TRƯỜNG ĐẠI HỌC FPT

Contiguous Storage

Objectives

How do you manage group data efficiently?

- Store
- Input
- Output
- Search
- Sort

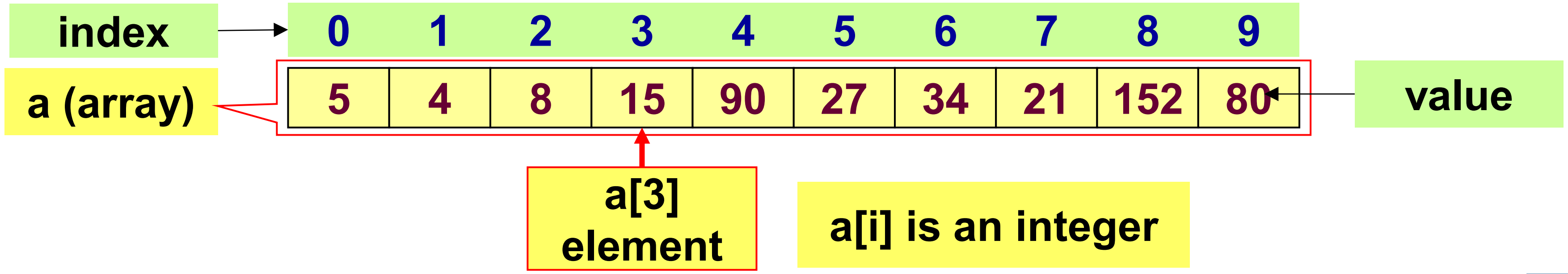
Contents

1- Contiguous Storage

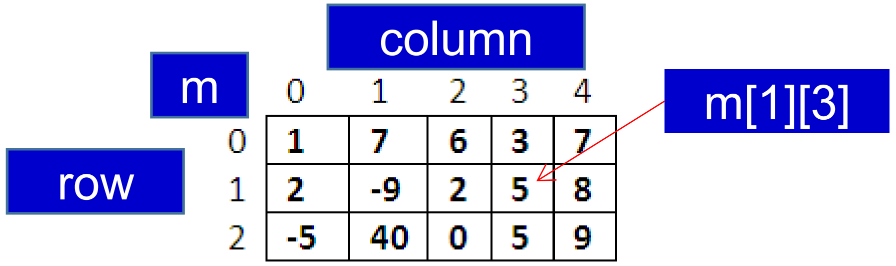
- Commonly, a set of the same meaning elements are considered.
- They are stored in a contiguous block of memory.
- Ex: Group of 10 int numbers → 40 bytes block is needed.
- Data are considered can be a group of some items which belong to some different data types → Contiguous memory block is partitioned 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 - Array

- An **array** is a data structure consisting of an ordered set of elements of common type that are stored contiguously in memory. Each element is identified by it's position (index).



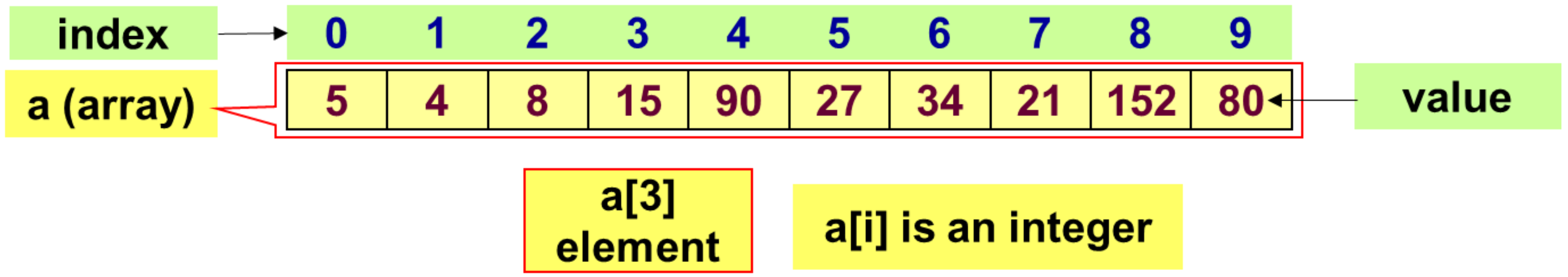
- **Dimension**: Direction that is used to perform an action on array.
- **Number of dimensions**: Number of indexes are used to specify an element.
- **Common arrays**: 1-D and 2-D arrays.
- **Name of an array**: An array has it's name.



	0	1	2	3	4
0	1	7	6	3	7
1	2	-9	2	5	8
2	-5	40	0	5	9

3 - One-dimensional Arrays (1-D)

- **1-D array**: a collection of items (elements, terms) which belong to the same data type and are stored contiguously in memory.
  - Each element has a unique index and holds a single value. Index numbering starts at 0 and extends to one less than the number of elements in the array.
  - To refer to a specific element, we write the array name followed by bracket notation around the element's index. Example: identifier[index]
- 1-D array structure:



1-D Array: 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.
- **Syntax**:

  DataType ArrayName[NumberOfElements];

- Example:

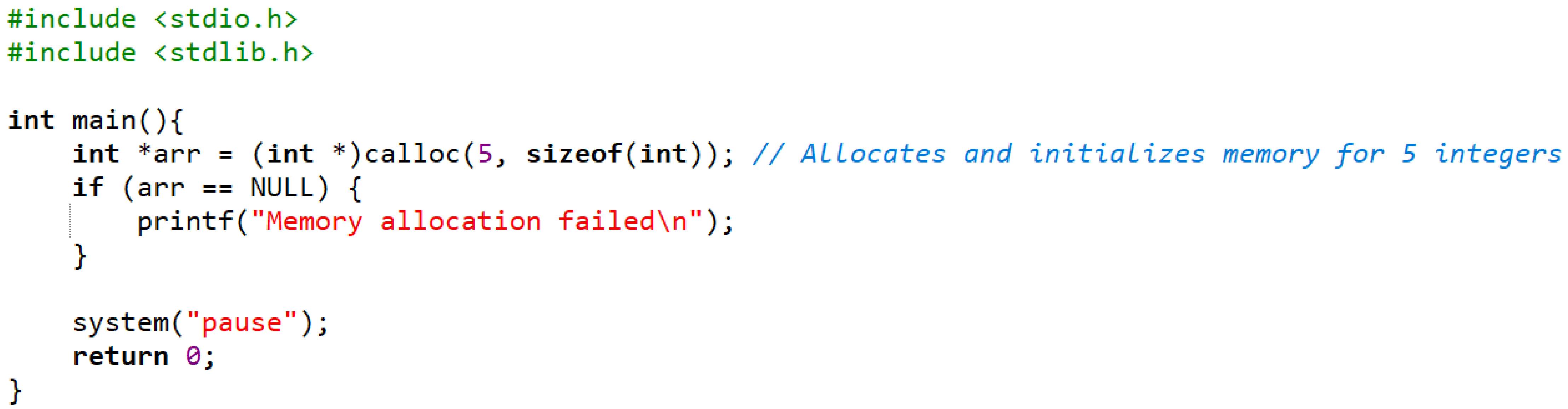
  int a1[5]; char s[12]; double a2[100];

- How compilers can determine the memory size of an array?

  => NumberOfElements * sizeof(dataType) → int a1[5] → 5 *sizeof(int) = 5*4 = 20 bytes

1-D Array: Declaration (cont.)

- 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)
- Example:

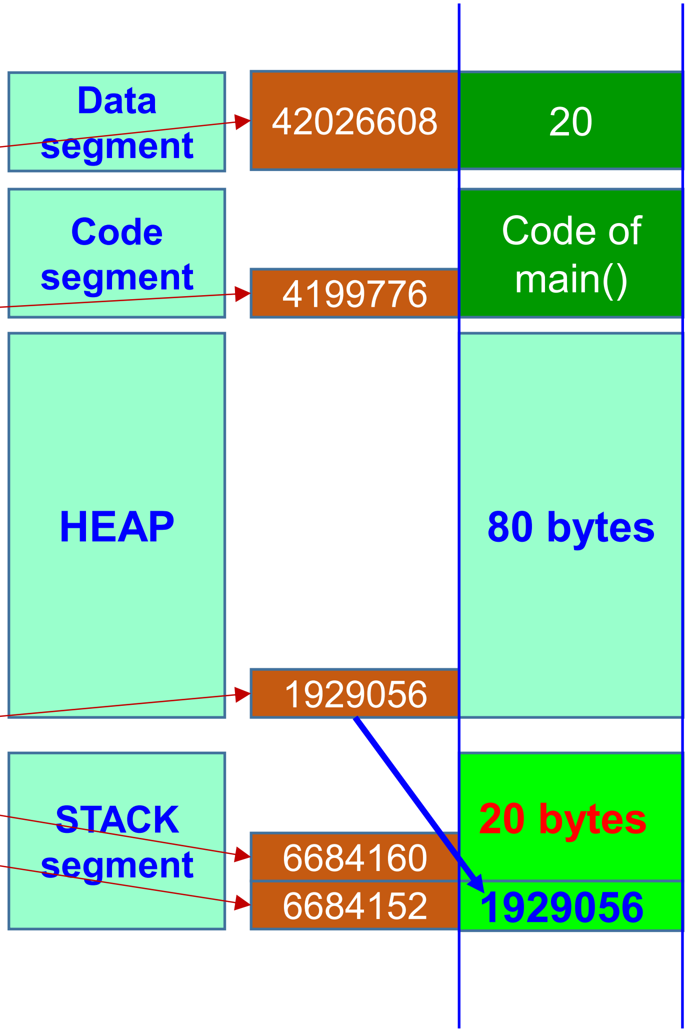


```c
#include <stdio.h>
#include <stdlib.h>

int main(){
    int *arr = (int *)calloc(5, sizeof(int)); // Allocates and initializes memory for 5 integers
    if (arr == NULL) {
        printf("Memory allocation failed\n");
    }

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

1-D Array: Example Memory Allocation

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

int MAX = 20;

int main()
{
    printf("MAX address = %u\n", &MAX);
    printf("main() address = %u\n", &main);

    int a1[5];          /* Static array of 5 integer numbers*/
    double *a2 = NULL;  /* Dynamic array of double numbers   */

    /* Allowcate a memory block for 10 double numbers */
    a2 = (double*)calloc(10, sizeof(double));

    printf("a1 address = %u\n", &a1);
    printf("a2 address = %u\nvalue of a2 = %u\n", &a2, a2);

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

```
MAX address = 4206608
main() address = 4199776
a1 address = 6684160
a2 address = 6684152
value of a2 = 1929056
```



1-D Arrays: Initialization & Accessing Elements

- **Initialize an array:**

```
DataType a[] = {value1, value2, … };
```

- **How to access the ith element of the array a?**

  a is the address of the first element. Based on operation on pointers:

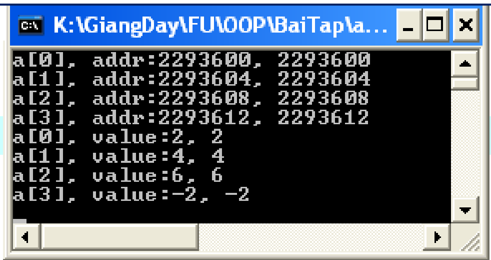
  → ***a+i*** : address of the ith element, another way: ***&a[i]***

  → ***(a+i)**: value of the ith element, another way: ***a[i]***

1-D Arrays: Initialization & Accessing Elements (cont.)

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

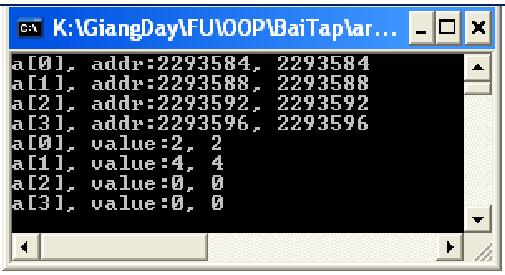
```
#include <stdio.h>n
#include <stdlib.h>
int main()
{  int a[]= {2,4,6,-2};
   int i;
   for (i=0;i<4;i++)
     printf("a[%d], addr:%u, %u\n", i, a+i, &a[i]);
   for (i=0;i<4;i++)
     printf("a[%d], value:%d, %d\n", i, *(a+i), a[i]);
   getchar();
   return 0;
}
```



```
a[0], addr:2293600, 2293600
a[1], addr:2293604, 2293604
a[2], addr:2293608, 2293608
a[3], addr:2293612, 2293612
a[0], value:2, 2
a[1], value:4, 4
a[2], value:6, 6
a[3], value:-2, -2
```

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;
}
```



```
a[0], addr:2293584, 2293584
a[1], addr:2293588, 2293588
a[2], addr:2293592, 2293592
a[3], addr:2293596, 2293596
a[0], value:2, 2
a[1], value:4, 4
a[2], value:0, 0
a[3], value:0, 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.

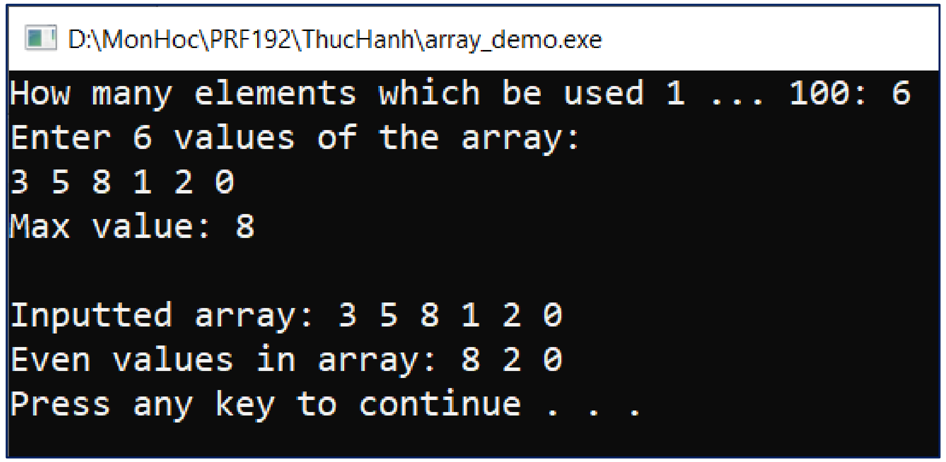
- Example 1:
  - Input an array of n integers → **void input (int* a, int n)**
- Example 2:
  - Input elements of an array of integers which it's number of element is stored at the pointer **pn** → **void input (int a[], int*pn)**
- Example 3:
  - Calculate the sum of an array of n integers → **int sum (int *a, int n)**
- Example 4:
  - Output an array of n double numbers → **void output (double a[], int n)**

1-D Array is a Function Parameter: Demo

- **Demo 1: 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.
- *Hint*:
  - **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

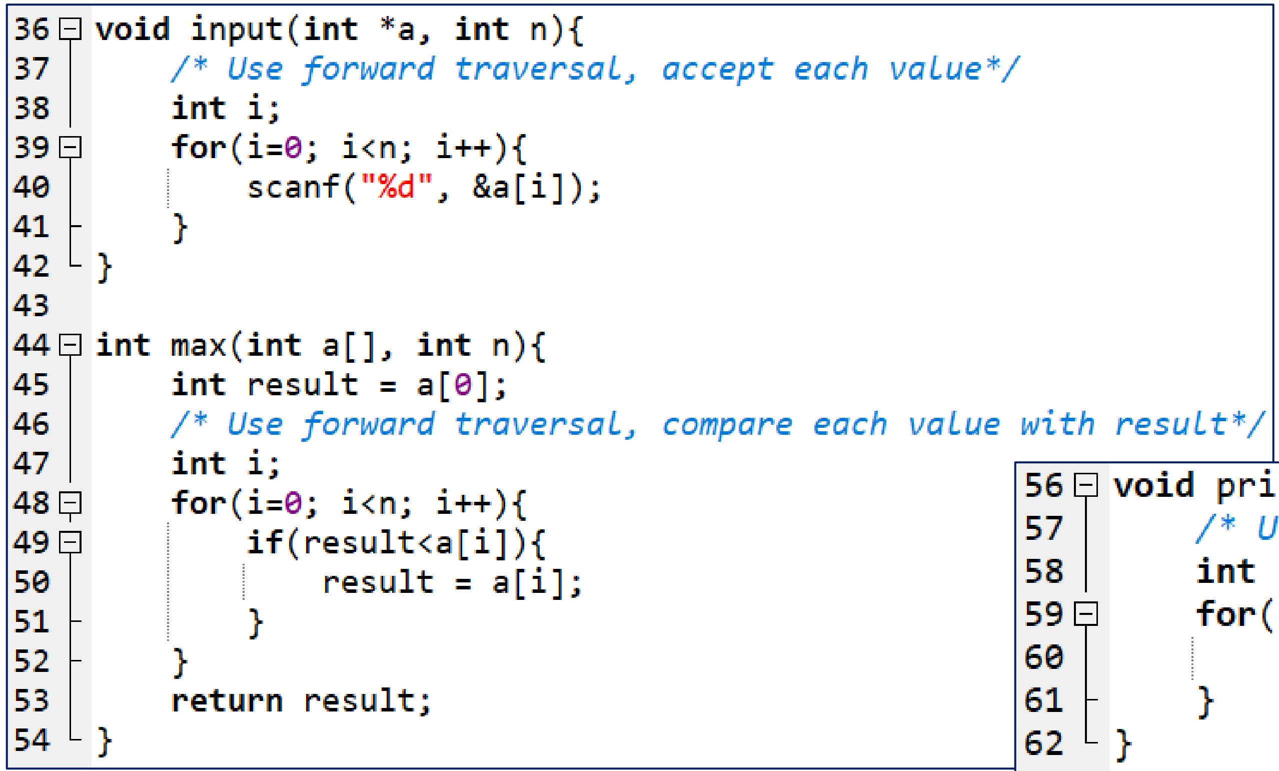
Array Funtion Parameter: Demo 1 (cont.)

```c
#include <stdio.h>
#include <stdlib.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 be used 1 ... %d: ", MAXN);
        scanf("%d", &n);
    }while(n<1 || n>MAXN);
    printf("Enter %d values of the array:\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);
    printf("\n");
    while(getchar()!='\n'); // Clear buffer

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

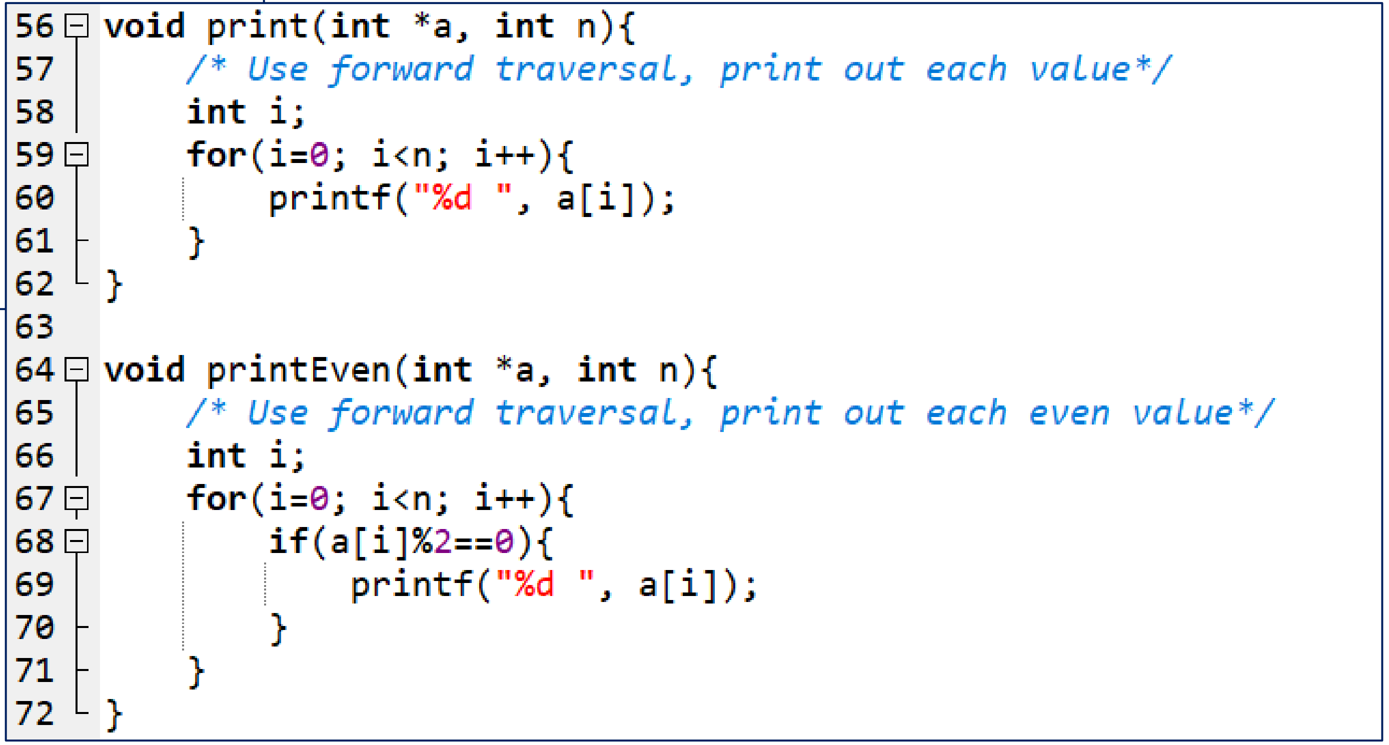

D:\MonHoc\PRF192\ThucHanh\array_demo.exe

```
How many elements which be used 1 ... 100: 6
Enter 6 values of the array:
3 5 8 1 2 0
Max value: 8

Inputted array: 3 5 8 1 2 0
Even values in array: 8 2 0
Press any key to continue . . .
```

Array Funtion Parameter: Demo 1 (cont.)



```c
void input(int *a, int n){
    /* Use forward traversal, accept each value*/
    int i;
    for(i=0; i<n; i++){
        scanf("%d", &a[i]);
    }
}

int max(int a[], int n){
    int result = a[0];
    /* Use forward traversal, compare each value with result*/
    int i;
    for(i=0; i<n; i++){
        if(result<a[i]){
            result = a[i];
        }
    }
    return result;
}
```

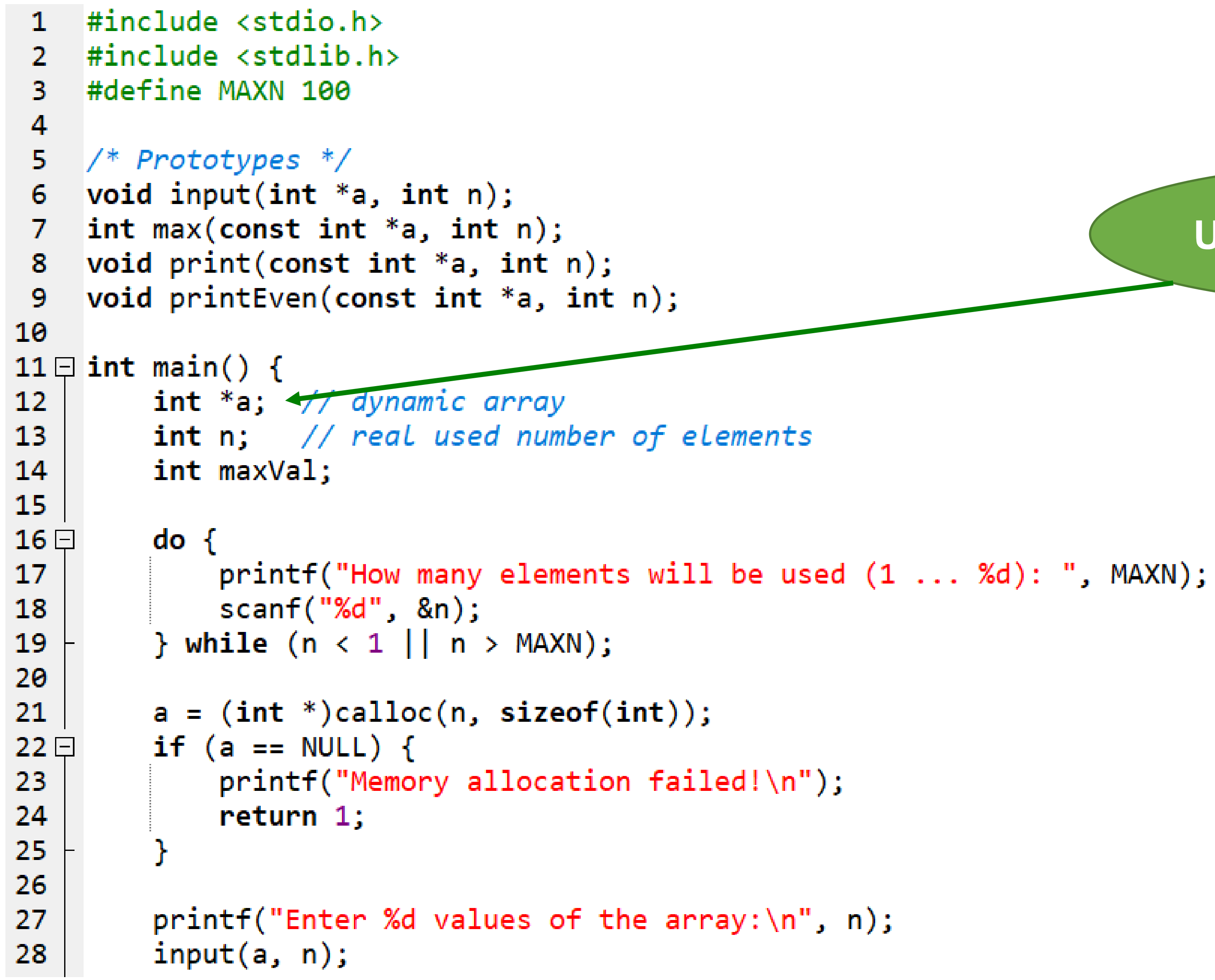


```c
void print(int *a, int n){
    /* Use forward traversal, print out each value*/
    int i;
    for(i=0; i<n; i++){
        printf("%d ", a[i]);
    }
}

void printEven(int *a, int n){
    /* Use forward traversal, print out each even value*/
    int i;
    for(i=0; i<n; i++){
        if(a[i]%2==0){
            printf("%d ", a[i]);
        }
    }
}
```

Array Function Parameter: Demo 1

- **Problems:**
  - If you allocate an array having 100 elements but 6 elements are used then memory is wasted.
  - 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
  - Can expand the size of the original array

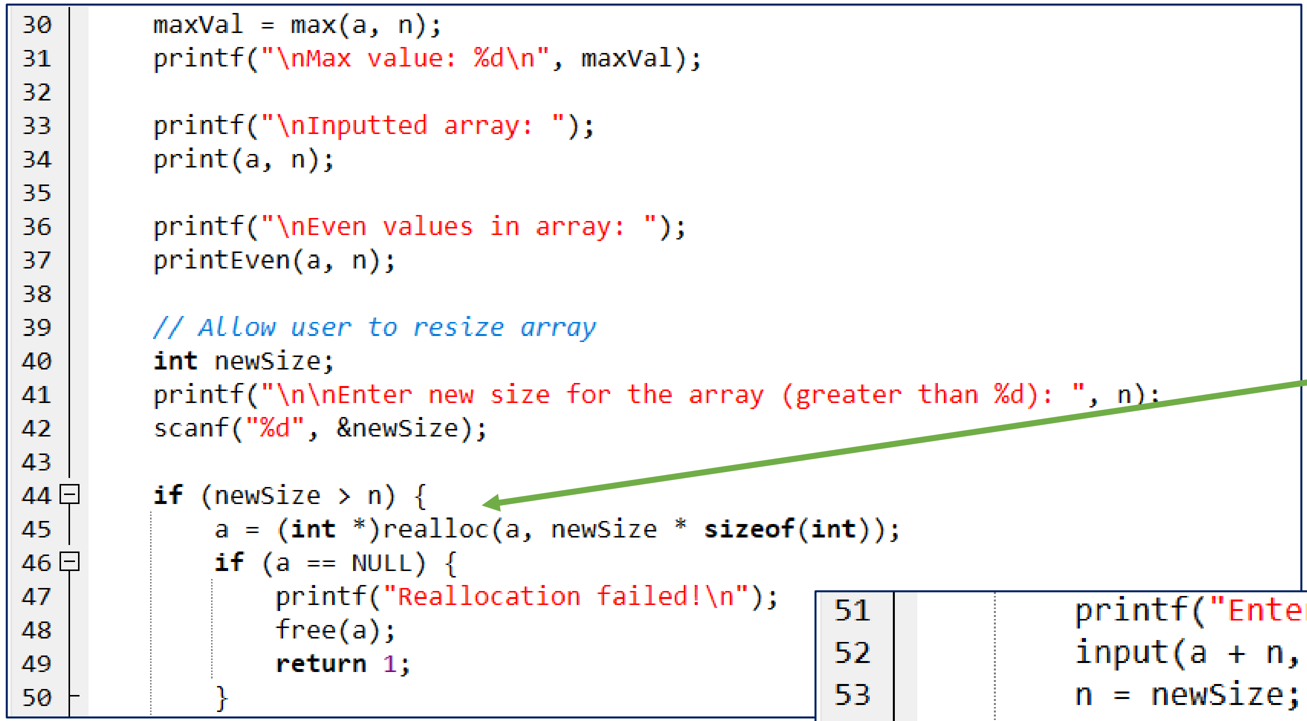
Array Funtion Parameter: Solution



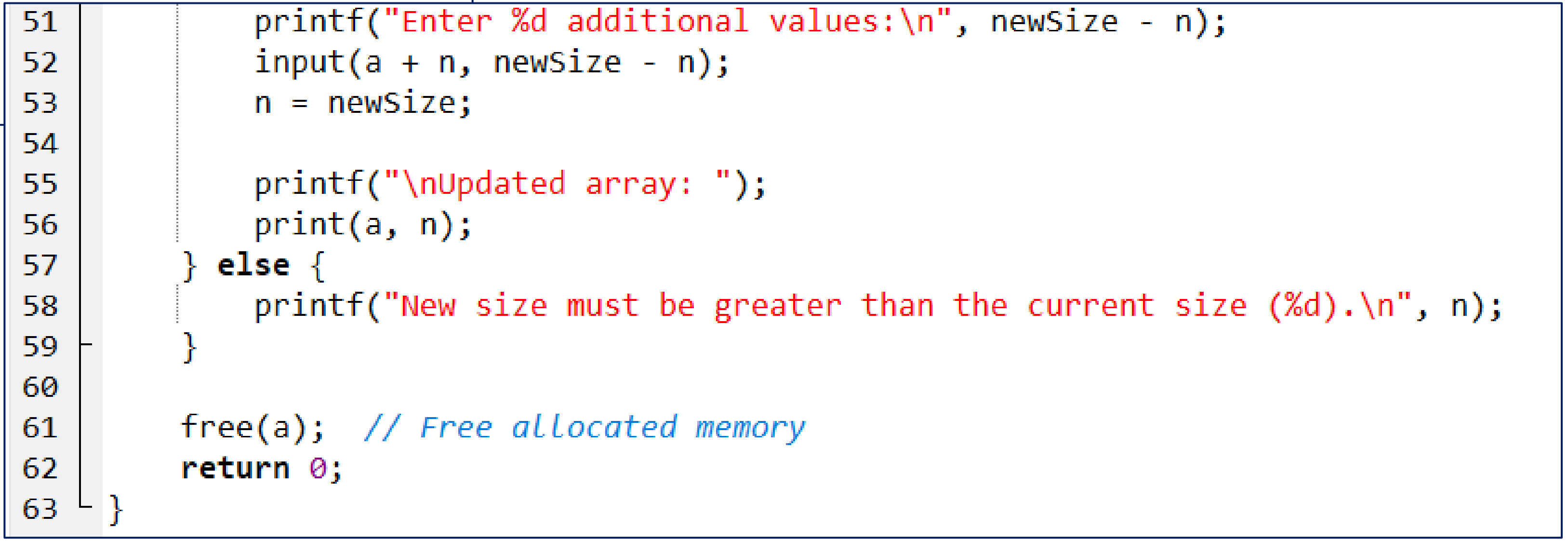
```c
#include <stdio.h>
#include <stdlib.h>
#define MAXN 100

/* Prototypes */
void input(int *a, int n);
int max(const int *a, int n);
void print(const int *a, int n);
void printEven(const int *a, int n);

int main() {
    int *a;  // dynamic array
    int n;   // real used number of elements
    int maxVal;

    do {
        printf("How many elements will be used (1 ... %d): ", MAXN);
        scanf("%d", &n);
    } while (n < 1 || n > MAXN);

    a = (int *)calloc(n, sizeof(int));
    if (a == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }

    printf("Enter %d values of the array:\n", n);
    input(a, n);
```

Use Dynamic Array

Solution (cont.)



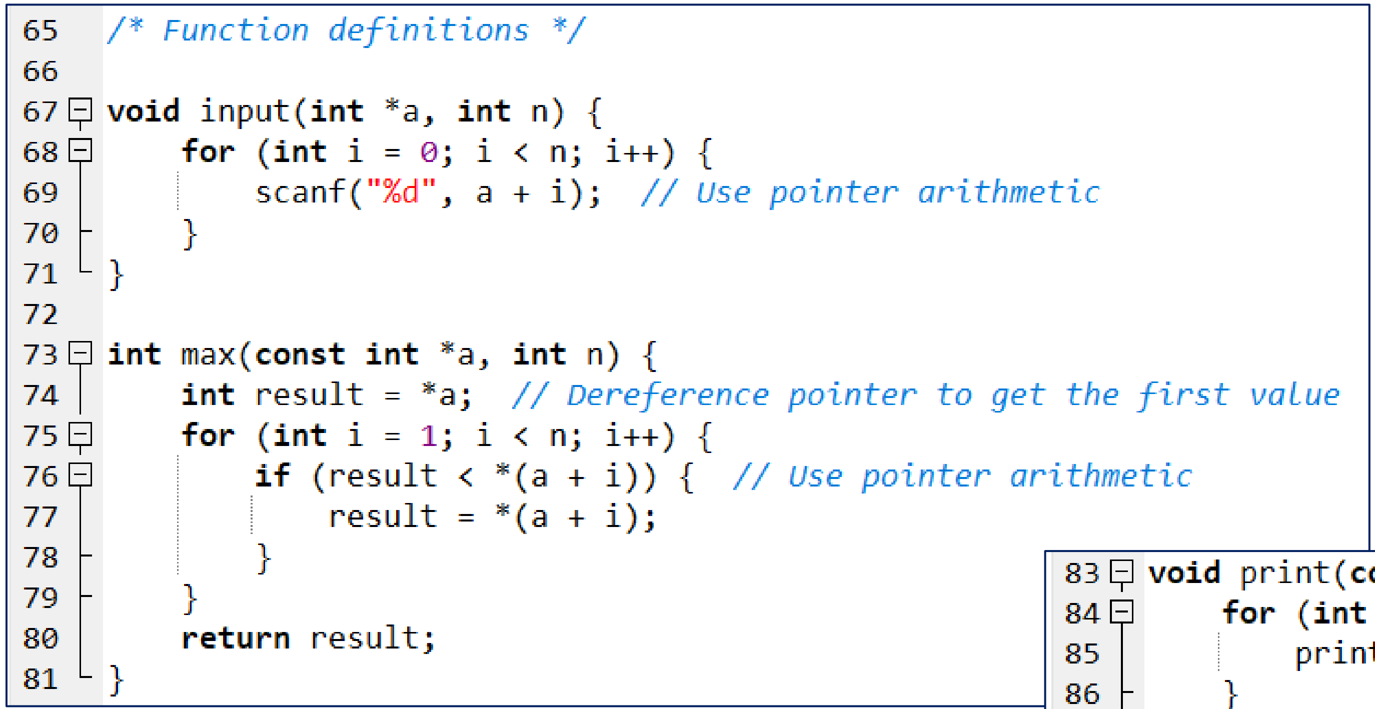
```c
    maxVal = max(a, n);
    printf("\nMax value: %d\n", maxVal);

    printf("\nInputted array: ");
    print(a, n);

    printf("\nEven values in array: ");
    printEven(a, n);

    // Allow user to resize array
    int newSize;
    printf("\n\nEnter new size for the array (greater than %d): ", n);
    scanf("%d", &newSize);

    if (newSize > n) {
        a = (int *)realloc(a, newSize * sizeof(int));
        if (a == NULL) {
            printf("Reallocation failed!\n");
            free(a);
            return 1;
        }
```



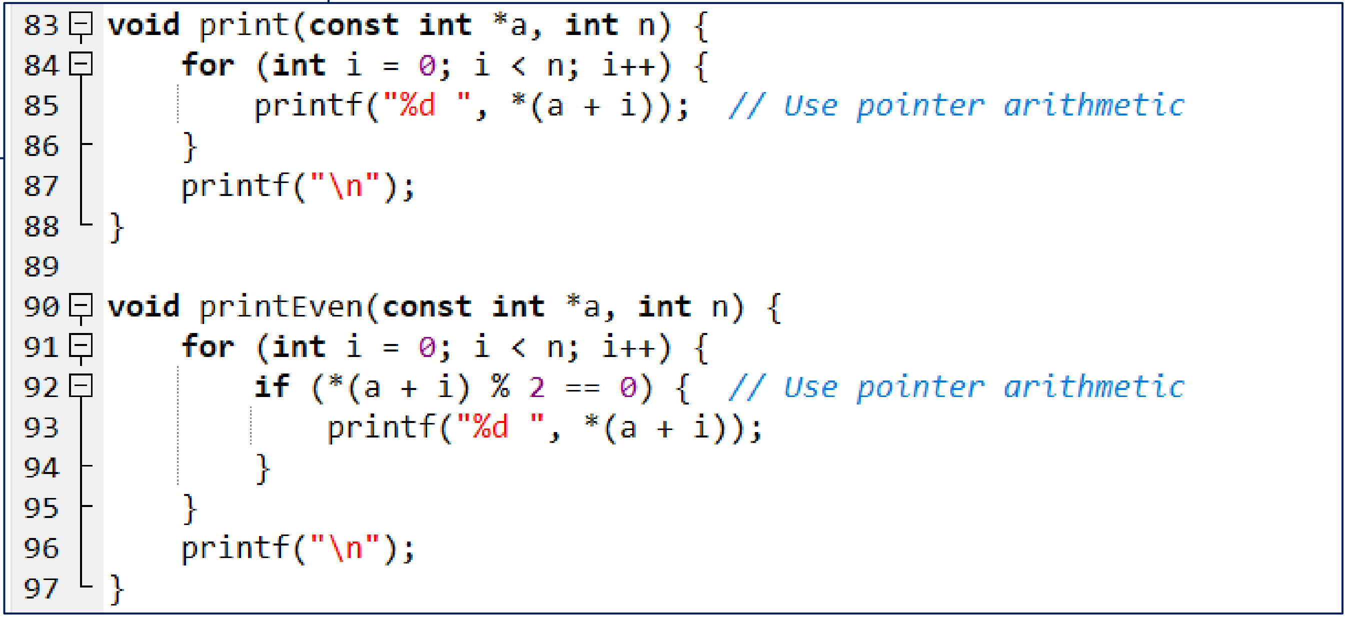
```c
        printf("Enter %d additional values:\n", newSize - n);
        input(a + n, newSize - n);
        n = newSize;

        printf("\nUpdated array: ");
        print(a, n);
    } else {
        printf("New size must be greater than the current size (%d).\n", n);
    }

    free(a);  // Free allocated memory
    return 0;
}
```

Expand the size of the original array

Solution (cont.)

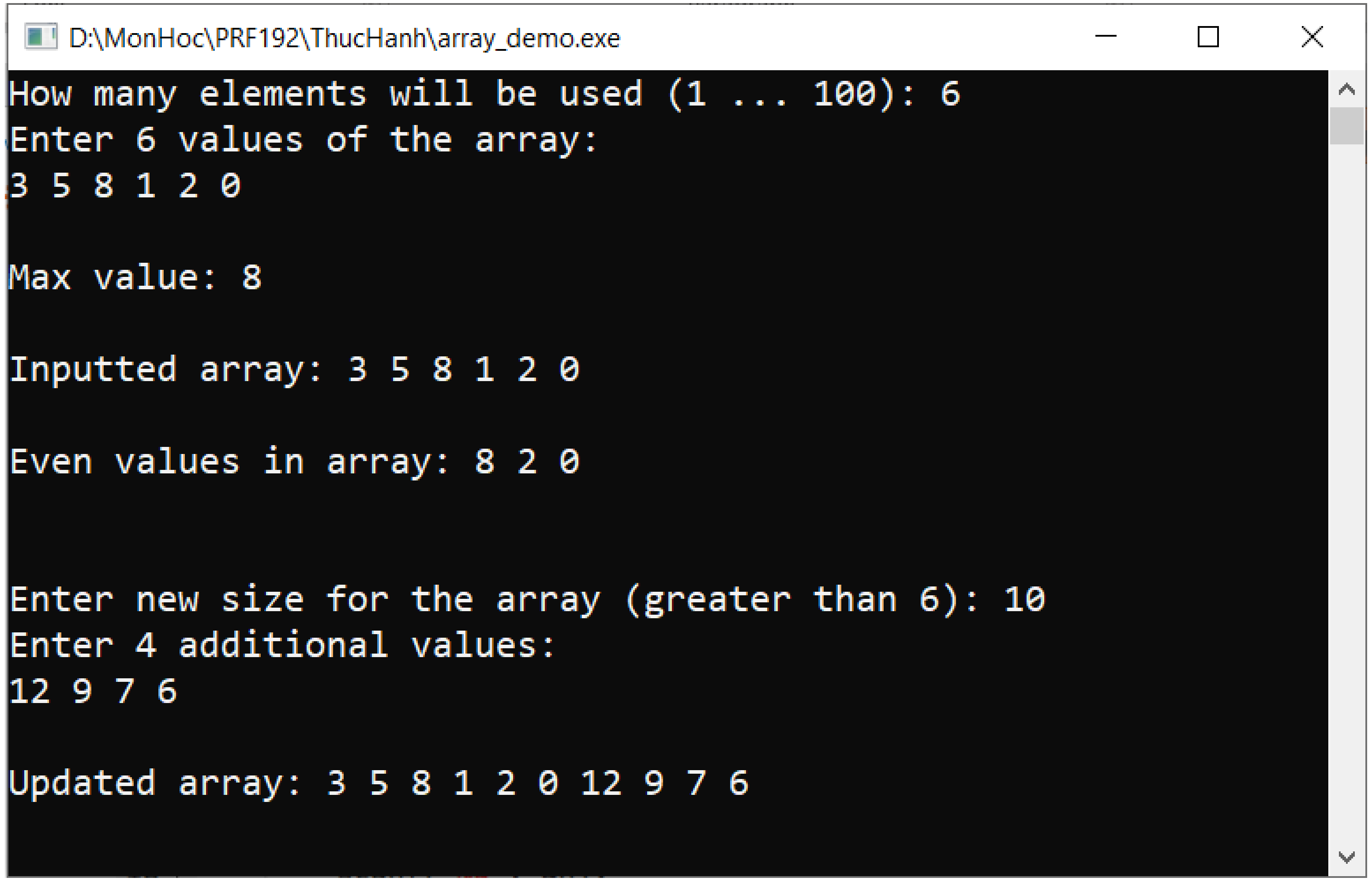


```c
/* Function definitions */

void input(int *a, int n) {
    for (int i = 0; i < n; i++) {
        scanf("%d", a + i);  // Use pointer arithmetic
    }
}

int max(const int *a, int n) {
    int result = *a;  // Dereference pointer to get the first value
    for (int i = 1; i < n; i++) {
        if (result < *(a + i)) {  // Use pointer arithmetic
            result = *(a + i);
        }
    }
    return result;
}
```



```c
void print(const int *a, int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", *(a + i));  // Use pointer arithmetic
    }
    printf("\n");
}

void printEven(const int *a, int n) {
    for (int i = 0; i < n; i++) {
        if (*(a + i) % 2 == 0) {  // Use pointer arithmetic
            printf("%d ", *(a + i));
        }
    }
    printf("\n");
}
```

Output Solution


D:\MonHoc\PRF192\ThucHanh\array_demo.exe

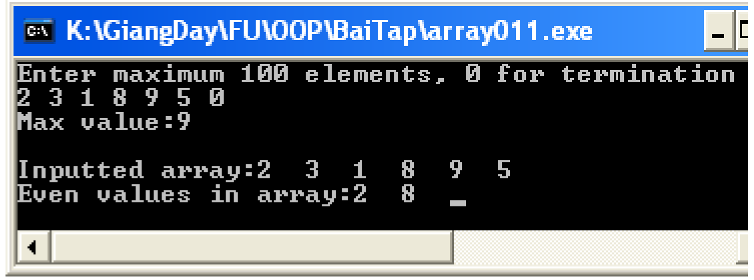
```
How many elements will be used (1 ... 100): 6
Enter 6 values of the array:
3 5 8 1 2 0

Max value: 8

Inputted array: 3 5 8 1 2 0

Even values in array: 8 2 0


Enter new size for the array (greater than 6): 10
Enter 4 additional values:
12 9 7 6

Updated array: 3 5 8 1 2 0 12 9 7 6
```

Exercise 1:

- 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.
- *Requirement*:
  - 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.

Exercise 1 (cont.)

```c
#include <stdio.h>
#define MAXN 100
/* Input an array, number of elements is stored at pn
   User will terminate inputting when 0 is entered.*/
void input(int*a, int *pn);
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;
    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;
}
```

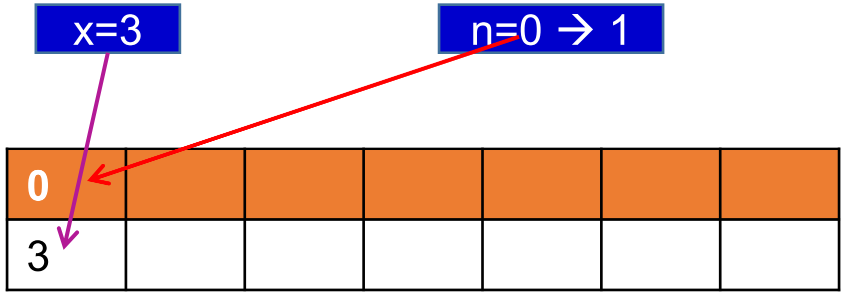

K:\GiangDay\FU\OOP\BaiTap\array011.exe

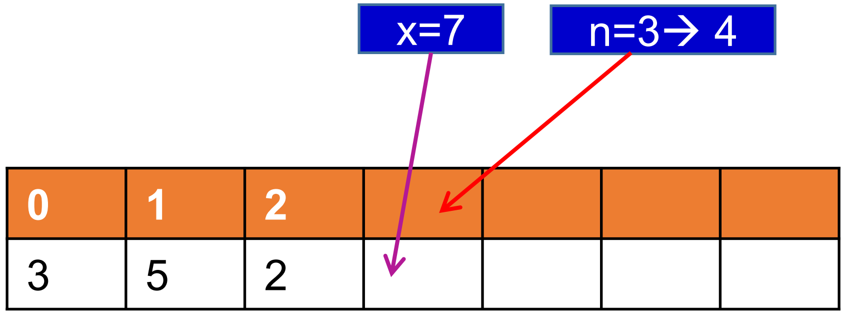
```
Enter maximum 100 elements, 0 for termination
2 3 1 8 9 5 0
Max value:9

Inputted array:2  3  1  8  9  5
Even values in array:2  8  _
```

Exercise 1 (cont.)

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

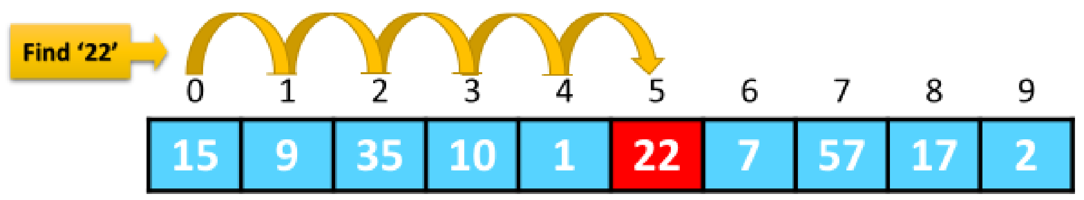
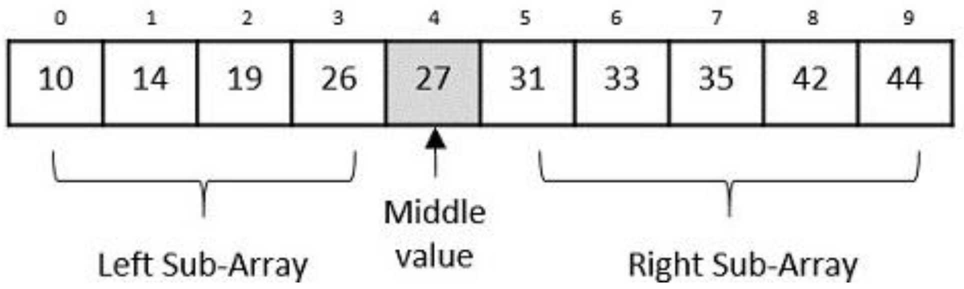
```c
int max(int a[], int n)
{
     /* Do yourself */
}
void print (int* a, int n)
{  /* Do yourself */
}
void printEven (int* a, int n)
{  /* Do yourself */
}
```





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
 Find '22' 0 1 2 3 4 5 6 7 8 9 15 9 35 10 1 22 7 57 17 2	 0 1 2 3 4 5 6 7 8 9 10 14 19 26 27 31 33 35 42 44 Left Sub-Array Middle value Right Sub-Array

Searching: Linear Search

- **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**

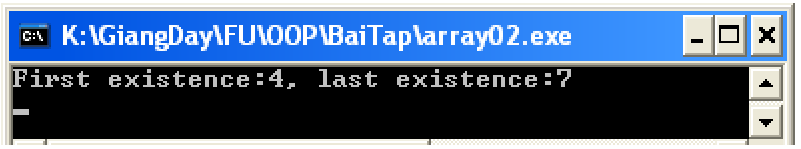
```
int  firstLinearSearch(int x, int a[], int n)
{
            int i;
            for ( i=0;  i<n;  i++)
                        if ( x == a[i] ) return i;
            return -1;
}
```

```
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;
}
```

Exercise 2: Using Linear Search algorithm

```
#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;
}
```

Do yourself

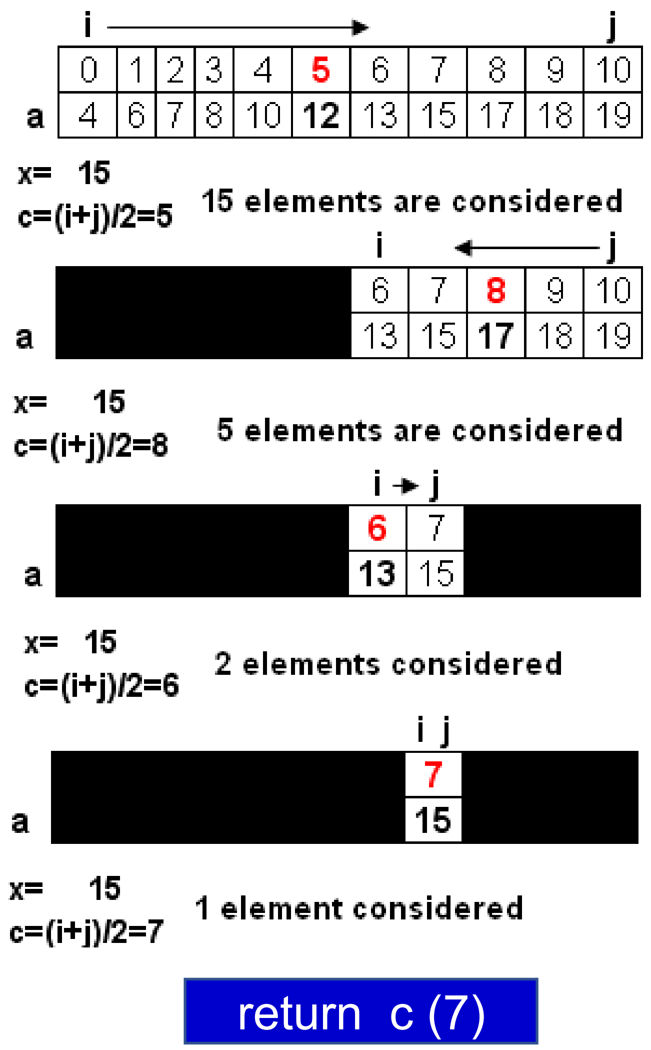

K:\GiangDay\FU\OOP\BaiTap\array02.exe

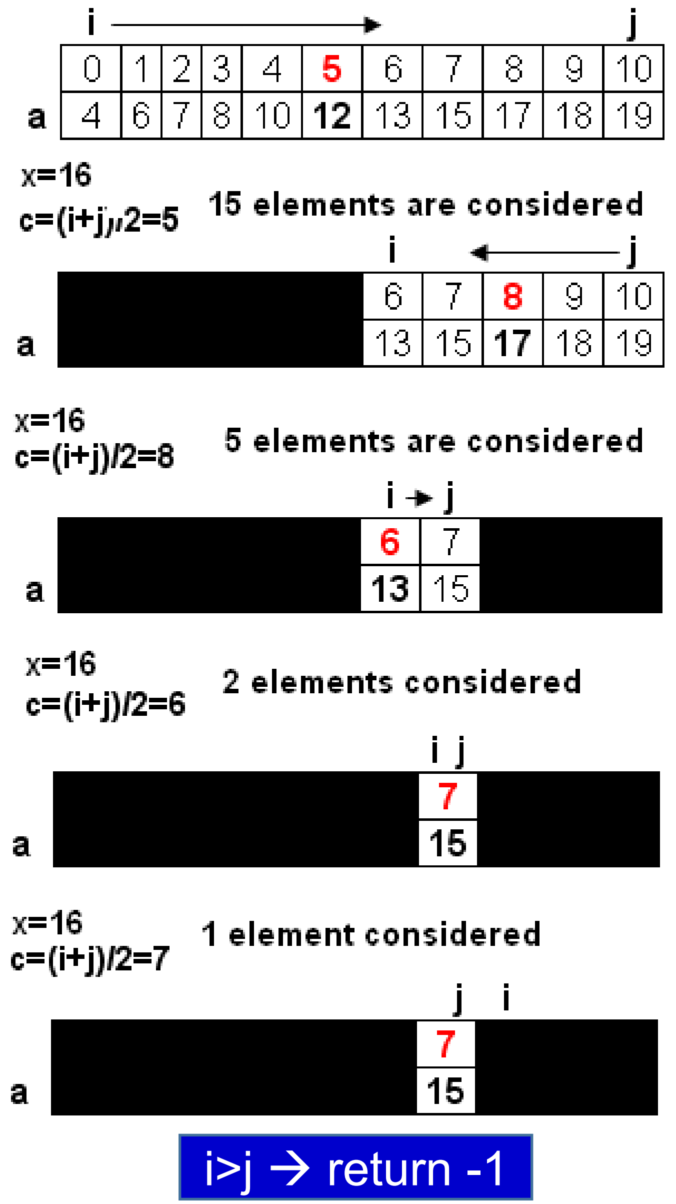
```
First existence:4, last existence:7
_
```

Searching: Binary Search

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

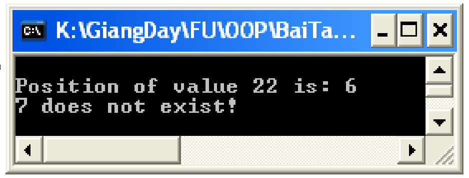
```
int  binarySearch ( int x, int a[], int n)
{
   int i=0, j= n-1, c ;
   while (i<=j)
     {
         c= (i+j)/2;
         if ( x== a[c] )  return c ;
         if (x < a[c] )  j = c-1;
         else i = c +1;
     }
    return -1;
}
```





Exercise 3: Using Binary Search algorithm

```c
#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;
}
```


K:\GiangDay\FU\OOP\BaiTa...

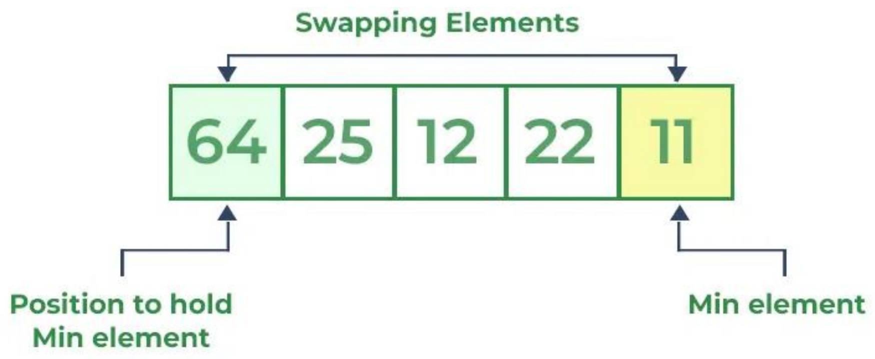
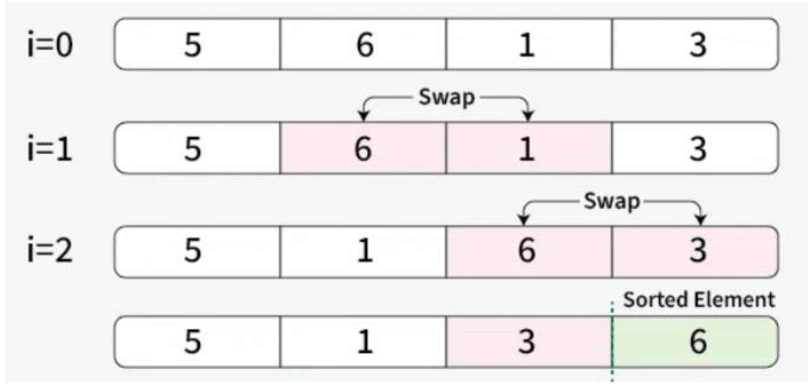
```
Position of value 22 is: 6
7 does not exist!
```

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

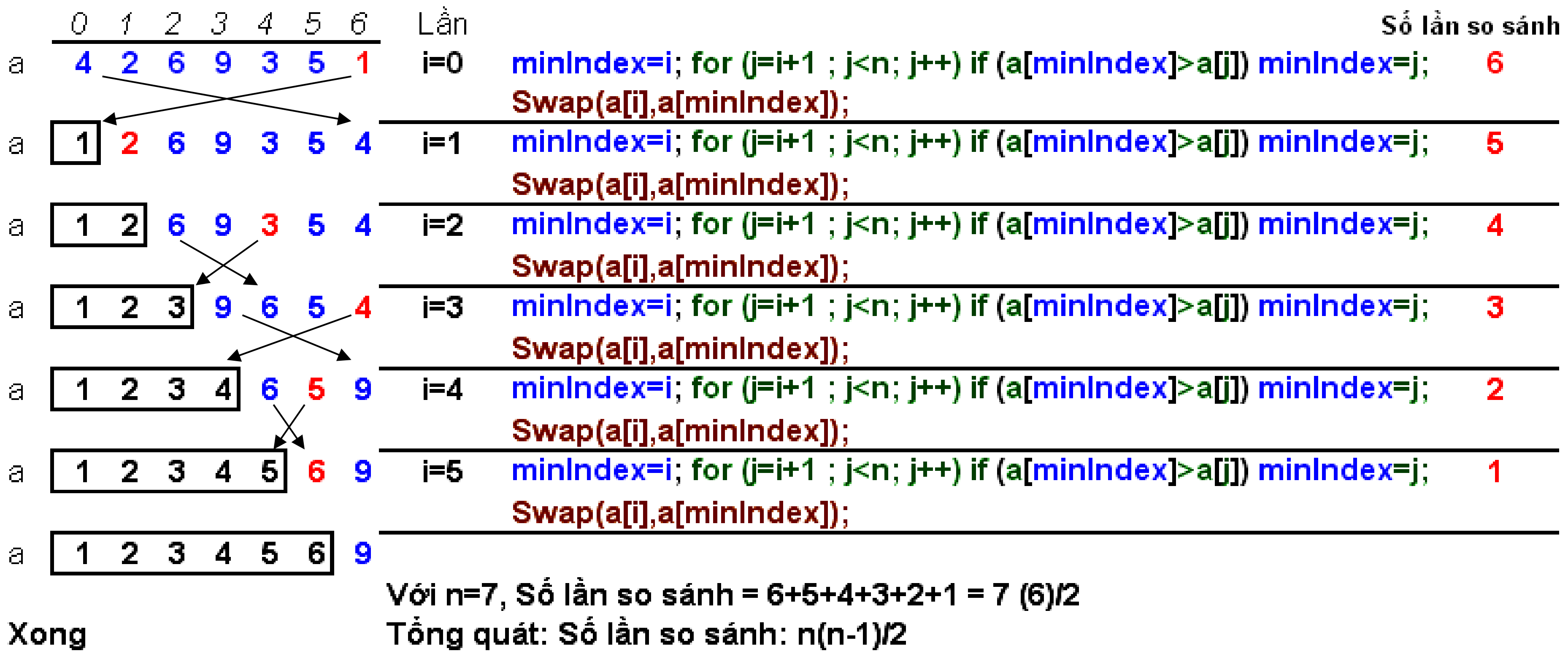
1-D Arrays: Sorting

- **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
 Swapping Elements 64 25 12 22 11 Position to hold Min element Min element	 i=0 5 6 1 3 Swap i=1 5 6 1 3 Swap i=2 5 1 6 3 Sorted Element 5 1 3 6

Sorting: 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

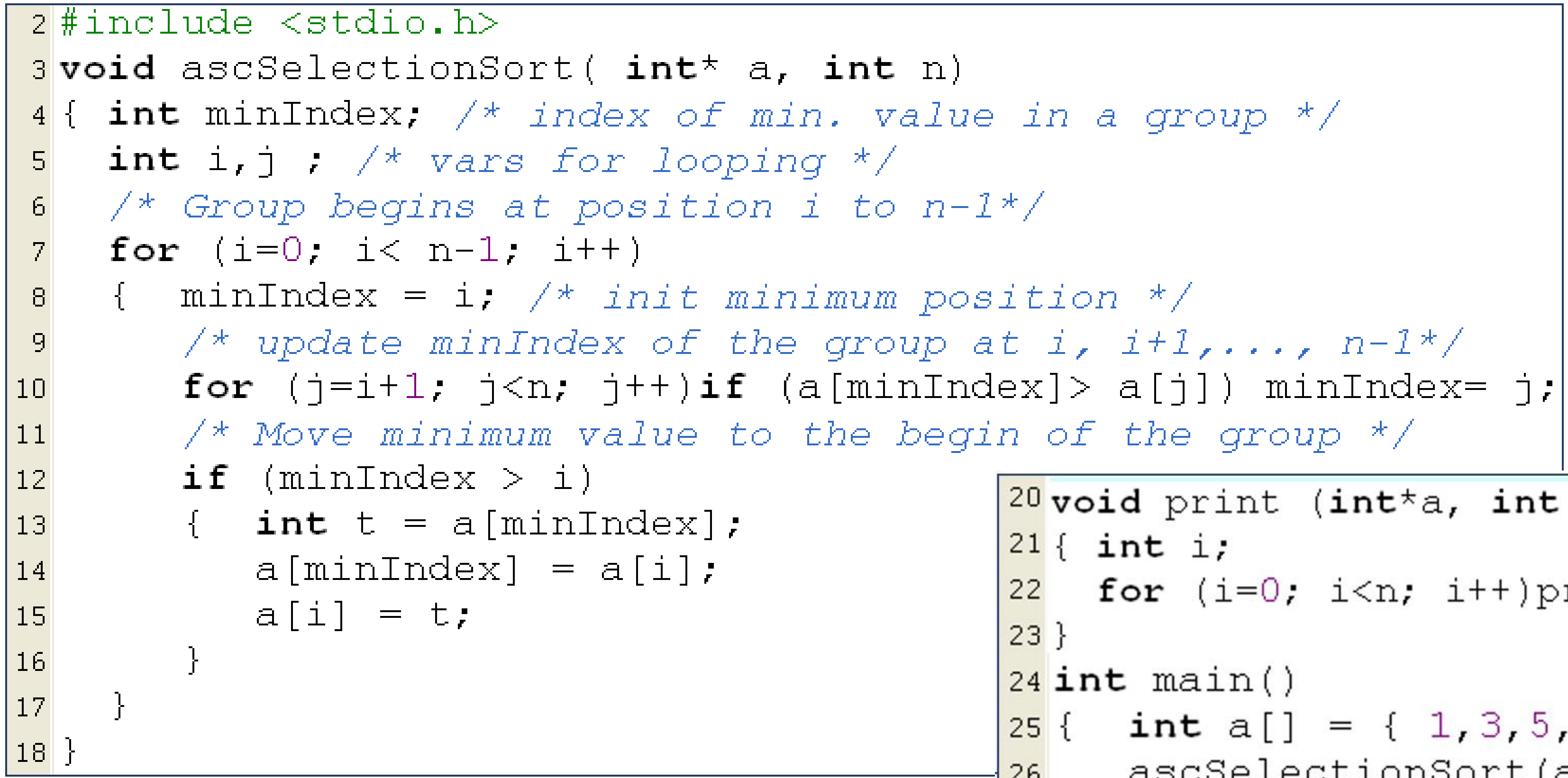


	0 1 2 3 4 5 6	Lần		Số lần so sánh
a	4 2 6 9 3 5 1	i=0	minIndex=i; for (j=i+1 ; j<n; j++) if (a[minIndex]>a[j]) minIndex=j; Swap(a[i],a[minIndex]);	6
a	1 2 6 9 3 5 4	i=1	minIndex=i; for (j=i+1 ; j<n; j++) if (a[minIndex]>a[j]) minIndex=j; Swap(a[i],a[minIndex]);	5
a	1 2 6 9 3 5 4	i=2	minIndex=i; for (j=i+1 ; j<n; j++) if (a[minIndex]>a[j]) minIndex=j; Swap(a[i],a[minIndex]);	4
a	1 2 3 9 6 5 4	i=3	minIndex=i; for (j=i+1 ; j<n; j++) if (a[minIndex]>a[j]) minIndex=j; Swap(a[i],a[minIndex]);	3
a	1 2 3 4 6 5 9	i=4	minIndex=i; for (j=i+1 ; j<n; j++) if (a[minIndex]>a[j]) minIndex=j; Swap(a[i],a[minIndex]);	2
a	1 2 3 4 5 6 9	i=5	minIndex=i; for (j=i+1 ; j<n; j++) if (a[minIndex]>a[j]) minIndex=j; Swap(a[i],a[minIndex]);	1
a	1 2 3 4 5 6 9			

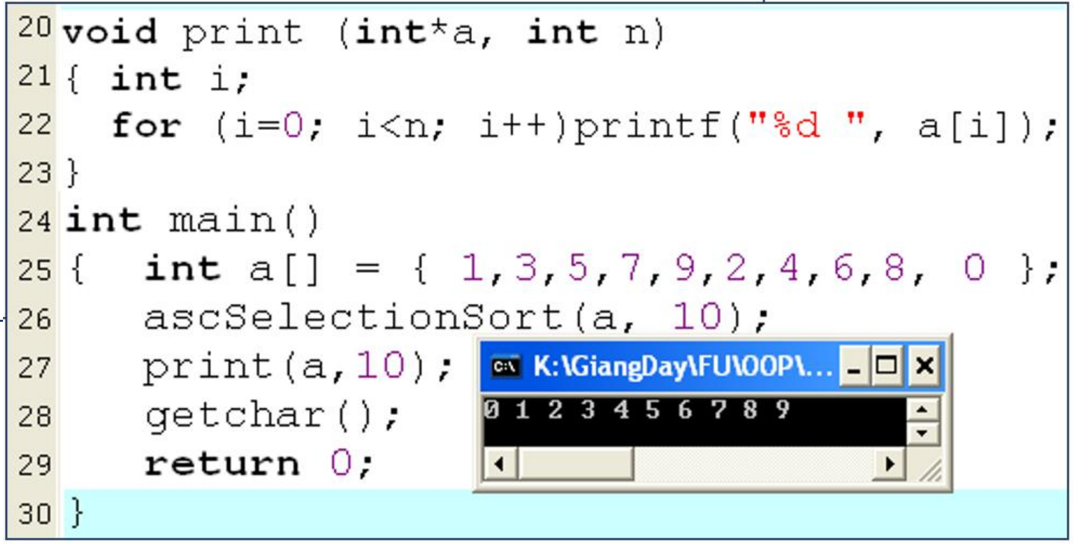
Xong

Với n=7, Số lần so sánh = 6+5+4+3+2+1 = 7 (6)/2
Tổng quát: Số lần so sánh: n(n-1)/2

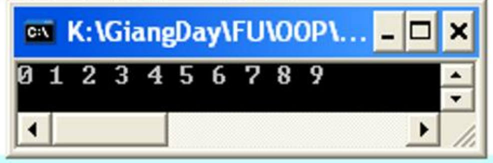
Selection Sort: Students complete the demo



```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)
{ 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 };
   ascSelectionSort(a, 10);
   print(a,10);
   getchar();
   return 0;
}
```

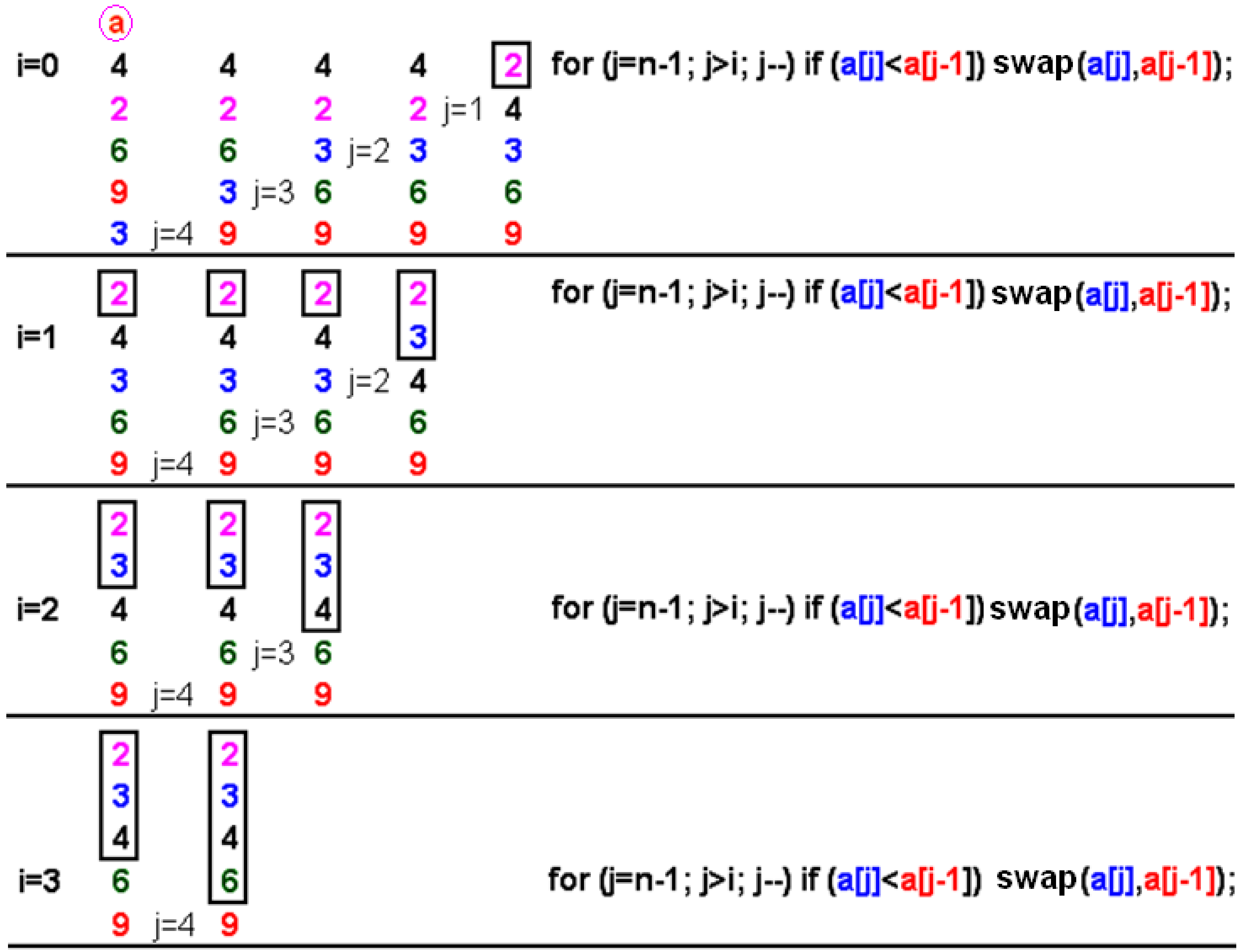


K:\GiangDay\FU\OOP\...

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

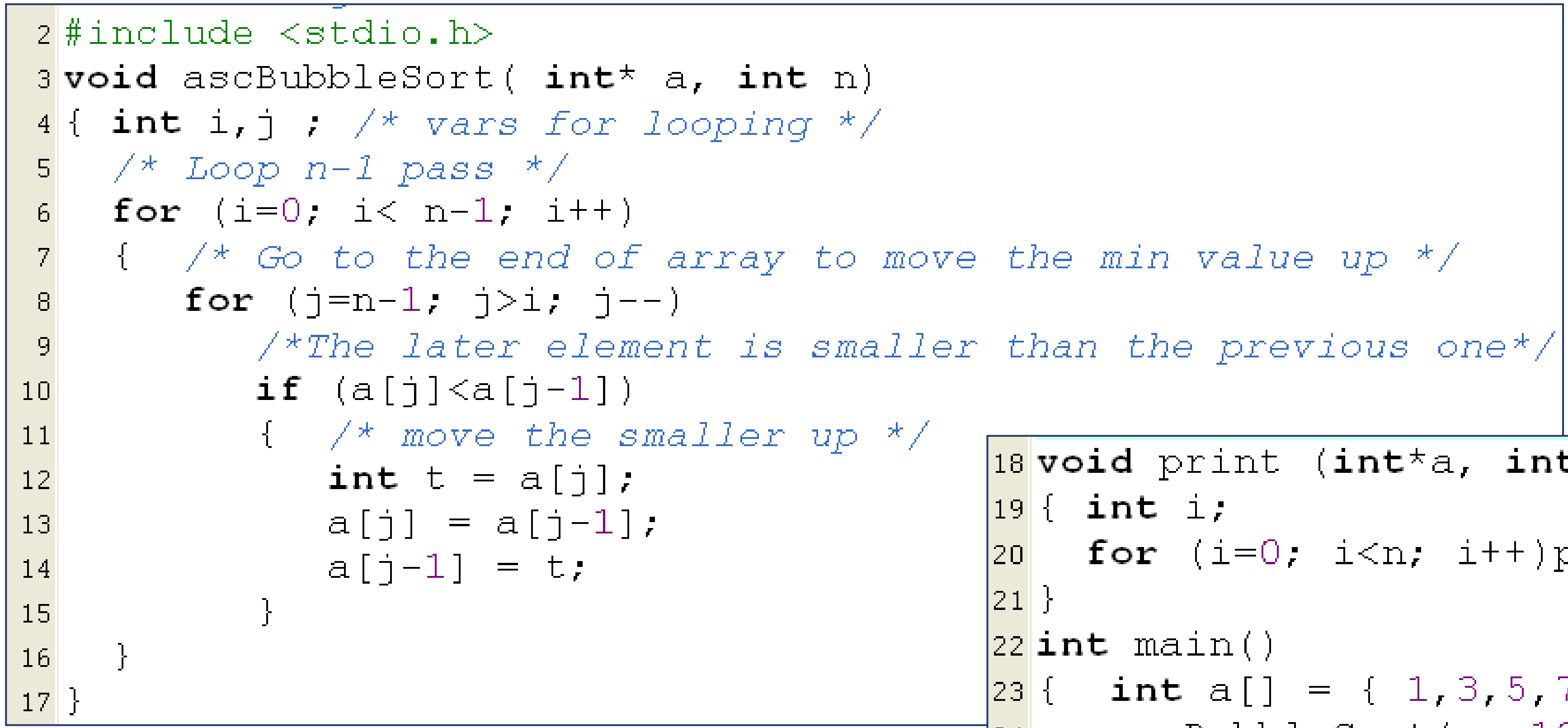
Sorting: Bubble Sort

- 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 <u>repeated until no swaps are needed</u>, which means the list is sorted

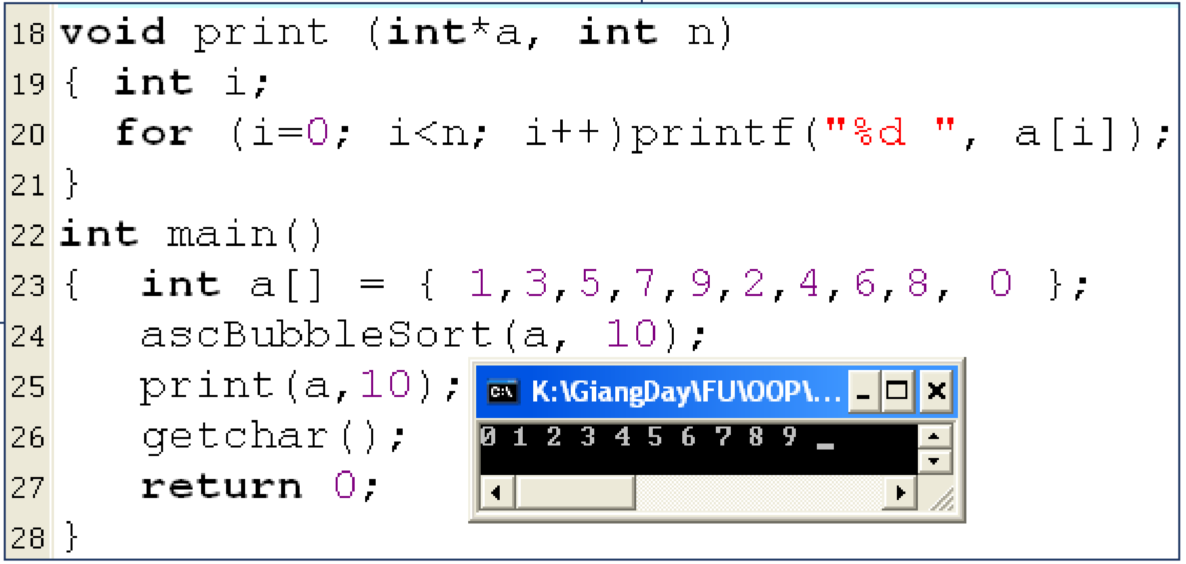


```
for (j=n-1; j>i; j--) if (a[j]<a[j-1]) swap(a[j],a[j-1]);
for (j=n-1; j>i; j--) if (a[j]<a[j-1]) swap(a[j],a[j-1]);
for (j=n-1; j>i; j--) if (a[j]<a[j-1]) swap(a[j],a[j-1]);
for (j=n-1; j>i; j--) if (a[j]<a[j-1]) swap(a[j],a[j-1]);
```

Bubble Sort: Demo



```c
#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;
         }
  }
}
```



```c
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;
}
```

K:\GiangDay\FU\OOP\...

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

1-D Arrays: A Case Study

- 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

Case Study: Problem Analyze

- 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**:

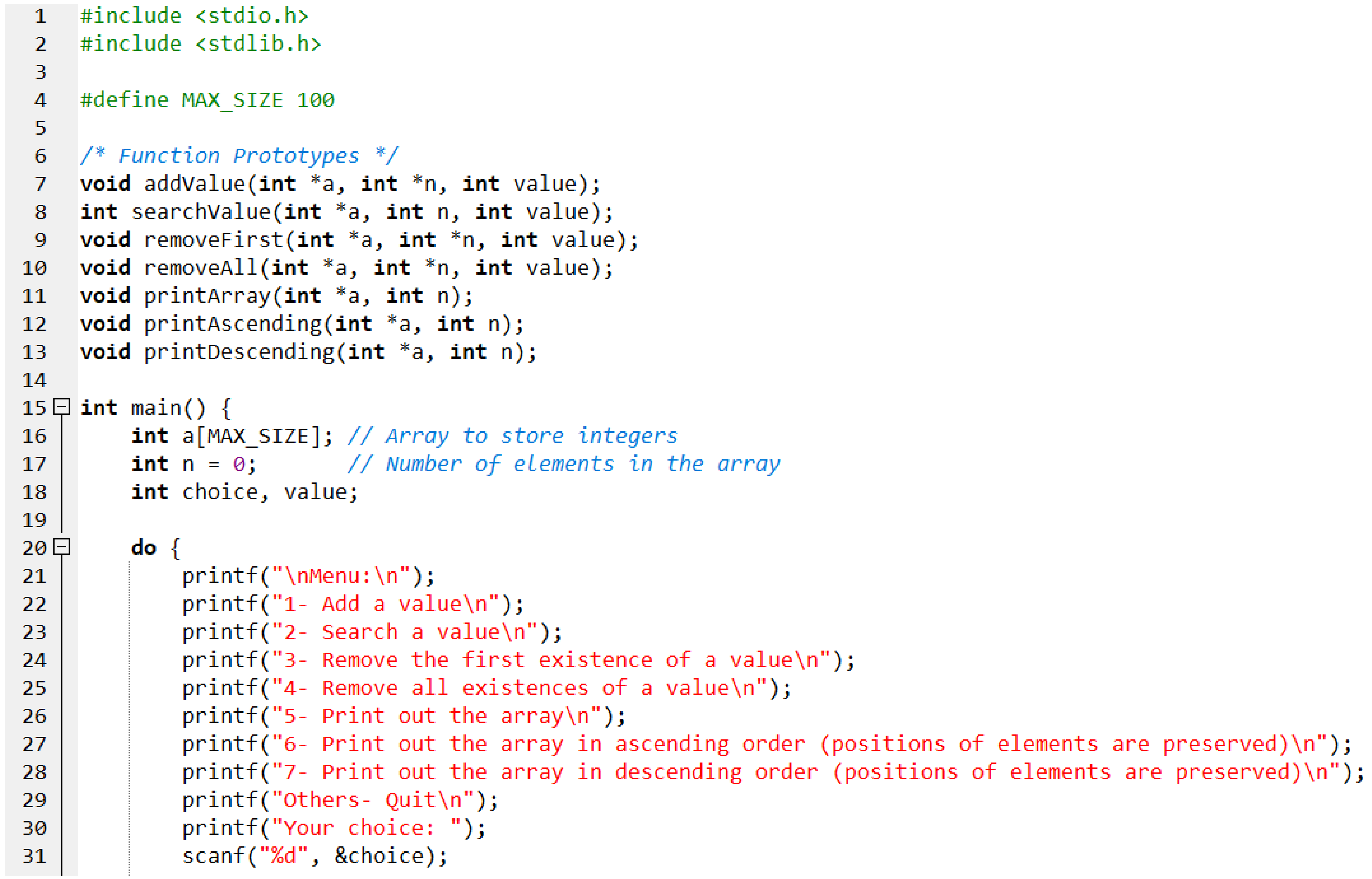
  Array of integers int a[100], n

  Searched/added/removed number → int value

Case Study: Problem Analyze (cont.)

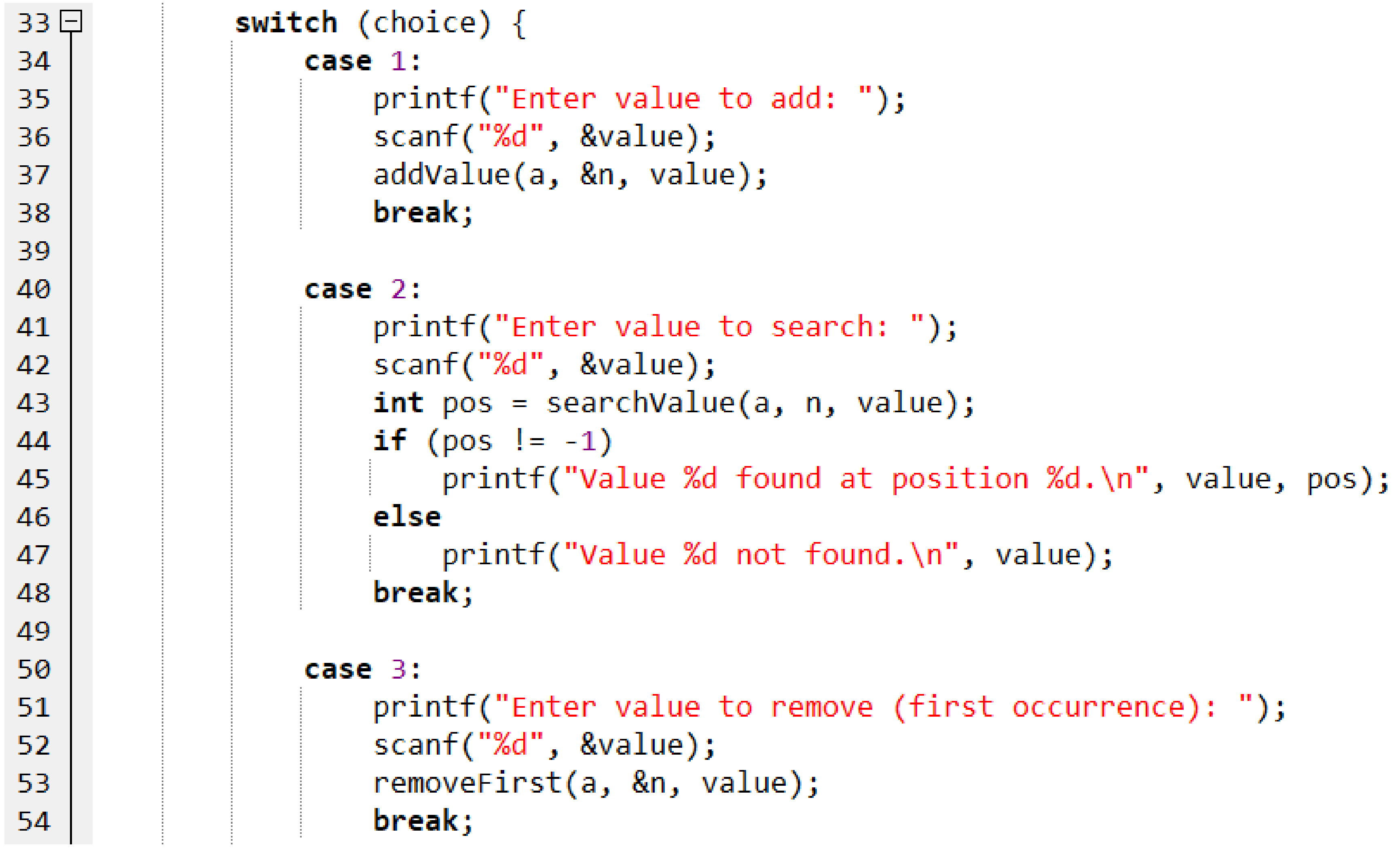
- **Functions** (cont.):
  - int **menu**() → Get user choice
  - int isFull(int *a, int n) - Testing whether an array is full or not
  - int isEmpty(int *a, int n) - Testing whether an array is empty or not
  - void **add**(int x, int*a, int*pn) → adding an element to the array will increase number 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 **printAsc**(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)

Case Study: Code Design



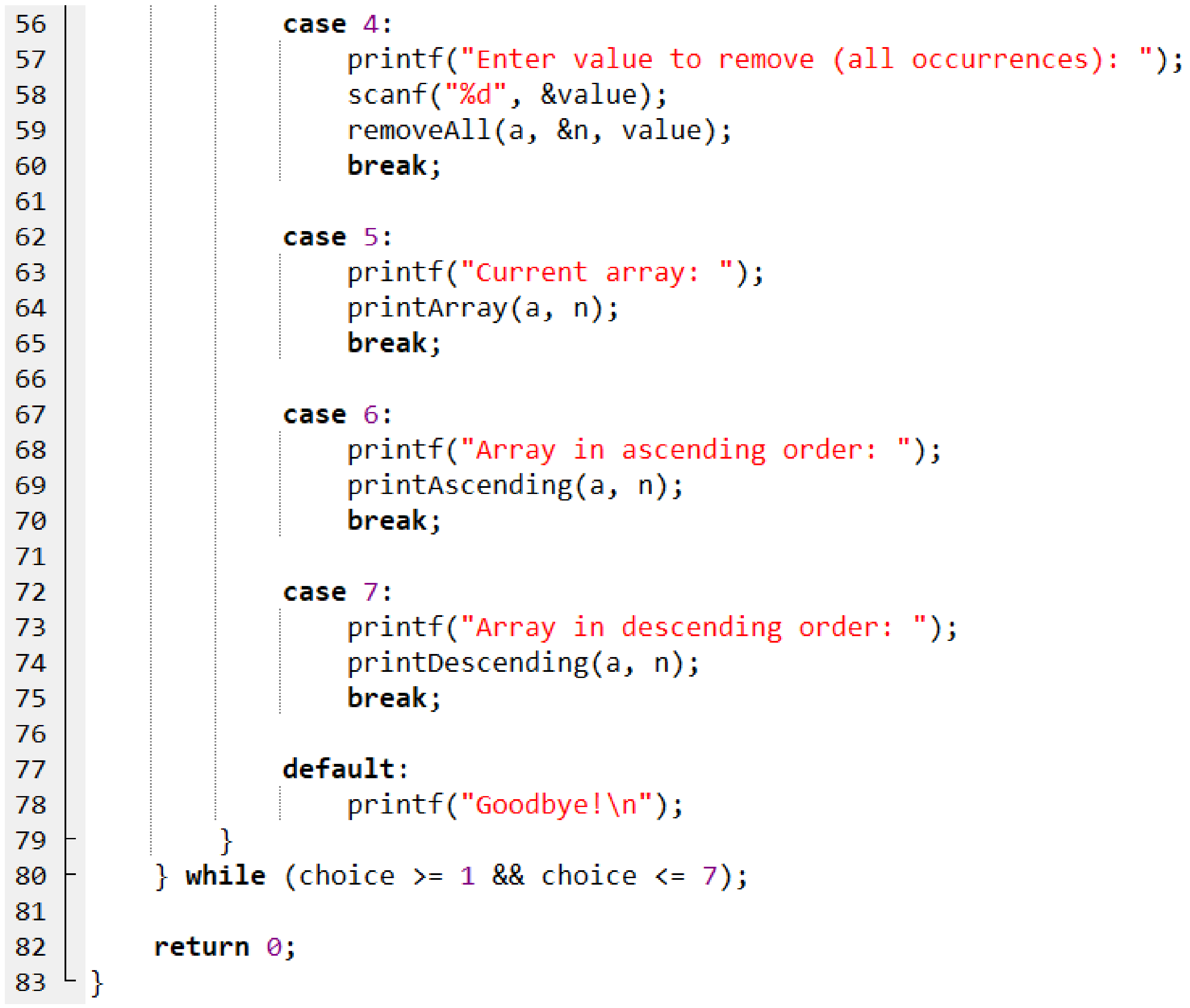
```c
#include <stdio.h>
#include <stdlib.h>

#define MAX_SIZE 100

/* Function Prototypes */
void addValue(int *a, int *n, int value);
int searchValue(int *a, int n, int value);
void removeFirst(int *a, int *n, int value);
void removeAll(int *a, int *n, int value);
void printArray(int *a, int n);
void printAscending(int *a, int n);
void printDescending(int *a, int n);

int main() {
    int a[MAX_SIZE]; // Array to store integers
    int n = 0;       // Number of elements in the array
    int choice, value;

    do {
        printf("\nMenu:\n");
        printf("1- Add a value\n");
        printf("2- Search a value\n");
        printf("3- Remove the first existence of a value\n");
        printf("4- Remove all existences of a value\n");
        printf("5- Print out the array\n");
        printf("6- Print out the array in ascending order (positions of elements are preserved)\n");
        printf("7- Print out the array in descending order (positions of elements are preserved)\n");
        printf("Others- Quit\n");
        printf("Your choice: ");
        scanf("%d", &choice);
```

Case Study: Code Design



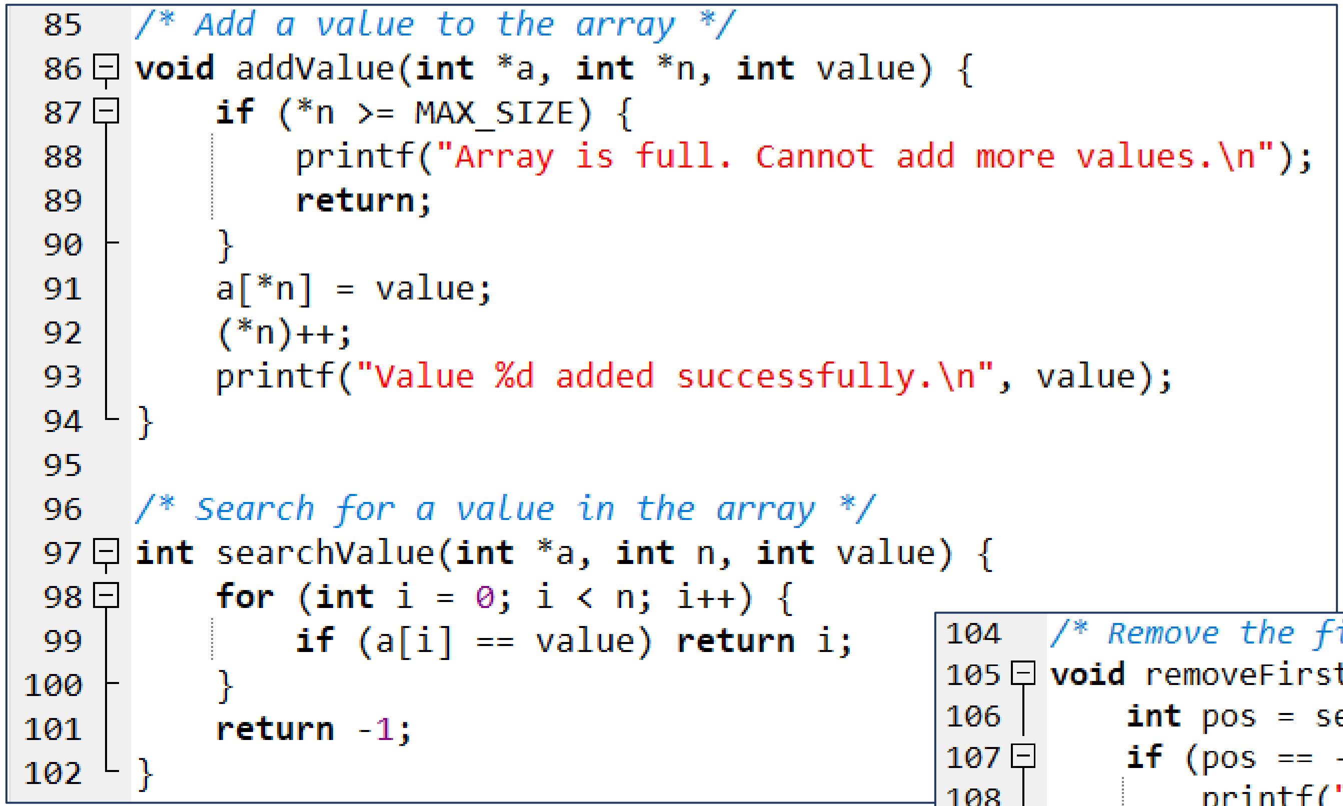
```c
        switch (choice) {
            case 1:
                printf("Enter value to add: ");
                scanf("%d", &value);
                addValue(a, &n, value);
                break;

            case 2:
                printf("Enter value to search: ");
                scanf("%d", &value);
                int pos = searchValue(a, n, value);
                if (pos != -1)
                    printf("Value %d found at position %d.\n", value, pos);
                else
                    printf("Value %d not found.\n", value);
                break;

            case 3:
                printf("Enter value to remove (first occurrence): ");
                scanf("%d", &value);
                removeFirst(a, &n, value);
                break;
```

Case Study: Code Design

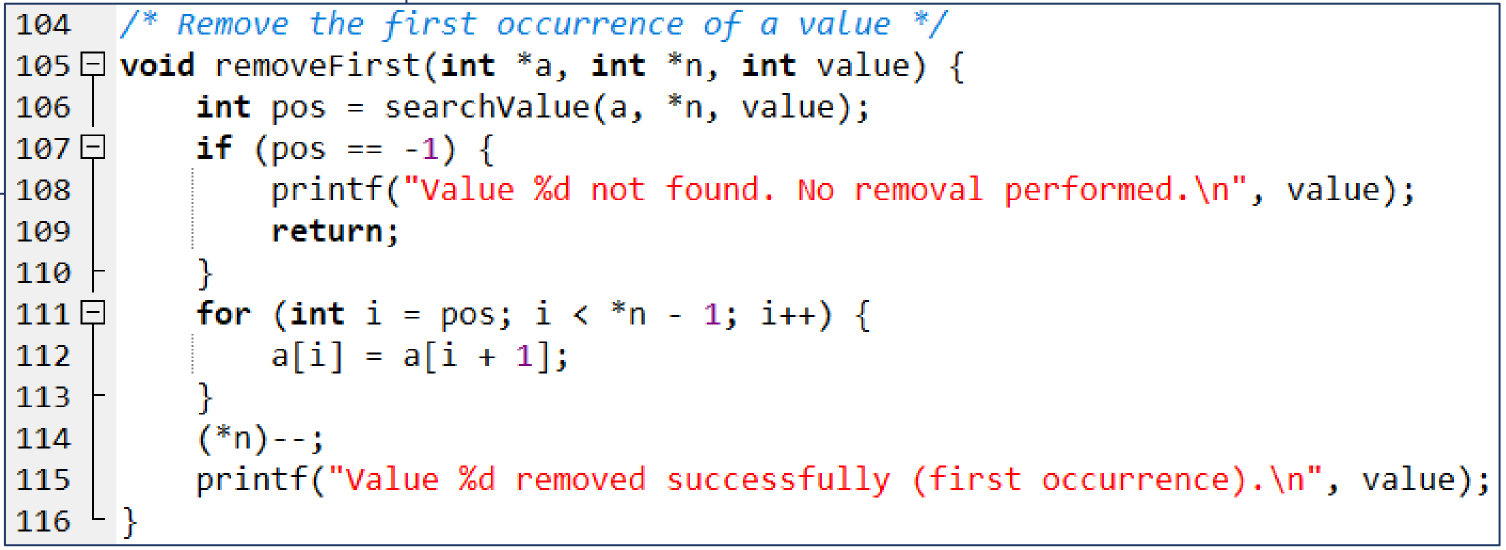


```c
            case 4:
                printf("Enter value to remove (all occurrences): ");
                scanf("%d", &value);
                removeAll(a, &n, value);
                break;

            case 5:
                printf("Current array: ");
                printArray(a, n);
                break;

            case 6:
                printf("Array in ascending order: ");
                printAscending(a, n);
                break;

            case 7:
                printf("Array in descending order: ");
                printDescending(a, n);
                break;

            default:
                printf("Goodbye!\n");
        }
    } while (choice >= 1 && choice <= 7);

    return 0;
}
```

Case Study: Code Design

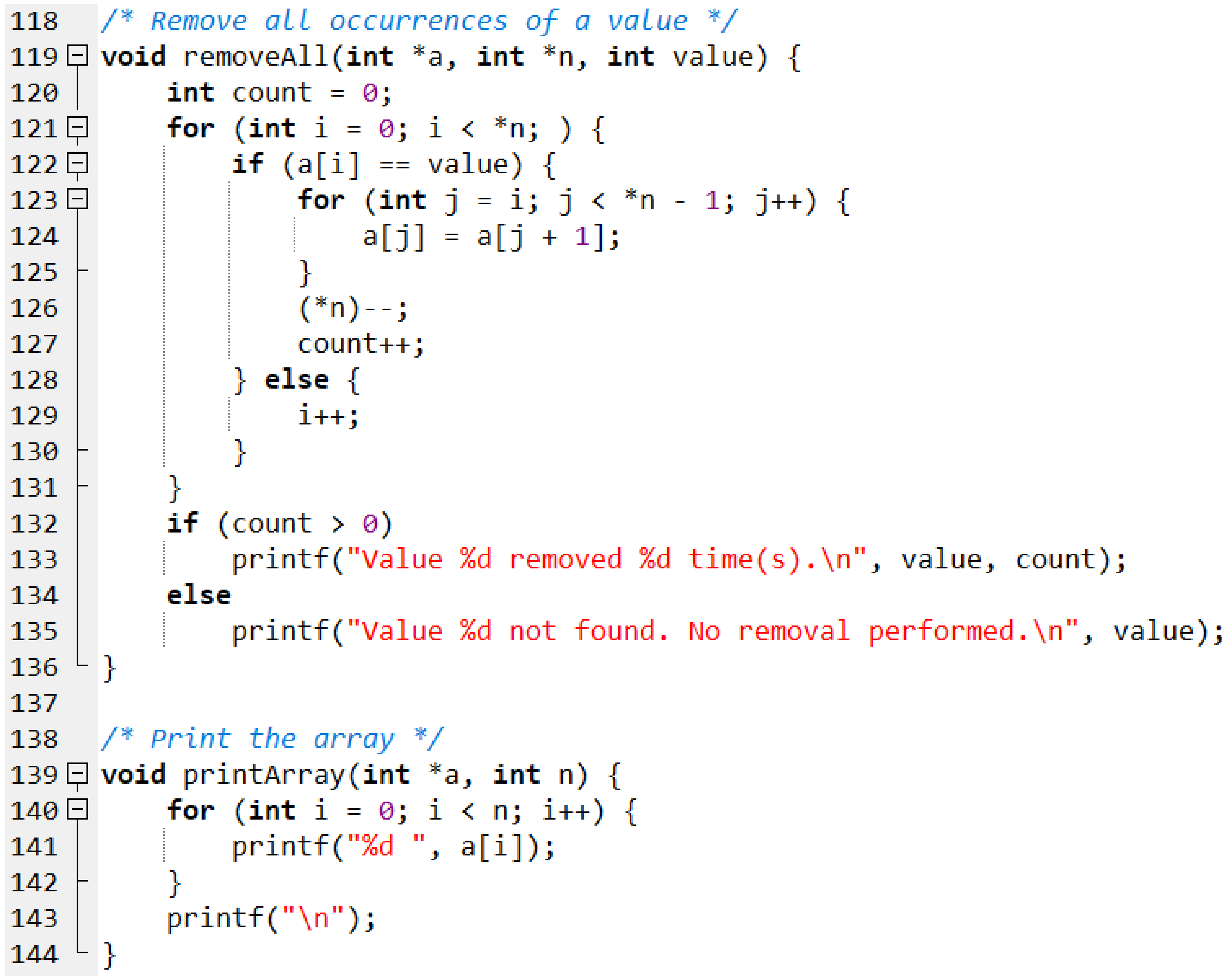


```c
/* Add a value to the array */
void addValue(int *a, int *n, int value) {
    if (*n >= MAX_SIZE) {
        printf("Array is full. Cannot add more values.\n");
        return;
    }
    a[*n] = value;
    (*n)++;
    printf("Value %d added successfully.\n", value);
}

/* Search for a value in the array */
int searchValue(int *a, int n, int value) {
    for (int i = 0; i < n; i++) {
        if (a[i] == value) return i;
    }
    return -1;
}
```



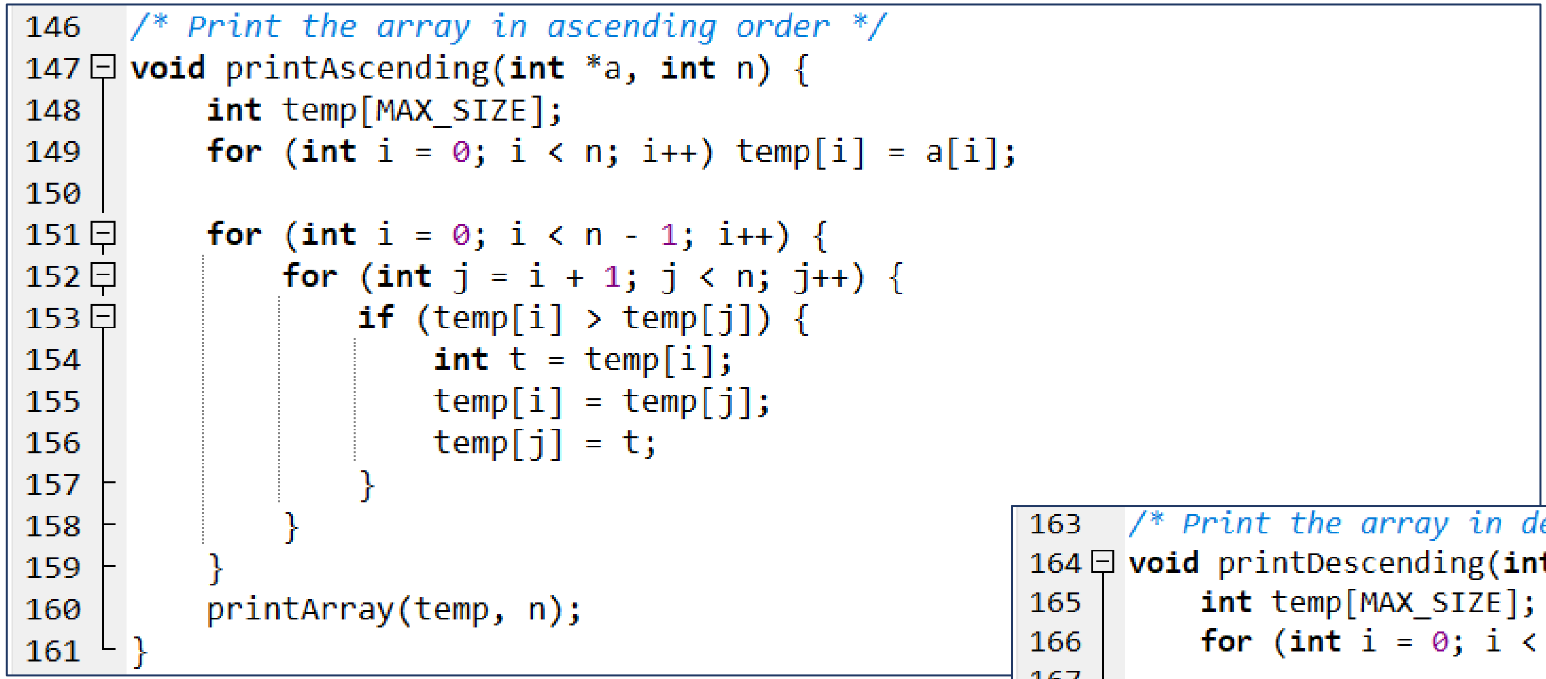
```c
/* Remove the first occurrence of a value */
void removeFirst(int *a, int *n, int value) {
    int pos = searchValue(a, *n, value);
    if (pos == -1) {
        printf("Value %d not found. No removal performed.\n", value);
        return;
    }
    for (int i = pos; i < *n - 1; i++) {
        a[i] = a[i + 1];
    }
    (*n)--;
    printf("Value %d removed successfully (first occurrence).\n", value);
}
```

Case Study: Code Design

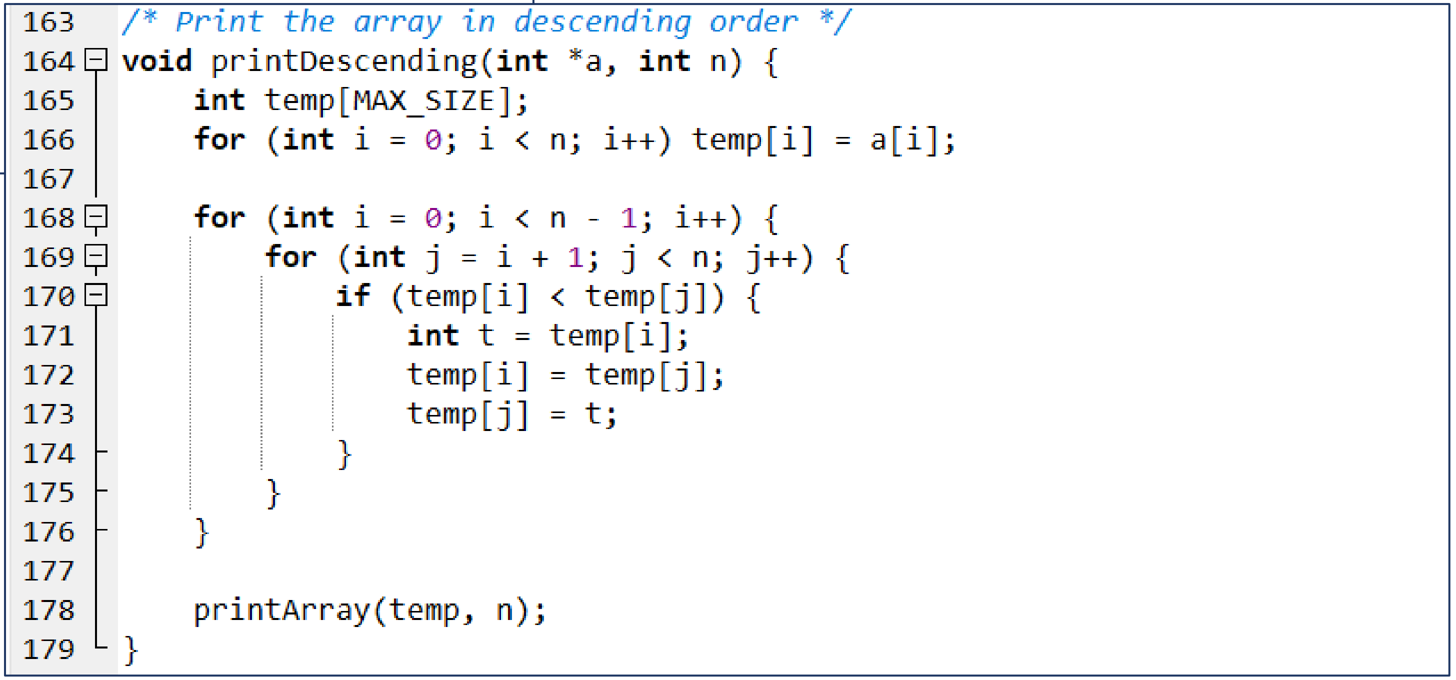


```c
/* Remove all occurrences of a value */
void removeAll(int *a, int *n, int value) {
    int count = 0;
    for (int i = 0; i < *n; ) {
        if (a[i] == value) {
            for (int j = i; j < *n - 1; j++) {
                a[j] = a[j + 1];
            }
            (*n)--;
            count++;
        } else {
            i++;
        }
    }
    if (count > 0)
        printf("Value %d removed %d time(s).\n", value, count);
    else
        printf("Value %d not found. No removal performed.\n", value);
}

/* Print the array */
void printArray(int *a, int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", a[i]);
    }
    printf("\n");
}
```

Case Study: Code Design



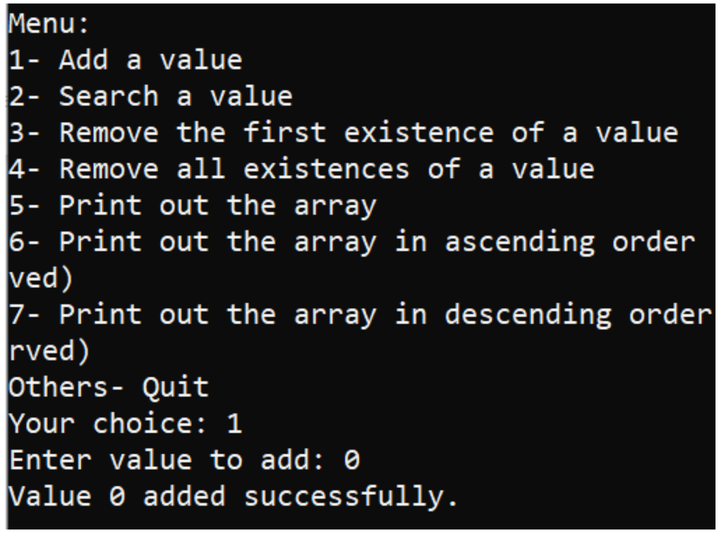
```c
/* Print the array in ascending order */
void printAscending(int *a, int n) {
    int temp[MAX_SIZE];
    for (int i = 0; i < n; i++) temp[i] = a[i];

    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (temp[i] > temp[j]) {
                int t = temp[i];
                temp[i] = temp[j];
                temp[j] = t;
            }
        }
    }
    printArray(temp, n);
}
```



```c
/* Print the array in descending order */
void printDescending(int *a, int n) {
    int temp[MAX_SIZE];
    for (int i = 0; i < n; i++) temp[i] = a[i];

    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (temp[i] < temp[j]) {
                int t = temp[i];
                temp[i] = temp[j];
                temp[j] = t;
            }
        }
    }

    printArray(temp, n);
}
```

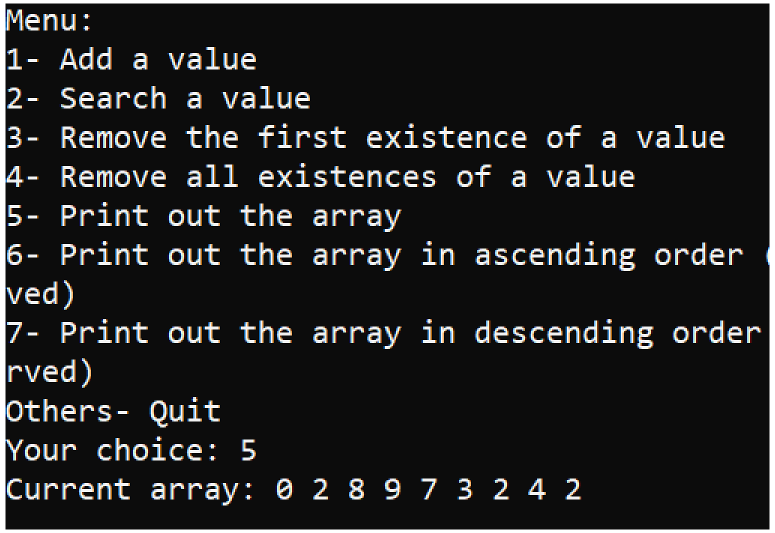
Compile & Run

Input: 9 integer numbers



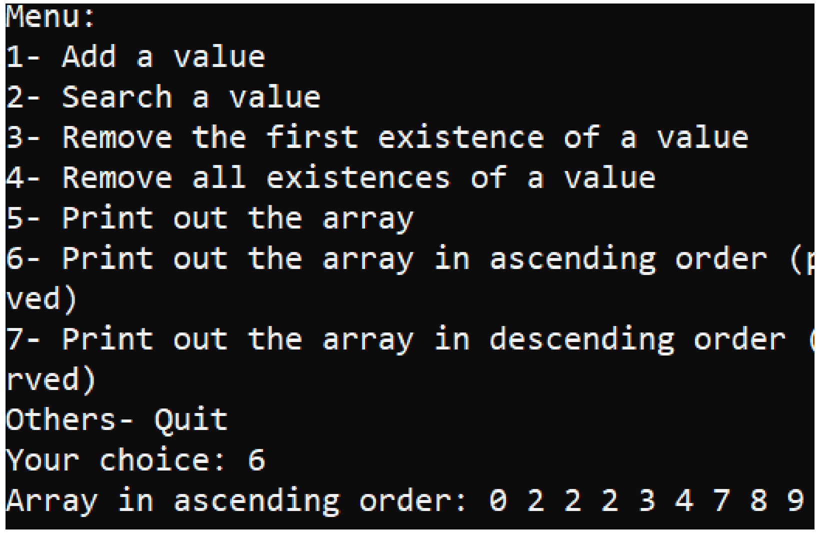
```
Menu:
1- Add a value
2- Search a value
3- Remove the first existence of a value
4- Remove all existences of a value
5- Print out the array
6- Print out the array in ascending order
ved)
7- Print out the array in descending order
rved)
Others- Quit
Your choice: 1
Enter value to add: 0
Value 0 added successfully.
```

Print all elements



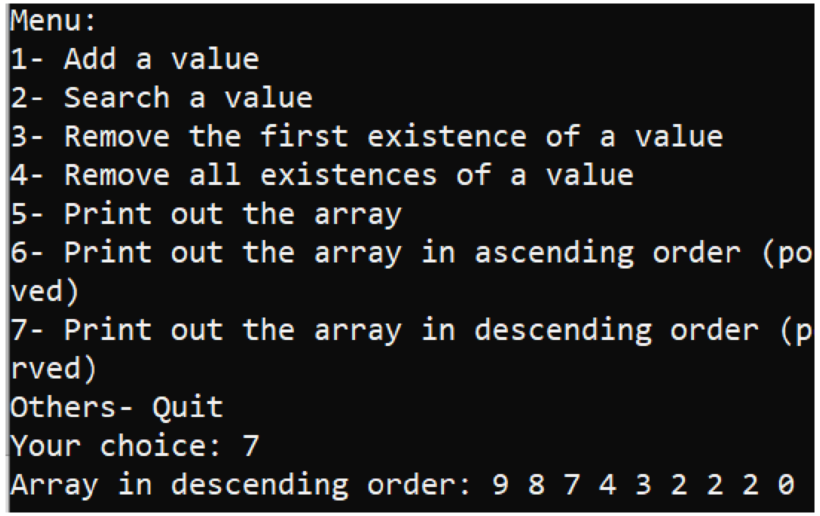
```
Menu:
1- Add a value
2- Search a value
3- Remove the first existence of a value
4- Remove all existences of a value
5- Print out the array
6- Print out the array in ascending order (
ved)
7- Print out the array in descending order
rved)
Others- Quit
Your choice: 5
Current array: 0 2 8 9 7 3 2 4 2
```

Print the array in ASC order



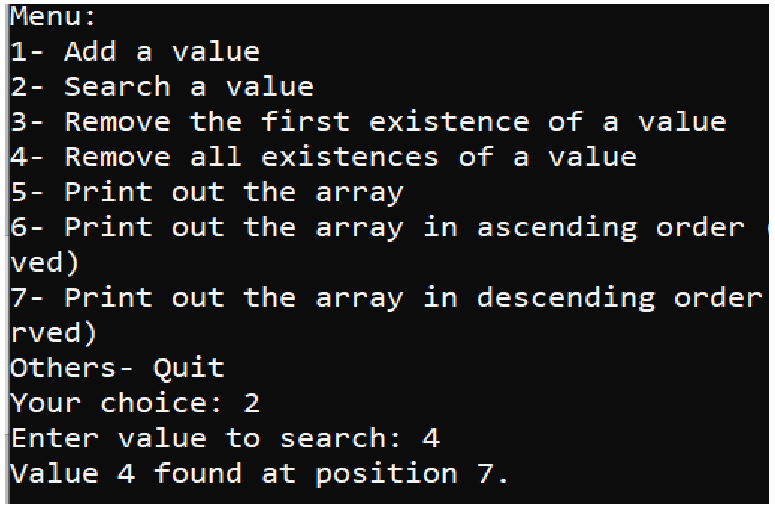
```
Menu:
1- Add a value
2- Search a value
3- Remove the first existence of a value
4- Remove all existences of a value
5- Print out the array
6- Print out the array in ascending order (p
ved)
7- Print out the array in descending order (
rved)
Others- Quit
Your choice: 6
Array in ascending order: 0 2 2 2 3 4 7 8 9
```

Print the array in DESC order



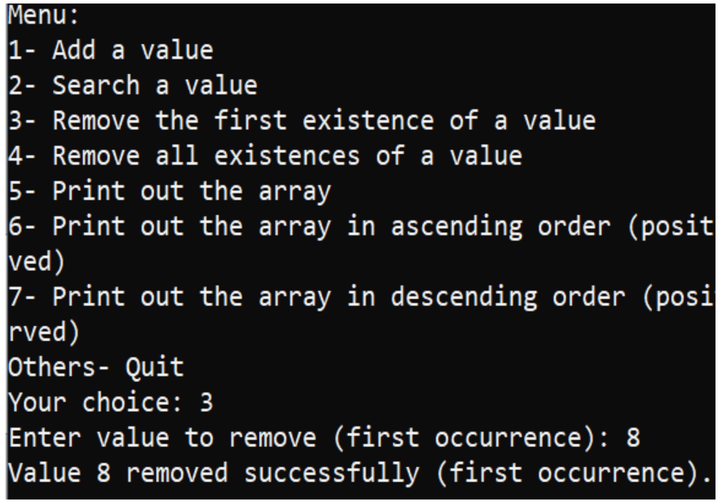
```
Menu:
1- Add a value
2- Search a value
3- Remove the first existence of a value
4- Remove all existences of a value
5- Print out the array
6- Print out the array in ascending order (po
ved)
7- Print out the array in descending order (p
rved)
Others- Quit
Your choice: 7
Array in descending order: 9 8 7 4 3 2 2 2 0
```

Search a value



```
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 (
ved)
7- Print out the array in descending order
rved)
Others- Quit
Your choice: 2
Enter value to search: 4
Value 4 found at position 7.
```

Remove the first exist of value



```
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 (posit
ved)
7- Print out the array in descending order (posi
rved)
Others- Quit
Your choice: 3
Enter value to remove (first occurrence): 8
Value 8 removed successfully (first occurrence).
```

Exercise 4:

- 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)

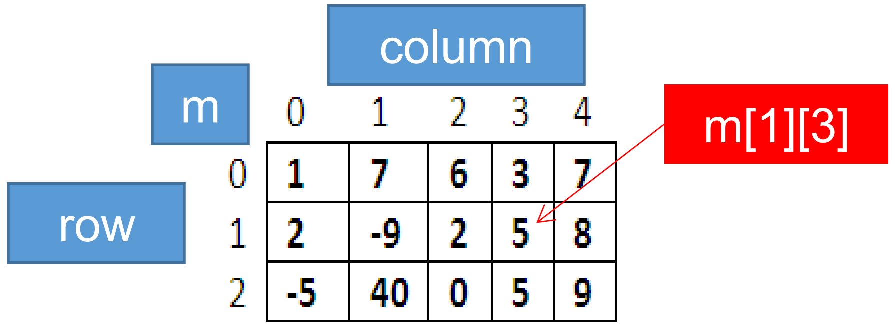
  Others- Quit

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:

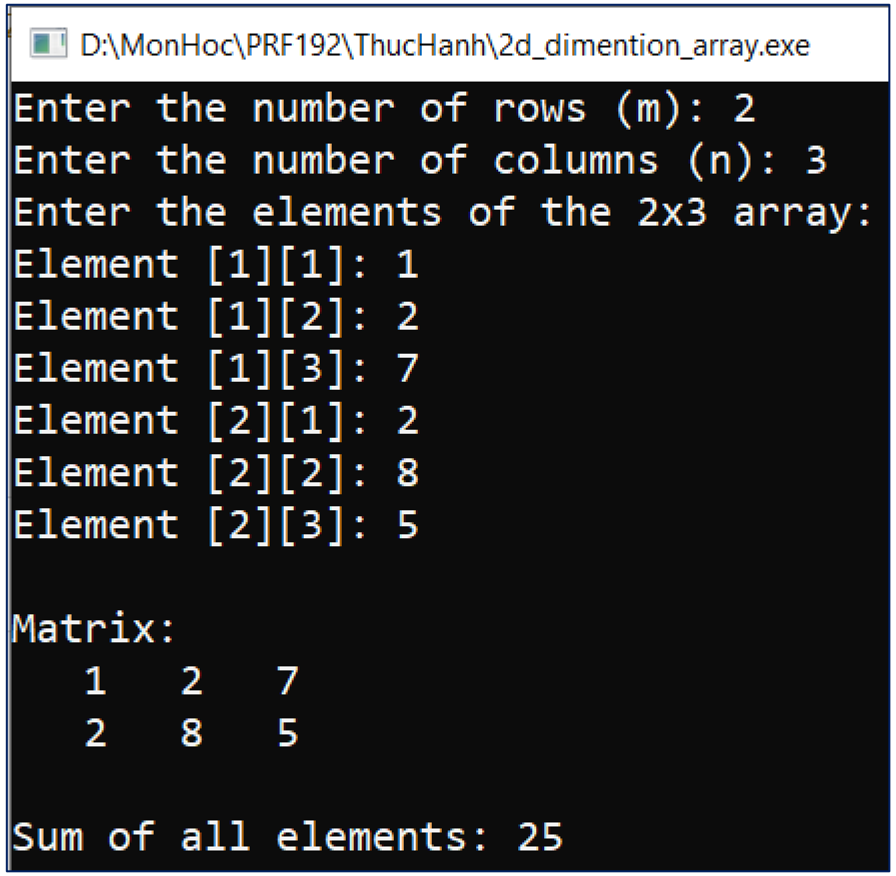
```
for ( i =0; i<row; i++)
{
        for ( j=0; j< column; j++)
                [if (condition)] Access m[i][j];
}
```



$$\begin{array}{c|ccccc} m & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & 1 & 7 & 6 & 3 & 7 \\ 1 & 2 & -9 & 2 & 5 & 8 \\ 2 & -5 & 40 & 0 & 5 & 9 \end{array}$$

Two-Dimensional Arrays: Example

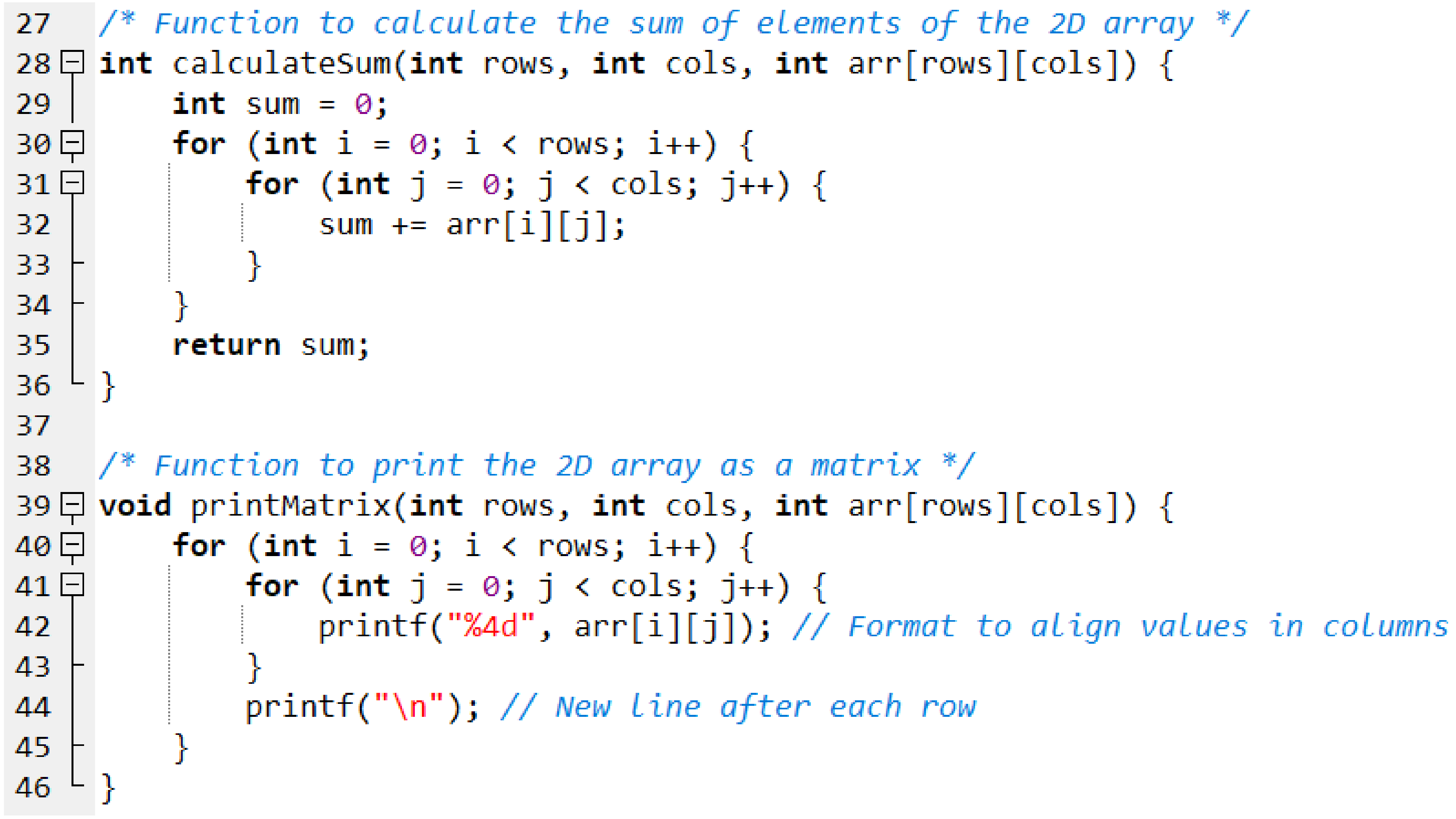
```c
#include <stdio.h>
/* Function Prototypes */
int calculateSum(int rows, int cols, int arr[rows][cols]);
void printMatrix(int rows, int cols, int arr[rows][cols]);

int main() {
    int m, n;
    printf("Enter the number of rows (m): ");
    scanf("%d", &m);
    printf("Enter the number of columns (n): ");
    scanf("%d", &n);
    int array[m][n];
    printf("Enter the elements of the %dx%d array:\n", m, n);
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            printf("Element [%d][%d]: ", i + 1, j + 1);
            scanf("%d", &array[i][j]);
        }
    }
    printf("\nMatrix:\n");
    printMatrix(m, n, array);
    int sum = calculateSum(m, n, array);
    printf("\nSum of all elements: %d\n", sum);
    return 0;
}
```



D:\MonHoc\PRF192\ThucHanh\2d_dimention_array.exe

```
Enter the number of rows (m): 2
Enter the number of columns (n): 3
Enter the elements of the 2x3 array:
Element [1][1]: 1
Element [1][2]: 2
Element [1][3]: 7
Element [2][1]: 2
Element [2][2]: 8
Element [2][3]: 5

Matrix:
   1   2   7
   2   8   5

Sum of all elements: 25
```

Two-Dimensional Arrays: Example (cont.)



```c
/* Function to calculate the sum of elements of the 2D array */
int calculateSum(int rows, int cols, int arr[rows][cols]) {
    int sum = 0;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            sum += arr[i][j];
        }
    }
    return sum;
}

/* Function to print the 2D array as a matrix */
void printMatrix(int rows, int cols, int arr[rows][cols]) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%4d", arr[i][j]); // Format to align values in columns
        }
        printf("\n"); // New line after each row
    }
}
```

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

```
DataType a[MAXN];
DataType 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

Structures

Objectives

After studying this section, you should be able to:

- What is C structure?
- When to use structures.
- Syntax of a structure.
- How to declare variable of type structure?
- Fields of a structure and how to initialize them.
- How to manipulate structure type

Contents

1. Structure Definition

- The structure in C is a user-defined data type that can be used to group items of possibly different types into a single type.
- Structures are also called records.
- The **struct** keyword is used to define the structure in the C programming language.
- Unlike arrays, a struct is composed of data of different types.
- You use structures to group data that belong together.
- Additionally, the values of a structure are stored in contiguous memory locations.

Structure Definition (cont.)

- Examples:

Student information:	**Bank account information:**
- student id,	- account number,
- last name,	- account type
- first name	- account holder
- major,	+ first name
- gender,	+ last name
…	- balance

- Data elements in a structure are called **fields** or **members**.
- Complex data structures can be formed by defining arrays of structs.

2. Struct Syntax

- Syntax of the structure type:

```
typedef struct{
        dataType1 field1;
        dataType2 field2;
        …
} structName;
```

Or

```
struct structName{
        dataType1 field1;
        dataType2 field2;
        …
};
```

Syntax Structure (cont.)

- Examples:

Example 1	Example 2	Example 3

typedef struct {
    int day;
    int month;
    int year;
} eventDate;
	typedef struct {
	    char name[20];
	    int age;
	} person;
		struct telephone {
		    char name[30];
		    int number;
		};

- How to declare variable of type structure?

```
structName variableName;
```

- Example: eventDate ev; person p; telephone tel;

3. Manipulating Structure Types

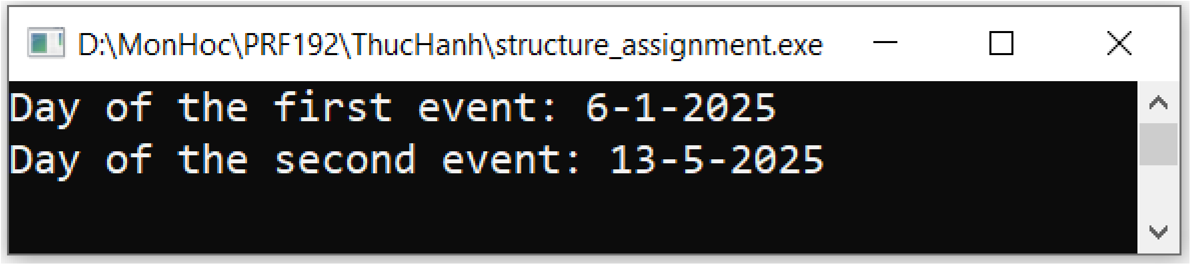
- **How to access a field in a structure**:
  - Use the direct component selection operator, which is a **period** ‘**.**’
  - The ‘**.**’ operator has the highest priority in the operator precedence.
  - Example:

```
person p1;
p1.name;
p1.age;
```

- **Structure assignment**:
  - The copy of an entire structure can be easily done by the assignment operator.
  - Each component in one structure is copied into the corresponding component in the other structure.

Example 1: Structure Type

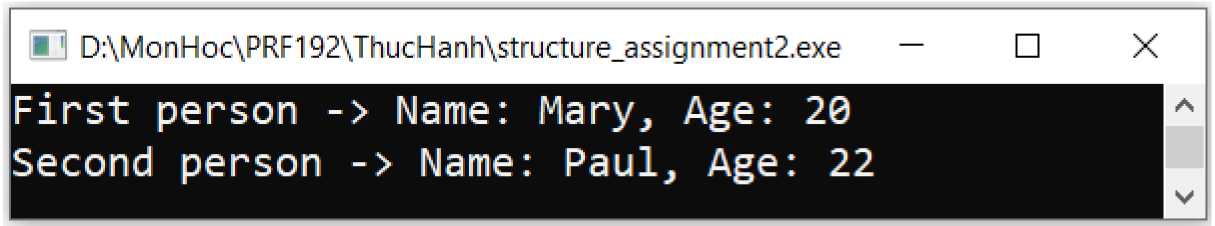
```c
#include <stdio.h>

typedef struct{
    int day;
    int month;
    int year;
} eventDate;

int main(){
    // Declare a variable with struct type
    eventDate ev1;

    // Assignment value into the members of 'ev1'
    ev1.day = 6;
    ev1.month = 1;
    ev1.year = 2025;

    // Or: Declare and Initialization
    eventDate ev2 = {13, 05, 2025};

    printf("Day of the first event: %d-%d-%d\n", ev1.day, ev1.month, ev1.year);
    printf("Day of the second event: %d-%d-%d\n", ev2.day, ev2.month, ev2.year);

    return 0;
}
```


D:\MonHoc\PRF192\ThucHanh\structure_assignment.exe

```
Day of the first event: 6-1-2025
Day of the second event: 13-5-2025
```

Example 2: Structure Type

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

struct person{
	char name[20];
	int age;
};

int main(){
	// Declare and Initialization
	struct person p1 = {"Mary", 20};
	struct person p2;
	strcpy(p2.name, "Paul"); // Copy 'Paul' to name of p2
	p2.age = 22;
	
	printf("First person -> Name: %s, Age: %d\n", p1.name, p1.age);
	printf("Second person -> Name: %s, Age: %d\n", p2.name, p2.age);
	
	return 0;
}
```


D:\MonHoc\PRF192\ThucHanh\structure_assignment2.exe

```
First person -> Name: Mary, Age: 20
Second person -> Name: Paul, Age: 22
```

4. Arrays of Structures

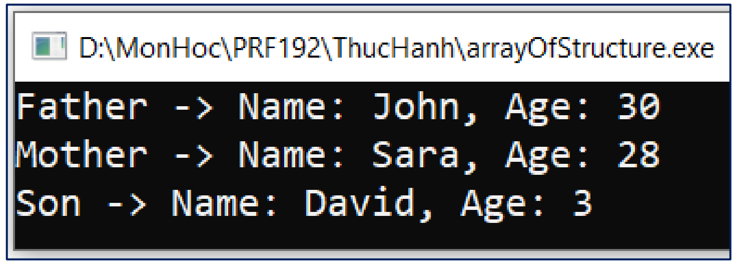
- We can also declare an array of structures.
- Recall the syntax of an array: dataType array_name[size];
  - **dataType** can any C type including struct type
- The array of structures can be simply manipulated as arrays of simple data types.

- **Example**:

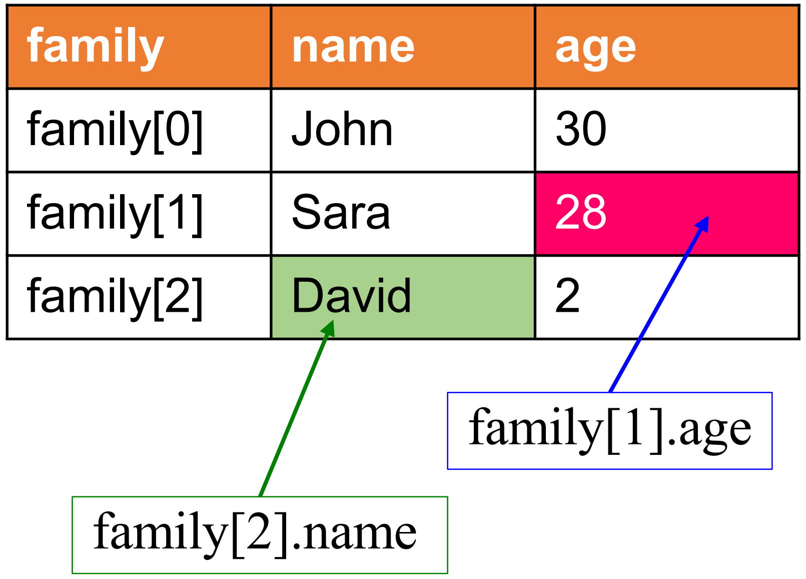
  Write a program to organize information, includes: **name** and **age** of three family members. Print out the information for all members on each line.

Example 1: Array of Structures

```c
#include <stdio.h>
#include <string.h>
typedef struct{
    char name[20];
    int age;
} person;

int main(){
    // Declara an array of structure
    person family[3];

    strcpy(family[0].name, "John"); family[0].age = 30;
    strcpy(family[1].name, "Sara"); family[1].age = 28;
    strcpy(family[2].name, "David"); family[2].age = 3;

    printf("Father -> Name: %s, Age: %d\n", family[0].name, family[0].age);
    printf("Mother -> Name: %s, Age: %d\n", family[1].name, family[1].age);
    printf("Son -> Name: %s, Age: %d\n", family[2].name, family[2].age);

    return 0;
}
```



```
Father -> Name: John, Age: 30
Mother -> Name: Sara, Age: 28
Son -> Name: David, Age: 3
```



family	name	age
family[0]	John	30
family[1]	Sara	28
family[2]	David	2

5. Function with a Structure Input Parameter

- When a structure variable is passed as an input argument to a function, all its component values are copied into the local structure variable.
- **Syntax**:

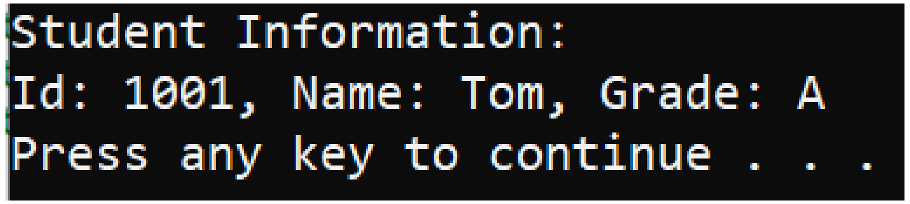
```
dataType functionName(structName parameter){ … }
```

- **Example**:

  Write a C program to print a student's information including: id, name, age. Use a structure to define the student's data structure and use a function with a parameter of type structure to print the information.

Function with a Structure Input Parameter: Example 2

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

typedef struct{
    int id;
    char name[20];
    char grade;
} student;

void printStudent(student s){
    printf("Id: %d, Name: %s, Grade: %c\n", s.id, s.name, s.grade);
}

int main(){
    // Declaration a variable with structure type
    student s1;
    // Assigment the value into members of structure
    s1.id = 1001; strcpy(s1.name, "Tom"); s1.grade = 'A';
    // Print student
    printf("Student Information:\n");
    printStudent(s1);

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

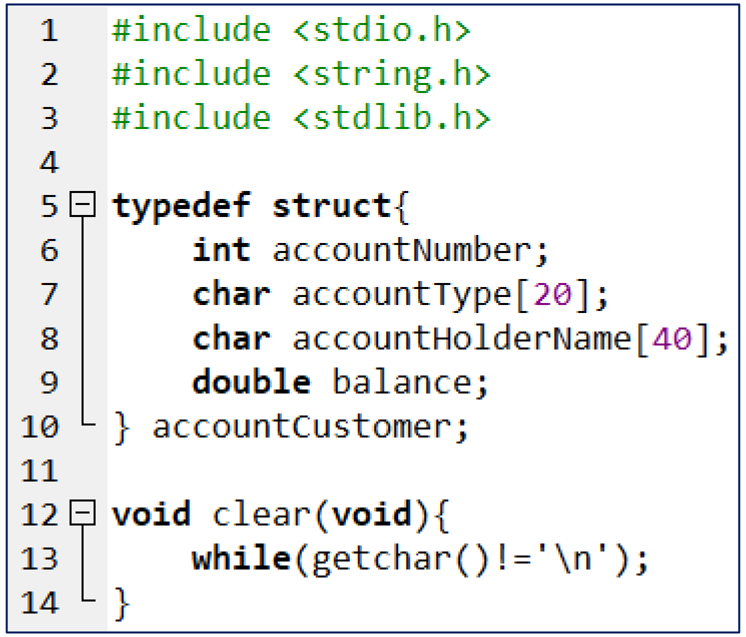


```
Student Information:
Id: 1001, Name: Tom, Grade: A
Press any key to continue . . .
```

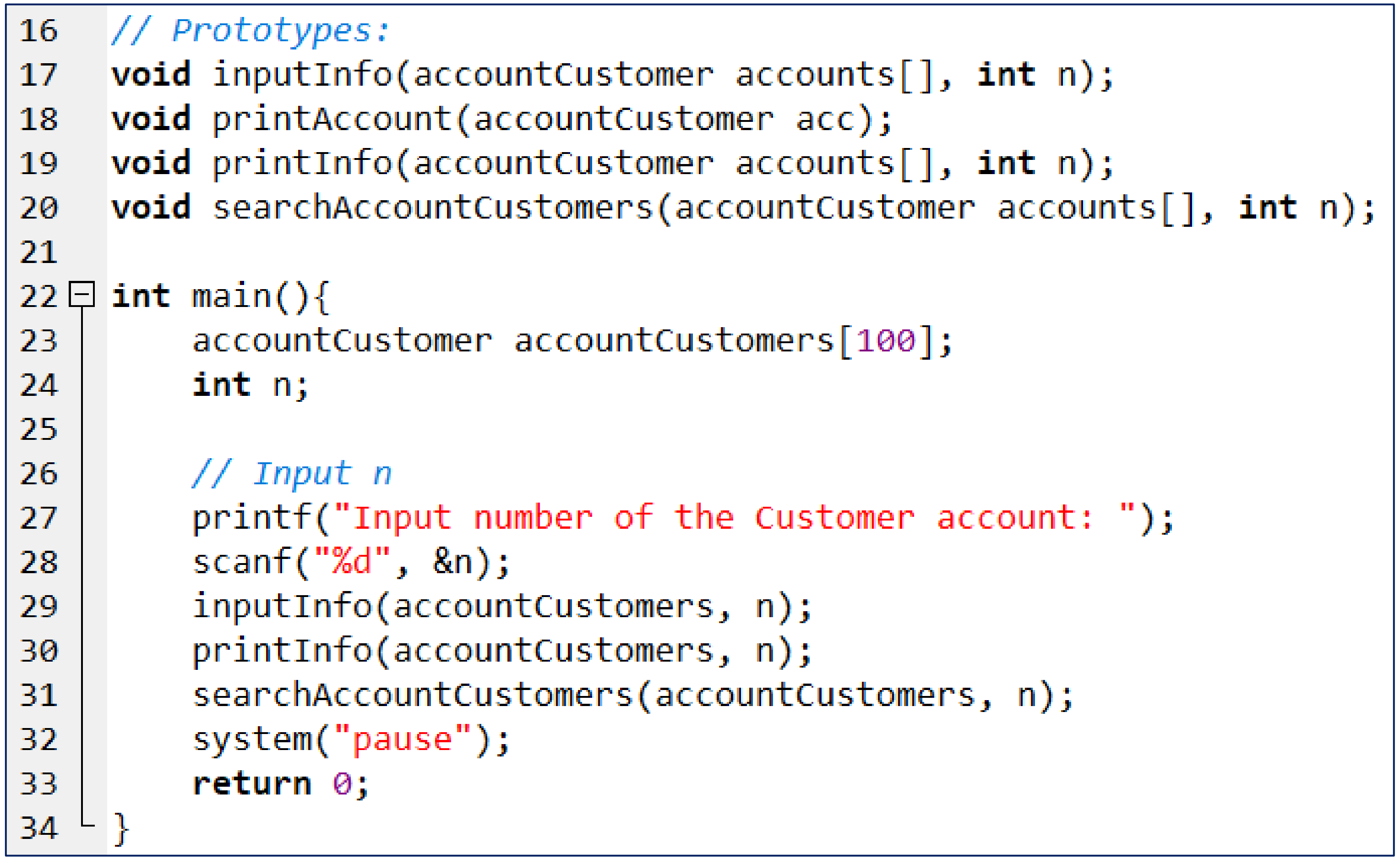
Case Study:

- Write a C program that allows users to manage account information of up to 100 customers. Customer account information includes: **accountNumber** (int); **accountType** (char[20]); **accountHolderName** (char[40]); **balance** (double).
- **Requirements**:
  - Enter customer account information from the keyboard
  - Print out a list of customer account information
  - Find and print out information on customers with a balance greater than $1000

Case Study: Code design

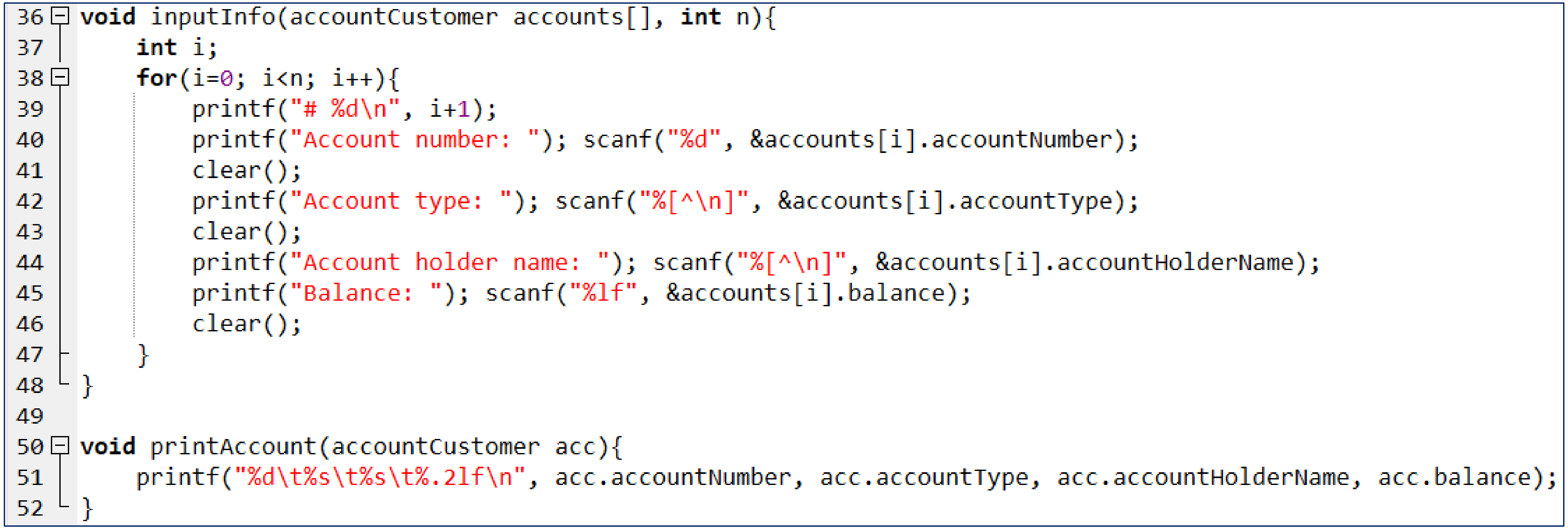


```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

typedef struct{
    int accountNumber;
    char accountType[20];
    char accountHolderName[40];
    double balance;
} accountCustomer;

void clear(void){
    while(getchar()!='\n');
}
```



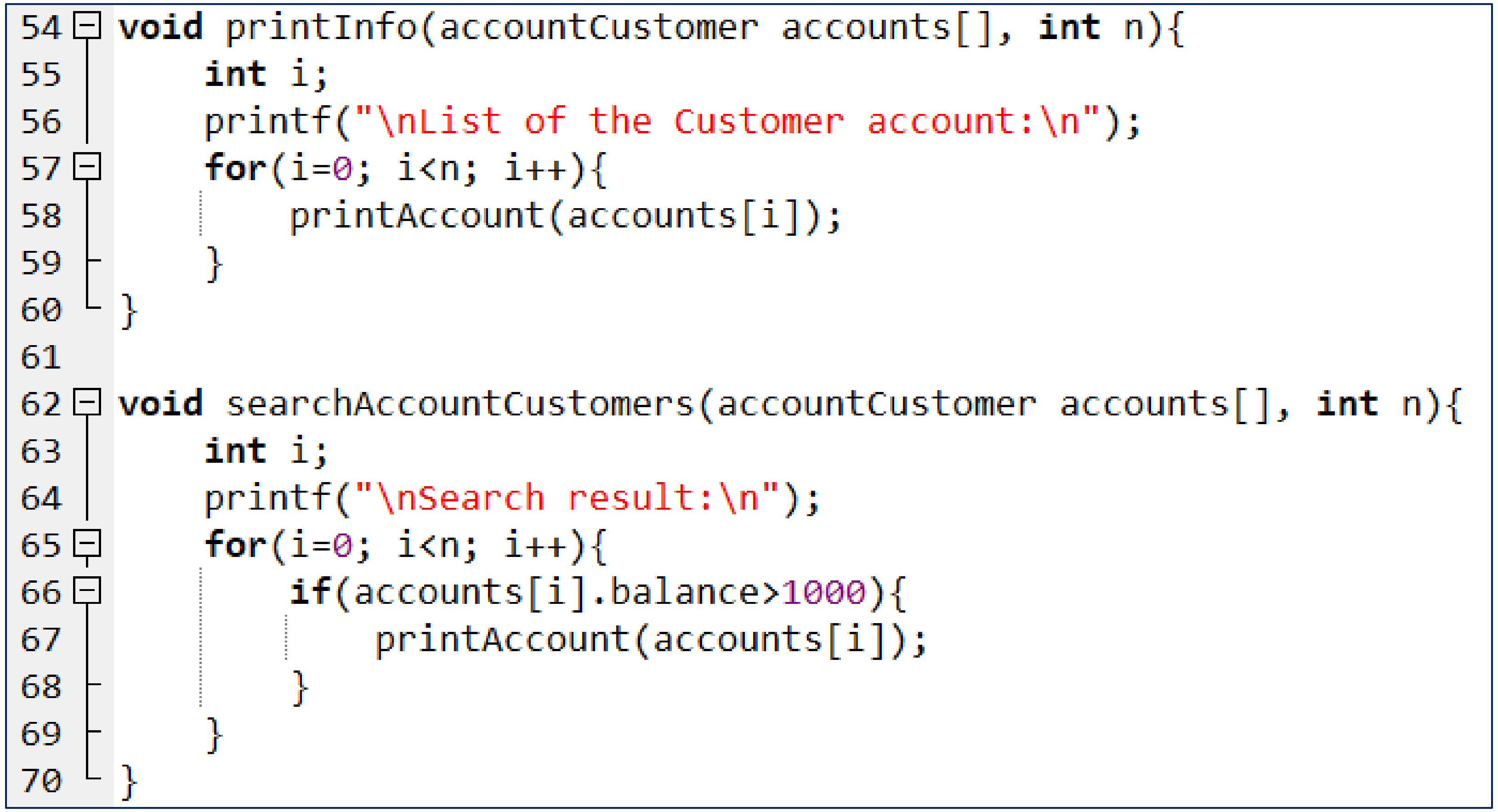
```c
// Prototypes:
void inputInfo(accountCustomer accounts[], int n);
void printAccount(accountCustomer acc);
void printInfo(accountCustomer accounts[], int n);
void searchAccountCustomers(accountCustomer accounts[], int n);

int main(){
    accountCustomer accountCustomers[100];
    int n;

    // Input n
    printf("Input number of the Customer account: ");
    scanf("%d", &n);
    inputInfo(accountCustomers, n);
    printInfo(accountCustomers, n);
    searchAccountCustomers(accountCustomers, n);
    system("pause");
    return 0;
}
```

Case Study: Code design

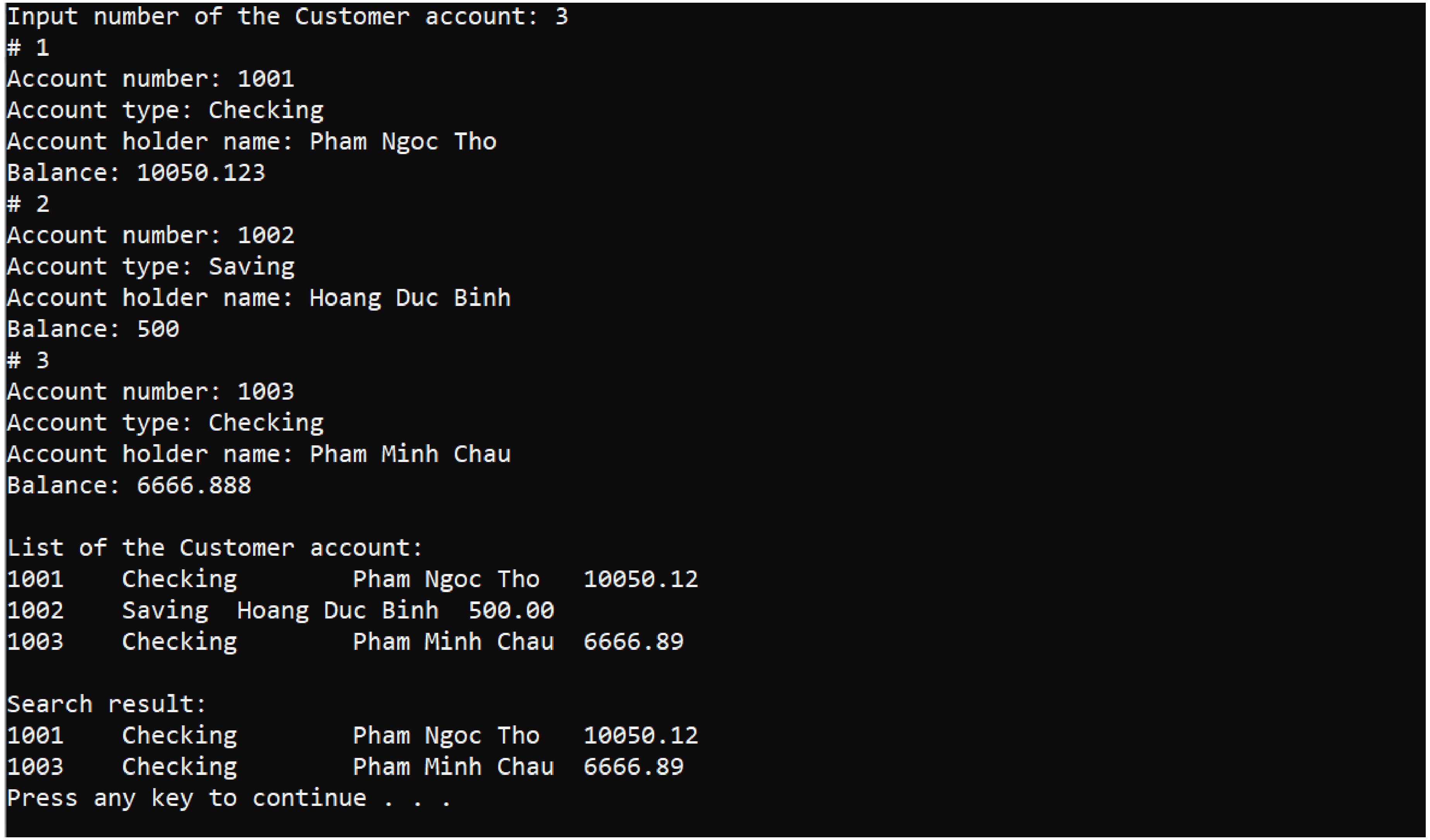


```c
void inputInfo(accountCustomer accounts[], int n){
    int i;
    for(i=0; i<n; i++){
        printf("# %d\n", i+1);
        printf("Account number: "); scanf("%d", &accounts[i].accountNumber);
        clear();
        printf("Account type: "); scanf("%[^\n]", &accounts[i].accountType);
        clear();
        printf("Account holder name: "); scanf("%[^\n]", &accounts[i].accountHolderName);
        printf("Balance: "); scanf("%lf", &accounts[i].balance);
        clear();
    }
}

void printAccount(accountCustomer acc){
    printf("%d\t%s\t%s\t%.2lf\n", acc.accountNumber, acc.accountType, acc.accountHolderName, acc.balance);
}
```

Case Study: Code design


```c
void printInfo(accountCustomer accounts[], int n){
    int i;
    printf("\nList of the Customer account:\n");
    for(i=0; i<n; i++){
        printAccount(accounts[i]);
    }
}

void searchAccountCustomers(accountCustomer accounts[], int n){
    int i;
    printf("\nSearch result:\n");
    for(i=0; i<n; i++){
        if(accounts[i].balance>1000){
            printAccount(accounts[i]);
        }
    }
}
```

Case Study: Compile & Run



```
Input number of the Customer account: 3
# 1
Account number: 1001
Account type: Checking
Account holder name: Pham Ngoc Tho
Balance: 10050.123
# 2
Account number: 1002
Account type: Saving
Account holder name: Hoang Duc Binh
Balance: 500
# 3
Account number: 1003
Account type: Checking
Account holder name: Pham Minh Chau
Balance: 6666.888

List of the Customer account:
1001    Checking        Pham Ngoc Tho   10050.12
1002    Saving  Hoang Duc Binh  500.00
1003    Checking        Pham Minh Chau  6666.89

Search result:
1001    Checking        Pham Ngoc Tho   10050.12
1003    Checking        Pham Minh Chau  6666.89
Press any key to continue . . .
```

Summary

- Understanding what a Structure type is?
- When to use a Structure type in a program
- How to define a Structure type and use a Structure in a program
- Know how to work with the components of a Structure