LP Compiladors: Muntanyes!

Albert López Alcácer

October 30, 2017

1 Introducció

Aquest document té l'objetiu de donar algunes explicacions sobre la implementació i instruccions d'execució de la pràctica de compiladors de l'assignatura LP: Muntanyes. En aquesta, s'ha fet un compilador per tal de poder interpretar un llenguatge simple per definir muntanyes, i fer operacions amb aquestes.

2 Implementació

2.1 Definició de tokens del llenguatge

Els tokens que s'han definit ens permetran fer l'anàlisi lèxica de l'input del compilador i es faran servir per poder aplicar la gramàtica i construir l'AST posterior. La columna de la descripció fa referència al significat que es voldrà que tinguin finalment aquests tokens.

Token	Descripció	Token	Descripció
AND "AND"	Operador lògic conjunció	ALM "#"	Clau referència a variable
OR "OR"	Operador lògic disjunció	ID "[a-zA-Z][a-zA-Z0-9]*"	Identificador o nom de variable
NOT "NOT"	Operador lògic negació	NUM "[0-9]+"	Constant numèrica
IF "if"	Clau començament de condició	MULT "*"	Multiplicador de part de muntanya
ENDIF "endif"	Clau final de condició	PUJADA "/"	Part pujada de muntanya
WHILE "while"	Clau començament de bucle	CIM "-"	Part cim de muntanya
ENDWHILE "endwhile"	Clau final de bucle	BAIXADA "\"	Part baixada de muntanya
ASIG "is"	Clau assignació	CONCAT ";"	Clau concatenació de muntanyes
MSIM "Match"	Clau funció de igualtat d'alçada	PLUS "+"	Operador de suma de enters
WSIM "Wellformed"	Clau funció de muntanya ben formada	LPAR "("	Clau començament de expressió booleana o parametres de funció
HSIM "Height"	Clau de funció d'alçada de muntanya	RPAR ")"	Clau tancament de expressió booleana o parametres de funció
DSIM "Draw"	Clau de funció per a dibuixar muntanya	GTHAN ">"	Operador relacional major amb enters
PSIM "Peak"	Clau de funció per a creació de Pic	LTHAN "<"	Operador relacional menor amb enters
VSIM "Valley"	Clau de funció per a creació de vall	EQUAL "=="	Operador relacional igual amb enters
CSIM "Complete"	Clau de funció de autocompletat de muntanya	COMA ","	Clau separació de paràmetres en funcions

L'ordre de en la especificació dels tokens és important. Per exemple, per tal de poder reservar les paraules claus (AND, OR, NOT, Peak, Valley, etc...), aquests cal posar-los abans del token de identificador de variable.

2.2 Gramàtica

La gramàtica que ens permet fer l'anàlisi sintàctic és la següent. L'arbre abstracte resultant ens servirà per poder analitzar la semàntica del programa i realitzar la execució tot mostrant possibles errors.

```
atom: NUM | (ALM! | )ID | height | match | wellformed;
expr: atom (PLUS^  atom)*;

match: MSIM^ LPAR! mountain COMA! mountain RPAR!;
wellformed: WSIM^ LPAR! mountain RPAR!;
height: HSIM^ LPAR! mountain RPAR!;
peakvalley: (PSIM^ | VSIM^) LPAR! expr COMA! expr COMA! expr RPAR!;

part: expr ((MULT^ (PUJADA | CIM | BAIXADA)) | );
mountexpr: part | peakvalley;
mountain: mountexpr (CONCAT^ mountexpr)*;
assign: ID ASIG^ mountain;

boolexpr4: atom ((GTHAN^ | LTHAN^ | EQUAL^) atom | );
boolexpr3: (NOT^ | ) boolexpr4;
boolexpr2: boolexpr3 (AND^ boolexpr2 | );
boolexpr: boolexpr2 (OR^ boolexpr | ) ;

condic: IF^ LPAR! boolexpr RPAR! mountains ENDIF!;
iter: WHILE^ LPAR! boolexpr RPAR! mountains ENDWHILE!;
draw: DSIM^ LPAR! mountain RPAR!;
complete: CSIM^ LPAR! ID RPAR!;
mountains: (assign | condic | draw | iter | complete)* << #0 = createASTlist(_sibling); >>;
input: mountains "@";
```

Un programa consisteix en un conjunt d'istruccions de tipus: assignació, condicional, iteració, dibuixar o instrucció de completar muntanya. Per tant, quan volem executar un seguit d'instruccions fem servir la gramàtica "mountains": El programa en si mateix, el cos d'un bucle while i als condicionals.
Per a avaluar una expressió booleana cridem a "boolexpr". Cal conservar la precedència habitual dels operadors, de manera que fem servir una regla o nivell diferent per a cada operador.
Per a construir muntanyes fem servir la gramàtica "mountain". De manera que iterativament anem concatenant expressions "mountexpr", que poden ser una part o una definició de pic/vall. Com que les parts d'una muntanya podrien ser una tripleta(1*/) o una muntanya existent, a la gramàtica "part" fem servir "expr" (que ens permetrà retornar un valor numeric constant o de variable, o una variable de muntanya existent). Posteriorment caldrà analitzar el que es retorna realment en temps d'execució.
Per a les funcions de Draw(), Height(), Wellformed() i Match() fem servir la gramàtica "mountain". I per a Complete() limitem que nomes es pugui completar una muntanya existent.
Finalment destacar que les expressions atòmiques "atom" podrien retornar un valor numèric, booleà, o una muntanya. Caldrà mostrar erros en segons quin casos.

2.3 Interpretació

En aquesta secció procedeixo a donar algunes explicacions sobre la implementació de la part d'interpretació.
Per a enmagatzermar valors numèrics i muntanyes faig servir dos maps diferents:
map<string,string> m;
map<string,int> v;
Amb això, les muntanyes es guarden directament com a cadenas de caràcters (com s'imprimiran) i no pas la seva definició.
Així doncs, per a mirar si una muntanya està ben formada es fa servir una expressió regular en la que es mira si tota la muntanya està formada per tripletes de literals que defineixen un pic o una vall:

- bool mountainWellformed(string mountain) { } en la que s'ha fet servir la llibreria <regex> disponible a partir de C++11 (g++ 4.9) per mirar que la montanya estigui formada per tripletes de pics o bé valls.

Per a completar una muntanya, es mira si l'últim caràcter és una pujada (per afegir "-\"), una baixada (per afegir "-/") o un cim (per afegir "/" o bé "\"). Per a mirar si la opció escollida forma una muntanya completa es crida a mountainWellformed():

- string completeMountain(string mountain) { }

Per a executar el programa es fa servir la funció "execute". Aquesta es la que va executant instrucció a instrucció fent (node->right). Per tant cal detectar quan es fa: assignació, dibuixar muntanya, completar muntanya, condiciónal if o bucle while.
Per un altre banda tenim la funció "evaluate". Aquesta s'encarrega d'evaluar les diferents parts de l'arbre sintàctica. El string retornat es fa servir quan s'espera una muntanya o part de muntanya com a valor de reton, així com num i cond quan esperem un valor numèric o booleà.
Segons la gramàtica, en les diferents crides a evaluate() cal asegurar-se que el valor que retorna la crida recursiva es del tipus correcte, en cas contrari es llença un error i la instrucció no s'executa.

- void execute(AST *a) { }
- string evaluate(AST *a, int& num, bool& cond) { }