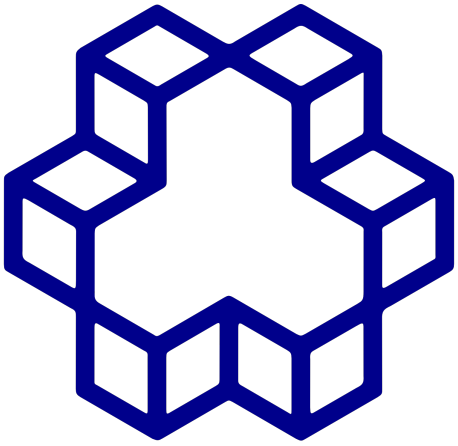
به نام خدا



دانشکده ریاضی
دانشگاه صنعتی خواجه نصیر الدین طوسی

شرح فاز دوم پروژه کامپایلر

پروژه درس کامپایلر
استاد دامن افشان

آفرین آخوندی ۴۰۱۱۵۲۸۳
عماد پورحسنی ۴۰۱۱۶۶۲۳

۱ مقدمه

این فایل، مشخصات پیاده سازی و جزئیات فاز دوم پروژه درس کامپایلر را شرح میدهد. تحلیل گر های لغوی و نحوی که در ادامه به شرح پیاده سازی و ساز و کار آن پرداخته شده است، توسط عماد پورحسنی و آفرین آخوندی تهیه شده است.
تحلیل گر لغوی به کمک سازنده تحلیل گر لغوی flex آماده شده است.
تحلیل گر نحوی به کمک سازنده تحلیل گر نحوی bison آماده شده است.
اطلاعات بیشتر را میتوانید در مخزن گیتهاب پروژه بررسی کنید.

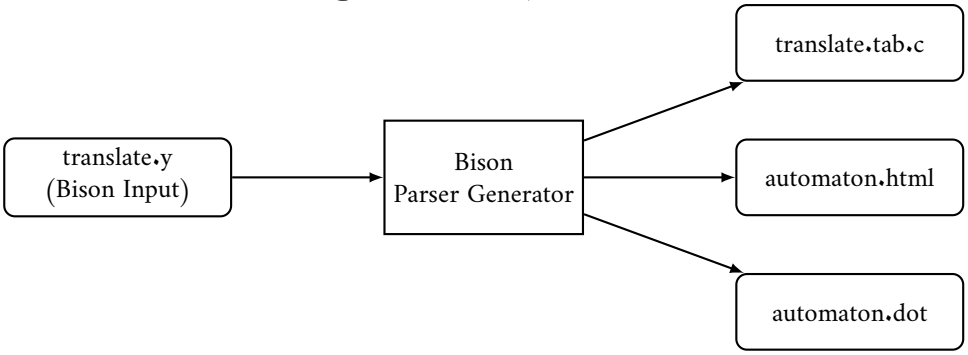
۲ پیاده سازی

در این بخش به ریز جزئیات پیاده سازی می پردازیم. ساختار دایرکتوری پروژه به صورت زیر است :

```
~/.../Compiler4031 >>> tree -r -L2
.
├── symboltable
│   ├── symtab.h.bak
│   ├── symtab.h
│   ├── symtab.c.bak
│   ├── symtab.c
│   ├── README.md
│   └── changelog.md
├── specification
│   ├── token_list.txt
│   ├── specification.ebnf
│   ├── specification.bnf
│   ├── README.md
│   └── diagram
├── README.pdf
├── README.md
├── projectSpecification.pdf
├── parser
│   ├── translate.y
│   ├── translate.tab.h
│   ├── translate.tab.c
│   ├── parser
│   ├── output.txt
│   ├── lex.yy.c
│   ├── input.txt
│   ├── automaton.png
│   ├── automaton.html
│   └── automaton.dot
├── LICENSE
└── lexer
    ├── TestCases
    ├── README.md
    ├── patterns.txt
    ├── output.txt
    ├── Makefile
    ├── lex.yy.c
    ├── lex.l
    └── input.txt
```

۱.۲ فایل translate.y

فایل مشخص کننده توکن ها ، ترتیب و اولویت عملگر ها (Associativity & Precedence) و همچنین گرامر مورد نظر و action هایی است که در هر مرحله reduction باید انجام شود. همچنین ساخت و استفاده از symboltable در این فایل اتفاق می افتد (دلیل این که چرا در فایل lex.l جدول نماد ها را نمیسازیم در بخش ۵.۲ توضیح داده شده است.)



این فایل ورودی برنامه bison است که سه خروجی دارد، خروجی translate.tab.c برنامه سی است که به کمک gcc و فایل های lex.yy.c و symtab.c پارسر را تولید میکند.

```
~/.../parser >>> bison --graph=automaton.dot -d --html=automaton.html translate.y
```

شکل ۱: کامند استفاده شده برای ایجاد سه فایل خروجی

درمورد translate.tab.c به طور کامل در بخش بعد صحبت شده است. درمورد دو فایل automaton.html و automaton.dot نیز در بخش های بعدی صحبت شده است.

۲.۲ فایل translate.tab.c

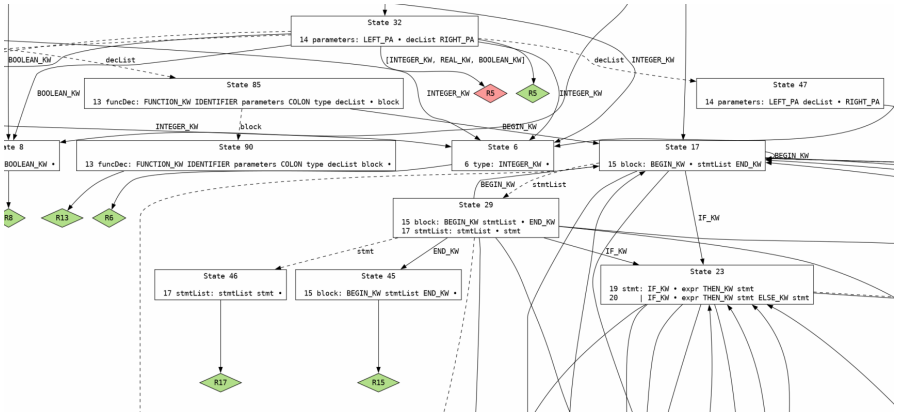
این فایل خروجی اصلی bison است که شامل کد سی برای پیاده‌سازی پارسر می‌باشد. این کد به صورت خودکار توسط bison تولید می‌شود و شامل چندین بخش مهم است. در قلب این فایل، یک stack قرار دارد که براساس گرامر تعریف شده در فایل translate.y عمل می کند. این ماشین با استفاده از جدول‌های پارس LALR(۱) که توسط bison تولید شده‌اند، تصمیم می گیرد که در هر مرحله چه عملی را انجام دهد. ساختار اصلی این فایل شامل موارد زیر است:
تعریف توابع و data structure مورد نیاز برای پارسر، جدول‌های حالت که مسیر پارس کردن را مشخص می کنند، توابع مدیریت خطا برای گزارش ارور های مربوط به مرحله تحلیل نحوی، و پیاده‌سازی اعمال معنایی که در فایل translate.y تعریف شده‌اند. این فایل همچنین با کد تولید شده توسط flex (lex.yy.c) و فایل تعریف جدول نمادها (symtab.c) ارتباط برقرار می کند تا یک پارسر کامل را تشکیل دهد.

۳.۲ فایل automaton.html و automaton.png

این دو فایل نمایش بصری اتوماتای پشته‌ای هستند که bison برای parse کردن استفاده می کند. فایل automaton.html یک نمایش تعاملی از اتوماتا را فراهم می کند که می‌توان در مرورگر مشاهده کرد. این نمایش شامل تمام حالت‌ها، انتقال‌ها و اعمال reduction است که پارسر می‌تواند انجام دهد. در نمایش تصویری، هر راس نشان‌دهنده یک حالت در پارسر LALR(۱) است و یال های بین راس ها نشان‌دهنده انتقال‌های ممکن بین حالت‌ها هستند.
با استفاده از این فایل ها میتوان:

- مسیر پارس کردن عبارات مختلف را دنبال کرد
- تشخیص داد که پارسر در کدام حالت‌ها تصمیم به کاهش (reduce) یا انتقال (shfit) می گیرد
- درک بهتری از نحوه حل conflict های shfit/reduce توسط bison به دست آورد

فایل automaton.dot نیز همین اطلاعات را در قالب گراف DOT ارائه می‌دهد که می‌تواند توسط ابزارهایی مانند Graphviz به تصاویر استاتیک تبدیل شود.
تصویر automaton.png با استفاده از ابزار graphviz تولید شده و قابل مشاهده است.



شکل ۲: بخشی از automaton کشیده شده در فایل automaton.png

```
~/.../parser >>> dot -Tpng automaton.dot -o automaton.png
```

شکل ۳: کامند مورد است فاده برای تولید automaton.png از automaton.dot

۴.۲ جدول نماد ها

جدول نماد ها به کمک hashtable پیاده سازی شده است و شرح آن در فایل symtab.c قابل مشاهده است. این هش تیبل به صورت خطی کار کرده و اعداد صحیح و حقیقی را در مرحله تحلیل لغوی ذخیره میکند.
در مرحله تحلیل نحوی ، شناسه های در زمان declaration شناسایی شده و با scope و type و kind مشخص در جدول نماد ها قرار میگیرند.
جدول نماد نهایی توسط parser در خروجی print میشود . (دقت کنید در فایل output قرار نمی گیرد.)

۵.۲ تغییرات در lex.l و برهمکنش آن با translate.y

تابع main از فایل lex.l برداشته شده و حالا main در فایل translate.y قرار دارد. فایل translate.tab.h که هدر فایل translate.tab.c است در فایل lex.l ، include میشود. و حالا lex.l توکن های مشخص شده در translate.y را به عنوان خروجی برمیگرداند، دلیل این کار این است که driver ما همان parser باشد و بتواند با تعاریف خودش از lexer درخواست token بکند.

۳ ساختن و اجرا کردن

برای ساخت و اجرای پروژه‌ی پارسر، یک سیستم ساخت مبتنی بر configure و Makefile پیاده‌سازی شده است. این سیستم امکان ساخت خودکار و مدیریت وابستگی‌های پروژه را فراهم می‌کند.

۱.۳ configure

اسکریپت configure وظیفه‌ی بررسی پیش‌نیازها و آماده‌سازی محیط برای ساخت پروژه را بر عهده دارد. این اسکریپت شامل بخش‌های زیر است:

```
#!/bin/bash
echo "Checking for required programs..."

# Check for GCC
if ! command -v gcc >/dev/null 2>&1; then
    echo "Error: gcc compiler not found"
    exit 1
fi
```

این اسکریپت به طور سیستماتیک موارد زیر را بررسی می‌کند:

- وجود کامپایلر gcc
- نصب بودن ابزار bison برای تولید پارسر
- نصب بودن ابزار flex برای تولید لکسر
- وجود هدرفایل‌های ضروری سیستمی

در صورت عدم وجود هر یک از موارد فوق، اسکریپت با پیام خطای مناسب خاتمه می‌یابد. برای اجرای اسکریپت configure، دستور زیر استفاده می‌شود:

```
./configure
```

۲.۳ ساختار Makefile

Makefile طراحی شده برای این پروژه شامل چندین بخش اصلی است که در ادامه تشریح می‌شوند:

۱.۲.۳ تعریف متغیرهای اصلی

متغیرهای پایه‌ای برای تنظیم ابزارها و پرچم‌های کامپایل به صورت زیر تعریف شده‌اند:

```
CC = gcc
YACC = bison
LEX = flex
CFLAGS = -Wall -g
LDFLAGS = -lfl
```

این متغیرها امکان پیکربندی مجدد ابزارها و گزینه‌های کامپایل را بدون نیاز به تغییر در سایر بخش‌های Makefile فراهم می‌کنند.

۲.۲.۳ قوانین تولید

قوانین اصلی برای تولید فایل‌های خروجی به شرح زیر پیاده‌سازی شده‌اند:

```
translate.tab.c translate.tab.h: translate.y
    $(YACC) -d translate.y

$(LEXER_SRC): $(LEXER_DIR)/lex.l translate.tab.h
    $(LEX) -o $(LEXER_SRC) $(LEXER_DIR)/lex.l
```

این قوانین به ترتیب عملیات زیر را انجام می‌دهند:

- تولید کد پارسر از فایل گرامر با استفاده از bison
- تولید کد لکسر از فایل قوانین لکسیکال با استفاده از flex
- کامپایل فایل‌های C تولید شده
- لینک کردن فایل‌های شیء و تولید فایل اجرایی نهایی

۳.۲.۳ مدیریت پروژه

برای مدیریت فایل‌های تولید شده، دستور clean به صورت زیر تعریف شده است:

```
clean:
    rm -f $(TARGET) $(OBJECTS) translate.tab.c translate.tab.h
```

این دستور امکان حذف تمامی فایل‌های تولید شده و شروع مجدد فرآیند ساخت را فراهم می‌کند.
دستورات اصلی برای استفاده از Makefile عبارتند از:

- ساخت پروژه: make
- پاک‌سازی فایل‌های تولید شده: make clean

سیستم ساخت طراحی شده از ویژگی تشخیص وابستگی‌ها بهره می‌برد و تنها فایل‌هایی را مجدداً می‌سازد که تغییر کرده‌اند. این ویژگی باعث بهینه‌سازی زمان ساخت و تسهیل فرآیند توسعه می‌شود.
در کل برای ساخت و اجرای برنامه ابتدا اسکریپت configure را اجرا کنید ، سپس make را اجرا کنید و نهایتا فایل parser آماده برای اجرا است.
در نظر داشته باشید که تمام این مراحل باید در دایرکتوری parser باشید.

۳.۳ نمونه ورودی و خروجی

نمونه ورودی برنامه :

```
program prg1;

integer num, divisor,quotient;
begin
 num := 61;
 divisor := 2;
 quotient := 0;
 if num=1 then
  return false;
 else if num=2 then
  return true;
 while divisor<(num/2) do
 begin
  quotient:=num/divisor;
  if divisor * quotient=num then
   return false;
  divisor:=divisor+1;
 end
 return true;
end
```

نمونه خروجی برنامه :

```
Emad Pourhassani -> 40116623
Afarin Akhoundi  -> 40115283
----------------------------
6 type -> INTEGER_KW
9 varList -> IDENTIFIER
10 varList -> varList COMMA IDENTIFIER
10 varList -> varList COMMA IDENTIFIER
4 decs -> type varList SEMICOLON
2 decList -> decs
12 funcList -> epsilon
```

```
33 expr -> INTEGER_NUMBER
18 stmt -> IDENTIFIER ASSIGN_OP expr SEMICOLON
16 stmtList -> stmt
33 expr -> INTEGER_NUMBER
18 stmt -> IDENTIFIER ASSIGN_OP expr SEMICOLON
17 stmtList -> stmtList stmt
33 expr -> INTEGER_NUMBER
18 stmt -> IDENTIFIER ASSIGN_OP expr SEMICOLON
17 stmtList -> stmtList stmt
38 expr -> IDENTIFIER
44 relop -> EQ_OP
33 expr -> INTEGER_NUMBER
31 expr -> expr relop expr
36 expr -> FALSE_KW
23 stmt -> RETURN_KW expr SEMICOLON
38 expr -> IDENTIFIER
44 relop -> EQ_OP
33 expr -> INTEGER_NUMBER
31 expr -> expr relop expr
35 expr -> TRUE_KW
23 stmt -> RETURN_KW expr SEMICOLON
19 stmt -> IF_KW expr THEN_KW stmt
20 stmt -> IF_KW expr THEN_KW stmt ELSE_KW stmt
17 stmtList -> stmtList stmt
38 expr -> IDENTIFIER
42 relop -> LT_OP
38 expr -> IDENTIFIER
33 expr -> INTEGER_NUMBER
28 expr -> expr DIV_OP expr
32 expr -> LEFT_PA expr RIGHT_PA
31 expr -> expr relop expr
38 expr -> IDENTIFIER
38 expr -> IDENTIFIER
28 expr -> expr DIV_OP expr
18 stmt -> IDENTIFIER ASSIGN_OP expr SEMICOLON
16 stmtList -> stmt
38 expr -> IDENTIFIER
38 expr -> IDENTIFIER
27 expr -> expr MUL_OP expr
44 relop -> EQ_OP
38 expr -> IDENTIFIER
31 expr -> expr relop expr
36 expr -> FALSE_KW
23 stmt -> RETURN_KW expr SEMICOLON
19 stmt -> IF_KW expr THEN_KW stmt
17 stmtList -> stmtList stmt
38 expr -> IDENTIFIER
33 expr -> INTEGER_NUMBER
29 expr -> expr ADD_OP expr
18 stmt -> IDENTIFIER ASSIGN_OP expr SEMICOLON
17 stmtList -> stmtList stmt
15 block -> BEGIN_KW stmtList END_KW
24 stmt -> block
21 stmt -> WHILE_KW expr DO_KW stmt
17 stmtList -> stmtList stmt
35 expr -> TRUE_KW
```

```
23 stmt -> RETURN_KW expr SEMICOLON
17 stmtList -> stmtList stmt
15 block -> BEGIN_KW stmtList END_KW
1 start -> PROGRAM_KW IDENTIFIER SEMICOLON decList funcList block
```