NWEN 241
C Fundamentals

Winston Seah

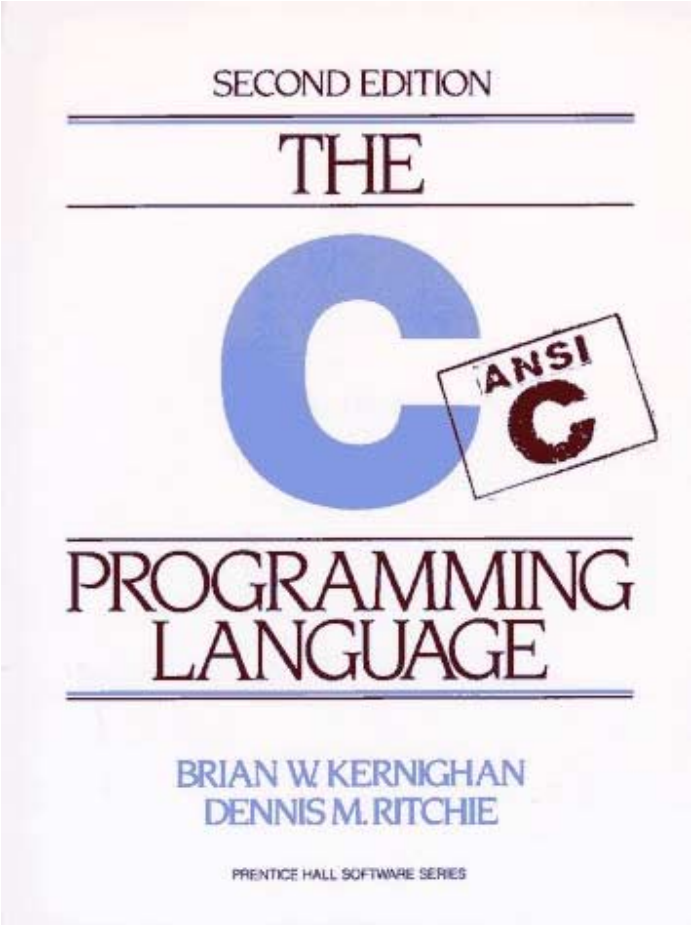
School of Engineering and Computer Science
Victoria University of Wellington

This Lecture

- Background about C
- Development environment & C program structures

Background and Characteristics

- Designed by Dennis Ritchie of Bell Labs in 1970s
- An outgrowth of B also developed at Bell Labs
- ANSI/ISO standard in early 1990s.
- Bridging the gap between machine language and high-level languages
  - Low-level features: fast/efficient (systems programming)
  - High-level features: structured programming (applications programming)



Comparing C, C++, Java

- The C Family of Languages: Interview with Dennis Ritchie, Bjarne Stroustrup, and James Gosling:
  - http://www.gotw.ca/publications/c_family_interview.htm

Comparing C, C++ and Java

- C is the basis for C++ and Java
  - C evolved into C++
  - C++ transmuted into Java
  - The “class” is an extension of “struct” in C
- Similarities
  - Java uses a syntax similar to C++ (for, while, …)
  - Java supports OOP as C++ does (class, inheritance, …)
- Differences
  - Java does not support pointer
  - Java frees memory by garbage collection
  - Java is more portable by using bytecode and virtual machine
  - Java does not support operator overloading
  - ….

Applications

- Operating systems
- Distributed systems
- Network programming
- Database applications
- Real-time and engineering applications
- Any application where performance is *paramount*

Development Environment

- Lab: CO246
- ID access cards (Swipe Cards): should work if you are registered in NWEN 241

- PC Unix workstations, Linux, KDE
- Network file system
- Tools: gcc, g++, gdb, eclipse, emacs, gedit, vi, vim
- Text editor vs IDE: text editor recommended

- Remote access:
  https://ecs.victoria.ac.nz/Support/TechNoteWorkingFromHome

Program Structure

- A C program consists of one or more *functions*
- A C program must have a `main` function

```
int main(void)
{ ...;
  return 0;
}
```

- Execution begins with the `main` function
- Java vs. C
  - C uses stand-alone functions
  - No stand-alone functions in Java
  - No global functions in Java

Program Structure

- Each function must contain:
  - A function *heading*, return type, function name, (an *optional* list of *arguments*)
  - A list of argument *declarations*, if arguments are included in heading
  - A *compound statement*

```
int function_name(int x, int y)
{
   ...
}
```

Program Structure

- An example (single function)

```
1.   /* A simple program */          /* comment */
2.
3.   #include <stdio.h>              /* library file access */
4.
5.   int main(void)                 /* function heading  */
6.   {
7.      printf("Hello world\n");     /* output statement */
8.
9.      return 0;                   /* return statement */
10.  }
```

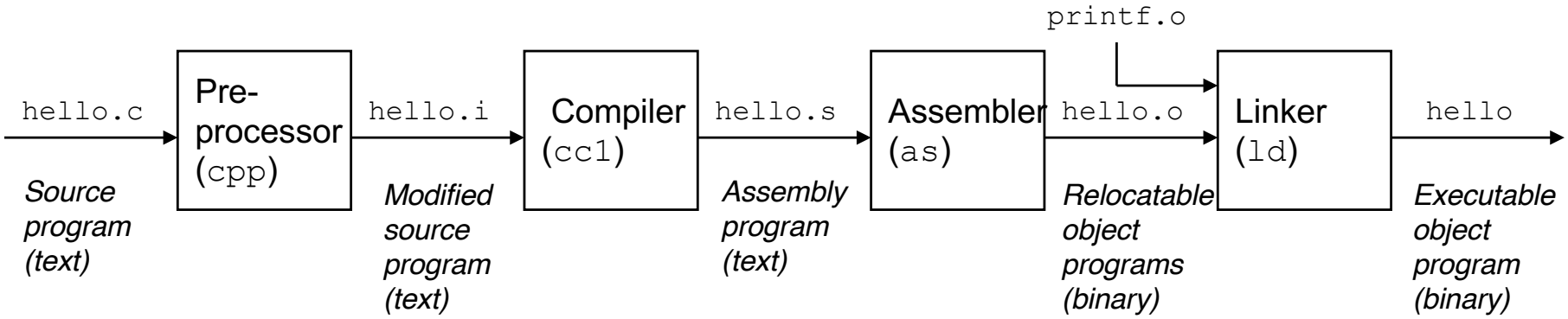
Writing a program & Compilation

```
# vi hello.c
```

…the screen will clear and you enter the vi text editor

```
# gcc hello.c
# ./a.out
Hello world
# gcc hello.c -o hello
# ./hello
Hello world
#
```

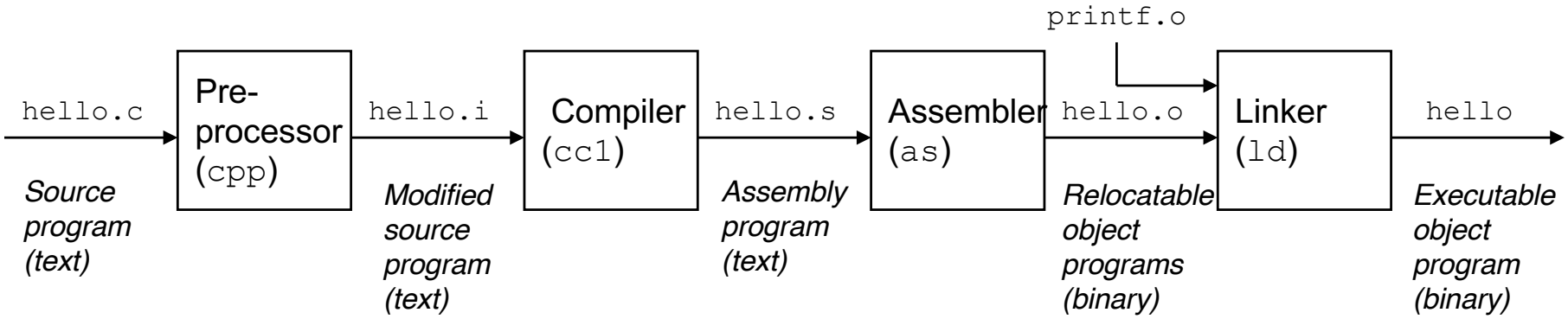
Compilation process



Preprocessing phase.

- The preprocessor (**cpp**) modifies the original C program according to directives that begin with the '#' character, e.g., #include <stdio.h> command in line 3 of **hello.c** tells the preprocessor to read the contents of the system header file **stdio.h** and insert it directly into the program text. The result is another C program, typically with the **.i** suffix.

Compilation process

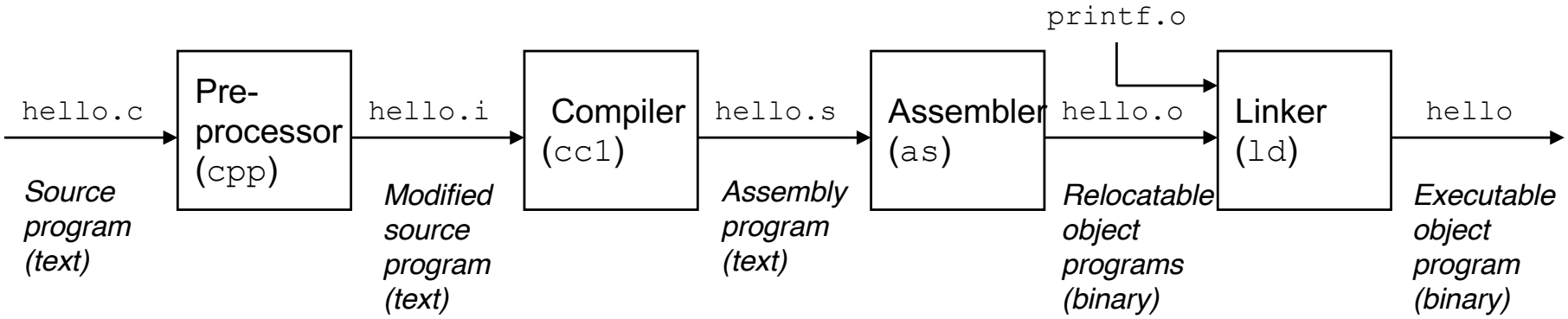


Compilation phase.

- The compiler (**cc1**) translates the text file **hello.i** into the text file **hello.s**, which contains an assembly-language program.

```
1    main:
2        subq    $8, %rsp
3        movl    $.LC0, %edi
4        call    puts
5        movl    $0, %eax
6        addq    $8, %rsp
7        ret
```

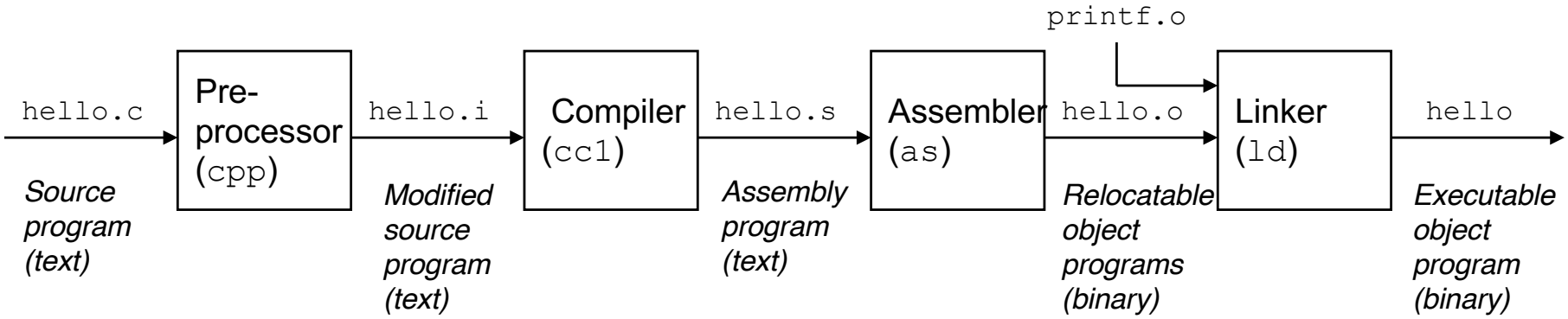
Compilation process



Assembly phase.

- The assembler (**as**) translates **hello.s** into machine-language instructions, packages them in a form known as a *relocatable object program*, and stores the result in the object file **hello.o**.
- This file is a binary file containing 17 bytes to encode the instructions for function main.
- If you try to open **hello.o** with a text editor, it would appear to be gibberish.

Compilation process



Linking phase.

- **printf** function, which is part of the standard C library provided by every C compiier.
- **printf** function resides in a separate precompiled object file called printf.o, which must be merged with our **hello.o** program.
- The linker (**ld**) performs this merging, creating an executable object file (or simply *executable*) that is ready to be loaded into memory and executed by the system.

Program Structure

- An example (single function)

```
/* Program to calculate the area of a circle */   /* comment */

#include <stdio.h>                    /* library file access */
#define PI 3.14        /* macro definition - symbolic constant */
#define SQ(x) ((x)*(x))               /* macro with arguments */

int main(void)                        /* function heading  */
{
   float radius, area;                /* variable declarations */

   printf("Radius = ");               /* output statement (prompt)*/
   scanf("%f", &radius);              /* input statement */

   area = 3.14 * radius * radius; /* assignment statement */
   printf("Area1 = %f\n", area);  /* output statement */

   area = PI * SQ(radius);            /* use macros */
   printf("Area2 = %f\n", area);  /* output statement */

   return 0;                          /* return statement */
}
```

Program Structure

- Another example (multiple functions)

```
/* Program to calculate the area of a circle */

#include <stdio.h>                      /* library file access */
#define PI 3.1415926            /* macro definition - symbolic constant */

float sq(float);                /* square function - function prototype */

int main(void)                          /* function heading  */
{
   float radius, area;                  /* variable declarations */

   printf("Radius = ");                 /* output statement (prompt)*/
   scanf("%f", &radius);                /* input statement */

   area = PI * sq(radius);              /* use square function */
   printf("Area = %f\n", area);         /* output statement */
   return 0;                            /* return statement */
}

float sq(float r)
{  return (r * r);}             /* square function - function definition*/
```

Summary

- C / C++ / Java
- C program structure

Next Lecture

- More on C fundamentals