Computer Programming I

Ming-Feng Tsai (Victor Tsai)

Dept. of Computer Science
National Chengchi University

Introduction to Computers

Objectives

- In this chapter, you'll learn:
  - Basic computer concepts
  - The different types of programming languages
  - The purpose of the C Standard Library
  - The element of a typical C program development environment

Introduction

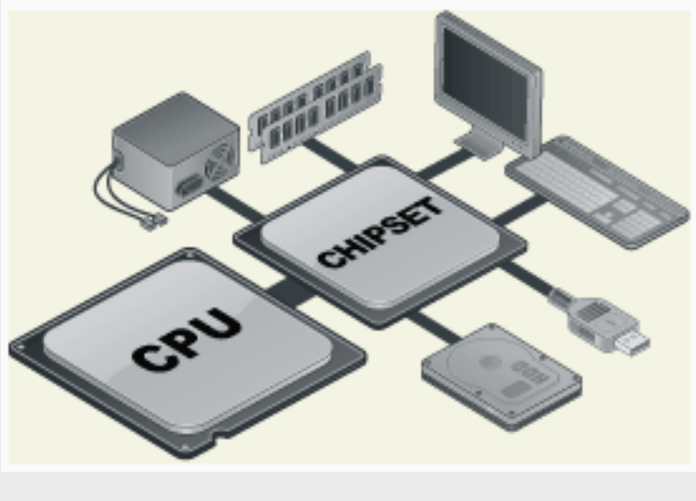
- Introduces programming in C, which was standardized in 1989 as ANSI X3.159-1989 in the United States through the American National Standards Institute (ANSI), then worldwide through the efforts of the International Standards Organization (ISO).
- We call this Standard C.
- We also introduce C99 (ISO/IEC 9899:1999)—the latest version of the C standard.

Computers: Hardware and Software

- A computer is a device that can perform computations and make logical decisions billions of times faster than human beings can.
- Today's fastest supercomputers can perform thousands of trillions (quadrillions) of instructions per second!
- Computers process data under the control of sets of instructions called computer programs
- The programs that run on a computer are referred to as software.

Computer Organization

- Every computer may be envisioned as divided into six logical units or sections:
  - Input unit
  - Output unit
  - Memory unit
  - Arithmetic and logic unit (ALU)
    - This “ manufacturing” section performs calculations, such as addition, subtraction, multiplication and division.
  - Central processing unit (CPU)
    - This “ administrative” section coordinates and supervises the operation of the other sections.
  - Secondary storage unit



Personal, Distributed and Client/ Server Computing

- In 1977, Apple Computer popularized personal computing.
- In 1981, IBM, the world's largest computer vendor, introduced the IBM Personal Computer (PC).
- Machines could be linked together in computer networks
  - Distributed computing.
  - Client/server computing
- C is widely used for writing software for operating systems, for computer networking and for distributed client/server applications.

Machine Languages, Assembly Languages and High-Level Languages

- Programmers write instructions in various programming languages, some directly understandable by computers and others requiring intermediate translation steps.
- Computer languages may be divided into three general types:
  - Machine languages
  - Assembly languages
  - High-level languages
- Any computer can directly understand only its own machine language
- Machine languages are machine dependent

Machine Languages, Assembly Languages and High-Level Languages

- To speed the programming process, high-level languages were developed in which single statements could be written to accomplish substantial tasks.
- Translator programs called compilers convert high-level language programs into machine language.

C Standard Library

- As you'll learn in Chapter 5, C programs consist of modules or pieces called functions.
- You can program all the functions you need to form a C program, but most C programmers take advantage of a rich collection of existing functions called the C Standard Library.

C Standard Library

- Avoid reinventing the wheel.
- Instead, use existing pieces—this is called software reusability.
- When programming in C you'll typically use the following building blocks:
  - C Standard Library functions
  - Functions you create yourself
  - Functions other people have created and made available to you

Performance Tips

Performance Tip 1.1

Using Standard C library functions instead of writing your own comparable versions can improve program performance, because these functions are carefully written to perform efficiently.

Performance Tip 1.2

Using Standard C library functions instead of writing your own comparable versions can improve program portability, because these functions are used in virtually all Standard C implementations.

Typical C Program Development Environment

- C systems generally consist of several parts: a program development environment, the language and the C Standard Library.
- C programs typically go through six phases
  - edit, preprocess, compile, link, load and execute.
- Phase 1 consists of editing a file.
  - This is accomplished with an editor program.
  - C program file names should end with the .c extension.

Typical C Program Development Environment (Cont.)

- Phase 2, the you give the command to compile the program.
  - The compiler translates the C program into machine language-code
    - In a C system, a preprocessor program executes automatically before the compiler's translation phase begins.
    - The C preprocessor obeys special commands called preprocessor directives
- Phase 3, the compiler translates the C program into machine-language code

Typical C development environment (Part 1 of 2)

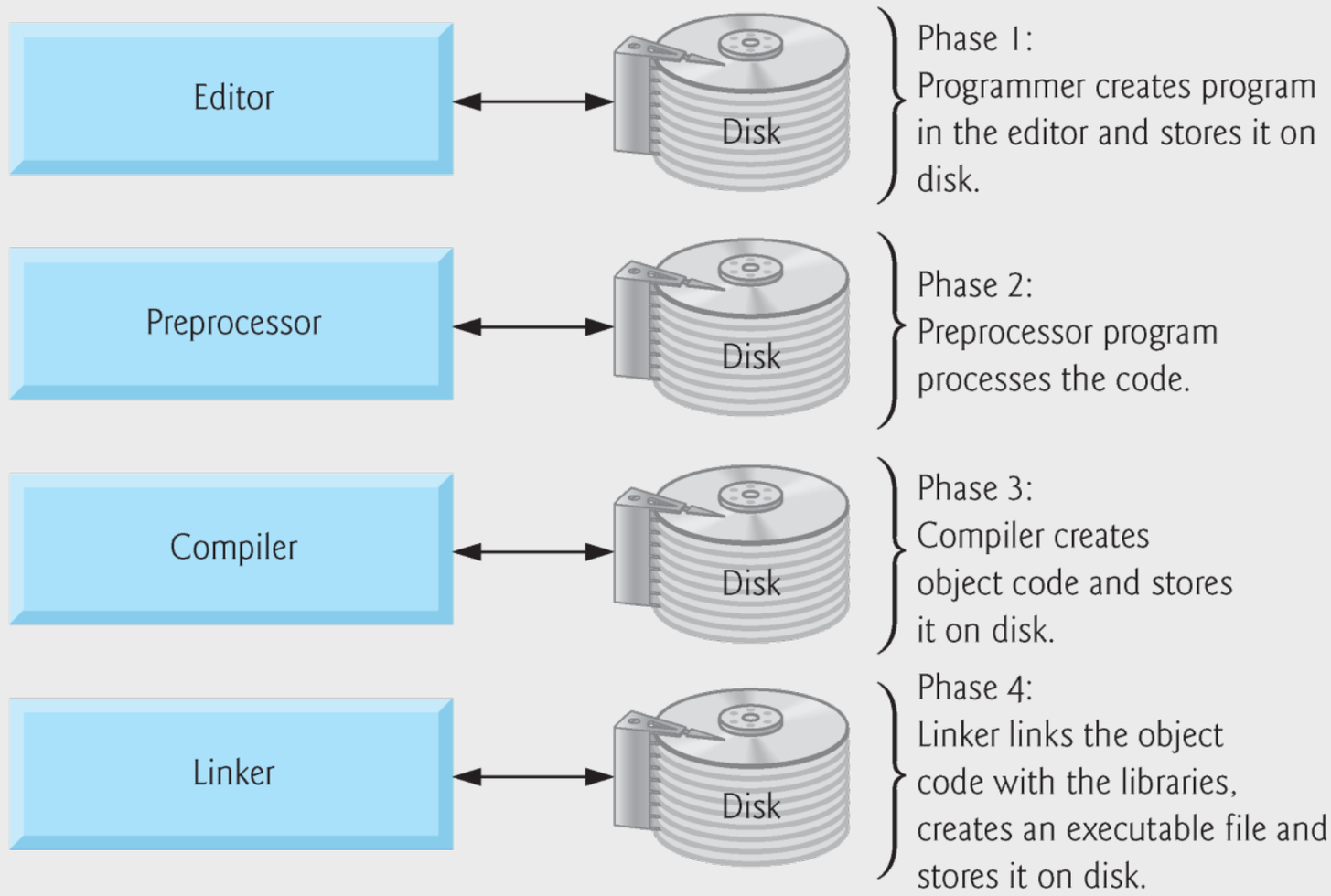


Fig. 1.1 | Typical C development environment. (Part 1 of 2.)

Typical C development environment (Part 2 of 2)

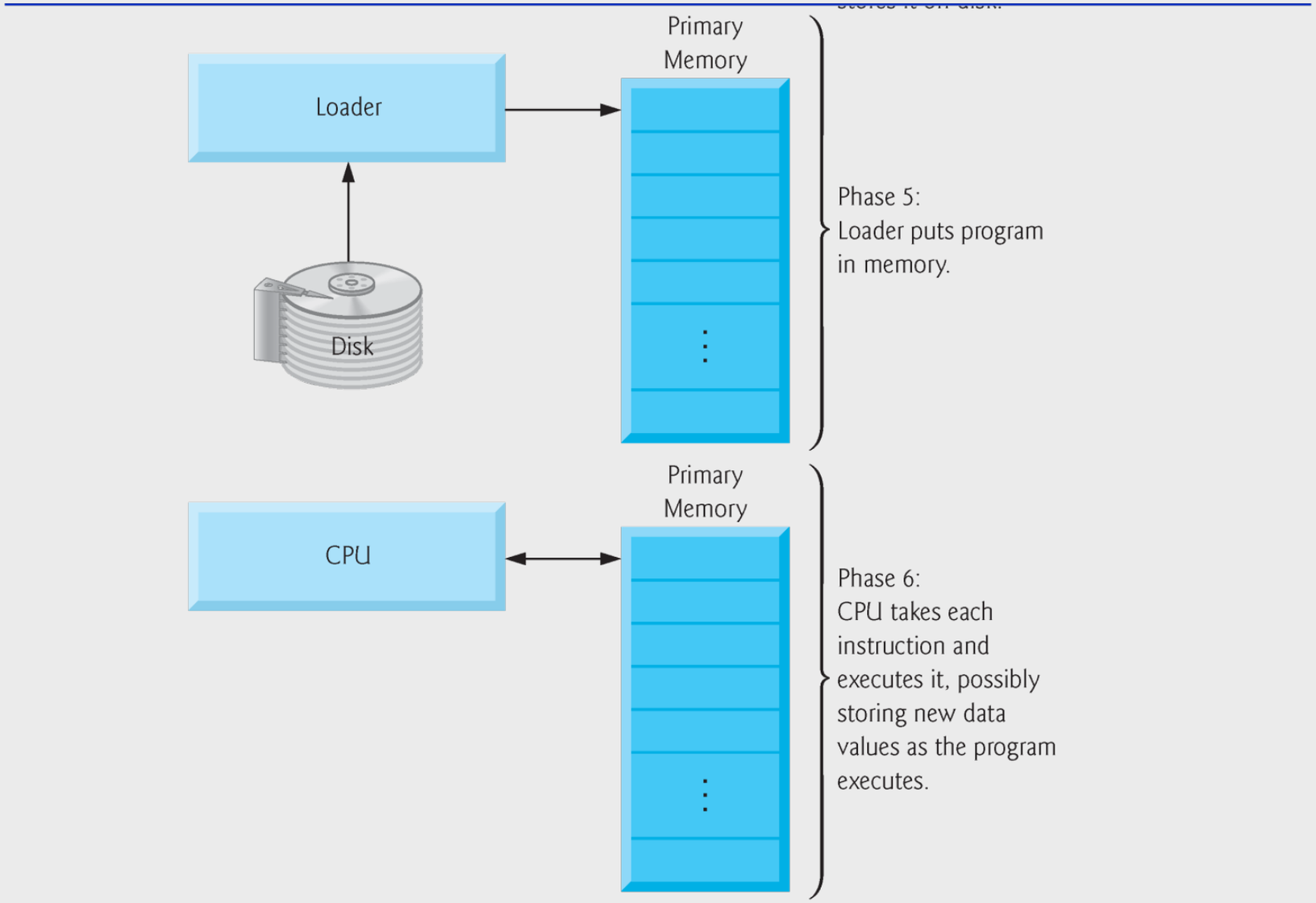


Fig. 1.1 | Typical C development environment. (Part 2 of 2.)

Typical C Program Development Environment (Cont.)

- Phase 4 is called linking.
  - A linker links the object code with the code for the missing functions to produce an executable file (with no missing pieces).
- On a typical Linux system, the command to compile and link a program is called cc (or gcc).
- To compile and link a program named welcome.c type
  - gcc welcome.c
- If the program compiles and links correctly, a file called a.out is produced.

Typical C Program Development Environment (Cont.)

- Phase 5 is called loading.
- Before a program can be executed, the program must first be placed in memory.
- This is done by the loader, which takes the executable image from disk and transfers it to memory.
- Additional components from shared libraries that support the program are also loaded.
- Finally, the computer, under the control of its CPU, executes the program one instruction at a time.
- To load and execute the program on a Linux system, type ./a.out at the Linux prompt and press Enter

Typical C Program Development Environment (Cont.)

- Most C programs input and/or output data.
- Certain C functions take their input from stdin (the standard input stream), which is normally the keyboard, but stdin can be connected to another stream.
- Data is often output to stdout (the standard output stream), which is normally the computer screen, but stdout can be connected to another stream.
- There is also a standard error stream referred to as stderr.
- The stderr stream (normally connected to the screen) is used for displaying error messages.

Notes About C

- C does not guarantee portability
- For additional technical details on C, read the C Standard document itself or the book by Kernighan and Ritchie (The C Programming Language, Second Edition).

Unix Introduction

Unix Introduction (1)

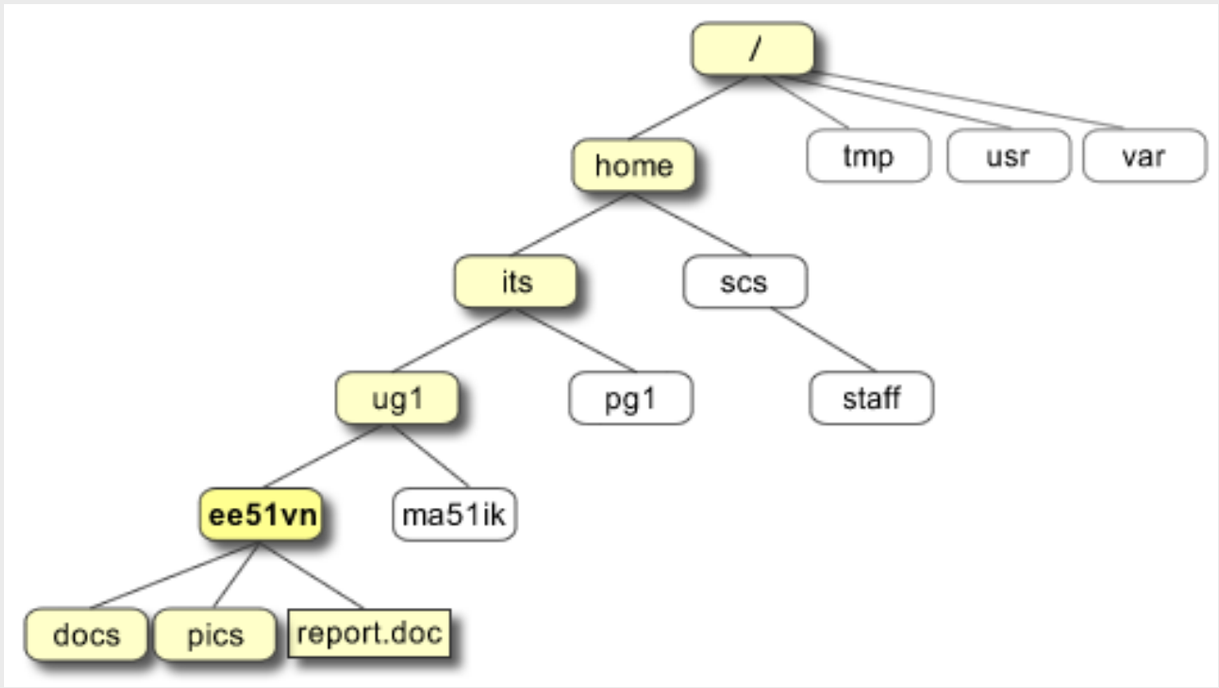
- The kernel
  - The hub of the operating system: it allocates time and memory to programs and handles the file store and communications in response to system calls
- The shell
  - Command Line Interpreter (CLI).
  - It interprets the commands the user types in and arranges for them to be carried out.

Unix Introduction (2)

- File and processes
  - Everything in Unix is either a file or a process.
  - Process
    - Executing programming identified by a unique PID (process identifier)
  - File
    - A collection of data
    - Examples: a document, the text of a program, binary files, a directory

Unix Introduction (3)

- The Directory Structure
  - The top of the hierarchy is traditionally called root.
  - The full path to the file report.doc is "/home/its/ug1/ee51vn/report.doc"



Basic Commands (1)

Connections and users	
whoami	prints the ID of the current logged-in user
who	prints a list of other users who are logged
Directories	
pwd	prints the full path of the current directory
cd dirPath	changes your current working directory to be dirPath
ls	prints a list of the files & directories
mkdir dirPath	creates a new directory of dirPath
Environment Variables	
echo $SHELL	shows the currently used shell
export	export PATH=$HOME/bin;$PATH
env	set and print environment variables
set	shows all shell variables

Basic Commands (2)

Displaying the contents of files	
cat filePath	prints contents of specified file
more filePath	prints contents of specified file one screen at a time
less filePath	similar to more
head filePath	prints first few lines at top of specified file
tail filePath	prints last few lines at bottom of specified file
file filePath	examines specified file and makes a good guess as to its type
cp from to	copies specified file
mv from to	same as cp except source file is deleted
rm filePath	deletes specified file(s)
rmdir filePath	delete specified directory (or directories)

Basic Commands (3)

Finding files that satisfy a criteria	
find dirPath -name '*.txt'	finds and lists all files under the specified directory whose name ends with .txt
Searching for text patterns inside files	
grep ‘keyword’ *.txt	searches all .txt files in current directory hierarchy for the text "keyword". For each match a filename and the matching line of text from the file is displayed.
Some other powerful utilities and commands	
wc filePath	displays number of lines, words, chars and bytes in that file
diff filePath1 filePath2	displays the line by line differences between two text files
comm filePath1 filePath2	like diff but displays common lines instead of differences
cmp filePath1 FilePath2	like diff, but for binary files
od filePath	displays specified binary file in octal or hex

Basic Commands (4)

Creating a symlink (a shortcut)	
ln -s source_file target_file	create a symlink or shortcut to that directory/file
File paths, directories and volumes	
whereis cmd	finds out the paths of the cmd in standard binary directories
which cmd	find the command actually been invoked.
On-line Manual Pages	
man cmd	displays the on-line manual pages

Introduction to C Programming

Introduction

- The C language facilitates a structured and disciplined approach to computer program design.
- In this chapter we introduce C programming and present several examples that illustrate many important features of C.

Hello World!!

- Example: fig02_01.c

```c
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   printf( "Welcome to C!\n" );

   return 0; /* indicate that program ended successfully */
} /* end function main */
```

```
^_^ mftsai@MBP [~/Classes/CPI_2011_fall/02/codes] ls
fig02_01.c fig02_03.c fig02_04.c fig02_05.c fig02_13.c
^_^ mftsai@MBP [~/Classes/CPI_2011_fall/02/codes] gcc fig02_01.c
^_^ mftsai@MBP [~/Classes/CPI_2011_fall/02/codes] ls
a.out      fig02_01.c fig02_03.c fig02_04.c fig02_05.c fig02_13.c
^_^ mftsai@MBP [~/Classes/CPI_2011_fall/02/codes] ./a.out
Welcome to C!
^_^ mftsai@MBP [~/Classes/CPI_2011_fall/02/codes]
```

Hello World!!

- Example: fig02_01.c

```
 3 #include <stdio.h>
 4
 5 /* function main begins program execution */
 6 int main( void )
 7 {
 8    printf( "Welcome to C!\n" );
 9
10    return 0; /* indicate that program ended successfully */
11 } /* end function main */
```

```
^_^ mftsai@MBP [~/Classes/CPI_2011_fall/02/codes] ls
fig02_01.c fig02_03.c fig02_04.c fig02_05.c fig02_13.c
^_^ mftsai@MBP [~/Classes/CPI_2011_fall/02/codes] gcc fig02_01.c
^_^ mftsai@MBP [~/Classes/CPI_2011_fall/02/codes] ls
a.out      fig02_01.c fig02_03.c fig02_04.c fig02_05.c fig02_13.c
^_^ mftsai@MBP [~/Classes/CPI_2011_fall/02/codes] ./a.out
Welcome to C!
^_^ mftsai@MBP [~/Classes/CPI_2011_fall/02/codes]
```

Hello World!!

- Example: fig02_01.c

```c
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   printf( "Welcome to C!\n" );

   return 0; /* indicate that program ended successfully */
} /* end function main */
```

```
^_^ mftsai@MBP [~/Classes/CPI_2011_fall/02/codes] ls
fig02_01.c fig02_03.c fig02_04.c fig02_05.c fig02_13.c
^_^ mftsai@MBP [~/Classes/CPI_2011_fall/02/codes] gcc fig02_01.c
^_^ mftsai@MBP [~/Classes/CPI_2011_fall/02/codes] ls
a.out      fig02_01.c fig02_03.c fig02_04.c fig02_05.c fig02_13.c
^_^ mftsai@MBP [~/Classes/CPI_2011_fall/02/codes] ./a.out
Welcome to C!
^_^ mftsai@MBP [~/Classes/CPI_2011_fall/02/codes]
```

Hello World!!

- Example: fig02_01.c

```
 3 #include <stdio.h>
 4
 5 /* function main begins program execution */
 6 int main( void )
 7 {
 8    printf( "Welcome to C!\n" );
 9
10    return 0; /* indicate that program ended successfully */
11 } /* end function main */
```

```
^_^ mftsai@MBP [~/Classes/CPI_2011_fall/02/codes] ls
fig02_01.c fig02_03.c fig02_04.c fig02_05.c fig02_13.c
^_^ mftsai@MBP [~/Classes/CPI_2011_fall/02/codes] gcc fig02_01.c
^_^ mftsai@MBP [~/Classes/CPI_2011_fall/02/codes] ls
a.out      fig02_01.c fig02_03.c fig02_04.c fig02_05.c fig02_13.c
^_^ mftsai@MBP [~/Classes/CPI_2011_fall/02/codes] ./a.out
Welcome to C!
^_^ mftsai@MBP [~/Classes/CPI_2011_fall/02/codes]
```

A Simple C Program: Printing a Line of Text (Cont.)

- Lines 1 and 2
- **`/* Fig. 2.1: fig02_01.c`**
  `A first program in C */`
  - begin with `/*` and end with `*/` indicating that these two lines are a comment.
  - You insert comments to document programs and improve program readability.
  - Comments do not cause the computer to perform any action when the program is run.

A Simple C Program: Printing a Line of Text (Cont.)

- Comments are ignored by the C compiler and do not cause any machine-language object code to be generated.
- C99 also includes the C++ language's // single-line comments in which everything from // to the end of the line is a comment.

A Simple C Program: Printing a Line of Text (Cont.)

- Line 3
  - **`#include <stdio.h>`**
  - is a directive to the C preprocessor
- Lines beginning with # are processed by the preprocessor before the program is compiled.
- Line 3 tells the preprocessor to include the contents of the standard input/output header (**`<stdio.h>`**) in the program.
  - This header contains information used by the compiler when compiling calls to standard input/output library functions such as **`printf`**.

A Simple C Program: Printing a Line of Text (Cont.)

- Line 6
  - **`int main( void ) { ... }`**
- is a part of every C program.
- The parentheses after main indicate that main is a program building block called a function.
- The keyword **`int`** to the left of main indicates that main "returns" an integer (whole number) value.
- The **`void`** in parentheses here means that the main function does not receive any parameters.

A Simple C Program: Printing a Line of Text (Cont.)

- Line 8
  - **`printf( "Welcome to C!\n" );`**
- instructs the computer to perform an action, namely to print on the screen the string of characters marked by the quotation marks.
  - A string is sometimes called a character string, a message or a literal.
  - The entire line, including **`printf`**, its argument within the parentheses and the semicolon (`;`), is called a statement.
  - Notice that the characters `\n` were not printed on the screen.
  - The backslash (`\`) is called an escape character.

A Simple C Program: Printing a Line of Text (Cont.)

- Escape sequence

Escape sequence	Description
\n	Newline. Position the cursor at the beginning of the next line.
\t	Horizontal tab. Move the cursor to the next tab stop.
\a	Alert. Sound the system bell.
\\	Backslash. Insert a backslash character in a string.
\"	Double quote. Insert a double-quote character in a string.

A Simple C Program: Printing a Line of Text (Cont.)

- Line 10
  - `return 0; /* indicate that program ended successfully */`
- is included at the end of every main function.
- The keyword return is one of several means we'll use to exit a function.

A Simple C Program: Printing a Line of Text (Cont.)

- We said that **`printf`** causes the computer to perform an action.
- As any program executes, it performs a variety of actions and makes decisions.
- In Chapter 3, we discuss this action/decision model of programming in depth.

A Simple C Program: Printing a Line of Text (Cont.)

- Example: fig02_03.c

```c
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   printf( "Welcome " );
   printf( "to C!\n" );

   return 0; /* indicate that program ended successfully */
} /* end function main */
```

```
^_^ mftsai@MBP [~/Classes/CPI_2011_fall/02/codes] gcc fig02_03.c
^_^ mftsai@MBP [~/Classes/CPI_2011_fall/02/codes] ./a.out
Welcome to C!
```

A Simple C Program: Printing a Line of Text (Cont.)

- Example: fig02_04.c

```
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main( void )
7  {
8     printf( "Welcome\nto\nC!\n" );
9
10    return 0; /* indicate that program ended successfully */
11 } /* end function main */
```

```
^_^ mftsai@MBP [~/Classes/CPI_2011_fall/02/codes] gcc fig02_04.c
^_^ mftsai@MBP [~/Classes/CPI_2011_fall/02/codes] ./a.out
Welcome
to
C!
```

Another Simple C Program: Adding Two Integers

- Uses the Standard Library function **`scanf`** to obtain two integers typed by a user at the keyboard, computes the sum of these values and prints the result using **`printf`**.

Another Simple C Program: Adding Two Integers

- Example: fig02_05.c

```c
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   int integer1; /* first number to be input by user  */
   int integer2; /* second number to be input by user */
   int sum; /* variable in which sum will be stored */.

   printf( "Enter first integer\n" ); /* prompt */
   scanf( "%d", &integer1 ); /* read an integer */

   printf( "Enter second integer\n" ); /* prompt */
   scanf( "%d", &integer2 ); /* read an integer */

   sum = integer1 + integer2; /* assign total to sum */

   printf( "Sum is %d\n", sum ); /* print sum */

   return 0; /* indicate that program ended successfully */
} /* end function main */
```

Another Simple C Program: Adding Two Integers

- Example: fig02_05.c

```c
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   int integer1; /* first number to be input by user  */
   int integer2; /* second number to be input by user */
   int sum; /* variable in which sum will be stored */.

   printf( "Enter first integer\n" ); /* prompt */
   scanf( "%d", &integer1 ); /* read an integer */

   printf( "Enter second integer\n" ); /* prompt */
   scanf( "%d", &integer2 ); /* read an integer */

   sum = integer1 + integer2; /* assign total to sum */

   printf( "Sum is %d\n", sum ); /* print sum */

   return 0; /* indicate that program ended successfully */
} /* end function main */
```

```
^_^ mftsai@MBP [~/Classes/CPI_2011_fall/02/codes] ./a.out
Enter first integer
10
Enter second integer
20
Sum is 30
```

Another Simple C Program: Adding Two Integers (Cont.)

- Lines 8–10

```
int integer1; /* first number to be input by user  */
int integer2; /* second number to be input by user */
int sum; /* variable in which sum will be stored */
```

- The names **integer1**, **integer2** and **sum** are the names of variables.
  - A variable is a location in memory where a value can be stored for use by a program.
  - These definitions specify that the variables **integer1**, **integer2** and **sum** are of type **int**, which means that these variables will hold integer values

Another Simple C Program: Adding Two Integers (Cont.)

- All variables must be defined with a name and a data type immediately after the left brace that begins the body of main before they can be used in a program.
- An identifier is a series of characters consisting of letters, digits and underscores (_) that does not begin with a digit.
- A variable name in C is any valid identifier.
- C is case sensitive—uppercase and lowercase letters are different in C, so a1 and A1 are different identifiers.

Another Simple C Program: Adding Two Integers (Cont.)

- A syntax error is caused when the compiler cannot recognize a statement.
- Syntax errors are also called compile errors, or compile-time errors.

Another Simple C Program: Adding Two Integers (Cont.)

- Line 12
  - `printf( "Enter first integer\n" ); /* prompt */`
- prints the literal Enter first integer on the screen and positions the cursor to the beginning of the next line.
  - This message is called a prompt because it tells the user to take a specific action.
- The next statement
  - `scanf( "%d", &integer1 ); /* read an integer */`
- uses **`scanf`** to obtain a value from the user.
- The **`scanf`** function reads from the standard input, which is usually the keyboard.

Another Simple C Program: Adding Two Integers (Cont.)

- The second argument of **`scanf`** begins with an ampersand (`&`)—called the address operator in C—followed by the variable name.

Another Simple C Program: Adding Two Integers (Cont.)

- Line 15
  - `printf( "Enter second integer\n" ); /* prompt */`
- Displays the message Enter second integer on the screen, then positions the cursor to the beginning of the next line.
- This **`printf`** also prompts the user to take action.
- The statement
  - `scanf( "%d", &integer2 ); /* read an integer */`
- obtains a value for variable **`integer2`** from the user.

Another Simple C Program: Adding Two Integers (Cont.)

- The assignment statement in line 18
  - **`sum = integer1 + integer2;`**
    `/*assign total to sum */`
- The statement is read as, "**`sum`** gets the value of **`integer1 + integer2`**." Most calculations are performed in assignments.
- The **=** operator and the **+** operator are called binary operators because each has two operands.

Another Simple C Program: Adding Two Integers (Cont.)

- Line 20
  - **`printf( "Sum is %d\n", sum ); /* print sum */`**
- calls function **`printf`** to print the literal Sum is followed by the numerical value of variable sum on the screen.
- This **`printf`** has two arguments
  - **`"Sum is %d\n"`** and **`sum`**.

Memory Concepts

- Variable names such as **`integer1`**, **`integer2`** and **`sum`** actually correspond to locations in the computer's memory.
- Every variable has a name, a type and a value.

```
scanf( "%d", &integer1 ); /* read an integer */
```

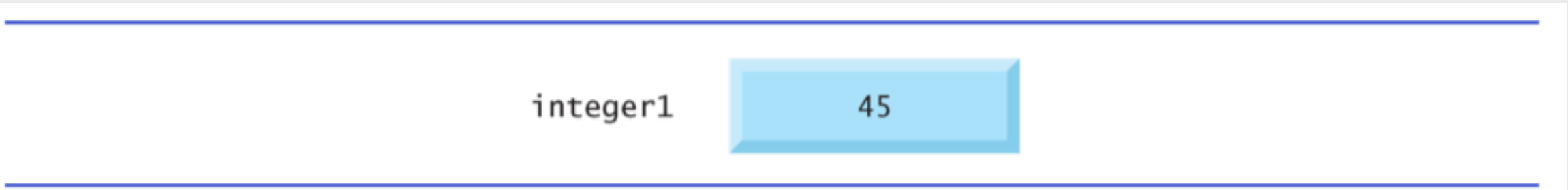


Fig. 2.6 | Memory location showing the name and value of a variable.

Memory Concepts

Memory Concepts

Fig. 2.7 | Memory locations after both variables are input.

Memory Concepts

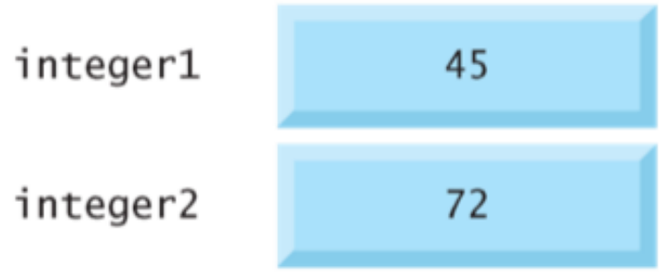


Fig. 2.7 | Memory locations after both variables are input.

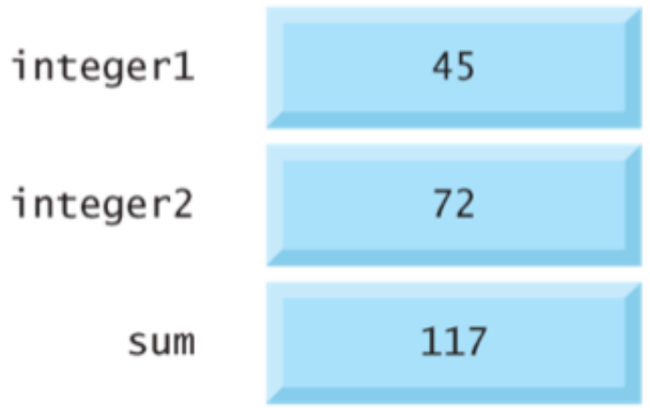


Fig. 2.8 | Memory locations after a calculation.

Arithmetic in C

C operation	Arithmetic operator	Algebraic expression	C expression
Addition	+	$f + 7$	`f + 7`
Subtraction	–	$p - c$	`p - c`
Multiplication	*	bm	`b * m`
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	`x / y`
Remainder	%	$r \bmod s$	`r % s`

Fig. 2.9 | Arithmetic operators.

Arithmetic in C

- Parentheses are used in C expressions in the same manner as in algebraic expressions.
- For example, to multiply a times the quantity **`b + c`** we write **`a * ( b + c )`**.

Precedence of arithmetic operators

Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they're evaluated left to right.
*	Multiplication	Evaluated second. If there are several, they're evaluated left to right.
/	Division	
%	Remainder	
+	Addition	Evaluated last. If there are several, they're evaluated left to right.
-	Subtraction	

Fig. 2.10 | Precedence of arithmetic operators.

Step 1. `y = 2 * 5 * 5 + 3 * 5 + 7;` *(Leftmost multiplication)*

`2 * 5 is 10`

Step 2. `y = 10 * 5 + 3 * 5 + 7;` *(Leftmost multiplication)*

`10 * 5 is 50`

Step 3. `y = 50 + 3 * 5 + 7;` *(Multiplication before addition)*

`3 * 5 is 15`

Step 4. `y = 50 + 15 + 7;` *(Leftmost addition)*

`50 + 15 is 65`

Step 5. `y = 65 + 7;` *(Last addition)*

`65 + 7 is 72`

Step 6. `y = 72` *(Last operation—place* `72` *in* `y`*)*

Fig. 2.11 | Order in which a second-degree polynomial is evaluated.

Decision Making: Equality and Relational Operators

- Executable C statements either perform actions (such as calculations or input or output of data) or make decisions (we'll soon see several examples of these).
- A simple version of C's if statement that allows a program to make a decision based on the truth or falsity of a statement of fact called a condition.

Equality and relational operators

Algebraic equality or relational operator	C equality or relational operator	Example of C condition	Meaning of C condition
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

Fig. 2.12 | Equality and relational operators.

Example: fig02_13.c

```c
int main( void )
{
    int num1; /* first number to be read from user  */
    int num2; /* second number to be read from user */

    printf( "Enter two integers, and I will tell you\n" );
    printf( "the relationships they satisfy: " );

    scanf( "%d%d", &num1, &num2 ); /* read two integers */

    if ( num1 == num2 ) {
        printf( "%d is equal to %d\n", num1, num2 );
    } /* end if */

    if ( num1 != num2 ) {
        printf( "%d is not equal to %d\n", num1, num2 );
    } /* end if */

    if ( num1 < num2 ) {
        printf( "%d is less than %d\n", num1, num2 );
    } /* end if */

    if ( num1 > num2 ) {
        printf( "%d is greater than %d\n", num1, num2 );
    } /* end if */

    if ( num1 <= num2 ) {
        printf( "%d is less than or equal to %d\n", num1, num2 );
    } /* end if */

    if ( num1 >= num2 ) {
        printf( "%d is greater than or equal to %d\n", num1, num2 );
    } /* end if */

    return 0; /* indicate that program ended successfully */
} /* end function main */
```

Example: fig02_13.c

```
 7 int main( void )
 8 {
 9     int num1; /* first number to be read from user  */
10     int num2; /* second number to be read from user */
11
12     printf( "Enter two integers, and I will tell you\n" );
13     printf( "the relationships they satisfy: " );
14
15     scanf( "%d%d", &num1, &num2 ); /* read two integers */
16
17     if ( num1 == num2 ) {
18         printf( "%d is equal to %d\n", num1, num2 );
19     } /* end if */
20
21     if ( num1 != num2 ) {
22         printf( "%d is not equal to %d\n", num1, num2 );
23     } /* end if */
24
25     if ( num1 < num2 ) {
26         printf( "%d is less than %d\n", num1, num2 );
27     } /* end if */
28
29     if ( num1 > num2 ) {
30         printf( "%d is greater than %d\n", num1, num2 );
31     } /* end if */
32
33     if ( num1 <= num2 ) {
34         printf( "%d is less than or equal to %d\n", num1, num2 );
35     } /* end if */
36
37     if ( num1 >= num2 ) {
38         printf( "%d is greater than or equal to %d\n", num1, num2 );
39     } /* end if */
40
41     return 0; /* indicate that program ended successfully */
42 } /* end function main */
```

Precedence of arithmetic operators

Operators	Associativity
()	left to right
* / %	left to right
+ -	left to right
< <= > >=	left to right
== !=	left to right
=	right to left

Fig. 2.14 | Precedence and associativity of the operators discussed so far.

C's keywords

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

Keywords added in C99

_Bool	_Complex	_Imaginary	inline	restrict

Fig. 2.15 | C's keywords.