A Programming Language Translator:
C to Forth:
Introduction

John E. Harbold

SVFIG – January 28, 2023

We present a programming language translator that will convert the C programming language to the Forth programming language

C to Forth: Introduction

- The C programming language is used in an enormous number of applications, libraries, operating systems, both general purpose and real-time. It would be nice to have this code converted to Forth, in essence why “reinvent the wheel”.
- The Forth programming language has a speed advantage over C because a C function has three sections, a stack setup, the code and a stack tear down. Forth does not have this handicap.

C to Forth: Introduction

- In the past, the 1980s and 1990s, there have been attempts to translate C to Forth by by using the parser generator, “yacc”, or the more modern, “bison”, and the associated lexical analyzer, “lex”, or the more modern, “flex” to create a C parser that will parse a C file and generate the associated Forth code. Remember, at this time was in the early days of C where the individual passes of the C compiler could not be run separately.
- Now, in the early 21st century, in the GNU Compiler Collection, (GCC), the C compiler, (gcc), the individual pass can be run to process any pre-processing directives, in particular, the #include directive to include any necessary declarations and pass 1 of the compiler that parses the resulting C code.

C to Forth: Introduction

- The first pass of the gcc will generate the necessary abstract syntax tree, (AST) describing parsed C code. The problem is finding the file that contains this information.
- There is another compiler, LLVM from the University of Illinois, in particular the C compiler, clang. This compiler will generate a human readable AST that can be analyzed to generate Forth code.
- Remember, the C language uses infix for any binary operators where the Forth language uses postfix. This is the reason for the AST.

A Simple C Program

```
uint32_t a = 1;

uint32_t b = 2;

uint32_t c;

int main(int ac, char*  av[])

{

    c = a + b;

    return 0;

}
```

Equivalent Forth Program

- variable a   1 a !
- variable b   2 b !
- variable c
- : main (ac av – status)
-    a @   b @   +   c !
-    0
- ;

Abstract Syntax Tree (AST)

- The AST describes the parsing of the first pass of the C compiler. It breaks down the individual C syntax into a more readable fashion.
- The following lines describe the individual AST name:
  - VarDecl – Variable Declaration
  - FunctionDecl – Function Declaration
  - ParmVarDecl – Parameter Variable Declaration
  - CompoundDecl – Compound Declaration
  - DeclRefExpr – Declaration Reference Expression
  - BinaryOperator – Binary Operator
  - ImplCastExpr – Implicit Cast Expression
  - ReturnStmt – Return Statement
- The following slides will show the individual AST members of the above example:

The First AST Statement
Variable Declaration

```
uint32_t a = 1;
```

```
|-VarDecl 0x154ee40 <globalVar.c:10:1, col:9> col:5 used a 'int' cinit
| `-IntegerLiteral 0x154eef0 <col:9> 'int' 1
```

variable a

1 a !

Variable Declaration

```
uint32_t b = 2;
```

```
|-VarDecl 0x154ee40 <globalVar.c:10:1, col:9> col:5 used b 'int' cinit
| `-IntegerLiteral 0x154eef0 <col:9> 'int' 2
```

variable b
2 b !

Variable Declaration

uint32_t c;

|-VarDecl 0x154ee40 <globalVar.c:10:1, col:9> col:5 used ***c 'int'***

variable c

Function Declaration

```
int main(int ac, char*  av[])
```

```
`-FunctionDecl 0x154f280 <line:18:1, line:57:1> line:19:1 main 'int (int, char **)'
  |-ParmVarDecl 0x154f048 <col:6, col:10> col:10 ac 'int'
  |-ParmVarDecl 0x154f160 <line:20:6, col:15> col:12 av 'char **':'char **'
```

```
: main  (ac av  -- status)
```

Compound Statement

c = a + b;

```
`-CompoundStmt 0x154f9e0 <line:21:1, line:57:1>
  |-BinaryOperator 0x154f3e0 <line:25:3, col:11> 'int' '='
  | |-DeclRefExpr 0x154f330 <col:3> 'int' lvalue Var 0x154efc8 'c' 'int'
  | `-BinaryOperator 0x154f3c0 <col:7, col:11> 'int' '+'
  |   |-ImplicitCastExpr 0x154f390 <col:7> 'int' <LValueToRValue>
  |   | `-DeclRefExpr 0x154f350 <col:7> 'int' lvalue Var 0x154ee40 'a' 'int'
  |   `-ImplicitCastExpr 0x154f3a8 <col:11> 'int' <LValueToRValue>
  |     `-DeclRefExpr 0x154f370 <col:11> 'int' lvalue Var 0x154ef28 'b' 'int'
```

a @   b @   +   c !

Return Statement

return 0;

```
`-ReturnStmt 0x154f9d0 <line:55:3, col:10>
  `-IntegerLiteral 0x154f9b0 <col:10> 'int' 0
```

0

The Last AST Statement

```
}
```

```
 ;
```

Further Work

- The program to analyze the AST and produce Forth Code will be described.
- The C programming language has an extensive syntax. Other syntax will be explored in the future.
- The Forth programming language has to be extended for other C syntax, especially the “union” and “enum”.
- Because the first pass of the C compiler does not execute any optimizations, optimization of the translated Forth code will be done on a per-Forth-word basis.

Questions?

See You Next Month