CSCI 2132: Software Development

Introduction to C

Norbert Zeh

Faculty of Computer Science
Dalhousie University

Winter 2019

The C Programming Language

Originally invented for writing OS and other **system software**

Inventor: Dennis Ritchie

Characteristics:

- Optimized for **speed** and programming **close to machine**
- **Manual memory management**, no garbage collection
- **0-overhead rule:** What you don’t write doesn’t happen
- No safety checks:
  - E.g.: No out of bounds checks
  - You need to know what you are doing

A Simple C Program

hello.c

```
#include <stdio.h>

int main() {
  printf(“Hello, world!\n”);
  return 0;
}
```

Compiling a C Program

```
$ gcc hello.c
$ ls -l
-rwx------  1 nzeh  staff  8432  7 Feb 23:02 a.out
-rw-------  1 nzeh  staff    80  7 Feb 23:02 hello.c
$ ./a.out
Hello, world!
```

```
$ gcc -o hello hello.c
$ ls -l
-rwx------  1 nzeh  staff  8432  7 Feb 23:02 hello
-rw-------  1 nzeh  staff    80  7 Feb 23:02 hello.c
$ ./hello
Hello, world!
```

The Compilation Process

Preprocessor:

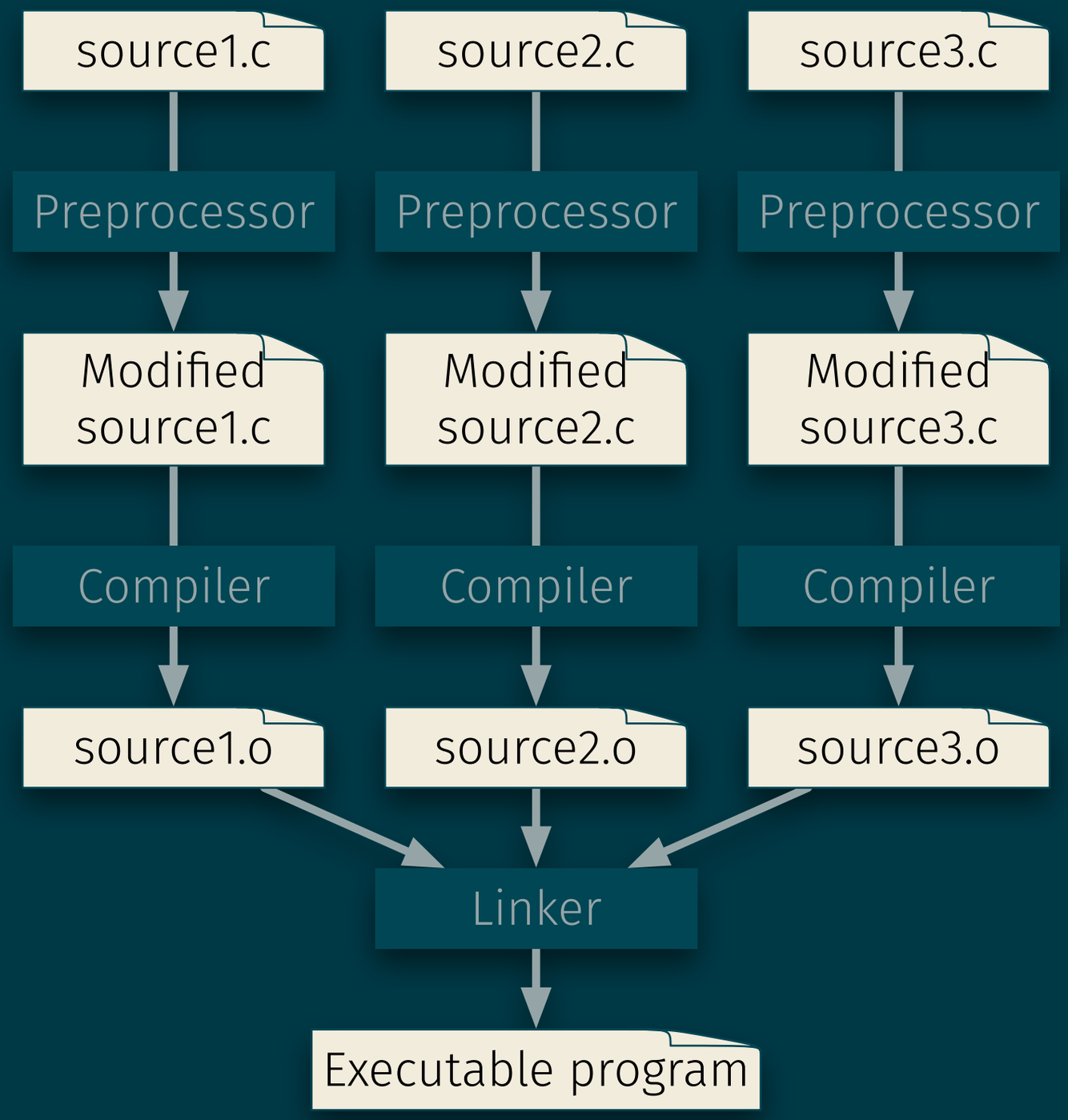
- Modify source code
- E.g., expand macros

Compiler:

- Translate source into object code

Linker:

- Combine one or more object files into executable code



General Form of a Simple Program

```
preprocessor directives

int main() {
  statements
}
```

```
#include <stdio.h>

int main() {
  printf("Hello, world!\n");
  return 0;
}
```

No arguments

Return type = int

preprocessor directive

main function

statements

Functions

= building blocks from which C programs are constructed

Function = named group of statements (for now)

“Special” kinds of functions:

- **Main function** = entry point of the program, called when the program is started
- **Library functions** = functions provided as part of the standard library

Nesting of functions is not allowed.

Statements

= commands to be executed

- Must end with a semicolon

Examples:

```
printf(“Hello, world!\n”);
```

```
return 0;
```

Printing Strings: `printf`

`printf` can be used to print string literals

String literal = sequence of characters and escape sequences enclosed by “ ... ”

Escape sequences (similar to Java):

- `\n` = newline
- `\t` = tab
- `\xHH` = character code in hexadecimal
- `\ooo` = character code in octal
- `\\` = literal backslash
- `\”` = literal “
- Others: `\r`, `\a`, `\b`, `\f`, `\v`, `\’`, `\?`

printf Examples

```
printf(“Hello, world!\n”);
```

```
printf(“Hello, world!”);
```

```
printf(“Hello,\nworld!\n”);
```

Comments

```
/* This is a one-line comment */
```

```
/* This is a
   multi-line
   comment */
```

```
// C99 also allows this style of comments
```

Variables

Variable = name for a memory location where data can be stored

Variables in C are statically typed:

- Type declared as part of program text
- Only values of this type can be assigned to variable
- Type casting allows us to cheat (use with care, as a last resort)
- Contrast with Python (anything can be stored in a variable)

Common types:

- `int` = integer
- `char` = character
- `float` = single-precision floating point number
- `double` = double-precision floating point number

Variable Declarations

```
int number_of_lines;
```

Type name
(before variable name)

Variable name
(after type name)

```
double length_in_inches;
```

Before C99, declarations must precede statements in function code.

C99 lifts this restriction. (It's still good practice.)

Operators

A rich and powerful set of operators was one of the innovations in C

Some operators (by decreasing precedence)**:**

- Unary (2): `+`, `-`, `++`, `--`, `!`, `~`
- Arithmetic (3): `*`, `/`, `%`
- Arithmetic (4): `+`, `-`
- Bitwise shifts (5): `<<`, `>>`
- Comparison (6): `<`, `>`, `<=`, `>=`
- Equality (7): `==`, `!=`
- Bitwise operations: `&` (8), `|` (10)
- Logical operations: `&&` (11), `||` (12)
- Assignment (14): `=`
- Update (14): `+=`, `*=`, `>>=`, `<<=`, `&=`, ...

Precedence can (of course) be overruled using parentheses.

Printing Variables: `printf`

`printf` allows us to print also variables by including placeholders in the string

Placeholders:

- `%d` = print an integer
- `%f` = print a single-precision float
- `%lf` = print a double-precision float
- `%s` = print a string
- `%c` = print a character
- `%.2f` = print single-precision float with 2 digits after decimal point

Printing Variables: `printf`

```
printf(“Height: %d\n”, height);
```

```
printf(“%s: %.2f\n”, “Profit”, profit);
```

Initializing Variables

Variables may have random variables if not initialized.

Declaration and initialization can happen in one step:

```
int height = 8;
double profit = 1030.56;
float profit = 1030.56f;
char c = ‘A’;
char c = ‘\n’;
```

Reading Input: `scanf`

Reading an `int` value:

```
scanf("%d", &height);
```

Reading a `float` value:

```
scanf("%f", &a_float);
```

Reading a `double` value:

```
scanf("%lf", &a_double);
```

Reading a `char` value:

```
scanf("%c", &ch);
```

The `&` is very important. You'll learn later why.

Defining Names for Constants

Macro:

```
#define NAME <some text>
```

- Preprocessor replaces every occurrence of `NAME` with `<some text>`
- `NAME` is not a variable!
- No checks whether the replacement of `NAME` with `<some text>` results in valid code.
- `<some text>` can be any sequence of tokens.

Example:

```
#define PI 3.14159
```

Defining Expressions as Macros

The value of a macro can be an expression:

```
#define RECIPROCAL_OF_PI (1.0 / 3.14159)
```

Be generous with parentheses:

- What would happen without parentheses in this example?

```
double pi = 1.0 / RECIPROCAL_OF_PI;
```

Identifiers

= names for variables, functions, user-defined types, macros, etc.

- May contain letters, underscores, and digits
- Must start with a letter or underscore
- (For now, avoid using underscore as the first letter)

Conventions:

- Functions, variables, types: `transpose_matrix`, `vector`, ...
- Macros: `PI`, `NUM_ROWS`, ...

A Simple Example

Write a program to help your Walmart cashier:

Inputs:

- Price of product before HST
- Payment made by customer

Output:

- Change due to customer = payment - price * (1 + HST)

hst.c

```
#include <stdio.h>

#define HST 15

int main() {
  double price, payment, balance;

  printf(“Enter price: “);
  scanf(“%lf”, &price);

  printf(“Enter payment: “);
  scanf(“%lf”, &payment);

  balance = payment - prince * (1.0 + HST / 100);
  printf(“Change due: %.2lf\n“, balance);
  return 0;
}
```