Compilation of C

Abdulrahman Alshammari, Dylan Chapp, Jaewoong Yoo

CISC 471/672 Compiler Construction, Fall 2016

University of Delaware

Language Overview

- C is an imperative, statically, weakly, typed language with manual memory management
- Software commonly implemented in C:
  - Operating systems
  - High performance scientific applications
  - Embedded systems

Key Language Features

- **Type System:**
  - **Static:** Type-safety properties of C programs can be determined at compile-time.
  - **Weak:** User can bypass the type-system
    e.g., pointer arithmetic or implicit type conversions
- **Manual memory management:**
  - Allows programmer good control over mapping of data structures to memory for performance-critical code

Compilation Stages with Example Outputs

C Source with Macros

```
1 #include<stdio.h>
2 #define GREETING "Hello world.\n"
3 int main(int argc, char** argv) {
4     printf(GREETING);
5 }
```

Macros
- Allows programmer to include needed library headers, define platform-specific constants, etc
- Not part of C grammar, hence preprocess before lex/parse

Pure C Source

```
f_chk (char * restrict,
estrict, va_list);
444 # 493 "/usr/include/stdio.h" 2 3 4
445 # 2 "main.c" 2
446
447 int main(int argc, char** argv) {
448     printf("Hello world.\n");
449 }
```

Preprocessor:
- Converts C source code containing macros—e.g., #define—into pure C source code

Clang AST

```
3> col:40 '__va_list_tag *':'__va_lis
:3:1, line:5:1> line:3:5 main 'int (i
536 |-ParmVarDecl 0x103858ae0 <col:10, col:14> col:14 argc 'int'
537 |-ParmVarDecl 0x103858b60 <col:20, col:27> col:27 argv 'char **'
538 `-CompoundStmt 0x103858e20 <col:33, line:5:1>
539   `-CallExpr 0x103858dc0 <line:4:5, col:20> 'int'
540     |-ImplicitCastExpr 0x103858da8 <col:5> 'int (*)(const char *, ...
541     | `-DeclRefExpr 0x103858ce8 <col:5> 'int (const char *, ...)' Fur
542     `-ImplicitCastExpr 0x103858e08 <line:2:18> 'const char *' <BitCas
543       `-ImplicitCastExpr 0x103858df0 <col:18> 'char *' <ArrayToPointe
544         `-StringLiteral 0x103858d48 <col:18> 'char [14]' lvalue "Hell
```

Lexer:
- Scan for valid C
- Scan for blocks of inline assembly

User-defined types
- Requires inter-mingling lexing and parsing
- If a definition of a new type is parsed, update lexer stage to accomodate

Parser:
- Hand-tuned recursive descent parser
- Constructs abstract syntax tree (AST) which can then be traversed to do various forms of semantic analysis

LLVM IR

```
-f80:128-n8:16:32:64-S128"
target triple = "x86_64-apple-darwin14.5.0"
4
5 @.str = private unnamed_addr constant [14 x i8] c"Hello world.\0A\00",
6
7 ; Function Attrs: nounwind
8 define i32 @main(i32 %argc, i8** %argv) #0 {
9   %1 = alloca i32, align 4
10  %2 = alloca i8**, align 8
11  store i32 %argc, i32* %1, align 4
12  store i8** %argv, i8*** %2, align 8
```

x86 Assembly

```
ext,regular,pure_instructions
, 10
                                    ## @main
8       push    rbp
9   Ltmp0:
10      .cfi_def_cfa_offset 16
11  Ltmp1:
12      .cfi_offset rbp, -16
13      mov rbp, rsp
14  Ltmp2:
15      .cfi_def_cfa_register rbp
16      sub rsp, 32
17      lea rax, [rip + L_.str]
18      mov dword ptr [rbp - 4], edi
19      mov qword ptr [rbp - 16], rsi
20      mov rdi, rax
21      mov al, 0
22      call    _printf
23      xor ecx, ecx
24      mov dword ptr [rbp - 20], eax ## 4-byte Spill
25      mov eax, ecx
26      add rsp, 32
27      pop rbp
28      ret
29      .cfi_endproc
30
31      .section    __TEXT,__cstring,cstring_literals
32  L_.str:                                 ## @.str
33      .asciz  "Hello world.\n"
34
35
36  .subsections_via_symbols
```

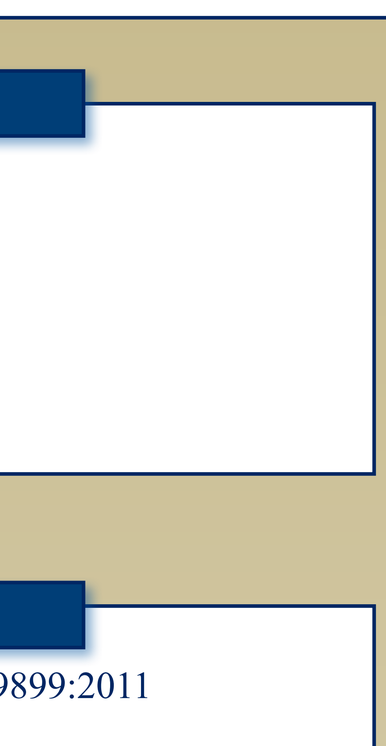
Code Generation:
- Generate assembly code from IR and from any blocks of inline assembly
- Rely on C Runtime Library to reduce code size

C Runtime Library
- Compiler and target architecture specific
- Contains low-level routines used by the compiler to create an executable that can interact with the OS through syscalls

Scoping Rules

Global vs. Block Scopes

Location in Memory



Challenges

References

- C11 Standard, ISO/IEC 9899:2011