Week 1 Lecture 2

XMUT-NWEN 241 - 2024 T2

Systems Programming

Mohammad Nekooei

School of Engineering and Computer Science

Victoria University of Wellington

Admin

- Exercise 0 is out.
- Exercise 1 will be out on Sep 9.

Content

Systems Programming

C Program Design

C Compilation Process

What is Systems Programming?

Systems Programming

- Systems programming refers to the implementation of **systems programs** or **software**

- Systems program / software:
  - Programs that support the **operation** and **use** of the computer system itself
  - Maybe used to support other software and application programs
  - May contain **low-level** or **architecture-dependent** code

- Low-level or architecture-dependent code:
  - Program that directly accesses registers or memory locations
  - Program that uses instructions specific to a computer architecture

Example Systems Programs

- Operating system
- Embedded system software (firmware)
- Device drivers
- Text editors, compilers, assemblers
- Virtual machines
- Server programs
  - Database systems
  - Network protocols

Why C?

- C supports **high-level** abstractions and **low-level** access to hardware at the same time
- **High-level abstractions:**
  - User-defined types (structures)
  - Data structures (stacks, queues, lists)
  - Functions
- **Low-level access to hardware:**
  - Possible access to registers
  - Dynamic memory allocation
  - Inclusion of assembly code

Comparing C, C++ and Java

- C is the basis for C++ and Java
  - C evolved into C++
  - C++ change 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

C Program Design

Program Structure

- A typical C program consists of
  - 1 or more **header files**
  - 1 or more C **source files**

```
/* Simple Program
 * structure
*/
```

Block Comment or multi-line comment

```
#include <stdio.h>
```

Preprocessor directive to include `stdio.h` header file which contains `printf` function *prototype*

```
int main(void)
{
   printf("Hello world\n");

   return 0;
}
```

`main` function *definition*, invoking `printf` to display "Hello, world", and return 0

Comments

```
/* Comment text */
```

- Compiler ignores everything from /* to */

```
// Comment text
```

- Compiler ignores everything from // to the end of the line
- This commenting style originated from C++ and was adopted by C (C99 standard)

Main Function

- A C program must have exactly one `main` function
- Execution begins with the `main` function

```
int main(void)
{
    /* Main function body */
}
```

Header File Inclusion

```
#include <filename>
```

- Include file named `filename`
- Preprocessor searches for file in pre-defined locations

```
#include "filename"
```

- Include file named `filename`
- Preprocessor searches for file in current directory first, then in locations specified by programmer

Header Files

- A header file usually contains function prototypes, constant definitions, type definitions, etc.

- Which header file to include?
  - Include header files that contain the function prototype, constant definition, type definition, etc., used in your program

Standard C Library Header Files

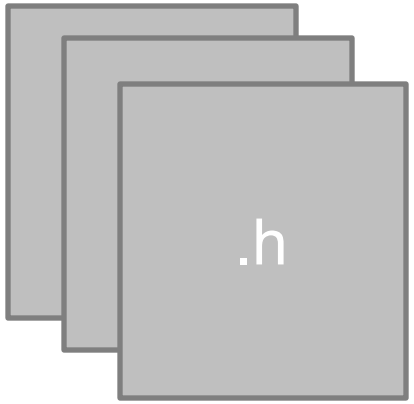
C provides a standard library* which consists of the following headers:

assert.h	float.h	math.h	stdarg.h	stdlib.h
ctype.h	limits.h	setjmp.h	stddef.h	string.h
errno.h	locale.h	signal.h	stdio.h	time.h

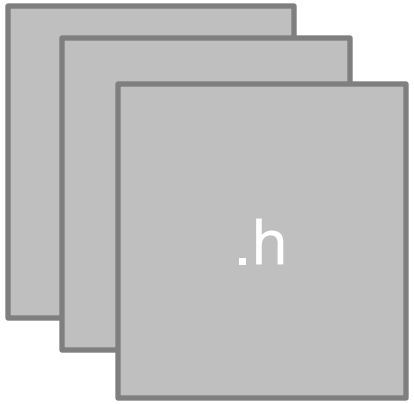
- To know more about the C standard library, visit
  https://www.tutorialspoint.com/c_standard_library/index.htm

*C99 and C11 standards added more header files.

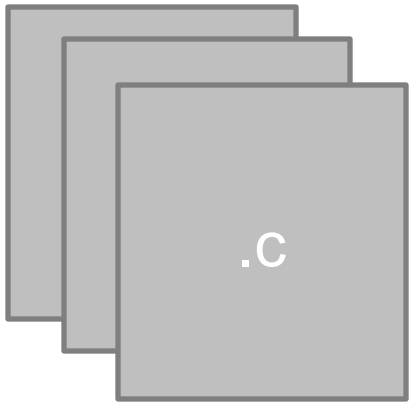
Large C Program



Header files from standard C library



Own header files



Source files

C Compilation Process

Compilation Process At A Glance

1) Preprocessing Phase

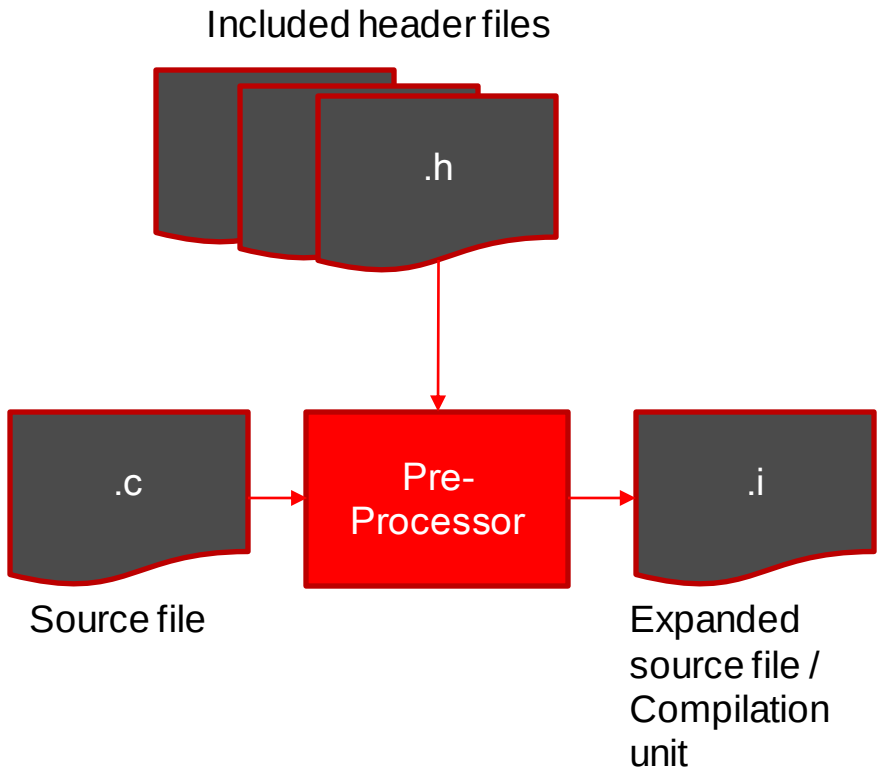
2) Compilation Phase

3) Assembly Phase

4) Linking Phase

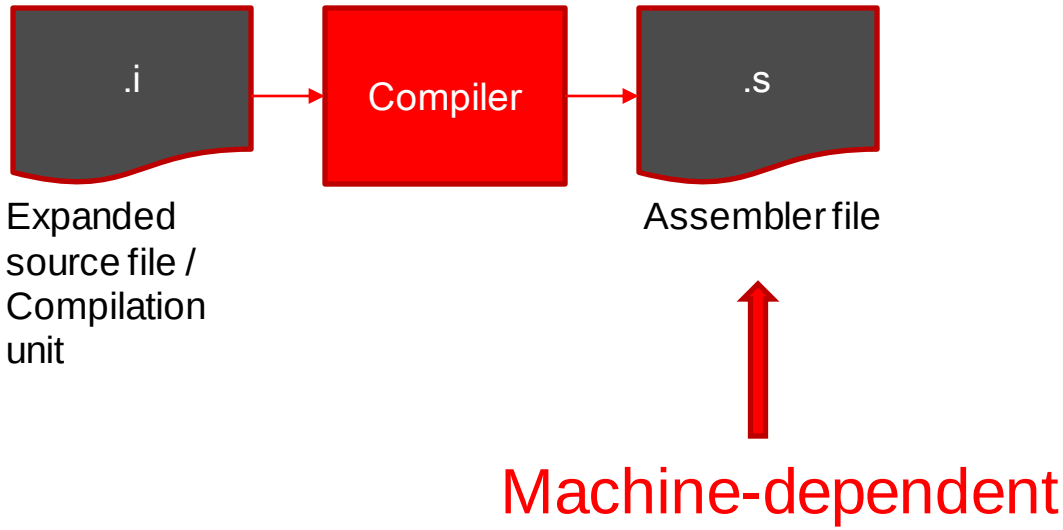
Preprocessing Phase

- The **preprocessor** modifies the original C program according to directives that begin with the '#' character
  - Example: #include <stdio.h> command 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 Phase

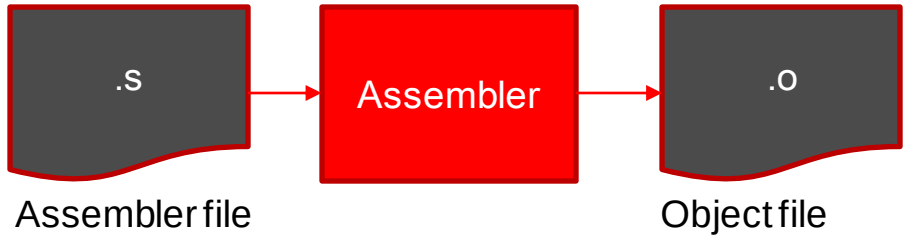
- The **compiler** translates the text file (**.i**) into the text file (**.s**), which contains an assembly-language program.



Assembly Phase

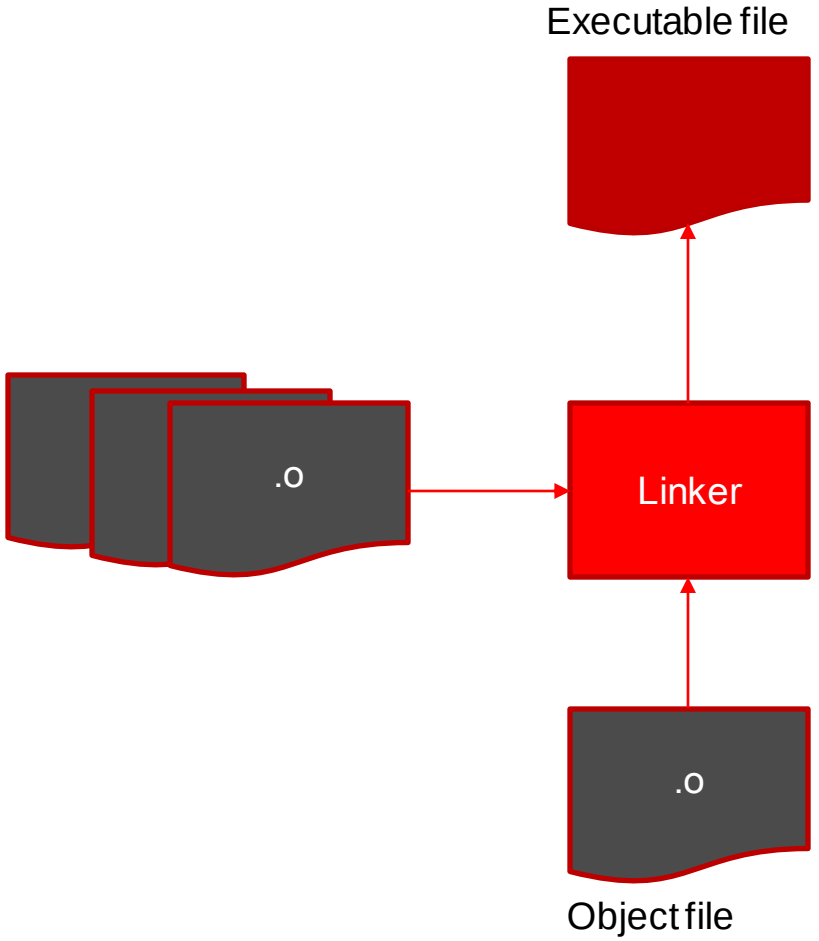
- The **assembler** translates assembler file (**.s**) into machine-language instructions, packages them in a form known as a *relocatable object program*, and stores the result in the *object file* (**.o**).

- Object files are binary - if you try to open one with a text editor, it would appear to be gibberish.



Linking Phase

- The **linker** looks for external object files needed by the program and merges these with the object file generated in the assembly phase, creating an executable object file (or simply *executable*) that is ready to be loaded into memory and executed by the system.



In Practice- GCC compiler

- All the phases can be done in one step using the GNU C Compiler (GCC)
- Usually, we use gcc to do all the phases and directly generate the binary executable
- We can also ask gcc to do certain phases

`hello.c`

```
#include <stdio.h>
int main(void)
{
   printf("Hello world\n");

   return 0;
}
```

```
gcc hello.c
```

Generates executable file **`a.out`**

```
gcc hello.c -o hello
```

Generates executable file **`hello`**

gcc Options

Phase	Gcc Option	Result	Output File
Preprocessing	-E	Compilation unit	.i .ii
Compilation	-S	Assembler file	.s
Assembly	-c	Object file	.o .obj
Linking		Executable	Binary excutable (.exe in Windows)

What You Need to Program in C

- **Text editor** to type in code
  - Any text editor will do (even notepad)
  - Suggested editors: Sublime Text, Atom, Kate (Linux only)

- **C toolchain** (pre-processor, compiler, assembler, linker, debugger)

- **Terminal** to run compilation commands and execute program

OR

- **IDE**