由片語學習C程式設計

台灣大學資訊工程系劉邦鋒著

台灣大學劉邦鋒老師講授

August 19, 2016

第九單元

指標

- 指標變數和一般變數的差別在於 C 程式語言中對變數值的解讀。
- 一般變數的值就是代表資料，而指標變數的值則代表另一個變數的記憶體位址。
- 一般變數有資料類別，例如整數，則我們就將這個變數的值當作一個整數。
- 指標變數也有資料類別，例如指向整數的指標，則我們就將這個變數的值當作一個整數的記憶體位址。

學習要點

指標變數的值代表另一個變數的記憶體位址。

片語 1: 宣告一個指向整數的指標變數。

```
1  int *iptr;
```

- 宣告一個指向整數的指標變數。
- 在宣告變數時在變數名稱前加星號 * 就代表這是一個指標變數。而星號之前的資料類別就是這個指標變數所能指到的變數資料類別。

特殊字元

在宣告變數時在變數名稱前加星號 * 就代表這是一個指標變數。

片語 2: 指向浮點數的指標變數

```
1 float *fptr;
2 double *dfptr;
```

- 指標變數也可以指向浮點數。

範例程式 3: (size.c) 指標變數所佔的位元組數

```
1  #include <stdio.h>
2  int main(void)
3  {
4     int *iptr;
5     float *fptr;
6     double *dptr;
7     printf("sizeof(iptr) = %d\n", sizeof(iptr));
8     printf("sizeof(fptr) = %d\n", sizeof(fptr));
9     printf("sizeof(dptr) = %d\n", sizeof(dptr));
10    return 0;
11 }
```

輸出

```
1 sizeof(iptr) = 8
2 sizeof(fptr) = 8
3 sizeof(dptr) = 8
```

- 不論是指向 4 個位元組的整數，或是 8 個位元組的倍準浮點數，指標變數都是佔 8 個位元組，因為他們都是存 8 個位元組 (64 bit) 的記憶體位址。

片語 4: 指定整數指標變數的值。

```
1 int i;
2 int *iptr1;
3 int *iptr2;
4 iptr1 = &i;
5 iptr2 = iptr1;
```

- 我們用 **整數指標變數** 來代替 **指向整數的指標變數**。
- 當一個整數指標變數 `iptr` 的值是另一個整數變數 `i` 的記憶體位址 時，稱 `iptr` **指向** `i`。
- 一個整數變數的記憶體位址 可以指定給一個整數指標變數當作值，也可以將一個整數指標變數的值指定給另一個整數指標變數當作值。

取值

片語 5: 使用指標變數所指到的變數。

```
1 i = *iptr;
2 *iptr = i;
```

- 當一個指標變數前面加上星號時，就代表從這個記憶體位址 **取值** (dereference)。
- 當一個指標 `iptr` 變數指向一個變數 `i` 時，`*iptr` 就代表變數 `i` 的值。
- `*iptr` 可以出現在 `=` 的右邊或左邊。
  - 如果在右邊，則代表從這個記憶體位址 取值。
  - 如果在左邊，則代表將來從這個記憶體位址 取值，會取出現在在 `=` 右邊的值。

學習要點

如果 ptr 是一個指向某資料類別的指標變數，*ptr 就如同一個該資料類別的變數。

初始化

- 程式在使用記憶體時，能使用的記憶體位址有一定的範圍，超出範圍就會產生執行錯誤。
- 必須假設指標變數 `iptr` 已經指向一個變數，這時才能使用 `*iptr` 取值。
- 如果一個指標變數沒有經過正確的初始化，那麼它的值極有可能不在正確的範圍內，就無法從記憶體正確取值。

NULL

片語 6: NULL 絕不指向任何有效的記憶體位址

```
1 #include <stdio.h>
2 ...
3 ptr = NULL;
```

- 有時我們希望程式能夠讓一個指標變數 **不指向** 任何有效的記憶體位址，方法就是將它初始化成一個 "特殊值"。
- 為此C 程式語言定義了 NULL 作為這樣的用途。一般這個值是定義成 0，因為 0 這個記憶體位址通常是保留給系統使用，一般程式是不能使用的。
- NULL 是在 <stdio.h> 標頭檔定義的，所以必須引入 <stdio.h>。

範例程式 7: (segment-fault.c) 沒有經過正確的初始化

```
1 #include <stdio.h>
2 int main(void)
3 {
4   int *iptr;
5   iptr = NULL;
6   printf("*iptr = %d\n", *iptr);
7   return 0;
8 }
```

- 執行這個範例會導致執行錯誤，因為 NULL 絕對不指向任何有效的記憶體位址。

範例程式 8: (init-assign.c) 使用一個指向整數的指標變數

```
 4   int i, k;
 5   int *iptr1, *iptr2;
 6   scanf("%d", &i);
 7   iptr1 = &i;
 8   iptr2 = iptr1;
 9   printf("i = %d\n", i);
10   printf("&i = %p\n", &i);
11   printf("iptr1 = %p\n", iptr1);
12   printf("&iptr1 = %p\n", &iptr1);
13   printf("iptr2 = %p\n", iptr2);
14   printf("&iptr2 = %p\n", &iptr2);
15   *iptr1 = 8;
16   printf("i = %d\n", i);
17   k = *iptr2 + 3;
18   printf("&k = %p\n", &k);
19   printf("k = %d\n", k);
```

執行過程

- 宣告兩個指向整數的指標變數 `iptr`， `iptr2`。
- 將整數 `i` 的位址指定給 `iptr1`，再將 `iptr1` 指定給 `iptr2`，所以`iptr2` 現在也指向 `i`。
- 取出 `*iptr2` 的值，加上 3，指定給 `k`。由於 `iptr2` 指向 `i`，而且 現在 `i` 的值是 8，所以會印出 11。
- 指標變數與一般變數的值都是一串 0 與 1，事實上是沒有區別的。 唯一不同之處是怎樣去解讀其中的內容。

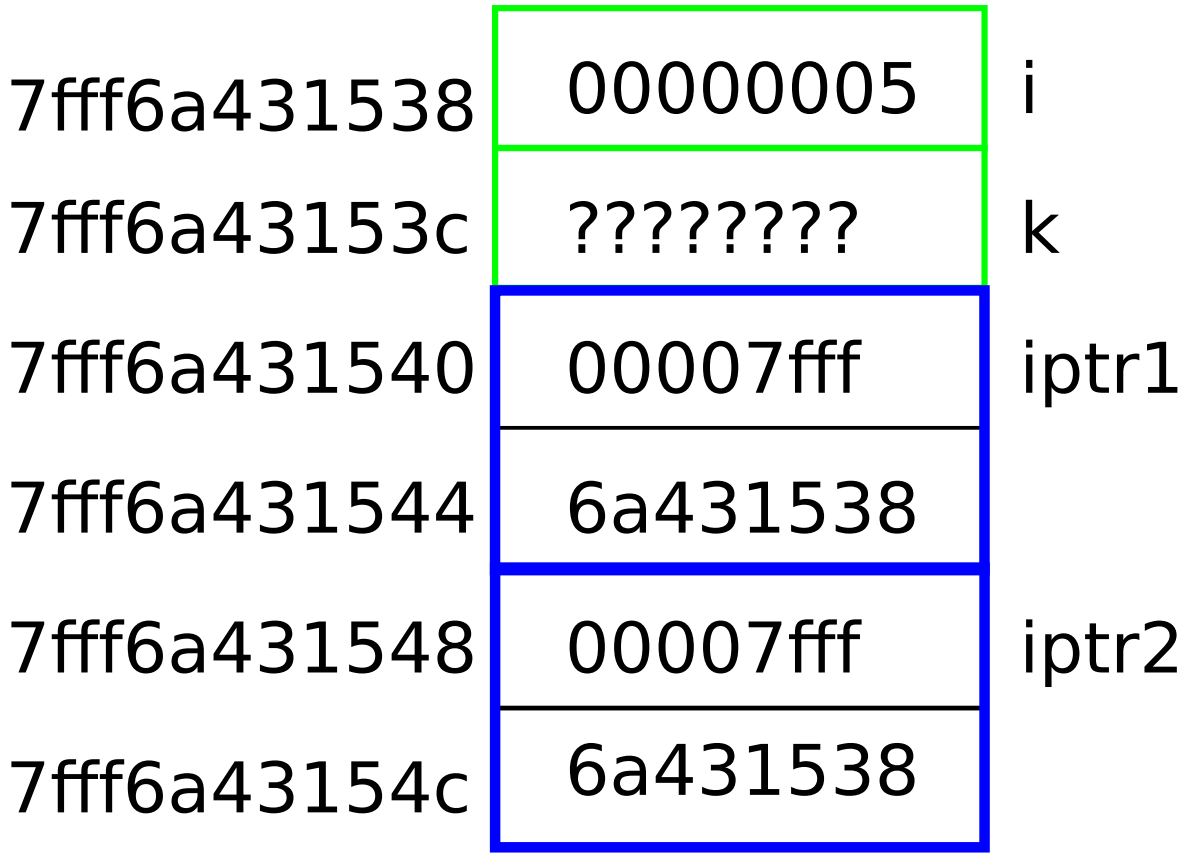
輸入

```
1 5
```

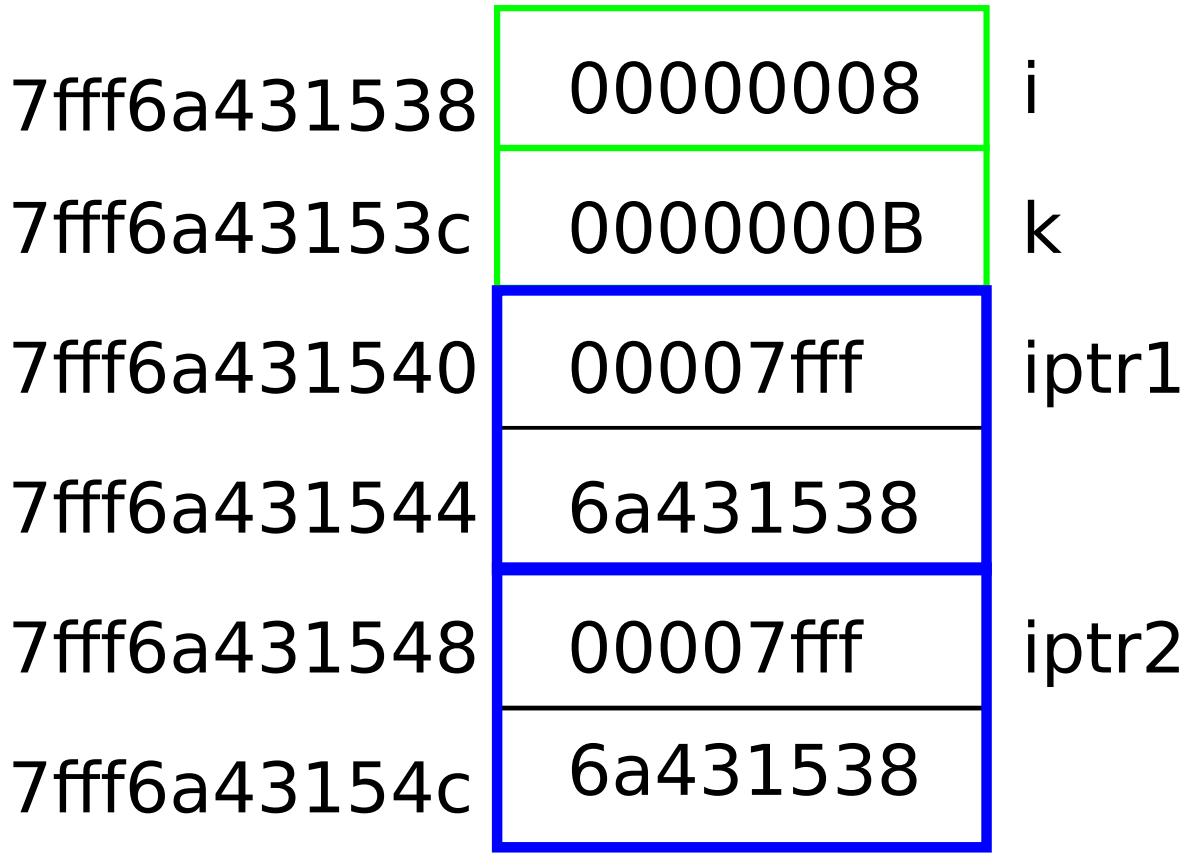
輸出

```
1 i = 5
2 &i = 0x7fff6a431538
3 iptr1 = 0x7fff6a431538
4 &iptr1 = 0x7fff6a431540
5 iptr2 = 0x7fff6a431538
6 &iptr2 = 0x7fff6a431548
7 i = 8
8 &k = 0x7fff6a43153c
9 k = 11
```

記憶體內容


7fff6a431538
00000005
i
7fff6a43153c
????????
k
7fff6a431540
00007fff
iptr1
7fff6a431544
6a431538
7fff6a431548
00007fff
iptr2
7fff6a43154c
6a431538

記憶體內容


7fff6a431538
00000008
i
7fff6a43153c
0000000B
k
7fff6a431540
00007fff
iptr1
7fff6a431544
6a431538
7fff6a431548
00007fff
iptr2
7fff6a43154c
6a431538

範例程式 9: (address-deference.c) * 取值與 & 取位址的關係

```
4   int i;
5   int *iptr = &i;
6   scanf("%d", &i);
7   printf("iptr = %p\n", iptr);
8   printf("&iptr = %p\n", &iptr);
9   printf("*iptr = %d\n", *iptr);
10  printf("*(&iptr) = %p\n", *(&iptr));
11  printf("&(*iptr) = %p\n", &(*iptr));
12  printf("*(*(&iptr)) = %d\n", *(*(&iptr)));
13  printf("*(&(*iptr)) = %d\n", *(&(*iptr)));
14  printf("&(*(&iptr)) = %p\n", &(*(&iptr)));
```

```
16    printf("i = %d\n", i);
17    printf("&i = %p\n", &i);
18    /* printf("*i = %p\n", *i); do not do this */
19    printf("*(&i) = %d\n", *(&i));
20    /* printf("&(*i) = %p\n", &(*i));
21       do not do this either */
```

- `*(&iptr)` 與 `&(*iptr)` 值都是 `i` 的記憶體位址，但兩者的意義略有不同。
- `*(&iptr)` 先對 `iptr` 取位址得到 `iptr` 的記憶體位址。然後再對這個記憶體位址取值。由於此時 `iptr` 指向 `i`，結果得到 `i` 的記憶體位址。
- `&(*iptr)` 先對 `iptr` 取值，由於此時 `iptr` 指向 `i`，`*iptr` 就如同 `i` 一樣。所以 `&(*iptr)` 就等同於取 `i` 的記憶體位址。

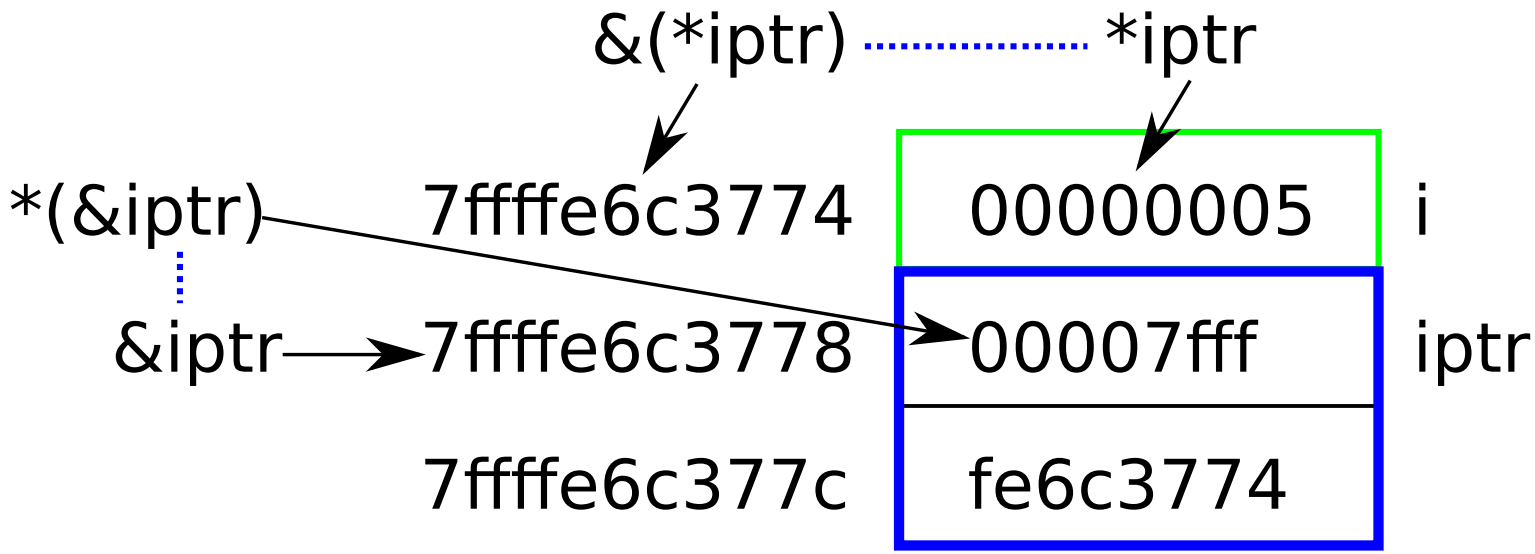
輸入

```
1 5
```

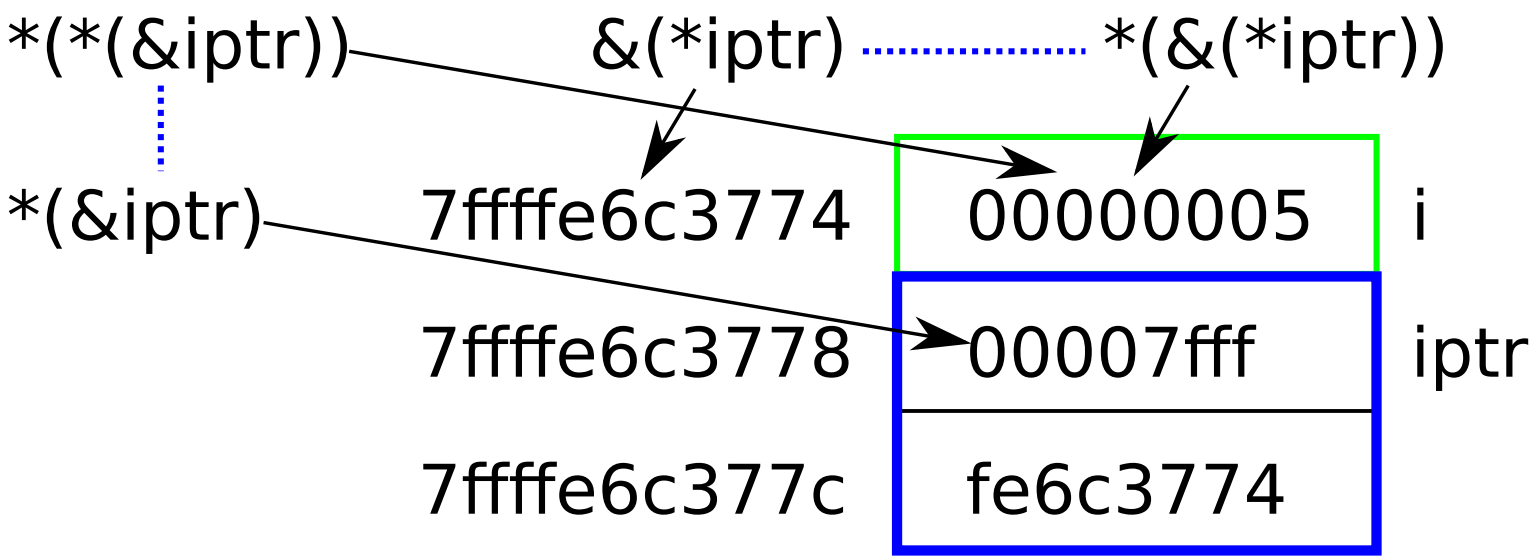
輸出

```
1  iptr = 0x7ffffe6c3774
2  &iptr = 0x7ffffe6c3778
3  *iptr = 5
4  *(&iptr) = 0x7ffffe6c3774
5  &(*iptr) = 0x7ffffe6c3774
6  *(*(&iptr)) = 5
7  *(&(*iptr)) = 5
8  &(*(&iptr)) = 0x7ffffe6c3778
9  i = 5
10 &i = 0x7ffffe6c3774
11 *(&i) = 5
```

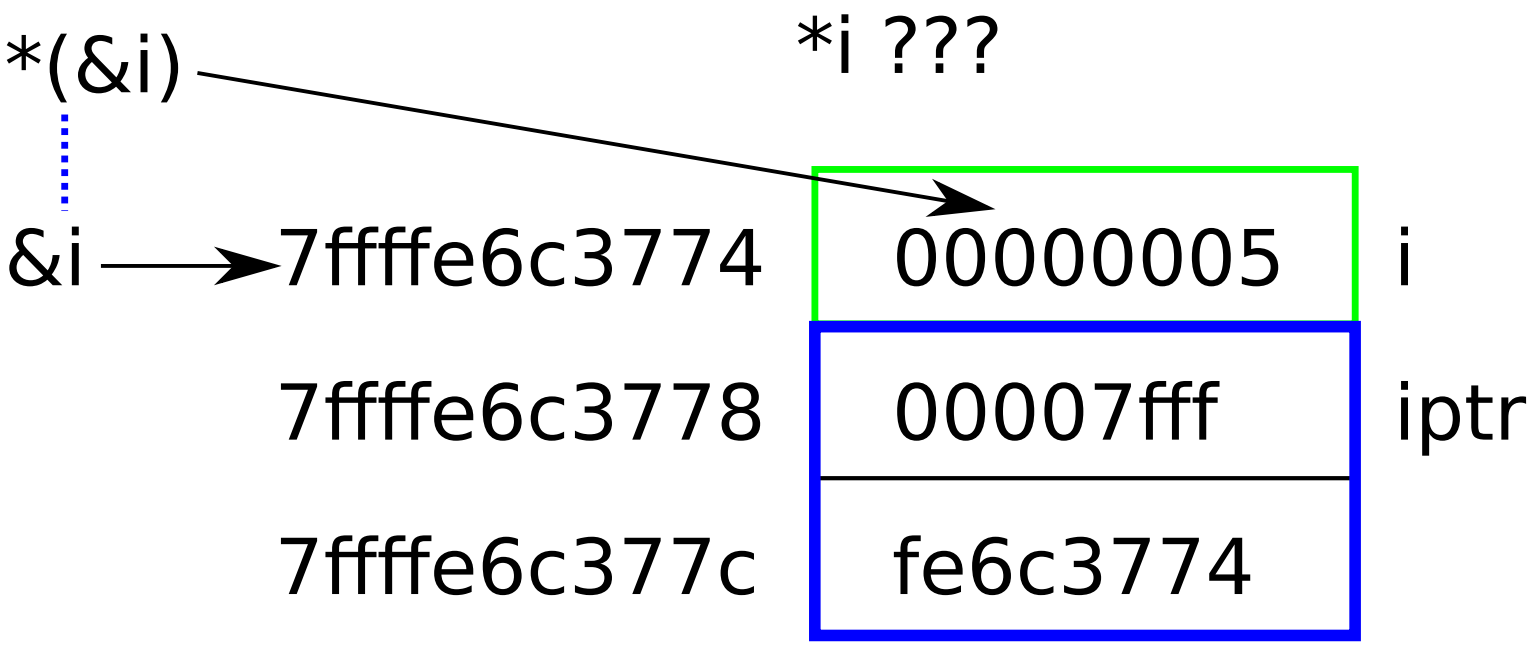
記憶體內容


&(*iptr)
*iptr
*(&iptr)
7ffffe6c3774
00000005
i
&iptr
7ffffe6c3778
00007fff
iptr
7ffffe6c377c
fe6c3774

記憶體內容


((&iptr))
&(*iptr)
*(&(*iptr))
*(&iptr)
7ffffe6c3774
00000005
i
7ffffe6c3778
00007fff
iptr
7ffffe6c377c
fe6c3774

記憶體內容


*(&i)
*i ???
&i
7ffffe6c3774
00000005
i
7ffffe6c3778
00007fff
iptr
7ffffe6c377c
fe6c3774

範例程式 10: (pointer-parameter.c) 指標參數傳遞

```
2  void pointer_inc(int *p1, int *p2)
3  {
4    printf("The address of p1 is %p\n", &p1);
5    printf("The value of p1 is %p\n", p1);
6    printf("The address of p2 is %p\n", &p2);
7    printf("The value of p2 is %p\n", p2);
8    *p1 += 1;
9    p1 = p2;
10   *p1 += 2;
11 }
```

```
13 int main(void)
14 {
15    int i, j;
16    int *iptr = &i;
17    scanf("%d", &i);
18    scanf("%d", &j);
19    printf("The address of i is %p\n", &i);
20    printf("The address of j is %p\n", &j);
21    printf("The address of iptr is %p\n", &iptr);
22    printf("i = %d, j = %d\n", i, j);
23    pointer_inc(iptr, &j);
24    printf("i = %d, j = %d\n", i, j);
25    *iptr += 5;
26    printf("i = %d, j = %d\n", i, j);
27    return 0;
28 }
```

輸入

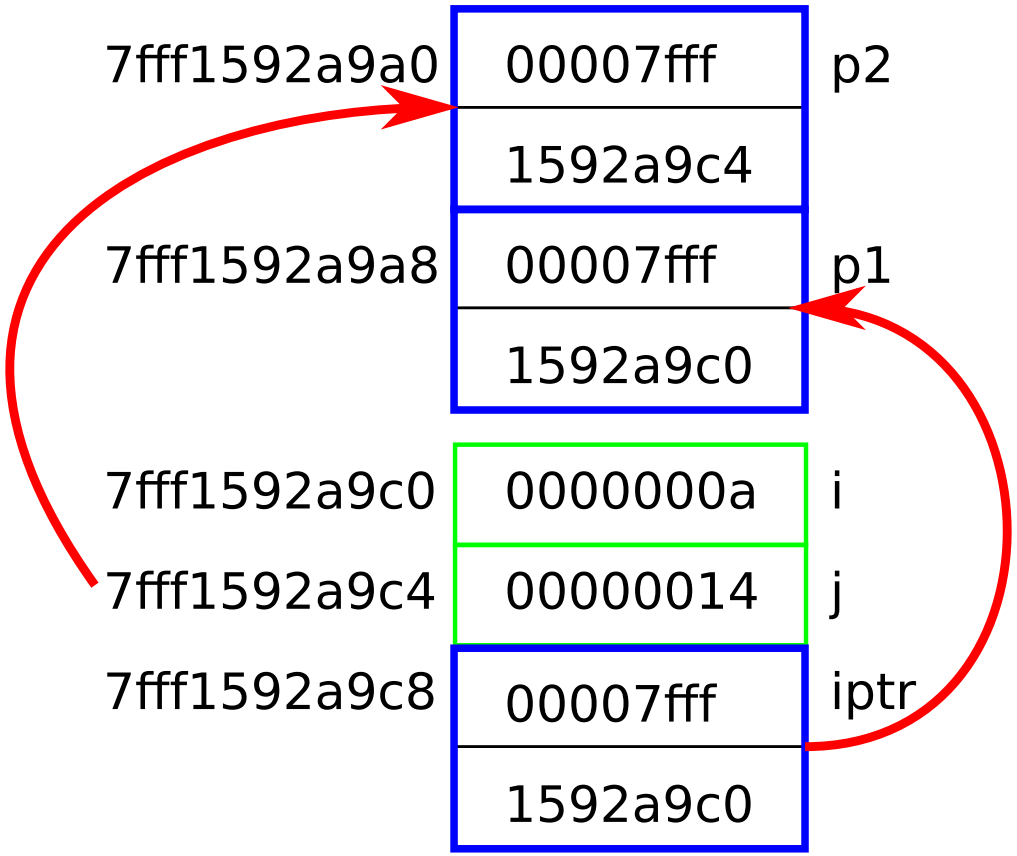
```
10 20
```

輸出

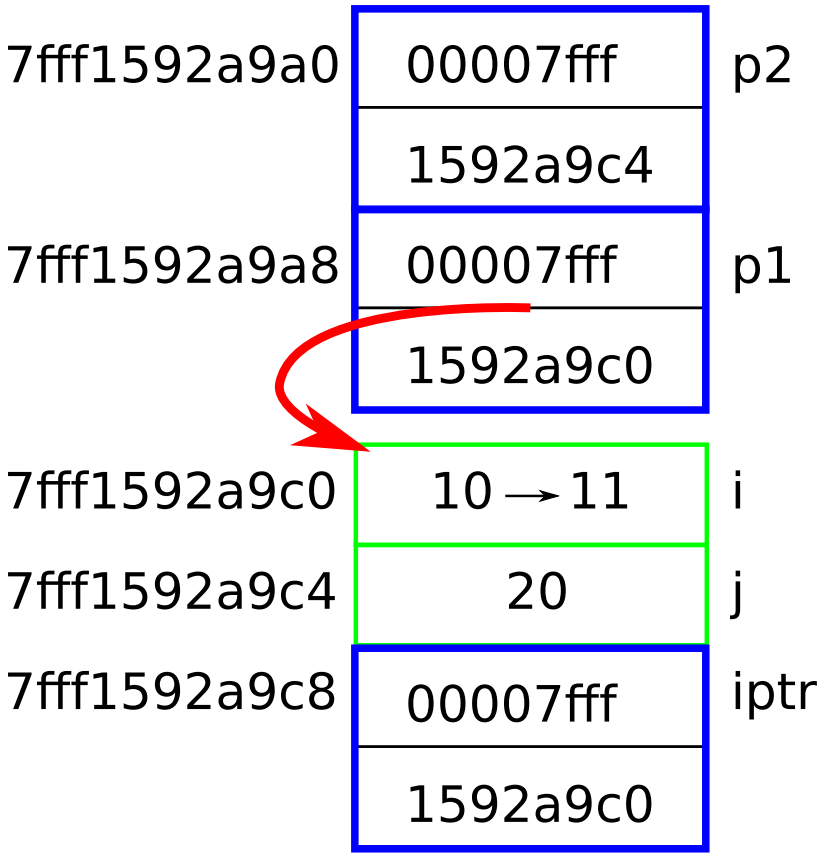
```
The address of i is 0x7fff1592a9c0
The address of j is 0x7fff1592a9c4
The address of iptr is 0x7fff1592a9c8
i = 10, j = 20
The address of p1 is 0x7fff1592a9a8
The value of p1 is 0x7fff1592a9c0
The address of p2 is 0x7fff1592a9a0
The value of p2 is 0x7fff1592a9c4
i = 11, j = 22
i = 16, j = 22
```

1. 實際參數與形式參數使用不同的記憶體，所以實際參數 iptr 和對應的形式參數 p1 位於不同的記憶體位址。
2. 雖然 iptr 和 p1 位於不同的記憶體位址，但他們都指向 i，所以 i 會加 1。
3. 形式參數 p2 拿到的值是實際參數 &j，所以 p2 指向 j。
4. 因為 p2 指向 j，所以會將 j 加 2。
5. 雖然 p1 已經改指向 j，但是 iptr 還是指向 i，因為 p1 和 iptr 是位於 **不同** 記憶體位址的不同變數，改變一個並不會改變另一個，
6. 加 5 到 i，而非 j。

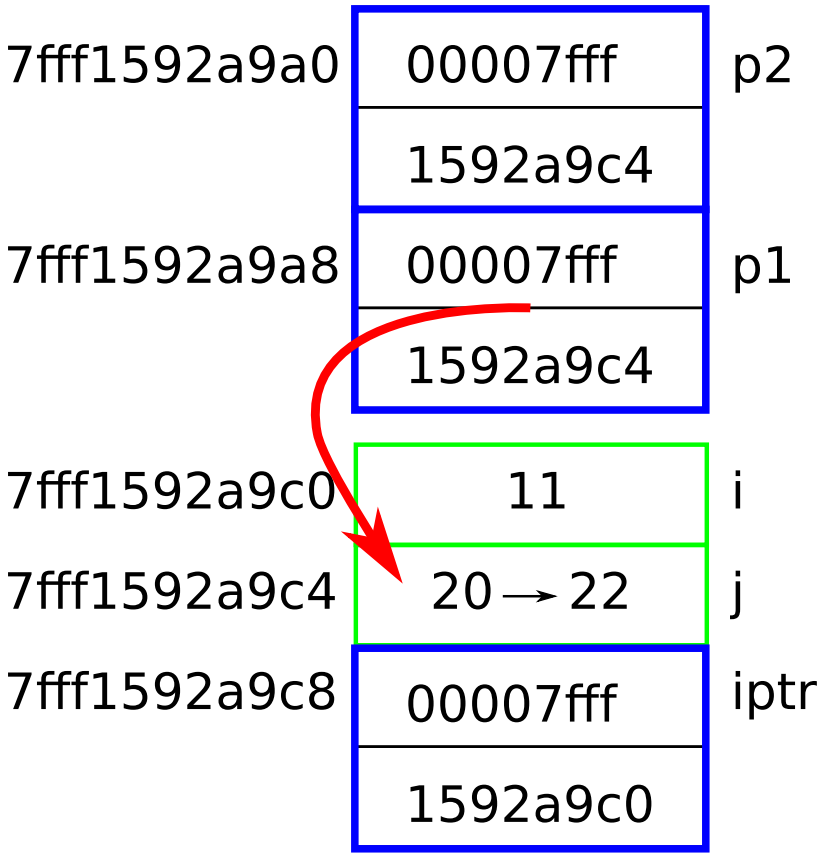
記憶體內容


7fff1592a9a0
00007fff
1592a9c4
p2
7fff1592a9a8
00007fff
1592a9c0
p1
7fff1592a9c0
0000000a
i
7fff1592a9c4
00000014
j
7fff1592a9c8
00007fff
1592a9c0
iptr

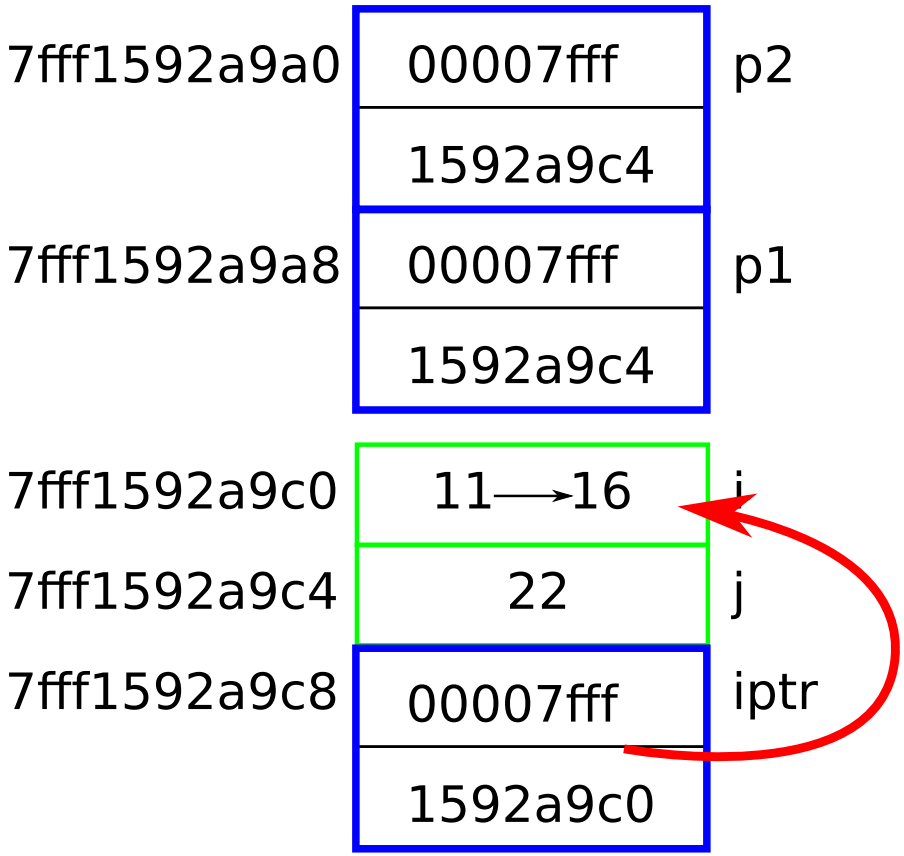
記憶體內容


7fff1592a9a0
00007fff
p2
1592a9c4
7fff1592a9a8
00007fff
p1
1592a9c0
7fff1592a9c0
10 → 11
i
7fff1592a9c4
20
j
7fff1592a9c8
00007fff
iptr
1592a9c0

記憶體內容


7fff1592a9a0
00007fff
p2
1592a9c4
7fff1592a9a8
00007fff
p1
1592a9c4
7fff1592a9c0
11
i
7fff1592a9c4
20 → 22
j
7fff1592a9c8
00007fff
iptr
1592a9c0

記憶體內容


7fff1592a9a0
00007fff
p2
1592a9c4
7fff1592a9a8
00007fff
p1
1592a9c4
7fff1592a9c0
11⟶16
i
7fff1592a9c4
22
j
7fff1592a9c8
00007fff
iptr
1592a9c0

片語 11: 指標指向陣列的起始記憶體位址

```
1 int array[100];
2 int *iptr;
3 ...
4 iptr = array;
5 ...
6 use *iptr as array[0]
7 ...
```

- `iptr = array;` 就是使 `iptr` 指向陣列的起始記憶體位址。
- 因為陣列的起始記憶體位址 是 `array[0]` 的位址，從此 `*iptr` 就如同 `array[0]`。有如用指標的語法從陣列內取元素。

片語 12: 將指標加 1

```
1 iptr++;
```

- C 程式語言中有一套指標的算術語法，讓指標可以加減一個整數，最常用的就是將指標加 1。
- 這裡的加 1 是指加 **一個元素的大小**。所以如果是整數，指標變數就是加 `sizeof(int) = 4`，而如果是倍準浮點數，指標變數就是加 `sizeof(double) = 8`。可以想像成成指向 **下一個** 元素。

範例程式 13: (inc3-with-pointer.c) 利用指標將陣列元素加 3

```
1  #include <stdio.h>
2  int main(void)
3  {
4    int a[5];
5    int i;
6    int *ptr;
7    for (i = 0; i < 5; i++)
8      scanf("%d", &(a[i]));
9    for (i = 0, ptr = a; i < 5; i++, ptr++) {
10     printf("%p\n", ptr);
11     *ptr += 3;
12   }
13   for (i = 0; i < 5; i++)
14     printf("a[%d] = %d\n", i, a[i]);
15   return 0;
16 }
```

- for 迴圈的初始部分為 ptr = a，讓 for 迴圈開始時，ptr 指向 a[0]。
- 調整部分為 ptr++， 讓 for 迴圈每經過一次，ptr 就指向下一個元素。
- 加 3 的部分寫成 *ptr += 3; 即可，因為此時 *ptr 和 a[i] 是一樣的。

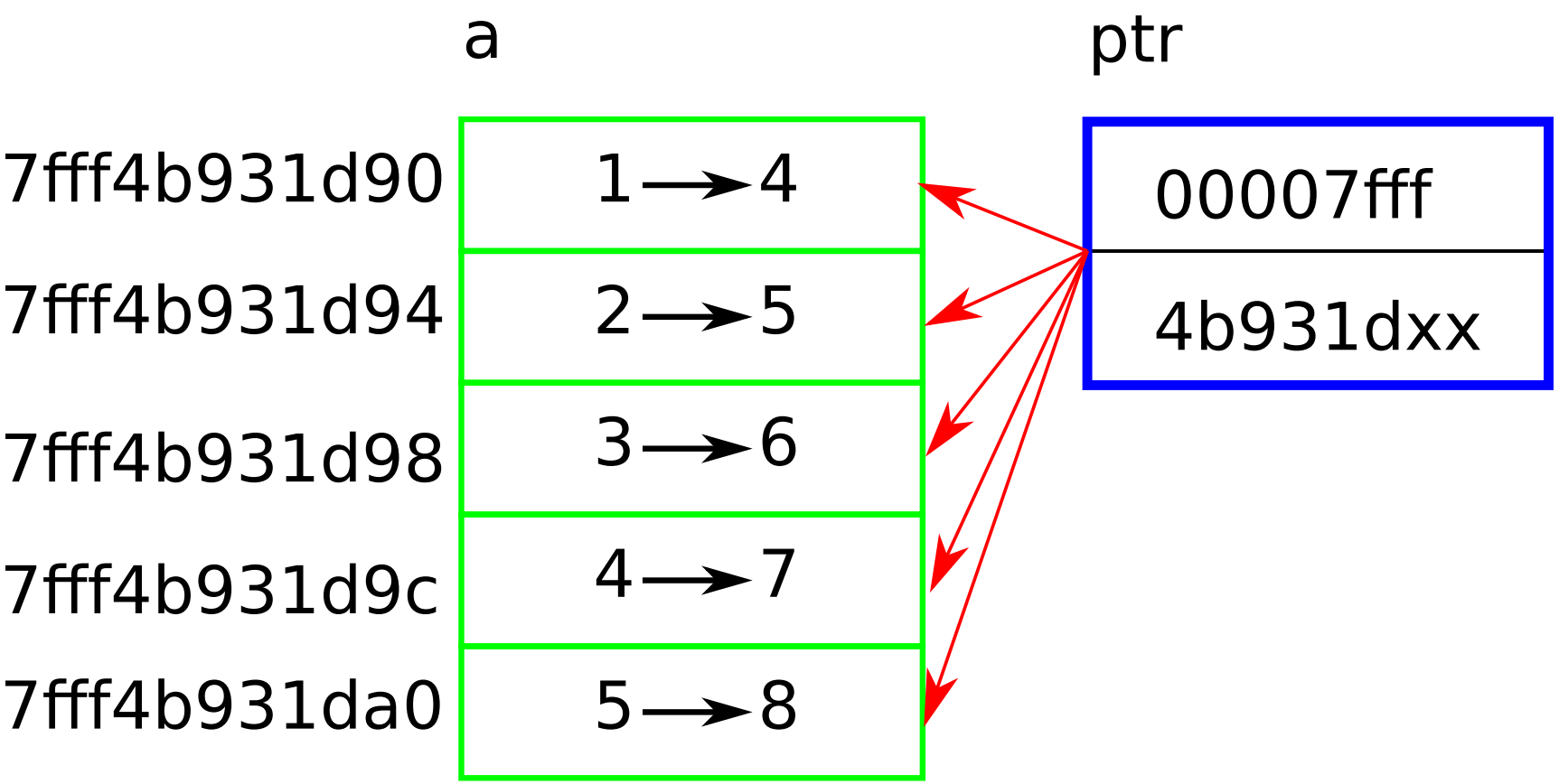
輸入

```
1 1 2 3 4 5
```

輸出

```
0x7fff4b931d90
0x7fff4b931d94
0x7fff4b931d98
0x7fff4b931d9c
0x7fff4b931da0
a[0] = 4
a[1] = 5
a[2] = 6
a[3] = 7
a[4] = 8
```

記憶體內容


a
ptr
7fff4b931d90
1→4
7fff4b931d94
2→5
7fff4b931d98
3→6
7fff4b931d9c
4→7
7fff4b931da0
5→8
00007fff
4b931dxx

片語 14: 指標也可用陣列的語法

```
1 *(iptr + i)
2 iptr[i]
```

- 指標變數加 i 就是指向下 i 個的意思。
- 指標變數 iptr 指向陣列 a 的起始位置，再加 i (如同往後算 i 個元素)，再用星號取值，那 *(iptr + i) 就正是 a[i]。

範例程式 15: (inc-with-array-index.c) 利用指標修改陣列

```
1  #include <stdio.h>
2  int main(void)
3  {
4    int a[5];
5    int i;
6    int *ptr;
7    for (i = 0; i < 5; i++)
8      scanf("%d", &(a[i]));
9    for (i = 0, ptr = a; i < 2; i++)
10     ptr[i] += 3;
11   for (i = 0; i < 5; i++)
12     printf("a[%d] = %d\n", i, a[i]);
13   printf("\n");
14   for (i = 0, ptr = &(a[2]); i < 2; i++)
15     ptr[i] += 3;
16   for (i = 0; i < 5; i++)
17     printf("a[%d] = %d\n", i, a[i]);
18   return 0;
19 }
```

- 指標變數使用註標語法時和原來陣列使用註標語法不同。
- 陣列 a 使用註標語法時 a[1] 永遠是同一個陣列元素。
  - 絕對座標
- 指標變數 ptr 使用註標語法時，ptr[1] 卻是 ptr 目前所指到位置的下一個陣列元素。
  - 相對座標

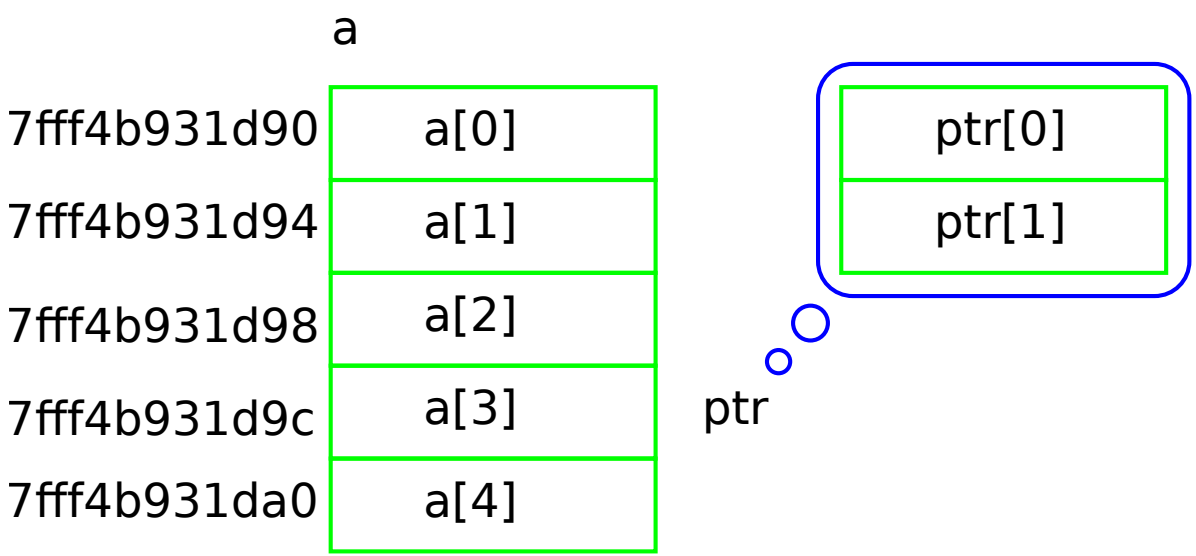
輸入

```
1  1 2 3 4 5
```

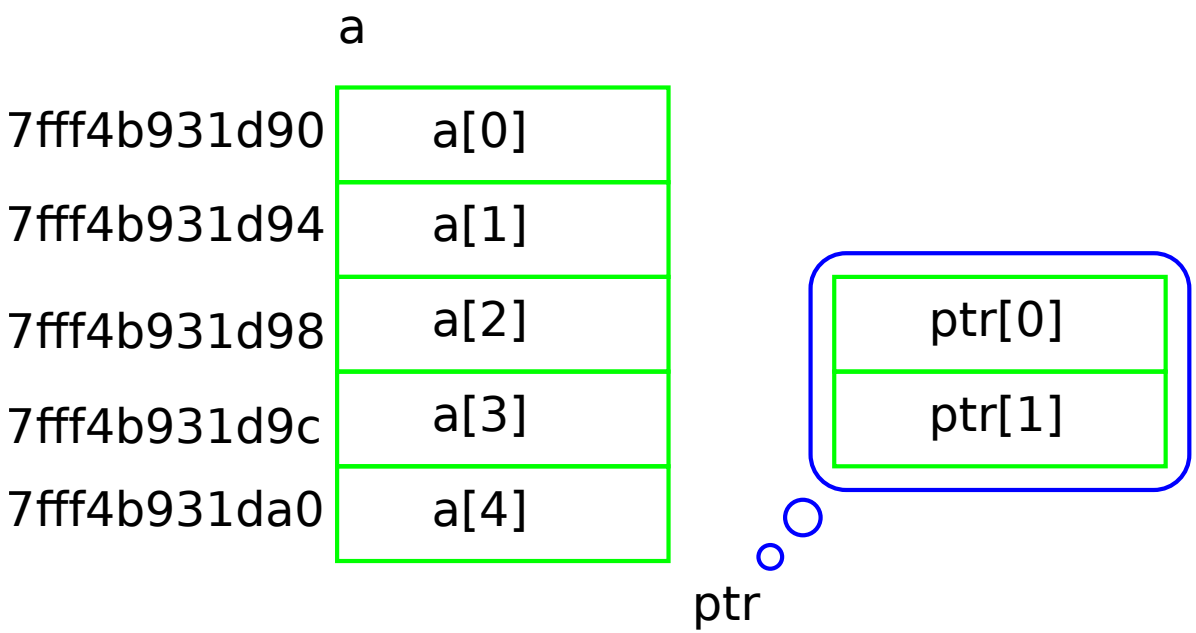
輸出

```
1  a[0] = 4
2  a[1] = 5
3  a[2] = 3
4  a[3] = 4
5  a[4] = 5
6
7  a[0] = 4
8  a[1] = 5
9  a[2] = 6
10 a[3] = 7
11 a[4] = 5
```

記憶體內容


a
7fff4b931d90
a[0]
7fff4b931d94
a[1]
7fff4b931d98
a[2]
7fff4b931d9c
a[3]
7fff4b931da0
a[4]
ptr
ptr[0]
ptr[1]

記憶體內容


a
7fff4b931d90
a[0]
7fff4b931d94
a[1]
7fff4b931d98
a[2]
7fff4b931d9c
a[3]
7fff4b931da0
a[4]
ptr[0]
ptr[1]
ptr

範例程式 16: (arith.c) 指標算術

```
1  #include <stdio.h>
2  int main(void)
3  {
4     int array[10];
5     int *iptr1 = &(array[3]);
6     int *iptr2 = iptr1 + 4;
7     printf("iptr1 = %p\n", iptr1);
8     printf("iptr2 = %p\n", iptr2);
9     printf("iptr2 - iptr1 = %d\n", iptr2 - iptr1);
10    return 0;
11 }
```

輸出

```
1 iptr1 = 0x7fffa00d9f8c
2 iptr2 = 0x7fffa00d9f9c
3 iptr2 - iptr1 = 4
```

- 指標算術可以將一個指標加一個常數得到一個指標，自然也可以減一個常數得到一個指標，而且兩個指標也可以相減得到一個常數。
- 這裡的常數都不是指一般以 位元組 為單位的 記憶體位址，而是以指標所指到 元素大小 為單位，

片語 17: 將整數指標當回傳值

```
1 int *first_positive(int *iptr);
```

- 指標也可以當回傳值， 作法是在函式名稱前加一個星號，可以想像 *first_positive 就如同一個整數。

範例程式 18: (first-positive.c) 將 `iptr` 所指到的記憶體中的第一個正整數的記憶體位址傳回

```
2  int *first_positive(int *ptr)
3  {
4    while (*ptr <= 0)
5      ptr++;
6    return ptr;
7  }
```

```
 9  int main(void)
10  {
11    int i;
12    int array[10];
13    int *iptr;
14
15    for (i = 0; i < 10; i++)
16      scanf("%d", &(array[i]));
17    iptr = first_positive(array);
18    printf("*iptr = %d\n", *iptr);
19    printf("iptr - array = %d\n", iptr - array);
20    iptr = first_positive(&(array[5]));
21    printf("*iptr = %d\n", *iptr);
22    printf("iptr - array = %d\n", iptr - array);
23    return 0;
24  }
```

- 第一次用 `array` 當實際參數呼叫 `first_positive`，結果回傳 `array[3]` 的記憶體位址，所以 `iptr` 取值就得到 5。
- 第二次用 `array[5]` 的記憶體位址 當實際參數呼叫 `first_positive`，結果回傳 `array[7]` 的記憶體位址，所以 `iptr` 取值就得到 6。

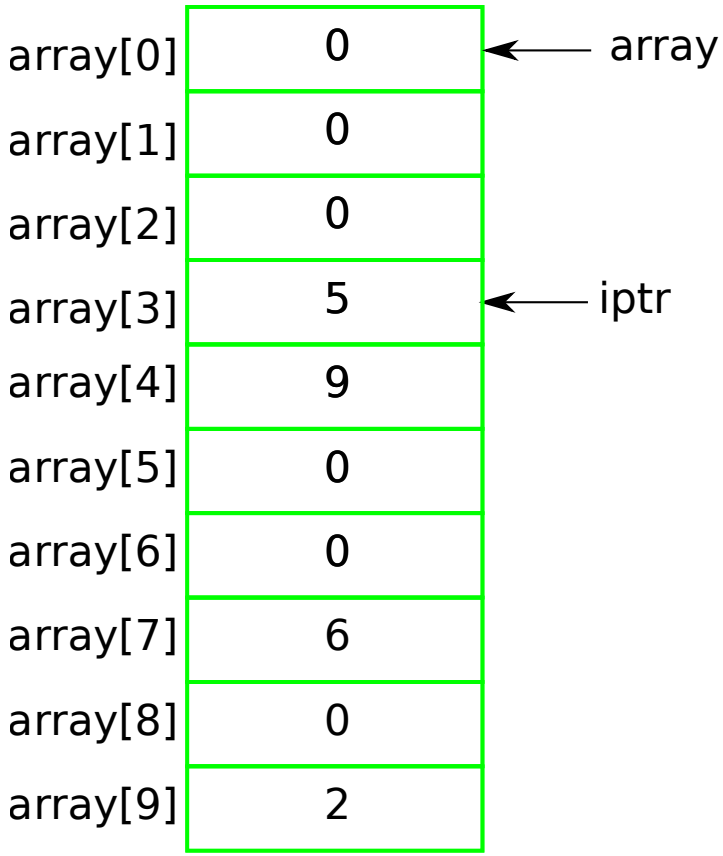
輸入

```
1 0 0 0 5 9 0 0 6 0 2
```

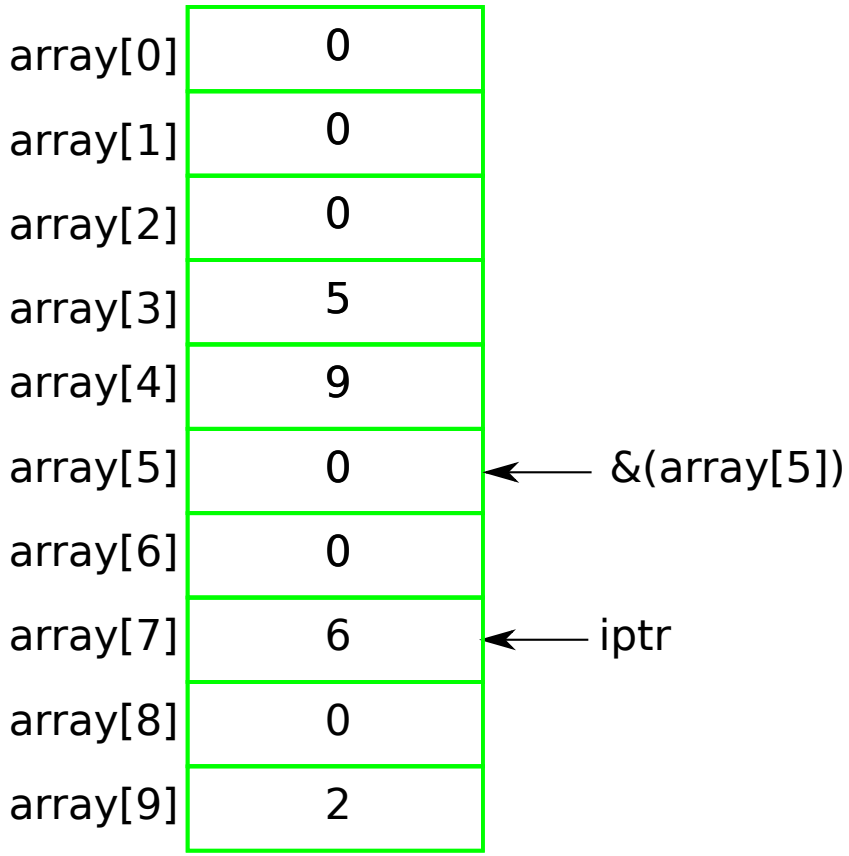
輸出

```
1 *iptr = 5
2 iptr - array = 3
3 *iptr = 6
4 iptr - array = 7
```

記憶體內容


array[0]
0
array
array[1]
0
array[2]
0
array[3]
5
iptr
array[4]
9
array[5]
0
array[6]
0
array[7]
6
array[8]
0
array[9]
2

記憶體內容


array[0]
0
array[1]
0
array[2]
0
array[3]
5
array[4]
9
array[5]
0
&(array[5])
array[6]
0
array[7]
6
iptr
array[8]
0
array[9]
2

用途

- 在動態配置記憶體時，我們可以直接向作業系統要求一塊記憶體使用，所以我們需要一個機制，讓程式有辦法能記住要來的記憶體記憶體位址，這個機制就是指標變數，
- 動態資料結構會將資料串連起來形成結構。此時資料結構的大小與形狀都是依動態要求而調整，此時我們就需要指標，來描述資料之間的連結關係。
- 在處理字串時，程式常常需要以記憶體位址來溝通。此時溝通的雙方未必能夠使用陣列的註標語法，因為其中一方未必能夠知道個陣列的起始記憶體位址。此時使用指標直接指到記憶體是最有效的溝通方法。

注意事項

- 除非是前述的動態配置記憶體，動態資料結構，及字串處理，否則請儘量避免使用指標。
- C 程式語言中對指標的使用沒有安全機制，初學很容易弄錯，而且難以除錯。
- 很多指標的使用是可以用陣列語法代替的，這樣不但容易閱讀，也容易除錯。
- 有人認為使用指標可增加效能，但是現代編譯器已經能產生非常好的執行檔。為了效能犧牲可讀性也許並不值得。