CHAPTER 4

Arrays(ch6)

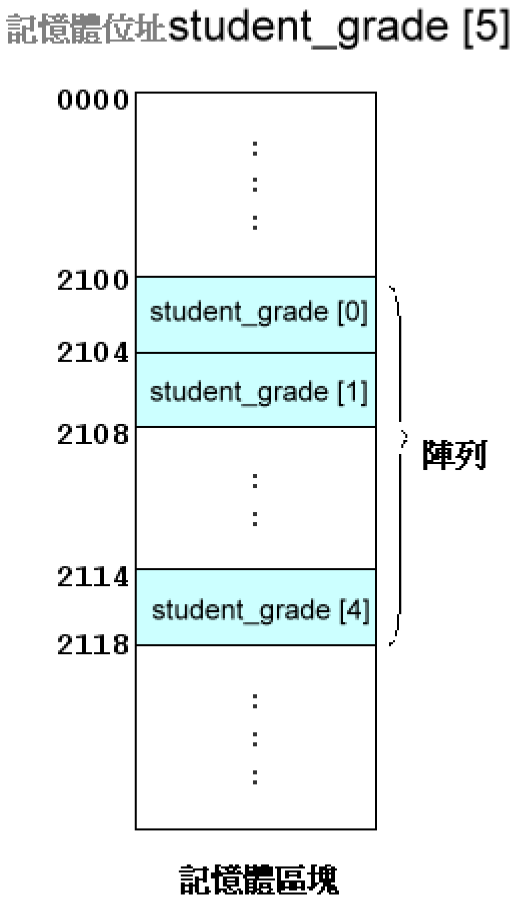
Arrays

- 陣列是一群具有相同名稱以及相同型別的記憶體位置
- 陣列 (Arrays) 是由相同型別的相關資料項所組成的資料結構，陣列主要儲存大量同性質資料，由於不需要使用不同的變數名稱，以及存取陣列元素的方便性，使得大多數的程式設計中，都看得到陣列的影子。
- 問題:宣告全班50人的成績
  - 如果沒有使用陣列需要宣告50個變數，才能使用50筆資料
    - int student0_grade; //代表學號0的學生
    - int student1_grade; //代表學號1的學生
    - ……
    - int student49_grade; //代表學號49的學生
  - 使用陣列，只需要宣告1個宣告，即可使用50筆資，料程式碼簡單且把聚集相同資料
    - int student_grade [50];
      - student_grade [0]; //代表學號0的學生
      - student_grade [1]; //代表學號1的學生
- 『陣列』與數學的「矩陣」非常類似。
  - 陣列中存放的每個資料稱之為元素(EX: student_grade [0])，相當於一個變數，我們只要透過索引(EX: [0], [1]..[49])，就可以直接取得陣列的指定元素
  - 使用陣列可以免除大量變數命名的問題，使得程式具有較高的可讀性

MMS Lab

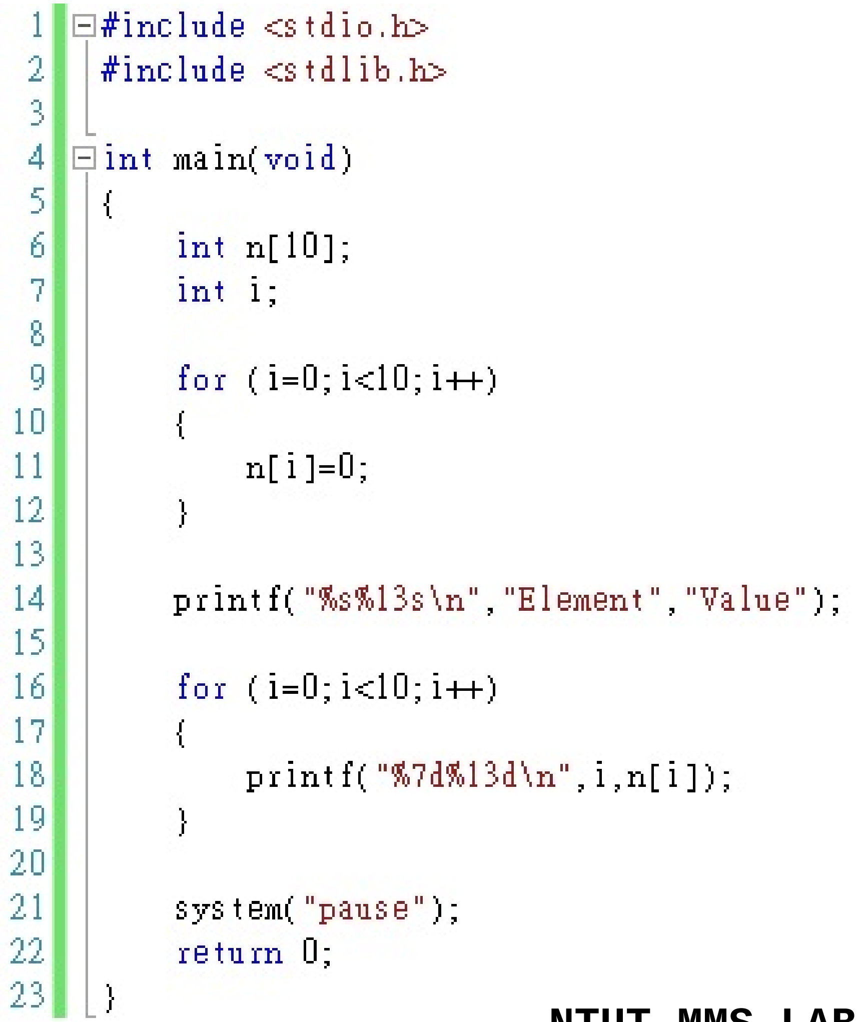
Arrays

- 宣告使用一維陣列
  - int student_grade [5];　　//5筆學生的成績(student_grade), 成績為整數(int)
  - char name[5]；　　//長度為5的姓名(name)字元陣列(char)
  - double student_weight[30]；　// 30筆學生的體重(student_weight), 體重為浮點數
- 每個陣列中的第一個元素均是第零個元素 (zeroth element)
  - student_grade [0] = 90;
    - 在student_grade [5]陣列中,學號0的學生, 分數為90分
  - student_grade [1] =95;
    - 在student_grade [5]陣列中,學號1的學生, 分數為95分
  - ….
  - student_grade [4] =88;
    - 在student_grade [5]陣列中,學號4的學生, 分數為88分

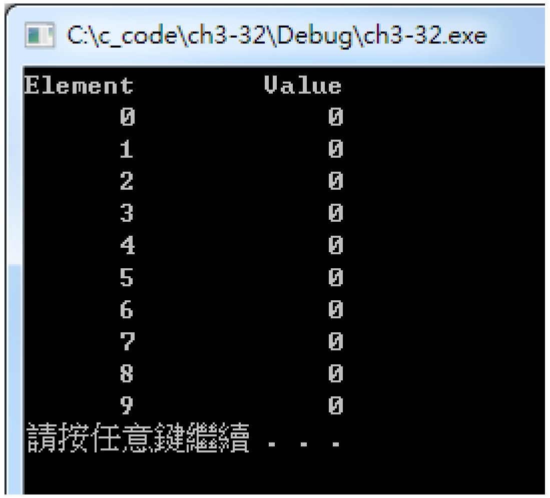


Arrays

- 基本練習，使用一維陣列
  - 陣列並不會自動地將初始值設定為零



```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int n[10];
    int i;

    for (i=0;i<10;i++)
    {
        n[i]=0;
    }

    printf("%s%13s\n","Element","Value");

    for (i=0;i<10;i++)
    {
        printf("%7d%13d\n",i,n[i]);
    }

    system("pause");
    return 0;
}
```



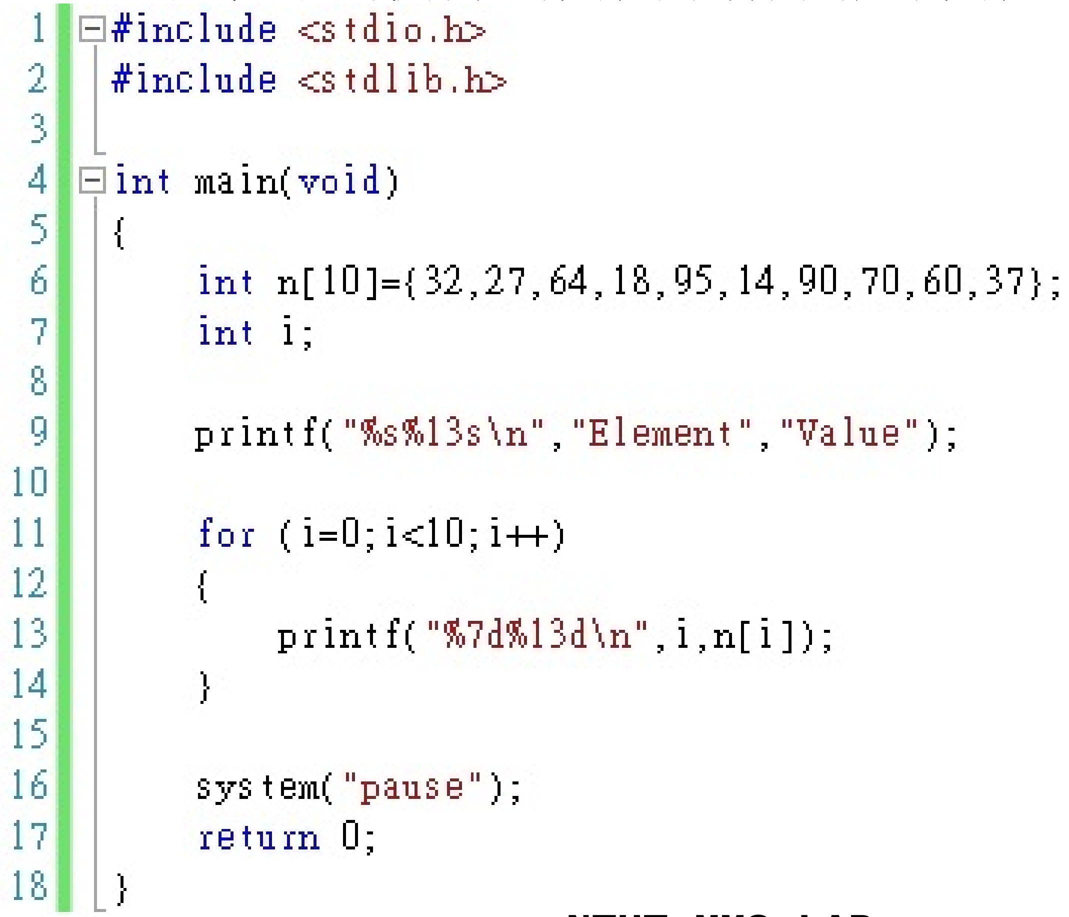
C:\c_code\ch3-32\Debug\ch3-32.exe

```
Element        Value
      0            0
      1            0
      2            0
      3            0
      4            0
      5            0
      6            0
      7            0
      8            0
      9            0
請按任意鍵繼續 . . .
```

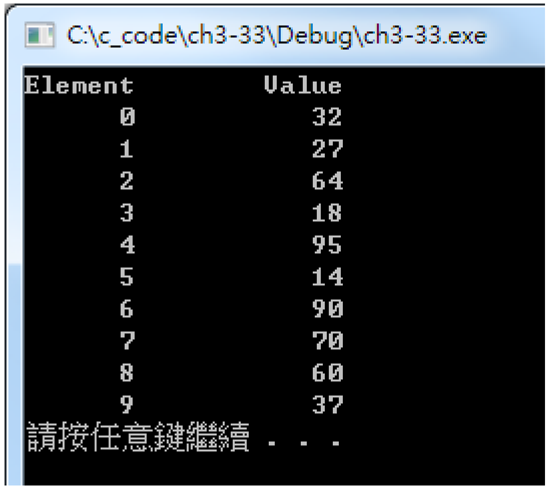
MMS Lab

Arrays

- 使用一維陣列，簡化初始值
  - int n[10] = {32, 27,64, 18, 95, 14, 95, 70, 60, 37}
  - int n[10] = {0}; //陣列中10個元素的初始值都為0
    - 如果給定的初始值的個數小於陣列的元素個數，則剩下的元素將自動指定初始值為零



```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int n[10]={32,27,64,18,95,14,90,70,60,37};
    int i;

    printf("%s%13s\n","Element","Value");

    for (i=0;i<10;i++)
    {
        printf("%7d%13d\n",i,n[i]);
    }

    system("pause");
    return 0;
}
```

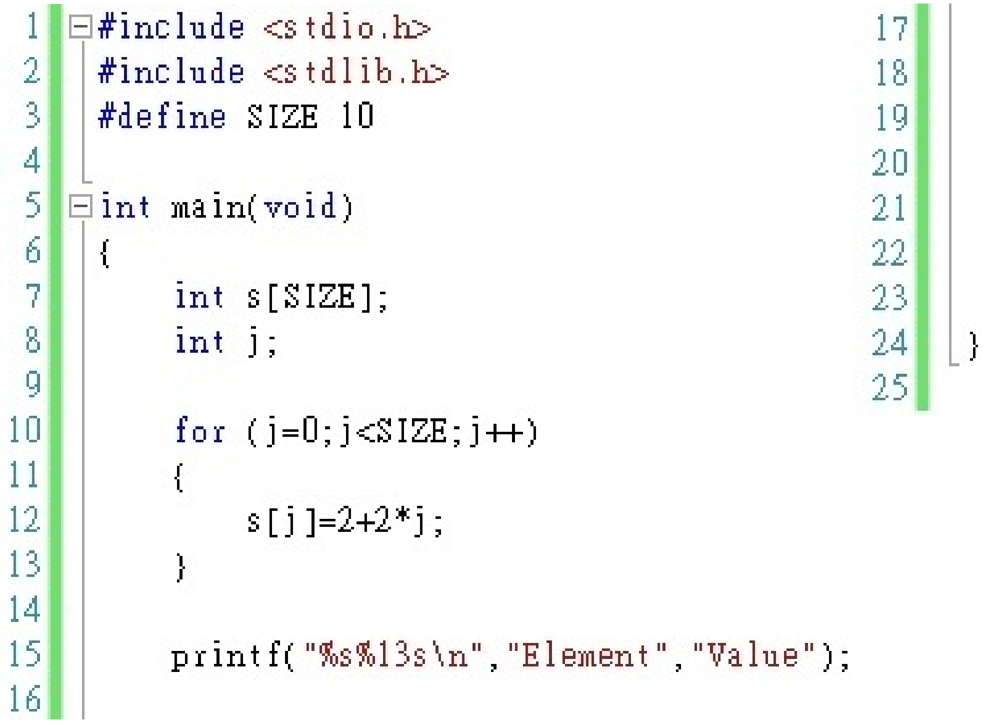


C:\c_code\ch3-33\Debug\ch3-33.exe

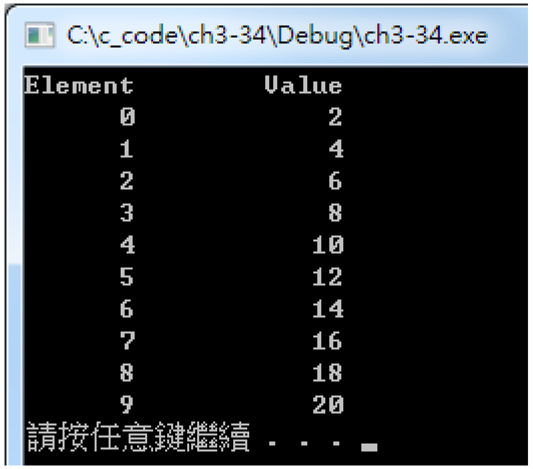
```
Element        Value
      0           32
      1           27
      2           64
      3           18
      4           95
      5           14
      6           90
      7           70
      8           60
      9           37
請按任意鍵繼續 . . .
```

Arrays

- 使用一維陣列，使用了#define前置處理器命令
  - #define SIZE 10 //注意沒有;
  - 可以依照不同的狀況，快速修改程式
    - 例如電子系有2個班級，兩個班的學生數不同，可以使用define快速的修改班級人數，寫一個程式，兩班皆可適用，讓程式具有擴充性



```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 10

int main(void)
{
        int s[SIZE];
        int j;

        for (j=0;j<SIZE;j++)
        {
                s[j]=2+2*j;
        }

        printf("%s%13s\n","Element","Value");
        for (j=0;j<SIZE;j++)
        {
                printf("%7d%13d\n",j,s[j]);
        }

        system("pause");
        return 0;
}
```



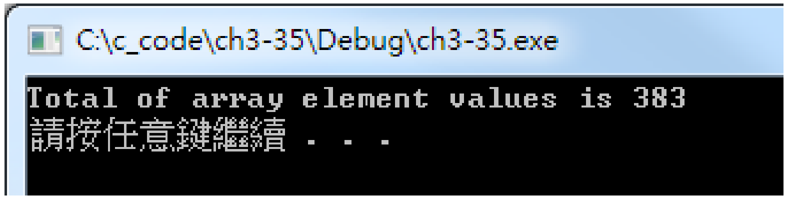
C:\c_code\ch3-34\Debug\ch3-34.exe

```
Element        Value
      0            2
      1            4
      2            6
      3            8
      4           10
      5           12
      6           14
      7           16
      8           18
      9           20
請按任意鍵繼續 . . .
```

Arrays

- 使用一維陣列，計算陣列的總和

```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 12

int main(void)
{
	int a[SIZE]={1,3,5,4,7,2,99,16,45,67,89,45};
	int i;
	int total=0;

	for (i=0;i<SIZE;i++)
	{
		total+=a[i];
	}

	printf("Total of array element values is %d\n",total);

	system("pause");
	return 0;
}
```

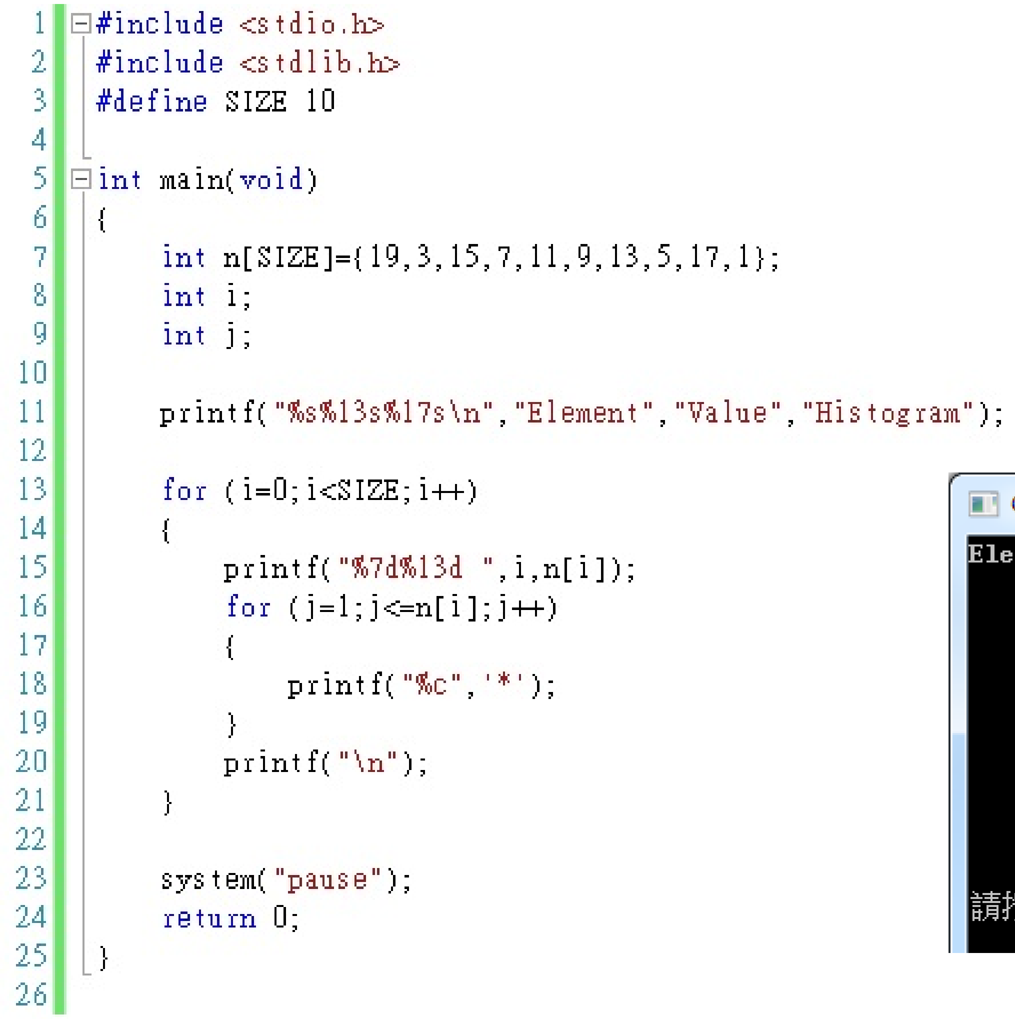


C:\c_code\ch3-35\Debug\ch3-35.exe

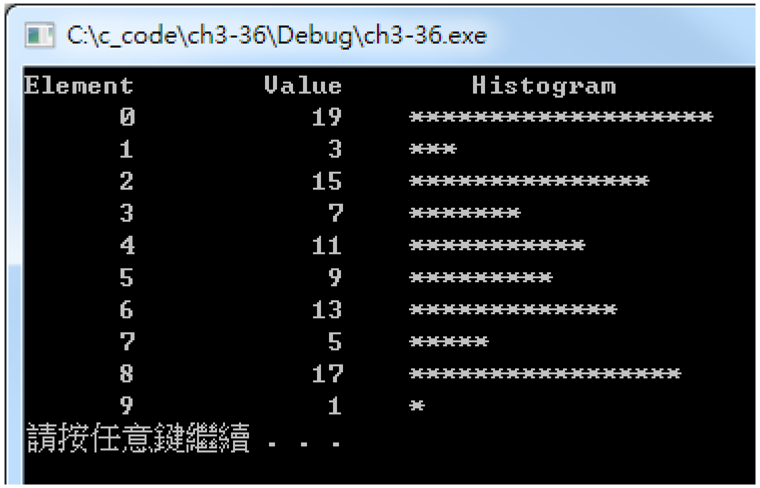
```
Total of array element values is 383
請按任意鍵繼續 . . .
```

Arrays

- 使用一維陣列，用星號圖型長短表示陣列元素的大小



```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 10

int main(void)
{
    int n[SIZE]={19,3,15,7,11,9,13,5,17,1};
    int i;
    int j;

    printf("%s%13s%17s\n","Element","Value","Histogram");

    for (i=0;i<SIZE;i++)
    {
        printf("%7d%13d ",i,n[i]);
        for (j=1;j<=n[i];j++)
        {
            printf("%c",'*');
        }
        printf("\n");
    }

    system("pause");
    return 0;
}
```

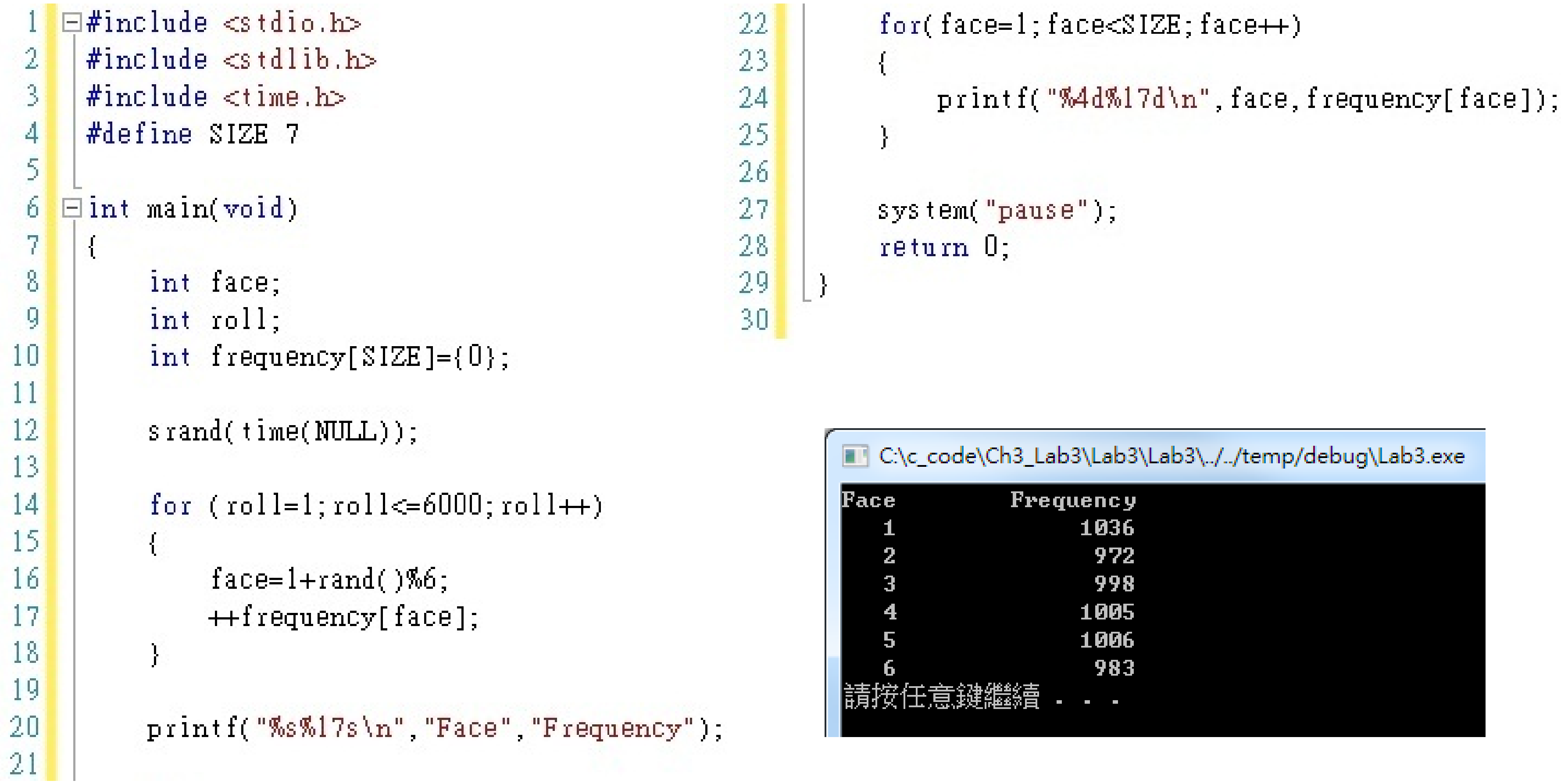


C:\c_code\ch3-36\Debug\ch3-36.exe

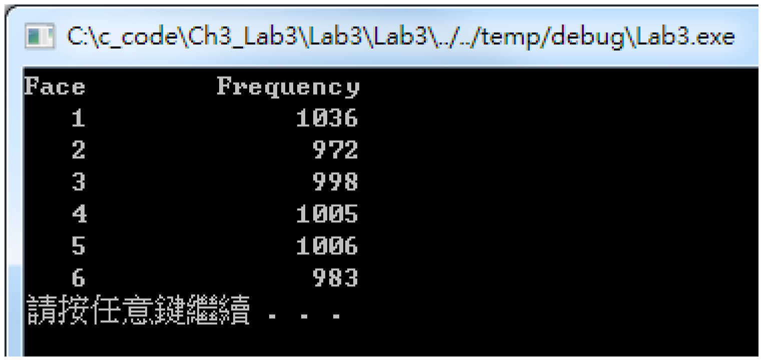
```
Element        Value        Histogram
      0           19    *******************
      1            3    ***
      2           15    ***************
      3            7    *******
      4           11    ***********
      5            9    *********
      6           13    *************
      7            5    *****
      8           17    *****************
      9            1    *
請按任意鍵繼續 . . .
```

Lab3

- 陣列版本的投擲一個六面的骰子6000次，計算每個面的個數



```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define SIZE 7

int main(void)
{
	int face;
	int roll;
	int frequency[SIZE]={0};

	srand(time(NULL));

	for (roll=1;roll<=6000;roll++)
	{
		face=1+rand()%6;
		++frequency[face];
	}

	printf("%s%17s\n","Face","Frequency");

	for(face=1;face<SIZE;face++)
	{
		printf("%4d%17d\n",face,frequency[face]);
	}

	system("pause");
	return 0;
}
```



C:\c_code\Ch3_Lab3\Lab3\Lab3\../../temp/debug\Lab3.exe

```
Face        Frequency
   1             1036
   2              972
   3              998
   4             1005
   5             1006
   6              983
請按任意鍵繼續 . . .
```

Arrays

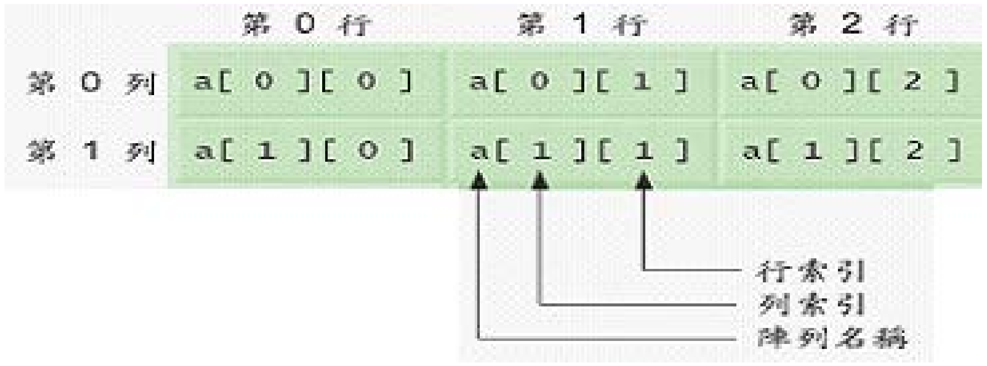
- 宣告使用二維陣列
  - EX:宣告整數 2x3 array，未設定初始值

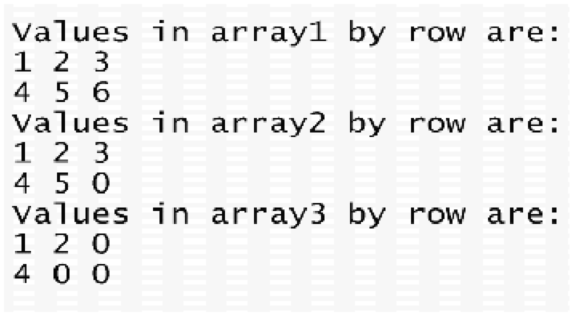
```
int a[ 2 ][ 3 ];
```

  - EX:宣告整數 2x3 array，設定初始值

```
int a[ 2 ][ 3 ] = { { 1, 2, 3 }, { 4, 5, 6 } };
int a[ 2 ][ 3 ] = { 1, 2, 3, 4, 5 };
int a[ 2 ][ 3 ] = { { 1, 2 }, { 4 } };
```

    - 如果某一列的初始值個數不夠的話，則此列剩下的元素會將初始值設定為零。





```
Values in array1 by row are:
1 2 3
4 5 6
Values in array2 by row are:
1 2 3
4 5 0
Values in array3 by row are:
1 2 0
4 0 0
```

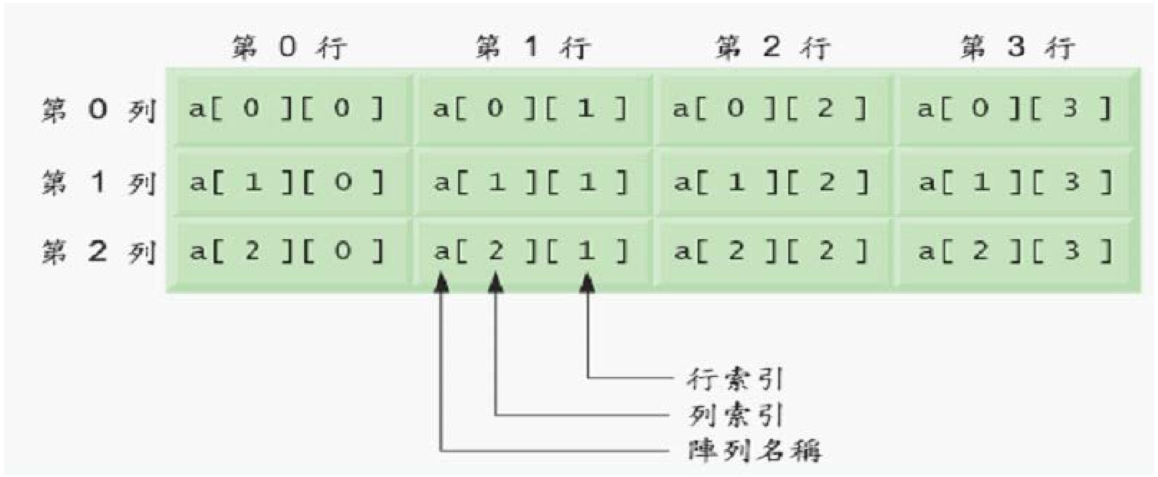
Arrays

- 使用二維陣列控制元素
  - EX:指定對第三列操作

```
for ( column = 0; column <= 3; column++ )
   {
      a[ 2 ][ column ] = 0;
   }
```

- 由於我們指定對第三列操作，因此第一個下標應該都是2 (0是第一列，1是第二列)。

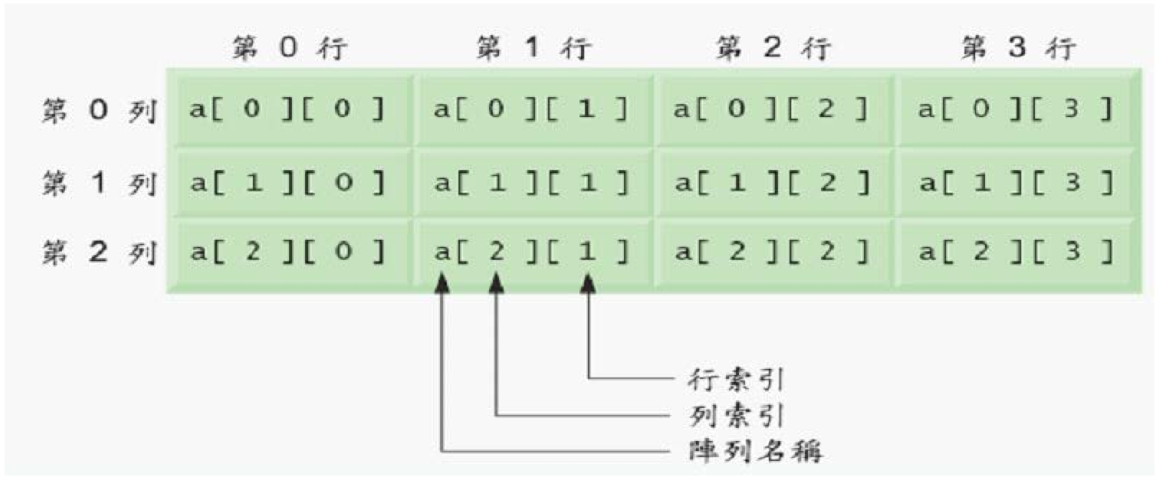
-
  ```
  a[ 2 ][ 0 ] = 0;
  a[ 2 ][ 1 ] = 0;
  a[ 2 ][ 2 ] = 0;
  a[ 2 ][ 3 ] = 0;
  ```



Arrays

- 二維陣列控制元素運算
- EX:將3x4二維陣列控制元素做加總

```
total = 0;
for ( row = 0; row <= 2; row++ )
{
   for ( column = 0; column <= 3; column++ )
   {
            total += a[ row ][ column ];
   }
}
```

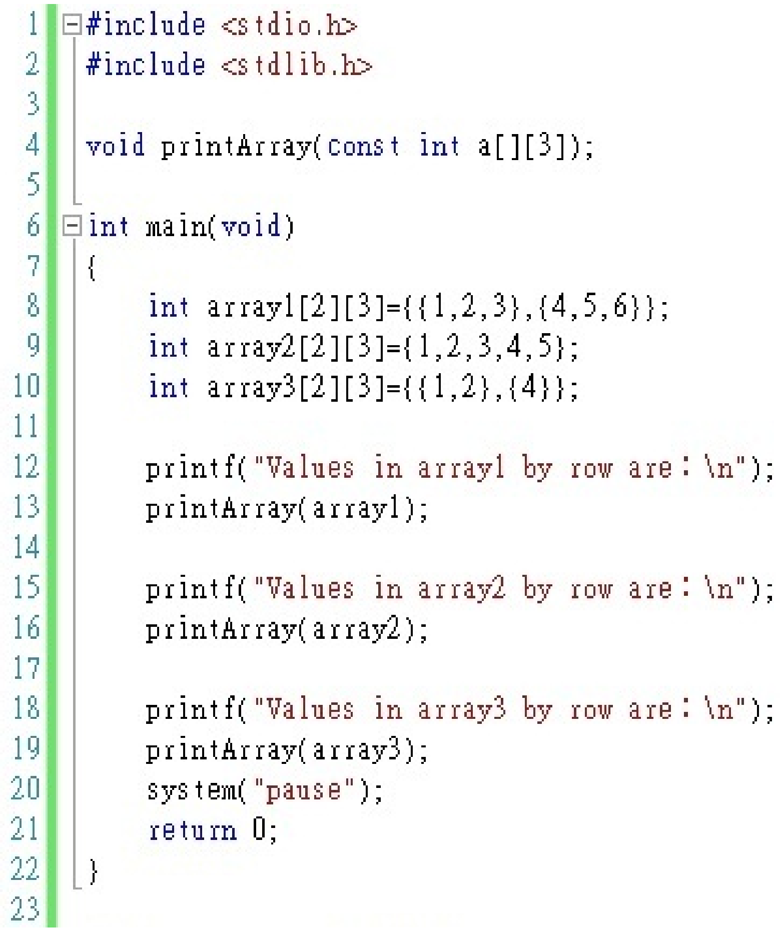


	第 0 行	第 1 行	第 2 行	第 3 行
第 0 列	a[0][0]	a[0][1]	a[0][2]	a[0][3]
第 1 列	a[1][0]	a[1][1]	a[1][2]	a[1][3]
第 2 列	a[2][0]	a[2][1]	a[2][2]	a[2][3]

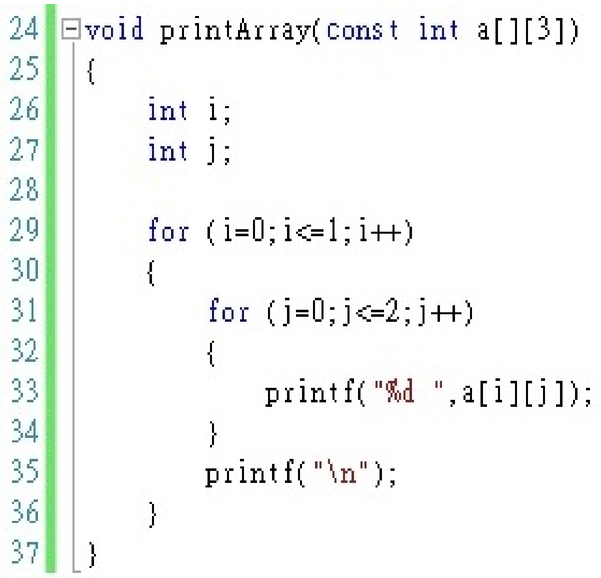
行索引
列索引
陣列名稱

Arrays

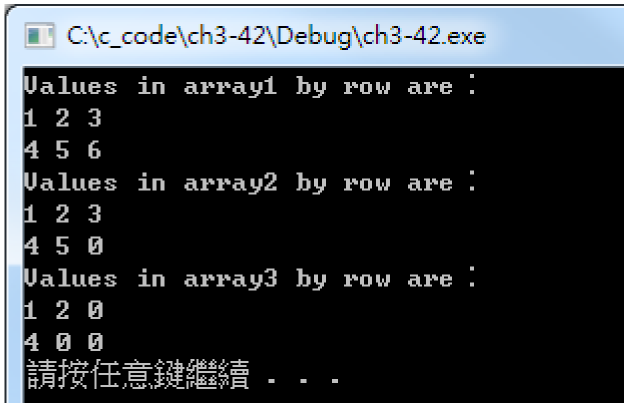
- 使用二維陣列



```
#include <stdio.h>
#include <stdlib.h>

void printArray(const int a[][3]);

int main(void)
{
    int array1[2][3]={{1,2,3},{4,5,6}};
    int array2[2][3]={1,2,3,4,5};
    int array3[2][3]={{1,2},{4}};

    printf("Values in array1 by row are:\n");
    printArray(array1);

    printf("Values in array2 by row are:\n");
    printArray(array2);

    printf("Values in array3 by row are:\n");
    printArray(array3);
    system("pause");
    return 0;
}

```



```
void printArray(const int a[][3])
{
    int i;
    int j;

    for (i=0;i<=1;i++)
    {
        for (j=0;j<=2;j++)
        {
            printf("%d ",a[i][j]);
        }
        printf("\n");
    }
}
```



C:\c_code\ch3-42\Debug\ch3-42.exe

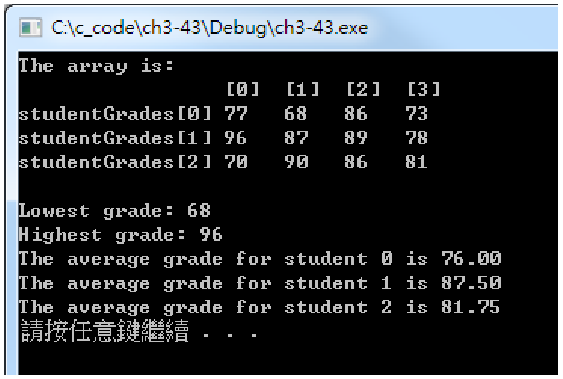
```
Values in array1 by row are:
1 2 3
4 5 6
Values in array2 by row are:
1 2 3
4 5 0
Values in array3 by row are:
1 2 0
4 0 0
請按任意鍵繼續 . . .
```

MMS Lab

Arrays

- 班上三位學生，每位學生有四個成績
  - 函式printArray: 以表列的方式清楚印出這個二維陣列
  - 函式minimum, maximum: 找出所有學生在本學期中的最低和最高成績
  - 函式average: 算出某位學生本學期的平均成績

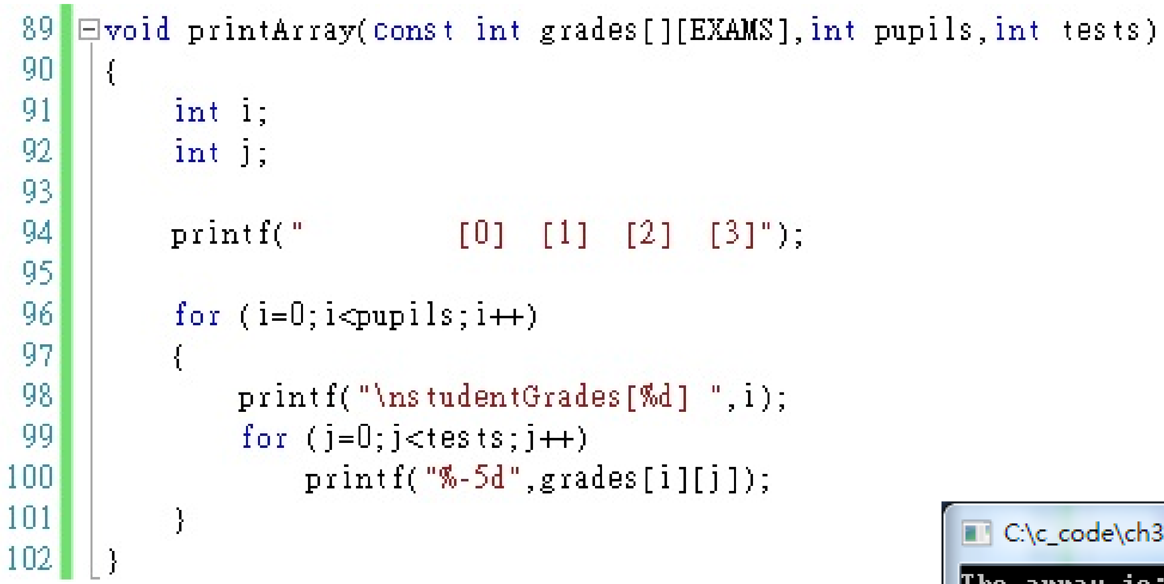
```c
#include <stdio.h>
#include <stdlib.h>
#define STUDENTS 3
#define EXAMS 4

int minimum(const int grades[][EXAMS],int pupils,int tests);
int maximum(const int grades[][EXAMS],int pupils,int tests);
double average(const int setOfGrades[],int tests);
void printArray(const int grades[][EXAMS],int pupils,int tests);

int main (void)
{
    int student;
    const int studentGrades[STUDENTS][EXAMS] =
    { {77,68,86,73},
      {96,87,89,78},
      {70,90,86,81} };

    printf("The array is:\n");
    printArray(studentGrades,STUDENTS,EXAMS);

    printf("\n\nLowest grade: %d\nHighest grade: %d\n",
        minimum(studentGrades,STUDENTS,EXAMS),
        maximum(studentGrades,STUDENTS,EXAMS));

    for (student=0;student<STUDENTS;student++)
    {
        printf("The average grade for student %d is %.2f\n",
            student,average(studentGrades[student],EXAMS));
    }

    system("pause");
    return 0;
}
```


C:\c_code\ch3-43\Debug\ch3-43.exe

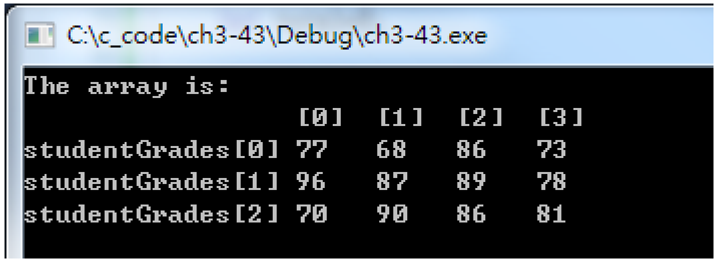
```
The array is:
                 [0]   [1]   [2]   [3]
studentGrades[0] 77    68    86    73
studentGrades[1] 96    87    89    78
studentGrades[2] 70    90    86    81

Lowest grade: 68
Highest grade: 96
The average grade for student 0 is 76.00
The average grade for student 1 is 87.50
The average grade for student 2 is 81.75
請按任意鍵繼續 . . .
```

Arrays

- 班上三位學生，每位學生有四個成績
  - 函式printArray: 以表列的方式清楚印出這個二維陣列



```c
void printArray(const int grades[][EXAMS],int pupils,int tests)
{
    int i;
    int j;

    printf("                 [0]  [1]  [2]  [3]");

    for (i=0;i<pupils;i++)
    {
        printf("\nstudentGrades[%d] ",i);
        for (j=0;j<tests;j++)
            printf("%-5d",grades[i][j]);
    }
}
```



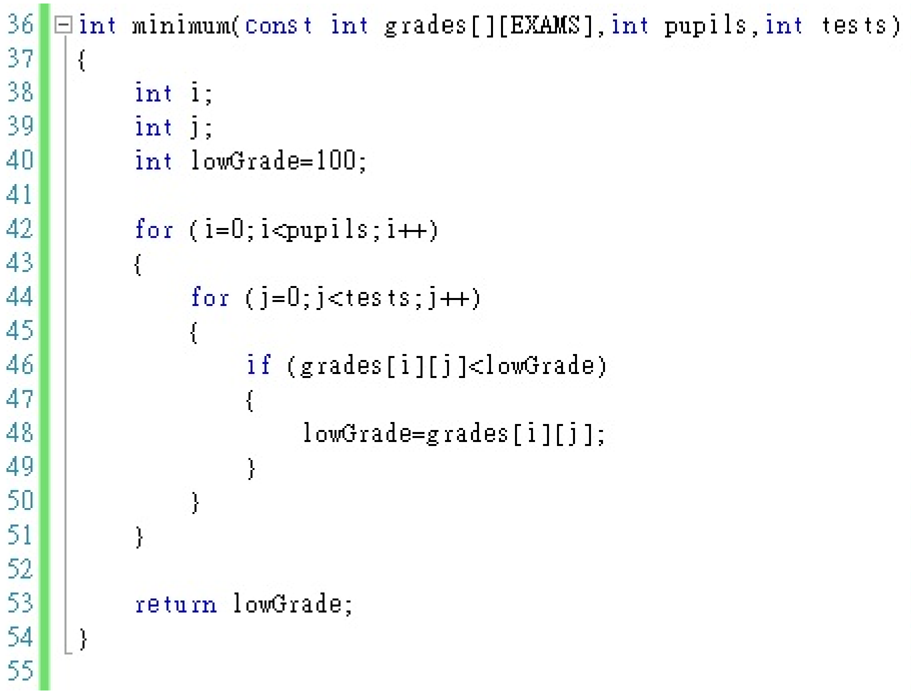
C:\c_code\ch3-43\Debug\ch3-43.exe

```
The array is:
                 [0]  [1]  [2]  [3]
studentGrades[0] 77   68   86   73
studentGrades[1] 96   87   89   78
studentGrades[2] 70   90   86   81
```

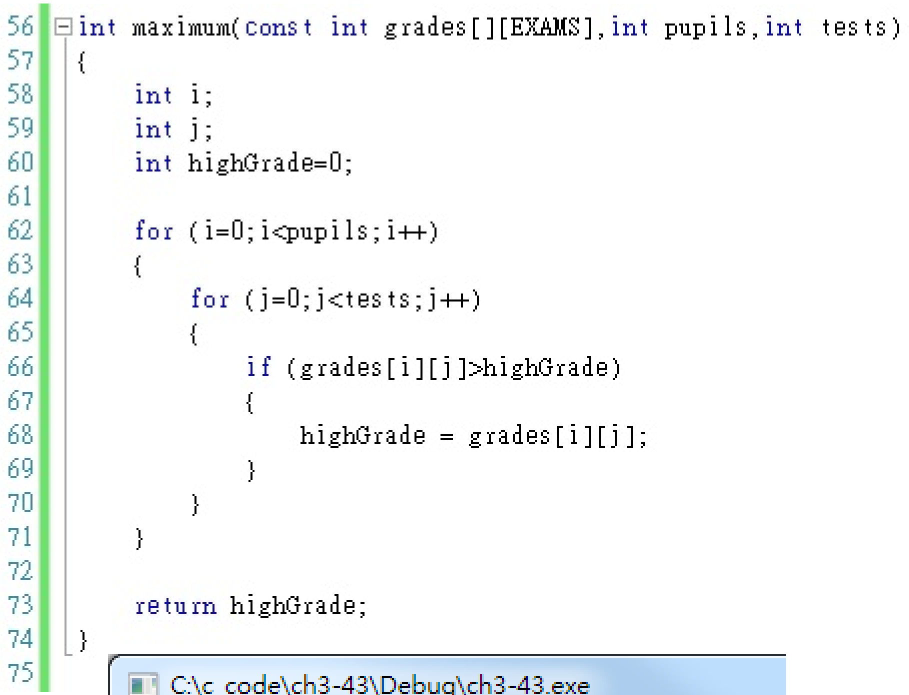
MMS Lab

Arrays

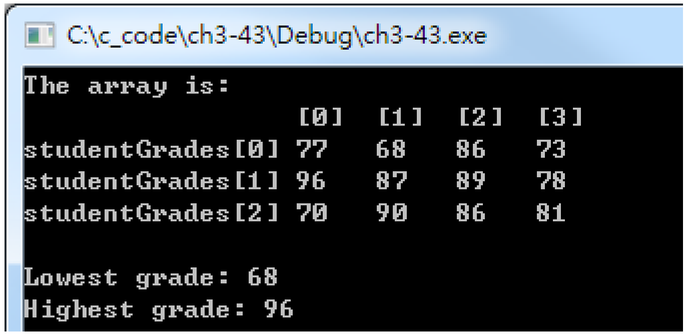
- 班上三位學生，每位學生有四個成績
  - 函式minimum, maximum: 找出所有學生在本學期中的最低和最高成績



```c
int minimum(const int grades[][EXAMS],int pupils,int tests)
{
    int i;
    int j;
    int lowGrade=100;

    for (i=0;i<pupils;i++)
    {
        for (j=0;j<tests;j++)
        {
            if (grades[i][j]<lowGrade)
            {
                lowGrade=grades[i][j];
            }
        }
    }

    return lowGrade;
}
```



```c
int maximum(const int grades[][EXAMS],int pupils,int tests)
{
    int i;
    int j;
    int highGrade=0;

    for (i=0;i<pupils;i++)
    {
        for (j=0;j<tests;j++)
        {
            if (grades[i][j]>highGrade)
            {
                highGrade = grades[i][j];
            }
        }
    }

    return highGrade;
}
```

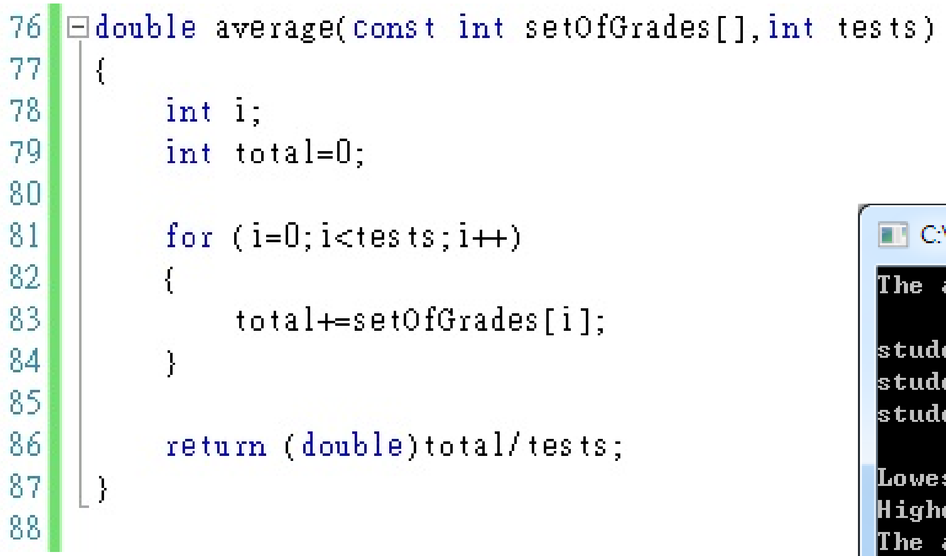


```
C:\c_code\ch3-43\Debug\ch3-43.exe

The array is:
                 [0]  [1]  [2]  [3]
studentGrades[0] 77   68   86   73
studentGrades[1] 96   87   89   78
studentGrades[2] 70   90   86   81

Lowest grade: 68
Highest grade: 96
```

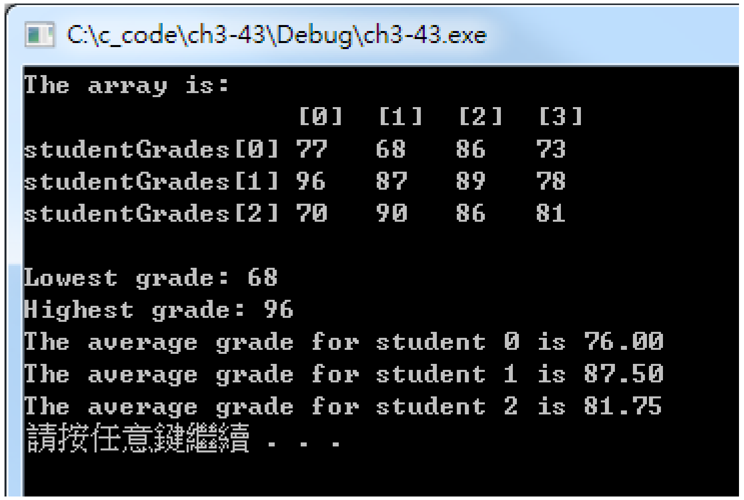
MMS Lab

Arrays

- 班上三位學生，每位學生有四個成績
  - 函式average: 算出某位學生本學期的平均成績



```
double average(const int setOfGrades[],int tests)
{
	int i;
	int total=0;

	for (i=0;i<tests;i++)
	{
		total+=setOfGrades[i];
	}

	return (double)total/tests;
}
```



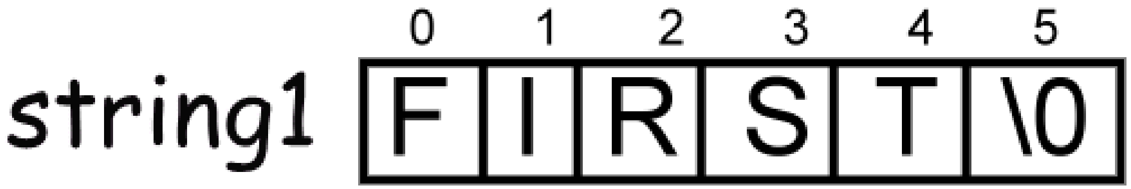
C:\c_code\ch3-43\Debug\ch3-43.exe

```
The array is:
                 [0]  [1]  [2]  [3]
studentGrades[0] 77   68   86   73
studentGrades[1] 96   87   89   78
studentGrades[2] 70   90   86   81

Lowest grade: 68
Highest grade: 96
The average grade for student 0 is 76.00
The average grade for student 1 is 87.50
The average grade for student 2 is 81.75
請按任意鍵繼續 . . .
```

Race: 增加學生人數
針對每個學生找出最高分和最低分的分數

Arrays

- 字串陣列
  - 例如: 將陣列string1的元素初始值設定為字串常數"first"中的各個字元
    - char string1[] = “FIRST";
    - char string1[] = { 'F', 'I', 'R', 'S', 'T', '\0' };
    - 陣列string1實際上含有6個元素
    - 空字元的字元常數表示法為'\0'
    - string1[0]是字元'f'，而string1[3]則是字元's'
    - 我們可以用printf和scanf ,轉換指定詞%s，直接從鍵盤輸入一個字串到字元陣列中, 直到遇到空字元'\0'為止
      - printf("%s", string1);
      - scanf(“%s”, string1); //不需要加上& string1

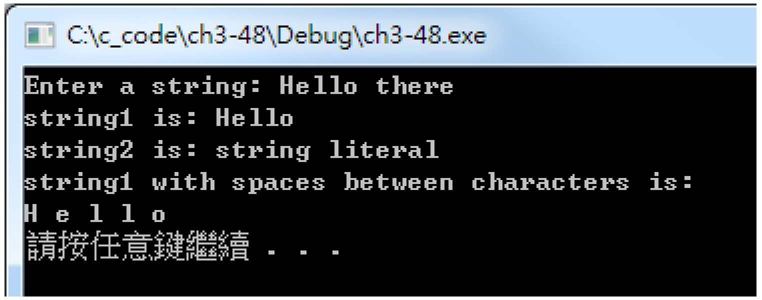


MMS Lab

Arrays

- 字串陣列
  - scanf函式會一直由鍵盤讀入字元，直到遇到第一個空白字元為止

```c
#include <stdio.h>
#include <stdlib.h>

int main (void)
{
    char string1[20];
    char string2[]="string literal";
    int i;

    printf("Enter a string: ");
    scanf("%s",string1);

    printf("string1 is: %s\nstring2 is: %s\n"
        "string1 with spaces between characters is: \n",
        string1,string2);

    for (i=0;string1[i] != '\0';i++)
        printf("%c ",string1[i]);

    printf("\n");
    system("pause");
    return 0;
}
```



C:\c_code\ch3-48\Debug\ch3-48.exe

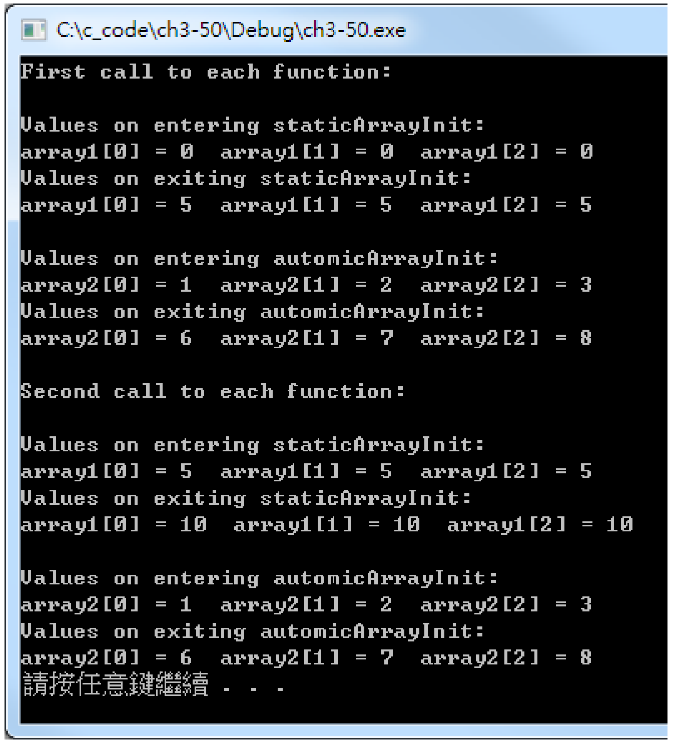
```
Enter a string: Hello there
string1 is: Hello
string2 is: string literal
string1 with spaces between characters is:
H e l l o
請按任意鍵繼續 . . .
```

Arrays

- 靜態陣列(static array)的使用
  - 宣告成static array會自動在編譯時期進行初始化
  - 假如你沒有明確地為static array設定初值，編譯器就會將陣列元素的初始值設定為零
  - static array仍保有上一次呼叫後的數值

Arrays

```c
#include <stdio.h>
#include <stdlib.h>

void staticArrayInit(void);
void automaticArrayInit(void);

int main (void)
{
    printf("First call to each function:\n");
    staticArrayInit();
    automaticArrayInit();

    printf("\n\nSecond call to each function:\n");
    staticArrayInit();
    automaticArrayInit();
    printf("\n");
    system("pause");
    return 0;
}

void staticArrayInit(void)
{
    static int array1[3];
    int i;

    printf("\nValues on entering staticArrayInit:\n");

    for (i=0;i<=2;i++)
        printf("array1[%d] = %d  ",i,array1[i]);

    printf("\nValues on exiting staticArrayInit:\n");

    for (i=0;i<=2;i++)
        printf("array1[%d] = %d  ",i,array1[i]+=5);
}

void automaticArrayInit(void)
{
    int array2[3]={1,2,3};
    int i;

    printf("\n\nValues on entering automicArrayInit:\n");

    for (i=0;i<=2;i++)
        printf("array2[%d] = %d  ",i,array2[i]);

    printf("\nValues on exiting automicArrayInit:\n");

    for (i=0;i<=2;i++)
        printf("array2[%d] = %d  ",i,array2[i]+=5);
}
```



```
C:\c_code\ch3-50\Debug\ch3-50.exe
First call to each function:

Values on entering staticArrayInit:
array1[0] = 0  array1[1] = 0  array1[2] = 0
Values on exiting staticArrayInit:
array1[0] = 5  array1[1] = 5  array1[2] = 5

Values on entering automicArrayInit:
array2[0] = 1  array2[1] = 2  array2[2] = 3
Values on exiting automicArrayInit:
array2[0] = 6  array2[1] = 7  array2[2] = 8

Second call to each function:

Values on entering staticArrayInit:
array1[0] = 5  array1[1] = 5  array1[2] = 5
Values on exiting staticArrayInit:
array1[0] = 10  array1[1] = 10  array1[2] = 10

Values on entering automicArrayInit:
array2[0] = 1  array2[1] = 2  array2[2] = 3
Values on exiting automicArrayInit:
array2[0] = 6  array2[1] = 7  array2[2] = 8
請按任意鍵繼續 . . .
```

MMS Lab

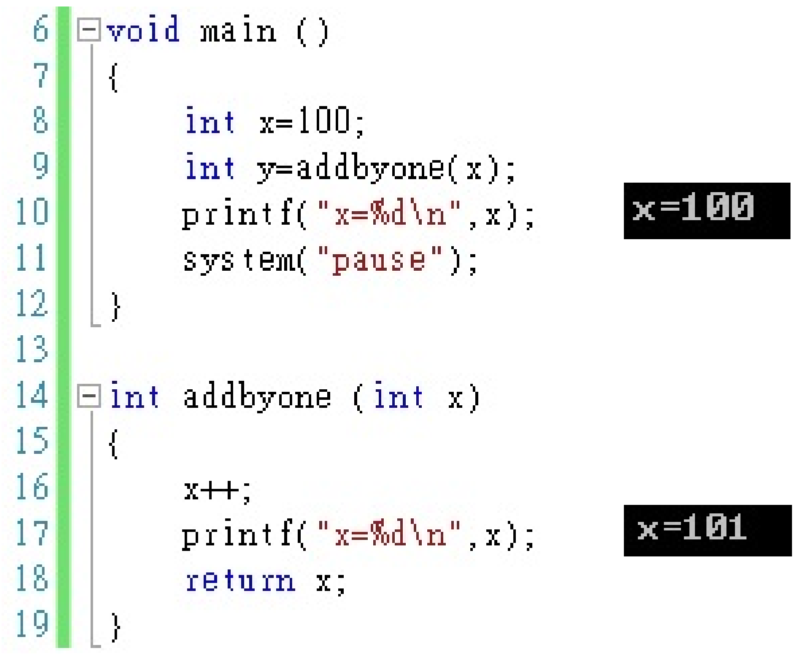
Arrays

- 函式的參數傳遞有三種
  - Call by value (傳值) //C和C++有支援
    - 函數呼叫: function(a, b)
    - 函數定義: void function(int x, int y)
  - Call by address (傳位址) //C和C++有支援
    - 函數呼叫: function(&a, &b)
    - 函數定義: void function(int * x, int *y)
  - Call by reference (傳參考) //只有C++有支援
    - 函數呼叫: function(a, b)
    - 函數定義: void function(int &x, int &y)
- Call by address (傳位址)和Call by reference (傳參考)具有相同結果， Call by reference (傳參考)主要簡化Call by address (傳位址)的符號運算

MMS Lab

Arrays

- 函式的參數傳遞有三種
  - Call by value
    - 函數呼叫: function(a, b)
    - 函數定義: void function(int x, int y)
    - 主要把數值拷貝到函示，函示與主程式的變數互不相干



```
void main ()
{
      int x=100;
      int y=addbyone(x);
      printf("x=%d\n",x);
      system("pause");
}

int addbyone (int x)
{
      x++;
      printf("x=%d\n",x);
      return x;
}
```

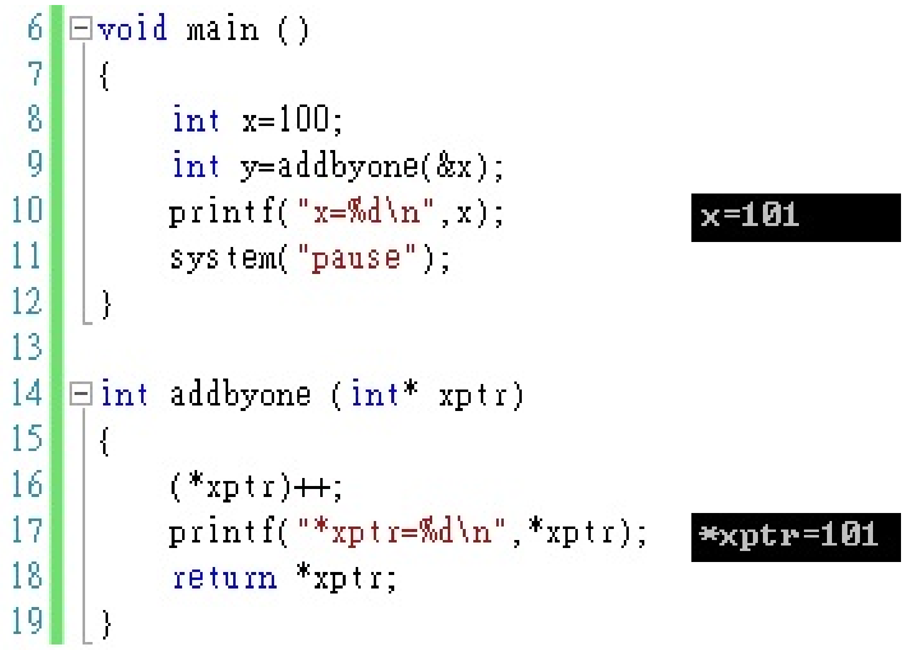
```
x=100
```

```
x=101
```

MMS Lab

Arrays

- 函式的參數傳遞有三種
  - Call by address
    - 函數呼叫: function(&a, &b)
    - 函數定義: void function(int * x, int *y)
    - 呼叫函數主要傳給函數位址(&x)，函數則以指標指導相對應的變數(*xptr)，函數運算會會更改相對應的變數內容



```
void main ()
{
    int x=100;
    int y=addbyone(&x);
    printf("x=%d\n",x);
    system("pause");
}

int addbyone (int* xptr)
{
    (*xptr)++;
    printf("*xptr=%d\n",*xptr);
    return *xptr;
}
```

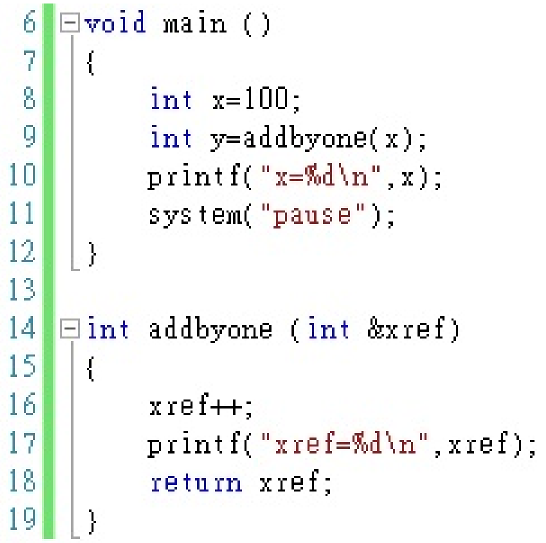
```
x=101
```

```
*xptr=101
```

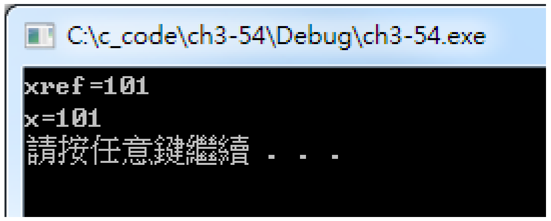
MMS Lab

Arrays

- 函式的參數傳遞有三種
  - Call by reference
    - 函數呼叫: function(a, b)
    - 函數定義: void function(int &x, int &y)
    - 呼叫函數主要傳給函數參考變數或物件(x)，函數會以位址(&xref)建立起相連等號，並表示使用相同記憶體空間，函數運算會會更改相對應的變數內容
    - 因為C沒有支援, 需要把main.c改成main.cpp



```
void main ()
{
      int x=100;
      int y=addbyone(x);
      printf("x=%d\n",x);
      system("pause");
}

int addbyone (int &xref)
{
      xref++;
      printf("xref=%d\n",xref);
      return xref;
}
```



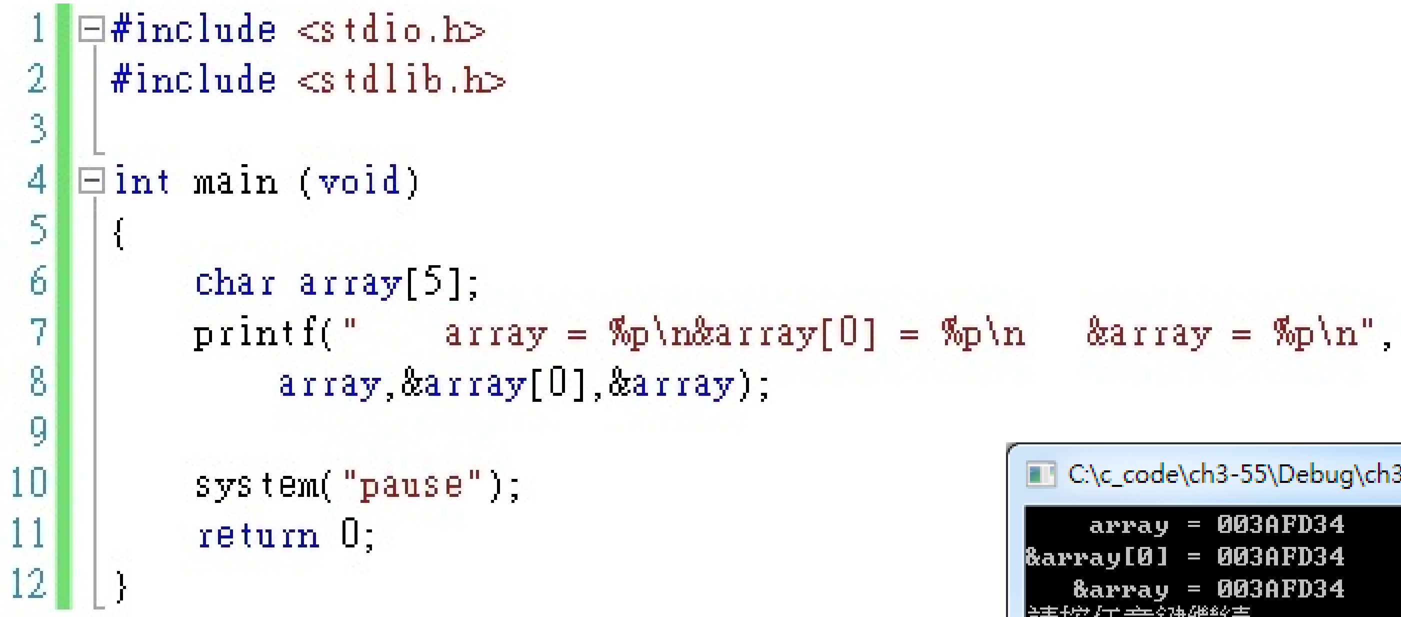
C:\c_code\ch3-54\Debug\ch3-54.exe

```
xref=101
x=101
請按任意鍵繼續 . . .
```

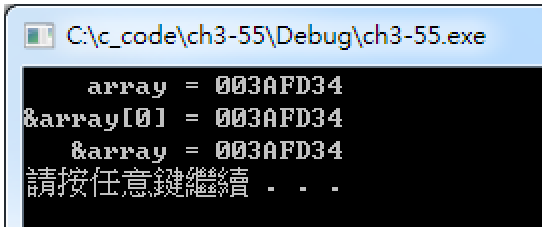
MMS Lab

Arrays

- 程式利用%p轉換指定詞 (一個用來列印位址的特殊轉換指定詞) 印出array，&array[0]和&array，來驗證陣列名稱確實是此陣列第一個元素所在的位址。
- %p轉換指定詞通常會將位址以十六進制數的形式印出來。



```
#include <stdio.h>
#include <stdlib.h>

int main (void)
{
    char array[5];
    printf("     array = %p\n&array[0] = %p\n   &array = %p\n",
        array,&array[0],&array);

    system("pause");
    return 0;
}
```



C:\c_code\ch3-55\Debug\ch3-55.exe

```
    array = 003AFD34
&array[0] = 003AFD34
   &array = 003AFD34
請按任意鍵繼續 . . .
```

Arrays

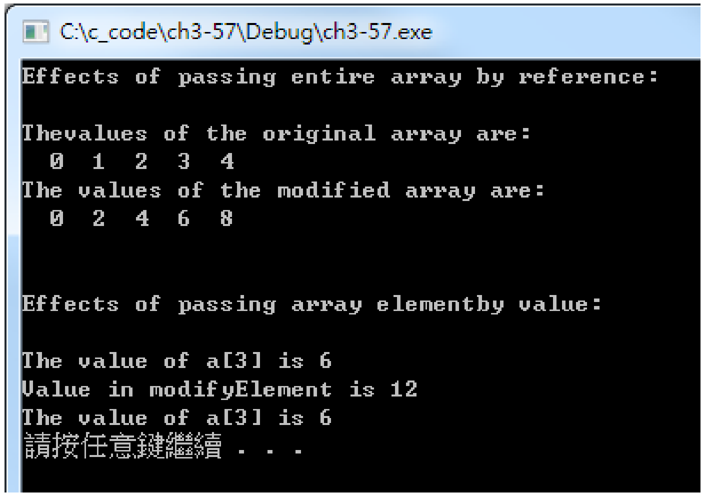
- 傳遞陣列引數給函式
  - 陣列(a[5])自動以Call by reference (傳參考) 來呼叫傳遞
    - 函數呼叫: **`modifyArray(a)`**
    - 函數定義: **`void modifyArray(int b[])`**
      - 參數b接收一個整數陣列
      - 陣列的中括號裡不需要指定陣列的大小

Arrays

- 傳遞陣列引數給函式

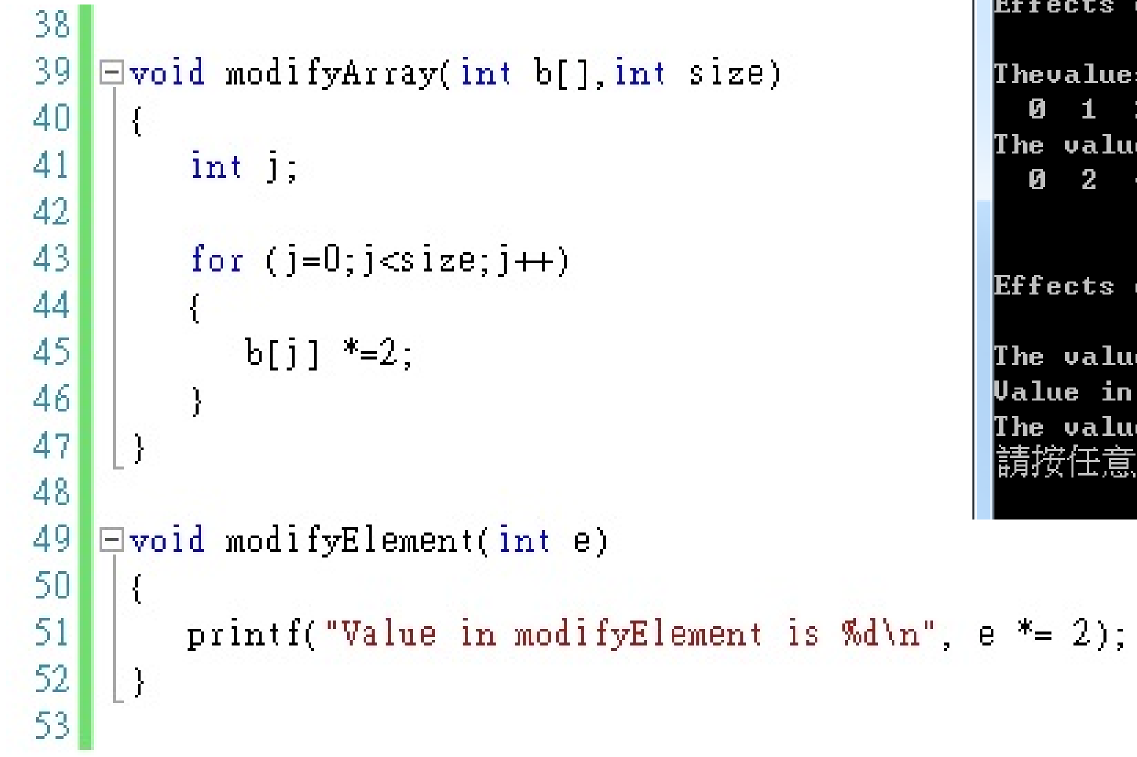
```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5

void modifyArray(int b[], int size);
void modifyElement(int e);

int main( void )
{
    int a[SIZE] = {0,1,2,3,4};
    int i;

    printf("Effects of passing entire array by reference:\n\nThe"
       "values of the original array are:\n");

    for (i=0;i<SIZE;i++)
    {
        printf("%3d",a[i]);
    }
    printf("\n");

    modifyArray(a, SIZE);
    printf("The values of the modified array are:\n");
    for (i=0;i<SIZE;i++)
    {
        printf("%3d",a[i]);
    }

    printf("\n\n\nEffects of passing array element"
       "by value:\n\nThe value of a[3] is %d\n",a[3]);

    modifyElement(a[3]);
    printf("The value of a[3] is %d\n", a[3]);

    system("pause");
    return 0;
}
```

Call by value

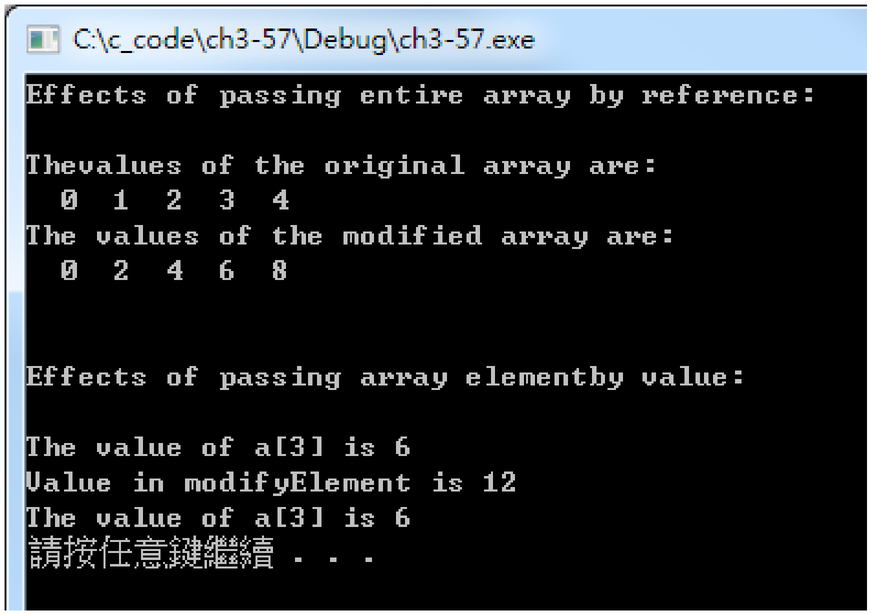


```
C:\c_code\ch3-57\Debug\ch3-57.exe
Effects of passing entire array by reference:

Thevalues of the original array are:
  0  1  2  3  4
The values of the modified array are:
  0  2  4  6  8


Effects of passing array elementby value:

The value of a[3] is 6
Value in modifyElement is 12
The value of a[3] is 6
請按任意鍵繼續 . . .
```

Arrays



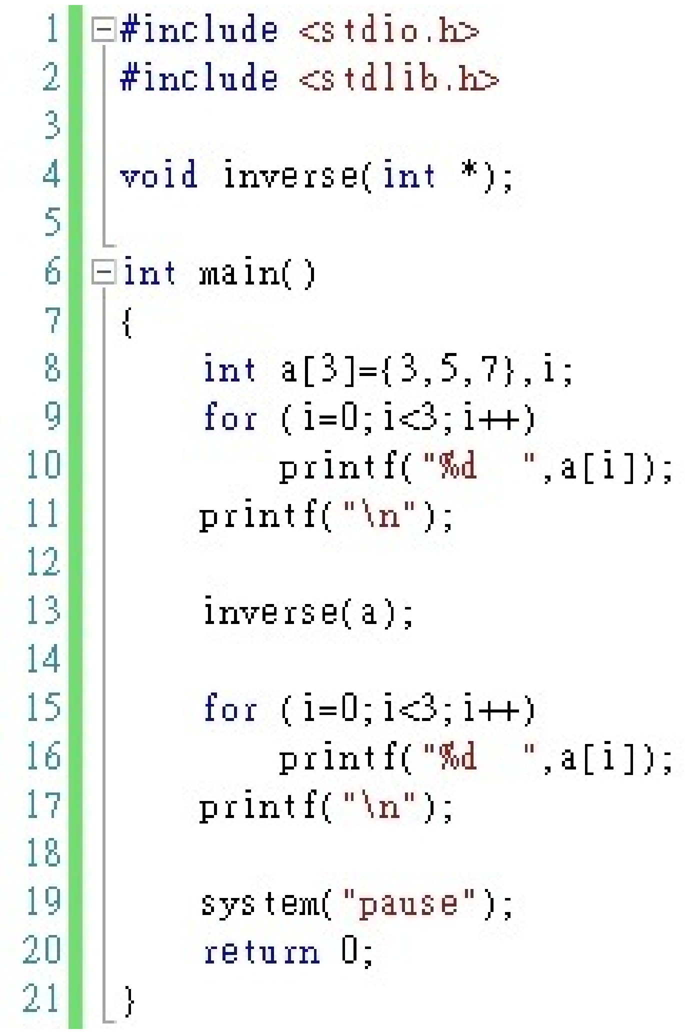
```
void modifyArray(int b[],int size)
{
    int j;

    for (j=0;j<size;j++)
    {
        b[j] *=2;
    }
}

void modifyElement(int e)
{
    printf("Value in modifyElement is %d\n", e *= 2);
}
```



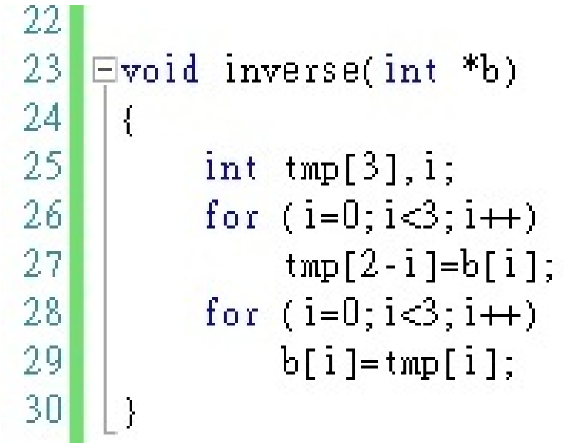
```
C:\c_code\ch3-57\Debug\ch3-57.exe
Effects of passing entire array by reference:

Thevalues of the original array are:
  0  1  2  3  4
The values of the modified array are:
  0  2  4  6  8


Effects of passing array elementby value:

The value of a[3] is 6
Value in modifyElement is 12
The value of a[3] is 6
請按任意鍵繼續 . . .
```

Arrays

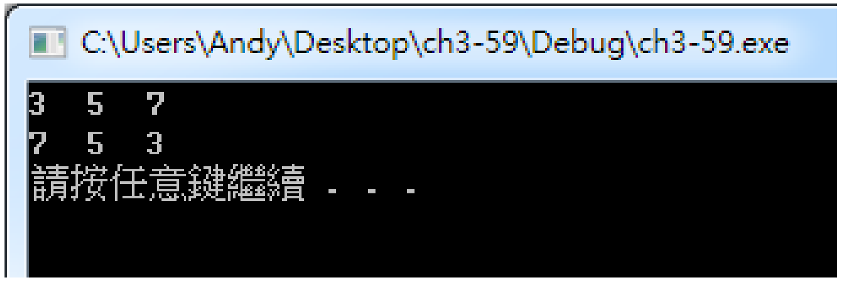
- **Call by Address**



```c
#include <stdio.h>
#include <stdlib.h>

void inverse(int *);

int main()
{
	int a[3]={3,5,7},i;
	for (i=0;i<3;i++)
		printf("%d  ",a[i]);
	printf("\n");

	inverse(a);

	for (i=0;i<3;i++)
		printf("%d  ",a[i]);
	printf("\n");

	system("pause");
	return 0;
}
```



```c

void inverse(int *b)
{
	int tmp[3],i;
	for (i=0;i<3;i++)
		tmp[2-i]=b[i];
	for (i=0;i<3;i++)
		b[i]=tmp[i];
}
```



```
C:\Users\Andy\Desktop\ch3-59\Debug\ch3-59.exe
3  5  7
7  5  3
請按任意鍵繼續 . . .
```

Arrays

- 陣列的排序(Sorting)資料是照特定的順序放置資料，例如遞增或遞減順序是電腦最重要的應用
  - 原本陣列: 26, 5, 81, 7, 63
  - 遞增排序: 5, 7, 26, 63, 81
- 陣列的排序(Sorting)種類
  - Bubble Sort
  - Selection Sort
  - Insertion Sort
  - Quick Sort
  - Heap Sort

Arrays

- 氣泡排序 (bubble sort或sinking sort)，因為較小的數值將會如氣泡浮出水面一樣，慢慢地上升至陣列的頂點，而較大的數值則會沉到陣列的尾端
- 原本陣列:

26	5	81	7	63

i=0

	a[0]	a[1]	a[2]	a[3]	a[4]
j=0	5	26	81	7	63
j=1	5	26	81	7	63
j=2	5	26	7	81	63
j=3	5	26	7	63	81

i=1

	a[0]	a[1]	a[2]	a[3]	a[4]
j=0	5	26	7	63	81
j=1	5	7	26	63	81
j=2	5	7	26	63	81
j=3	5	7	26	63	81

i=2

	a[0]	a[1]	a[2]	a[3]	a[4]
j=0	5	7	26	63	81
j=1	5	7	26	63	81
j=2	5	7	26	63	81
j=3	5	7	26	63	81

i=3

	a[0]	a[1]	a[2]	a[3]	a[4]
j=0	5	7	26	63	81
j=1	5	7	26	63	81
j=2	5	7	26	63	81
j=3	5	7	26	63	81

- 氣泡排序 (bubble sort)
  - 每次都由左至右，數字兩兩比對
    - 若前面的數字比後面大，則前後交換，即較小的數值如氣泡浮出水面(前後交換)

```
tmp = a[j];
a[j] = a[j+1];
a[j+1] = tmp;
```

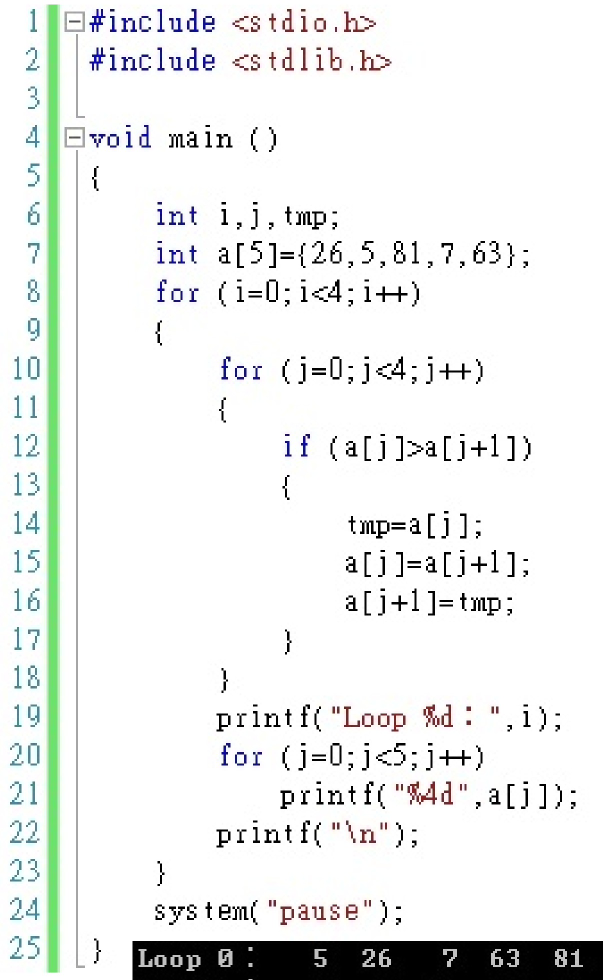
    - 否則不換
  - 對調動作不能只用以下的兩個指定動作來進行

```
a[ i ] = a[ i + 1 ];
a[ i + 1 ] = a[ i ];
```

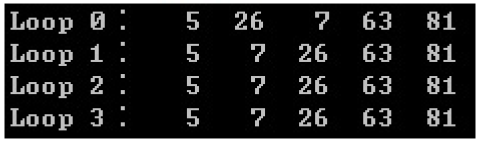
a[0]	*a[1]*	*a[2]*	*a[3]*	*a[4]*
26	5	81	7	63

Arrays

- 氣泡排序 (bubble sort)



```
#include <stdio.h>
#include <stdlib.h>

void main ()
{
	int i,j,tmp;
	int a[5]={26,5,81,7,63};
	for (i=0;i<4;i++)
	{
		for (j=0;j<4;j++)
		{
			if (a[j]>a[j+1])
			{
				tmp=a[j];
				a[j]=a[j+1];
				a[j+1]=tmp;
			}
		}
		printf("Loop %d：",i);
		for (j=0;j<5;j++)
			printf("%4d",a[j]);
		printf("\n");
	}
	system("pause");
}
```



```
Loop 0 :   5  26   7  63  81
Loop 1 :   5   7  26  63  81
Loop 2 :   5   7  26  63  81
Loop 3 :   5   7  26  63  81
```

a[0]	a[1]	a[2]	a[3]	a[4]
26	5	81	7	63

	a[0]	a[1]	a[2]	a[3]	a[4]
i=0 j=0	5	26	81	7	63
j=1	5	26	81	7	63
j=2	5	26	7	81	63
j=3	5	26	7	63	81

	a[0]	a[1]	a[2]	a[3]	a[4]
i=1 j=0	5	26	7	63	81
j=1	5	7	26	63	81
j=2	5	7	26	63	81
j=3	5	7	26	63	81

Arrays

- 氣泡排序 (bubble sort)的優點是它很容易撰寫。
- 但氣泡排序執行得相當慢，因為每次的交換只能朝元素的最終位置前進一步
- 尤其是在排序很大的陣列時
- 在習題中，我們將發展出一種較有效率的氣泡排序法
- 一些遠比氣泡排序法有效率的排序方法已經發展出來
  - Bubble Sort
  - Selection Sort
  - Insertion Sort
  - Quick Sort
  - Heap Sort

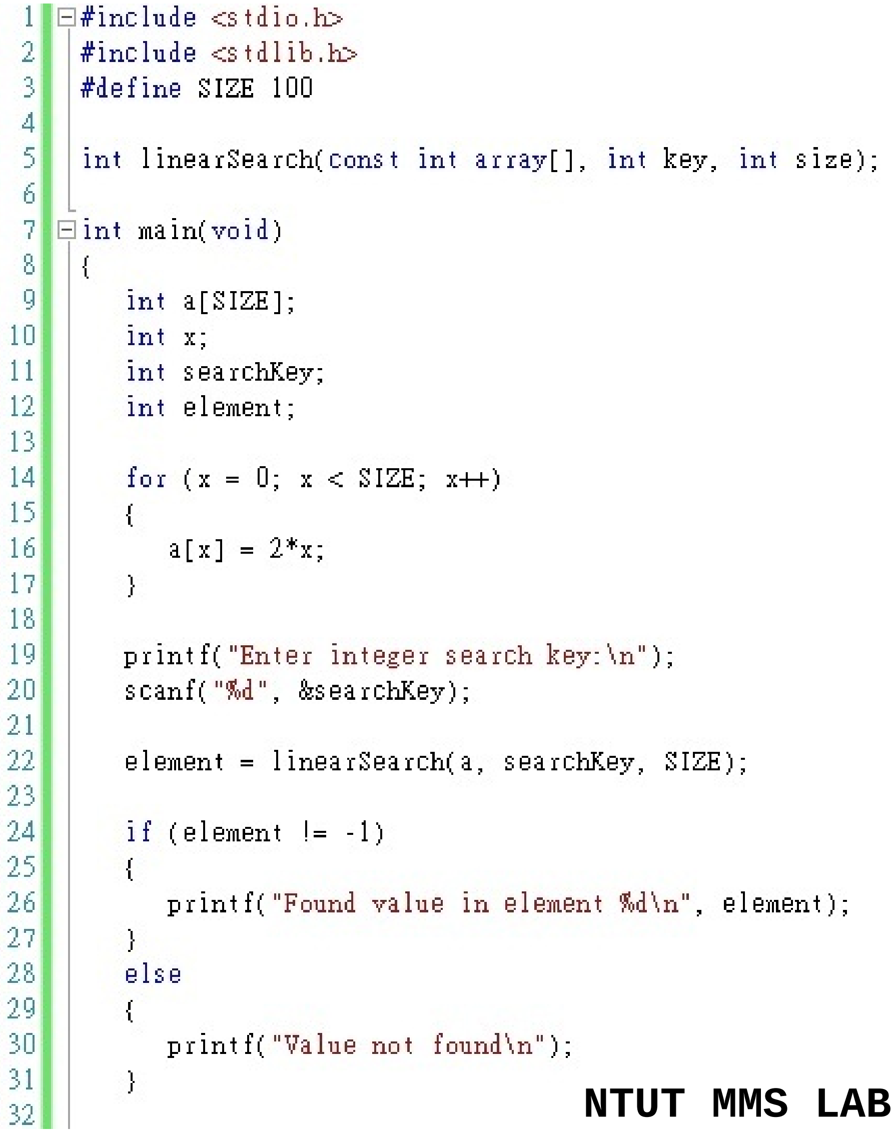
Arrays

- 搜尋 (searching)
  - 找出陣列中某個元素的過程稱為搜尋 (searching)
  - 搜尋陣列中是否有一個符合某個關鍵值 (key value) 的數值
- 兩種搜尋(searching)的技術介紹
  - 最簡單的線性搜尋 (linear search)
  - 較有效率 (也較複雜) 的二元搜尋 (binary search)

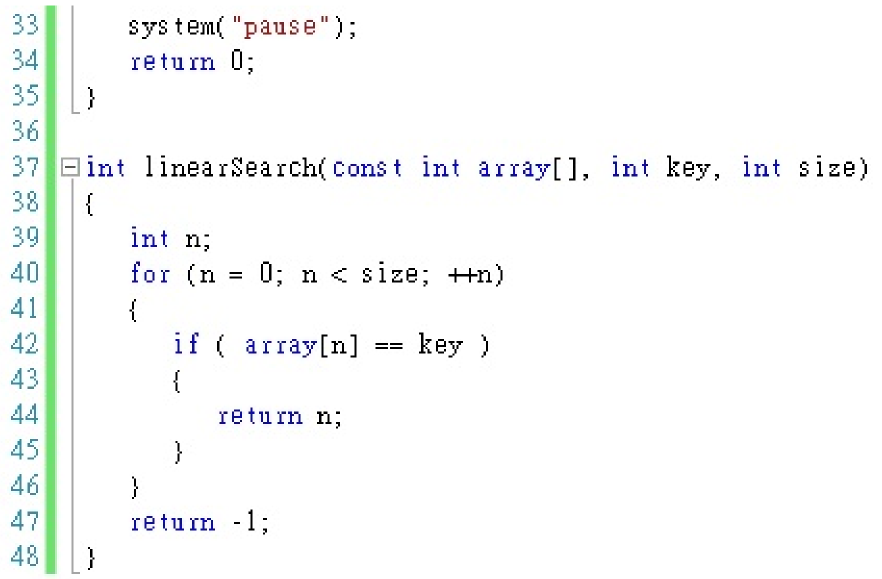
Arrays

- linear search

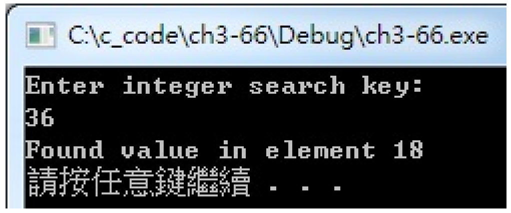
a[0]	a[1]	a[2]	a[3]	….	a[99]
0	2	4	6	…	198



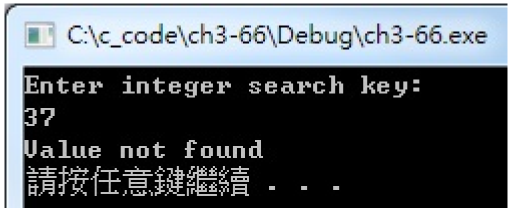
```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 100

int linearSearch(const int array[], int key, int size);

int main(void)
{
    int a[SIZE];
    int x;
    int searchKey;
    int element;

    for (x = 0; x < SIZE; x++)
    {
        a[x] = 2*x;
    }

    printf("Enter integer search key:\n");
    scanf("%d", &searchKey);

    element = linearSearch(a, searchKey, SIZE);

    if (element != -1)
    {
        printf("Found value in element %d\n", element);
    }
    else
    {
        printf("Value not found\n");
    }

```



```c
    system("pause");
    return 0;
}

int linearSearch(const int array[], int key, int size)
{
    int n;
    for (n = 0; n < size; ++n)
    {
        if ( array[n] == key )
        {
            return n;
        }
    }
    return -1;
}
```



```
C:\c_code\ch3-66\Debug\ch3-66.exe
Enter integer search key:
36
Found value in element 18
請按任意鍵繼續 . . .
```



```
C:\c_code\ch3-66\Debug\ch3-66.exe
Enter integer search key:
37
Value not found
請按任意鍵繼續 . . .
```

Arrays

- 對於小型的陣列或未排序過的陣列而言，線性搜尋(linear search) 可以表現的很好，但是將線性搜尋(linear search)
- 用在大型陣列上，就很沒有效率。
- 如果陣列已經排序過了，則我們可以用速度很快的二元搜尋法，二元搜尋演算法(binary search)在每次比較之後，就可以將已排序陣列中一半的元素刪去不考慮。

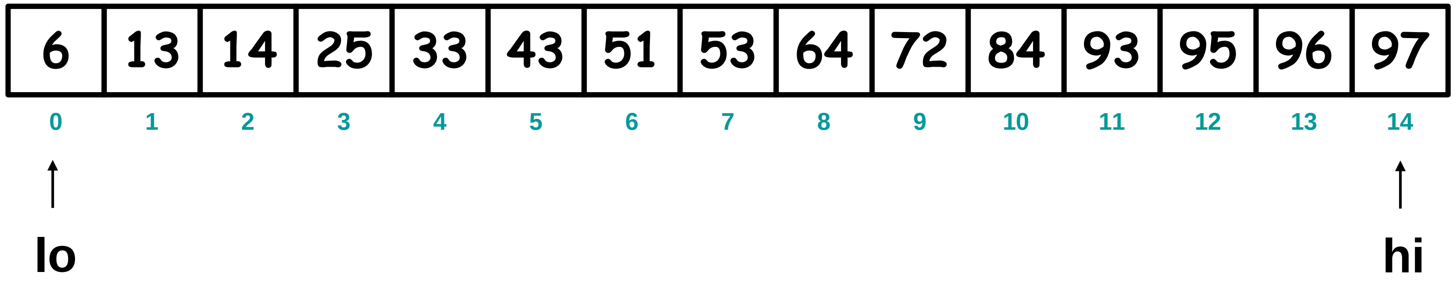
Arrays

- binary search先找出已經排序陣列的中間元素，將之與搜尋關鍵值作比較
  1. 如果相等的話，表示已找到要找的元素，就將此元素的陣列下標傳回
  2. 如果不相等，此時問題便簡化成只需搜尋陣列的某一半
     1. 如果搜尋的關鍵值小於陣列的中間元素，就搜尋陣列的前半部
     2. 否則就會搜尋陣列的後半部

a[0]	a[1]	a[2]	a[3]	a[4]
5	7	26	63	81

Binary Search

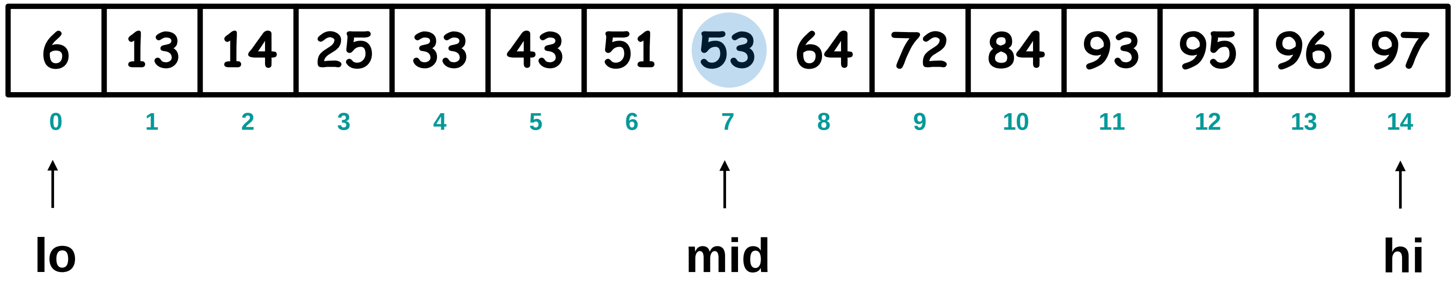
- Binary search. Given value and sorted array a[], find index i such that a[i] = value, or report that no such index exists.
- Invariant. Algorithm maintains a[lo] ≤ value ≤ a[hi].
- Ex. Binary search for 33.



MMS Lab

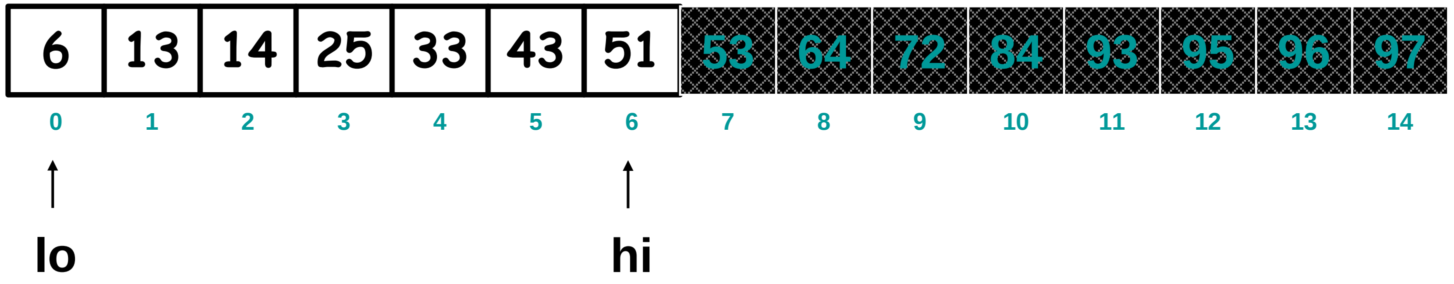
Binary Search

- Binary search. Given value and sorted array a[], find index i such that a[i] = value, or report that no such index exists.
- Invariant. Algorithm maintains a[lo] ≤ value ≤ a[hi].
- Ex. Binary search for 33.



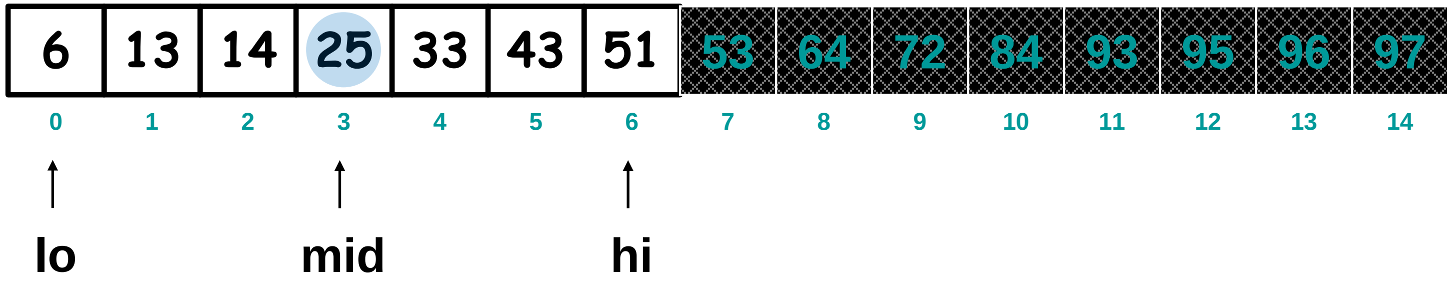
Binary Search

- Binary search. Given value and sorted array a[], find index i such that a[i] = value, or report that no such index exists.
- Invariant. Algorithm maintains a[lo] ≤ value ≤ a[hi].
- Ex. Binary search for 33.



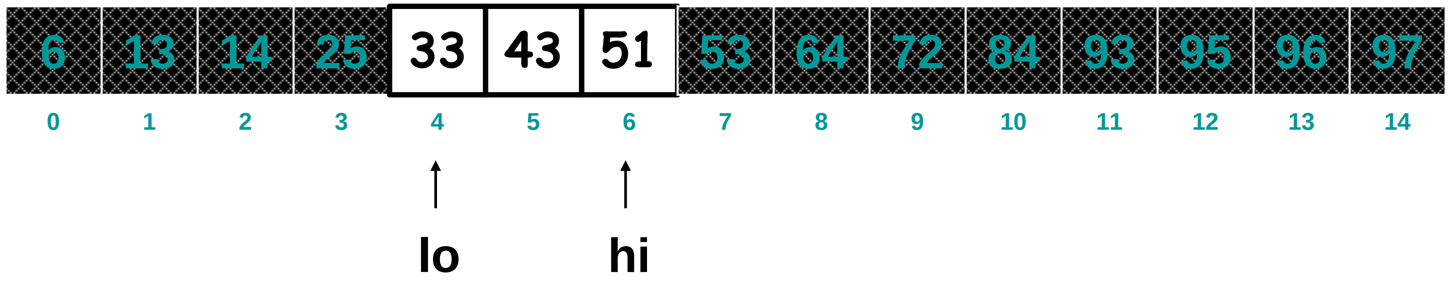
Binary Search

- Binary search. Given value and sorted array a[], find index i such that a[i] = value, or report that no such index exists.
- Invariant. Algorithm maintains a[lo] ≤ value ≤ a[hi].
- Ex. Binary search for 33.



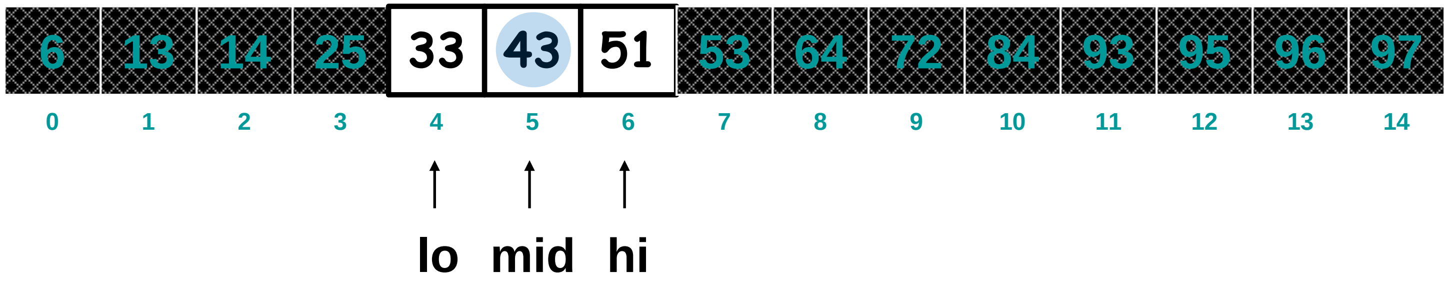
Binary Search

- Binary search. Given value and sorted array a[], find index i such that a[i] = value, or report that no such index exists.
- Invariant. Algorithm maintains a[lo] ≤ value ≤ a[hi].
- Ex. Binary search for 33.



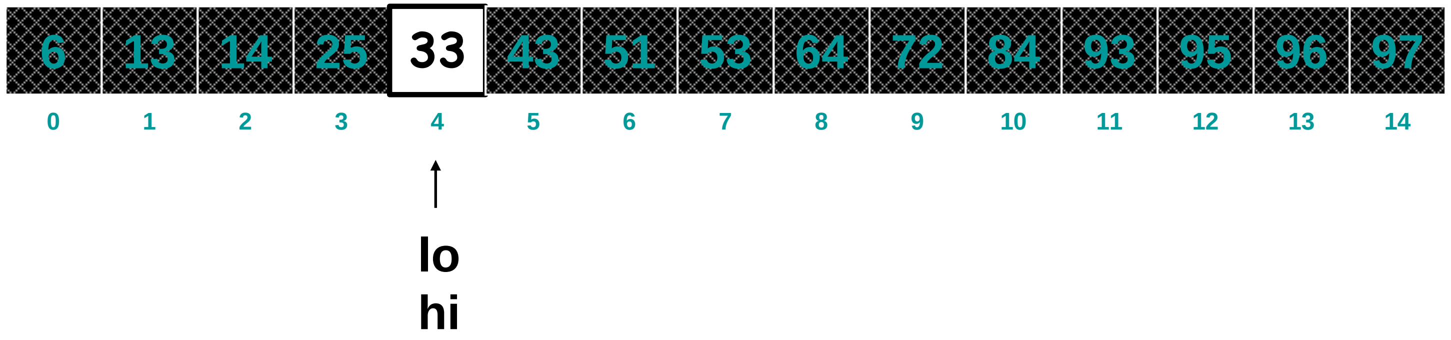
Binary Search

- Binary search. Given value and sorted array a[], find index i such that a[i] = value, or report that no such index exists.
- Invariant. Algorithm maintains a[lo] ≤ value ≤ a[hi].
- Ex. Binary search for 33.



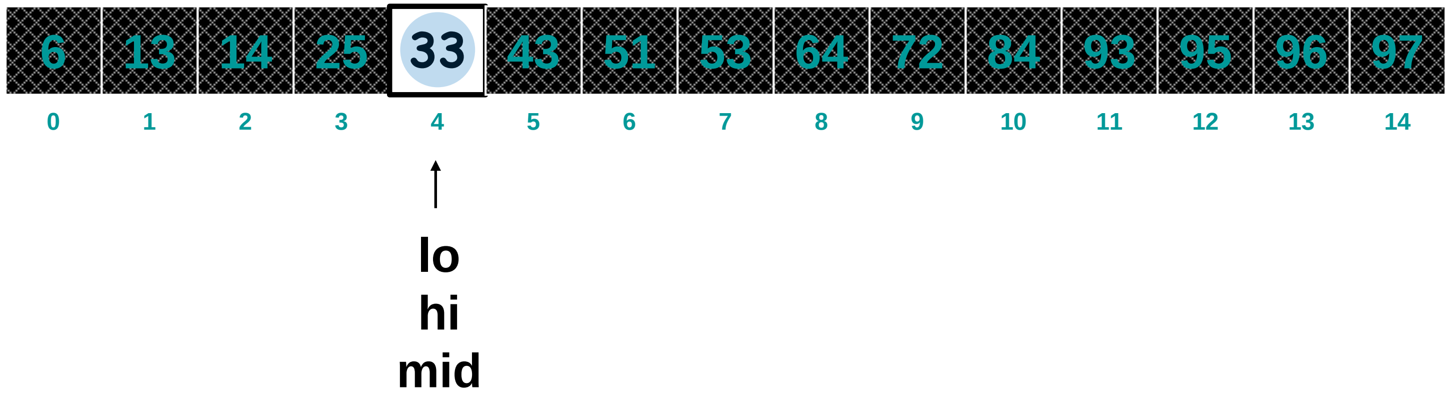
Binary Search

- Binary search. Given value and sorted array a[], find index i such that a[i] = value, or report that no such index exists.
- Invariant. Algorithm maintains a[lo] ≤ value ≤ a[hi].
- Ex. Binary search for 33.



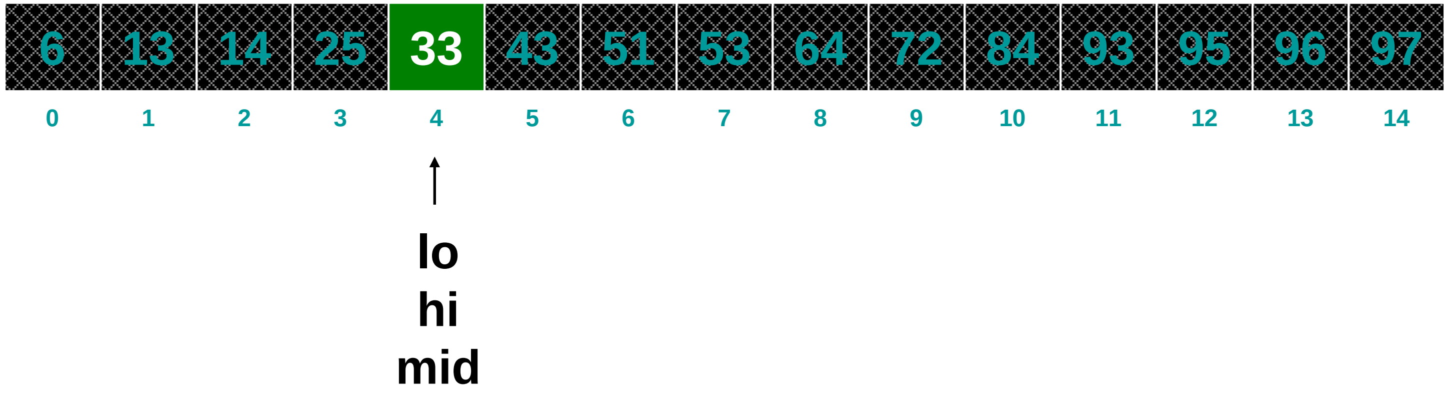
Binary Search

- Binary search. Given value and sorted array a[], find index i such that a[i] = value, or report that no such index exists.
- Invariant. Algorithm maintains a[lo] ≤ value ≤ a[hi].
- Ex. Binary search for 33.



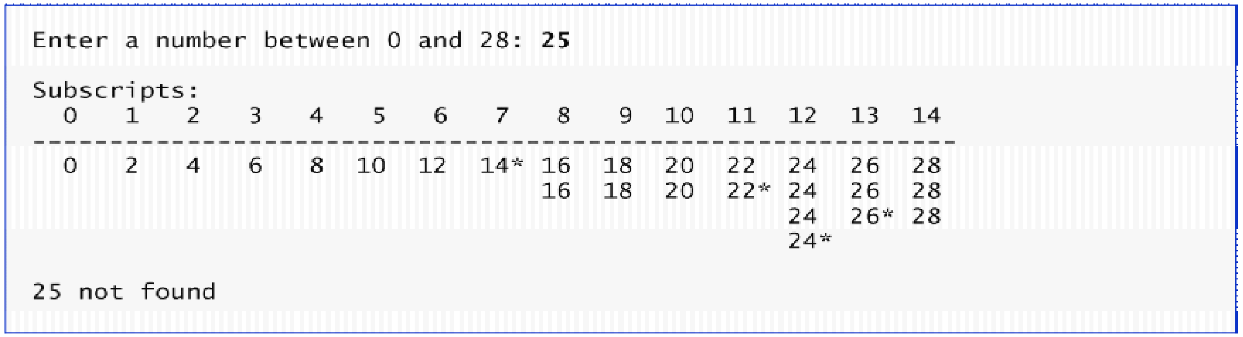
Binary Search

- Binary search. Given value and sorted array a[], find index i such that a[i] = value, or report that no such index exists.
- Invariant. Algorithm maintains a[lo] ≤ value ≤ a[hi].
- Ex. Binary search for 33.

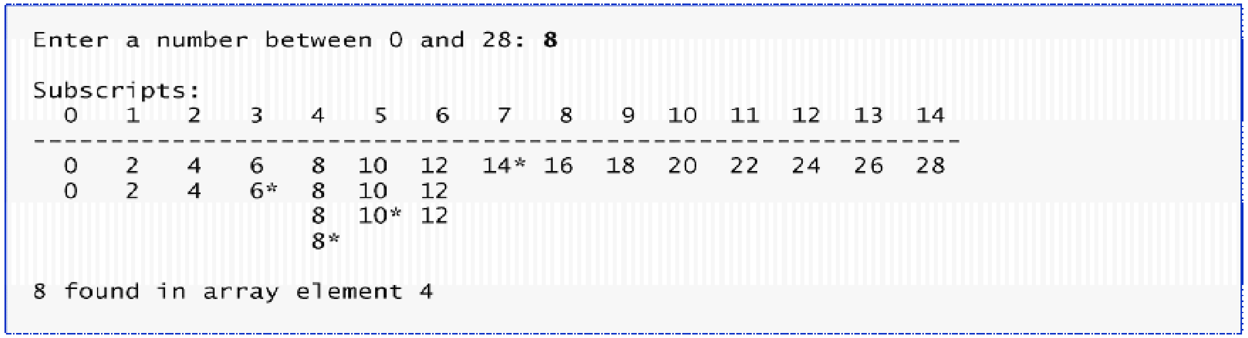


Arrays

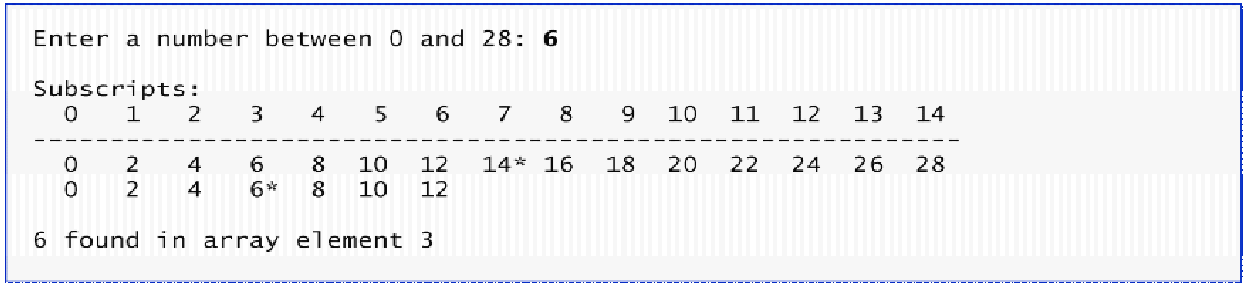
- binary search



```
Enter a number between 0 and 28: 25

Subscripts:
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14
------------------------------------------------------------
  0   2   4   6   8  10  12  14* 16  18  20  22  24  26  28
                                 16  18  20  22* 24  26  28
                                                 24  26* 28
                                                 24*

25 not found
```



```
Enter a number between 0 and 28: 8

Subscripts:
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14
------------------------------------------------------------
  0   2   4   6   8  10  12  14* 16  18  20  22  24  26  28
  0   2   4   6*  8  10  12
                  8  10* 12
                  8*

8 found in array element 4
```



```
Enter a number between 0 and 28: 6

Subscripts:
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14
------------------------------------------------------------
  0   2   4   6   8  10  12  14* 16  18  20  22  24  26  28
  0   2   4   6*  8  10  12

6 found in array element 3
```