



简单翻译器

编译原理实验

组长：软件 22 王雨婷 2012013328

组员：软件 22 王华清 2012013317

组员：软件 22 孙 浩 2012013315



2015-6-30

编译原理

实验环境和实验选题

操作系统: windows 7 旗舰版 JAVA: JDK 1.8

ANTLR: antlr 3.5.2 以及 antlrworks-1.4.3

源语言: C++ 目标语言: javascript 测试程序: 四则运算计算

实验目的及要求

实现一种语言的编译器，将输入的源语言的程序翻译成目标语言程序。

实验内容

首先根据 C++ 和 javascript 的语法差异设计文法，利用 antlr 工具生成 Lexer 和 Parser 的 java 文件，编写语法制导翻译，通过 antlrworks 生成语法分析树文法进行检测、验证。之后再输入 C++ 的四则运算程序，输出对应的 javascript 程序。

源程序——四则运算

```
相关引用
设定全局变量
int main() {
    while(true) {
        读入输入的算式，调用 play 函数，输出计算结果。}
}
double play(char flag) { //操作式子, flag 为式子结束标识
    for 式子未结束 {
        if 遇到数字 通过一个 while 对数字进行提取操作。
        else if 遇到括号 产生一个新的式子。
        else if 遇到操作符 栈为空则入栈，否则比较操作符与栈顶操作符优先级，
            并选择对当前优先的算式进行计算。}
    while 栈内操作符非空 {
        根据栈内操作符完成剩下算式的计算。}
    return 计算结果
}
```

C++ 和 javascript 的语法对比

首先我们仔细研究了两种语言之间的相似性和差异性。主要对比如下：

函数声明：

C++——有返回值的声明，有参数类型声明。

```
returnType functionID (paraType para, ..., paraType para) {}
```

Javascript——无返回值，内部有 return 操作，无需传递参数类型。

```
functionID = function(para list) {}
```

函数传参：

C++——需要传递参数类型，支持引用类型&。

Javascript——无需传递参数的数据类型，除对 object 类型之外不支持引用的类型。

变量类型

C++——明确区分数据类型。

Javascript——除了对数组是 array 类型之外基本不区分数据类型，都为 var 类型。

构建数组

C++——arrayType arrayID[] = {para}。

Javascript——var arrayID = new Array(para)。

整体结构

C++需要 main 函数作为程序的入口，对语法规则限制更多，更加规范。而 Javascript 没有特定的 main 函数作为程序的起始地址，整个程序布局由 function 定义语句块作为分界，相对来说限制较少，语言更灵活。

文法设计（部分）

以下是我们设计的文法。在设计的过程中我们注意消除了左递归，提取了左公因子，而且文法的名称非常易懂。

```
program(程序) -> statements(语句群)
statements(语句群) -> statement(语句) statements(语句群) | e
statement(语句) -> type func_or_var(函数或者变量)
                    | "#include<" ID ">"
                    | 'using' 'namespace' ID ';'
                    | COMMENT(注释)
func_or_var -> dec_func(函数) | dec_var(变量)
//函数声明
dec_fun -> dec_func_name ' {' func_implement(函数实现) ' } ' semi_colon
dec_func_name -> ID ' (' dec_param(参数们) ' ) '
dec_param(参数们) -> type ID dec_param_(参数递归) | e
dec_param_(参数递归) -> , dec_param | e
func_implement(函数实现) -> block
//变量声明
dec_var -> ID dec_array ' ; ' (数组定义)
          | ID dec_array ' = ' array_value ' ; ' (数组定义+赋值)
          | ID dec_expression dec_var_ (正常变量)
dec_array -> [ INT ]
array_value -> { INT ints }
dec_var_ -> ' ; ' | ' , ' dec_var
dec_expression -> expr_value | e
```

工具使用

本次实验中用到了开源的语法分析器——antlr，由上述的文法设计编译好文法文件，通过 antlr 处理 .g 文件可生成对应的词法分析器和语法分析器的 java 文件。

除此之外，为了验证文法设计的合法性，我们利用 antlrworks 对文法产生语法分析树，从而检测、验证设计的文法。

支持语法

在本次编译器中我们实现了如下的语法结构：

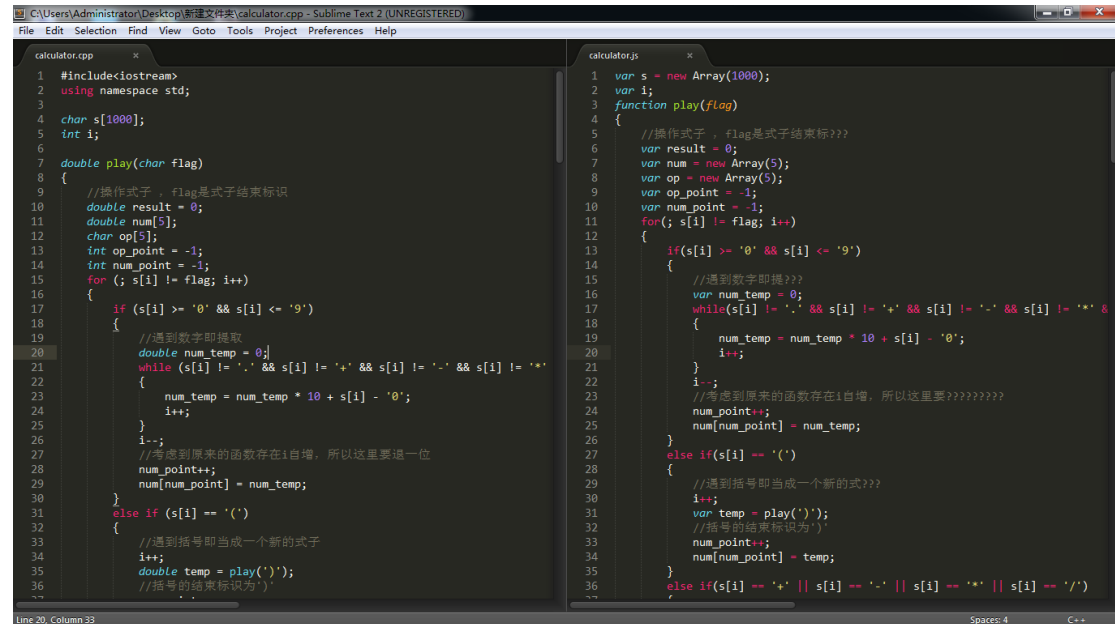
函数定义，函数调用；全局变量，局部变量；数组定义、初始化；数组下标取值\；while 循环结构；for 循环结构；if-else if-else 选择分支结构；switch-case 选择分支结构；int, double, char, void 类型；逻辑运算符；算术运算符；++、--；输入、输出；以及注释的转化。

在编译器中我们把动作内嵌到产生式中，让编译的过程更加高效。

除此之外，我们还实现了 javascript 的格式化。在产生式中增加 tab 属性(继承属性)，记录每行代码需要缩进的值，最后输出时候对于 js 语句进行调整，使得输出的语法可读性更强。

实验结果

左边是 C 语言，右边是 Javascript。



The image shows a side-by-side comparison of C and JavaScript code for a calculator. The left pane, titled 'calculator.cpp', contains C code using `std::string` and `std::vector` to parse and evaluate an expression. The right pane, titled 'calculator.js', contains JavaScript code using arrays to store the expression and a recursive function `play` to evaluate it. Both implementations handle basic arithmetic operations and parentheses.

```
calculator.cpp
1 #include<iostream>
2 using namespace std;
3
4 char s[1000];
5 int i;
6
7 double play(char flag)
8 {
9     //操作式子 , flag是式子结束标识
10    double result = 0;
11    double num[5];
12    char op[5];
13    int op_point = -1;
14    int num_point = -1;
15    for (; s[i] != flag; i++)
16    {
17        if (s[i] >= '0' && s[i] <= '9')
18        {
19            //遇到数字即提取
20            double num_temp = 0;
21            while (s[i] != '.' && s[i] != '+' && s[i] != '-' && s[i] != '*' && s[i] != '/')
22            {
23                num_temp = num_temp * 10 + s[i] - '0';
24                i++;
25            }
26            //考虑到原来的函数存在i自增, 所以这里要退一位
27            num_point++;
28            num[num_point] = num_temp;
29        }
30        else if (s[i] == '(')
31        {
32            //遇到括号即当成一个新的式子
33            i++;
34            double temp = play('(');
35            //括号的结束标识为')'
36            i++;
37        }
38        else if (s[i] == '+' || s[i] == '-' || s[i] == '*' || s[i] == '/')
39        {
40            op_point++;
41            op[op_point] = s[i];
42            i++;
43        }
44    }
45    //计算结果
46    result = num[0];
47    for (int j = 1; j <= num_point; j++)
48    {
49        if (op[j] == '+') result += num[j];
50        else if (op[j] == '-') result -= num[j];
51        else if (op[j] == '*') result *= num[j];
52        else if (op[j] == '/') result /= num[j];
53    }
54    return result;
55 }
```

```
calculator.js
1 var s = new Array(1000);
2 var i;
3 function play(flag)
4 {
5     //操作式子 , flag是式子结束标识???
6     var result = 0;
7     var num = new Array(5);
8     var op = new Array(5);
9     var op_point = -1;
10    var num_point = -1;
11    for (; s[i] != flag; i++)
12    {
13        if (s[i] >= '0' && s[i] <= '9')
14        {
15            //遇到数字即提取???
16            var num_temp = 0;
17            while(s[i] != '.' && s[i] != '+' && s[i] != '-' && s[i] != '*' && s[i] != '/')
18            {
19                num_temp = num_temp * 10 + s[i] - '0';
20                i++;
21            }
22            //考虑到原来的函数存在i自增, 所以这里要????????
23            num_point++;
24            num[num_point] = num_temp;
25        }
26        else if (s[i] == '(')
27        {
28            //遇到括号即当成一个新的式子???
29            i++;
30            var temp = play('(');
31            //括号的结果标识为')'
32            num_point++;
33            num[num_point] = temp;
34        }
35        else if (s[i] == '+' || s[i] == '-' || s[i] == '*' || s[i] == '/')
36        {
37            op_point++;
38            op[op_point] = s[i];
39            i++;
40        }
41    }
42    //计算结果
43    result = num[0];
44    for (int j = 1; j <= num_point; j++)
45    {
46        if (op[j] == '+') result += num[j];
47        else if (op[j] == '-') result -= num[j];
48        else if (op[j] == '*') result *= num[j];
49        else if (op[j] == '/') result /= num[j];
50    }
51    return result;
52 }
```