بسمه تعالی

سید مهدی مهدوی مرتضوی - ۴۰۰۳۰۴۹۰

گزارش نهایی پروژه اصول طراحی کامپایلرها - زمستان ۱۴۰۳

مقدمه و مشخصات کلی پروژه

این پروژه شامل طراحی یک کامپایلر ساده با استفاده از ابزارهای Flex و Yacc میباشد. این کامپایلر قادر است عبارات محاسباتی شامل عملگر های ریاضی (+، -، *، /) و پرانتز را (با شرکت پذیری و اولویت تعیین شده) تجزیه و تحلیل کرده، three address code را تولید کند و نتایج نهایی را محاسبه و چاپ نماید. پروژه شامل سه فایل اصلی است:

۱. **Parser.y:** فایل مربوط به تحلیل گر نحوی یا Syntax analyzer
۲. **Scanner.l:** فایل مربوط به تحلیل گر لغوی یا Lexical analyzer
۳. **common.h:** فایل هدر مشترک بین هر دو کد (برای مشخص کردن نوع توکن های عددی (توضیح در ادامه))

ساختار و منطق فایل‌ها

فایل هدر (common.h):

این فایل شامل یک تعریف ساختاری (`struct`) به نام `Factor` است که به عنوان یک داده عمومی برای ذخیره اطلاعات متغیرها و مقادیر استفاده می‌شود و مشخص میکند که آیا توکن از نوع عدد، یک عدد است یا یک متغیر temporary (موقت)؟

- **ساختار Factor:**
  - `is_temp`: این flag نشان می‌دهد که آیا متغیر یک متغیر موقت است یا یک عدد؟ (مثلا آیا عدد 1 هست یا متغیر موقت t1)
  - `Value`: مقدار عددیِ متغیر موقت (temporary) یا عددی که مستقیماً در عبارت وجود دارد .

این فایل برای اشتراک‌گذاری داده های نوع `Factor` بین فایل های Scanner.lex و Parser.y استفاده می‌شود که نقش کلیدی در تولید کد three address دارد (در واقع کلید محاسبات رشته ورودی میباشد).

فایل Scanner.l:

۱. **هدف:** این فایل برای تحلیل لغوی عبارت ورودی و استخراج توکن ها طراحی شده است و مسئول شناسایی توکن‌های موجود در نمونه رشته ورودی است. این توکن‌ها بعد از استخراج به فایل تحلیلگر نحوی (گرامری) ارسال می‌شوند.

۲. **توکن‌های تعریف‌شده:**
   - `id`: شناسایی متغیرها (مانندx ، y، variable و ...)
   - NUMBER: شناسایی اعداد صحیح
   - عملگرها: +، - ، × (*) و ÷ (/)؛ به ترتیب اولویت از سمت راست
   - سایر کاراکترهای استفاده شده: (،) ، = و ;
   - فاصله‌های خالی (`white space` ها) نادیده گرفته می‌شوند.

۳. **منطق و عملکرد فایل:**
   - ابتدا الگوهای مختلف (Patterns) برای توکن‌ها تعریف می‌شوند (طبق سینتکس استاندارد flex)
   - در صورت شناسایی هر الگو، مقدار مربوطه به متغیر `yylval` اختصاص داده شده و نوع توکن به تحلیل گر نحوی (Parser.y) ارسال می‌شود (اگر توکن از نوع ID باشد، `yylval.str = yytext` و اگر توکن از نوع NUMBER باشد، مقدار`yylval` برابر است با `atoi (yytext)` که برابر است با مقدار صحیح `yytext` (وقتی که خود `yytext` رشته ای از نوع عدد صحیح است) و `atoi` هم یک تابع استاندارد در زبان C میباشد که مخفف ASCII to Integer هست.
   - کاراکترهای غیرمجاز شناسایی و پیام خطا (از نوع `Lexical Error`) چاپ می‌شود.

فایل Parser.y:

۱. **هدف:** این فایل برای تحلیل نحوی (سینتکس) رشته ورودی و همچنین تولید three address code طراحی شده است (همانطور که دیده میشود، در اول این فایل، توکن ها و نوع آنها مشخص گردیده است)؛ همچنین SDT استفاده شده از نوع s-attributed بوده و تمامی attribute ها، از نوع synthesized میباشند (هم `$$` ها و هم `$` های عددی) چون اصولا فایل lex، از l-attributed SDT پشتیبانی نمیکند.

۲. **گرامر استفاده‌شده:** گرامر استفاده شده قادر است عبارات محاسباتی شامل اعداد صحیح، عملگرهای جمع، تفریق، ضرب و تقسیم، پرانتزها، و عملیات انتصاب (assign) را تجزیه کند. گرامر به صورت مبهم تعریف شده، اما با استفاده از قوانین زیر اولویت عملگرها و شرکت‌پذیری آنها مشخص شده است:

- + و – (دارای **اولویت بالاتر**): شرکت‌پذیری **راست**
- * و / (**اولویت پایین تر**): شرکت‌پذیری **چپ**
- مشخص کردن اولویت بندی عملگر ها و شرکت پذیری آنها (تعریف در خطوط پایین تر به منزله اولویت بالاتر):

- `%left MULT, DIV`
- `%right PLUS, MINUS`

- مشخص کردن متغیر شروع و تایپ متغیر (non-terminal) های استفاده شده با استفاده از سه خط کد زیر:

- `%start stmt`
- `%type <str> stmt`
- `%type <val> expr`

۳. **توابع اصلی:**
- `get_lvalue`: شناسایی متغیر (ID) سمت چپ عبارت در انتصاب.
- `generate_code`: تولید کننده کد three address برای هر عبارت (به صورت نوشته شده برای عبارات دارای متغیر موقت یا عدد ثابت (با استفاده از `Factor struct` تعریف شده و Flag و Value تعریف شده برای آن در فایل هدر)؛ این تابع در action مربوط به production مربوطه، فراخوانی شده است.
- `calculate_result`: محاسبه نتیجه (مقدار نهایی عبارت ورودی) با در نظر گرفتن اولویت و شرکت‌پذیری عملگرهای تعریف شده (همچنین بیان ارور `Division by zero` برای عملگر تقسیم (در صورت تقسیم یک متغیر موقت یا عدد بر عدد 0)).
- `reverse_number`: معکوس کردن اعداد صحیح که تک رقمی و مضرب عدد ۱۰ نیستن (مطابق با نیاز پروژه و قواعد عنوان شده).
- `yyerror`: چاپ و نمایش خطاهای نحوی (از نوع `Syntax error`).

۴. **مدیریت مقادیر موقت :**مقادیر موقت (مانند `t1` و `t2` و تا هر تعداد متغیر موقتی که مورد استفاده قرار میگیرد) در آرایه `temp_results` ذخیره شده و برای محاسبات بعدی استفاده می‌شوند (اگر متغیر از نوع متغیر موقت (`t`) بود، در ایندکس متناظر همان متغیرِ `t` در آرایه ذخیره میگردند).

۵. **نحوه اجرا:**
- عبارت ورودی خوانده می‌شود.
- توکن‌ها توسط فایل Flex شناسایی شده و به فایل yacc (bison) ارسال می‌شوند.
- تحلیل نحوی (سینتکسی یا گرامری) بر رشته ورودی انجام شده و کد three address تولید می‌شود.
- نتیجه نهایی (با استفاده از توابع نامبرده شده) محاسبه و در نهایت با استفاده از action تعیین شده برای production مربوط به stmt، نتایج نهایی نمایش داده می‌شوند.
- برای اجرای و دیدن نتایج، دستورات زیر (به همراه یک نمونه مثال برای رشته ورودی) در در یک محیط یونیکسی (با استفاده از شبیه ساز command line لینوکس در ویندوز با نام wsl) تست شده اند:

- `bison -d -o parser.c Parser.y`
- `flex -o lexer.c Scanner.l`
- `gcc -o compiler parser.c lexer.c -lm`
- `echo "b = 20*(24/6)+45 -60;" | ./compiler`

تست کیس‌ها و نتایج خروجی

تست کیس اول:

```
---------- Compiler output ----------
t1 = 2 * 0;
t2 = t1 + 10;
t3 = 23 - t2;
b  = t3;

----------- Final results -----------
Assignment: b = 32 -(2 *0) + 10;
Temporary variable of result: t3
The Final Result: 31
```

تست کیس دوم:

```
---------- Compiler output ----------
t1 = 10 - 76;
t2 = 254 + t1;
t3 = 65 + 1;
t4 = t2 * t3;
variable  = t4;

----------- Final results -----------
Assignment: variable =452 +10- 67* 56+1 ;
Temporary variable of result: t4
The Final Result: 64185
```

تست کیس سوم:

```
---------- Compiler output ----------
t1 = 32 * 42;
t2 = 5 + 54;
t3 = t2 - 61;
t4 = t1 / t3;
c  = t4;

----------- Final results -----------
Assignment: c = 23* 24/(5 +45)-16;
Temporary variable of result: t4
The Final Result: 301
```

جمع‌بندی

این پروژه یک کامپایلر ساده طراحی کرده است که قادر به تحلیل عبارات محاسباتی، تولید کد three address و محاسبه نتایج نهایی (طبق قوانین، قواعد و نمونه ورودی های عنوان شده) می‌باشد. در طراحی و پیاده سازی این پروژه، از فایل های نوع Flex و Yacc (به ترتیب برای چک کردن لغات و سینتکس (به اصطلاح نحو کردن) رشته ورودی) و همچنین یک هدر فایل مشترک طبق نیاز ها و موارد گفته شده استفاده گردیده است که با اجرای مرحله به مرحله دستورات پروژه طبق سینتکس استاندارد آن (که با توجه به هدر فایل استفاده شده در فایل l. (که parser.h هست) میباشد)، نتایج نهایی نمایش داده خواهند شد.