A C Primer: Introduction

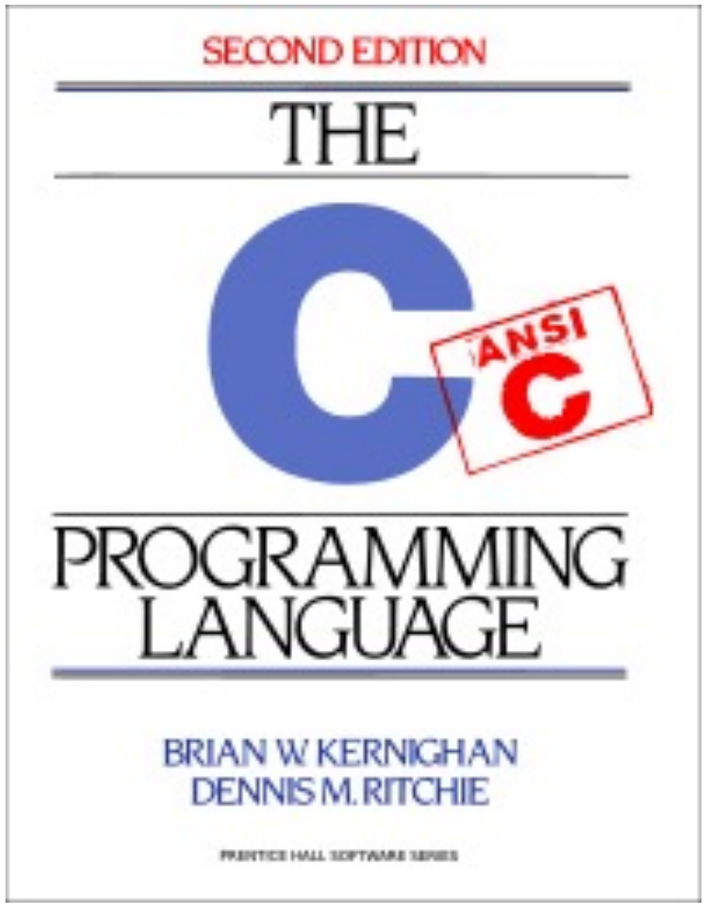
Ion Mandoiu
Laurent Michel
Revised by M. Khan, J. Shi, and W. Wei

Overview

- C in context, briefly
- C resources
- Getting started with examples

Very brief history

- Classic Book
  - Kernighan & Ritchie, The “K&R” C
- C is still evolving
  - Different standard versions
    - K&R C 1978, 1988
    - ANSI C 1989
      - C89, C90 (ISO 9899:1990)
    - C99 1999
    - C11 2011
    - C18 2018



Design principles

- C should be…
  - Simple
  - Easy to compile
  - Typed
  - Support low-level memory access
  - Ideal for embedded controller, OS, …
- Yet…
  - C is *powerful*
  - C is *fast*

Paradigm

- C is a purely **procedural** language
  - No object-orientation whatsoever
- Central Dogma
  - Object of interest: Computations
  - Main abstraction tool: Procedures/functions
    - Caller / Callee
    - Abstracts away “How things are done”
  - Programming means
    - Organizing processes as procedures
    - Composing processes through procedure calls

Procedural Programming in C

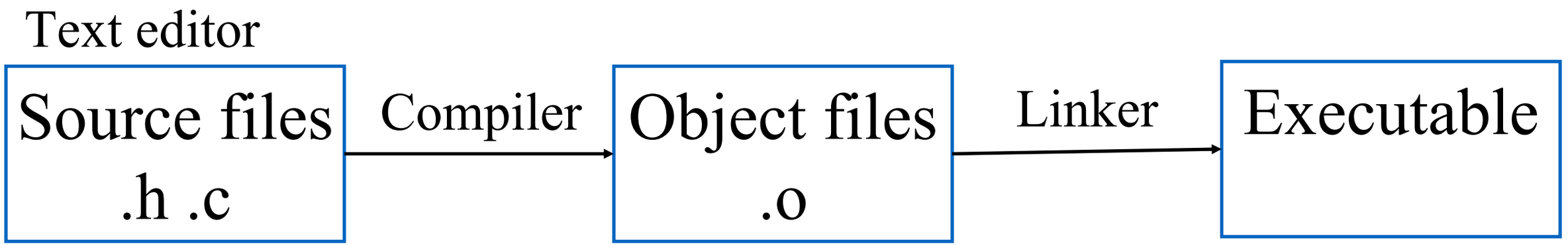
- Adheres to the philosophy
- Generates very efficient code
- Exposes as many low-level details you wish to see
- Provides full control over memory management (no Garbage Collection!)
- The Programmer is **fully in charge**

Resources

- What you do need….
  - A C compiler
  - A linker
  - A text editor
- We will use the GNU toolchain
  - Compiler: gcc (or cc)
  - Linker: ld (invoked by gcc)
  - Editor: vim, nano, emacs, …

Workflow

- Use text editor to edit source files
- Use compiler to generate .o files
- Use linker to link multiple .o files into executables



Hello world!

```c
#include <stdio.h>

/* comments */
// single line comments

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

Comments

- The compiler ignores everything between “/*” and “*/”
  - Comments are meant to be human readable!
  - Comments can be multi-line
  - Cannot be nested
- Single line comments (C99)
  - Everything starting from "//" is ignored in a line

```
/* my first program */

/**********************
 * multiple lines
 **********************/

// Single line
```

The 'main' function

- A *special* function that defines the entry point for the program.
  - This is where the Operating System transfers control once the program starts
  - void indicates no arguments
  - int before 'main' indicates the main function returns an integer

```c
#include <stdio.h>

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

printf

- What is ‘printf’ ?
  - A C library function to print on the standard output for the process
  - It takes at least a string as argument
    - Between double quotations, like "This is a string."
- ‘\n’ is a newline character

```c
#include <stdio.h>

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

Why do we ‘return’ anything from main() ?

- When the process terminates, it can tell the O.S. how things went.
  - This is the way to report back.
- Returning 0 means ‘everything went according to plans’

```c
#include <stdio.h>

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

Including header files

- What's up with #include ?

  - It "imports" a header file with the specification of functions that exist in a library to be linked with the program

printf

Compile

- What is *cc* doing really ?
  - Three steps
    - preprocesses hello.c
    - compiles hello.c to hello.o
    - links hello.o with libc
    - writes executable file a.out
  - You **can** and **often will** separate those steps!

```c
#include <stdio.h>

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

Terminal

```
$ cc hello.c
$ ./a.out
```

Execute

- The name of the executable can be changed
  - a.out is the default
- ./ indicates a.out in the current directory
  - Otherwise the OS searches directories in 'PATH'

Terminal

```
$ cc hello.c  -o hello
$ ./hello
```

A second program

- Purpose
  - Read an integer from the standard input: **n**
  - Compute the sum of all integers between 1 and **n**
  - Print the result on the standard output
- New concepts
  - Standard input and output

A second program

```
/*
  Compute the sum of the integers
  from 1 to n, for a given n
*/
#include <stdio.h>

int main(void) {
   int i, n, sum;
   sum = 0;
   printf("Enter n:\n");
   scanf("%d", &n);
   i = 1;
   while (i <= n) {
      sum = sum + i;
      i = i + 1;
   }
   printf("Sum from 1 to %d = %d\n", n, sum);
   return 0;
}
```

A second program

```
/*
  Compute the sum of the integers
  from 1 to n, for a given n
*/
#include <stdio.h>

int main(void) {
   int i, n, sum;
   sum = 0;
   printf("Enter n:\n");
   scanf("%d", &n);
   i = 1;
   while (i <= n) {
      sum = sum + i;
      i = i + 1;
   }
   printf("Sum from 1 to %d = %d\n", n, sum);
   return 0;
}
```

Tells compiler that the function takes no arguments

A second program

```
/*
  Compute the sum of the integers
  from 1 to n, for a given n
*/
#include <stdio.h>

int main(void) {
  int i, n, sum;
  sum = 0;
  printf("Enter n:\n");
  scanf("%d", &n);
  i = 1;
  while (i <= n) {
     sum = sum + i;
     i = i + 1;
  }
  printf("Sum from 1 to %d = %d\n", n, sum);
  return 0;
}
```

Local variable declarations

A second program

```
/*
  Compute the sum of the integers
  from 1 to n, for a given n
*/
#include <stdio.h>

int main(void) {
   int i, n, sum;
   sum = 0;
   printf("Enter n:\n");
   scanf("%d", &n);
   i = 1;
   while (i <= n) {
      sum = sum + i;
      i = i + 1;
   }
   printf("Sum from 1 to %d = %d\n", n, sum);
   return 0;
}
```

Reads integer (%d) from input and stores it in “n”

A second program

```
/*
  Compute the sum of the integers
  from 1 to n, for a given n
*/
#include <stdio.h>

int main(void) {
   int i, n, sum;
   sum = 0;
   printf("Enter n:\n");
   scanf("%d", &n);
   i = 1;
   while (i <= n) {
      sum = sum + i;
      i = i + 1;
   }
   printf("Sum from 1 to %d = %d\n", n, sum);
   return 0;
}
```

Assignment expressions

A second program

```
/*
  Compute the sum of the integers
  from 1 to n, for a given n
*/
#include <stdio.h>

int main(void) {
   int i, n, sum;
   sum = 0;
   printf("Enter n:\n");
   scanf("%d", &n);
   i = 1;
   while (i <= n) {
      sum = sum + i;
      i = i + 1;
   }
   printf("Sum from 1 to %d = %d\n", n, sum);
   return 0;
}
```

↔ sum += i;

A second program

```
/*
  Compute the sum of the integers
  from 1 to n, for a given n
*/
#include <stdio.h>

int main(void) {
   int i, n, sum;
   sum = 0;
   printf("Enter n:\n");
   scanf("%d", &n);
   i = 1;
   while (i <= n) {
      sum = sum + i;
      i = i + 1;
   }
   printf("Sum from 1 to %d = %d\n", n, sum);
   return 0;
}
```

↔ i += 1; ↔ i++;

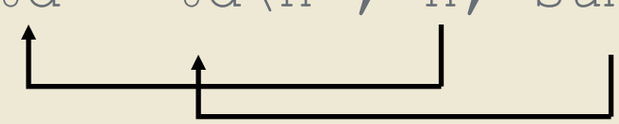
A second program

```
/*
  Compute the sum of the integers
  from 1 to n, for a given n
*/
#include <stdio.h>

int main(void) {
  int i, n, sum;
  sum = 0;
  printf("Enter n:\n");
  scanf("%d", &n);
  i = 1;
  while (i <= n) {
    sum = sum + i;
    i = i + 1;
  }
  printf("Sum from 1 to %d = %d\n", n, sum);
  return 0;
}
```

Loop

A second program

```
/*
  Compute the sum of the integers
  from 1 to n, for a given n
*/
#include <stdio.h>

int main(void) {
   int i, n, sum;
   sum = 0;
   printf("Enter n:\n");
   scanf("%d", &n);
   i = 1;
   while (i <= n) {
      sum = sum + i;
      i = i + 1;
   }
   printf("Sum from 1 to %d = %d\n", n, sum);
   return 0;
}
```



Terminal

```
$ cc sum.c
$ ./a.out
Enter n:
100
Sum from 1 to 100 = 5050
$
```

Summary

- A C program consists of functions
  - main() is the entrance of a program
- A function consists of a sequence of statements
- Variables can be declared in a function (like main)
  - These are local variables
  - Variables must be declared
- Statements are terminated with ';'
  - Empty statements are allowed

```
;;; // three empty statements
```

- Include proper header files for library functions

- Study the remaining slides yourself

The 'main' function taking arguments

- The 'main' function can take two arguments:
  - argc: the number of arguments
  - argv: an array of arguments

  Will learn how to use these arguments later.

```c
#include <stdio.h>

int main(int argc,char* argv[])
{
   printf("Hello world!\n");
   return 0;
}
```

- Some systems support three arguments (environment variables)

If-else statements

```
// Will learn more about if-else statements in coming weeks.
// Simple if statemetents are similar to ones in Python
```

```
# Python
if a >= 0:
    b = c + a
    d = d + 5
else:
    b = c - a
    d = d + 10
```

```
// C
if (a >= 0) {
    b = c + a;
    d = d + 5;
}
else {
    b = c - a;
    d = d + 10;
}
```

Parentheses around the condition

Use { and } to enclose multiple statements

A statement ends with ;

else branch