



برمجة ++C من تحليل المشكلات إلى تصميم البرنامجو الطبعة الخامسة

C++ Programming: From Problem Analysis to Program Design, Fifth Edition

Chapter 10: Functions & Applications of Arrays

الفصل العاشر: الوظائف و
تطبيقات المصفوفات

A QUICK REVIEW ON ARRAYS

مراجعة سريعة للصفائف

Accessing Array Components (cont'd.)

```
list[3] = 10;  
list[6] = 35;  
list[5] = list[3] + list[6];
```

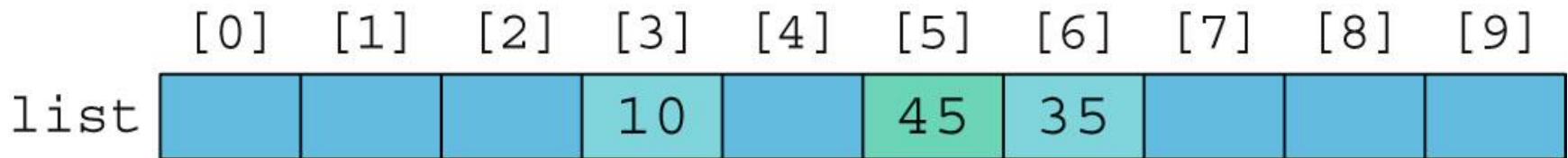


FIGURE 8-5 Array `list` after execution of the statements `list[3]= 10;`, `list[6]= 35;`, and `list[5] = list[3] + list[6];`

Processing One-Dimensional Arrays (cont'd.)

معالجة المصفوفات أحادية البعد (تابع)

- Given the declaration:

```
int list[100];    //array of size 100
int i;
```

- Use a `for` loop to access array elements:

```
for (i = 0; i < 100; i++)    //Line 1
    cin >> list[i];         //Line 2
```

- بالنظر إلى الإعلان:

```
قائمة [100] int ؛ // مجموعة بحجم 100
إنت أنا
```

- إستخدم بالنسبة حلقة للوصول إلى عناصر المصفوفة:

```
ل (i = 0 ؛ i < 100 ؛ // i ++ السطر 1
سينما >> قائمة [i] ؛ // خط 2
```

Partial Initialization of Arrays During Declaration

التهيئة الجزئية للصفائف أثناء الإعلان

- The statement:

```
int list[10] = {0};
```

- Declares an array of 10 components and initializes all of them to zero

- The statement:

```
int list[10] = {8, 5, 12};
```

- Declares an array of 10 components and initializes `list[0]` to 8, `list[1]` to 5, `list[2]` to 12
- All other components are initialized to 0

- البيان:

```
قائمة int [10] = {0};
```

- يعلن عن مصفوفة من 10 مكونات ويهيئها جميعًا إلى الصفر

- البيان:

```
قائمة int [10] = {8, 5, 12};
```

- تعلن عن مصفوفة من 10 مكونات وتهيئتها قائمة [0] إلى 8 قائمة [1] إلى 5 قائمة [2] حتى 12
- تتم تهيئة كافة المكونات الأخرى إلى 0

Some Restrictions on Array Processing

• العملية الكلية: أي عملية تعالج المصفوفة بأكملها كوحدة واحدة

– غير مسموح به في المصفوفات في C++

- Aggregate operation: any operation that manipulates the entire array as a single unit
 - Not allowed on arrays in C++

- Example:

• مثال:

```
int myList[5] = {0, 4, 8, 12, 16}; //Line 1
int yourList[5]; //Line 2

yourList = myList; //illegal
```

- Solution:

• المحلول:

```
for (int index = 0; index < 5; index ++)  
    yourList[index] = myList[index];
```

Two- and Multidimensional Arrays

• صفيف ثنائي الأبعاد: مجموعة من عدد ثابت من المكونات (من نفس النوع) مرتبة في بعدين
– تسمى أحيانًا المصفوفات أو الجداول

- Two-dimensional array: collection of a fixed number of components (of the same type) arranged in two dimensions

– Sometimes called matrices or tables

- Declaration syntax:

• بناء جملة الإعلان:

```
dataType arrayName[intExp1][intExp2];
```

- intExp1 and intExp2 are expressions with positive integer values specifying the number of rows and columns in the array

– intExp2 و intExp1 هي تعبيرات ذات قيم صحيحة موجبة تحدد عدد الصفوف والأعمدة في المصفوفة

Accessing Array Components

• الوصول إلى المكونات في مصفوفة ثنائية الأبعاد:

- Accessing components in a two-dimensional array:

```
arrayName[indexExp1][indexExp2]
```

– Where `indexExp1` and `indexExp2` are expressions with positive integer values, and specify the row and column position

- Example:

– أين الفهرس وفهرس هي تعبيرات ذات قيم صحيحة موجبة ، وتحدد موضع الصف والعمود

مثال: المبيعات $[5] [3] = 25.75$ ؛

```
sales[5][3] = 25.75;
```


Accessing Array Components (cont'd.)

| sales | [0] | [1] | [2] | [3] | [4] |
|-------|-----|-----|-----|-------|-----|
| [0] | | | | | |
| [1] | | | | | |
| [2] | | | | | |
| [3] | | | | | |
| [4] | | | | | |
| [5] | | | | 25.75 | |
| [6] | | | | | |
| [7] | | | | | |
| [8] | | | | | |
| [9] | | | | | |

sales [5] [3]

FIGURE 8-14 sales[5][3]

Print ^{مطبعة}

• استخدم حلقة متداخلة لإخراج مكونات مصفوفة ثنائية الأبعاد:

- Use a nested loop to output the components of a two dimensional array:

```
for (row = 0; row < NUMBER_OF_ROWS; row++)  
{  
    for (col = 0; col < NUMBER_OF_COLUMNS; col++)  
        cout << setw(5) << matrix[row][col] << " ";  
  
    cout << endl;  
}
```

ARRAYS & FUNCTIONS

المصفوفات والوظائف

المصفوفات كمعاملات للوظائف

Consider the following function:

```
void funcArrayAsParam(int listOne[], double listTwo[])
{
    .
    .
    .
}
```

is a

- The size of the array is usually omitted
 - If provided, it is ignored by the compiler
- C++ does not allow functions to return a value of the type array

- يتم تمرير المصفوفات بالإشارة فقط
- الرمز & يكون/ليس تستخدم عند التصريح عن مصفوفة كمعامل رسمي
- عادة ما يتم حذف حجم المصفوفة
- – إذا تم توفيره ، يتم تجاهله من قبل المترجم
- لا تسمح لغة C++ للوظائف بإرجاع قيمة مصفوفة النوع

```
//Function to print the elements of an int array.  
//The array to be printed and the number of elements  
//are passed as parameters. The parameter listSize  
//specifies the number of elements to be printed.  
void printArray(const int list[], int listSize)  
{  
    int index;  
  
    for (index = 0; index < listSize; index++)  
        cout << list[index] << " ";  
}
```

Parameters



Base Address of an Array and Array in Computer Memory

- The base address of an array is the address, or memory location of the first array component
- If `list` is a one-dimensional array, its base address is the address of `list[0]`
- When we pass an array as a parameter, the base address of the actual array is passed to the formal parameter

- العنوان الأساسي للمصفوفة هو العنوان أو موقع الذاكرة لمكون المصفوفة الأول
- لو قائمة هي مصفوفة ذات بعد واحد ، عنوانها الأساسي هو عنوان قائمة `[0]`
- عندما نقوم بتمرير مصفوفة كمعامل ، يتم تمرير العنوان الأساسي للمصفوفة الفعلية إلى المعلمة الرسمية

Base Address of an Array and Array in Computer Memory (cont'd.)

العنوان الأساسي للصفيف والصفيف في ذاكرة الكمبيوتر (تابع)

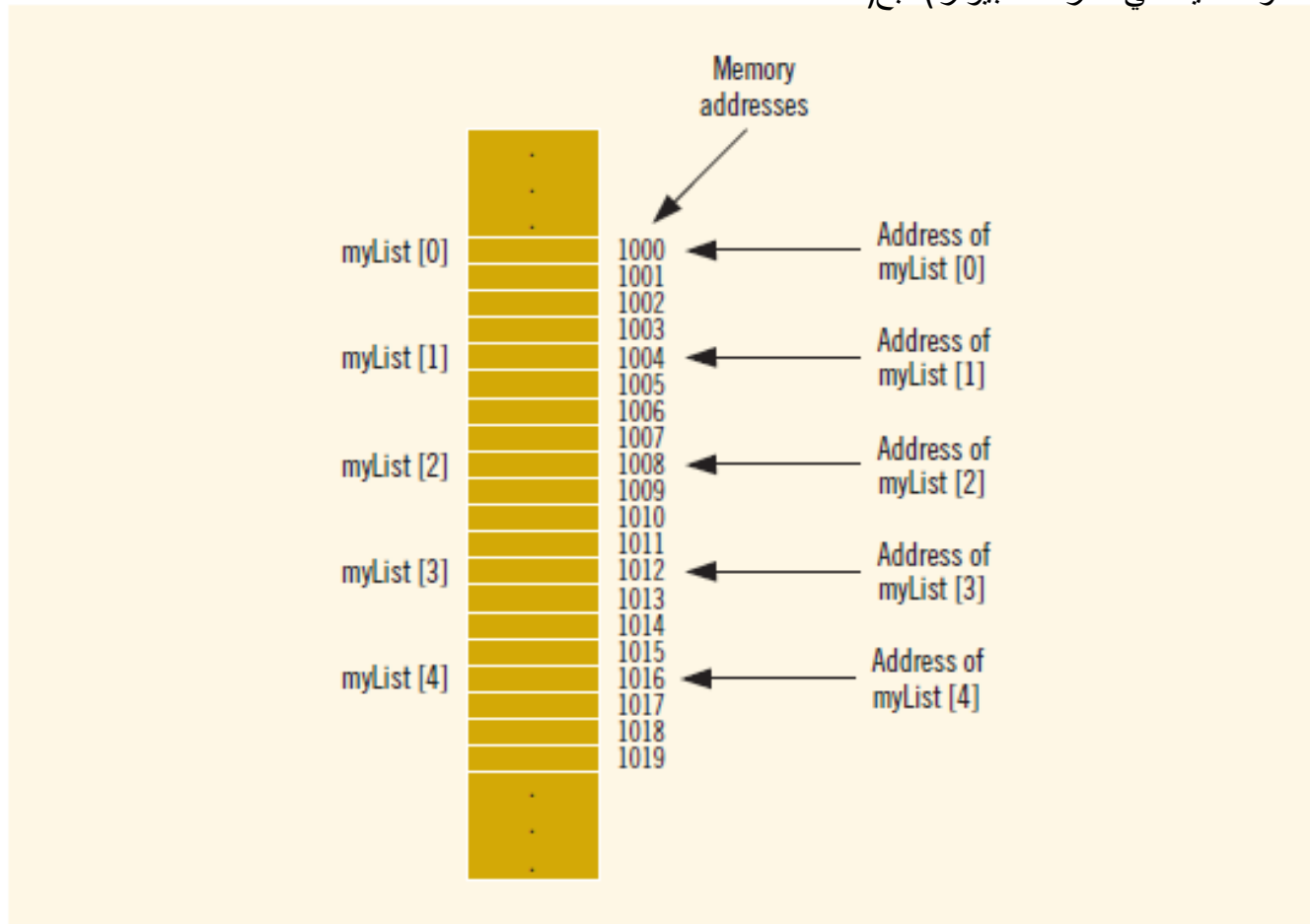


FIGURE 9-7 Array `myList` and the addresses of its components

Example on Functions Using Arrays

مثال على دوال باستخدام المصفوفات

ليس في النص!

Not in text!

اكتب برنامجًا يقرأ مصفوفة من 5 أعداد صحيحة ، ثم اطبع المصفوفة مرة أخرى ، واعثر على متوسطها ، ثم اعرضها مرة أخرى على المستخدم باستخدام الوظائف التالية:

Write a program that reads an array of 5 integers, print back the array, find their average, and display it back to the user using the following functions:

`readArray` ، `printArray` ، معدل

`readArray, printArray, average`


```
#include <iostream>
using namespace std;
```

```
void readArray(int theArray[], int sizeOfarray);
void printArray(int theArray[], int sizeOfarray);
double average(int numbers[], int size);
```

```
const int N=5;
```

```
int main()
{
```

```
    int array[N];
```

```
    readArray(array,N);
```

```
    printArray(array, N);
```

```
    cout << "\n\n The average is: " << average(array, N) << endl;
```

```
    return 0;
}
```

```
void readArray(int theArray[], int sizeOfarray)
{
    for (int i = 0; i < sizeOfarray; i++)
    {
        cout<<"\n Enter element no. "<<i<<" : ";
        cin >> theArray[i];
    }
}
```

```
void printArray(int theArray[], int sizeOfarray)
{

    cout<<"\n The Array you entered is: "<<endl<<endl;
    for (int i = 0; i < sizeOfarray; i++)
    {
        cout <<" "<<theArray[i]<<" ";
    }
}
```

```
double average(int numbers[], int size)
{
    double sum = 0.0;
    double arrayAverage;

    for (int i = 0; i < size; i++)
        sum = sum + numbers[i];

    arrayAverage = sum / size;

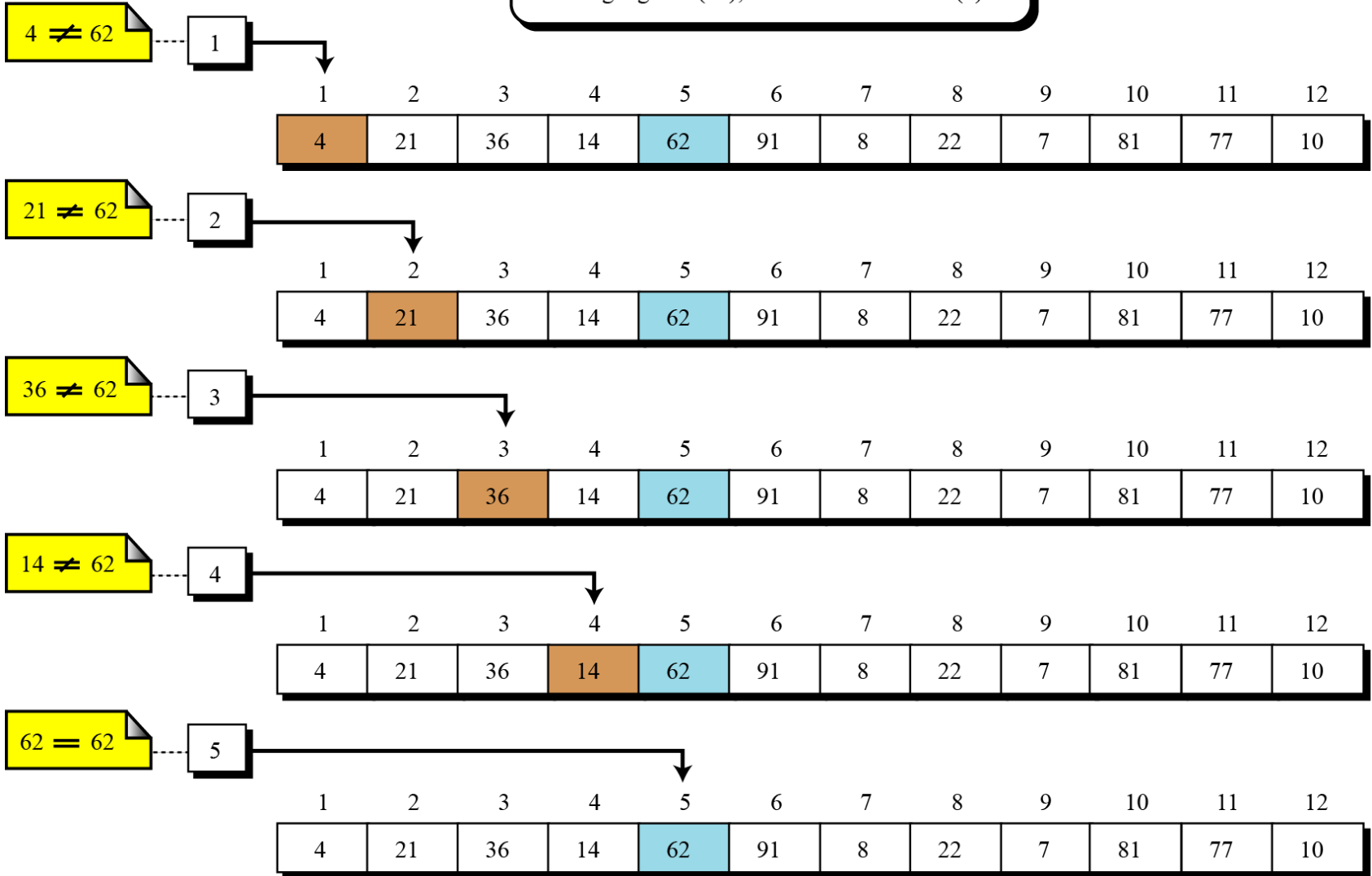
    return arrayAverage;
}
```

Sequential Search

- Sequential search or linear search
 - Searching a list for a given item
 - Starting from the first array element
 - Compare `searchItem` with the elements in the array
 - Continue the search until either you find the item or no more data is left in the `list` to compare with `searchItem`

- البحث المتسلسل أو البحث الخطي
 - البحث في قائمة عن عنصر معين
 - بدءا من أول عنصر مصفوفة
 - يقارن البحث مع العناصر الموجودة في المصفوفة
 - تابع البحث حتى تعثر على العنصر أو لم يتبق المزيد من البيانات في قائمة للمقارنة مع البحث

Target given (62); Location wanted (5)



Sequential Search Function

Ex 9-8 P509 in Text

وظيفة البحث المتسلسل
مثال 9-8 ص 509 في النص

```
int seqSearch(const int list[], int listLength, int searchItem)
{
    int loc;
    bool found = false;

    loc = 0;

    while (loc < listLength && !found)
        if (list[loc] == searchItem)
            found = true;
        else
            loc++;

    if (found)
        return loc;
    else
        return -1;
}
```

Sequential Search (cont'd.)

The Full Program

البحث المتسلسل (تابع)
البرنامج الكامل

```
// This program illustrates how to use a sequential search in a
// program.

#include <iostream>
using namespace std;

const int ARRAY_SIZE = 5;

int seqSearch(const int list[], int listLength, int searchItem);

int main()
{
    int intList[ARRAY_SIZE];
    int number;

    cout << " Enter " << ARRAY_SIZE << " integers." << endl;

    for (int index = 0; index < ARRAY_SIZE; index++)
        cin >> intList[index];
    cout << endl;

    cout << " Enter the number to be "<< "searched: ";
    cin >> number;
    cout << endl;
```

Sequential Search (cont'd.)

The Full Program

البحث المتسلسل (تابع)
البرنامج الكامل

```
int pos = seqSearch(intList, ARRAY_SIZE, number);

if (pos != -1)
    cout << " " << number << " is found at position " << pos << endl;
else
    cout << " " << number << " is not in the list." << endl;

return 0;
}

int seqSearch(const int list[], int listLength, int searchItem)
{
    int loc;
    bool found = false;
    loc = 0;

    while (loc < listLength && !found)
        if (list[loc] == searchItem)
            found = true;
        else
            loc++;
    if (found)
        return loc;
    else
        return -1;
}
```


Sequential Search (cont'd.)

البحث المتسلسل (تابع)

- List with 1000 elements
 - Search item is the second item
 - Sequential search makes two key comparisons
 - Search item is the 900th item
 - Sequential search makes 900 key comparisons
 - Search item is not in the list
 - Sequential search makes 1000 key comparisons

- قائمة تحتوي على 1000 عنصر
 - عنصر البحث هو العنصر الثاني
 - البحث المتسلسل يجعل مقارنتين رئيسيتين
 - عنصر البحث هو العنصر رقم 900
 - البحث المتسلسل يجعل 900 مقارنة رئيسية
 - عنصر البحث ليس في القائمة
 - البحث المتسلسل يجعل 1000 مقارنة رئيسية

Sequential Search (cont'd.)

البحث المتسلسل (تابع)

- Sequential search
 - Not very efficient for large lists
 - On average, number of key comparisons equal to half the size of the list
 - Does not assume that the list is sorted

- البحث المتسلسل

- ليست فعالة جدا للقوائم الكبيرة
- في المتوسط ، عدد المقارنات الرئيسية يساوي نصف حجم القائمة
- لا تفترض أن القائمة مرتبة

Binary Search

بحث ثنائي

- A sequential search is not very efficient for large lists. It typically searches about half of the list. However, if the list is sorted, you can use another search algorithm called binary search.
- A binary search is much faster than a sequential search. In order to apply a binary search, the list must be sorted.
- One of the sort techniques used is *Bubble sort*
 - البحث المتسلسل ليس فعالاً جداً للقوائم الكبيرة. يبحث عادةً عن نصف القائمة. ومع ذلك ، إذا تم فرز القائمة ، يمكنك استخدام خوارزمية بحث أخرى تسمى البحث الثنائي.
 - البحث الثنائي أسرع بكثير من البحث المتسلسل. من أجل تطبيق بحث ثنائي ، يجب فرز القائمة.
 - واحدة من تقنيات الفرز المستخدمة هي *فقاعة الفرز*

Binary Search

بحث ثنائي

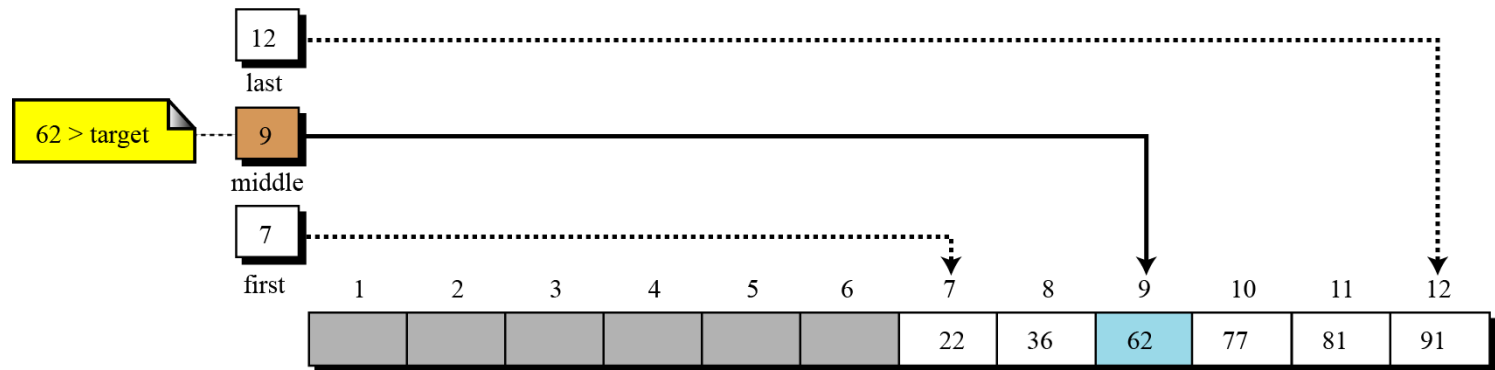
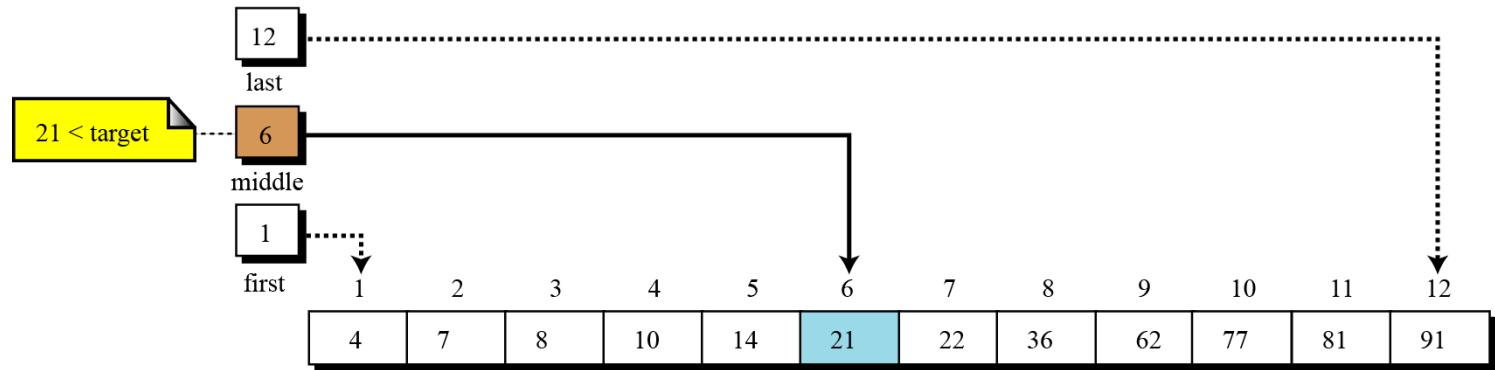
- A binary search starts by testing the data in the element at the middle of the list. This determines whether the target is in the first half or the second half of the list.
- If it is in the first half, there is no need to further check the second half.
- If it is in the second half, there is no need to further check the first half. In other words, we eliminate half the list from further consideration.

• يبدأ البحث الثنائي باختبار البيانات الموجودة في العنصر في منتصف القائمة. يحدد هذا ما إذا كان الهدف في النصف الأول أو النصف الثاني من القائمة.

• إذا كان في النصف الأول ، فلا داعي لمزيد من التحقق من النصف الثاني .

• إذا كان في النصف الثاني ، فلا داعي لمزيد من التحقق من النصف الأول .بعبارة أخرى ، نحذف نصف القائمة من مزيد من الدراسة.

Target given (22); Location wanted (7)



Binary Search

Paper & Pencil

بحث ثنائي
ورق وقلم رصاص

Task1:

Determine if 75 is in the following list:

| | | | | | | | | | | | |
|----|----|---|---|----|----|----|----|----|----|----|----|
| 39 | 19 | 8 | 4 | 25 | 34 | 45 | 66 | 48 | 95 | 89 | 75 |
|----|----|---|---|----|----|----|----|----|----|----|----|

مهمة: 1

حدد ما إذا كان 75 في القائمة التالية:

Binary Search Function

```
int binarySearch(const int list[], int listLength, int searchItem)
{
    int first = 0;
    int last = listLength - 1;
    int mid;

    bool found = false;

    while (first <= last && !found)
    {
        mid = (first + last) / 2;

        if (list[mid] == searchItem)
            found = true;
        else if (list[mid] > searchItem)
            last = mid - 1;
        else
            first = mid + 1;
    }

    if (found)
        return mid;
    else
        return -1;
} //end binarySearch
```

Group
Work

**Task2: Show the
memory contents by
tracing the values of the
variables:
First, Mid & Last**

Bubble Sort

فقاعة الفرز

- `list[0] ... list[n - 1]`
 - List of n elements, indexed 0 to $n - 1$

- قائمة `list[0] ... list[n - 1]`
 - قائمة العناصر n ، مفهرسة من 0 إلى $n - 1$

| list | |
|----------------------|----|
| <code>list[0]</code> | 10 |
| <code>list[1]</code> | 7 |
| <code>list[2]</code> | 19 |
| <code>list[3]</code> | 5 |
| <code>list[4]</code> | 16 |

FIGURE 10-1 List of five elements

Bubble Sort (cont'd.)

- Series of $n - 1$ iterations
 - Successive elements `list[index]` and `list[index + 1]` of `list` are compared
 - If `list[index] > list[index + 1]`
 - Elements `list[index]` and `list[index + 1]` are swapped
- Smaller elements move toward the top
- Larger elements move toward the bottom

- سلسلة من $n - 1$ - التكرارات
 - العناصر المتتالية قائمة [فهرس [وقائمة فهرس $+ 1$ من قائمة تم مقارنتها
 - لو قائمة [فهرس $>$ [قائمة فهرس $+ 1$
 - عناصر قائمة [فهرس [وقائمة فهرس $+ 1$ يتم تبديلها
- العناصر الأصغر تتحرك نحو الأعلى
- تتحرك العناصر الأكبر نحو الأسفل

Bubble Sort (cont'd.)

فرز الفقاعات (تابع)

مثال: قم بفرز القائمة التالية باستخدام Bubble Sort

Example: Sort the following list using Bubble Sort

Iteration 1: Sort `list[0]...list[4]`. Figure 10-2 shows how the elements of `list` get rearranged in the first iteration.

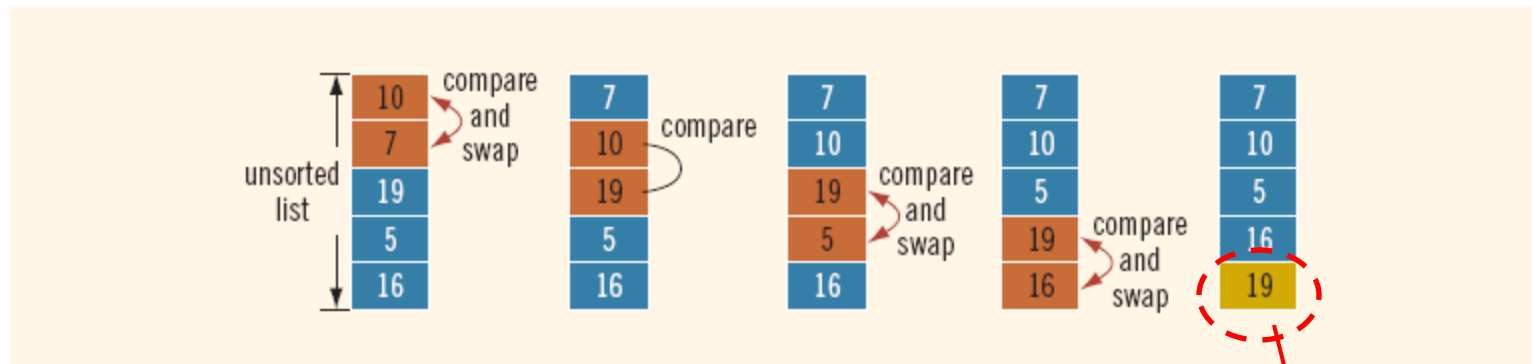


FIGURE 10-2 Elements of `list` during the first iteration

unnecessary
comparison

غير ضروري
مقارنة

Bubble Sort (cont'd.)

فرز الفقاعات (تابع)

Iteration 2: Sort `list[0...3]`. Figure 10-3 shows how the elements of `list` get rearranged in the second iteration.

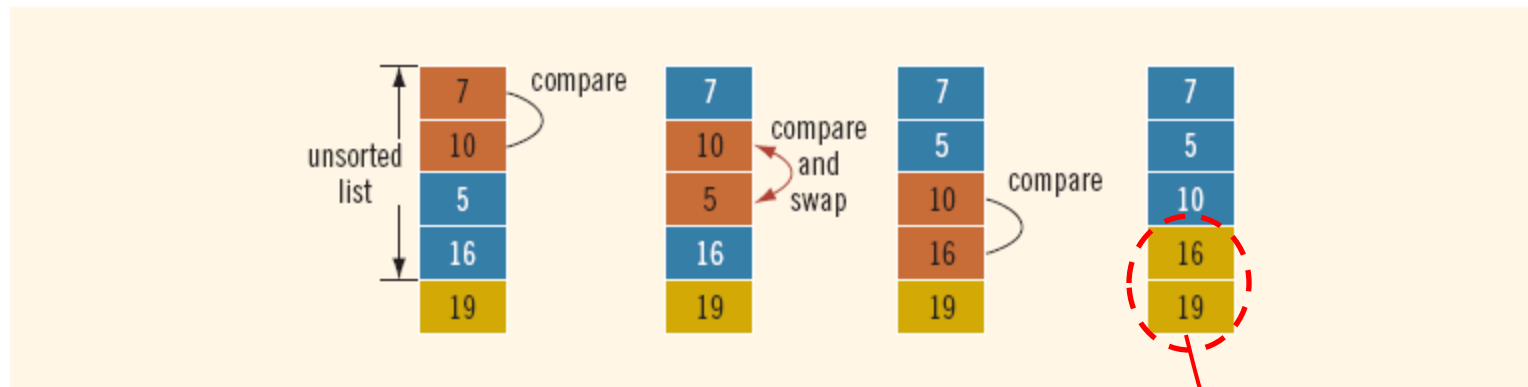


FIGURE 10-3 Elements of `list` during the second iteration

unnecessary
comparisons

غير ضروري
مقارنات

Bubble Sort (cont'd.)

فرز الفقاعات (تابع)

Iteration 3: Sort `list[0...2]`. Figure 10-4 shows how the elements of `list` get rearranged in the third iteration.

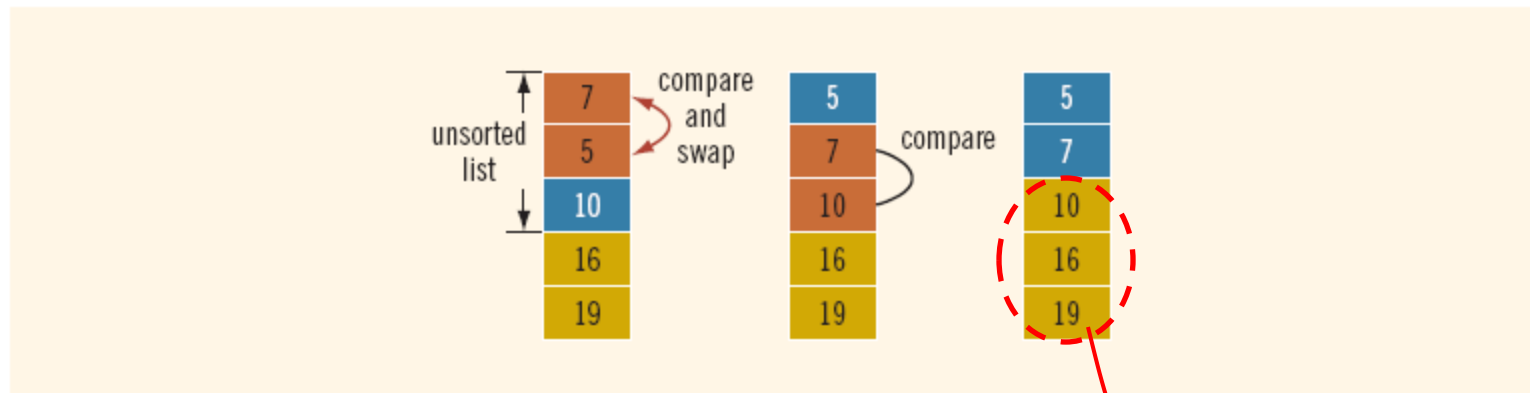


FIGURE 10-4 Elements of `list` during the third iteration

unnecessary
comparisons

Bubble Sort (cont'd.)

فرز الفقاعات (تابع)

Iteration 4: Sort `list[0...1]`. Figure 10-5 shows how the elements of `list` get rearranged in the fourth iteration.

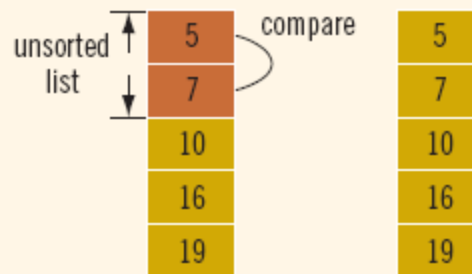


FIGURE 10-5 Elements of `list` during the fourth iteration

End of Solution

Bubble Sort Function

وظيفة فرز الفقاعات

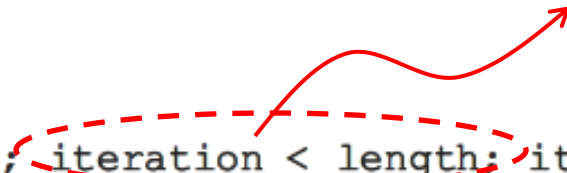
```
void bubbleSort(int list[], int length)
{
    int temp;
    int iteration;
    int index;
    for (iteration = 1; iteration < length; iteration++)
    {
        for (index = 0; index < length - iteration; index++)
            if (list[index] > list[index + 1])
            {
                temp = list[index];
                list[index] = list[index + 1];
                list[index + 1] = temp;
            }
    }
}
```

Bubble Sort (cont'd.)

فرز الفقاعات (تابع)

```
void bubbleSort(int list[], int length)
{
    int temp;
    int iteration;
    int index;
    for (iteration = 1; iteration < length; iteration++)
    {
        for (index = 0; index < length - iteration; index++)
            if (list[index] > list[index + 1])
            {
                temp = list[index];
                list[index] = list[index + 1];
                list[index + 1] = temp;
            }
    }
}
```

Iterations=length-1



Bubble Sort (cont'd.)

فرز الفقاعات (تابع)

تحدي: اعرض محتويات الذاكرة عن طريق تتبع قيم المتغيرات

```
void bubbleSort(int list[], int length)
{
    int temp;
    int iteration;
    int index;
    for (iteration = 1; iteration < length; iteration++)
    {
        for (index = 0; index < length - iteration; index++)
            if (list[index] > list[index + 1])
            {
                temp = list[index];
                list[index] = list[index + 1];
                list[index + 1] = temp;
            }
    }
}
```

Challenge: Show the memory contents by tracing the values of the variables

Save unnecessary comparisons

Example 10-1

مثال 10-1

برنامج كامل على Bubble Sort

Full Program on Bubble Sort

PP 568 in text

PP 568 في النص