函式：參數傳遞、變數生命週期

資訊之芽語法班 2015 suhorng

參數傳遞：陣列

- 傳遞陣列的語法：在參數寫

```
type arrayName[][dim2][dim3]...[dimN]
```

注意**沒有** **`dim1`**

- 範例

```
void printArray (int length, int arr[]) {
  //                         ^^^^^^^^^
  std::cout << arr[0];
  for (int i = 1; i < length; i = i+1) {
    std::cout << " " << arr[i];
  }
  std::cout << "\n";

}
```

參數傳遞：陣列

- 傳遞陣列的語法：在參數寫

```
type arrayName[][dim2][dim3]...[dimN]
```

注意**沒有** **`dim1`**

- 範例

```
void printMatrix(int height, int mat[][50]) {
  //                         ^^^^^^^^^^^^^
  for (int i = 0; i < height; i = i+1) {
    for (int j = 0; j < 50; j = j+1) {
      std::cout << mat[i][j] << " ";
    }
    std::cout << "\n";
  }

}
```

參數傳遞：陣列

- 說明
  - 傳遞陣列時，陣列的第一個維度省略不寫（即使寫了，電腦也**不會檢查**是否正確）
  - 函式裡面**不知道陣列的第一個維度**，只知道第二、三、四、.....個維度
    - ⇒ 所以要把陣列的長度也用參數傳進去
- 在函式裡面改陣列，等同修改外面的陣列
- **不可以**回傳(區域)陣列

參數傳遞：陣列

- 範例：~~區間覆蓋~~

```
void increase_by(int arr[], int begin, int end, int by) {
  for (int i = begin; i != end; i = i+1)
    arr[i] = arr[i] + by;
}

int main() {
  int line[100];

  for (int i = 0; i < 100; i = i+1)
    line[i] = 0;

  increase_by(line, 5, 10, +3); // line[5..9]  增加 3
  increase_by(line, 2, 7, -5);  // line[2..6]  減少 5
  increase_by(line, 9, 30, +7); // interval[9..29] 增加 7
}
```

參數傳遞：陣列

- 範例：矩陣相乘

```
// 20*20 的矩陣相乘
// C := A * B
void multiply(int A[][20], int B[][20], int C[][20]) {
  for (int i = 0; i < 20; i = i+1)
    for (int j = 0; j < 20; j = j+1)
      C[i][j] = 0;

  for (int i = 0; i < 20; i = i+1) {
    for (int j = 0; j < 20; j = j+1) {
      for (int k = 0; k < 20; k = k+1) {
        C[i][j] = C[i][j] + A[i][k]*B[k][j];
      }
    }
  }
}
```

相互配合的函數

- 陣列裡面存著隱藏的資料，有一些函式操作這個陣列

```
// 初始化 block，回傳 blockNum
int initBlocks(int blocks[][2]);

// 是否在任何一個 block 中。若是，回傳索引值
int inAny(int blockNum, int blocks[][2], int y, int x);

// 回傳新的 blockNum
int remove(int blockNum, int blocks[][2], int idx);
```

- 外面的人只能呼叫這些函式，不能改陣列

```
int blockNum, blocks[100][2]; // 存 (y,x) 座標

blockNum = initBlocks(blocks);
int found = inAny(blockNum, blocks, 5, 3);
if (found >= 0)
  blockNum = remove(blockNum, blocks, found);
```

參數傳遞怎麼 work？

- 大家來找碴

```
void swap(int a, int b) {
  int c = a;
  a = b;
  b = c;
  // 執行完後 a 跟 b 的內容互換
}

int main() {
  int x = 9, y = 23;
  swap(x, y);
  // 成功？
}
```

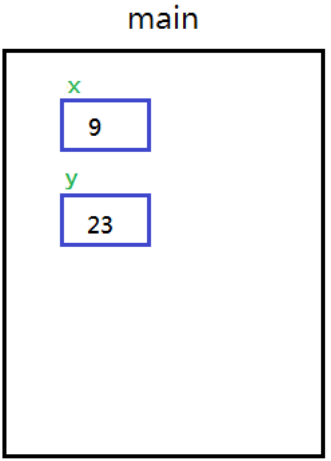
參數傳遞怎麼 work？

- **打比方. `x` 跟 `y` 從頭到尾沒動到**

```
void swap(int a, int b) {
  int c = a;
  a = b;
  b = c;
  // 執行完後 a 跟 b 的內容互換
}

int main() {
  int x = 9, y = 23;

      // swap(x, y)
          int a = x, b = y;
          int c = a;
          a = b;
          b = c;

}
```

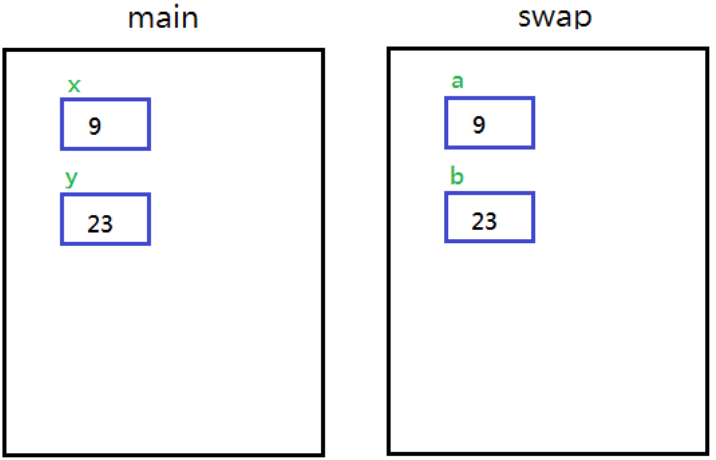
參數傳遞怎麼 work？

```
swap(x, y);
```



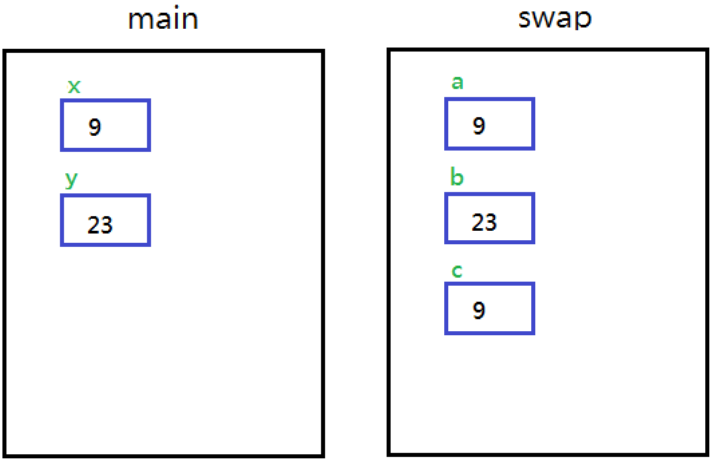
參數傳遞怎麼 work？

```
void swap(int a, int b) {
```



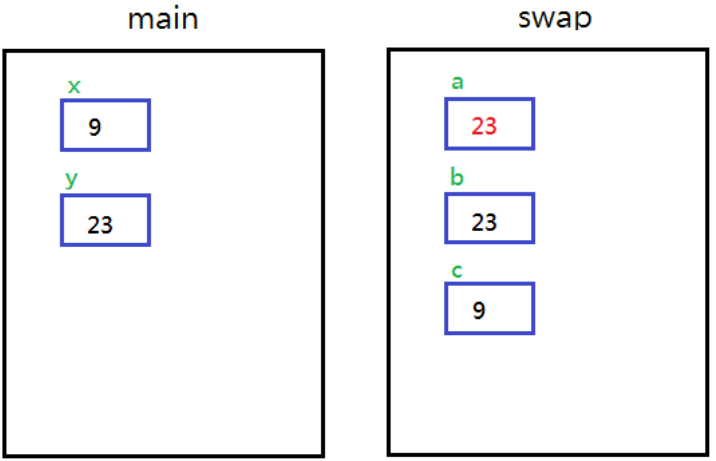
參數傳遞怎麼 work？

```
int c = a;
```



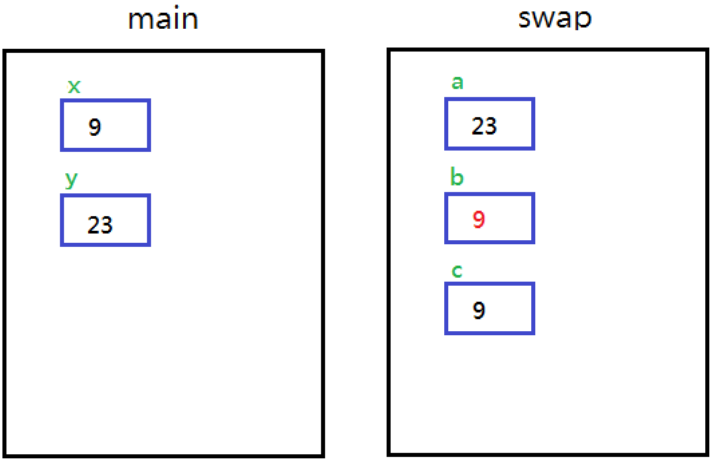
參數傳遞怎麼 work？

```
a = b;
```



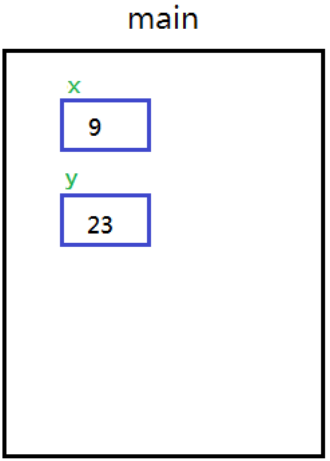
參數傳遞怎麼 work？

```
b = c;
```



參數傳遞怎麼 work？

```
}
```



參數傳遞怎麼 work？

- 特殊情況：陣列

```
void setCdr(int a[], int v) {
  a[1] = v;
}

int main() {
  int cons[2] = {1,2};

  // cons 是 {1, 2}
  setCdr(cons, -1);
  // cons 是 {1, -1}

}
```

參數傳遞怎麼 work？(**錯誤比喻**)

- 可以打比方嗎？

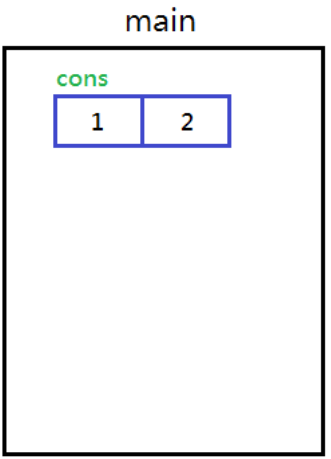
```
void setCdr(int a[], int v) {
  a[1] = v;
}

int main() {
  int cons[2] = {1,2};

  // cons 是 {1, 2}
      // setCdr(cons, -1)
            int a[] = cons, v = -1;
            a[1] = v;

}
```

- **沒有** `int a[] = cons` 這種寫法（不知道什麼意思）

參數傳遞怎麼 work？

- 更改前

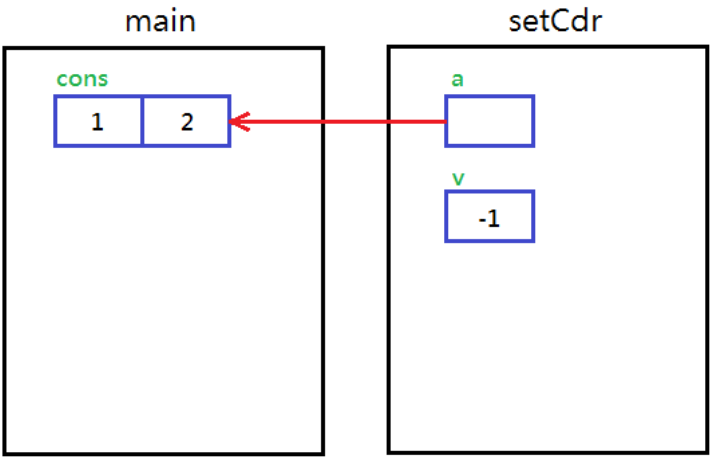
```
setCdr(cons, -1);
```



參數傳遞怎麼 work？

- 更改前

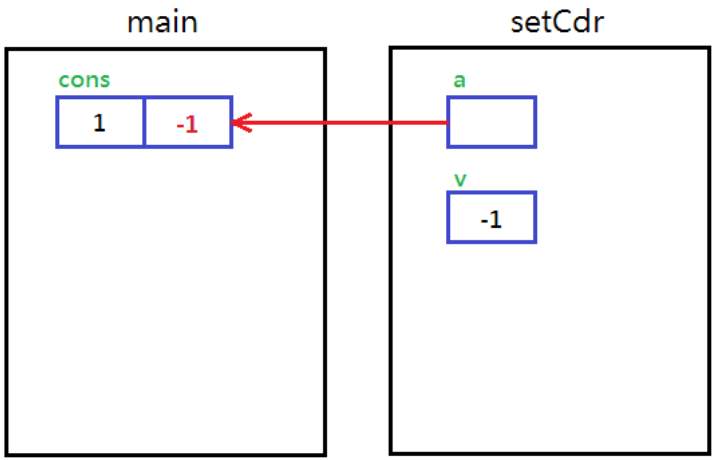
```
void setCdr(int a[], int v) {
```



參數傳遞怎麼 work？

- 更改後

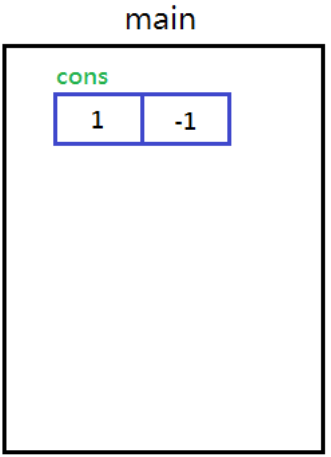
```
a[1] = v;
```



參數傳遞怎麼 work？

- 更改後；`a` 是個**指標**，裡面存著 `cons` 陣列的**起始位址**

```
}
```

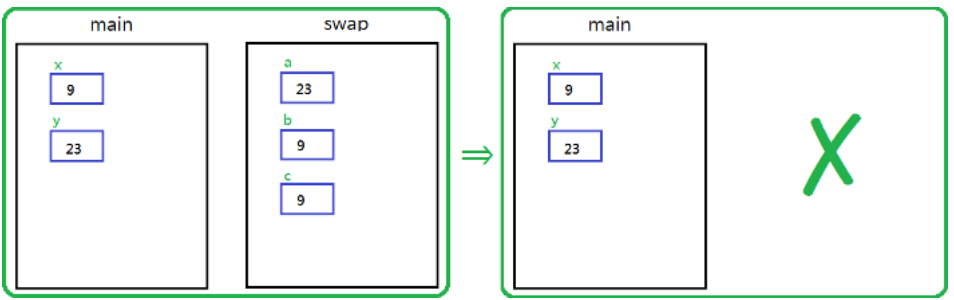


區域變數、參數的生命週期

- 執行時，參數、區域變數生命週期只在函式內

```
void swap(int a, int b) {
  int c = a;
  a = b;
  b = c;
}
```

  - 剛才圖中的 `a`, `b`, `c`

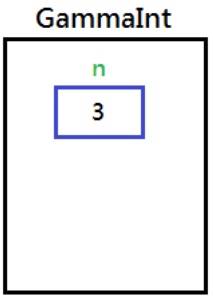


區域變數、參數的生命週期

- 自己呼叫自己：不同次的參數、區域變數相互獨立

```
int GammaInt(int n) {
  if (n == 1) return 1;
  n = n-1;
  return n * GammaInt(n);
}
```

- 執行過程 `GammaInt(3)`

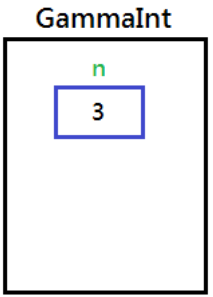


區域變數、參數的生命週期

- 自己呼叫自己：不同次的參數、區域變數相互獨立

```
int GammaInt(int n) {
  if (n == 1) return 1;
  n = n-1;
  return n * GammaInt(n);
}
```

- 執行過程 `int GammaInt(int n) {`

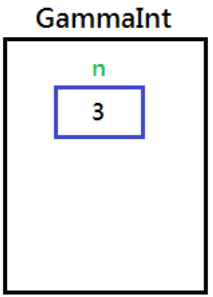


區域變數、參數的生命週期

- 自己呼叫自己：不同次的參數、區域變數相互獨立

```
int GammaInt(int n) {
  if (n == 1) return 1;
  n = n-1;
  return n * GammaInt(n);
}
```

- 執行過程 `if (n == 1) return 1;`

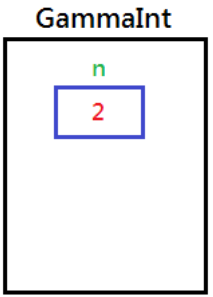


區域變數、參數的生命週期

- 自己呼叫自己：不同次的參數、區域變數相互獨立

```
int GammaInt(int n) {
  if (n == 1) return 1;
  n = n-1;
  return n * GammaInt(n);
}
```

- 執行過程 `n = n-1`

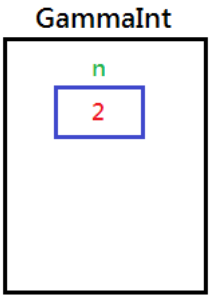


區域變數、參數的生命週期

- 自己呼叫自己：不同次的參數、區域變數相互獨立

```
int GammaInt(int n) {
  if (n == 1) return 1;
  n = n-1;
  return n * GammaInt(n);
}
```

- 執行過程 `return n * GammaInt(n);`

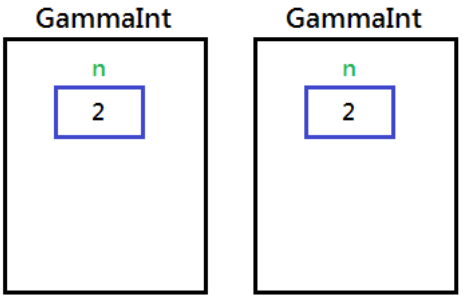


區域變數、參數的生命週期

- 自己呼叫自己：不同次的參數、區域變數相互獨立

```
int GammaInt(int n) {
  if (n == 1) return 1;
  n = n-1;
  return n * GammaInt(n);
}
```

- 執行過程 `int GammaInt(int n) {`

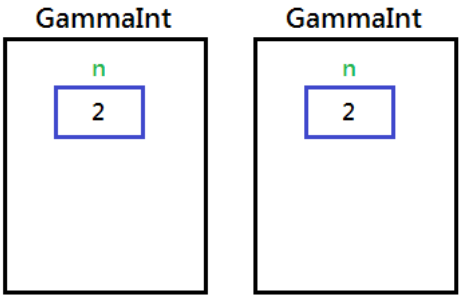


區域變數、參數的生命週期

- 自己呼叫自己：不同次的參數、區域變數相互獨立

```
int GammaInt(int n) {
  if (n == 1) return 1;
  n = n-1;
  return n * GammaInt(n);
}
```

- 執行過程 `if (n == 1) return 1;`

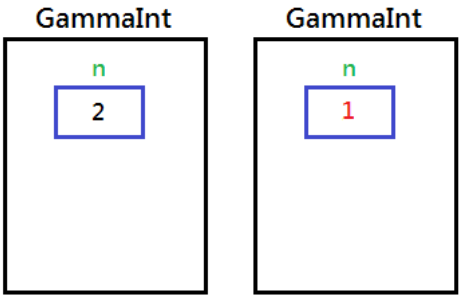


區域變數、參數的生命週期

- 自己呼叫自己：不同次的參數、區域變數相互獨立

```
int GammaInt(int n) {
  if (n == 1) return 1;
  n = n-1;
  return n * GammaInt(n);
}
```

- 執行過程 `n = n-1;`

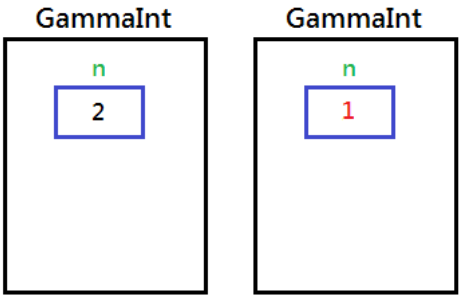


區域變數、參數的生命週期

- 自己呼叫自己：不同次的參數、區域變數相互獨立

```
int GammaInt(int n) {
  if (n == 1) return 1;
  n = n-1;
  return n * GammaInt(n);
}
```

- 執行過程 `return n * GammaInt(n);`

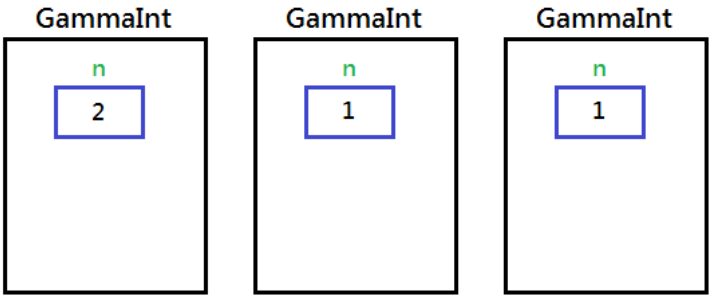


區域變數、參數的生命週期

- 自己呼叫自己：不同次的參數、區域變數相互獨立

```
int GammaInt(int n) {
  if (n == 1) return 1;
  n = n-1;
  return n * GammaInt(n);
}
```

- 執行過程 `int GammaInt(int n) {`

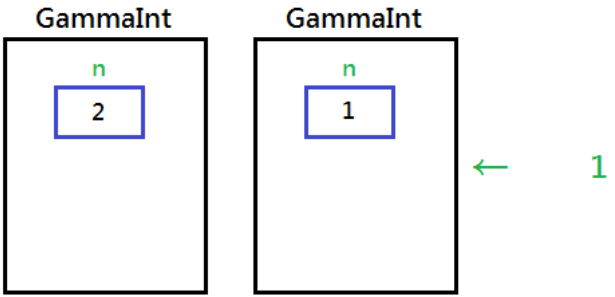


區域變數、參數的生命週期

- 自己呼叫自己：不同次的參數、區域變數相互獨立

```
int GammaInt(int n) {
  if (n == 1) return 1;
  n = n-1;
  return n * GammaInt(n);
}
```

- 執行過程 `if (n == 1) return 1;`

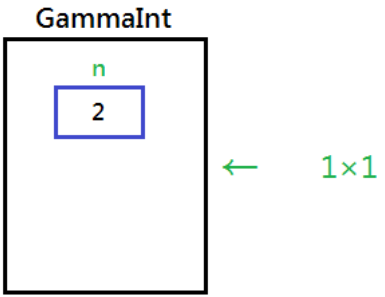


區域變數、參數的生命週期

- 自己呼叫自己：不同次的參數、區域變數相互獨立

```
int GammaInt(int n) {
  if (n == 1) return 1;
  n = n-1;
  return n * GammaInt(n);
}
```

- 執行過程 }

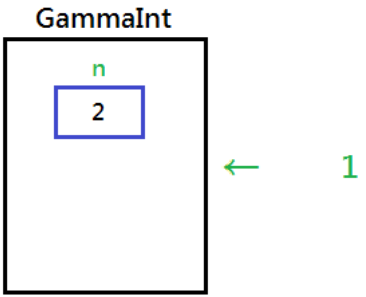


區域變數、參數的生命週期

- 自己呼叫自己：不同次的參數、區域變數相互獨立

```
int GammaInt(int n) {
  if (n == 1) return 1;
  n = n-1;
  return n * GammaInt(n);
}
```

- 執行過程 }



區域變數、參數的生命週期

- 自己呼叫自己：不同次的參數、區域變數相互獨立

```
int GammaInt(int n) {
  if (n == 1) return 1;
  n = n-1;
  return n * GammaInt(n);
}
```

- 執行過程 `}`

← 2×1

區域變數、參數的生命週期

- 自己呼叫自己：不同次的參數、區域變數相互獨立

```
int GammaInt(int n) {
  if (n == 1) return 1;
  n = n-1;
  return n * GammaInt(n);
}
```

- 執行過程 `}`

← 2

函式名稱的可見性

- 宣告後函式才可見

```
// test 不可見

void test();


// test 可見
```

函式名稱的可見性

- 直接實作亦可

```
// test 不可見

void test() {
}

// test 可見
```

函式名稱的可見性

- 範例

```
void test();

int main() {
  test();
}

void test() {
  std::cout << "in test()\n";
}
```