程式設計

第6章 指標
Pointer

蘇維宗(Wei-Tsung Su)
suwt@au.edu.tw
564D

目標

認識指標
函式呼叫參數傳遞方式
動態配置記憶體
指標陣列
命令列參數

函式指標*

認識指標

Understanding Pointer

指標

指標(pointer)是一種特殊的變數，只能用來儲存另一個變數的**記憶體位址**。

```
1.   int num = 100;          //整數變數
2.   int *ptr = NULL;        //指向整數變數的整數指標
```

變數名稱	記憶體位址	資料
num	2021	100

指標(續)

指標(pointer)是一種特殊的變數，只能用來儲存另一個變數的**記憶體位址**。

```
1.   int num = 100;          //整數變數
2.   int *ptr = NULL;        //指向整數變數的整數指標
```

註1:定義指標時須指定型別

註2:指標的值為**NULL**代表沒有指向任何東西

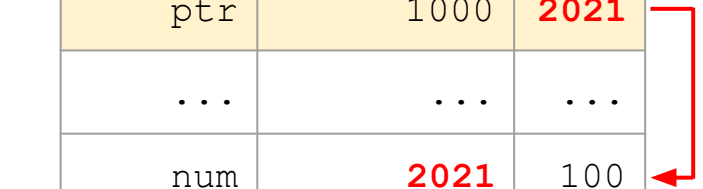
變數名稱	記憶體位址	資料
ptr	1000	NULL
...	...	...
num	2021	100

指標(續)

指標(pointer)是一種特殊的變數，只能用來儲存另一個變數的**記憶體位址**。

```
1.   int num = 100;          //整數變數
2.   int *ptr = NULL;        //指向整數變數的整數指標
3.   ptr = &num;             //將ptr指向num
```

變數名稱	記憶體位址	資料
ptr	1000	2021
...	...	...
num	2021	100



註1:利用**&**運算子取得變數的記憶體位址

指標(續)

指標(pointer)是一種特殊的變數，只能用來儲存另一個變數的**記憶體位址**。

```
1.   int num = 100;          //整數變數
2.   int *ptr = NULL;        //指向整數變數的整數指標
3.   ptr = &num;             //將ptr指向num
4.   printf("%d\n", num);    //印出num的值:?
5.   printf("%p\n", &num);   //印出num的記憶體位址:?
6.   printf("%p\n", ptr);    //印出ptr的值 : ?
7.   printf("%d\n", *ptr);   //印出ptr指向的變數的值 : ?
```

變數名稱	記憶體位址	資料
ptr	1000	2021
...	...	...
num	2021	100

註1:利用*運算子取得指標指向之變數的值

練習

可能用到的觀念
指標

1. 定義兩種不同型別的變數
2. 定義兩個指標指向兩個變數
3. 利用指標印出變數的記憶體位址
4. 利用指標印出變數的 值

A few moments later …

指向字元陣列的指標

charpt.c

```
1.   char name[] = "CSIE";
2.   char *ptr1 = name;
3.   char *ptr2 = name + 2;
```

註:請問字元陣列中所有元素的記憶體位址?

變數名稱	記憶體位址	資料
name	2025	'C'
	?	'S'
	?	'I'
	?	'E'
	?	'\0'

指向字元陣列的指標(續)

charpt.c

```
1.   char name[] = "CSIE";
2.   char *ptr1 = name;
3.   char *ptr2 = name + 2;
```

變數名稱	記憶體位址	資料
ptr1	1008	2025
...	...	...
name	2025	'C'
	?	'S'
	?	'I'
	?	'E'
	?	'\0'

指向字元陣列的指標(續)

charpt.c

```
1.   char name[] = "CSIE";
2.   char *ptr1 = name;
3.   char *ptr2 = name + 2;
4.
5.   printf("%p\n", &ptr1);                  //印出?
6.   printf("%p %c\n", ptr1, *ptr1);         //印出?
7.   printf("%p %c\n", ptr1+1, *(ptr1+1));   //印出?
8.   printf("%p %c\n", ptr2, *ptr2);         //印出?
9.   printf("%s\n", ptr2);                   //印出?
```

變數名稱	記憶體位址	資料
ptr2	1000	?
ptr1	1008	2025
...	...	...
name	2025	'C'
	?	'S'
	?	'I'
	?	'E'
	?	'\0'

指向整數陣列的指標

intpt.c

```
1.   int score[] = {10, 20, 30, 40};
2.   int *ptr1 = score;
3.   int *ptr2 = score + 2;
```

變數名稱	記憶體位址	資料
score	2000	10
	?	20
	?	30
	?	40

註:請問整數陣列中所有元素的記憶體位址?

指向整數陣列的指標

intpt.c

```
1.   int score[] = {10, 20, 30, 40};
2.   int *ptr1 = score;
3.   int *ptr2 = score + 2;
```

變數名稱	記憶體位址	資料
ptr1	1008	2000
...	...	...
score	2000	10
	?	20
	?	30
	?	40

指向整數陣列的指標

intpt.c

```
1.   int score[] = {10, 20, 30, 40};
2.   int *ptr1 = score;
3.   int *ptr2 = score + 2;
4.
5.   printf("%p\n", &ptr1);                  //印出?
6.   printf("%p %d\n", ptr1, *ptr1);         //印出?
7.   printf("%p %d\n", ptr1+1, *(ptr1+1));   //印出?
8.   printf("%p %d\n", ptr2, *ptr2);         //印出?
```

變數名稱	記憶體位址	資料
ptr2	1000	?
ptr1	1008	2000
...	...	...
score	2000	10
	?	20
	?	30
	?	40

練習:找出偶數

輸入n個整數(n<=50)儲存在整數陣列中，請印出陣列中所有偶數。

限制條件:
以指標存取陣列內的元素

可能用到的觀念
指標與陣列

輸入	輸出
1 2 3 4 5 6	2 4 6
20 19 2 27	20 2

A few moments later …

函式呼叫參數傳遞方式

Function Call by Value, Address

傳值(Call by Value)

顧名思義，函式呼叫採用**傳值**傳遞參數，是將變數的**值**傳入函式。

那麼，請問此程式會印出什麼?

vswap.c

```
1.  void vswap(int x, int y) {
2.    int temp = x;
3.    x = y;
4.    y = temp;
5.  }
6.
7.  int main(void) {
8.    int num1 = 10, num2 = 5;
9.    printf("%d %d\n", num1, num2);
10.   vswap(num1, num2);
11.   printf("%d %d\n", num1, num2);
12.   return EXIT_SUCCESS;
13. }
```

傳值(Call by Value)

變數名稱	記憶體位址	資料
num2	2000	5
num1	2004	10

vswap.c

```
1.  void vswap(int x, int y) {
2.    int temp = x;
3.    x = y;
4.    y = temp;
5.  }
6.
7.  int main(void) {
8.    int num1 = 10, num2 = 5;
9.    printf("%d %d\n", num1, num2);
10.   vswap(num1, num2);
11.   printf("%d %d\n", num1, num2);
12.   return EXIT_SUCCESS;
13. }
```

傳值(Call by Value)

變數名稱	記憶體位址	資料
y	1012	5
x	1016	10
...	...	...
num2	2000	5
num1	2004	10

vswap.c

```
1.  void vswap(int x, int y) {
2.    int temp = x;
3.    x = y;
4.    y = temp;
5.  }
6.
7.  int main(void) {
8.    int num1 = 10, num2 = 5;
9.    printf("%d %d\n", num1, num2);
10.   vswap(num1, num2);
11.   printf("%d %d\n", num1, num2);
12.   return EXIT_SUCCESS;
13. }
```

傳值(Call by Value)

變數名稱	記憶體位址	資料
temp	1000	10
...	...	...
y	1012	5
x	1016	10
...	...	...
num2	2000	5
num1	2004	10

vswap.c

```
1.  void vswap(int x, int y) {
2.    int temp = x;
3.    x = y;
4.    y = temp;
5.  }
6.
7.  int main(void) {
8.    int num1 = 10, num2 = 5;
9.    printf("%d %d\n", num1, num2);
10.   vswap(num1, num2);
11.   printf("%d %d\n", num1, num2);
12.   return EXIT_SUCCESS;
13. }
```

傳值(Call by Value)

變數名稱	記憶體位址	資料
temp	1000	10
...	...	...
y	1012	5
x	1016	5
...	...	...
num2	2000	5
num1	2004	10

vswap.c

```
1.  void vswap(int x, int y) {
2.    int temp = x;
3.    x = y;
4.    y = temp;
5.  }
6.
7.  int main(void) {
8.    int num1 = 10, num2 = 5;
9.    printf("%d %d\n", num1, num2);
10.   vswap(num1, num2);
11.   printf("%d %d\n", num1, num2);
12.   return EXIT_SUCCESS;
13. }
```

傳值(Call by Value)

變數名稱	記憶體位址	資料
temp	1000	10
...	...	...
y	1012	10
x	1016	5
...	...	...
num2	2000	5
num1	2004	10

vswap.c

```
1.  void vswap(int x, int y) {
2.    int temp = x;
3.    x = y;
4.    y = temp;
5.  }
6.
7.  int main(void) {
8.    int num1 = 10, num2 = 5;
9.    printf("%d %d\n", num1, num2);
10.   vswap(num1, num2);
11.   printf("%d %d\n", num1, num2);
12.   return EXIT_SUCCESS;
13. }
```

傳址(Call by Address)

顧名思義，函式呼叫採用**傳址**傳遞參數，是將變數的**記憶體位址**傳入函式。

那麼，請問此程式會印出什麼？

aswap.c

```
1.  void aswap(int *x, int *y) {
2.    int temp = *x;
3.    *x = *y;
4.    *y = temp;
5.  }
6.
7.  int main(void) {
8.    int num1 = 10, num2 = 5;
9.    printf("%d %d\n", num1, num2);
10.   aswap(&num1, &num2);
11.   printf("%d %d\n", num1, num2);
12.   return EXIT_SUCCESS;
13. }
```

傳址(Call by Address)

變數名稱	記憶體位址	資料
num2	2000	5
num1	2004	10

aswap.c

```
1.  void aswap(int *x, int *y) {
2.    int temp = *x;
3.    *x = *y;
4.    *y = temp;
5.  }
6.
7.  int main(void) {
8.    int num1 = 10, num2 = 5;
9.    printf("%d %d\n", num1, num2);
10.   aswap(&num1, &num2);
11.   printf("%d %d\n", num1, num2);
12.   return EXIT_SUCCESS;
13. }
```

傳址(Call by Address)

變數名稱	記憶體位址	資料
y	1012	2000
x	1020	2004
...	...	...
num2	2000	5
num1	2004	10

aswap.c

```
1.  void aswap(int *x, int *y) {
2.    int temp = *x;
3.    *x = *y;
4.    *y = temp;
5.  }
6.
7.  int main(void) {
8.    int num1 = 10, num2 = 5;
9.    printf("%d %d\n", num1, num2);
10.   aswap(&num1, &num2);
11.   printf("%d %d\n", num1, num2);
12.   return EXIT_SUCCESS;
13. }
```

傳址(Call by Address)

變數名稱	記憶體位址	資料
temp	1000	10
...	...	...
y	1012	2000
x	1020	2004
...	...	...
num2	2000	5
num1	2004	10

aswap.c

```
1.  void aswap(int *x, int *y) {
2.    int temp = *x;
3.    *x = *y;
4.    *y = temp;
5.  }
6.
7.  int main(void) {
8.    int num1 = 10, num2 = 5;
9.    printf("%d %d\n", num1, num2);
10.   aswap(&num1, &num2);
11.   printf("%d %d\n", num1, num2);
12.   return EXIT_SUCCESS;
13. }
```

傳址(Call by Address)

變數名稱	記憶體位址	資料
temp	1000	10
...	...	...
y	1012	2000
x	1020	2004
...	...	...
num2	2000	5
num1	2004	5

aswap.c

```
1.  void aswap(int *x, int *y) {
2.    int temp = *x;
3.    *x = *y;
4.    *y = temp;
5.  }
6.
7.  int main(void) {
8.    int num1 = 10, num2 = 5;
9.    printf("%d %d\n", num1, num2);
10.   aswap(&num1, &num2);
11.   printf("%d %d\n", num1, num2);
12.   return EXIT_SUCCESS;
13. }
```

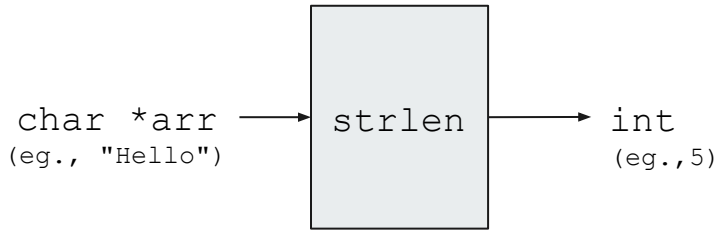
傳址(Call by Address)

變數名稱	記憶體位址	資料
temp	1000	10
...	...	...
y	1012	2000
x	1020	2004
...	...	...
num2	2000	10
num1	2004	5

aswap.c

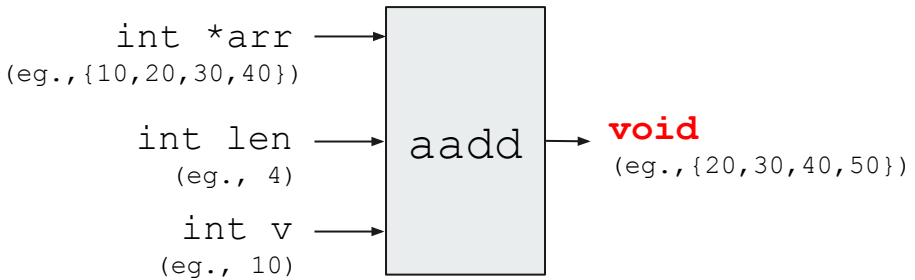
```
1.  void aswap(int *x, int *y) {
2.    int temp = *x;
3.    *x = *y;
4.    *y = temp;
5.  }
6.
7.  int main(void) {
8.    int num1 = 10, num2 = 5;
9.    printf("%d %d\n", num1, num2);
10.   aswap(&num1, &num2);
11.   printf("%d %d\n", num1, num2);
12.   return EXIT_SUCCESS;
13. }
```

傳址(Call by Address):計算字串長度函式




```
1.  int strlen(char *str) {
2.    int len = 0;
3.    //利用指標計算字串長度
4.    return len;
5.  }
6.
7.  int main() {
8.    char name[] = "Hello";
9.    int len = strlen(name);
10.   printf("strlen(%s) = %d\n", name, len);
11.   return EXIT_SUCCESS;
12. }
```

傳址(Call by Address):整數陣列常數相加




```
1.  void aadd(int *arr, int len, int v) {
2.    //利用指標完成整數陣列常數相加
3.  }
4.
5.  int main() {
6.    int score[] = {10, 20, 30, 40};
7.    aadd(score, 4, 10);
8.    int i;
9.    for(i=0;i<4;i++)
10.     printf("%d ", *(score+i));
11.   printf("\n");
12.   return EXIT_SUCCESS;
13. }
```

練習:泡沫排序

輸入不超過10個元素的整數陣列，輸出排序後的結果(小到大)

限制條件:
以函式實作泡沫排序
以指標存取陣列內的元素

可能用到的觀念
指標、傳址

輸入	輸出
2 1 3	1 2 3
31 22 12 42	12 22 31 42

A few moments later …

練習:印出子字串

輸入(1)字串、(2)子字串起始位置、與(3)子字串的長度後印出指定的子字串。

限制條件:
以函式實作印出子字串
以指標存取陣列內的元素

可能用到的觀念
指標、傳址、fgets函式

```
char input[100];
fgets(input, 100, stdin);
```

輸入	輸出
hello 0 4	hell
Hello World 4 5	o Wor

A few moments later …

動態配置記憶體

Dynamic Memory Allocation

複習:靜態記憶體配置

陣列(array)是一種**靜態**的記憶體配置，因為在**編譯時期(compile time)**就會配置所需的記憶空間，所以無法改變大小。

```
int stdId[50];    //儲存50個學生的學號
```

但是如果只有10個學生？ 浪費了40個整數的記憶體空間
或是如果超過50個學生？ 沒有足夠的空間儲存

怎麼解決？

動態記憶體配置

C語言的動態記憶體配置(dynamic memory allocation)允許程式依據需要在**執行階段(run time)**配置所需的記憶體空間。

配置記憶體函式(`size`的單位為位元組)

```
void *malloc(size_t size); // memory allocation
```

釋放記憶體函式(**非常重要!**)

```
void free(void *ptr);
```

動態記憶體配置(續)

例如，假設只需要5個整數的儲存空間

```
1.  //配置可儲存5個整數的記憶體空間
2.  int *ptr = NULL;
3.  ptr = (int*) malloc(5 * sizeof(int));
4.  memset(ptr, 0, 5 * sizeof(int));
5.  *(ptr+2) = 80;
6.  free(ptr);
```

變數名稱	記憶體位址	資料
ptr	2000	NULL

動態記憶體配置(續)

例如，假設只需要5個整數的儲存空間

```
1.  //配置可儲存5個整數的記憶體空間
2.  int *ptr = NULL;
3.  ptr = (int*) malloc(5 * sizeof(int));
4.  memset(ptr, 0, 5 * sizeof(int));
5.  printf("%d\n", *(ptr+2));
6.  *(ptr+2) = 80;
7.  printf("%d\n", *(ptr+2));
8.  free(ptr);
```

變數名稱	記憶體位址	資料
	1012	?
	1016	?
	1020	?
	1024	?
	1028	?
...	...	...
ptr	2000	1012

動態記憶體配置(續)

例如，假設只需要5個整數的儲存空間

```
1.  //配置可儲存5個整數的記憶體空間
2.  int *ptr = NULL;
3.  ptr = (int*) malloc(5 * sizeof(int));
4.  memset(ptr, 0, 5 * sizeof(int));
5.  printf("%d\n", *(ptr+2));
6.  *(ptr+2) = 80;
7.  printf("%d\n", *(ptr+2));
8.  free(ptr);
```

變數名稱	記憶體位址	資料
	1012	0
	1016	0
	1020	0
	1024	0
	1028	0
...	...	...
ptr	2000	1012

動態記憶體配置(續)

例如，假設只需要5個整數的儲存空間

```
1.  //配置可儲存5個整數的記憶體空間
2.  int *ptr = NULL;
3.  ptr = (int*) malloc(5 * sizeof(int));
4.  memset(ptr, 0, 5 * sizeof(int));
5.  printf("%d\n", *(ptr+2));
6.  *(ptr+2) = 80;
7.  printf("%d\n", *(ptr+2));
8.  free(ptr);
```

變數名稱	記憶體位址	資料
	1012	0
	1016	0
	1020	80
	1024	0
	1028	0
...	...	...
ptr	2000	1012

動態記憶體配置(續)

例如，假設只需要5個整數的儲存空間

```
1.  //配置可儲存5個整數的記憶體空間
2.  int *ptr = NULL;
3.  ptr = (int*) malloc(5 * sizeof(int));
4.  memset(ptr, 0, 5 * sizeof(int));
5.  printf("%d\n", *(ptr+2));
6.  *(ptr+2) = 80;
7.  printf("%d\n", *(ptr+2));
8.  free(ptr); //記憶體釋放後才可被重新配置
```

變數名稱	記憶體位址	資料
	1012	0
	1016	0
	1020	80
	1024	0
	1028	0
...	...	...
	2000	

練習:學生成績計算

輸入學生數n, 動態產生3個陣列儲存學生姓名(字串)與程式設計成績(整數)與資工導論成績(整數)。印出每個學生的各科成績與平均成績以及各科的班平均。

限制條件:
動態配置記憶體

可能用到的觀念
指標、動態配置記憶體

輸入	輸出
2 Bob 100 90 Alice 90 90	Bob 100 90 95 Alice 90 90 90 95 90 92
3 Bob 100 80 Alice 90 90 John 80 80	Bob 100 80 90 Alice 90 90 90 John 80 80 80 90 83 86

A few moments later …

不釋放記憶體會發生什麼事?

```
1.  int *ptr = NULL;
2.  ptr = (int*) malloc(2 * sizeof(int));
3.  *(ptr+1) = 80;
4.  ptr = (int*) malloc(2 * sizeof(int));
```

變數名稱	記憶體位址	資料
ptr	1000	NULL

不釋放記憶體會發生什麼事? (續)

```
1.  int *ptr = NULL;
2.  ptr = (int*) malloc(2 * sizeof(int));
3.  *(ptr+1) = 80;
4.  ptr = (int*) malloc(2 * sizeof(int));
```

變數名稱	記憶體位址	資料
	1500	?
	1504	?
...	...	...
ptr	1000	1500

不釋放記憶體會發生什麼事? (續)

```
1.  int *ptr = NULL;
2.  ptr = (int*) malloc(2 * sizeof(int));
3.  *(ptr+1) = 80;
4.  ptr = (int*) malloc(2 * sizeof(int));
```

變數名稱	記憶體位址	資料
	1500	?
	1504	80
...	...	...
ptr	1000	1500

不釋放記憶體會發生什麼事? (續)

```
1.   int *ptr = NULL;
2.   ptr = (int*) malloc(2 * sizeof(int));
3.   *(ptr+1) = 80;
4.   ptr = (int*) malloc(2 * sizeof(int));
```

變數名稱	記憶體位址	資料
	1000	?
	1004	?
...	...	...
	1500	?
	1504	80
...	...	...
ptr	1000	1000

沒有指標指向這塊未被釋放 (free)的記憶體空間，所以在此程式結束之前都無法被重新配置。這個現象稱為**記憶體洩漏(memory leak)**。

指標陣列
Array of Pointers

指標陣列

指標陣列就是用來儲存指標變數的陣列，可以下列方式定義

```
1.  //宣告4個指向字串的指標
2.  char *suit[4] = {"Spade","Heart","Diamond","Club"};
```

請問這個指標陣列是如何被儲存在記憶體中?

指標陣列(續)

想想看為什麼指標陣列在記憶體長這樣?

```
1.  char *suit[4] = {"Spade","Heart","Diamond","Club"};
2.  printf("%p\n", suit);            //印出?
3.  printf("%p\n", suit+1);          //印出?
4.  printf("%p\n", *(suit+2));       //印出?
5.  printf("%s\n", *(suit+2));       //印出?
6.  printf("%p\n", suit[3]);         //印出?
7.  printf("%s\n", suit[3]);         //印出?
8.  printf("%c\n", suit[1][2]);      //印出?
9.  printf("%c\n", *(suit+1)[2]);    //印出?
10. printf("%c\n", (*(suit+1))[2]); //印出?
```

變數名稱	記憶體位址	資料
	1000	"Spade"
	1006	"Heart"
	1012	"Diamond"
	1020	"Club"
...	...	...
suit	3000	1000
	3008	1006
	3016	1012
	3024	1020

練習

撰寫程式印出右方表格的 ?處以驗證指標陣列的儲存方式

可能用到的觀念

指標陣列

變數名稱	記憶體位址	資料
	?	"Spade"
	?	"Heart"
	?	"Diamond"
	?	"Club"
...	...	...
suit	?	?
	?	?
	?	?
	?	?

A few moments later …

命令列參數

Command Line Arguments

命令列參數

如何在執行程式時給予參數?

例如, 求n!

```
$ ./fact 5
120
```

或求m到n之間所有數的總和

```
$ ./sum 5 7
18
```

回來看看main()的傳入參數

傳入參數：int argc　　命令列參數個數(包含執行檔)
傳入參數:char *argv[]　命令列參數

```
1.  int main(int argc, char *argv[]) {
2.    printf("%d\n", argc);
3.    int i;
4.    for(i=0;i<argc;i++) {
5.      printf("%s\n", *(argv+i));
6.    }
7.    return EXIT_SUCCESS;
8.  }
```

```
$ ./a.out you can't pass
```

變數名稱	記憶體位址	資料
	1000	"a.out"
	1006	"you"
	1010	"can't"
	1016	"pass"
...	...	...
argv	3000	1000
	3008	1006
	3016	1010
	3024	1016
argc	3032	4

練習:整數合

在執行程式時輸入兩個整數m與n，計算m到n之間所有整數和。

限制條件:
需使用命另列參數指標陣列

可能用到的觀念
命列列參數指標陣列

輸入	輸出
./a.out 3 1	NO
./a.out 2 4	9
./a.out -1 3	5

A few moments later …

函式指標

Function Pointer

函式指標

首先，要知道的是函式必須載入到記憶體後才能被呼叫。

顧名思義，函式指標指向的是**函式**在記憶體中的位址。

```
1.   int add(int x, int y) {return x + y;};
2.   int (*fun)(int x, int y) = NULL;
3.   fun = add;
```

變數名稱	記憶體位址	資料
add()	2000	program

函式指標(續)

首先，要知道的是函式必須載入到記憶體後才能被呼叫。

顧名思義，函式指標指向的是**函式**在記憶體中的位址。

```
1.   int add(int x, int y) {return x + y;};
2.   int (*fun)(int x, int y) = NULL;
3.   fun = add;
```

變數名稱	記憶體位址	資料
fun	1000	NULL
...	..	...
add()	2000	program

函式指標(續)

首先，要知道函式必須被載入到記憶體後才可以被呼叫。

顧名思義，函式指標指向的是**函式**在記憶體中的位址。

```
1.  int add(int x, int y) {return x + y;};
2.  int (*fun)(int x, int y) = NULL;
3.  fun = add;
4.  fun(10,5); //呼叫add(10,5)
```

變數名稱	記憶體位址	資料
fun	1000	2000
...	..	...
add()	2000	program

未使用函式指標

```
1.  int add(int x, int y){
2.    return x + y;
3.  }
4.
5.  int sub(int x, int y){
6.    return x - y;
7.  }
```

```
8.  int main() {
9.    int x, y, opt;
10.   scanf("%d %d", &x, &y);
11.   scanf("%d", &opt);
12.   switch(opt) {
13.   case 0:
14.     printf("%d\n", add(x, y)); break;
15.   case 1:
16.     printf("%d\n", sub(x, y)); break;
17.   }
18.   return EXIT_SUCCESS;
19. }
```

使用函式指標

```c
int add(int x, int y){
  return x + y;
}

int sub(int x, int y){
  return x - y;
}
//定義函式指標(介面要一樣)
int (*fun)(int x, int y);
```

```c
int main() {
  int x, y, opt;
  scanf("%d %d", &x, &y);
  scanf("%d", &opt);
  switch(opt) {
  case 0:
    fun = add;
    printf("%d\n", fun(x, y)); break;
  case 1:
    fun = sub;
    printf("%d\n", fun(x, y)); break;
  }
  return EXIT_SUCCESS;
}
```

好像有點脫褲子放屁?

使用函式指標陣列

```
1.   int add(int x, int y){
2.     return x + y;
3.   }
4.
5.   int sub(int x, int y){
6.     return x - y;
7.   }
8.   //定義函式指標陣列
9.   int (*fun[])(int x, int y) = {add, sub};
```

```
10.  int main() {
11.    int x, y, opt;
12.    scanf("%d %d", &x, &y);
13.    scanf("%d", &opt);
14.
15.    //當opt為0時，呼叫add()
16.    //當opt為1時，呼叫sub()
17.    printf("%d\n", fun[opt](x, y));
18.
19.    return EXIT_SUCCESS;
20.  }
```

Q & A

Computer History Museum, Mt. View, CA