Week 1 Lecture 2

NWEN 241
Systems Programming

Jyoti Sahni

Jyoti.sahni@ecs.vuw.ac.nz

Admin Stuff: Exercise 1

- Exercise 1 is out

- Please visit course wiki (https://ecs.wgtn.ac.nz/Courses/NWEN241_2024T1/Exercises) for more details
  - For quick help, e-mail nwen241-staff@ecs.vuw.ac.nz
  - Otherwise, attend any of the Helpdesk sessions

- Submit your answers to Exercise 1 on or before 06 Mar 2024 23:59

Content

- Systems Programming
- C Program Design
- C Compilation Process
- C Fundamentals
  - Identifiers and reserved keywords
  - Data types, constants and literals
  - Operators and expressions

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++?

- C/C++ supports **high-level** abstractions and **low-level** access to hardware at the same time
- **High-level abstractions:**
  - User-defined types (structures and classes)
  - 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++ 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

C Program Design

Program Structure

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

```
#include <stdio.h>

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

   return 0;
}
```

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

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

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 the pre-defined locations as specified by the implementation

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

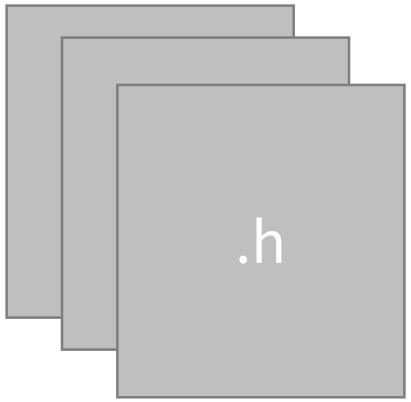
C provides a standard library* which consists of the following headers:

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

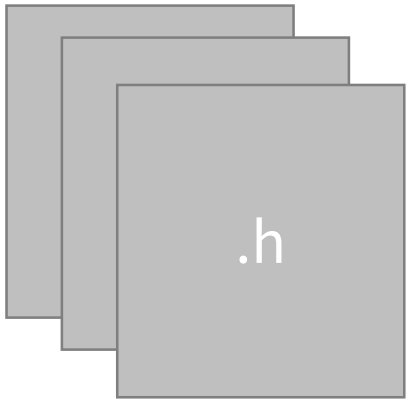
- To know more about the C standard library, visit
  https://www.tutorialspoint.com/c_standard_library/index.htm

*C99 and C11 standards added more header files.

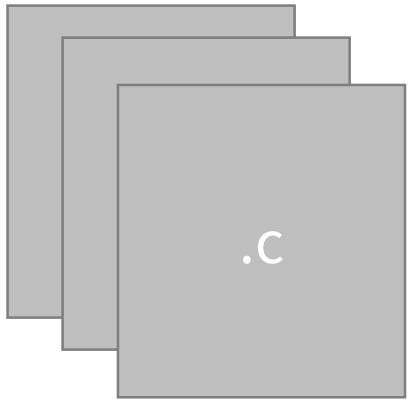
Large C Program



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

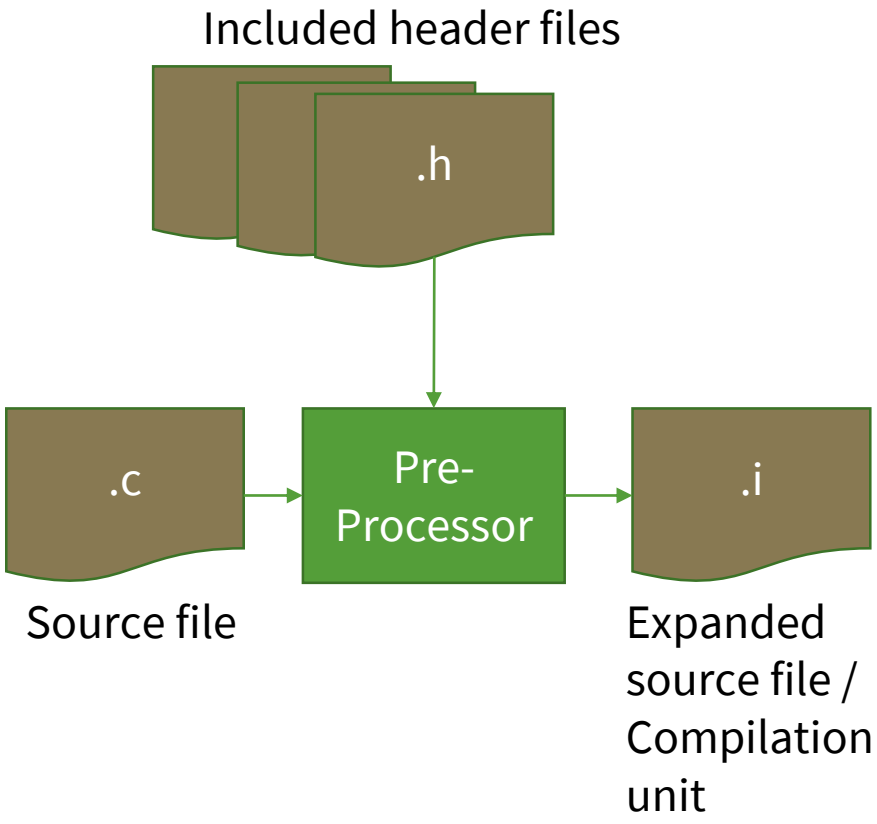
3) Assembly Phase

4) Linking Phase

Preprocessing Phase

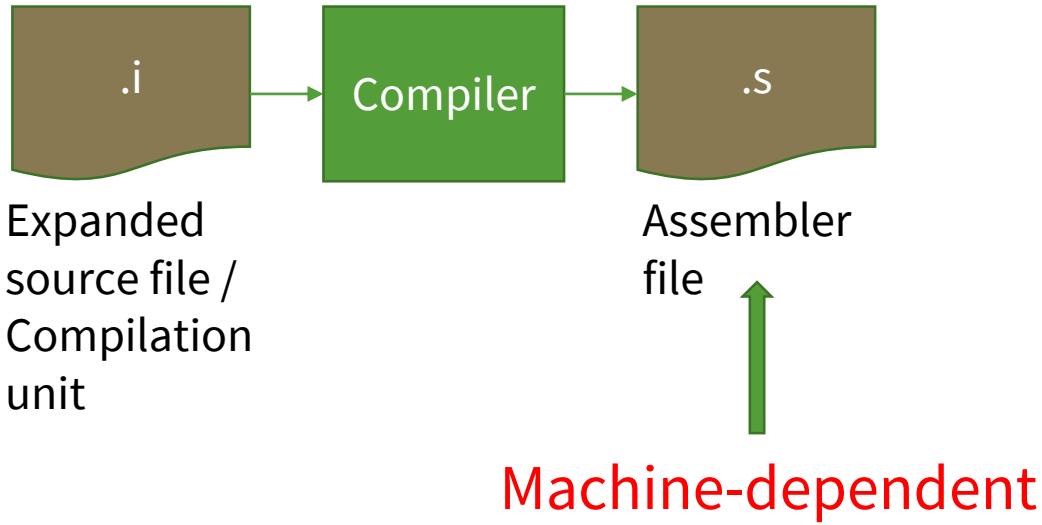
- The **preprocessor** modifies the original C program according to directives that begin with the '#' character
  - Example: #include <stdio.h> command tells the preprocessor to read the contents of the system header file **stdio.h** and insert it directly into the program text.

- The result is another C program, typically with the **.i** suffix.



Compilation Phase

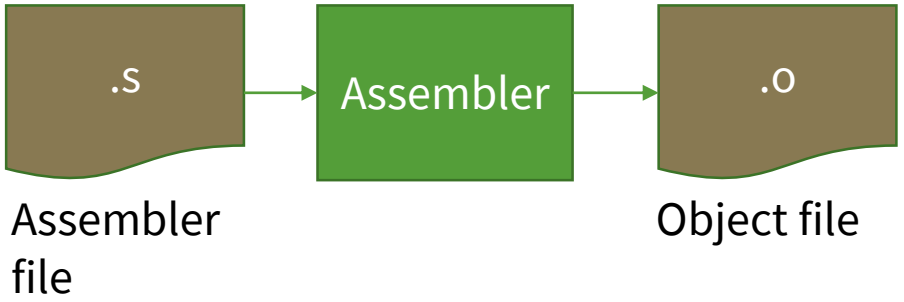
- The **compiler** translates the text file (**.i**) into the text file (**.s**), which contains an assembly-language program.



Assembly Phase

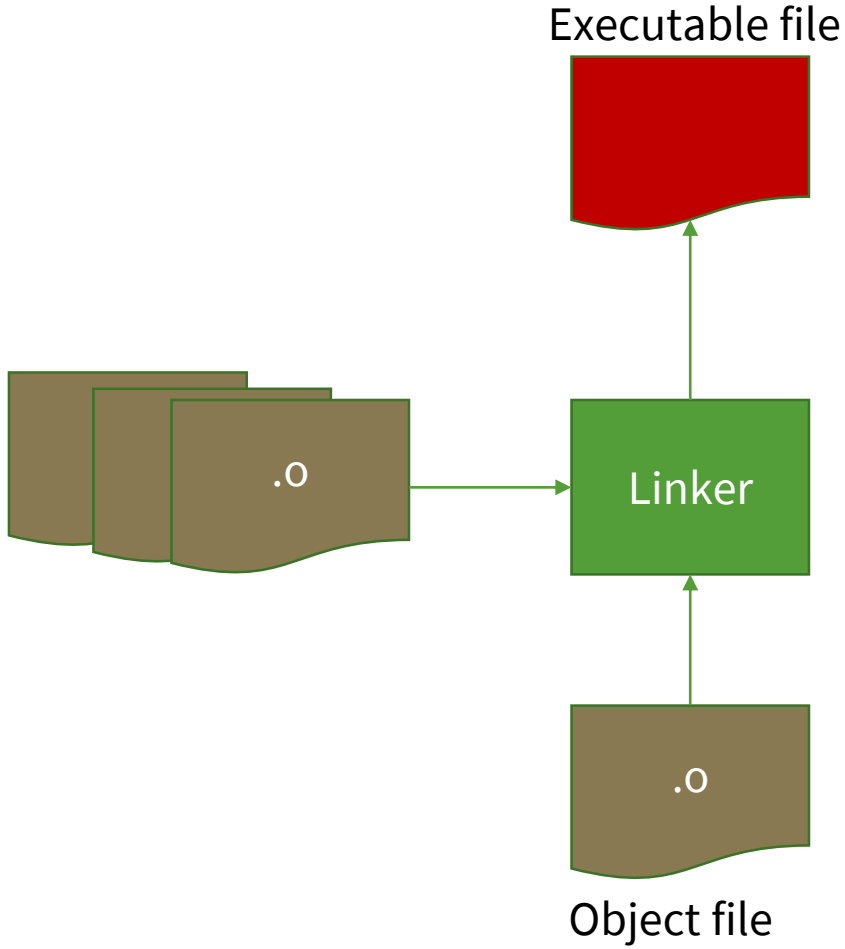
- The **assembler** translates assembler file (**.s**) into machine-language instructions, packages them in a form known as a *relocatable object program*, and stores the result in the *object file* (**.o**).

- 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

- All the phases can be done in one step using the GNU C Compiler (gcc)

`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`**

What You Need to Program in C/C++

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

- **C/C++ toolchain** (pre-processor, compiler, assembler, linker, debugger)
  - Already installed in ECS lab computers (CO246) and servers

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

C Fundamentals

Identifiers

- Identifier is used to name **macros**, variables, **functions**, **structs**, **unions**, and other entities in a computer program

- Java and C have similar rules for identifiers, except:
  - In C, $ is not allowed in identifiers (though some compilers allow $)

Rules on Identifiers

- An identifier is a sequence of letters and digits
  - The first character must be a letter
    - The underscore character _ counts as a letter
    - Upper and lower case letters are different
- Identifiers may have any length
  - Usually, only the first 31 characters are significant
  - For macro names, only the first 63 characters are significant
- Reserved keywords cannot be used as identifiers!

Examples

`counter`	Valid: consists of letters
`_Temp_variable_2`	Valid: consists of letters and digits
`1myVariable`	Invalid: first character is not a letter
`continue`	Invalid: reserved word

Reserved Keywords

```
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
```

Data Types

- Recall: Java has 8 basic data types which have fixed sizes

Data Type	Size (bytes)
`boolean`	1
`byte`	1
`char`	2
`short`	2
`int`	4
`long`	8
`float`	4
`double`	8

Data Types

- C data types:

Data Type	Size (bytes)
~~boolean~~	~~1~~
~~byte~~	~~1~~
char	~~2~~ 1
short (short int)	~~2~~ Machine-dependent
int	~~4~~ Machine-dependent
long (long int)	~~8~~ Machine-dependent
long long (long long int)	Machine-dependent
float	~~4~~ Machine-dependent
double	~~8~~ Machine-dependent
long double	Machine-dependent

Integral types

Float types

Data Type Size

- Sizes of different types
  - Use `sizeof()` to find out
  - Some of the types size may vary from machine to machine

- The following rules are always guaranteed:
  - `sizeof(char) == 1`
  - `sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long) <= sizeof(long long)`
  - `sizeof(float) <= sizeof(double) <= sizeof(long double)`

Data Types

- Integral types can either be `signed` or `unsigned`

```
signed int var1;    // Signed integer

unsigned int var2;  // Unsigned integer

int var1;  // If signed or unsigned is not present, default is signed
```

char Data Type

- `unsigned char`: 0 to 255; `signed char`: -128 to 127
- `char` is meant to hold **1 ASCII character**

0	NUL	1	SOH	2	STX	3	ETX	4	EOT	5	ENQ	6	ACK	7	BEL
8	BS	9	HT	10	NL	11	VT	12	NP	13	CR	14	SO	15	SI
16	DLE	17	DC1	18	DC2	19	DC3	20	DC4	21	NAK	22	SYN	23	ETB
24	CAN	25	EM	26	SUB	27	ESC	28	FS	29	GS	30	RS	31	US
32	SP	33	!	34	"	35	#	36	$	37	%	38	&	39	'
40	(	41	)	42	*	43	+	44	,	45	-	46	.	47	/
48	0	49	1	50	2	51	3	52	4	53	5	54	6	55	7
56	8	57	9	58	:	59	;	60	<	61	=	62	>	63	?
64	@	65	A	66	B	67	C	68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K	76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S	84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[	92	\	93	]	94	^	95	_
96	`	97	a	98	b	99	c	100	d	101	e	102	f	103	g
104	h	105	i	106	j	107	k	108	l	109	m	110	n	111	o
112	p	113	q	114	r	115	s	116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	{	124	|	125	}	126	~	127	DEL

Variable Declaration

- Similar syntax as Java
- A variable must be declared before it can be used
- A variable may be initialized in its declaration
  - If variable name is followed by an equals sign and an expression, the latter serves as an *initializer*

```
int i = 0, j = 1, k = 2;
char c = 'A';
float f = 1.25;
```

- Possible initializers
  - Constant
  - Expression

Constants and Literals

- Constants are **fixed values** that cannot be changed during a program's execution
- The fixed values are called **literals**
- Literals
  - Integer
  - Floating Point
  - Character
  - *String*
  - *Enumeration*

Declaring Constants

- Constants can be declared using `const` qualifier or `#define` pre-processor
- Such named constants are also called **symbolic constants**

```
const float PI = 3.14;
const int MAX = 12345;
```

```
#define PI 3.14
#define MAX 12345
```

Next Lecture

- Literals (continued)
- Operators and expressions
- Functions
- Function-Like Macros