

CSE 2421

The C Language – Part 1

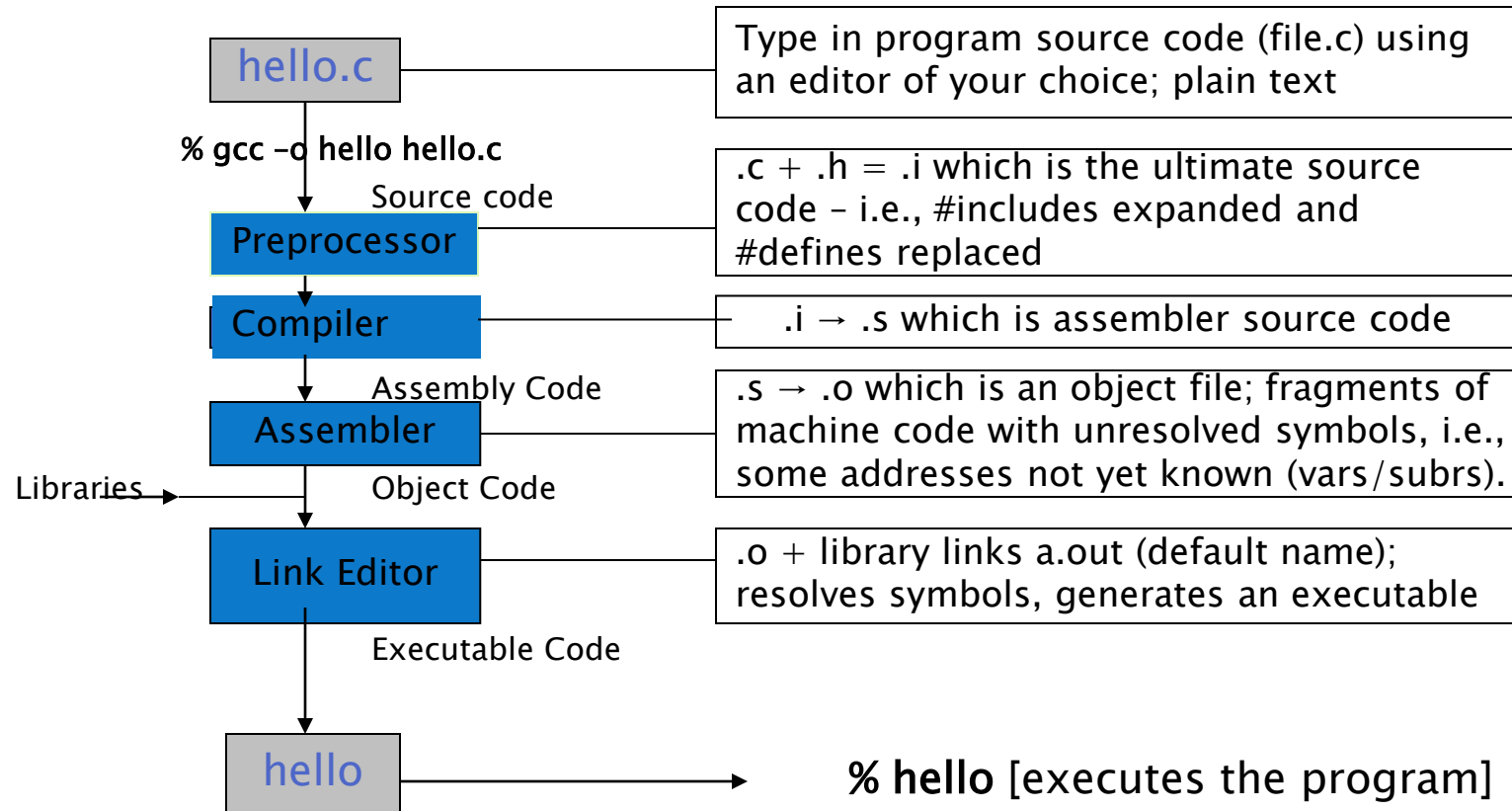
Required Reading:

Computer Systems: A Programmer's Perspective, 3rd Edition,


Chapter 1 thru Section 1.3

Chapter 2 thru Section 2.1.2

The compilation (or build) system



Down the Rabbit Hole!

PE¹⁰¹ a windows executable walkthrough  Ange Albertini
corkami.com

Dissected PE

simple.exe

header
technical details about the executable

sections
contents of the executable

DOS header
shows it's a binary

PE header
shows PE's metadata

optional header
executable information

data directories
pointers to extra structures (exports, imports...)

sections table
defines how the file is loaded in memory

code
link between the executable and (Windows) libraries

imports
link between the executable and (Windows) libraries

data
information used by the code

Hexadecimal dump

ASCII dump

Fields

Values

Explanation

Sections table

x86 assembly

Equivalent C code

Imports structures

Consequences

Strings

This is the whole file, however, most PE files contain more elements. Explanations are simplified, for conciseness.

version: 1.00 May 2012

Loading process

1 Headers

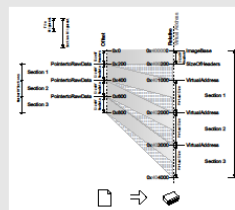
The DOS Header is parsed
The PE Header is parsed
(its offset is DOS Header's e_lfanew)
The Optional Header is parsed
(it follows the PE header)

2 Sections table

Sections table is parsed
(it is located at offset (OptionalHeader + SizeOfOptionalHeader))
It contains NumberOfSections elements
It is checked for validity with alignments:
FileAlignment and SectionAlignment

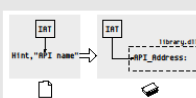
3 Mapping

The file is mapped in memory according to:
the ImageBase
the Sections table



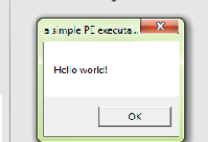
4 Imports

Data Directories are parsed
They follow the OptionalHeader
their number is NumberOfImports
Imports are always IAT
Imports are parsed
each descriptor specifies a DLLName
the DLL is loaded in memory
IAT and INT are parsed simultaneously
for each API in INT
its address is written in the IAT entry



5 Execution

Code is called at the EntryPoint
the calls of the code go via the IAT to the APIs



Notes

MZ HEADER aka DOS_HEADER

Starts with 'MZ' (initials of Mark Zbikowski MS-DOS developer)

PE HEADER aka IMAGE_FILE_HEADER / COFF file header

Starts with 'PE' (Portable Executable)

OPTIONAL HEADER aka IMAGE_OPTIONAL_HEADER

Optional only for non-standard PEs but required for executables

RVA Relative Virtual Address

Address relative to ImageBase (at ImageBase, RVA = 0)

Almost all addresses of the headers are RVAs

In code, addresses are not relative.

INT Import Name Table

Null-terminated list of pointers to Hint, Name structures

IAT Import Address Table

Null-terminated list of pointers

On file it is a copy of the INT


After loading it points to the imported APIs

HINT

Index in the exports table of a DLL to be imported

Not required but provides a speed-up by reducing look-up

C Language Overview

- Basic Data Types
 - Constants
 - Variables
 - Identifiers
 - Keywords
 - Basic I/O (covered in a separate set of slides)
 - Control structures (if, while, for, etc.) (covered in a separate set of slides)
- 

Basic Data Types

- Integer Types /* some may be slower than others */
 - char – smallest addressable unit, ***always*** 8 bits; each byte has its own address
 - short – not used as much; typically 16 bits
 - int – default type for an integer constant value; typically 32 bits
 - long – do you really need it?; counterintuitive, typically 32 or 64 bits today
 - long long – at least 64 bits on platforms that support it (not available in C90)
- Floating point Types – these are usually “inexact”
 - float – single precision (about 6 decimal digits of precision) (32 bits)
 - double – double precision (about 15 decimal digits of precision) (64 bits)
 - long double – about 30 decimal digits of precision (128 bits)
 - double is constant default unless suffixed with ‘f’
- **Note that variables of type char are guaranteed to always be one byte.**
- There is no fixed or maximum size for a type in C (except for char; otherwise, size depends on implementation), but the following relationships must hold:
 - `sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long)`
 - `sizeof(float) <= sizeof(double) <= sizeof(long double)`

C99 Adds Fixed-Size Types & More

- ▶ In `<stdint.h>` we find
 - `int32_t` and `uint32_t` are 32 bits
 - `int64_t` and `uint64_t` are 64 bits
- ▶ `long long int` is 64 bits on our system
- ▶ `_Bool` is now built in but in `<stdbool.h>` we find:
 - `bool` which is a type compatible with `int`
 - `true` gives us a fixed true value so that true values compare as equal
 - `false`

Derived Types

- Beside the basic types, there is a conceptually infinite number of derived types constructed from the fundamental types in the following ways:
 - arrays of data (variables or derived types) of a given type;
 - pointers to data of a given type;
 - structures containing a sequence of data (variables or derived types) of various types;
 - unions capable of containing any of various types.
- In general these ways of constructing new types (variables or derived types) can be applied recursively
 - An array of pointers to some type
 - An array of characters (i.e. a string)
 - Structures that contain pointers
 - And so on.

Constants

```
char          'A', 'B'
int           123, -1, 2147483647, 040 (octal), 0xab (hexadecimal)
unsigned int  123u, 2107433648, 040U (octal), 0X02 (hexadecimal)
long          123L, 0x1FFF1 (hexadecimal)
unsigned long 123ul, 0777UL (octal)
float         1.23F, 3.14e+0f
double        1.23, 2.718281828
long double   1.23L, 9.99E-9L
```

- Special characters
 - Not convenient to type on a keyboard
 - Use single quotes i.e. '\n'
 - Looks like two characters but is really only one

\a	alert (bell) character	\\	backslash
\b	backspace	\?	question mark
\f	formfeed	\'	single quote
\n	newline	\"	double quote
\r	carriage return	\ooo	octal number
\t	horizontal tab	\xhh	hexadecimal number
\v	vertical tab		

Declaration of Constants

- 2 ways to do it
 - Put the const keyword after the type keyword, or before the type keyword
 - **Note:** The compiler treats these as *variables* to which any assignment is invalid.
 - This means the declared **const** must be initialized with its (constant) value as part of the declaration, because the compiler will not allow statements which make assignments to it later! Treated as a read-only variable.
- Examples:
 - `float const PI = 3.141593f;`
 - `const float PI = 3.141593f; // Why do we do this?`
- Uppercase used for declared constants, and also for those with `#define` (see below) by convention.
- Symbolic constants (with the `#define` directive – below) can be used anywhere a literal constant can be used, but constants defined with the `const` keyword can only be used *where variables can be used*. More on this later (with examples).
- We will say more about *constants as function parameters*, *pointers to constants*, and *constant pointers* later.

Symbolic Constants

- A name that substitutes for a value that cannot be changed
- Can be used to define a:
 - Constant
 - Statement
 - Mathematical expression
- Uses a preprocessor directive:
- `#define <name> <value>`
 - `<name>` is a text string with no white space; `<value>` is any *text string* (so it can be a mathematical expression, for example `3.1415927 * r * r`)
 - REMINDER: No semi-colon is used for preprocessor directives.
- Coding convention is to use all capital letters for the name: `#define AREA 3.141593 * r * r`
- Can be used any place you would use the actual value
- All occurrences are replaced by the preprocessor before the program is compiled by the compiler.
- Examples:
 - The use of `EXIT_SUCCESS` in `hello.c` code
 - `#define PI 3.141593`
 - `#define TRUE 1 /* there are many true values */`

Variable Declarations

- Purpose: define a variable (can also be a constant) before it is used.
- Format: *type identifier (, identifier) ;* Note: the parentheses here indicate *any number of identifiers*, each preceded by a comma
- Initial value: can be assigned, but is not required (unless it is a constant)
 - `int i, j = 5, k;`
 - `char code, category;`
 - `int i = 123;`
 - `const float PI = 3.1415926535f;`
 - `double const PI = 3.1415926535;`
- Type conversion: aka, type casting
 - Casting “larger” types to “smaller” types is dangerous, and should be done with extreme caution!!!
 - To cast a variable to a different type explicitly, use: `(type) identifier`
 - `int i = 65;`
 - `char ch; /* range -128 to 127 */`
 - `ch = (char) i; /* What is the value of ch? */`

Identifier Naming Style

- Identifier Naming **Rules**: names for variables, constants, types and functions.
 - Can use a-z, A-Z, 0-9, and _ (i.e., alphanumeric, digits, and underscore)
 - Case sensitive
 - The first character must be a letter or _ (Don't use _ , though, because it is used for system purposes).
 - Keywords are reserved words, and may not be used as identifiers (See the following slide for C keywords)
 - **No guarantee** that any value past the 31st character will be recognized. (i.e will let you use more characters, but no guarantee that it will parse it.)
- Identifier Naming **Style** (the grader will enforce these)
 - Separate words with ' _ ' (this is the original style in C) **OR** capitalize the first character of each word after the first (e.g., char_count or charCount)
 - Use all UPPERCASE for symbolic constants or macro (code chunk) definitions.
 - **Be consistent. Be consistent. Be consistent.**
 - Be meaningful: Write “self-documenting code”; i.e., identifiers should give a clear idea of what a variable, constant, type or function is being used for.
- Sample Identifiers
 - i0, j1, student_name, studentName, student_score, studentScore...

Keywords – reserved identifiers

- Purpose: reserves a word or identifier to have a particular meaning
- The meanings of keywords — and, indeed, the meaning of the notion of keyword — differs widely from language to language.
- You shouldn't use them for any other purpose in a C program. They are allowed, of course, within double quotation marks (as part of a string to be assigned or printed, for example; this is not using an identifier, actually).

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while