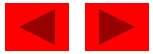
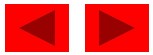
Chapter 2 - Introduction to C Programming

Outline



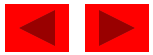
Objectives

- In this chapter, you will learn:
  - To be able to write simple computer programs in C.
  - To be able to use simple input and output statements.
  - To become familiar with fundamental data types.
  - To understand computer memory concepts.
  - To be able to use arithmetic operators.
  - To understand the precedence (順序， order of evaluation) of arithmetic operators.
  - To be able to write simple decision making statements.
  - To understand C's fundamental and modified data types



2.1 Introduction

- C programming language
  - *Structured* and *disciplined* approach to program design
- Structured programming
  - Introduced in chapters 3 and 4
  - Used throughout the remainder of the book
- Steps to write a program
  - Define the problem to be solved with the computer
  - Design the program's input/output (what the user should give (Data)/see (Information))
  - Break the problem into logical steps to achieve this output
  - Write the program (with an editor)
  - Compile the program
  - Test the program to make sure it performs as you expected

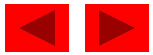


2.2 A Simple C Program: Printing a Line of Text

```
1   /* Fig. 2.1: fig02_01.c
2      A first program in C */
3   #include <stdio.h>
4
5   /* function main begins program execution */
6   int main()
7   {
8      printf( "Welcome to C!\n" );
9
10     return 0; /* indicate that program ended successfully */
11
12  } /* end function main */
```

```
Welcome to C!
```

- Comments (註解)
  - Text surrounded by /* and */ is ignored by computer
  - Text followed by // is ignored by computer (C++ style)
  - Used to describe program
- **#include <stdio.h>**
  - Preprocessor directive (前置處理器指令): Tells computer to load contents of a certain file (header files, 標頭檔)
  - <stdio.h> allows standard input/output operations

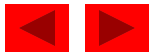


2.2 A Simple C Program: Printing a Line of Text

```
1   /* Fig. 2.1: fig02_01.c
2      A first program in C */
3   #include <stdio.h>
4
5   /* function main begins program execution */
6   int main()
7   {
8      printf( "Welcome to C!\n" );
9
10     return 0; /* indicate that program ended successfully */
11
12  } /* end function main */
```

```
Welcome to C!
```

- **int main() 或 int main(void)**
  - Each C (and C++) program contains one or more functions, exactly one of which must be main (每個 C 程式必定有一個 main() 函數，而且只能有一個)
  - Parenthesis ( ) is used to indicate a function
  - int means that main "returns" an integer value
  - Braces ({ and }) indicate a block (程式區塊): The body of every function must be contained in braces

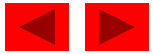


2.2 A Simple C Program: Printing a Line of Text

- `printf( "Welcome to C!\n" );`
  - Instructs computer to perform an action
    - i.e., prints the *string of characters* within quotes (" ") on screen
  - Entire line is called a statement (敘述句)
    - All statements must end with a semicolon (`;`, also known as the *statement terminator*)
  - Argument (參數，引數)

    Function (*argument*), e.g.,

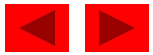
    ```
    printf( "Welcome to C!\n" );
    ```
  - Escape character (`\`, 跳脫字元)
    - Indicates that printf should do something out of the ordinary
    - `\n` is the newline character



2.2 A Simple C Program: Printing a Line of Text

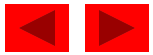
Escape sequence	Description
\n	Newline. Position the cursor at the beginning of the next line.
\t	Horizontal tab. Move the cursor to the next tab stop.
\a	Alert. Sound the system bell.
\\	Backslash. Insert a backslash character in a string.
\"	Double quote. Insert a double-quote character in a string.

Tab: 欄標 跳欄



2.2 A Simple C Program: Printing a Line of Text

- `return 0;`
  - A way to exit a function
  - `return` 0, in this case, means that the program is terminated normally
- Right brace `}`
  - Indicates end of `main` has been reached
- Linker
  - When a function is called, linker locates it in the library
  - Inserts it into object program
  - If function name is misspelled, the linker will produce an error message because it will not be able to find function in the library



Basics of a Typical C Program Development Environment

- Phases of C Programs:

1. *Edit*
2. *Preprocess*
3. *Compile*
4. *Link*
5. *Load*
6. *Execute*

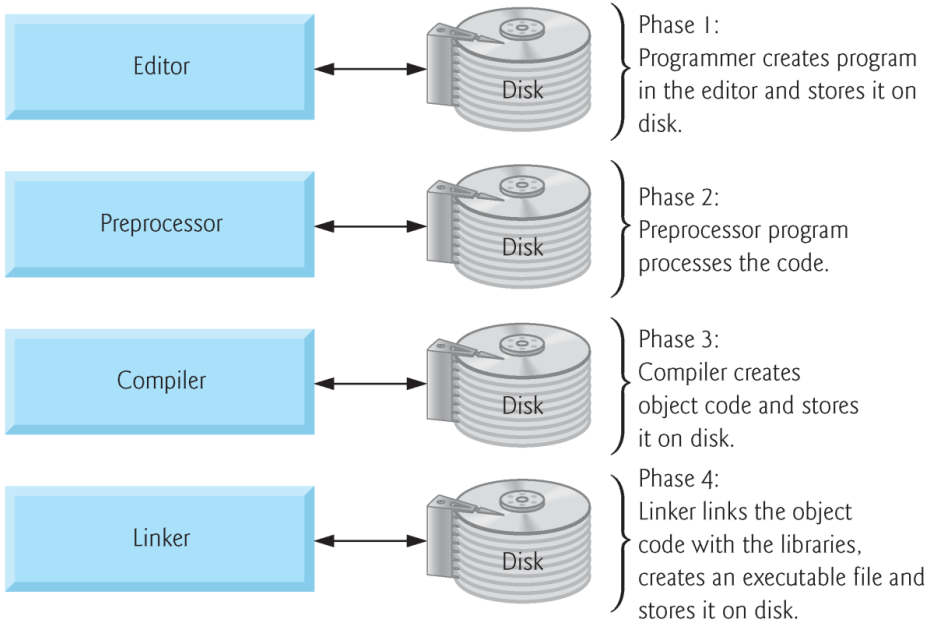


Fig. 1.1 | Typical C development environment. (Part 1 of 2.)

Basics of a Typical C Program Development Environment

- Phases of C Programs:

1. *Edit*
2. *Preprocess*
3. *Compile*
4. *Link*
5. *Load*
6. *Execute*

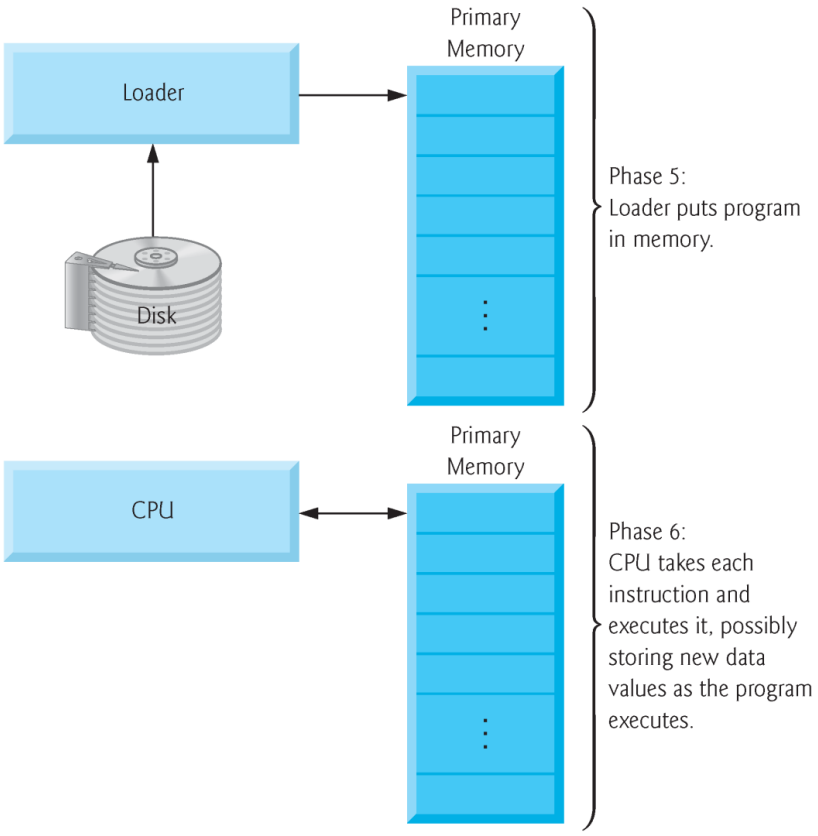
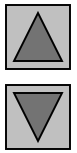


Fig. 1.1 | Typical C development environment. (Part 2 of 2.)



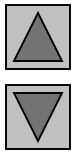
Outline

fig02_03.c

```
1  /* Fig. 2.3: fig02_03.c
2     Printing on one line with two printf statements */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     printf( "Welcome " );
9     printf( "to C!\n" );
10
11    return 0; /* indicate that program ended successfully */
12
13 } /* end function main */
```

Program Output

```
Welcome to C!
```



Outline

fig02_04.c

```c
/* Fig. 2.4: fig02_04.c
   Printing multiple lines with a single printf */
#include <stdio.h>

/* function main begins program execution */
int main()
{
   printf( "Welcome\nto\nC!\n" );

    return 0; /* indicate that program ended successfully */

} /* end function main */
```

Program Output

```
Welcome
to
C!
```

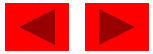
Debug the Following Source Code

Identify and correct the errors in the following program:

```
1     /* Fig. 2.1: fig02_01e.c
2        A first program in C */
3     #include <stdio.h>;
4
5     /* function main begins program execution */
6     int main();
7     {
8        print( "Welcome to C!\n" )
9
10       return 0; // indicate that program ended successfully
11
12    /* end function main */
```

Ans:

```
3     #include <stdio.h>
6     int main()
8        printf( "Welcome to C!\n" );
12    }/* end function main */
```

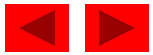
Debug the Following Source Code

Identify and correct the errors in the following program:

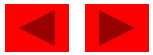
```
1     // Fig. 2.1: fig02_01e.c
2        A first program in C */
3     #include <stdio.h>
4
5     /* function main begins program execution */
6     int Main()
7     {
8        printf( Welcome to C!\n );
9
10       return 0; /* indicate that program ended successfully */
11    }
12    /* end function main */
```

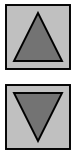
Ans:

```
1     /* Fig. 2.1: fig02_01e.c
2     int main()
8        printf("Welcome to C!\n");
```

Another Simple C Program - Adding Two Integers





Outline

Another Program – Adding Two Integers

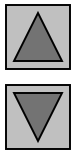
fig02_05.c

```c
/* Fig. 2.5: fig02_05.c
   Addition program */
#include <stdio.h>

/* function main begins program execution */
int main()
{
   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 */
```

宣告整數變數
int integer1;
變數型態 變數名稱;

從鍵盤讀取整數數值，並放到變數 integer1 的位置，注意變數名稱前要加 &

計算部份，將 integer1、integer2 相加後的結果給 sum



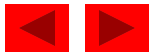
Outline

Program Output

```
Enter first integer
45
Enter second integer
72
Sum is 117
```

2.3 Another Simple C Program: Adding Two Integers

- As before
  - Comments, `#include <stdio.h>` and `int main()`
- `int integer1, integer2, sum;`
  - Definition of variables
    - Variables: **locations in memory** where a value can be stored
  - `int` means the variables can hold *integers* (`-1, 3, 0, 47`)
  - Variable names (identifiers)
    - `integer1`, `integer2`, `sum`
    - Identifiers: consist of letters, digits (cannot begin with a digit) and underscores (`_`). They are *case sensitive*.
  - Definitions must appear before executable statements
    - If an executable statement references an undeclared variable it will produce a syntax (compiler) error



2.3 Another Simple C Program: Adding Two Integers

- `scanf( "%d", &integer1 );`
  - Obtains a value from the user
    - `scanf` uses *standard input* (usually **keyboard**)
  - This `scanf` statement has two arguments
    - `%d` - indicates data should be a *decimal integer*
    - `&integer1` - location in memory to store variable (也就是，指向整數變數 integer1 在記憶體的位置)
    - `&` is confusing in beginning – for now, just remember to include it with the variable name in `scanf` statements. It will be discussed later (i.e., concept of *pointer*)
  - When executing the program the user responds to the `scanf` statement by
    1. **typing in a number, then**
    2. **pressing the *enter* (return) key**

2.3 Another Simple C Program: Adding Two Integers

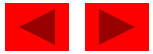
- = (assignment operator)
  - Assigns a *value* (on the right) to a *variable* (on the left)
  - Is a binary operator (has two operands)

```
sum = variable1 + variable2;
```

sum gets variable1 + variable2;

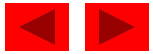
  - Variable receiving value is on the left
- `printf( "Sum is %d\n", sum );`
  - Similar to `scanf`
    - `%d` means decimal integer will be printed
    - `sum` specifies what integer will be printed
  - Calculations can be performed inside `printf` statements

```
printf( "Sum is %d\n", integer1 + integer2 );
```

2.4 Memory Concepts

- Variables
  - Variable names correspond to locations in the computer's memory
  - Every variable has:
    (1) a name, (2) a type, (3) a size and (4) a value
  - Whenever a new value is placed into a variable (through `scanf`, for example), it replaces (and destroys) the previous value
  - Reading variables from the corresponding memory does not change them



2.4 Memory Concepts

- A visual representation

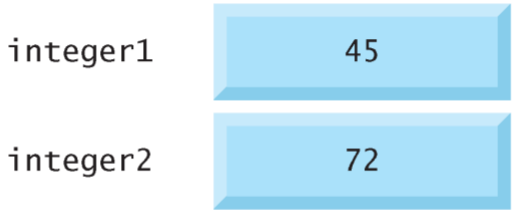


Fig. 2.7 | Memory locations after both variables are input.

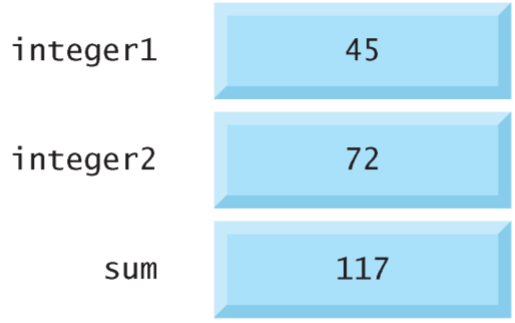
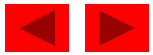


Fig. 2.8 | Memory locations after a calculation.

2.5 Arithmetic

- Arithmetic calculations
  - Use * for multiplication and / for division
  - Integer division (/) *truncates remainder*
    - `7 / 5` evaluates to `1`
  - Modulus operator (%) *returns the remainder*
    - `7 % 5` evaluates to `2`
- Operator precedence (優先順序)
  - Some arithmetic operators act before others (e.g., multiplication before addition)
    - Use parenthesis when needed
  - Example: Find the average of three variables `a`, `b` and `c`
    - Do not use: `a + b + c / 3`
    - Use: `(a + b + c ) / 3`

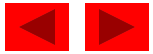


2.5 Arithmetic

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`

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 Division Remainder	Evaluated second. If there are several, they're evaluated left to right.
+ –	Addition Subtraction	Evaluated last. If there are several, they're evaluated left to right.



2.5 Arithmetic

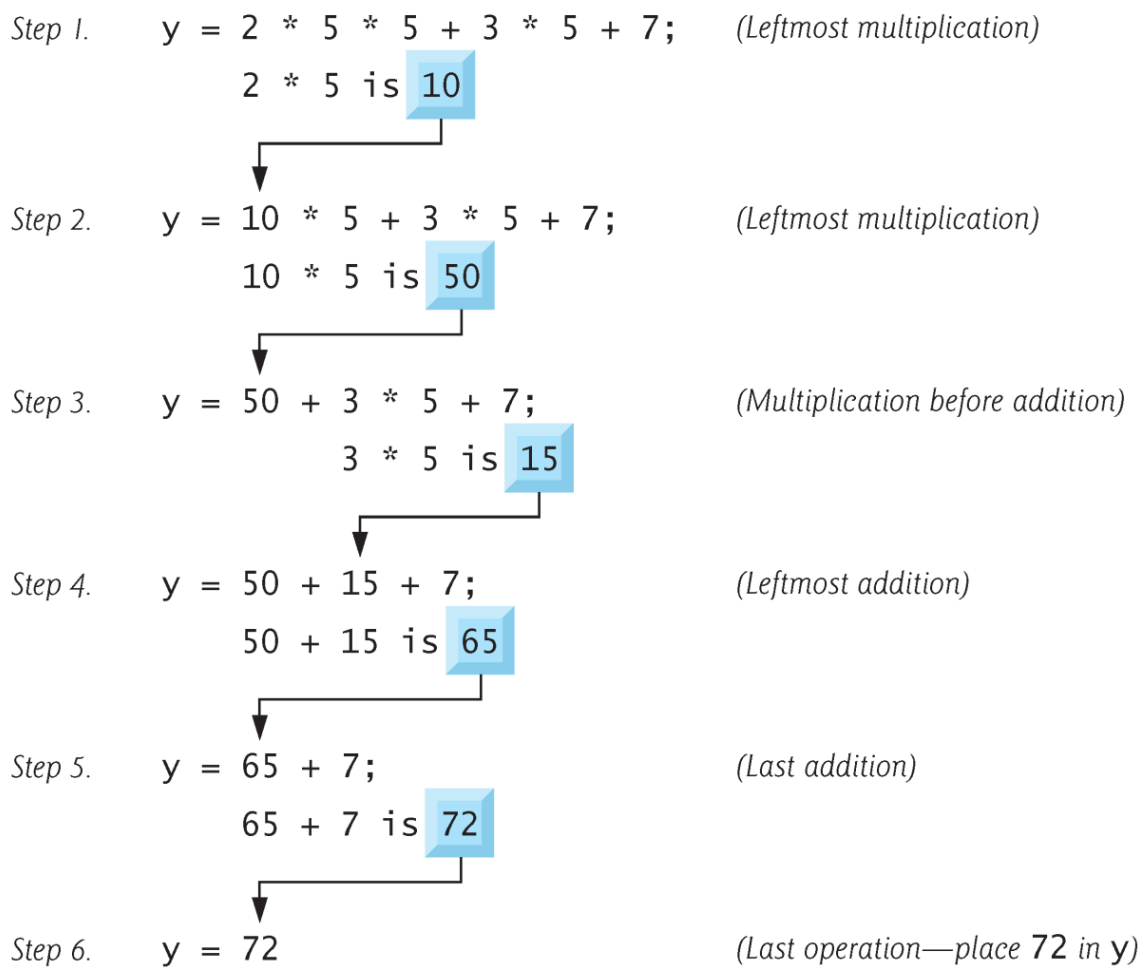
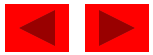
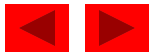


Fig. 2.11 | Order in which a second-degree polynomial is evaluated.



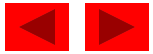
2.6 Decision Making: Equality and Relational Operators

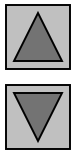
- Two types of executable statements
  - Perform *actions* (calculations, input/output of data)
  - Perform *decisions*, e.g., print `"pass"` or `"fail"` given the value of a test grade
- `if` control statement
  - Simple version in this section, more detail later
  - If a condition is `true`, then the body of the `if` statement executed
    - 0 is `false`, non-zero is `true`
  - Control always resumes after the `if` structure
- Keywords
  - Special words *reserved* for C
  - Cannot be used as identifiers or variable names



2.6 Decision Making: 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





Outline

fig02_13.c (Part 1 of 2)

```
1  /* Fig. 2.13: fig02_13.c
2     Using if statements, relational
3     operators, and equality operators */
4  #include <stdio.h>
5
6  /* function main begins program execution */
7  int main()
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
```

```
if( 條件式 )
{
    statement 1;
    statement 2;
    . . .
     如果合乎條件的話，
     就做大括號中的部份；
}
```

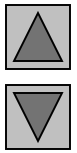
或者，如果只有一個 **statement** 時，可簡化成

```
if( 條件式 )
    statement   ;
```

Question:

```
if ( 條件式 )   ;
printf( "This is a test.\n" );
```

執行結果是??



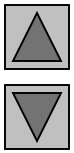
Outline

```
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
43 } /* end function main */
```

fig02_13.c (Part 2 of 2)

```
Enter two integers, and I will tell you
the relationships they satisfy: 3 7
3 is not equal to 7
3 is less than 7
3 is less than or equal to 7
```

Program Output



Outline

Program Output (continued)

```
Enter two integers, and I will tell you
the relationships they satisfy: 22 12
22 is not equal to 12
22 is greater than 12
22 is greater than or equal to 12
```

```
Enter two integers, and I will tell you
the relationships they satisfy: 7 7
7 is equal to 7
7 is less than or equal to 7
7 is greater than or equal to 7
```

More on if Statements

```
if( 條件式 )
{
   statement 1;
   statement 2;
   statement 3;
}
```

```
if( 條件式 )
   statement 1;
statement 2;
statement 3;
```

```
if( 條件式 ) ;
   statement 1;
   statement 2;
   statement 3;
```

```
if( 條件式 )
{
   statement 1;
   statement 2;
}
   statement 3;
```

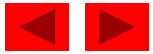

2.6 Decision Making: Equality and Relational Operators

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

Precedence and associativity of the operators discussed so far.

2.7 Data Types and Variables

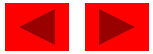
- C's Fundamental Data Type
  - `int` Integral numbers such as 1, 2, 3 and so on
  - `float` Low/medium precision real numbers
  - `double` Medium/high precision real numbers
  - `char` Text characters such as 'a', 'b', '@' and so on
- C's Modified Data Type
  - `short int` small to medium sized integral numbers
  - `long int` Medium to large sized integral numbers, such as -245 563, 123 456
  - long double Medium/high value/precision real numbers such as 2.0×10^{2310}



```
/* SIZEOF.C--Program to tell the size of the C variable */
/*           type in bytes */

#include <stdio.h>

main()
{

    printf( "\nA char             is %d bytes", sizeof( char ));
    printf( "\nAn int             is %d bytes", sizeof( int ));
    printf( "\nA short            is %d bytes", sizeof( short ));
    printf( "\nA long             is %d bytes", sizeof( long ));
    printf( "\nAn unsigned char   is %d bytes", sizeof( unsigned char ));
    printf( "\nAn unsigned int    is %d bytes", sizeof( unsigned int ));
    printf( "\nAn unsigned short  is %d bytes", sizeof( unsigned short ));
    printf( "\nAn unsigned long   is %d bytes", sizeof( unsigned long ));
    printf( "\nA float            is %d bytes", sizeof( float ));
    printf( "\nA double           is %d bytes", sizeof( double ));
    printf( "\nA long double      is %d bytes\n", sizeof( long double ));

    return 0;
}
```

```
A char             is 1  bytes
An int             is 4  bytes
A short            is 2  bytes
A long             is 4  bytes
An unsigned char   is 1  bytes
An unsigned int    is 4  bytes
An unsigned short  is 2  bytes
An unsigned long   is 4  bytes
A float            is 4  bytes
A double           is 8  bytes
A long double      is 8  bytes  -  for Visual C++ Compiler
A long double      is 10  bytes -  for Borland Compiler
```


- **Bi**nary Digi**t**s (bit): 1 and 0
  - The computer can combine the two digital states to represent letters, numbers, colors, sounds, images, shapes, and even odors.
  - An "on" or "off" electronic state is represented by a bit, short for binary digit
- Encoding Systems: Bits (位元) and Bytes (位元組)
  - Bits are combined according to an encoding system to represent letters, numbers, and special characters, collectively referred to as alphanumeric characters
  - The combination of bits used to represent a character is called a byte (**B**inar**y** **Te**rm, 8 bits/byte)
  - 8 bits = byte
- Representation of a Character
  - ASCII (American Standard Code for Information Interchange) is the most popular encoding system for PCs and data communication
    - ASCII – 7 bits
    - ANSI – 8 bits/byte
    - UNICODE – 16 bits
    - Big5 – 16 bits
- Storage Capacities
  - KB (kilobyte) = 2^{10} Bytes = 1,024 Bytes ≈ 10^3 Bytes
  - MB (megabyte) = 2^{20} Bytes = 1,024 KB = 1,048,576 Bytes ≈ 10^6 Bytes
  - GB (gigabyte) = 2^{30} Bytes = 1,024 MB ≈ 10^9 Bytes
  - TB (terabyte) = 2^{40} Bytes = 1,024 GB ≈ 10^{12} Bytes



Typical Size and Range of Data Types

For Borland Compiler

Data Type	Size Bytes	Min Value	Max Value
char	1	-128	127
short int	2	-32768	32767
int	4	-2147483648	2147483647
long int	4	-2147483648	2147483647
float	4	1.17549e-38	3.40282e+38
double	8	2.22507e-308	1.79769e+308
long double	10	3.3621e-4932	1.18973e+4932

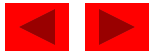
For Visual C++ and C Compiler

Data Type	Size Bytes	Min Value	Max Value
char	1	-128	127
short int	2	-32768	32767
int	4	-2147483648	2147483647
long int	4	-2147483648	2147483647
float	4	1.17549e-38	3.40282e+38
double	8	2.22507e-308	1.79769e+308
long double	8	2.22507e-308	1.79769e+308

1 byte, $2^8 = 256$

2 bytes, $2^{16} = 65536$

4 bytes, $2^{32} = 4294967296$



Errors in Addition of Two Large Integers

```
/* IntegerError.c
   Error in large integer addition
   Overflow in integer addition
   IntegerError.c
*/

#include <stdio.h>

int main()
{  int A1, A2, A3, B1, B2;

   A1 = 1500000000;
   A2 = 1500000000;
   A3 =  500000000;

   B1 = A1 + A2;
   B2 = A1 + A3;

   printf( "A1 + A2 = %d + %d = %d\n", A1, A2, B1 );
   printf( "A1 + A3 = %d + %d = %d\n", A1, A3, B2 );

   return 0; /* indicates successful termination */

} /* end main */
```

int 4 byte

B1=3,000,000,000> 2,147,483,647

B2=2,000,000,000< 2,147,483,647

```
A1 + A2 = 1500000000 + 1500000000 = -1294967296

A1 + A3 = 1500000000 + 500000000 = 2000000000
```


Conversion between Types

```
/*Test integer/float Conversion by calculating 5/3 + 4 testIntFloat.c */


#include <stdio.h>

int main()
{  int A1, A2, A3;
   float B1, B2, B3, B4, B5, B6, B7, B8, B9, B10;

   A1 =  3;
   A2 =  5;
   A3 =  4;

   B1 = A2/A1 + A3;
   B2 = A2/3.0 + A3;
   B3 = (float)A2/(float)A1 + A3;
   B4 = (float)A2/A1 + A3   ;
   B5 = A2/(float)A1 + A3   ;
   B6 = A2/A1 + (float)A3   ;
   B7 = (float)A3 + A2/A1   ;
   B8 = (float)(A2/A1) + A3 ;
   B9 = A3 + (float)A2/A1   ;
   B10= A2/A1*(float)A1 + A3;

   printf( " A1 = 3 ; A2 = 5 ; A3 = 4 \n\n");
   printf( " A2/A1 + A3                 = %f\n", B1);
   printf( " A2/5.0 + A3                = %f\n", B2);
   printf( " (float)A2/(float)A1 + A3   = %f\n", B3);
   printf( " (float)A2/A1 + A3          = %f\n", B4);
   printf( " A2/(float)A1 + A3          = %f\n", B5);
   printf( " A2/A1 + (float)A3          = %f\n", B6);
   printf( " (float)A3 + A2/A1          = %f\n", B7);
   printf( " (float)(A2/A1) + A3        = %f\n", B8);
   printf( " A3 + (float)A2/A1          = %f\n", B9);
   printf( " A2/A1*(float)A1 + A3       = %f\n", B10);

   return 0; /* indicates successful termination */

} /* end main */
```

Outputs:

```
 A1 = 3 ; A2 = 5 ; A3 = 4


 A2/A1 + A3                 = 5.000000
 A2/3.0 + A3                = 5.666667
 (float)A2/(float)A1 + A3   = 5.666667
 (float)A2/A1 + A3          = 5.666667
 A2/(float)A1 + A3          = 5.666667
 A2/A1 + (float)A3          = 5.000000
 (float)A3 + A2/A1          = 5.000000
 (float)(A2/A1) + A3        = 5.000000
 A3 + (float)A2/A1          = 5.666667
 A2/A1*(float)A1 + A3       = 7.000000
```


Variables

A **variable** is a **named** *data storage location* in your computer's *memory*.

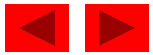
Every variable has a name, a type, a size and a value

By using a variable's name in your program, you are, in effect, *referring to the data stored there*.

Variable *Names*

To use variables in your C programs, you must know how to create variable names. In C, variable names must adhere to the following rules:

- The name can contain *letters*, *digits*, and *underscore* character (_).
- The *first character* of the name must be a *letter*. The underscore is also a legal first character, but its use is not recommended.
- *Case matters* (that is, upper- and lowercase letters). Thus, the names `count` and `Count` refer to two different variables.
- C *keywords* can't be used as variable names. A keyword is a word that is part of the C language.



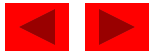
Keywords

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
```

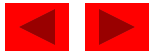



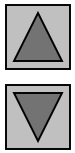
Some Examples of Legal and Illegal C Variable Names

Variable Name	Legality
`Percent`	Legal
`y2x5__fg7h`	Legal, but not advised
`annual_profit`	Legal
`_1990_tax`	Legal but not advised
`savings#account`	Illegal: Contains the illegal character #
`double`	Illegal: Is a C keyword
`9winter`	Illegal: First character is a digit

Because C is case-sensitive, the names percent, PERCENT, and Percent would be considered as *three different variables*.

For many compilers, a C variable name can be up to 31 characters long. (It can actually be longer than that, but the compiler looks at only the first 31 characters of the name.) With this flexibility, you can create variable names that reflect the data being stored.





Outline

Another Program – Adding Two Integers

fig02_05.c

```
1  /* Fig. 2.5: fig02_05.c
2     Addition program */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int integer1; /* first number to be input by user  */
9     int integer2; /* second number to be input by user */
10    int sum;      /* variable in which sum will be stored */
11
12    printf( "Enter first integer\n" );  /* prompt */
13    scanf( "%d", &integer1 );           /* read an integer */
14
15    printf( "Enter second integer\n" ); /* prompt */
16    scanf( "%d", &integer2 );           /* read an integer */
17
18    sum = integer1 + integer2;          /* assign total to sum */
19
20    printf( "Sum is %d\n", sum );       /* print sum */
21
22    return 0;  /* indicate that program ended successfully */
23
24 } /* end function main */
```

宣告整數變數
int integer1;
變數型態 變數名稱;

從鍵盤讀取整數數值，並放到變數 integer1 的位置，注意變數名稱前要加 &

計算部份，將 integer1、integer2 相加後的結果給 sum

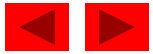
More on printf( ) Conversion Specifiers

The format string must contain one ***conversion specifier*** for each ***printed variable***. **`printf()`** then displays each variable as directed by its corresponding conversion specifier. For example, if you're printing a variable that is a signed decimal integer (types int and long), use the **`%d`** conversion specifier. For an unsigned decimal integer (types unsigned int and unsigned long), use **`%u`**. For a floating-point variable (types float and double), use the **`%f`** specifier.

Specifier	Meaning	Types Converted	Examples
`%c`	Single character	char	`A`
`%d`	Signed decimal integer	int, short	`1234`
`%ld`	Signed long decimal integer	long	`1234`
`%f` or `%.3f` or `%15.3f`	Decimal floating-point number	float, double	`1234567.890000;` `1234567.890` `1234567.890`
`%s`	Character string	char arrays	`This is a test`
`%u`	Unsigned decimal integer	unsigned int, unsigned short	`1234`
`%lu`	Unsigned long decimal integer	unsigned long	`1234`
`%e or %E`	Floating-point value in exponential notation	float, double	`1.234568e+006;` `1.234568E+006`
`%g or %G`	Floating-point value in f or e (or E) form, whichever is shorter	float, double	`1.23457e+006`

```c
/* printf_format testing */
/* Printing floating-point numbers with
   floating-point conversion specifiers */

#include <stdio.h>

int main()
{  float  test1;
   double test2;

   test1 = 1234567.890123456789;
   test2 = 1234567.890123456789;

   printf( "%f\t%f\n",           test1,test2 );
   printf( "%.3f\t%.3f\n\n",     test1,test2 );
   printf( "%.8f\t%.8f\n\n\n",   test1,test2 );
   printf( "%e\t%e\n",           test1,test2 );
   printf( "%E\t%E\n\n",         test1,test2 );
   printf( "%.4e\t%.4e\n\n",     test1,test2 );
   printf( "%.10e\t%.10e\n\n\n", test1,test2 );
   printf( "%g\t%g\n",           test1,test2 );
   printf( "%G\t%G\n",           test1,test2 );

   return 0; /* indicates successful termination */

} /* end main */
```


```
Outputs                                          Format Specifiers

1234567.875000  1234567.890123                   "%f\t%f\n"
1234567.875     1234567.890                      "%.3f\t%.3f\n\n"

1234567.87500000        1234567.89012346         "%.8f\t%.8f\n\n\n"


1.234568e+006   1.234568e+006                    "%e\t%e\n"
1.234568E+006   1.234568E+006                    "%E\t%E\n\n"

1.2346e+006     1.2346e+006                      "%.4e\t%.4e\n\n"

1.2345678750e+006       1.2345678901e+006        "%.10e\t%.10e\n\n\n"

1.23457e+006    1.23457e+006                     "%g\t%g\n"
1.23457E+006    1.23457E+006                     "%G\t%G\n"
```

Case Study – Converting Miles to Kilometers

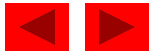
- Steps to write a program (repeat)
  1. Define the problem to be solved with the computer
  2. Design the program's input/output (what the user should give/see)
  3. Break the problem into logical steps to achieve this output
  4. Write the program (with an editor)
  5. Compile the program
  6. Test the program to make sure it performs as you expected
- Step 1: Define Problem:
  - Convert Miles to Kilometers
- Step 2: Identify Input/Output
  - `Input: miles /* the distance in miles */`
  - `Output: kms /* the distance in kilometers */`
- Step 3: Devise Algorithm (演算法、演算步驟)

  Step 3.1: Get the distance in miles (from keyboard)

  Step 3.2: Convert the distance to kilometers

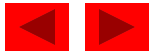
  The distance in kilometers is 1.609 times the distance in miles

  Step 3.3: Display the distance (on screen)
- Step 4: Write the program



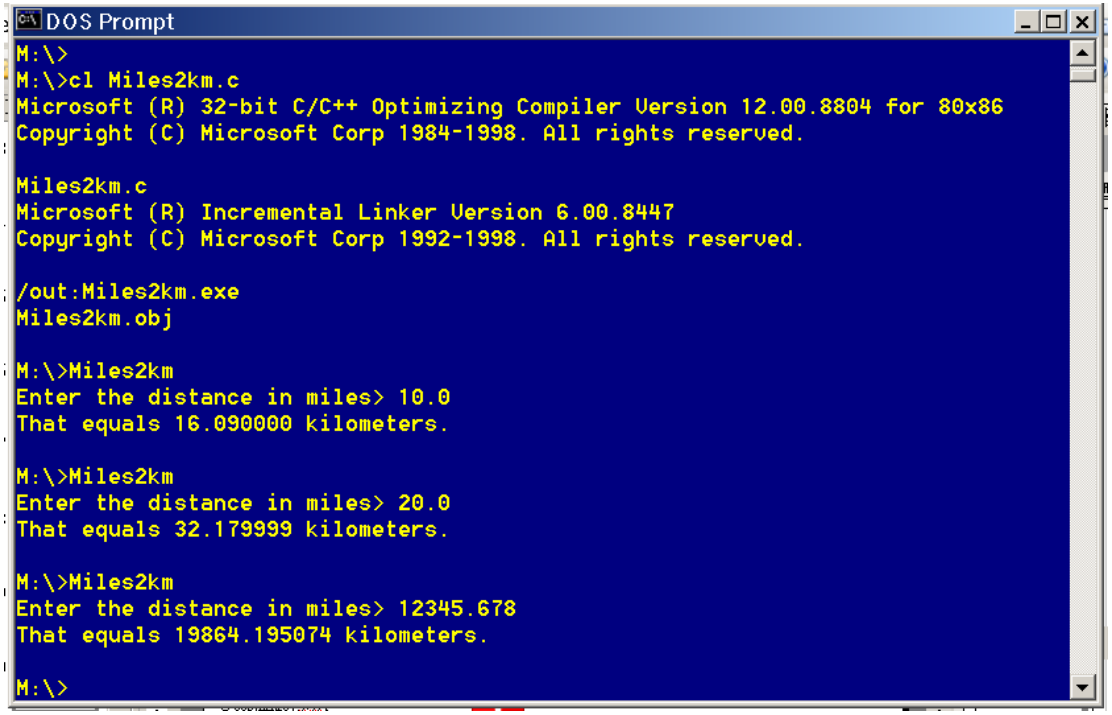
Case Study – Converting Miles to Kilometers

```
/*
 * program Mile2Km.c
 * Converts distance in miles to kilometers.
 */

#include <stdio.h>                /* printf, scanf definitions */

int main()
{
      float  miles,         /* input - distance in miles.       */
             kms,           /* output - distance in kilometers  */
             kms_per_mile; /* conversion constant              */

      /* Get the distance in miles. */
      printf("Enter the distance in miles> ");
      scanf("%f", &miles);

      /* Convert the distance to kilometers. */
      kms_per_mile = 1.609;
      kms = kms_per_mile * miles;

      /* Display the distance in kilometers. */
      printf("That equals %f kilometers.\n", kms);

      return 0;
}
```

Case Study – Converting Miles to Kilometers

- Step 5: Compile the program
  - Using Visual C++ or any ANSI-C Compiler
  - If something goes wrong during compiling – syntax errors?
- Step 6: Testing
  - To verify the program works properly, enter a few test values of miles (e.g., 10.0 miles).
  - If something goes wrong during executing (running) the program – logical errors?



Exercises

2.7 Identify and correct the errors in each of the following statements (Note: there may be more than one error per statement):

a) `scanf( "d", value );`

ANS: `scanf( "%d", &value );`

b) `printf( "The product of %d and %d is %d"\n, x, y );`

ANS: `printf("The product of %d and %d is %d\n", x, y, z );`

c) `firstNumber + secondNumber = sumOfNumbers`

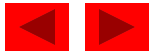
ANS: `sumOfNumbers = firstNumber + secondNumber;`

d)
```
if ( number => largest )
      largest == number;
```

ANS:
```
if ( number >= largerst )
        largest = number;
```

e)
```
*/ Program to determine the largest of three integers
   /*
```

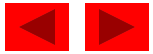
ANS:
```
/* Program to determine the largest of three integers
   */
```

Exercises

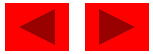
2.7 Identify and correct the errors in each of the following statements (Note: there may be more than one error per statement):

```
f) Scanf( "%d", anInteger );
ANS: scanf( "%d", &anInteger );
g) printf("Remainder of %d divided by %d is\n", x, y, x%y);
ANS: printf("Remainder of %d divided by %d is %d\n", x, y,
  x%y );
h) if ( x = y );
      printf( %d is equal to %d\n", x, y );
ANS: if ( x == y ) /* ; removed */
        printf( "%d is equal to %d\n", x, y );
i) print( "The sum is %d\n," x + y );
ANS: printf( "The sum is %d\n", x + y );
j) Printf( "The value you entered is: %d\n, &value );
ANS: printf( "The value you entered is: %d\n", value );
```

Review

- In this chapter, you have learned:
  - To be able to write simple computer programs in C.
  - To be able to use simple input and output statements.
  - To become familiar with fundamental data types.
  - To understand computer memory concepts.
  - To be able to use arithmetic operators.
  - To understand the precedence (order of evaluation) of arithmetic operators.
  - To be able to write simple decision making statements.
  - To understand C's fundamental and modified data types



2.7 Data Types and Variables (補充)