Slots 18-19-20-21
Contiguous Storage

Module F: Arrays, Simple Data Structure
Contiguous Storage
Searching
Sorting

Objectives

- How to manage a group of data?
  - Store
  - Input
  - Output
  - Search
  - Sort
  - …

Content

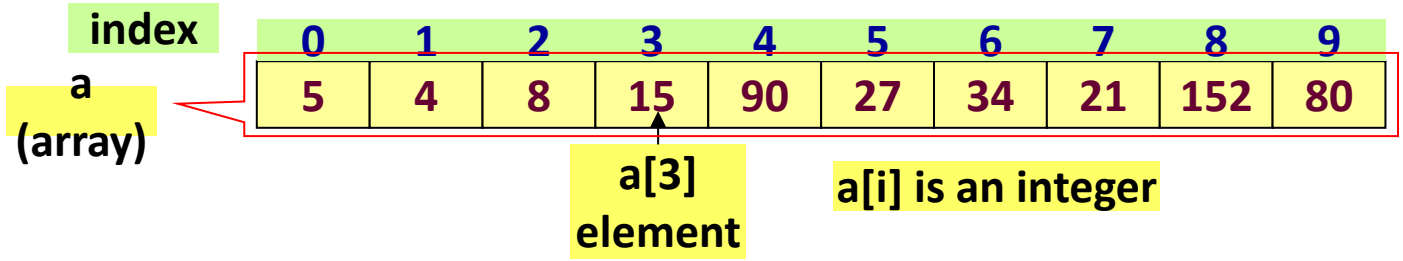
- Introduction to contiguous storage
- Arrays
- One-dimensional Arrays
  - Declaration
  - Memory Allocation
  - Initialization
  - Accessing elements
  - Traversing
  - 1-D Arrays are parameters of functions
  - Searching
  - Sorting
- 2-D Arrays

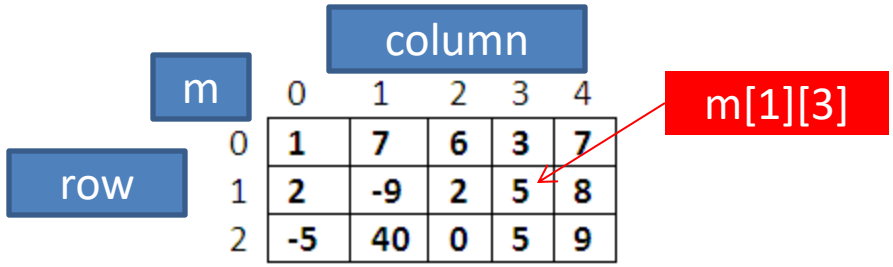
1- Contiguous Storage

- Commonly, a group of the same meaning elements are considered.
- They are stored in a contiguous block of memory.
- Ex: Group of 10 int numbers → 40 bytes block is needed.
- Data are considered can be a group of some items which belong to some different data types → Contiguous memory block is partitioned into some parts which have different size, one part for an item.
- Data structure: A structure of data stored.
- Array is the simplest data structure which contains some items which belong to the sane data type.
- Common used operations on a group: Add, Search, Remove, Update, Sort

2- Arrays

Array: A group of elements which belong to the same data type. Each element is identified by it's position (index).



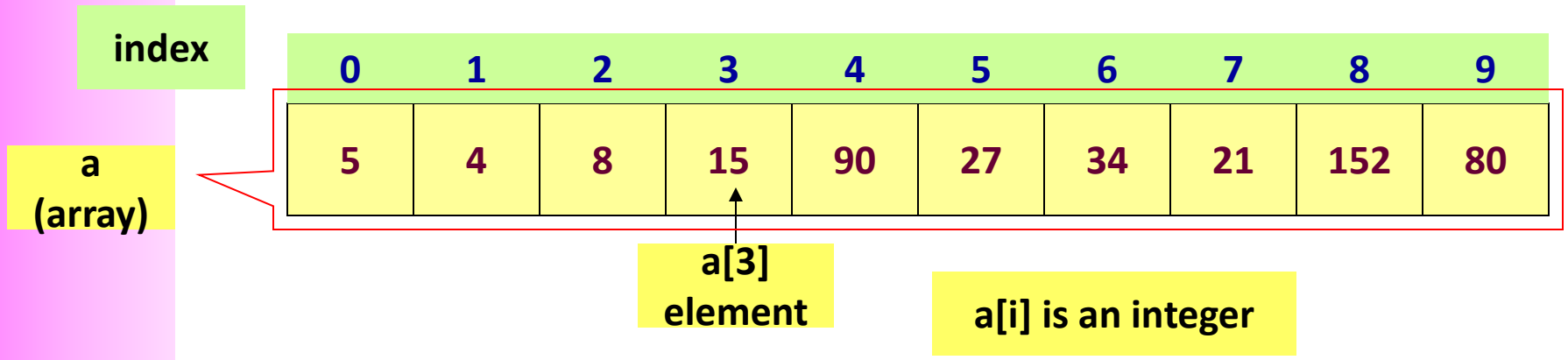


- ***Dimension***: Direction that is used to perform an action on array.
- ***Number of dimensions***: Number of indexes are used to specify an element.
- ***Common arrays***: 1-D and 2-D arrays.
- ***Name of an array***: An array has it's name.

3- One Dimensional (1-D)Arrays

- 1-D array: a collection of items (elements, terms) which belong to the same data type and are stored contiguously in memory.

→Each element is identified by a unique index of it's position in the array (an integer from 0).



1-D Arrays: Declaration

- If the array is stored in the stack segment → Use a STATIC array → The compiler will determine the array's storage at compile-time.
- If the array is stored in the heap → Use a pointer (DYNAMIC array) → The array's storage will be allocated in the heap at run-time through memory allocating functions (malloc, calloc, realloc)

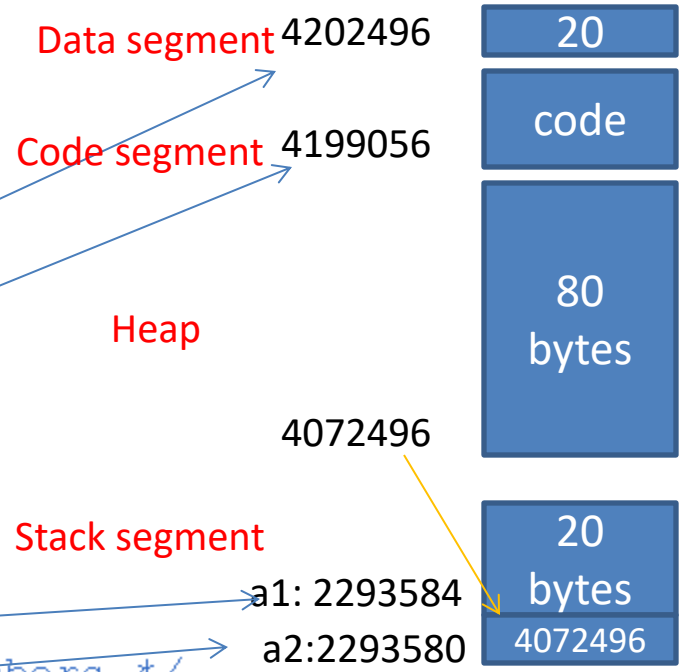
```
DataType  ArrayName[NumberOfElements] ;

Examples:
int  a1[5];
char  s[12];
double  a3[100];
```

How compilers can determine the memory size of an array?
NumberOfElements * sizeof(dataType)
→ int a1[5] → 5 *sizeof(int) = 5*4 = 20 bytes

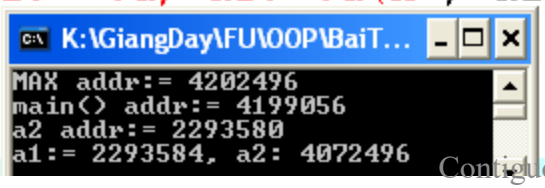
```
float  *a;
a = (float*)calloc (10, sizeof(float)); /* allocate a block of 10 float numbers */
```

1-D Arrays: Memory Allocation

The name of the array is the address of the first element.




```
#include <stdio.h>
#include <stdlib.h>
int MAX=20;
int main()
{   int a1[5]; /* static array of 5 int numbers */
    double *a2=NULL; /* dynamic array of double numbers */
    printf ("MAX addr:= %u\n", &MAX);
    printf ("main() addr:= %u\n", &main);
    /* allocate a mem. block for 10 double numbers */
    a2 = (double*)calloc(10,sizeof(double));
    printf ("a2 addr:= %u\n", &a2);
    printf ("a1:= %u, a2: %u\n", a1, a2);
    getchar();
    return 0;
}
```



1-D Arrays: Initialization & Accessing Elements

Initialize an array:

Type a[] = {val1, val2, … };

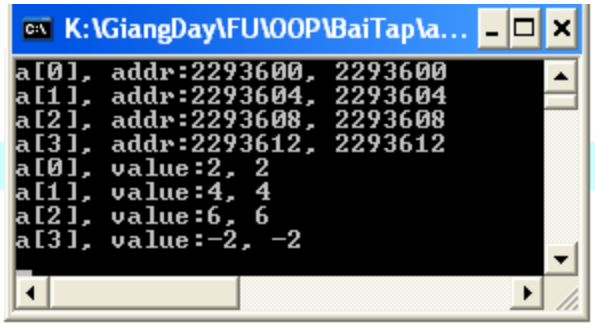
How to access the i^{th} element of the array a?

- ***a*** is the address of the first element. Based on operation on pointers:
  - → ***a+i*** : address of the i^{th} element, another way: ***&a[i]***
  - → ****(a+i)***: value of the i^{th} element, another way: ***a[i]***

1-D Arrays: Init. & Accessing…

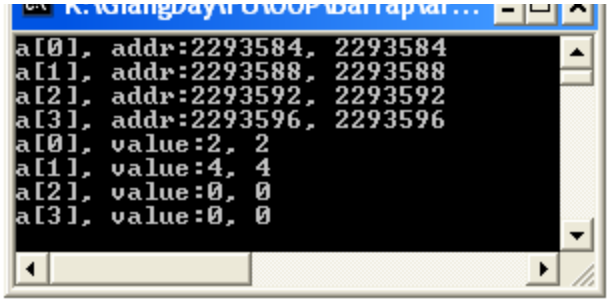
Compiler will automatically count number of initial values to determine the size of array memory

```
#include <stdio.h>n
#include <stdlib.h>
int main()
{   int a[]= {2,4,6,-2};
    int i;
    for (i=0;i<4;i++)
      printf("a[%d], addr:%u, %u\n", i, a+i, &a[i]);
    for (i=0;i<4;i++)
      printf("a[%d], value:%d, %d\n", i, *(a+i), a[i]);
    getchar();
    return 0;
}
```



The size of array memory is pre-defined.
Compiler will fill 0 to elements which are not initialized.

```
#include <stdio.h>n
#include <stdlib.h>
int main()
{   int a[5]= {2,4};
    int i;
    for (i=0;i<4;i++)
      printf("a[%d], addr:%u, %u\n", i, a+i, &a[i]);
    for (i=0;i<4;i++)
      printf("a[%d], value:%d, %d\n", i, *(a+i), a[i]);
    getchar();
    return 0;
}
```



int a[5];
Elements contain un-predictable values because they are local variables.
TEST IT !!!!

1-D Arrays: Traversing

- A way to visit each element of an array
- Suppose that the 1-D array, named ***a***, containing ***n*** elements.
- ***Forward traversal***:

```
int i;
for (i=0; i<n; i++)
{   [if (condition)] Access a[i];
}
```

- ***Backward traversal***:

```
int i;
for (i=n-1; i >=0; i--)
{  [if (condition)] Access a[i];
}
```

1-D Array is a Function Parameter

- The array parameter of a function is the pointer of the first element of the array.
- ***Input an array of n integers***

void input (int* a, int n)

- Input elements of an array of integers which it's number of element is stored at the pointer ***pn***

void input (int a[], int*pn)

- Output an array of n double numbers

void output (double a[], int n)

- Calculate the sum of an array of n integers

int sum (int *a, int n)

Array Function Parameter: Demo.

- **Develop a C-program that will:**
  -Accept values to an integer array that may contain 100 elements.
  - Print out the it's maximum value.
  - Print out it's elements.
  - Print out it's even values.

- **Nouns:**
  – Constant: MAXN=100
  –Static array of integers → int a[MAXN]
  –Real number of elements → int n
  –Maximum value → int maxVal.

- **Verbs:**
  -Begin
  -Input n (one value)
  -Input a, n (**function**)
  - maxVal = get maximum value in a, n (**function**)
  - Print out maxVal (one value)
  - Print out a, n (**function**)
  - Print even values in a, n (**function**)
  - End

Array Function Parameter: Demo 1.

```c
/* Array Parameters Demo.*/
#include <stdio.h>
#define MAXN 100
/* Prototypes */
void input(int*a, int n);
int max(int a[], int n);
void print (int* a, int n);
void printEven (int* a, int n);
int main()
{   int a[MAXN]; /* static array of 100 integers */
    int n; /* real used number of elements */
    int maxVal;
    do
    {  printf("How many elements which will be used 1..%d:", MAXN);
       scanf("%d", &n);
    }
    while (n<1 || n>MAXN);
    printf("Enter %d values of the arrays:\n", n);
    input(a,n);
    maxVal  = max (a,n);
    printf("Max value:%d\n", maxVal);
    printf("\nInputted array:");
    print(a,n);
    printf("\nEven values in array:");
    printEven(a,n);
    while (getchar()!='\n');getchar();
    return 0;
}
```

```
K:\GiangDay\FU\OOP\BaiTap\array01.exe
How many elements which will be used 1..100:6
Enter 6 values of the arrays:
3 5 8 1 2 0
Max value:8

Inputted array:3  5  8  1  2  0
Even values in array:8  2  0
```

Array Function Parameter: Demo 1.

```
29 void input(int*a, int n)
30 { /* Use forward traversal, accept each value */
31   int i;
32   for (i=0; i<n; i++) scanf("%d", &a[i]);
33 }
34 int max(int a[], int n)
35 { int result = a[0];
36   /* Use forward traversal, compare each value with result */
37   int i;
38   for (i=1; i<n; i++)
39      if (result<a[i]) result=a[i];
40   return result;
41 }
42 void print (int* a, int n)
43 { /* Use forward traversal, print out each value */
44   int i;
45   for (i=0; i<n; i++) printf("%d  ", a[i]);
46 }
47 void printEven (int* a, int n)
48 { /* Use forward traversal, print out each value */
49   int i;
50   for (i=0; i<n; i++)
51     if (a[i]%2==0) printf("%d  ", a[i]);
52 }
```

Array Function Parameter: Demo 1.

- Comments
  - If you allocate an array having 100 elements but 6 elements are used then memory is wasted.
  - If If you allocate an array having 100 elements but 101 elements are used then there is a lack of memory.
- Solution: Use a dynamic array.

Array Function Parameter: Demo 1.

```c
/* Array Parameters Demo.*/
#include <stdio.h>
#define MAXN 100
/* Prototypes */
void input(int*a, int n);
int max(int a[], int n);
void print (int* a, int n);
void printEven (int* a, int n);
int main()
{   int a[MAXN]; /* static integer */
    int n; /* real used number of elements */
    int maxVal;
    do
    {  printf("How many elements which will be used 1..%d:", MAXN);
       scanf("%d", &n);
    }
    while (n<1 || n>MAXN);
    printf("Enter %d values of the arrays:\n", n);
    input(a,n);
    maxVal  = max (a,n);
    printf("Max value:%d\n", maxVal);
    printf("\nInputted array:");
    print(a,n);
    printf("\nEven values in array:");
    printEven(a,n);
    while (getchar()!='\n');getchar();
    return 0;
}
```

int* a

replace

insert

a = (int*) calloc (n, sizeof(int));

```
K:\GiangDay\FU\OOP\BaiTap\array01.exe
How many elements which will be used 1..100:6
Enter 6 values of the arrays:
3 5 8 1 2 0
Max value:8

Inputted array:3  5  8  1  2  0
Even values in array:8  2  0
```

Other functions are preserved.

Array Function Parameter: Demo 1.

```
29 void input(int*a, int n)
30 { /* Use forward traversal, accept each value */
31   int i;
32   for (i=0; i<n; i++) scanf("%d", &a[i]);
33 }
34 int max(int a[], int n)
35 { int result = a[0];
36   /* Use forward traversal, compare each value with result */
37   int i;
38   for (i=1; i<n; i++)
39      if (result<a[i]) result=a[i];
40   return result;
41 }
42 void print (int* a, int n)
43 { /* Use forward traversal, print out each value */
44   int i;
45   for (i=0; i<n; i++) printf("%d  ", a[i]);
46 }
47 void printEven (int* a, int n)
48 { /* Use forward traversal, print out each value */
49   int i;
50   for (i=0; i<n; i++)
51     if (a[i]%2==0) printf("%d  ", a[i]);
52 }
```

Array Function Parameter: Demo 2.

- **Develop a C-program that will:**

  -Accept values to an integer array that may contains 100 elements. The input will terminate when user enters the value of zero.

  - Print out the it's maximum value.

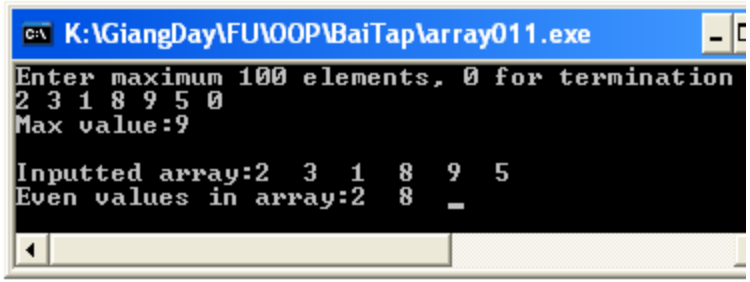
  - Print out it's elements.

  - Print out it's even values.

The difference between this problem with the previous one is the input operation can terminate abruptly when 0 is accepted.
➔Memory block of the array needs to be allocated in excess
➔The function for input values of the array must be modified for this case and the number of elements is updated after each valid value is accepted.

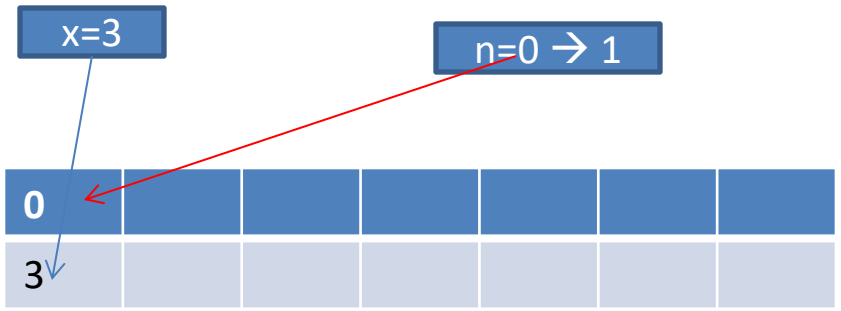
Array Function Parameter: Demo 2.

```
2  #include <stdio.h>
3  #define MAXN 100
4  /* Input an array, number of elements is stored at pn
5     User will terminate inputting when 0 is entered.*/
6  void input(int*a, int *pn);
7  int max(int a[], int n);
8  void print (int* a, int n);
9  void printEven (int* a, int n);
10 int main()
11 {   int a[MAXN]; /* static array of 100 integers */
12     int n; /* real used number of elements */
13     int maxVal;
14     input(a,&n);
15     maxVal  = max (a,n);
16     printf("Max value:%d\n", maxVal);
17     printf("\nInputted array:");
18     print(a,n);
19     printf("\nEven values in array:");
20     printEven(a,n);
21     while (getchar()!='\n');getchar();
22     return 0;
23 }
```

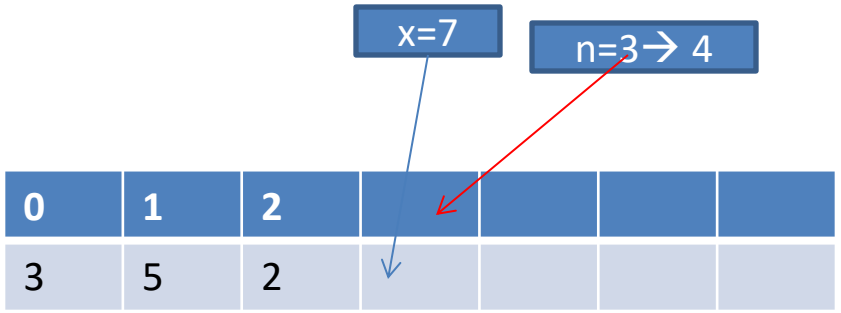


Array Function Parameter: Demo 2.

```
24 void input(int*a, int *pn)
25 { *pn=0; /* reset the number of elements */
26   printf ("Enter maximum %d elements, 0 for termination\n", MAXN);
27   int x; /* inputted value */
28   do
29   { scanf("%d", &x);
30     if (x!=0) a[(*pn)++] = x;
31   }
32   while (x!=0 && *pn < MAXN);
33 }
```




```
34 int max(int a[], int n)
35 {
     /* Do yourself */
42 }
43 void print (int* a, int n)
44 { /* Do yourself */
47 }
48 void printEven (int* a, int n)
49 { /* Do yourself */
53 }
```



1-D Arrays: Searching

- A search algorithm finds the record of interest using the key array
- Return value: The positional index at which the interest value is found.
- Two common search algorithms are
  - linear search
  - binary search

1-D Arrays: Searching…

Linear search: Find the position of the value x in the array a having n elements.

Search the value of 6 in the array a having 8 items.

5	9	2	7	6	5	2	5
i=0	1	2	3	**4**			

Search the value of 12 in the array a having 8 items.

5	9	2	7	6	5	2	5	
i=0	1	2	3	4	5	6	7	**-1**

There may be n comparisons performed.

```
int  firstLinearSearch ( int x, int a[], int n)
{  int i;
   for ( i=0;  i<n;  i++)
      if ( x == a[i] ) return i;
   return -1;
}
```

```
int  lastLinearSearch ( double x, double *a, int n)
{  int i;
   for ( i=n-1;  i>=0;  i--)
      if ( x == a[i] ) return i;
   return -1;
}
```

1-D Arrays: Linear Searching…

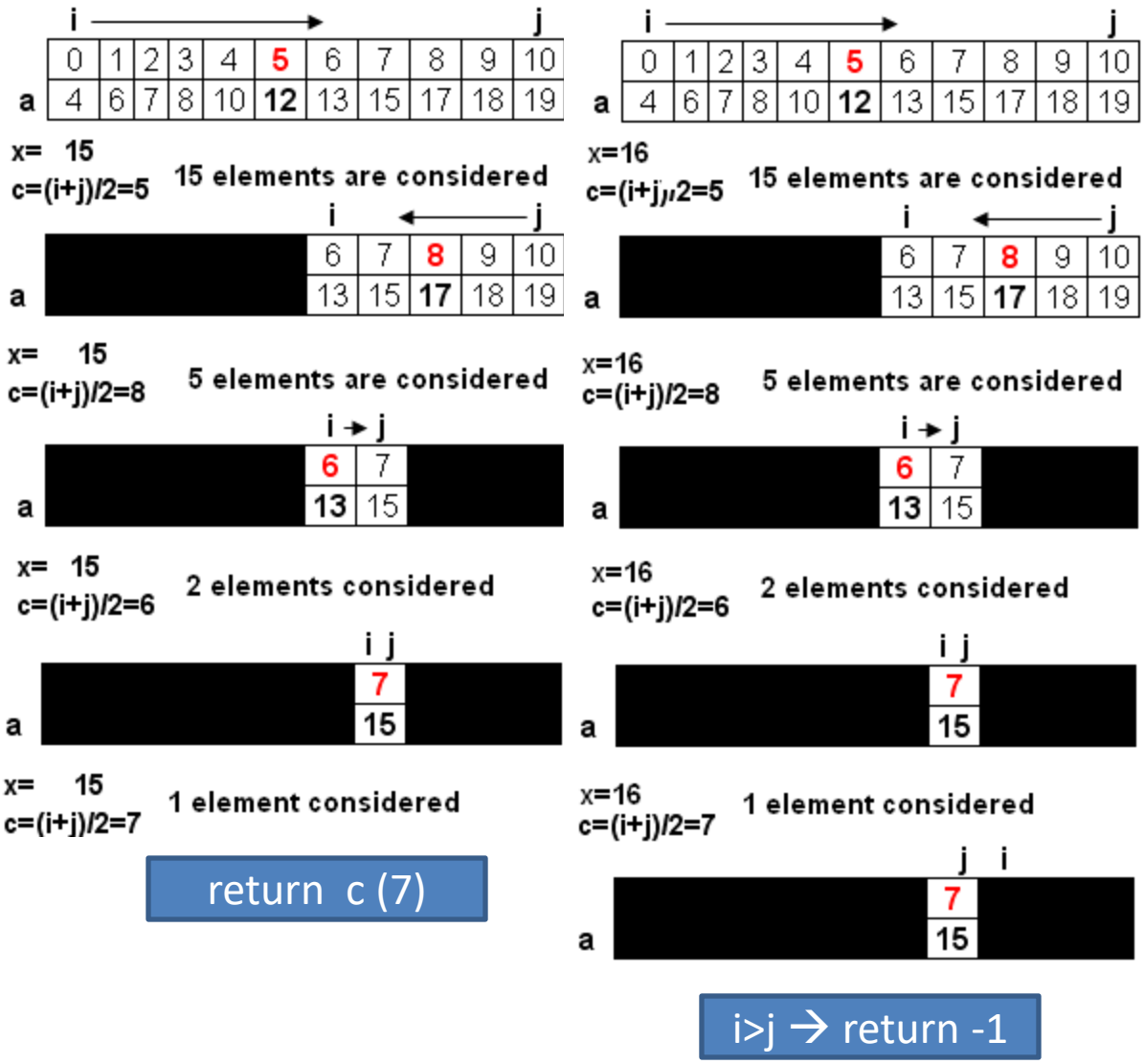
```
/* Linear search Demo. */
#include <stdio.h>
int  firstLinearSearch ( int x, int a[], int n)
{
    /* Your code */
}
int  lastLinearSearch ( int x, int a[], int n)
{
    /* Your code */
}
int main()
{   int a[] = { 3,34,5,1,2,8,9,2,9 }, x=2;
    int pos1= firstLinearSearch(x,a,9);
    if (pos1>=0)
    {   int pos2= lastLinearSearch(x,a,9);
        printf("First existence:%d, last existence:%d\n", pos1, pos2);
    }
    else printf("%d does not exist!\n", x);
    getchar();
    return 0;
}
```

1-D Arrays: Binary Searching…

Binary search

- Condition for application: Values in the array were sorted.

```
int  binarySearch ( int x, int a[], int n)
{  int i=0, j= n-1, c ;
   while (i<=j)
    {   c= (i+j)/2;
        if ( x== a[c] ) return c ;
        if (x < a[c] ) j = c-1;
        else i = c +1;
    }
   return -1;
}
```



1-D Arrays: Binary Searching…

```
#include <stdio.h>
int  binarySearch (int x, int a[], int n)
{
  /* YOUR CODE */
}
int main()
{   int a[] = { 1, 4, 8, 10, 12, 16, 22, 24 };
    int n=8, k1= 22, k2= 7;
    int pos1= binarySearch(k1,a,n);
    int pos2= binarySearch(k2,a,n);
    if (pos1>=0) printf("\nPosition of value %d is: %d", k1, pos1);
    else printf("\n%d does not exist!", k1);
    if (pos2>=0) printf("\nPosition of value %d is: %d", k2, pos2);
    else printf("\n%d does not exist!", k2);
    getchar();
    return 0;
}
```

Evaluation:

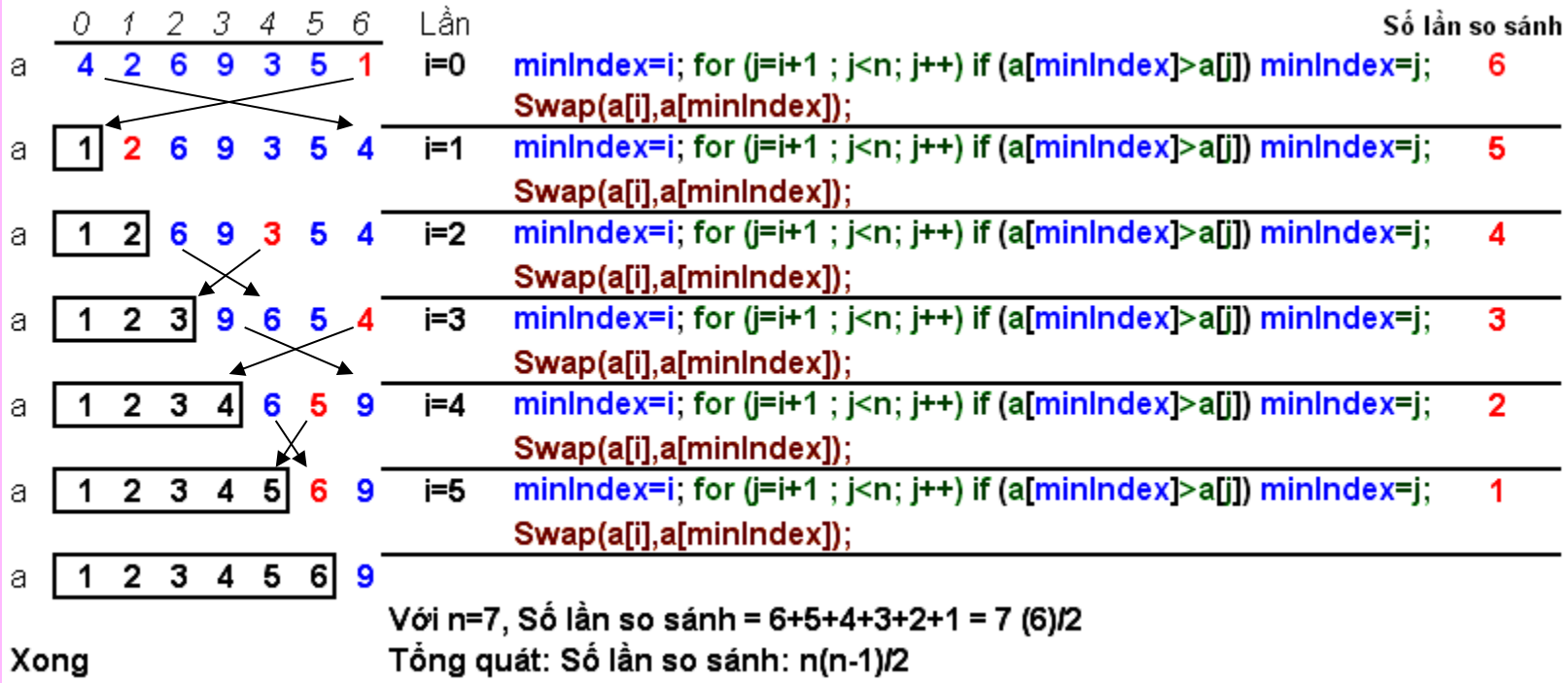
No. of elements considered	No. of comparisons
$n= 2^m$	1
2^{m-1}	1
2^{m-2}	1
…	…
2^0	1
Sum	$m+1 = \log_2(n) +1$

1-D Arrays: Sorting

- Sorting: Changing positions of elements in an array so that values are in a order based on a pre-defined order relation.
- Default order relation in set of numbers: Value order
- Default order relation in a set of characters/ strings: Dictionary order
- Only two sorting algorithms are introduced here.
  - Selection Sort
  - Bubble Sort

1-D Arrays: Selection Sort

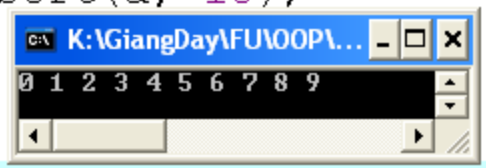
- Find the minimum value in the list
- Swap it with the value in the first position
- Repeat the steps above for remainder of the list



1-D Arrays: Selection Sort

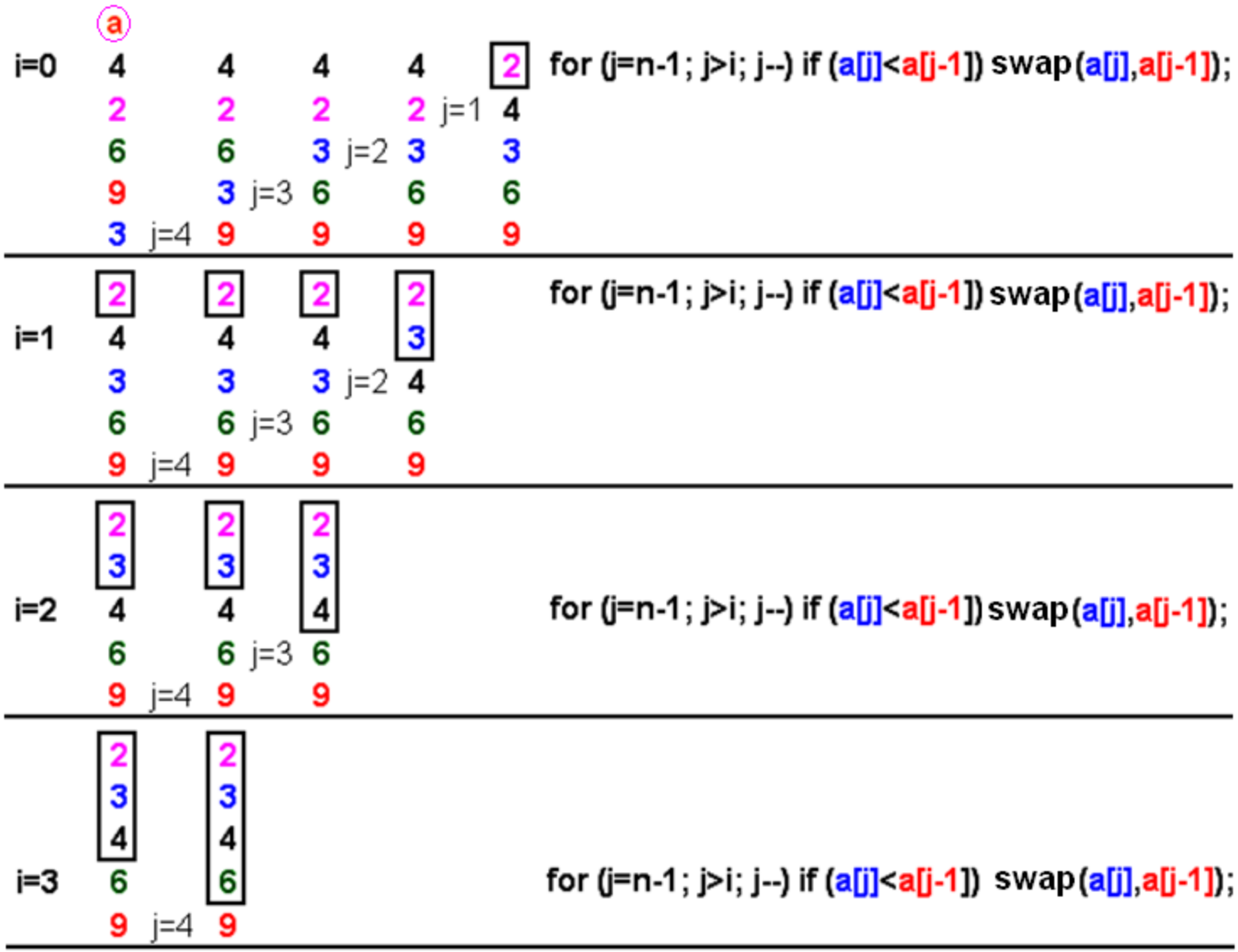
```c
#include <stdio.h>
void ascSelectionSort( int* a, int n)
{ int minIndex; /* index of min. value in a group */
  int i,j ; /* vars for looping */
  /* Group begins at position i to n-1*/
  for (i=0; i< n-1; i++)
  {  minIndex = i; /* init minimum position */
     /* update minIndex of the group at i, i+1,..., n-1*/
     for (j=i+1; j<n; j++)if (a[minIndex]> a[j]) minIndex= j;
     /* Move minimum value to the begin of the group */
     if (minIndex > i)
     {  int t = a[minIndex];
        a[minIndex] = a[i];
        a[i] = t;
     }
  }
}
```

```c
void print (int*a, int n)
{
    /* Your code */

}
int main()
{  int a[] = { 1,3,5,7,9,2,4,6,8, 0 };
   ascSelectionSort(a, 10);
   print(a,10);
   getchar();
   return 0;
}
```



1-D Arrays: Bubble Sort

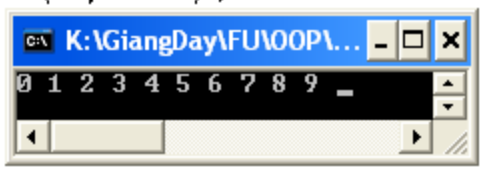
- It works by repeatedly stepping through the list to be sorted, comparing two items at a time and swapping them if they are in the wrong order.
- The pass through the list is repeated until no swaps are needed, which means the list is sorted



1-D Arrays: Bubble Sort…

```
#include <stdio.h>
void ascBubbleSort( int* a, int n)
{ int i,j ; /* vars for looping */
  /* Loop n-1 pass */
  for (i=0; i< n-1; i++)
  {  /* Go to the end of array to move the min value up */
     for (j=n-1; j>i; j--)
        /*The later element is smaller than the previous one*/
        if (a[j]<a[j-1])
        {  /* move the smaller up */
           int t = a[j];
           a[j] = a[j-1];
           a[j-1] = t;
        }
  }
}
```

```
void print (int*a, int n)
{ int i;
  for (i=0; i<n; i++)printf("%d ", a[i]);
}
int main()
{  int a[] = { 1,3,5,7,9,2,4,6,8, 0 };
   ascBubbleSort(a, 10);
   print(a,10);
   getchar();
   return 0;
}
```



1-D Arrays: A Sample

- Develop a C-program that helps user managing an 1-D array of integers (maximum of 100 elements) using the following simple menu:
- 1- Add a value
- 2- Search a value
- 3- Remove the first existence of a value
- 4- Remove all existences of a value
- 5- Print out the array
- 6- Print out the array in ascending order (positions of elements are preserved)
- 7- Print out the array in descending order (positions of elements are preserved)
- Others- Quit

1-D Arrays: A Sample…

- In this program, user can freely add or remove one or more elements to/ from the array. So, an extra memory allocation is needed (100 items).
- **Data**:

  Array of integers → int a[100], n

  searched/added/removed number → int value
- **Functions**:
  - int **menu**() → Get user choice
  - int isFull(int *a, int n) - Testing whether an array is full or not
  - int isEmpty(int *a, int n) - Testing whether an array is empty or not
  - void **add**(int x, int*a, int*pn) → adding an element to the array will increase number of elements
  - int **search**(int x, int *a, int n) → return a position found in the array
  - int **removeOne** (int pos, int*a, int*pn) → Removing a value at the position pos will decrease number of elements → return 1: successfully, 0: fail
  - int **remove All**(int x, int*a, int*pn) → Removing a value will decrease number of elements → return 1: successfully, 0: fail
  - void **print**Asc(int*a, int n) – printing array, elements are preserved
  - void **printDesc**(int*a, int n) – printing array, elements are preserved
  - void **print**(int*a, int n)

1-D Arrays: A Sample…

```
K:\GiangDay\FU\OOP\BaiTap\Array_Sample0...

One-Dimensional Array of Integers
1- Add  a value
2- Search a value
3- Remove the first existence of a value
4- Remove all existences of a value
5- Print out the array
6- Print out the array in ascending order
7- Print out the array in descending order
Others- Quit
Select:1
Input an added value:0
Added

One-Dimensional Array of Integers
1- Add  a value
2- Search a value
3- Remove the first existence of a value
4- Remove all existences of a value
5- Print out the array
6- Print out the array in ascending order
7- Print out the array in descending order
Others- Quit
Select:1
Input an added value:2
Added
```

```
One-Dimensional Array of Integers
1- Add  a value
2- Search a value
3- Remove the first existence of a value
4- Remove all existences of a value
5- Print out the array
6- Print out the array in ascending order
7- Print out the array in descending order
Others- Quit
Select:5
0 2 8 9 7 3 2 4 2
One-Dimensional Array of Integers
```

After values 0 , 2, 8, 9, 7, 3, 2, 4, 2 are added. Use menu 5 to view them.

```
0 2 8 9 7 3 2 4 2
One-Dimensional Array of Integers
1- Add  a value
2- Search a value
3- Remove the first existence of a value
4- Remove all existences of a value
5- Print out the array
6- Print out the array in ascending order
7- Print out the array in descending order
Others- Quit
Select:2
Input the searched value:2
Postion is found:1
```

Search option

```
0 2 8 9 7 3 2 4 2
One-Dimensional Array of Integers
1- Add  a value
2- Search a value
3- Remove the first existence of a value
4- Remove all existences of a value
5- Print out the array
6- Print out the array in ascending order
7- Print out the array in descending order
Others- Quit
Select:3
Input the removed value:8
Removed!

One-Dimensional Array of Integers
1- Add  a value
2- Search a value
3- Remove the first existence of a value
4- Remove all existences of a value
5- Print out the array
6- Print out the array in ascending order
7- Print out the array in descending order
Others- Quit
Select:5
0 2 9 7 3 2 4 2 2
```

Remove one option

1-D Arrays: A Sample…

```
0 2 9 7 3 2 4 2 2
One-Dimensional Array of Integers
1- Add  a value
2- Search a value
3- Remove the first existence of a value
4- Remove all existences of a value
5- Print out the array
6- Print out the array in ascending order
7- Print out the array in descending order
Others- Quit
Select:4
Input a value that will be remove all:2
Removed!

One-Dimensional Array of Integers
1- Add  a value
2- Search a value
3- Remove the first existence of a value
4- Remove all existences of a value
5- Print out the array
6- Print out the array in ascending order
7- Print out the array in descending order
Others- Quit
Select:5
0 9 7 3 4
```

Remove all option

```
0 3 4 7 9
One-Dimensional Array of Integers
1- Add  a value
2- Search a value
3- Remove the first existence of a value
4- Remove all existences of a value
5- Print out the array
6- Print out the array in ascending order
7- Print out the array in descending order
Others- Quit
Select:5
0 9 7 3 4
```

Print out in ascending order (elements are preserved)

```
0 9 7 3 4
One-Dimensional Array of Integers
1- Add  a value
2- Search a value
3- Remove the first existence of a value
4- Remove all existences of a value
5- Print out the array
6- Print out the array in ascending order
7- Print out the array in descending order
Others- Quit
Select:7
9 7 4 3 0
One-Dimensional Array of Integers
1- Add  a value
2- Search a value
3- Remove the first existence of a value
4- Remove all existences of a value
5- Print out the array
6- Print out the array in ascending order
7- Print out the array in descending order
Others- Quit
Select:5
0 9 7 3 4
```

Print out in descending order (elements are preserved)

1-D Arrays: A Sample…

```c
/* Array_Sample01.c    1-D Array Demonstration */
#include <stdio.h>
#include <stdlib.h>
#define MAXN 100
 /* Get user choice*/
int menu()
{  printf("\nOne-Dimensional Array of Integers");
   printf("\n1- Add  a value");
   printf("\n2- Search a value");
   printf("\n3- Remove the first existence of a value");
   printf("\n4- Remove all existences of a value");
   printf("\n5- Print out the array");
   printf("\n6- Print out the array in ascending order");
   printf("\n7- Print out the array in descending order");
   printf("\nOthers- Quit");
   printf("\nSelect:");
   int choice;
   scanf("%d", &choice);
   return choice;
}
/* Testing whether an array is full or not */
int isFull (int*a, int n)
{  return n==MAXN;
}
/* Testing whether an array is empty or not */
int isEmpty (int*a, int n)
{  return n==0;
}
```

1-D Arrays: A Sample…

```
29 /*adding an element to the the end of array will increase number of elements */
30 void add(int value, int*a, int*pn)
31 { a[*pn] = value ; /* add it to the end of the array */
32   (*pn)++;
33 }
34 /* Find the first existence of x in the array - Linear searching */
35 int search(int x, int *a, int n)
36 {   int i;
37    for (i= 0; i<n; i++) if (a[i]==x) return i;
38    return -1;
39 }
40 /*Removing the element at a position in an array will decrease number of elements
41   return 1: remove successfully, 0: remove fail*/
42 int removeOne (int pos, int*a, int*pn)
43 {   if (pos<0 || pos >=*pn) return 0;
44    int i;
45    for (i=pos; i<*pn-1; i++) a[i]=a[i+1];
46    (*pn)--; /* decrease number of elements */
47    return 1; /* successfully */
48 }
```

index	0	1	2	3	4	5	6	7	8
	0	2	9	7	3	2	4	2	2
		9	7	3	2	4	2	2	

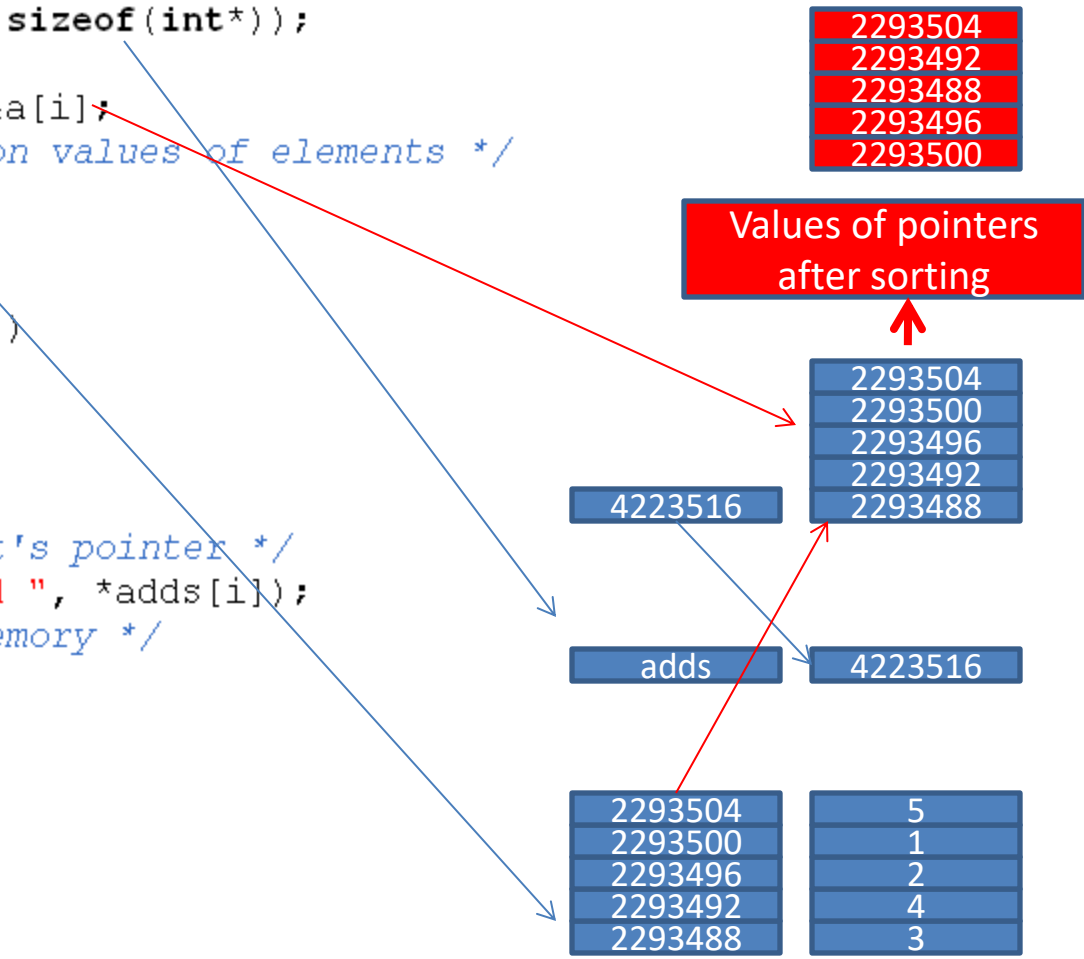
1-D Arrays: A Sample…

```
49  /*Removing all existences of a value from the array.Return 1: success, 0:fail*/
50 int removeAll(int x, int*a, int*pn)
51 {   int result =0;
52     /* Remove from the end of the array. So, no value is missed */
53     int i, j;
54     for (i=(*pn)-1; i>=0; i--)
55       if ( a[i]==x)
56         /* Shift up all elements after the position i */
57         {   result =1;
58             for (j=i; j<(*pn)-1; j++) a[j]=a[j+1];
59             (*pn)--;
60         }
61     return result;
62 }
```

0	1	2	3	4	5	6	7	8
0	2	9	7	3	2	4	2	2
0	2	9	7	3	2	4	2	
0	2	9	7	3	2	4		
0	2	9	7	3	2	4		
0	2	9	7	3	4			
0	2	9	7	3	4			
0	2	9	7	3	4			
0	2	9	7	3	4			
0	9	7	3	4				
0	9	7	3	4				

1-D Arrays: A Sample…

```
63 /* Print the array in ascending order, positions of elements are preserved */
64 void printAsc(int*a, int n)
65 {  /* Get addresses of elements */
66    int** adds =(int**)calloc(n, sizeof(int*));
67    int i,j;
68    for(i=0; i<n; i++) adds[i]= &a[i];
69    /* Asc Sort addresses based on values of elements */
70    int* t;
71    for (i=0;i<n-1; i++)
72       for (j=n-1; j>i; j--)
73         if (*adds[j]< *adds[j-1])
74         { t=adds[j];
75           adds[j]=adds[j-1];
76           adds[j-1]=t;
77         }
78    /* Print elements based on it's pointer */
79    for (i=0;i<n; i++) printf("%d ", *adds[i]);
80    free(adds); /* de-allocate memory */
81 }
```



1-D Arrays: A Sample…

```
82 /* Print the array in descending order, positions of elements are preserved */
83 void printDesc(int*a, int n)
84 {  /* Get addresses of elements */
85    int** adds = (int**) calloc (n, sizeof(int*));
86    int i, j;
87    for(i=0; i<n; i++) adds[i]= &a[i];
88    /* DEsc Sort addresses based on values of elements */
89    int* t;
90    for (i=0;i<n-1; i++)
91       for (j=n-1; j>i; j--)
92         if (*adds[j]> *adds[j-1])
93         { t=adds[j];
94           adds[j]=adds[j-1];
95           adds[j-1]=t;
96         }
97    /* Print elements based on it's pointer */
98    for (i=0;i<n; i++) printf("%d ", *adds[i]);
99    free(adds); /* de-allocate memory */
100 }
101 /* Print elements of the arrays */
102 void print(int*a, int n)
103 {  int i;
104    for (i=0;i<n;i++) printf("%d ", a[i]);
105 }
```

1-D Arrays: A Sample…

```
106 int main()
107 {   int a[MAXN]; /* array of integers */
108     int n=0;  /* Initial number of elements */
109     int value;/* added/ searched/ removed value */
110     int userChoice;
111     do
112     {  userChoice= menu();
113        switch(userChoice)
114        {  case 1:
115                if (isFull(a,n)) printf("\nSorry! The array is full.\n");
116                else
117                {  printf ("Input an added value:");
118                   scanf("%d", &value);
119                   add(value, a, &n);
120                   printf("Added\n");
121                }
122                break;
123           case 2:
124                if (isEmpty(a,n)) printf("\nSorry! The array is empty.\n");
125                else
126                {  printf ("Input the searched value:");
127                   scanf("%d", &value);
128                   int pos = search(value, a, n);
129                   if (pos<0) printf("Not found!\n");
130                   else printf("Postion is found:%d\n", pos);
```

1-D Arrays: A Sample…

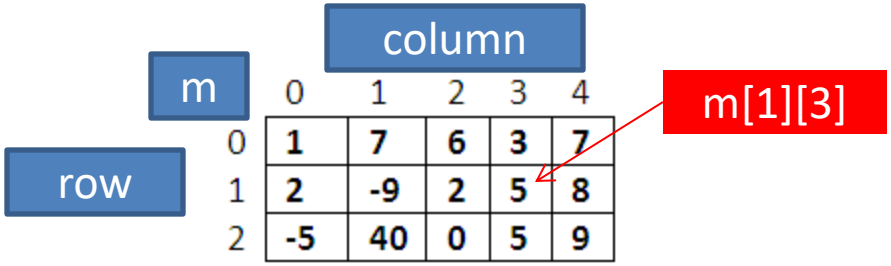
```
131             }
132             break;
133         case 3:
134             if (isEmpty(a,n)) printf("\nSorry! The array is empty.\n");
135             else
136             {  printf ("Input the removed value:");
137                scanf("%d", &value);
138                int pos = search(value, a, n);
139                if (pos<0) printf("Not found!\n");
140                else
141                {  removeOne (pos, a, &n);
142                   printf("Removed!\n");
143                }
144             }
145             break;
146         case 4:
147             if (isEmpty(a,n)) printf("\nSorry! The array is empty.\n");
148             else
149             {  printf ("Input a value that will be remove all:");
150                scanf("%d", &value);
151                if (removeAll(value, a, &n) ==0) printf("Not found!\n");
152                else printf("Removed!\n");
153             }
154             break;
```

1-D Arrays: A Sample…

```
155             case 5:
156                     print(a,n);
157                     break;
158             case 6:
159                     printAsc(a,n);
160                     break;
161             case 7:
162                     printDesc(a,n);
163                     break;
164             default: printf("\nGoodbye.\n");
165         }
166     }
167     while (userChoice>0 && userChoice<8);
168     getchar();
169     return 0;
170 }
```

4- Two-Dimensional Arrays

- A group of elements which belong the same data type and they are divided into some rows and some column (it is called as matrix also).
- Each element is identified by two indexes (index of row, index of column).



Traversing a matrix:

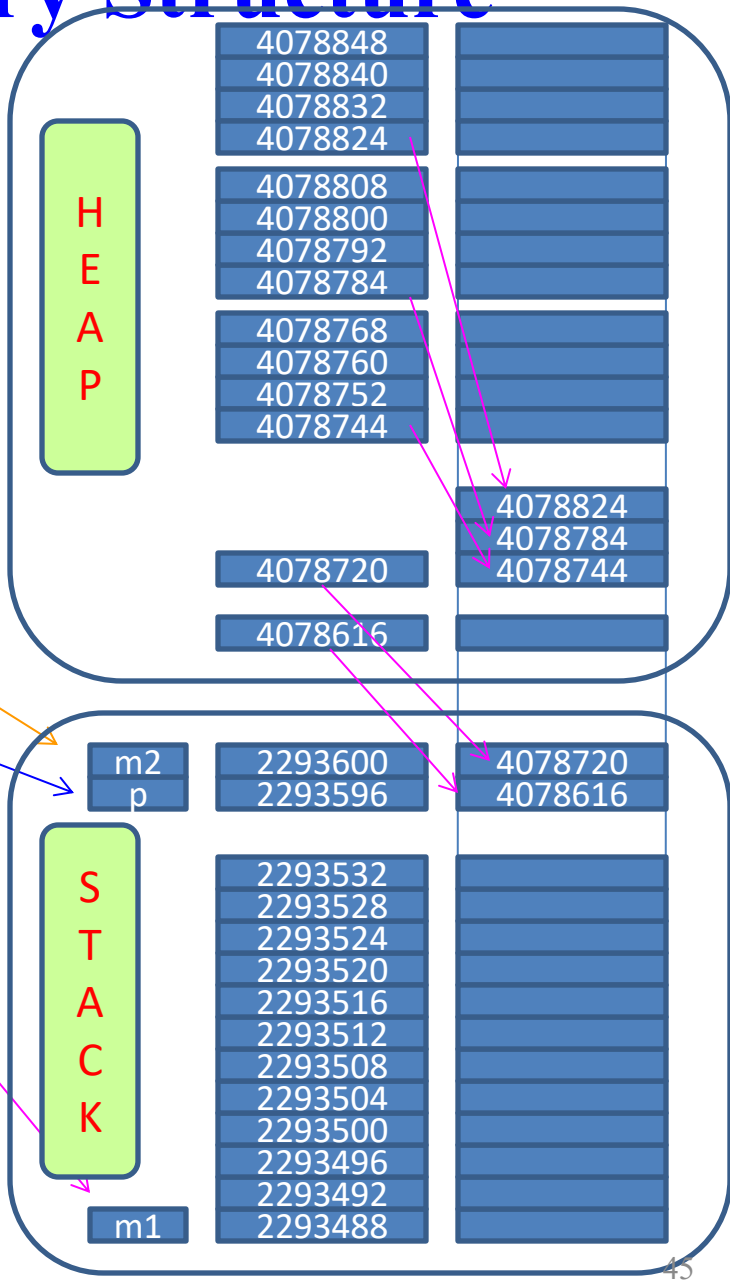
```
for ( i =0; i<row; i++)
{   for ( j=0; j< column; j++)
        [if (condition)] Access m[i][j];
}
```

The next slide will demonstrate how static and dynamic 2-D arrays are stored.

3- 2-D Arrays: Memory Structure

```
2  #include <stdio.h>
3  int main()
4  { int r=3, c=4, m1[r][c], i, j;
5    double** m2;
6    m2= (double**) calloc( r, sizeof(double*));
7    m2[0]= (double*) calloc(4, sizeof(double));
8    m2[1]= (double*) calloc(4, sizeof(double));
9    int*p= (int*)malloc(sizeof(int));
10   m2[2]= (double*) calloc(4, sizeof(double));
11   printf("\nMemory of m1:\n");
12   printf("m1: addr:=%u, value:%u\n", &m1, m1);
13   for (i=0;i<r;i++)
14    { for (j=0;j<c;j++) printf("%8u", &m1[i][j]);
15      printf("\n");
16    }
17   printf("\np: addr:=%u, value:%u\n", &p, p);
18   printf("\nMemory of m2:\n");
19   printf("m2: addr:=%u, value:%u\n", &m2, m2);
20   for (i=0;i<r;i++)
21     { for (j=0;j<c;j++) printf("%8u", &m2[i][j]);
22       printf("\n");
23     }
24   getchar();
25   return 0;
26 }
```

```
Memory of m1:
m1: addr:=2293488, value:2293488
 2293488 2293492 2293496 2293500
 2293504 2293508 2293512 2293516
 2293520 2293524 2293528 2293532

p: addr:=2293596, value:4078616

Memory of m2:
m2: addr:=2293600, value:4078720
 4078744 4078752 4078760 4078768
 4078784 4078792 4078800 4078808
 4078824 4078832 4078840 4078848
```



Static 2-D Arrays Demo.

Accept a matrix maximum 20x20.
Print out maximum value in it, print out the matrix.

```
1  /* Static Matric Demo.*/
2  #include <stdio.h>
3  #define MAXR 20
4  #define MAXC 20
5  /* Input a mtrix of ints, num of rows and column are known */
6  void input(int m[][MAXC], int r, int c);
7  int max (int m[][MAXC], int r, int c);
8  void print (int m[][MAXC], int r, int c);
9  int main()
10 {  int m[MAXR][MAXC]; /* Declare a static matrix*/
11    int r, c; /* real used number of rows and columns */
12    int maxVal;
13    do
14    {  printf("Enter number of rows and columns of the matrix:");
15       scanf("%d%d", &r, &c);
16    }
17    while (r<1 || r >MAXR || c<1 || c > MAXC);
18    printf("Enter a matrix %d x %d\n", r, c);
19    input(m, r, c);
20    maxVal  = max (m, r, c);
21    printf("Max value:%d\n", maxVal);
22    printf("\nInputted matrix:\n");
23    print(m, r, c);
24    while (getchar()!='\n');getchar();
25    return 0;
26 }
```

Keep in your mind the way to specify a matrix as a parameter of a function (the number of column must be pre-defined.).

Static 2-D Arrays Demo.

```
K:\GiangDay\FU\OOP\BaiTap\matrix1.exe
Enter number of rows and columns of the matrix:3 4
Enter a matrix 3 x 4
Value at [0][0]:1
Value at [0][1]:2
Value at [0][2]:3
Value at [0][3]:4
Value at [1][0]:5
Value at [1][1]:6
Value at [1][2]:7
Value at [1][3]:8
Value at [2][0]:9
Value at [2][1]:0
Value at [2][2]:1
Value at [2][3]:2
Max value:9

Inputted matrix:
      1      2      3      4
      5      6      7      8
      9      0      1      2
```

```
36 int max(int m[][MAXC], int r, int c)
37 {   int result = m[0][0];
38     int i, j;
39     for (i=0;i<r; i++)
40         for (j=0; j<c; j++)
41           if (result < m[i][j]) result=m[i][j];
42     return result;
43 }
44 void print (int m[][MAXC], int r, int c)
45 {   int i, j;
46     for (i=0;i<r; i++)
47     { for (j=0; j<c; j++) printf("%7d", m[i][j]);
48       printf("\n");
49     }
50 }
```

```
27 void input(int m[][MAXC], int r, int c)
28 { int i, j;
29   for (i=0;i<r; i++) /* Enter values to each row */
30   {   for (j=0; j<c; j++) /* Enter value to each column */
31       { printf("Value at [%d][%d]:", i, j);
32         scanf("%d", &m[i][j]);
33       }
34   }
35 }
```

Summary

- Array is the simplest data structure for a group of elements which belong to the same data type.
- Each element in an array is identified by one or more index beginning from 0.
- Number of dimensions: Number of indexes are used to identify an element.
- Static arrays → Stack segment

```
Type a[MAXN];
Type m[MAXROW][MAXCOL];
```

- Dynamic array: Use pointer and allocate memory using functions

```
double *a = (double*)calloc(n, sizeof(double));
int** m = (int**) calloc(row, sizeof(int*));
for (i=0; i<row; i++) m[i]= (int*)calloc(col, sizeof(int));
```

Summary

- Accessing elements in an array:

1-D Array (a)		2-D Array (m)	
Address	***Value***	***Address***	***Value***
&a[index]	a[index]	&m[i][j]	m[i][j]
a+index	*(a+index)		
Compiler determines the address of an element:			
a + index*sizeof(DataType)		m + (i*NumCol + j)*sizeof(DataType)	

- **Common operations on arrays:**
  - Add an element
  - Search an element
  - Remove an element
  - Input
  - Output
  - Sort

- Base of algorithms on arrays: Traversing

Exercise- Do yourself

- Develop a C-program that helps user managing an 1-D array of real numbers(maximum of 100 elements) using the following simple menu:
- 1- Add a value
- 2- Search a value
- 3- Print out the array
- 4- Print out values in a range (minVal<=value<=maxVal, minVal and maxVal are inputted)
- 5- Print out the array in ascending order (positions of elements are preserved)
- Others- Quit

Thank You