現代 C 語言程式設計 (試讀本)

Release 1.2.0

Michelle Chen

Nov 09, 2021

目錄

版權聲明 (COPYRIGHT)

著作權 © 2021 Michelle Chen，保留一切權利。未經授權任意拷貝、引用、翻印、散佈，均屬違法。

若未另外聲明，本書所有的程式碼皆採用 Apache 2.0 授權，歡迎各位讀者在符合授權的前提下使用本書的程式碼。若本書中的程式有使用到第三方軟體，則以該軟體原本的授權方式為準。

免責聲明 (DISCLAIMER)

本書的內容 (文字、圖片、電腦程式等) 僅為一般性質的資訊，而非正式的技術文件。我們致力於保持本書的內容是即時和正確的，但我們無法保證本書內容的完整性、即時性、正確性、可靠性。本書的內容仍可能因人為錯誤、技術性問題等因素造成錯誤。

此外，我們也無法擔保本書使用者因使用本書或直接或間接受到本書的內容所致的任何損失或傷害。本書使用者應自行評估、判斷本書的內容對自己的風險。

本書使用者可以透過本書的超連結前往外部網站，但我們無法控制外部網站的性質、內容和可得性。我們在本網站中加入這些連結不代表我們推薦這些連結的內容或為這些連結的內容背書。我們無法擔保這些連結的內容。

當你使用本書時，表示你同意本書的聲明，會自行評估、判斷使用本書所導致的風險。

版本演進 (HISTORY)

- 1.2.0
  - 改版內容：在 Visual Studio 2022 中建立 C 專案
  - 微調一些版面
- 1.1.0
  - 新增內容：在 Visual Studio 2019 中使用 Clang
  - 新增內容：實用的 C 巨集
- 1.0.5
  - 修改和修正少量文字
- 1.0.4
  - 小幅增修基本概念的章節
- 1.0.3
  - 修改和 Visual C++ 相關的文字
  - 小幅修改 GCC 或 Clang 的章節
- 1.0.2
  - 小幅修改 GCC 或 Clang 的章節
  - 小幅修改 Visual C++ 的章節
  - 小幅修改資料型態的章節
  - 新增計算陣列大小的方式
  - 修復基本輸出入的範例程式
- 1.0.1
  - 修改 C 語言簡介的章節
  - 修改基本概念的章節
  - 修改前置處理器的章節
- 1.0.0
  - 首次發佈

本書所使用的記述方式

對於 C 程式碼，會以語法高亮來輔助閱讀：

```c
#include <main>

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

    return 0;
}
```

同樣地，對於 C 程式碼片段，也會以語法高亮來輔助閱讀：

```c
/* Excerpt */
assert(0 != strcmp("hello", "goodbye"));
```

為了便於在電子書閱讀器上閱讀本書，我們的範例程式碼不會完全遵守 K&R 風格。我們會儘可能地確保排列過的程式碼仍可正確運作。

按照 Unix 的慣例，終端機會以 `$` 符號來表示指令提示符：

```
$ cd path/to/project
```

當使用 `root` 操作 Unix 終端機時，則會改用 `#` 來表示指令提示符：

```
# apt install gcc
```

按照 Windows 的慣例，終端機會以 `>` 來表示指令提示符。為了簡化，不顯示工作目錄：

```
> cd path\to\project
```

前言

學完陣列和指標後，就有足夠的預備知識學習 C 字串。C 語言沒有獨立的字串型別，而 C 字串是以 `char` 或 `wchar_t` 為基礎型別的陣列，所以要有先前文章的鋪陳才容易學習 C 字串。

C 語言的字串方案

在 C 語言中，常見的字串方案有以下數種：

- 固定寬度字元陣列
  - 字元陣列 (character array)
  - 寬字元陣列 (wide character array)，使用 **wchar.h** 函式庫
- 多字節編碼 (multibyte encodings)，像是 Big5 (大五碼) 或 GB2312 等
- 統一碼 (Unicode)：包括 UTF-8、UTF-16、UTF-32 等

有些方案為等寬字元，有些方案則採不等寬字元編碼。預設的字元陣列僅能處理英文文字，其他方案則是為了處理多國語文文字而產生的。

例如，在支援 Unicode 的終端機環境，可以透過 *wchar_t* 印出中文字串：

```c
#include <locale.h>
#include <wchar.h>

int main(void)
{
    /* Trick to print multibyte strings. */
    setlocale(LC_CTYPE, "");

    wchar_t *s = L"你好，世界";
    printf("%ls\n", s);

    return 0;
}
```

由於本文的目的是了解字串的基本操作，我們仍然是以預設的字元陣列為準，不考慮多國語言的情境。

C 字串微觀

我們由 "Hello World" 字串來看 C 字串的組成：

'H'	'e'	'l'	'l'	'o'	' '	'W'	'o'	'r'	'l'	'd'	'\0'

由上圖可知，C 字串除了依序儲存每個字元外，在尾端還會額外加上一個 '\0' 字元，代表字串結束。由於 C 字串需要尾端的 '\0' 字元來判斷字串結束，我們在處理字串時，別忘了在字串尾端加上該字元。

C 語言不會幫程式設計者檢查字串是否正確，而會直接認定字串尾端有附加 '\0'。當字串尾端沒有附加 '\0' 時，會造成字串相關演算法的錯誤。

我們把上圖所示的字元陣列寫成以下的程式碼：

```
#include <assert.h>
#include <string.h>

int main(void)
{
    char s[] = \
        {'H', 'e', 'l', 'l', 'o', ' ',
         'W', 'o', 'r', 'l', 'd', '\0'};

     assert(0 == strcmp(s, "Hello World"));

     return 0;
 }
```

由此可知 C 字串在本質上是尾端為 NULL 的字元陣列。當我們在實作字串相關演算法時也要以這個想法來寫程式。

接下來，我們會介紹數個字串操作的情境。由於 C 標準函式庫已經有 **string.h** 函式庫，在採作字串時應優先使用該函式庫，而非重造輪子；本文展示的程式僅供參考。

計算 C 字串長度

計算字串長度時，不包含尾端的結束字尾，所以 C 字串 "happy" 的字串長度為 5 而非 6。

可參考以下的範例程式碼：

```
#include <assert.h>                    /*  1 */
#include <stddef.h>                    /*  2 */
#include <string.h>                    /*  3 */
```

(continues on next page)

(continued from previous page)

```
int main(void)                          /*  4 */
{                                       /*  5 */
    char s[] = "hello";                 /*  6 */

    size_t sz = 0;                      /*  7 */

    for (size_t i = 0; s[i]; i++) {     /*  8 */
        sz++;                           /*  9 */
    }                                   /* 10 */

    assert(sz == strlen(s));            /* 11 */

    return 0;                           /* 12 */
}                                       /* 13 */
```

我們藉由走訪字串來計算字串的長度，這段動作位於第 8 行至第 10 行。當字串走到尾端的零值時，`s[i]` 會回傳零，這時迴圈會結束。藉由走訪字串陣列一輪就可以知道字串長度。

複製 C 字串

一般使用 `strcpy` 函式的範例，都是預先配置某個長度的字元陣列；本例略加修改，先動態計算來源字串的長度，再由堆積 (heap) 動態配置一塊新的字元陣列，將原本的字元逐一複製到目標字串即完成。參考以下程式碼：

```
#include <assert.h>                     /*  1 */
#include <stddef.h>                     /*  2 */
#include <stdio.h>                      /*  3 */
#include <stdlib.h>                     /*  4 */
#include <string.h>                     /*  5 */

int main(void)                          /*  6 */
{                                       /*  7 */
  char s[] = "Hello World";             /*  8 */

  size_t sz_s = strlen(s);              /*  9 */
  /* Add trailing zero. */
  size_t sz = sz_s + 1;                 /* 10 */

  char *out = \
    malloc(sz * sizeof(char));          /* 11 */
  if (!out) {                           /* 12 */
```

(continues on next page)

(continued from previous page)

```
    fprintf(
      stderr,
      "Failed to allocate C string\n");   /* 13 */
    return 1;                             /* 14 */
  }                                       /* 15 */

  for (size_t i = 0; i < sz_s; i++) {     /* 16 */
    out[i] = s[i];                        /* 17 */
  }                                       /* 18 */

  out[sz-1] = '\0';                       /* 19 */

  assert(
    0 == strcmp(out, "Hello World"));     /* 20 */

  free(out);                              /* 21 */

  return 0;                               /* 22 */
}                                         /* 23 */
```

要配置記憶體前，要先知道記憶體的長度，所以我們在第 10 行計算字串的長度。

接著，我們在第 11 行為字串配置記憶體。

拷貝字串的方式為逐一拷貝字元陣列內的字元，該動作位於第 16 行至第 18 行。

此外，還要為字元加上尾端的零值，該動作為於第 19 行。

在本例中，由於 `out` 是由 heap 配置記憶體，使用完要記得手動釋放，該動作位於第 21 行。

相接兩個 C 字串

原本的 `strcat` 函式需預先估計目標字串的長度，筆者略為修改，採用動態計算字串長度後生成所需長度的字元陣列，最後將原本的字串逐一複製過去。範例程式碼如下：

```
#include <assert.h>                       /*  1 */
#include <stdio.h>                        /*  2 */
#include <stdlib.h>                       /*  3 */
#include <string.h>                       /*  4 */

int main(void)
{                                         /*  5 */
  char s_a[] = "Hello ";                  /*  6 */
  char s_b[] = "World";                   /*  7 */
```

(continues on next page)

(continued from previous page)

```
  size_t sz_a = strlen(s_a);              /*  8 */
  size_t sz_b = strlen(s_b);              /*  9 */
  size_t sz = sz_a + sz_b + 1;            /* 10 */

  char *out = \
    malloc(sz * sizeof(char));            /* 11 */
  if (!out) {                             /* 12 */
    fprintf(
      stderr,
      "Failed to allocate"
      " a C string\n");                   /* 13 */
    return 1;                             /* 15 */
  }                                       /* 16 */

  for (size_t i = 0; i < sz_a; i++) {     /* 17 */
    out[i] = s_a[i];                      /* 18 */
  }                                       /* 19 */

  for (size_t i = 0; i < sz_b; i++) {     /* 20 */
    out[i+sz_a] = s_b[i];                 /* 21 */
  }                                       /* 22 */

  out[sz-1] = '\0';                       /* 23 */

  assert(
    0 == strcmp(out, "Hello World"));     /* 24 */

  free(out);                              /* 25 */

  return 0;                               /* 26 */
}                                         /* 27 */
```

如同上一節的例子，我們要先計算記憶體的長度，所以我們在第 10 行計算字串長度，並預先加入尾端零值的長度。

本例有兩段字串，所以拷貝字串的動作要分兩輪。在第 17 行至第 19 行間拷貝第一個字串。在第 20 行至 22 行間拷貝第二個字串。

別忘了加上尾端零值，這段動作發生在第 23 行。

在本例中，由於 `out` 是由 heap 配置記憶體，使用完要記得手動釋放。本動作在第 25 行完成。

檢查兩個 C 字串是否相等

檢查字串相等的方式為逐一檢查字元陣列的字元是否相等。可參考以下範例程式碼：

```
#include <assert.h>                 /*  1 */
#include <stdbool.h>                /*  2 */

int main(void)                      /*  3 */
{                                   /*  4 */
    char a[] = "happy";             /*  5 */
    char b[] = "happy hour";        /*  6 */

    bool is_equal = true;           /*  7 */

    char *p = a;                    /*  8 */
    char *q = b;                    /*  9 */
    while (*p && *q) {              /* 10 */
        /* Unequal character. */    /* 11 */
        if (*p != *q) {             /* 12 */
            is_equal = false;       /* 13 */
            goto END;               /* 14 */
        }                           /* 15 */

        p++; q++;                   /* 16 */
    }                               /* 17 */

    /* Unequal string length. */    /* 18 */
    if (*p != *q) {                 /* 19 */
        is_equal = false;           /* 20 */
        goto END;                   /* 21 */
    }                               /* 22 */

END:                                /* 23 */
    assert(false == is_equal);      /* 24 */

    return 0;                       /* 25 */
}                                   /* 26 */
```

我們不希望影響原本的字元陣列，所以我們在第 8 行及第 9 行分別拷貝兩字元陣列的位址。

檢查字元陣列的動作位於第 10 行至第 17 行。當字元不相等時，直接結束比較字元陣列的迴圈。

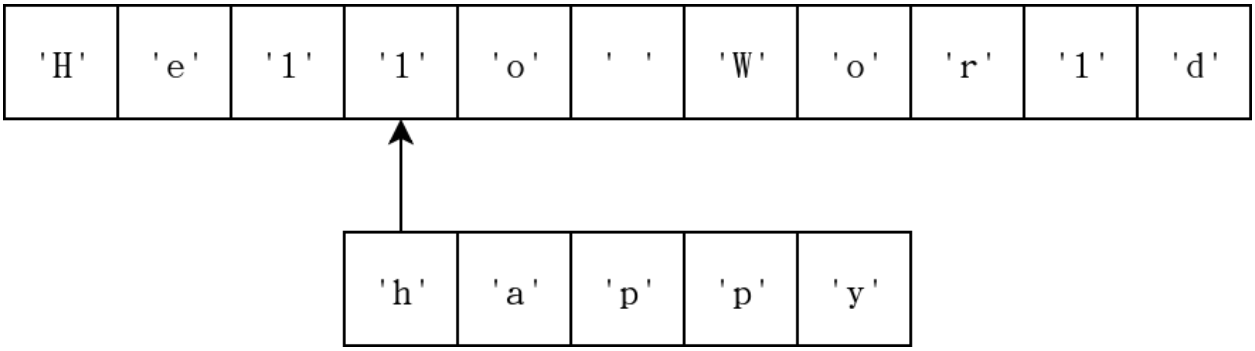
兩字元有可能部分相等，但兩者不等長。所以我們在第 19 行至第 21 行檢查兩字元陣列的尾端是否相等。

在這裡我們不直接檢查字串長度，因為這樣會多走訪一輪字串。我們先逐一檢查字元是否

相等，在尾段則檢查兩字串長度是否相等。

尋找子字串

尋找子字串的示意圖如下：



本命題的想法相當簡單，我們逐一走訪原字串，在每個位置檢查是否符合子字串。以下是參考實作：

```
#include <assert.h>                     /*  1 */
#include <stdbool.h>                    /*  2 */

int main(void)                          /*  3 */
{                                       /*  4 */
    /* Original string. */              /*  5 */
    char s[] =
        "The quick brown fox "
        "jumps over the lazy dog";      /*  6 */
    /* Substring. */                    /*  7 */
    char ss[] = "lazy";                 /*  8 */

    bool is_found = false;              /*  9 */

    char *p = s;                        /* 10 */
    while (*p) {                        /* 11 */
        bool temp = true;               /* 12 */

        char *q = p;                    /* 13 */
        char *r = ss;                   /* 14 */
        while (*q && *r) {              /* 15 */
            if (*q != *r) {             /* 16 */
                temp = false;           /* 17 */
                break;                  /* 18 */
            }                           /* 19 */
```

(continues on next page)

(continued from previous page)

```
            q++; r++;                   /* 20 */
        }                               /* 21 */

        if (!(*r)) {                    /* 22 */
            is_found = true;            /* 23 */
            break;                      /* 24 */
        }                               /* 25 */

        p++;                            /* 26 */
    }                                   /* 27 */

    assert(is_found);                   /* 28 */

    return 0;                           /* 29 */
}                                       /* 30 */
```

走訪原字串的迴圈位於第 11 行至第 27 行。在走訪時，我們在每個位置逐一比較子字串是否相符。比較子字串的迴圈位於第 15 行至第 21 行。

要注意每次走訪子字串時，都要重新拷貝字串的位址，才可以重覆走訪。

結語

在本文中，我們學習數個字串相關的基本演算法。學習這些演算法的目的不是要取代內建字串處理函式，而是要在沒有內建函式可用時，有能力自己實作新的函式。