدیک است. این نمایش را به سادگی می‌توان از روی source code s ایجاد کرد و امکان تغییرات کد برای بهینه‌سازی عملکردی به طور آسانی وجود دارد. اما برای بهینه‌سازی ماشین مقصد این روش ترجیح کمتری دارد.
- نمایش سطح پایین (Low Level IR) – این نوع از نمایش به ماشین مقصد نزدیک‌تر است و برای ثبت و تخصیص حافظه، انتخاب مجموعه دستورالعمل‌ها و غیره مناسب است. این روش برای بهینه‌سازی‌های وابسته به ماشین خوب است.

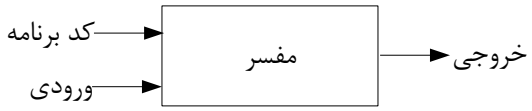
۲- تفاوت مفسر، کامپایلر و ماشین مجازی چیست ؟ مزایا و معایب هر یک را با تحلیل خودتان ،مطالعه ی کتاب مرجع و جستجو در اینترنت و مراجع دیگر بیان نمایئد .

پاسخ :

کامپایلر یک برنامه است که میتواند source code را بخواند و آن را به یک زبان دیگر ترجمه کند(کد نهایی) و اگر کدنهایی یک کد زبان ماشین قابل اجرا باشد ، کاربر میتواند آن را اجرا کند و برنامه اجرایی ، ورودی ها را دریافت و پردازش نماید و خروجی های متناسب تولید کند، مطابق شکل زیر :

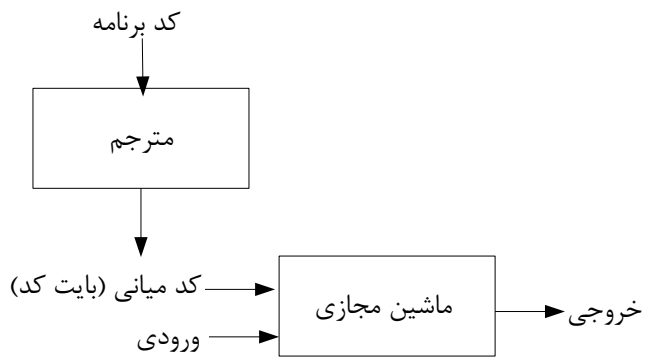


مفسر یک نوع دیگری از پردازنده ی زبان است که به جای تولید کدنهایی به عنوان ترجمه ی کدبرنامه، مستقیماً کد برنامه را اجرا میکند و با توجه به ورودی های کاربر ، خروجی متناسب تولید میکند. مطابق شکل زیر :



ماشین واقعی ، دستورات را به صورت سخت افزاری اجرا میکند در حالی که ماشین مجازی همان کار را به صورت نرم‌افزاری انجام میدهد مثلاً در مورد زبان جاوا داریم :

پردازشگر زبان جاوا ، ترکیبی از کامپایلر و مفسر است (طبق تصویر زیر). کد برنامه ی جاوا در ابتدا به یک کد میانی (jbc ≡*java byte code*) ترجمه میشد و در نهایت توسط ماشین مجازی ، تفسیر میگردد. مزیت این ترکیب در این است که بایت-کد روی یک ماشین کامپایل می‌شود و میتواند (احتمالاً از طریق شبکه) روی یک ماشین دیگر تفسیر گردد. اگر ماشین مجازی نداشتیم نیاز به یک مبدل بومی وابسته به ماشین برای تبدیل jbc به زبان ماشین داشتیم.



۳- بررسی کنید تشخیص هریک از خطاهای زیر مربوط به کدام بخش از کامپایلر است.(با ذکر دلیل)

الف) استفاده از واژه els به جای else

ب) برابر نبودن تعداد پرانتزهای باز و بسته در یک عبارت محاسباتی

ج) به کار بردن عبارت a[m] که در آن a یک آرایه با ۱۰ خانه و m یک متغیر از نوع float با مقدار ۱.۵ است.

د) فراخوانی تابع func(1.5,1.2) که prototype آن به صورت int func (int arg1 , float arg2) است .

ه) استفاده از متغیر a در حالی که قبلاً آن را تعریف نکرده ایم.

پاسخ :

الف) Lex Analyzer

ب) Syntax Analyzer

ج) Semantic Analyzer

د) Semantic Analyzer

ه) Semantic Analyzer

Classification of Compile-time error :

* Lex Analyzer : This includes misspellings of identifiers, keywords or operators
* Syntax Analyzer : missing semicolon or unbalanced parenthesis
* Semantic Analyzer : incompatible value assignment or type mismatches between operator and operand
* Logical : code not reachable, infinite loop.

۴- الگوی هریک از موارد زیر را با عبارات منظم (RE) مشخص کنید

الف) تمام شناسه ها در زبان C

ب) URL های وب

ج) اعداد اعشاری ثابت

د) آدرس‌های ایمیل

پاسخ :

در ابتدا تعریفهای مورد نیاز :

$$\Sigma_{digit} = \{0,1,2,3,4,5,6,7,8,9\}$$

$$\Sigma_{letter} = \{A,B,C,\ldots,Z,a,b,c,\ldots,z\}$$

$$\Sigma_{digits} = (\Sigma_{digit})^{+}$$

$$\Sigma_{optFract} = \text{‘.’}\ \Sigma_{digits} \mid \varepsilon$$

$$\Sigma_{optExp} = (E (\text{‘+’} \mid \text{‘-’} \mid \varepsilon)\ \Sigma_{digits}) \mid \varepsilon$$

$$\Sigma_{protocol} = (((\text{“http”} \mid \text{”https”})\ .\ \text{“://”}) \mid \varepsilon)\ .\ (\text{“www.”} \mid \varepsilon)$$

$\Sigma_{path} = (\Sigma_{letter} \mid \text{“-”} \mid \text{“@”} \mid \text{“:”} \mid \Sigma_{digit} \mid \text{“%”} \mid \text{“.”} \mid \text{“_”} \mid \text{“+”} \mid \text{“~”} \mid \text{“#”} \mid \text{“=”})^{+}$

$\Sigma_{query} = (\Sigma_{letter}|\text{“-”}|\text{“@”}|\text{“:”}|\Sigma_{digit}|\text{“%”}|\text{“.”}|\text{“_”}|\text{“+”}|\text{“~”}|\text{“#”}|\text{“=”}|\text{“?”}|\text{“&”}|\text{“//”})^{*}$

$\Sigma_{domain} = \Sigma_{letter}^{+}$ // like com , ir , org and etc.

$\Sigma_{word} = (\Sigma_{letter} \mid \Sigma_{digit} \mid \text{“_”})^{+}$

الف) $((\Sigma_{letter}) . (\text{’_‘} \mid \Sigma_{letter} \mid \Sigma_{digit})^{*}) \mid ((\text{’_‘})^{+} . (\Sigma_{letter} \mid \Sigma_{digit})^{+} . (\text{’_‘} \mid \Sigma_{letter} \mid \Sigma_{digit})^{*})$

ب) $\Sigma_{protocol} \bullet \Sigma_{path} \bullet \text{“.”} \bullet \Sigma_{domain} \bullet \Sigma_{query}$

ج) $\Sigma_{digits} . \Sigma_{optFract} . \Sigma_{optExp}$

د) $\Sigma_{word}^{+}.((\text{“-”}|\text{“.”})^{*}.\Sigma_{word})^{*}.\text{“@”}.\Sigma_{word}^{+}.((\text{“-”}|\text{“.”})^{*}.\Sigma_{word})^{*}.(\text{“.”}.\Sigma_{word})^{+}$

۵- برای هریک از موارد زیر DFA رسم کنید به طوری که توکن های با نوع مختلف تفکیک شوند یعنی برای نوع شناسه و هریک از کلمات کلیدی حالت نهایی مجزایی وجود داشته باشد.

الف) شناسه ها در زبان C

ب) همه ی کلمات کلیدی int , return , register , if , continue , const در زبان C

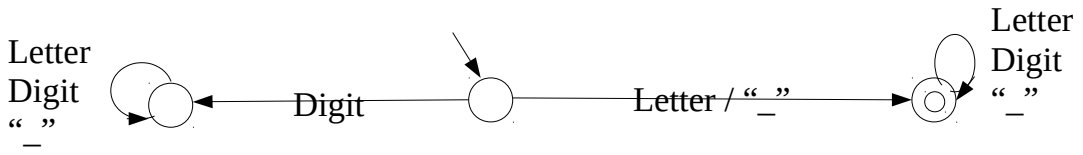
ج) DFA های ایجاد شده در قسمت‌های قبل را به یک DFA واحد تبدیل کنید .

پاسخ :

الف)

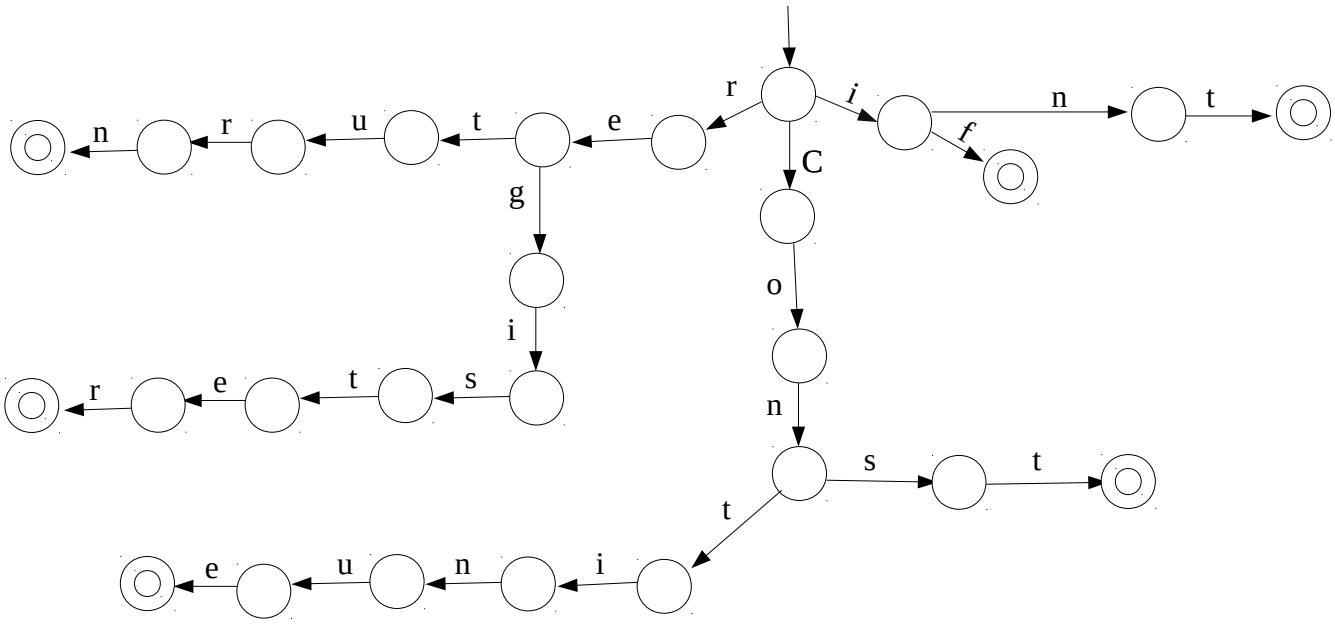
اولاً :

$\Sigma = \{\{a,b,..,z,A,B,\ldots,Z\} \equiv \text{Letter} , \{0,1,\ldots,9\} \equiv \text{Digit} , \text{“_”}\}$



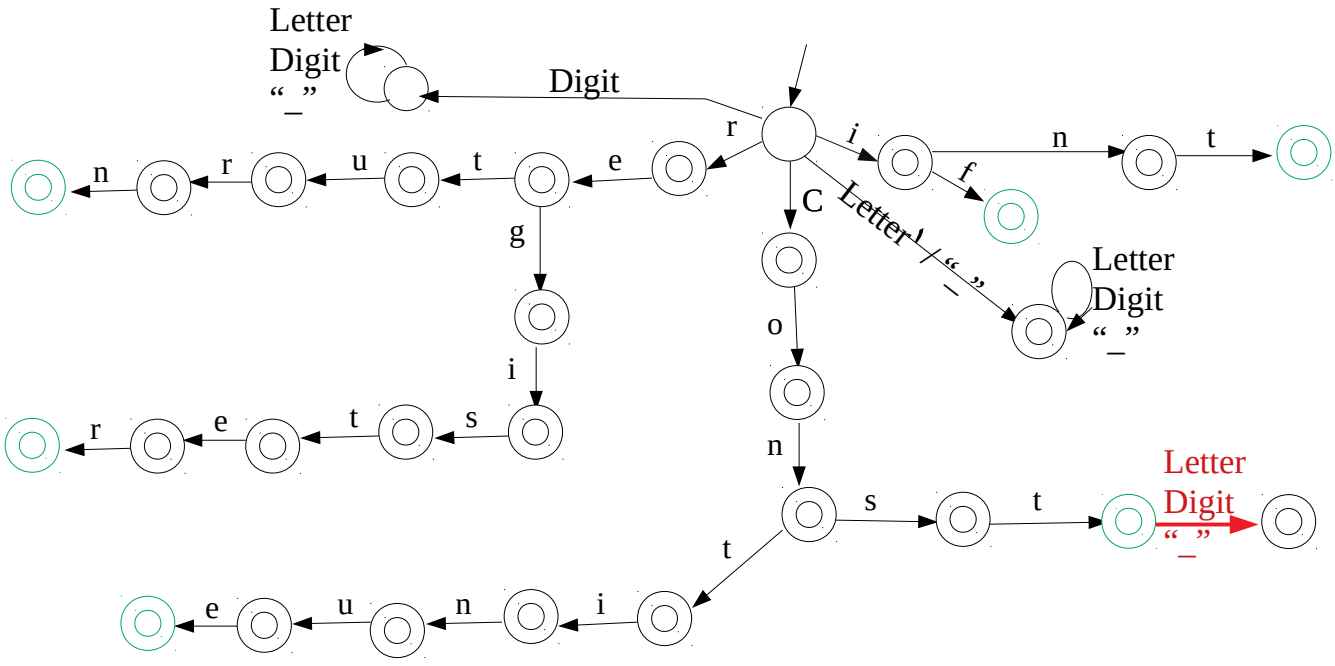
(توجه ! به جهت خلاصه تر شدن دیاگرام از عبارات Letter , Digit استفاده کرده ام.)

ب)



(توجه ! به جهت خلاصه تر شدن دیاگرام ، برخی یال ها رسم نشده‌اند ... این یال ها در اصل همگی به یک استیت هالت یا تله می‌روند و FSM در آن استیت گیر می‌افتد [به معنای وجود یک طوقه.])

ج)



توجه ۱ : خروجی FSM در final state های سبز رنگ ، بیانگر تشخیص کلمات کلیدی , int , return , register , if , continue , const در زبان C است و در final state های مشکی رنگ ، بیانگر تشخیص یک شناسه ی معمولی است.

توجه ۲ : منظور از Letter[1] تمام حروف {a,b,..,z,A,B,…,Z} به جز {i , c , r}

توجه ۳ : به جهت خلاصه تر شدن دیاگرام ، برخی یال ها رسم نشده‌اند ... این یال ها شامل سایر حروف[1] و ـــ و ارقام در اصل همگی به یک final state می‌روند که بیانگر تشخیص یک شناسه ی معمولی است.(به عنوان مثال یک یال به رنگ قرمز رسم شده است پس به این ترتیب از مابقی استیت ها هم یال مشابهی به این final state وارد می‌شود و در اصل خروجی FSM در این final state های مشکی رنگ به معنای تشخیص یک شناسه ی معمولی است)

۶- توکن های A, B , C به صورت زیر مشخص شده اند.فرض کنید از استراتژی longest match و سپس با اولویت ترتیب تعریف برای مشخص کردن(مانند lex) نوع توکن استفاده می‌شود .خروجی رشته‌های ورودی داده شده چیست ؟

TOKEN_A cd^*a^*

TOKEN_B c^*a^*cd

TOKEN_C c^*b

الف) cdccd

ب) caccbd

پاسخ :

الف)

TOKEN_A

TOKEN_B

ب)

TOKEN_A

TOKEN_C

d

---

1 {a,b,..,z,A,B,…,Z} - {حروفی که به ازای آن‌ها از استیت مورد نظر یال خارج شده است}

۷- یک برنامه ی lex بنویسید که آدرس یک فایل و یک کلمه را به عنوان ورودی دریافت کرده سپس به عنوان خروجی در خط اول تعداد کل رخدادهای کلمه ی مذکور در فایل و در هر یک ازخطوط بعد ابتدا شماره خط رخداد و سپس با جداساز : خط حاوی کلمه را در خروجی چاپ کند .

برای مثال اگر کلمه ی hello در خطوط ۶ و ۱۵ و ۱۹ فایل تکرار شده است ، خروجی باید مطابق زیر باشد : 3

6 : hello world

15 : there is a hello world here

19 : hello world again

پاسخ :

```
%{
   #include <iostream>
   #include <string.h>
   #include <vector>
   using namespace std;

   // Global Variables
   int count=0;
   string pattern;
   vector<string> output;
%}

%option noyywrap

line     .*
newline  \n

%%
{line}    {
          if(string(yytext).find(pattern)!= string::npos)
            {
              char str[100];
              sprintf(str,"%d : %s",yylineno,yytext);
              output.push_back(str);
              count++;
            }
          }

{newline}   {yylineno++;}
%%

int main(int argc, char* argv[])
{
  // argv[1] is a file name
  // argv[2] is a pattern
  yyin = fopen(argv[1],"r");
  pattern = argv[2];
  yylex();
  cout << count << endl;
  for(int i=0;i<output.size();i++)
    cout << output[i] << endl;
  return 0;
}
```

۸- یک برنامه ی lex بنویسید که آدرس یک فایل .cpp را به عنوان ورودی دریافت کرده و تمام کامنت های موجود در آن را حذف کرده و کد به دست آمده را مجدداً در فایل مذکور ذخیره کند. (کامنت های تک خطی با // شروع شده و کامنت های چند خطی بین /* و */ قرار میگیرند.)

پاسخ :

```
%{
        #include <stdio.h>
        #include <string.h>
%}

%x C_COMMENT
%option noyywrap

%%
"/*"          { BEGIN(C_COMMENT); }
<C_COMMENT>"*/" { BEGIN(INITIAL); }
<C_COMMENT>.    { }
"//".*                          { }
%%

int main(int argc,char* argv[])
{
        yyin = fopen(argv[1],"r");
        yyout = fopen(argv[1],"r+");
        yylex();
        return 0;
}
```

۹- یک برنامه ی lex بنویسید که آدرس یک فایل .cpp را به عنوان ورودی دریافت کرده و سپس در هرخط از خروجی نام هر یک از variable های تعریف شده در برنامه را چاپ کند. (نوع variable ها میتواند string , int , float , char باشد و variable میتوانند به صورت تکی همانند int vat =1 یا چند تایی همانند char x,y,z تعریف شده باشد.)

پاسخ :

```
%{
 #include <iostream>
 #include <vector>
 using namespace std;
 vector<string> sample;
%}

%option noyywrap

prototype     "int"|"float"|"char"|"bool"|"string"
sample_regex   ({prototype})[\t\ ]+[^";"\n]+";"

%%
{sample_regex}     { sample.push_back(yytext); }
.                  ;
\n                 ;
%%

int main(int argc,char* argv[])
{
 // argv[1] is a file name
 yyin = fopen(argv[1],"r");
 yylex();
 for(int i=0;i<sample.size();i++)
 {
  if(sample[i].find("int")<=sample[i].size())
   sample[i].erase(0,3);
  else if(sample[i].find("float")<=sample[i].size())
   sample[i].erase(0,5);
  else if(sample[i].find("char")<=sample[i].size() || sample[i].find("bool")<=sample[i].size())
   sample[i].erase(0,4);
  else if(sample[i].find("string")<=sample[i].size())
   sample[i].erase(0,6);
```

```
int new_id=0;
  int value=0;
  int limit=sample[i].size();
  for(int j=0;j<limit;j++)
  {
   if(sample[i].find(" ")==0)
    {
     sample[i].erase(0,1);
    }
   else if(sample[i].find("=")==0)
    {
     sample[i].erase(0,1);
     value=1;
    }
   else if(sample[i].find(",")==0)
    {
     sample[i].erase(0,1);
     value=0;
     new_id=1;
    }
   else if(sample[i].find(";")==0)
    {
     value=0;
     cout << endl;
     break;
    }
   else if(value)
    {
     sample[i].erase(0,1);
    }
   else
    {
     if(new_id)
     {
      cout << endl;
      new_id=0;
     }
     cout << sample[i][0];
     sample[i].erase(0,1);
    }
  }
 }
 return 0;
}
```