به نام خدا

پروژه کامپایلر

علی رحیمی - ۹۹۳۲۱۲۰

پاییز ۱۴۰۳

پروژه شامل ۲ فایل اصلی می باشد:

1. Lexer.l
2. Parser.y

این پروژه به کمک دو ابزار bison و flex پیاده سازی شده است.

Lexer.l

این فایل مسئول شناسایی توکن‌ها در کد ورودی است. توکن‌ها واحدهای زبانی پایه‌ای هستند که parser از آن‌ها استفاده می‌کند.

توکن های شناسایی شده شامل:

- Num
- ID
- ADD / SUB / MUL / DIV
- White space
- Assign
- Semicolon
- Parenthesis

هستند که با توجه به ورودی کاربر تجزیه می شوند.

Parser.y

این بخش با استفاده از دستورات گرامری، ساختار ورودی را تایید و کد میانی تولید می کند. از Bison استفاده شده و parsing را اجرا می کند.

قواعد گرامری برنامه:

```
%%
program
    : stmt
    | program stmt
    ;
```

```
stmt
  : expr SEMICOLON
    {
      printf("Statement completed\n"
    }
  | assignment SEMICOLON
    {
      printf("Statement completed\n");
    }

  ;
```

```
assignment
  : ID '=' expr
    {
      double val = getValue($3);
      vars[$1[0] - 'a'] = val;
      printf("%s = %s \n",  $1, $3);
      printf("%d;\n", (int)val);
    }
```

```
expr : expr ADD expr
    {
      sprintf($$, "t%d", tempCounter);
      double val1 = getValue($1);
      double val2 = getValue($3);
      double result  = val1 + val2;
      double finalResult  = needReverese(result);

      temps[tempCounter] = finalResult;

      printf("%s = %s + %s;\n", $$, $1, $3);
      tempCounter++;
    }
```

```
  | expr SUB expr
    {
      sprintf($$, "t%d", tempCounter);
      double val1 = getValue($1);
      double val2 = getValue($3);
      double result  = val1 - val2;
      double finalResult  = needReverese(result);

      temps[tempCounter] = finalResult;

      printf("%s = %s - %s;\n", $$, $1, $3);
      tempCounter++;
    }
```

```
| expr DIV expr
  {
    sprintf($$, "t%d", tempCounter);
    double val1 = getValue($1);
    double val2 = getValue($3);

    if(val2 == 0) {
      fprintf(stderr, "Division by zero error\n");
      exit(1);
    }

    double result  = val1 / val2;
    double finalResult  = needReverese(result);

    temps[tempCounter] = finalResult;
    printf("%s = %s / %s;\n", $$, $1, $3);
    tempCounter++;
  }
```

```
| expr MUL expr
  {
    sprintf($$, "t%d", tempCounter);
    double val1 = getValue($1);
    double val2 = getValue($3);
    double result  = val1 * val2;
    double finalResult  = needReverese(result);

    temps[tempCounter] = finalResult;
    printf("%s = %s * %s;\n", $$, $1, $3);
    tempCounter++;
  }
```

```
  | LPAREN expr RPAREN
    {
      strcpy($$, $2);
    }
```

```
| NUM
  {
    double result = atof($1);
    double number = needReverese(result);
    sprintf($$, "%g", number);
  }
```

```
  | ID
    {
      strcpy($$, $1);
    }
```

```
| SUB expr %prec UMINUS
  {
    sprintf($$, "t%d", tempCounter);
    double val2 = getValue($2);
    double result  = -val2;
    double finalResult  = needReverese(result);

    temps[tempCounter] = finalResult;
    printf("%s = -%s;\n", $$, $2);
    tempCounter++;
  }
;
```

```c
int main(int argc, char **argv) {
    memset(vars, 0, sizeof(vars));
    memset(temps, 0, sizeof(temps));

    if (argc > 1) {
        if (!(yyin = fopen(argv[1], "r"))) {
            perror(argv[1]);
            return 1;
        }
    }

    yyparse();

    if (argc > 1) {
        fclose(yyin);
    }

    return 0;
}
```

توابع کمکی:

```c
int reverse(int number) {
    int reversed = 0;
    while (number != 0) {
        reversed = reversed * 10 + (number % 10);
        number /= 10;
    }
    return reversed;
}
```

این تابع برای معکوس کردن اعداد استفاده می شود

```c
double needReverese(double value) {

    int sign = (value < 0) ? -1 : 1;
    double absVal = fabs(value);

    int intPart = (int)absVal;


    if (intPart != 0 && (intPart % 10) != 0) {
        intPart = reverse(intPart);
    }

    return sign * (intPart);
}
```

این تابع برای بررسی اینکه آیا نتیجه یا عبارت نیاز به معکوس کردن دارد استفاده می شود

```c
double getValue(const char* str) {
    if(str[0] == 't') {
        int index = atoi(str+1);
        return temps[index];
    } else if(str[0] >= 'a' && str[0] <= 'z') {
        return vars[str[0] - 'a'];
    } else {
        return atof(str);
    }
}
```

این تابع برای استخراج مقدار عبارت استفاده می شود

نکات پیاده سازی:

برای شروع برنامه و انجام عملیات parsing باید ساختار مناسبی انتخاب نمود. من از ساختار union برای نگهداری id, num, nonterminal ها استفاده کردم.

```
%union {
    char id[500];
    char num[500];
    char nonTerminal[500];
}
```

توکن ها و تایپ های برنامه شامل

```
%token <num> NUM
%token <id>  ID
%token ADD SUB MUL DIV
%token SEMICOLON
%token LPAREN RPAREN
%token UMINUS

%type <nonTerminal> program stmt expr assignment
```

می باشند که در فایل پارسر آمده است.

اولویت های برنامه طبق داکیومنت با استفاده از right و left مشخص شده است

```
%right '='
%left  MUL DIV
%right ADD SUB
%right UMINUS
```

ساختار کلی برنامه بدین شکل است که بر اساس گرامر، اکشن آن ها اجرا می شود و حاصل عبارت بدست می آید.

```
expr : expr ADD expr
    {
      sprintf($$, "t%d", tempCounter);
      double val1 = getValue($1);
      double val2 = getValue($3);
      double result  = val1 + val2;
      double finalResult  = needReverese(result);

      temps[tempCounter] = finalResult;

      printf("%s = %s + %s;\n", $$, $1, $3);
      tempCounter++;
    }
```

به عنوان نمونه در دستور به علاوه ما نیاز است که مقدار ۱ و مقدار ۳ را عملیات + را انجام دهیم، پس مقادیر آن ها با استفاده از تابع کمکی getValue گرفته می شود و نتیجه ذخیره می شود. حال طبق شرط سوال معکوس کردن حین ورودی و حین انجام محاسبات باید صورت بگیرد پس دوباره از توابع کمکی استفاده کرده و تابع needReverese را بر روی خروجی اعمال کرده و خروجی نهایی بدست می آید. چون نیاز داریم که کد های میانی را ذخیره و نمایش دهیم پس در temps ذخیره می کنیم.
پرینت اکشن هم در ادامه آمده است.

برای اجرا کردن این کد (روی مک) هم نیاز به این دستورات است:

1. flex lexer.l
2. bison -d parser.y
3. gcc -o compiler lex.yy.c parser.tab.c
4. ./compiler

باید حتما flex و bison و gcc نصب شده روی سیستم باشند.