基礎程式設計技巧(三)
字元與字串

許胖

板燒高中

July 13, 2015

大綱

大綱

大綱

#include

大綱

字元

字元

- 字元：一個位元組長，是介於 -128 到 127 的整數

字元

字元

- 字元：一個位元組長，是介於 -128 到 127 的整數
  - 保留字爲 `char`

字元

字元

- 字元：一個位元組長，是介於 -128 到 127 的整數
  - 保留字爲 `char`
- ASCII：一個編碼表，將字元的編碼對應到一個符號

字元

字元

- 字元：一個位元組長，是介於 -128 到 127 的整數
  - 保留字爲 `char`
- ASCII：一個編碼表，將字元的編碼對應到一個符號
- 表示法

字元

字元

- 字元：一個位元組長，是介於 -128 到 127 的整數
  - 保留字爲 `char`
- ASCII：一個編碼表，將字元的編碼對應到一個符號
- 表示法
  - 單引號夾著一個符號：'0'

字元

字元

- 字元：一個位元組長，是介於 -128 到 127 的整數
  - 保留字爲 `char`
- ASCII：一個編碼表，將字元的編碼對應到一個符號
- 表示法
  - 單引號夾著一個符號：'0'
  - 或者是字元的 ASCII 編碼 (8 進位)：'\60'

字元

字元

- 字元：一個位元組長，是介於 -128 到 127 的整數
  - 保留字爲 `char`
- ASCII：一個編碼表，將字元的編碼對應到一個符號
- 表示法
  - 單引號夾著一個符號：'0'
  - 或者是字元的 ASCII 編碼 (8 進位)：'\60' $\Rightarrow$ '0'

字元

字元

- 字元：一個位元組長，是介於 -128 到 127 的整數
  - 保留字爲 `char`
- ASCII：一個編碼表，將字元的編碼對應到一個符號
- 表示法
  - 單引號夾著一個符號：'0'
  - 或者是字元的 ASCII 編碼 (8 進位)：'\60' ⇒ '0'

ASCII 特性

- 某些字元是連續編碼

字元

字元

- 字元：一個位元組長，是介於 -128 到 127 的整數
  - 保留字爲 `char`
- ASCII：一個編碼表，將字元的編碼對應到一個符號
- 表示法
  - 單引號夾著一個符號：'0'
  - 或者是字元的 ASCII 編碼 (8 進位)：'\60' ⇒ '0'

ASCII 特性

- 某些字元是連續編碼
  - '0' 到 '9'

字元

字元

- 字元：一個位元組長，是介於 -128 到 127 的整數
  - 保留字爲 `char`
- ASCII：一個編碼表，將字元的編碼對應到一個符號
- 表示法
  - 單引號夾著一個符號：'0'
  - 或者是字元的 ASCII 編碼 (8 進位)：'\60' ⇒ '0'

ASCII 特性

- 某些字元是連續編碼
  - '0' 到 '9'
  - 'A' 到 'Z'

字元

字元

- 字元：一個位元組長，是介於 -128 到 127 的整數
  - 保留字爲 `char`
- ASCII：一個編碼表，將字元的編碼對應到一個符號
- 表示法
  - 單引號夾著一個符號：'0'
  - 或者是字元的 ASCII 編碼 (8 進位)：'\60' ⇒ '0'

ASCII 特性

- 某些字元是連續編碼
  - '0' 到 '9'
  - 'A' 到 'Z'
  - 'a' 到 'z'

字元

字元

- 字元：一個位元組長，是介於 -128 到 127 的整數
  - 保留字爲 `char`
- ASCII：一個編碼表，將字元的編碼對應到一個符號
- 表示法
  - 單引號夾著一個符號：'0'
  - 或者是字元的 ASCII 編碼 (8 進位)：'\60' ⇒ '0'

ASCII 特性

- 某些字元是連續編碼
  - '0' 到 '9'
  - 'A' 到 'Z'
  - 'a' 到 'z'

常見的 ASCII 編碼

字元

字元

- 字元：一個位元組長，是介於 -128 到 127 的整數
  - 保留字爲 `char`
- ASCII：一個編碼表，將字元的編碼對應到一個符號
- 表示法
  - 單引號夾著一個符號：'0'
  - 或者是字元的 ASCII 編碼 (8 進位)：'\60' ⇒ '0'

ASCII 特性

- 某些字元是連續編碼
  - '0' 到 '9'
  - 'A' 到 'Z'
  - 'a' 到 'z'

常見的 ASCII 編碼

- '\0'：0

字元

字元

- 字元：一個位元組長，是介於 -128 到 127 的整數
  - 保留字爲 `char`
- ASCII：一個編碼表，將字元的編碼對應到一個符號
- 表示法
  - 單引號夾著一個符號：`'0'`
  - 或者是字元的 ASCII 編碼 (8 進位)：`'\60'` $\Rightarrow$ `'0'`

ASCII 特性

- 某些字元是連續編碼
  - `'0'` 到 `'9'`
  - `'A'` 到 `'Z'`
  - `'a'` 到 `'z'`

常見的 ASCII 編碼

- `'\0'` : 0
- `'\n'` : 32

字元

字元

- 字元：一個位元組長，是介於 -128 到 127 的整數
  - 保留字爲 `char`
- ASCII：一個編碼表，將字元的編碼對應到一個符號
- 表示法
  - 單引號夾著一個符號：`'0'`
  - 或者是字元的 ASCII 編碼 (8 進位)：`'\60'` $\Rightarrow$ `'0'`

ASCII 特性

- 某些字元是連續編碼
  - `'0'` 到 `'9'`
  - `'A'` 到 `'Z'`
  - `'a'` 到 `'z'`

常見的 ASCII 編碼

- `'\0'` : 0
- `'\n'` : 32
- `'0'` : 48

字元

字元

- 字元：一個位元組長，是介於 -128 到 127 的整數
  - 保留字爲 `char`
- ASCII：一個編碼表，將字元的編碼對應到一個符號
- 表示法
  - 單引號夾著一個符號：`'0'`
  - 或者是字元的 ASCII 編碼 (8 進位)：`'\60'` $\Rightarrow$ `'0'`

ASCII 特性

- 某些字元是連續編碼
  - `'0'` 到 `'9'`
  - `'A'` 到 `'Z'`
  - `'a'` 到 `'z'`

常見的 ASCII 編碼

- `'\0'` : 0
- `'\n'` : 32
- `'0'` : 48
- `'A'` : 65

字元

字元

- 字元：一個位元組長，是介於 -128 到 127 的整數
  - 保留字爲 `char`
- ASCII：一個編碼表，將字元的編碼對應到一個符號
- 表示法
  - 單引號夾著一個符號：`'0'`
  - 或者是字元的 ASCII 編碼 (8 進位)：`'\60'` $\Rightarrow$ `'0'`

ASCII 特性

- 某些字元是連續編碼
  - `'0'` 到 `'9'`
  - `'A'` 到 `'Z'`
  - `'a'` 到 `'z'`

常見的 ASCII 編碼

- `'\0'` : 0
- `'\n'` : 32
- `'0'` : 48
- `'A'` : 65
- `'a'` : 97

判斷英數字

方法

利用字元連續性，例如判斷數字：

判斷英數字

方法

利用字元連續性，例如判斷數字：

```
bool isMyDigit(char ch) {
  return ('0' <= ch) && (ch <= '9');
  // 檢查是否在此區間
}
```

判斷英數字

方法

利用字元連續性，例如判斷數字：

```
bool isMyDigit(char ch) {
  return ('0' <= ch) && (ch <= '9');
  // 檢查是否在此區間
}
```

內建函數

在 <cctype> 中，判斷是否爲

判斷英數字

方法

利用字元連續性，例如判斷數字：

```
bool isMyDigit(char ch) {
  return ('0' <= ch) && (ch <= '9');
  // 檢查是否在此區間
}
```

內建函數

在 <cctype> 中，判斷是否爲

- isalnum(char)：英數字

判斷英數字

方法

利用字元連續性，例如判斷數字：

```
bool isMyDigit(char ch) {
  return ('0' <= ch) && (ch <= '9');
  // 檢查是否在此區間
}
```

內建函數

在 <cctype> 中，判斷是否爲

- isalnum(char)：英數字
- isalpha(char)：英文字母

判斷英數字

方法

利用字元連續性，例如判斷數字：

```
bool isMyDigit(char ch) {
  return ('0' <= ch) && (ch <= '9');
  // 檢查是否在此區間
}
```

內建函數

在 <cctype> 中，判斷是否爲

- isalnum(char)：英數字
- isalpha(char)：英文字母
- isdigit(char)：數字

判斷英數字

方法

利用字元連續性，例如判斷數字：

```
bool isMyDigit(char ch) {
  return ('0' <= ch) && (ch <= '9');
  // 檢查是否在此區間
}
```

內建函數

在 <cctype> 中，判斷是否爲

- isalnum(char)：英數字
- isalpha(char)：英文字母
- isdigit(char)：數字
- islower(char)：小寫字母

判斷英數字

方法

利用字元連續性，例如判斷數字：

```
bool isMyDigit(char ch) {
  return ('0' <= ch) && (ch <= '9');
  // 檢查是否在此區間
}
```

內建函數

在 <cctype> 中，判斷是否爲

- isalnum(char)：英數字
- isalpha(char)：英文字母
- isdigit(char)：數字
- islower(char)：小寫字母
- isupper(char)：大寫字母

判斷英數字

方法

利用字元連續性，例如判斷數字：

```
bool isMyDigit(char ch) {
  return ('0' <= ch) && (ch <= '9');
  // 檢查是否在此區間
}
```

內建函數

在 <cctype> 中，判斷是否爲

- isalnum(char)：英數字
- isalpha(char)：英文字母
- isdigit(char)：數字
- islower(char)：小寫字母
- isupper(char)：大寫字母
- etc.

字元轉數字

- 利用連續編碼的特性，加上 `char` 是存整數，可以將字元轉數字

字元轉數字

- 利用連續編碼的特性，加上 `char` 是存整數，可以將字元轉數字
- 常用到爛掉的技巧

字元轉數字

- 利用連續編碼的特性，加上 char 是存整數，可以將字元轉數字
- 常用到爛掉的技巧

範例：字元轉數字

```
int charToDigit(char ch) {
  return ch - '0'; // 直接減 '0'，不需特別記 48
}
```

字元轉數字

- 利用連續編碼的特性，加上 char 是存整數，可以將字元轉數字
- 常用到爛掉的技巧

範例：字元轉數字

```
int charToDigit(char ch) {
  return ch - '0'; // 直接減 '0'，不需特別記 48
}
```

註

- 同樣的道理可以套用在大小寫字母中。

字元轉數字

- 利用連續編碼的特性，加上 char 是存整數，可以將字元轉數字
- 常用到爛掉的技巧

範例：字元轉數字

```
int charToDigit(char ch) {
  return ch - '0'; // 直接減 '0'，不需特別記 48
}
```

註

- 同樣的道理可以套用在大小寫字母中。
- 要是今天要轉換的字串是 "10" 呢？這方法還會湊效嗎？

字元的輸入

輸入字元

```
char a, b;
cin >> a >> b;
cout << "X" << a << "D" << b << "D" << endl;
```

字元的輸入

輸入字元

```
char a, b;
cin >> a >> b;
cout << "X" << a << "D" << b << "D" << endl;
```

觀察

輸入「1 2」會發生什麼事？

字元的輸入

輸入字元

```
char a, b;
cin >> a >> b;
cout << "X" << a << "D" << b << "D" << endl;
```

觀察

輸入「1 2」會發生什麼事？ Ans: b 會讀到空白

字元的輸入

輸入字元

```
char a, b;
cin >> a >> b;
cout << "X" << a << "D" << b << "D" << endl;
```

觀察

輸入「1 2」會發生什麼事？ Ans: b 會讀到空白

注意

輸入字元時，會讀入空白和換行。

C++ 字串

C 與 C++ 字串

C++ 字串

- 在 <string> 中。

C 與 C++ 字串

C++ 字串

- 在 <string> 中。
- 是一個類別。

C 與 C++ 字串

C++ 字串

- 在 <string> 中。
- 是一個類別。

C 字串

- 本質上是字元陣列

C 與 C++ 字串

C++ 字串

- 在 <string> 中。
- 是一個類別。

C 字串

- 本質上是字元陣列
- 宣告時就是字元陣列

C 與 C++ 字串

C++ 字串

- 在 <string> 中。
- 是一個類別。

C 字串

- 本質上是字元陣列
- 宣告時就是字元陣列

字串宣告

C 與 C++ 字串

C++ 字串

- 在 <string> 中。
- 是一個類別。

C 字串

- 本質上是字元陣列
- 宣告時就是字元陣列

字串宣告

- `string str;`

C 與 C++ 字串

C++ 字串

- 在 <string> 中。
- 是一個類別。

C 字串

- 本質上是字元陣列
- 宣告時就是字元陣列

字串宣告

- `string str;`
- `char str[110];`

C 與 C++ 字串

C++ 字串

- 在 <string> 中。
- 是一個類別。

C 字串

- 本質上是字元陣列
- 宣告時就是字元陣列

字串宣告

- `string str;`
- `char str[110];`

常數

字串是由兩個雙引號夾著一堆字元："XD"

C 字串儲存狀況

C 字串

```
char str[10] = "bird";
```

C 字串儲存狀況

C 字串

```
char str[10] = "bird";
```

實際儲存狀況

'b'	'i'	'r'	'd'	'\0'	未知	未知	未知	未知	未知

C 字串儲存狀況

C 字串

```
char str[10] = "bird";
```

實際儲存狀況

'b'	'i'	'r'	'd'	'\0'	未知	未知	未知	未知	未知

注意

'\0' 代表字串的結尾，佔一個字元，因此在使用 C 字串時要多注意空間上的問題。

字串操作

字串操作

字串操作

字串操作

- 取長度

字串操作

字串操作

- 取長度
- 比較字串

字串操作

字串操作

- 取長度
- 比較字串
- 複製字串

字串操作

字串操作

- 取長度
- 比較字串
- 複製字串
- 串接字串

字串操作

字串操作

- 取長度
- 比較字串
- 複製字串
- 串接字串
- 其他操作

字串操作：取長度

C++ 字串

- `str.size()`

字串操作：取長度

C++ 字串

- `str.size()`

C 字串

字串操作：取長度

C++ 字串

- `str.size()`

C 字串

- 在 `<cstring>` 中

字串操作：取長度

C++ 字串

- `str.size()`

C 字串

- 在 `<cstring>` 中
- `strlen(const char *)`

字串操作：取長度

C++ 字串

- `str.size()`

C 字串

- 在 `<cstring>` 中
- `strlen(const char *)`

C++ 字串用法

```
for (int i = 0; i < str.size(); i++) {}
```

字串操作：取長度

C++ 字串

- str.size()

C 字串

- 在 <cstring> 中
- strlen(const char *)

C++ 字串用法

```
for (int i = 0; i < str.size(); i++) {}
```

C 字串用法

```
for (int i = 0; i < strlen(str); i++) {}
```

字串操作：取長度

C++ 字串

- `str.size()`

C 字串

- 在 <cstring> 中
- `strlen(const char *)`

C++ 字串用法

```
for (int i = 0; i < str.size(); i++) {}
```

C 字串用法

```
for (int i = 0; i < strlen(str); i++) {}
```

盡量少這樣用！

原理

原理

原理

原理

- strlen() 是函數

原理

原理

- strlen() 是函數
  - strlen() 計算長度的方法，就是從傳入指標開始數，數到 '\0' 字元爲止！

原理

原理

- strlen() 是函數
  - strlen() 計算長度的方法，就是從傳入指標開始數，數到 '\0' 字元爲止！
  - 在沒優化下，每次 for 迴圈判斷時都會執行 strlen() 一次！

原理

原理

- strlen() 是函數
  - strlen() 計算長度的方法，就是從傳入指標開始數，數到 '\0' 字元爲止！
  - 在沒優化下，每次 for 迴圈判斷時都會執行 strlen() 一次！
  - 在字串長度不變下，這個方法會讓程式變慢

原理

原理

- strlen() 是函數
  - strlen() 計算長度的方法，就是從傳入指標開始數，數到 '\0' 字元爲止！
  - 在沒優化下，每次 for 迴圈判斷時都會執行 strlen() 一次！
  - 在字串長度不變下，這個方法會讓程式變慢
- str.size() 是取值

原理

原理

- strlen() 是函數
  - strlen() 計算長度的方法，就是從傳入指標開始數，數到 '\0' 字元爲止！
  - 在沒優化下，每次 for 迴圈判斷時都會執行 strlen() 一次！
  - 在字串長度不變下，這個方法會讓程式變慢
- str.size() 是取值
  - 字串本身就會把長度算好存起來，因此沒這問題。

原理

原理

- strlen() 是函數
  - strlen() 計算長度的方法，就是從傳入指標開始數，數到 '\0' 字元爲止！
  - 在沒優化下，每次 for 迴圈判斷時都會執行 strlen() 一次！
  - 在字串長度不變下，這個方法會讓程式變慢
- str.size() 是取值
  - 字串本身就會把長度算好存起來，因此沒這問題。

改良

宣告變數儲存長度。

原理

原理

- strlen() 是函數
  - strlen() 計算長度的方法，就是從傳入指標開始數，數到 '\0' 字元爲止！
  - 在沒優化下，每次 for 迴圈判斷時都會執行 strlen() 一次！
  - 在字串長度不變下，這個方法會讓程式變慢
- str.size() 是取值
  - 字串本身就會把長度算好存起來，因此沒這問題。

改良

宣告變數儲存長度。

```
int len = strlen(str);
for (int i = 0; i < len; i++) {}
```

字串操作：比較字串

C++ 字串

- `strA == strB`

字串操作：比較字串

C++ 字串

- `strA == strB`

C 字串

字串操作：比較字串

C++ 字串

- `strA == strB`

C 字串

- 在 `<cstring>` 中

字串操作：比較字串

C++ 字串

- `strA == strB`

C 字串

- 在 `<cstring>` 中
- `strcmp(strA, strB)`

字串操作：比較字串

C++ 字串

- strA == strB

C 字串

- 在 <cstring> 中
- strcmp(strA, strB)

註

```
strcmp(const char *, const char *);
```

字串操作：比較字串

C++ 字串

- strA == strB

C 字串

- 在 <cstring> 中
- strcmp(strA, strB)

註

```
strcmp(const char *, const char *);
```

原理

比較 strA 和 strB 每個字元，直到某一邊先碰到 '\0' 爲止。

差異

C++ 字串

- strA == strB

C 字串

- 在 <cstring> 中
- strcmp(strA, strB)

差異

C++ 字串

- `strA == strB`

C 字串

- 在 `<cstring>` 中
- `strcmp(strA, strB)`

回傳值

差異

C++ 字串

- `strA == strB`

C 字串

- 在 `<cstring>` 中
- `strcmp(strA, strB)`

回傳值

- C++ 字串如果相等回傳 `true`，否則回傳 `false`。

差異

C++ 字串

- `strA == strB`

C 字串

- 在 <cstring> 中
- `strcmp(strA, strB)`

回傳值

- C++ 字串如果相等回傳 `true`，否則回傳 `false`。
- `strcmp()` 函數原理類似兩字串相減

差異

C++ 字串

- `strA == strB`

C 字串

- 在 <cstring> 中
- `strcmp(strA, strB)`

回傳值

- C++ 字串如果相等回傳 `true`，否則回傳 `false`。
- `strcmp()` 函數原理類似兩字串相減
  1. 相同回傳 0

差異

C++ 字串

- strA == strB

C 字串

- 在 <cstring> 中
- strcmp(strA, strB)

回傳值

- C++ 字串如果相等回傳 true，否則回傳 false。
- strcmp() 函數原理類似兩字串相減
  1. 相同回傳 0
  2. strA 的字典順序比 strB 大，回傳正值 (通常是 1)

差異

C++ 字串

- `strA == strB`

C 字串

- 在 <cstring> 中
- `strcmp(strA, strB)`

回傳值

- C++ 字串如果相等回傳 `true`，否則回傳 `false`。
- `strcmp()` 函數原理類似兩字串相減
  1. 相同回傳 0
  2. strA 的字典順序比 strB 大，回傳正值 (通常是 1)
  3. strA 的字典順序比 strB 小，回傳負值 (通常是 -1)

字串操作：複製字串

C++ 字串

- `dest = src`

字串操作：複製字串

C++ 字串

- `dest = src`

C 字串

字串操作：複製字串

C++ 字串

- `dest = src`

C 字串

- 在 `<cstring>` 中

字串操作：複製字串

C++ 字串

- `dest = src`

C 字串

- 在 `<cstring>` 中
- `strcpy(dest, src)`

字串操作：複製字串

C++ 字串

- dest = src

C 字串

- 在 <cstring> 中
- strcpy(dest, src)

註

```
strcpy(char *dest, const char *src);
strncpy(char *dest, const char *src, size_t n);
```

字串操作：複製字串

C++ 字串

- dest = src

C 字串

- 在 <cstring> 中
- strcpy(dest, src)

註

```
strcpy(char *dest, const char *src);
strncpy(char *dest, const char *src, size_t n);
```

原理

- 將每個 src 的字元一一複製到 dest 中，直到碰到 '\0' 爲止。

字串操作：複製字串

註

```
strcpy(char *dest, const char *src);
strncpy(char *dest, const char *src, size_t n);
```

字串操作：複製字串

註

```
strcpy(char *dest, const char *src);
strncpy(char *dest, const char *src, size_t n);
```

注意

字串操作：複製字串

註

```
strcpy(char *dest, const char *src);
strncpy(char *dest, const char *src, size_t n);
```

注意

- 因爲複製字串時，會一直複製直到碰到 '\0' 爲止。

字串操作：複製字串

註

```
strcpy(char *dest, const char *src);
strncpy(char *dest, const char *src, size_t n);
```

注意

- 因爲複製字串時，會一直複製直到碰到 '\0' 爲止。
- 當 dest 長度太短，strcpy() 就會寫出 dest，造成 Runtime Error。

字串操作：複製字串

註

```
strcpy(char *dest, const char *src);
strncpy(char *dest, const char *src, size_t n);
```

注意

- 因爲複製字串時，會一直複製直到碰到 '\0' 爲止。
- 當 dest 長度太短，strcpy() 就會寫出 dest，造成 Runtime Error。
- 避免這種情況可以使用 strncpy()

字串操作：複製字串

註

```
strcpy(char *dest, const char *src);
strncpy(char *dest, const char *src, size_t n);
```

注意

- 因爲複製字串時，會一直複製直到碰到 '\0' 爲止。
- 當 dest 長度太短，strcpy() 就會寫出 dest，造成 Runtime Error。
- 避免這種情況可以使用 strncpy()
  - 第三個參數代表複製的字元數。

字串操作：複製字串

註

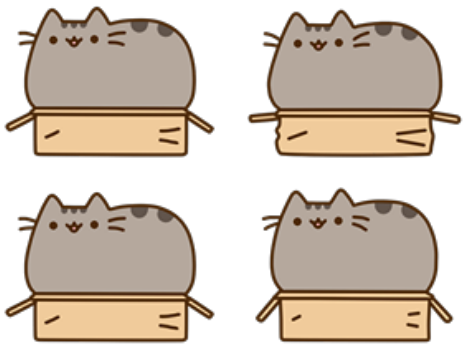
```
strcpy(char *dest, const char *src);
strncpy(char *dest, const char *src, size_t n);
```

注意

- 因爲複製字串時，會一直複製直到碰到 '\0' 爲止。
- 當 dest 長度太短，strcpy() 就會寫出 dest，造成 Runtime Error。
- 避免這種情況可以使用 strncpy()
  - 第三個參數代表複製的字元數。
  - 注意！使用 strncpy() 在 dest 後面不會補 '\0'， 因此要記得補。

字串操作：串接字串

Figure: 4 隻 pusheen



字串操作：串接字串

Figure: 4 隻 pusheen

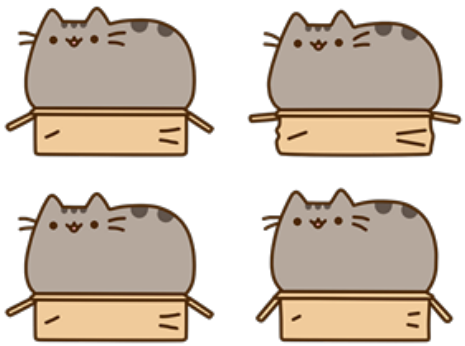
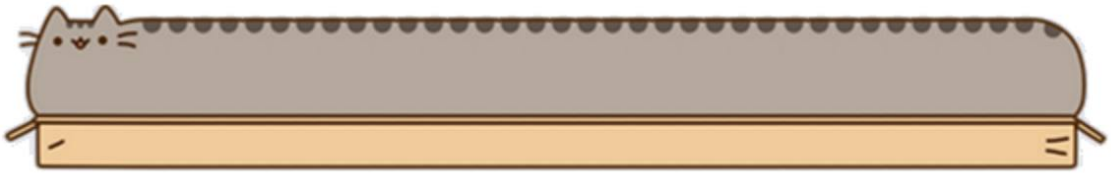


Figure: strcat(pusheen)



字串操作：串接字串

C++ 字串

- `dest += src`

字串操作：串接字串

C++ 字串

- `dest += src`

C 字串

字串操作：串接字串

C++ 字串

- `dest += src`

C 字串

- 在 `<cstring>` 中

字串操作：串接字串

C++ 字串

- `dest += src`

C 字串

- 在 `<cstring>` 中
- `strcat(dest, src)`

字串操作：串接字串

C++ 字串

- `dest += src`

C 字串

- 在 <cstring> 中
- `strcat(dest, src)`

註

```
strcat(char *dest, const char *src);
strncat(char *dest, const char *src, size_t n);
```

字串操作：串接字串

C++ 字串

- dest += src

C 字串

- 在 <cstring> 中
- strcat(dest, src)

註

```
strcat(char *dest, const char *src);
strncat(char *dest, const char *src, size_t n);
```

原理

- 從 dest 第一個 '\0' 開始串接，直到碰到 src 的 '\0' 爲止，與 strcpy 有同樣的問題。

其他字串操作

其他字串操作

其他字串操作

其他字串操作

- `strstr()`

其他字串操作

其他字串操作

- `strstr()`
- `strchr()`

其他字串操作

其他字串操作

- `strstr()`
- `strchr()`
- `strrchr()`

其他字串操作

其他字串操作

- strstr()
- strchr()
- strrchr()
- strtok()

其他字串操作

其他字串操作

- `strstr()`
- `strchr()`
- `strrchr()`
- `strtok()`

說明

- 概念較複雜，或者用自己的方法處理就好。

其他字串操作

其他字串操作

- `strstr()`
- `strchr()`
- `strrchr()`
- `strtok()`

說明

- 概念較複雜，或者用自己的方法處理就好。
- 讀者有興趣可以自己查資料。

字串小結

Table: 常見字串操作

	C++ 字串	C 字串
取長度	`str.size()`	`strlen(str)`

字串小結

Table: 常見字串操作

	C++ 字串	C 字串
取長度	`str.size()`	`strlen(str)`
比較字串	`strA == strB`	`strcmp(strA, strB)`

字串小結

Table: 常見字串操作

	C++ 字串	C 字串
取長度	`str.size()`	`strlen(str)`
比較字串	`strA == strB`	`strcmp(strA, strB)`
複製字串	`dest = src`	`strcpy(dest, src)`

字串小結

Table: 常見字串操作

	C++ 字串	C 字串
取長度	`str.size()`	`strlen(str)`
比較字串	`strA == strB`	`strcmp(strA, strB)`
複製字串	`dest = src`	`strcpy(dest, src)`
串接字串	`dest += src`	`strcat(dest, src)`

字串小結

Table: 常見字串操作

	C++ 字串	C 字串
取長度	`str.size()`	`strlen(str)`
比較字串	`strA == strB`	`strcmp(strA, strB)`
複製字串	`dest = src`	`strcpy(dest, src)`
串接字串	`dest += src`	`strcat(dest, src)`

差異

字串小結

Table: 常見字串操作

	C++ 字串	C 字串
取長度	`str.size()`	`strlen(str)`
比較字串	`strA == strB`	`strcmp(strA, strB)`
複製字串	`dest = src`	`strcpy(dest, src)`
串接字串	`dest += src`	`strcat(dest, src)`

差異

- C++ 字串是類別，C 字串是字元陣列

字串小結

Table: 常見字串操作

	C++ 字串	C 字串
取長度	`str.size()`	`strlen(str)`
比較字串	`strA == strB`	`strcmp(strA, strB)`
複製字串	`dest = src`	`strcpy(dest, src)`
串接字串	`dest += src`	`strcat(dest, src)`

差異

- C++ 字串是類別，C 字串是字元陣列
- C++ 字串操作多是運算子或成員函數，C 字串是函數

字串小結

Table: 常見字串操作

	C++ 字串	C 字串
取長度	`str.size()`	`strlen(str)`
比較字串	`strA == strB`	`strcmp(strA, strB)`
複製字串	`dest = src`	`strcpy(dest, src)`
串接字串	`dest += src`	`strcat(dest, src)`

差異

- C++ 字串是類別，C 字串是字元陣列
- C++ 字串操作多是運算子或成員函數，C 字串是函數
- C 字串處理速度較 C++ 來得快，但要注意會不會超出陣列範圍以及補 '\0'

C++ 字串轉 C 字串

成員函數

- 在 <string> 中

C++ 字串轉 C 字串

成員函數

- 在 <string> 中
- str.c_str()

C++ 字串轉 C 字串

成員函數

- 在 <string> 中
- str.c_str()

範例

```
string str = "bird";
cout << str.c_str() << endl;
```

C++ 字串轉 C 字串

成員函數

- 在 <string> 中
- str.c_str()

範例

```
string str = "bird";
cout << str.c_str() << endl;
```

用途

- C 語言的函數大多只能用 C 字串來做，因此我們可以藉由這個函數把 string 轉成 C 字串。

C 字串轉 C++ 字串

範例

```
char strA[110] = "bird";
string strB = strA; // 直接丟進去
cout << strB << endl;
cout << strB.size() << endl;
```

字串練習

練習一

```
char str1[110] = "abcdefghijklmnopqrstuvwxyz";
```

長度爲何？

字串練習

練習一

```
char str1[110] = "abcdefghijklmnopqrstuvwxyz";
```

長度爲何？用 sizeof 和 strlen 有何不同呢？

字串練習

練習一

```
char str1[110] = "abcdefghijklmnopqrstuvwxyz";
```

長度爲何？用 sizeof 和 strlen 有何不同呢？做完上面的問題後，試著思考下面的語句又和上面的問題有什麼差異呢？

字串練習

練習一

```
char str1[110] = "abcdefghijklmnopqrstuvwxyz";
```

長度爲何？用 sizeof 和 strlen 有何不同呢？做完上面的問題後，試著思考下面的語句又和上面的問題有什麼差異呢？

- strlen("abcdefghijklmnopqrstuvwxyz");

字串練習

練習一

```
char str1[110] = "abcdefghijklmnopqrstuvwxyz";
```

長度爲何？用 sizeof 和 strlen 有何不同呢？做完上面的問題後，試著思考下面的語句又和上面的問題有什麼差異呢？

- strlen("abcdefghijklmnopqrstuvwxyz");
- sizeof("abcdefghijklmnopqrstuvwxyz");

字串練習

練習一

```
char str1[110] = "abcdefghijklmnopqrstuvwxyz";
```

長度爲何？用 sizeof 和 strlen 有何不同呢？做完上面的問題後，試著思考下面的語句又和上面的問題有什麼差異呢？

- strlen("abcdefghijklmnopqrstuvwxyz");
- sizeof("abcdefghijklmnopqrstuvwxyz");

提示

「記憶體」和「字串」之間的差別與關係。

字串練習

練習二

```
char str2[110] = "bird";
char str3[4];
```

字串練習

練習二

```
char str2[110] = "bird";
char str3[4];
```

- 我們可以用 strcpy(str3, str2); 嘛？

字串練習

練習二

```
char str2[110] = "bird";
char str3[4];
```

- 我們可以用 strcpy(str3, str2); 嘛？
- 如果不行，若我們使用 strncpy(str3, str2, 4); 來避免超出記憶體呢？試著 cout 出來觀察？

字串練習

練習二

```
char str2[110] = "bird";
char str3[4];
```

- 我們可以用 strcpy(str3, str2); 嘛？
- 如果不行，若我們使用 strncpy(str3, str2, 4); 來避免超出記憶體呢？試著 cout 出來觀察？

提示

strcpy 的用法。

字串練習

練習三

- ```
  char str4[110] = "cat\0bird";
  ```

  請問這個字串的長度爲何？
  ```

字串練習

練習三

- ```
  char str4[110] = "cat\0bird";
  ```

  請問這個字串的長度爲何？若是 `strlen(str4 + 4)` 的回傳值爲何？
  ```

字串練習

練習三

- ```
  char str4[110] = "cat\0bird";
  ```

  請問這個字串的長度爲何？若是 `strlen(str4 + 4)` 的回傳值爲何？`strlen(str4 + 8)` 呢？
  ```

字串練習

練習三

- `char str4[110] = "cat\0bird";`

  請問這個字串的長度爲何？若是 `strlen(str4 + 4)` 的回傳值爲何？`strlen(str4 + 8)` 呢？
- 今天我們使用 `strcpy(str4, "dog")`，會得到什麼結果？

字串練習

練習三

- ```
  char str4[110] = "cat\0bird";
  ```

  請問這個字串的長度爲何？若是 `strlen(str4 + 4)` 的回傳值爲何？`strlen(str4 + 8)` 呢？
- 今天我們使用 `strcpy(str4, "dog")`，會得到什麼結果？此時印出 `str4` 和 `str4 + 4` 會有什麼反應呢？
  ```

字串練習

練習三

- `char str4[110] = "cat\0bird";`

  請問這個字串的長度爲何？若是 `strlen(str4 + 4)` 的回傳值爲何？`strlen(str4 + 8)` 呢？
- 今天我們使用 `strcpy(str4, "dog")`，會得到什麼結果？此時印出 `str4` 和 `str4 + 4` 會有什麼反應呢？
- 如果我們對原先的 `str4` 執行 `strcat(str4, "dog")`，有得到你預期的結果嘛？

字串練習

練習三

- `char str4[110] = "cat\0bird";`

  請問這個字串的長度爲何？若是 `strlen(str4 + 4)` 的回傳值爲何？`strlen(str4 + 8)` 呢？
- 今天我們使用 `strcpy(str4, "dog")`，會得到什麼結果？此時印出 `str4` 和 `str4 + 4` 會有什麼反應呢？
- 如果我們對原先的 `str4` 執行 `strcat(str4, "dog")`，有得到你預期的結果嘛？
- 要是 `strcat(str4 + 4, "dog")` 呢？

字串練習

練習三

- `char str4[110] = "cat\0bird";`

  請問這個字串的長度爲何？若是 `strlen(str4 + 4)` 的回傳值爲何？`strlen(str4 + 8)` 呢？
- 今天我們使用 `strcpy(str4, "dog")`，會得到什麼結果？此時印出 `str4` 和 `str4 + 4` 會有什麼反應呢？
- 如果我們對原先的 `str4` 執行 `strcat(str4, "dog")`，有得到你預期的結果嘛？
- 要是 `strcat(str4 + 4, "dog")` 呢？

提示

'\0' 的用途和指標的用法。

大綱

大綱

練習

ZeroJudge a013: 羅馬數字

給你兩個羅馬數字，做相減後輸出羅馬數字。

練習

ZeroJudge a013: 羅馬數字

給你兩個羅馬數字，做相減後輸出羅馬數字。

ZeroJudge d251: 94北縣賽-3-羅馬數字 (Roman)

給你羅馬數字表示的時間，輸出加上 7 小時 30 分時差後的羅馬數字時刻。

練習

ZeroJudge d369: 1. 羅馬數字 / UVa 11616: Roman Numerals

給你羅馬數字，轉換爲阿拉伯數字；給你阿拉伯數字，轉換爲羅馬數字。

練習

ZeroJudge d369: 1. 羅馬數字 / UVa 11616: Roman Numerals

給你羅馬數字，轉換爲阿拉伯數字；給你阿拉伯數字，轉換爲羅馬數字。

UVa 344 / Ruby 344: Roman Digititis

給你 n，輸出 n 以內需要用到多少個羅馬數字符號？

大綱