Chapter7：資料型態

Signed Integer

正負號由最左邊的 bit 判斷：0 是正的，1 是負的。
舉例：16-bit integer 最大的值表示為 0111 1111 1111 1111 (2^{15} - 1)
宣告：`int`

Unsigned Integer

因為沒有正負號判斷(皆為正)，所以最左邊的 bit 也算是值得一部分。
舉例：16-bit unsigned integer 最大的值表示為 1111 1111 1111 1111 (2^{16} – 1)
宣告：`unsigned int`

另外還有其他整數類型：
`short int` 也可寫成 `short`，範圍比 `int` 來的小
`long int` 也可寫成 `long`，範圍必 `int` 來的大

On 16-bit machine

Type	*Smallest Value*	*Largest Value*
`short int`	–32,768	32,767
`unsigned short int`	0	65,535
`int`	–32,768	32,767
`unsigned int`	0	65,535
`long int`	–2,147,483,648	2,147,483,647
`unsigned long int`	0	4,294,967,295

On 32-bit machine

Type	*Smallest Value*	*Largest Value*
`short int`	–32,768	32,767
`unsigned short int`	0	65,535
`int`	–2,147,483,648	2,147,483,647
`unsigned int`	0	4,294,967,295
`long int`	–2,147,483,648	2,147,483,647
`unsigned long int`	0	4,294,967,295

On 64-bit machine

Type	*Smallest Value*	*Largest Value*
`short int`	–32,768	32,767
`unsigned short int`	0	65,535
`int`	–2,147,483,648	2,147,483,647
`unsigned int`	0	4,294,967,295
`long int`	-2^{63}	$2^{63}-1$
`unsigned long int`	0	$2^{64}-1$

標頭檔<limits.h>定義了各種整數的範圍大小

Integer Constant

Decimal constants contain digits between 0 and 9, but must not begin with a zero:

15 255 32767

Octal constants contain only digits between 0 and 7, and must begin with a zero:

017 0377 077777

Hexadecimal constants contain digits between 0 and 9 and letters between a and f, and always begin with 0x:

0xf 0xff 0x7fff

The letters in a hexadecimal constant may be either upper or lower case:

0xff 0xfF 0xFf 0xFF 0Xff 0XfF 0XFf 0XFF

一般來說 decimal integer 會儲存為 `int`，但如果數值太大超過 `int` 的儲存空間，會儲存為 `long int`；如果一開始儲存為 `long int` 但數值太大的話，編譯器會嘗試把它儲存為 `unsigned long int`。

如果是 octal integer 或 hexadecimal integer 尋找適合的儲存種類順序會是：
`int, unsigned int, long int, unsigned long int`

To force the compiler to treat a constant as a long integer, just follow it with the letter `L` (or `l`):

```
15L  0377L  0x7fffL
```

To indicate that a constant is unsigned, put the letter `U` (or `u`) after it:

```
15U  0377U  0x7fffU
```

L and U may be used in combination:

```
0xffffffffUL
```

The order of the `L` and `U` doesn't matter, nor does their case.

Integer Overflow

Integer overflow 指的是當儲存位元數不足以負荷要儲存的值的時候。

1. signed integer 發生 overflow 的時候會是 undefined behavior。

2. unsigned integer 發生 overflow 的時候還是會有值，因為編譯器會忽略多出來的位元數只用截掉之後的位元來計算值，舉例來說，如果在 16-bit integer number 65635 上加一的話，結果會是 0。

Reading and Writing integer

1. 當讀取或輸出 `int` 的值時後，會用 `d`
2. 當讀取或輸出 `unsigned int` 值的時候，不會用 `d`，而是用 `u, o, x`

```
unsigned int u;

  scanf("%u", &u);  /* reads  u in base 10 */
  printf("%u", u);  /* writes u in base 10 */
  scanf("%o", &u);  /* reads  u in base  8 */
  printf("%o", u);  /* writes u in base  8 */
  scanf("%x", &u);  /* reads  u in base 16 */
  printf("%x", u);  /* writes u in base 16 */
```

3. 當讀取或輸出 `short int` 值的時候，會在 `d, u, o, x` 前面加上 `h`

```
short int s;
scanf("%hd", &s);
printf("%hd",s);
```

當讀取或輸出 `long int` 值的時候，會在 `d, u, o, x` 前面加上 `l`

```
long int l;
scanf("%ld", &l);
printf("%ld", l);
```

Floating Types

`float`	Single-precision floating-point
`double`	Double-precision floating-point
`long double`	Extended-precision floating-point

基本上現代大部分的電腦浮點數是依照 IEEE standard 754：

Type	*Smallest Positive Value*	*Largest Value*	*Precision*
`float`	1.17549×10^{-38}	3.40282×10^{38}	6 digits
`double`	2.22507×10^{-308}	1.79769×10^{308}	15 digits

在標頭檔<`float.h`>裡有定義各個浮點數的特性

- Valid ways of writing the number 57.0:

  `57.0  57.  57.0e0  57E0  5.7e1  5.7e+1`
  `.57e2  570.e-1`

  e 或 E 都可以

在預設中，floating constant 會被儲存為 double-precision number

如過想要儲存為 single-precision number，要在數字後面加上 `f` 或 `F`，
舉例：`57.0f`

如果想要儲存為 `long double` 形式，要在數字後面加上 `l` 或 `L`，
舉例：`57.0L`

Conversion Specification

如果要讀取或輸出 single-precision floating-point 的值，要使用 conversion specification `%e, %f, %g`
讀取 `double` 形式時，要在 `e, f, g` 前面加上 `l` 或 `L`，輸出 `double` 形式則不用再加上 `l`
舉例：

```
double d;
scanf("%lf", &d);
printf("%f", d);
```

讀取或輸出 long double 時，要再 e, f, g 前面加上 l 或 L
舉例：

```
long double l;
scanf("%lf", &l);
printf("%lf", l);
```

最後整理：
讀取時，double 和 long double 要加上 l，而 float 不用
輸出時，long double 要加上 l，而 double 和 float 都不用

Character types

char 是唯一的形式，每一台電腦根據的規則有可能不一樣，所以同一個字元 char 的值在不同電腦上不一定相同。現今大多是根據 ASCII (American Standard Code for Information Interchange)，是 7-bit code。

宣告一個 char 變數 ch，ch 可以被指派一個字元，要用''單引號。
(要注意 ch 只能被指派一個字元，不能被派一多個字元，更不能是字串)
舉例：

```
char ch;
ch = 'A';
ch = 'a';
ch = '0';
ch = ' ';
ch = 'abc';/*** WRONG ***/
ch = "a";/*** WRONG ***/
ch = "abc;"/*** WRONG ***/
```

''單引號是一種運算子，可以算出一個字元的值

Reading and Writing Characters using `scanf` and `printf`

```
char ch;
scanf("%c", &ch);
printf("%c", ch);
```

一般來說，scanf 在讀取一個字元之前不會跳過 white-space character，也就是說如果你輸入一個空白鍵，那麼 ch 就會是一個空白鍵。如果要跳過一個空白鍵去讀去下一個字元，可以這樣寫

```
scanf(" %c", &ch);/*skip white space, then read ch*/
```

Reading and Writing Character using `getchar` and `putchar`

```
char ch;
ch = getchar();/*reads a character and stores it in ch*/
putchar(ch);
```

注意 getchar()要用指派的

用 putchar 和 getchar 會比用 printf 和 scanf 還要快

Type Conversion

執行運算的時候，operand 應該要用相同大小，也就是 bit 的數量要一樣。
當有不同 operand 是不同的大小(形式)做運算時，編譯器可能會自行生成指令改變某些 operand 的形式，讓這個運算可以執行。

可以運算：

two 16-bit integer

不能運算：

a 16-bit integer and a 32-bit integer

a 32-bit integer and a 32-bit floating-point number

The Usual Arithmetic Conversions

現在有 f 形式為 float 和 i 形式為 int，做運算 f + i。

把 i 形式換成 float 會比把 f 形式換成 int 安全，
理由是前者最壞的情況只可能因為轉換造成精度不準確，
而後者轉換後可能會因為不在 int 的範圍內造成沒有意義的運算。

混著浮點數的運算 (差在大小)

1. 如果有一個 operand 是 long double，那麼其他都要轉換成 long double
2. 如果沒有 long double，但是有一個 double，那麼其他都要轉換成 double
3. 如果沒有 long double 和 double，但是有一個 float，那麼其他都要轉成 float

舉例：
如果有一個 operand 型別是 long int，其他 operand 的型別是 double，那麼 long int 就會被轉換成 double

Casting

雖然 C 會自動轉換很方便，有時我們還是需要更精確的轉換。

A cast expression

(*type-name*) *expression*

求浮點數的小數部分

```
float f, frac_part;

  frac_part = f - (int) f;
```

把 f 轉換成 int

```
i = (int) f;  /* f is converted to int */
```

除法運算強制轉換

```
float quotient;
  int dividend, divisor;

  quotient = dividend / divisor;
```

為了防止可能的截斷，我們可以這樣轉換

```
quotient = (float) dividend / divisor;
```

把 dividend 轉換成 float 會使得編譯器也把 divisor 轉換成 float
C 把(type-name)看成是 unary operator，所以上述轉換等同是
((float) dividend) / divisor
同樣效果的其他表達

```
quotient = dividend / (float) divisor;
quotient = (float) dividend / (float) divisor;
```

要注意轉換時是否已經發生 overflow

```
long i;
  int j = 1000;

  i = j * j;    /* overflow may occur */
```

解決方法

```
i = (long) j * j;
```

不能寫成

```
i = (long) (j * j);   /*** WRONG ***/
```

因為 overflow 可能已經發生

Type Definition

我們可以使用`#define` 創造出 Boolean type：

```
#define BOOL int
```

用 `typedef` 或許是更好的方法：

```
typedef int Bool;
```

舉例：

```
Bool flag;   /* same as int flag; */
```

讓變數名稱、宣告更好理解：

```
typedef float Dollars;
Dollars cash_in, cash_out;
```

而不用寫成

```
float cash_in, cash_out;
```

The `sizeof` Operator

`sizeof` (*type-name*)

是一個 unsigned integer 表示一種形式在記憶體中需要的大小，以 byte 計數。
像是 `sizeof(char)` 是 1，`sizeof(int)` 在 32-bit machaine 中是 4。

`sizeof` 可以不需要括號，像是 `sizeof(i)` 可以寫成 `sizeof i`，
但是如過有運算子的時候就要注意，像是 `sizeof(i + j)` 就不能寫成 `sizeof i + j`，因為編譯器會看成 `(sizeof i) + j`。