C - Programming

Arrays

19CSE102 Computer Programming

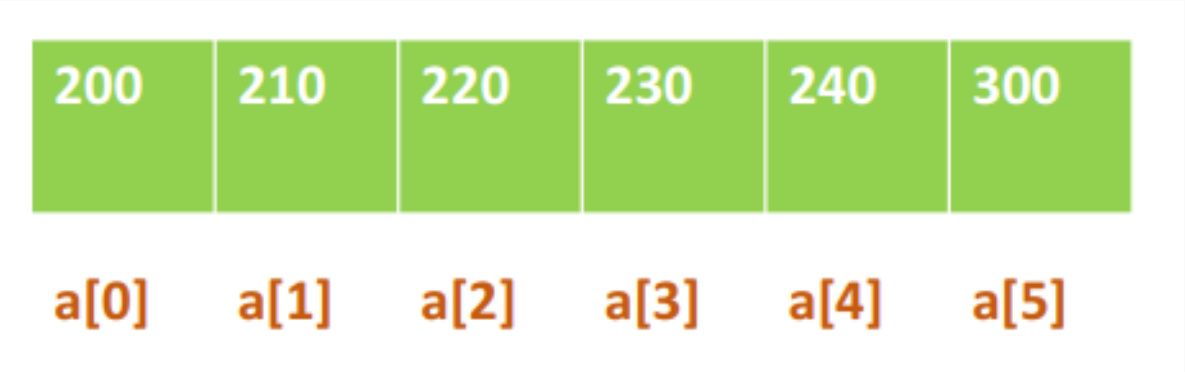
Semester 2, 2020 Batch

Arrays - Introduction

- An array is a collection of data items, all of the same type, and are accessed using a common name.
- In other words, an array is a variable that can store multiple data items of same data type.
- Usually, arrays are used to store and manipulate huge amounts data items (it is not feasible to declare separate variables when the number of data items to be stored is very large)

Arrays

- The locations of the array are accessed using the array name combined with an index. For example, an array a[] that can store six integers can be visualized as follows.



- The number of indices required to access an item in an array is called **dimension of an array**. The above array is a one dimensional array since we need only one index to access any item.

- An array that requires two indices is called a **matrix** (2-D array).

Declaring and Initializing 1-D arrays

- Like variables, the arrays also must be declared before they are used
- General syntax of 1-D array declaration statement
  data-type array-name [*size*];
- Examples

```
int c[6];
int[6] a;
char ch[3];
```

- Following examples show the initialization of 1-D arrays.

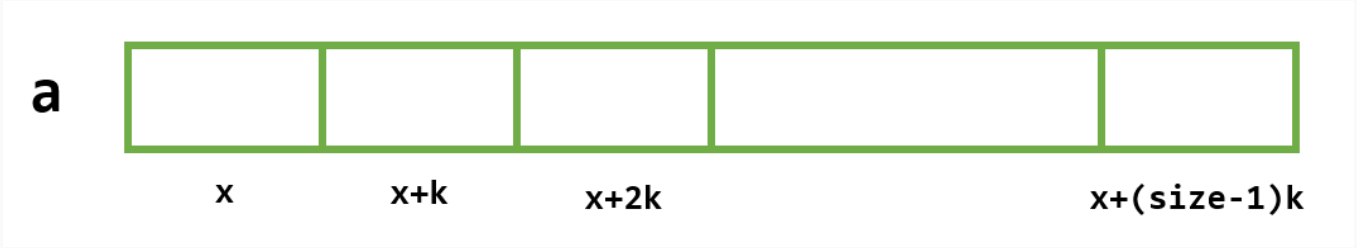
```
int a[6]={10,11,12,13,14,15};
char ch[3]={'a','b','c'};
int b[]={1,2,3,4,5};
int d[10]={0};
```

Points to note

- An array must be declared with three attributes:
  1. type of data
  2. the number of data it can hold (size), and
  3. the identifier (name).
- The array size must be a positive integer number or an expression that evaluates to a positive integer
- The array index starts at 0 and ends at (size–1)
- C never checks whether the array index is valid—either at compile time or when the program is running

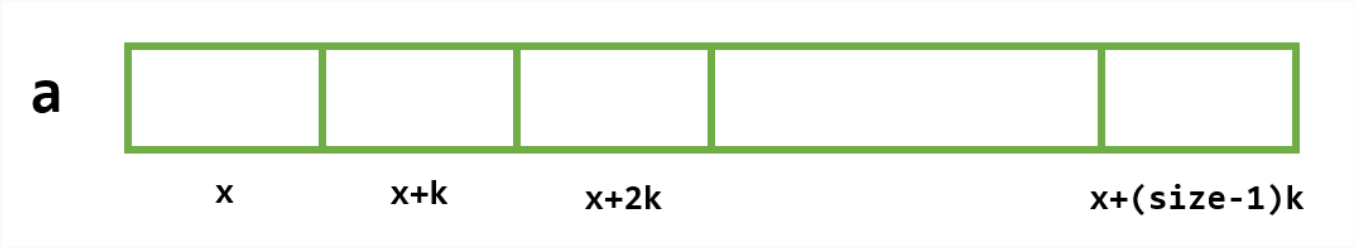
Memory allocation of arrays

- Memory allocation of an array is done in a contiguous manner. That is, Starting from a given memory location, the successive array elements are allocated space in consecutive memory locations.



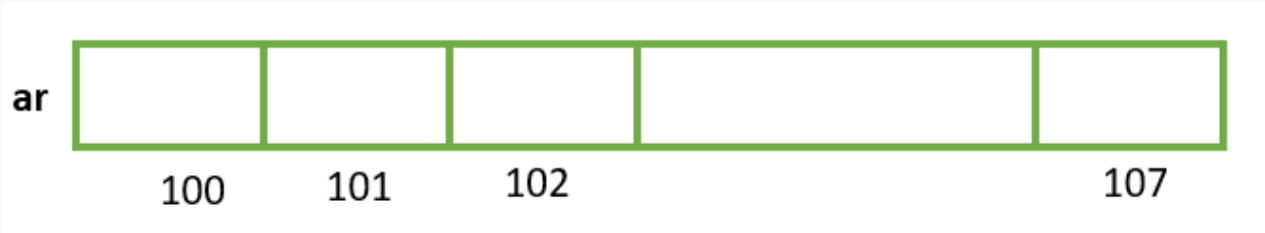
- a is the name of the array
- x is the starting address
- k is the number of bytes required for one element (for example k is 4 for integer array)
- The element a[i] is allocated memory at $x + i * k$

Memory allocation of arrays

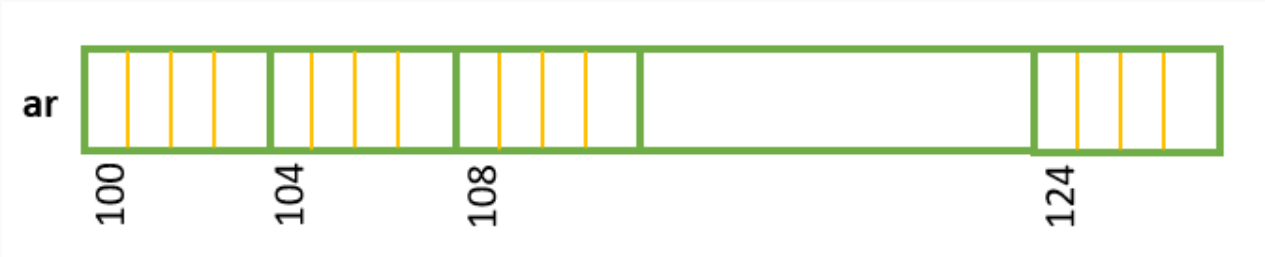


Assume that the memory allocation of the array starts at location 100 (that is x =100) and the the size of the array is 7.

- If the array ar is a character array, then k = 1. The memory allocation is shown below



- If the array ar is an integer array, then k = 4. The memory allocation is shown below



Accessing array elements

The following program shows how the array elements are accessed in C.

```c
int main(){
int ar[10];
/*A for loop to populate the array*/
for(int i=0;i<10;i++) scanf("%d", &ar[i]);

/*A for loop to print the array*/
for(int i=0;i<10;i++) printf("%d", ar[i]);

return 0;
}
```

Problem 1

Write a C program to find the average of the elements in an integer array consisting of 10 elements

Problem 1 - Solution

Write a C program to find the average of the elements in an integer array consisting of 10 elements

```c
int main(){
int ar[10], sum=0;

/*Populating the array */
for(int i=0;i<10;i++)
    scanf("%d", &ar[i]);

/*Adding the array elements*/
for(int i=0;i<10;i++) sum+=ar[i];

printf("Average = %f", sum/(float)10);
return 0;
}
```

Problem 2

Write a C program to find the largest element in an array.

Solution

Write a C program to find the largest element in an array.

```c
int main()
{
    int ar[10], max=0;
    /*Populating the array */
    for(int i=0;i<10;i++) scanf("%d", &ar[i]);

    max=ar[0];
    for(int i=1;i<10;i++)
        if (ar[i]>max) max=ar[i];

    printf("largest element = %d", max);
    return 0;
}
```

Linear Search

To search an element in a given unsorted array.

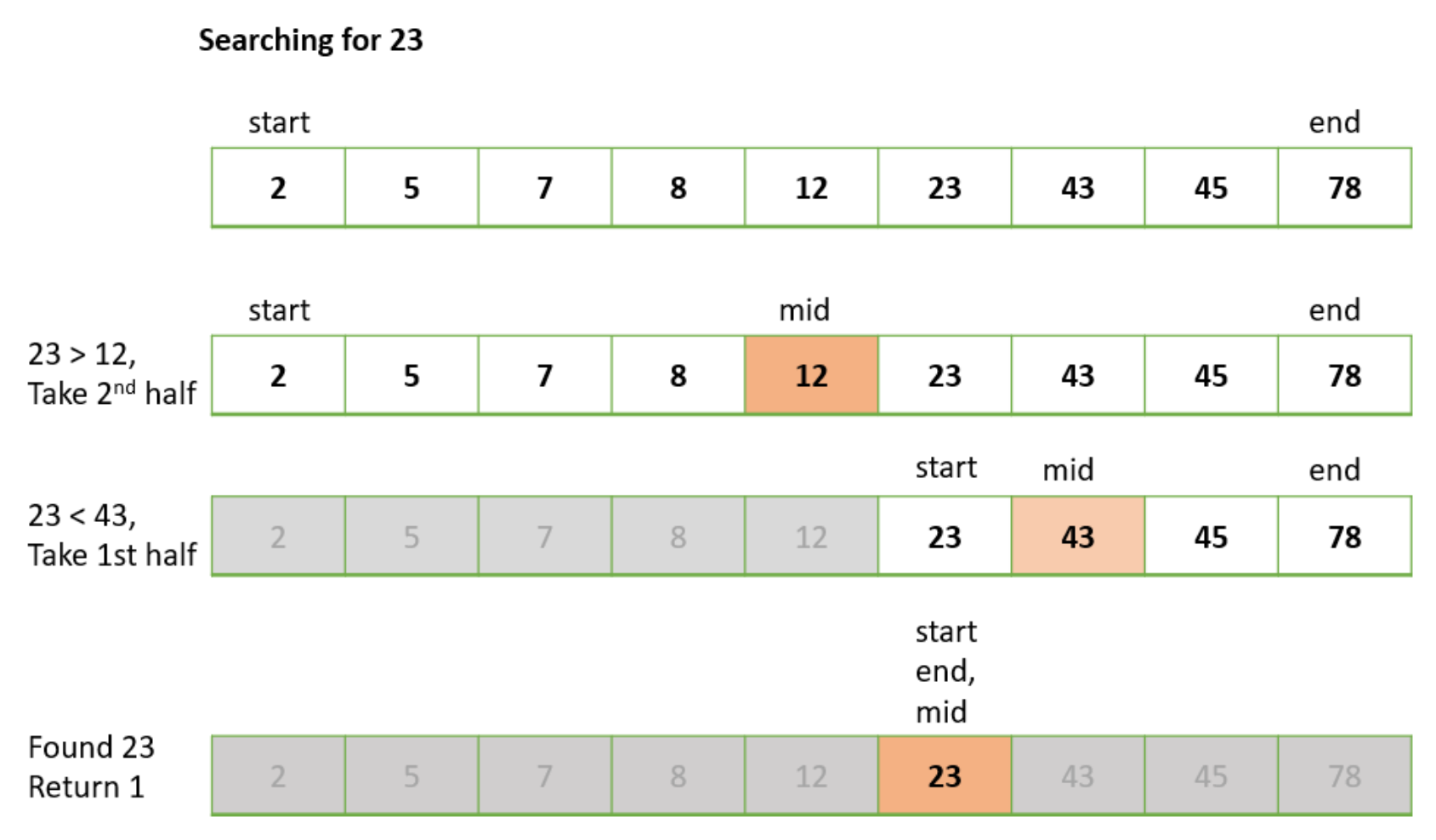
```c
int main()
{
    int n;
    scanf("%d",&n);
    int ar[n], key, flag=0;
    for(int i=0;i<n;i++) scanf("%d", &ar[i]);
    printf("Enter the search element");
    scanf("%d",&key);

    for(int i=1;i<n;i++)
        if (ar[i]==key) {flag=1; break;}
    if (flag)
        printf("The element found");
    else
        printf("The element NOT found");
    return 0;
}
```

Binary Search

- Used to search an element in a sorted array.
- Checks the middle element in the given array. If it is not the key, then any one half of the array can be ignored.
- In every step we reduce the size of the array by half. This improves the efficiency.

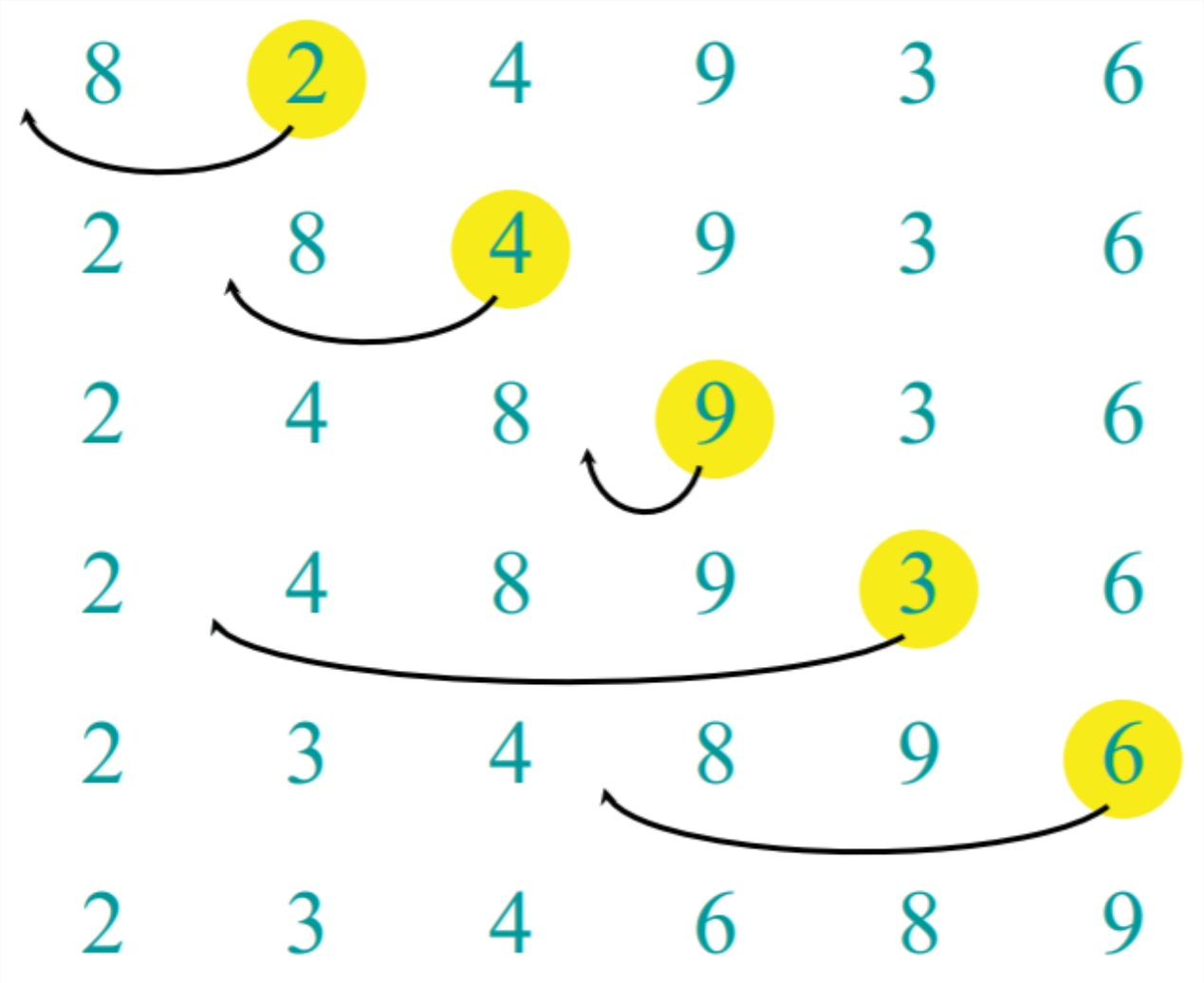
```
1  int binary_search (int x[], int start,int end, int key)
2  {
3      if(start>end) return -1;
4      mid = (start+end) / 2;
5      if (x[mid]== key) return 1;
6      else if (x[mid]>key)
7          return binary_search (x,start,mid-1,key);
8      else return binary_search (x,mid+1,end,key);
9  }
```

Binary Search example



Insertion Sort

Used to sort an array. Let us see an example first.



Insertion Sort

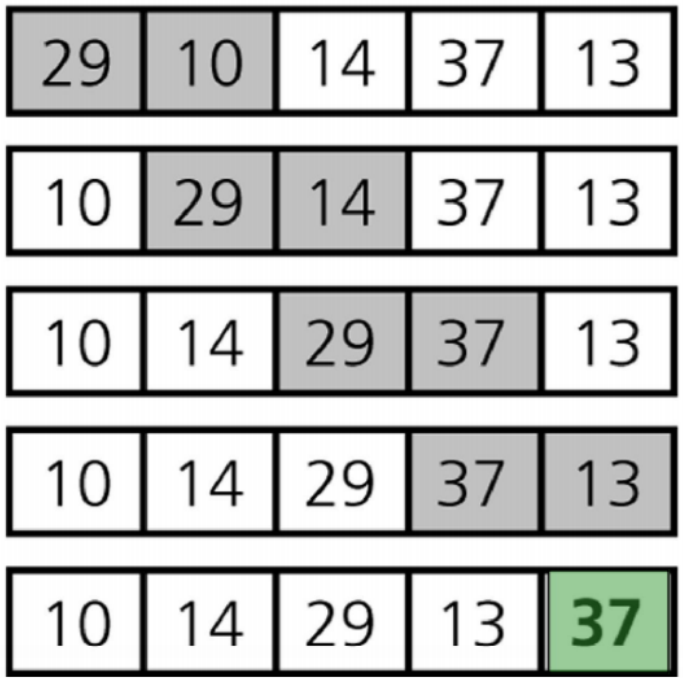
- The outer loop takes the element (from the unsorted part) to be inserted.
- The inner loop moves the elements so as to make a space for the chosen element to be inserted, in the array.

```
1  void insertionSort(int array[], int size) {
2    for (int pass = 1; pass < size; pass++)
3    {
4          int key = array[pass];
5          int j = pass - 1;
6          while (key < array[j] && j >= 0)
7          {
8              array[j + 1] = array[j];
9              j--;
10         }
11     array[j + 1] = key;
12   }
13 }
```

Bubble Sort

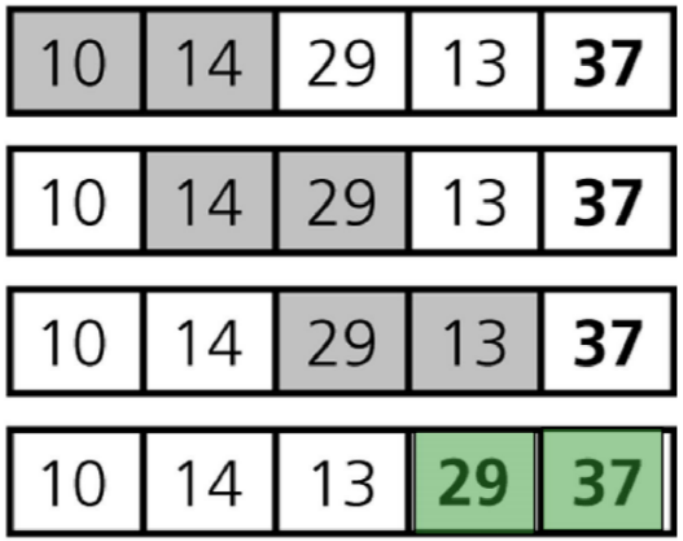
Used to sort an array. Let us see an example first.

(a) Pass 1



At the end of **Pass 1**, the largest item **37** is at the last position.

(b) Pass 2



At the end of **Pass 2**, the second largest item **29** is at the second last position.

Bubble Sort

- In each pass (one iteration of the outer loop), the largest element in the unsorted array is fixed.
- The inner loop compares the elements in pairs to find the largest element in each pair

```
1  void bubbleSort(int arr[], int n)
2  {
3      int i, j, temp;
4      for (i = 0; i < n-1; i++){
5          for (j = 0; j < n-i-1; j++){
6              if (arr[j] > arr[j+1])
7              {
8                  temp=arr[j];
9                  arr[j]=arr[j+1];
10                 arr[j+1]=temp;
11             }
12         }
13     }
14 }
```