Université Gustave Eiffel

BERNIER Aurélien
BLONDEL Lucas

Compiler : User Manual

Summary

I - Preamble

Presentation

The objective of the project was to create a compiler for a given language derived from C, based on the syntax analysis project completed earlier in the year.

Functioning

The compilation of a program occurs in several stages :

- First, the program is analyzed lexically using flex, then syntactically using bison. Any lexical or syntactical errors detected will be reported, and the program will not be compiled. If no errors are found, an abstract syntax tree will be constructed from the TPC program.

- Several passes through the tree will then be performed to identify potential semantic errors. The first pass of the abstract syntax tree builds the symbol tables and reports redefinition semantic errors (such as a variable already declared or a function redefined). For this, the table.h module is used.

- A second pass checks for other semantic errors like type mismatches, undeclared main, or undeclared variables. For this, the sem.h module is used.

- A final pass handles the translation of the tree into a NASM program. For this, the trad.h module is used.

II - TPC language

Identifiers

Any identifier used as a variable or parameter in a program must be declared before its use and in the appropriate declaration section.

A global variable and a function cannot have the same name.

A local variable and a parameter of the same function cannot have the same name.

Functions

Every program must include the special function main, which starts the execution and must return an int.

The use of return statements must match the return type of the function in which they appear. A function call cannot be used as an expression if the return type of the called function is void.

Function arguments are passed by value, as pointers are not allowed in TPC. Both direct and indirect recursive functions are permitted. Any function declaration without parameters must include "void".

Pre-written functions are available: getchar(), getint(), putint(INT), putchar(CHAR).

The functions getchar() and getint() allow reading a character and an integer in decimal notation entered by the user via the keyboard. They return the character or integer as a return value. If the first character read is neither a digit nor a minus sign, getint() terminates the program execution, and the program returns the value 5, indicating an input/output error.

The functions putchar() and putint() allow displaying a character and an integer on the standard output, passed as parameters. The integer is displayed in decimal notation. The functions putchar() and putint() do not return a value.

The compiler directly writes these functions in NASM, and the TPC code cannot redefine them.

Types

The typing of expressions is similar to C :

- The int type represents signed integers coded on 4 bytes.

- Any expression of type char that is used in an operation is implicitly converted to int. For example, if count is of type int, count='a' is correct and should not trigger a warning.

- Any expression can be interpreted as a boolean, with the convention that 0 represents "false" and any other value represents "true", and the result of any boolean operation is of type int. In particular, the negation operator produces a result of 1 when applied to 0.

- If an expression of type int is assigned to an LValue of type char, the compiler emits a warning but produces a functional target program. For example, if my_char is of type char, my_char=97 should trigger a warning.

- In function calls, each actual parameter is treated as an assignment regarding typing; the return Exp statement is also considered an assignment. For example, in a function that returns an int, return 'a' is correct, but in a function that returns a char, return 97 should trigger a warning.

Arrays

The semantics for arrays are the same as in the C language, except as noted below.

In a variable declaration of array type, the size of the array must be strictly positive. An array of size zero is a semantic error.

In a statement, the only operators applicable to an array are the brackets, as in duration[2], and passing the array as an argument in a function call, as in bissextile(duration,12);.

Applying any other operator to an array, or using it as a boolean, as in if (duration) return;, is a semantic error.

III - Using the compiler

Compilation and launch

To compile the compiler, use the command : make

To use the compiler, use : ./bin/tpcc [*OPTIONS*] < *[FILE*]
The possible options are :
-t / --tree to display the syntax tree
-h / --help which displays a short usage guide for the program
-s /--symtabs which displays the symbol tables

A test suite is available to test the compiler or to serve as an example of a program written in TPC.

To run it, simply use the command : bash test/test.bash, a test report will be generated and placed in the "test" directory, providing a score for the compiler.

Example of use

Let's use the compiler on a TPC program : ./bin/tpcc -s < test.tpc.

```
int tab[5], a;

int sum(int a, int b) {
    return a + b;
}

int main(void) {
    int b;
    b = 4;
    a = 2;
    putint(getint());
    return sum(a, b);
}
```

Figure 1: Program written in TPC

We then obtain the following symbol tables in the terminal :

```
Table 0 :
|Type : int array, Ident : tab, Taille : 5, Fonct : non, Adr : 0
|Type : int, Ident : a, Taille : 0, Fonct : non, Adr : 40
|Type : void, Ident : putint, Taille : 0, Fonct : oui, Adr : 48
|Type : int, Ident : getint, Taille : 0, Fonct : oui, Adr : 56
|Type : int, Ident : sum, Taille : 0, Fonct : oui, Adr : 64
|Type : int, Ident : main, Taille : 0, Fonct : oui, Adr : 72

Table 1 :
|Type : int, Ident : n, Taille : 0, Param : oui, Adr : -8

Table 2 :
VIDE

Table 3 :
|Type : int, Ident : a, Taille : 0, Param : oui, Adr : -8
|Type : int, Ident : b, Taille : 0, Param : oui, Adr : -16

Table 4 :
|Type : int, Ident : b, Taille : 0, Param : non, Adr : -8
```

Figure 2: Tables produced from the TPC program figure 1

We can note several elements :

- to simplify, we have conventionally decided that table 0 will be for global variables and the following tables will be for functions.

- the type of a variable or function is represented by an integer from 0 to 4. 0 corresponds to void, 1 to char, 2 to int, 3 to char array, and 4 to int array. We have added 2 different types for arrays to simplify type checking later.

- a function table without parameters or local variables is simply marked as VIDE (empty) (e.g., getint).

- the Taille field corresponds to the size of an array, so it will be 0 for a variable.

- in function tables, the param field indicates if the variable is a parameter. Similarly, in the global table, the func field indicates if it is a function.

- adr corresponds to the relative address of the variable in the function table. In the global table, it is a global address. To simplify translation to nasm, variable addresses in function tables are negative because local variable addresses are negative relative to rbp.

- by convention, functions are always listed after global variables in the global table, and parameters are always listed before local variables in function tables.

- function tables are in the same order as the function declarations.

After lexical, syntactic, and semantic analysis, the program is translated into NASM in a file named _anonymous.asm placed at the root of the project.

It can then be compiled using the command make bin/_anonymous and executed with ./bin/_anonymous.