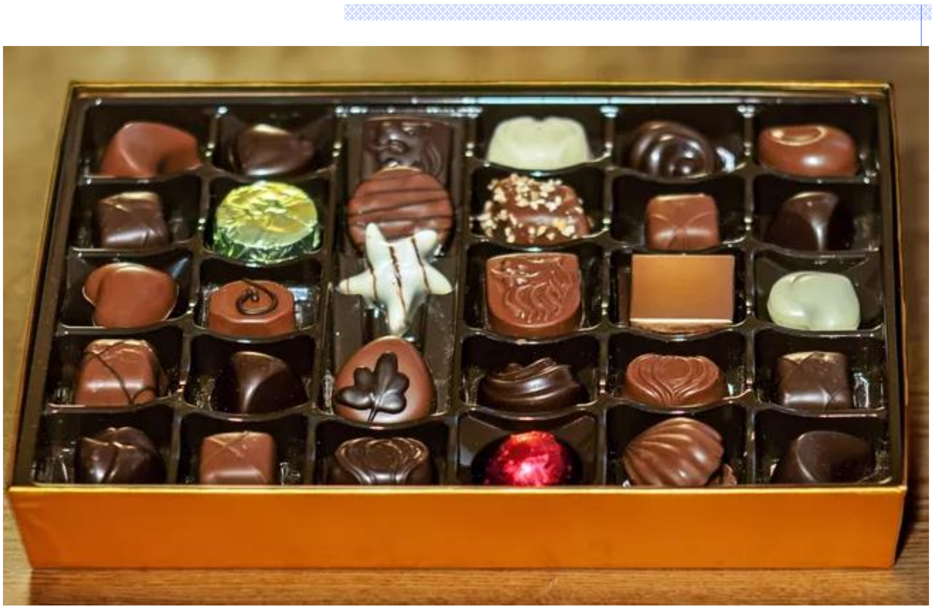
BIRZEIT UNIVERSITY

# Chapter7: Arrays

Computer Science Department



## Arrays - Introduction

- Until now: simple data types use a single memory cell to store a variable.
- Sometimes, we need to *group data items* together in main memory than to allocate an individual memory cell for each variable.
- Example: A program that processes exam scores for a class.
- Here, it would be easier to write if all the scores were stored in one area of memory and were able to be accessed as a group

## Arrays

- An array is a data structure that contains a number of values, or else elements, of the same type
- Each element can be accessed by its position within the array
- An array may have any number of dimensions, but we'll focus on the simplest and most usual kinds
  - the one-dimensional arrays and
  - the two-dimensional arrays
- To introduce you in arrays, we'll show you how to use arrays of integers and floating point numbers
- We'll discuss other types of arrays, as well as their close relationship to pointers in later chapters

## Declaring one-dimensional Array

- To declare an one-dimensional array, you must specify its name (array_name), the number of its elements (number_of_elements) and its data type (data_type), like below:

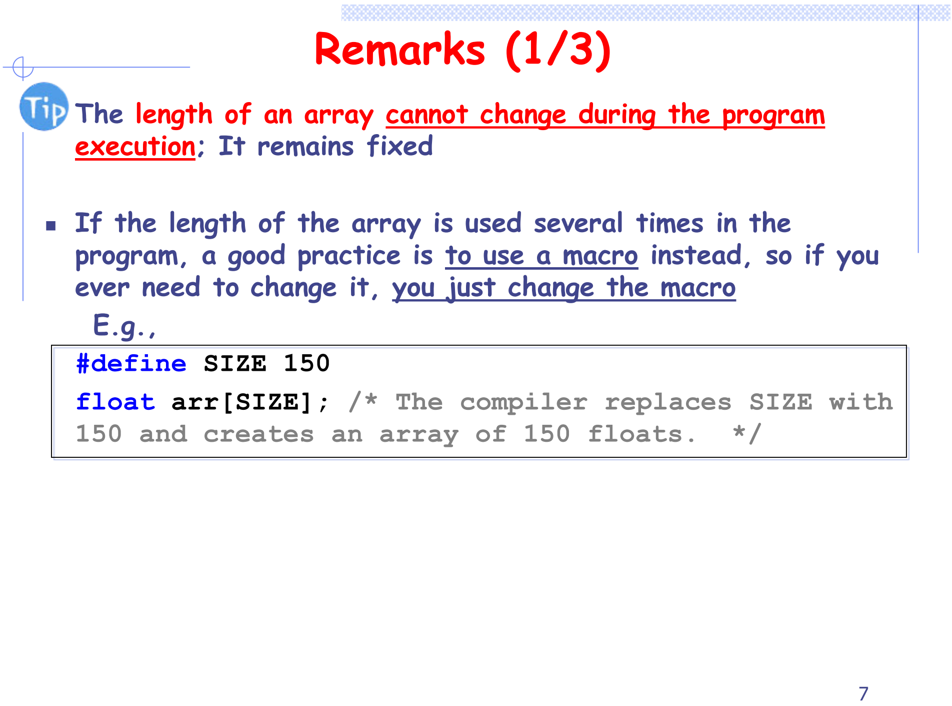
```
data_type array_name[number_of_elements];
```

### Remarks

- The number_of_elements (also called the length of the array) is specified by an integer constant expression greater than 0 enclosed in brackets [ ]
- All the elements are the same type and
- An array may be of any type (e.g., int, float, char,...)

## Array Declaration Examples

```
int a[10];
/*array a, with 10 elements of type int */

double arr[5];
/*array arr, with 5 elements of type double */

float x[2000];
/*array x, with 2000 elements of type float */
```



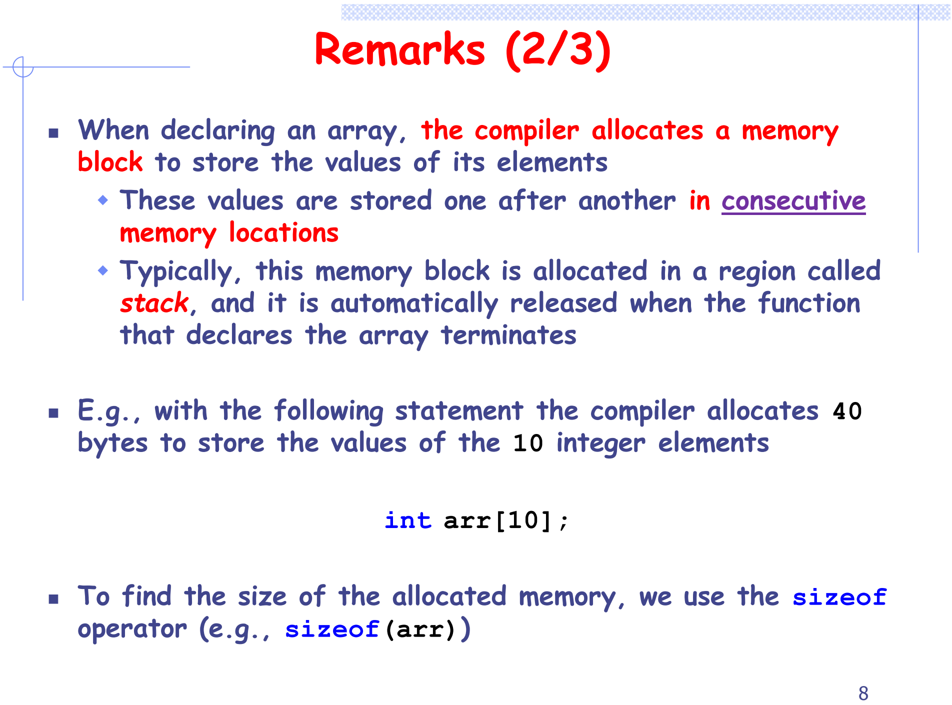
# Remarks (1/3)

The length of an array cannot change during the program execution; It remains fixed

- If the length of the array is used several times in the program, a good practice is to use a macro instead, so if you ever need to change it, you just change the macro

  E.g.,

```
#define SIZE 150
float arr[SIZE]; /* The compiler replaces SIZE with
150 and creates an array of 150 floats.  */
```



# Remarks (2/3)

- When declaring an array, the compiler allocates a memory block to store the values of its elements
  - These values are stored one after another in consecutive memory locations
  - Typically, this memory block is allocated in a region called *stack*, and it is automatically released when the function that declares the array terminates

- E.g., with the following statement the compiler allocates 40 bytes to store the values of the 10 integer elements

```
int arr[10];
```

- To find the size of the allocated memory, we use the sizeof operator (e.g., sizeof(arr))

## Remarks (3/3)

Tip The maximum memory size that can be allocated for an array depends on the available memory in the stack

E.g., the following program may not run in your computer, unless the available stack size is large enough to hold the values

```
#include <stdio.h>
int main(void)
{
        double arr[300000];
        return 0;
}
```

Tip When memory is scarce, don't declare an array with more length than needed, in order to avoid waste of memory

## Accessing Array Elements

- To access an array element, we write the array's name followed by the element's index enclosed in brackets `[]`
- The index specifies the position of the element within the array and it can be an integer constant, variable or expression
- In an array of `n` elements, the first one is stored in position `[0]`, the second one in position `[1]` and the last one in `[n-1]`
- E.g., the statement:

```
float grd[1000];
```

declares the array `grd` with `1000` elements of type `float`, named `grd[0]`, `grd[1]`, ... `grd[999]`

# Remarks (1/2)

- An array element can be used in the same way as an ordinary variable, e.g.,

```
int i, j, a[10], b[10];

a[0] = 2; /* The value of the first element becomes 2. */
a[9] = a[0]; /* The value of the last element becomes equal to the
value of the first element, that is, 2. */
i = j = a[0]+2; /* The values of i and j become equal to 4. */
a[i+j] = 300; /* Since i+j = 4+4 = 8, the value of the ninth element
becomes 300. */
b[i+1] = a[j]; /* The value of the sixth element of array b becomes
equal to the value of the fifth element of array a.*/
```

- A common error occurs when we want to copy one array into another

E.g., it looks pretty natural to write `b = a;`
However, this plausible assignment is illegal

# Example

- The following program declares an array of 5 integers, assigns the values 10, 20, 30, 40, and 50 to its elements and displays those greater than 20

```
#include <stdio.h>
int main(void)
{
      int i, arr[5];

      arr[0] = 10;
      arr[1] = 20;
      arr[2] = 30;
      arr[3] = 40;
      arr[4] = 50;
      for(i = 0; i < 5; i++)
      { /* The braces are not necessary; we use them to make the code
clearer. */
            if(arr[i] > 20)
                  printf("%d\n", arr[i]);
      }
      return 0;
}
```

# Remarks

- C does not check if the index is out of the array bounds. It is the programmer's responsibility to assure that this won't happen. If it does, the behavior of the program is unpredictable.
- E.g., consider the next program
- Since `arr` contains three elements, 0 to 2
- In the last iteration (`i=3`) the statement `arr[3] = 100;` assigns a value to a non-existing array element

```c
#include <stdio.h>
int main(void)
{
        int i, j = 20, arr[3];

        for(i = 0; i < 4; i++)
            arr[i] = 100;

        printf("%d\n", j);
        return 0;
}
```

# Array Initialization (1/3)

- Like ordinary variables, an array can be initialized when it is declared, while uninitialized elements get the arbitrary values of their memory locations, just like an uninitialized variable
  1. In the most common form, the array initializer is a list of values enclosed in braces `{}` and separated by commas `(,)`
     - The list is allowed to end with a comma
     - The values must be constant expressions, variables are not allowed

     For example, with the declaration:

```c
int arr[4] = {10, 20, 30, 40};
```

the value of `arr[0]` is initialized to 10

the value of `arr[1]` is initialized to 20

the value of `arr[2]` is initialized to 30

and the value of the last element `arr[3]` is initialized to 40

# Array Initialization (2/3)

2. If the initialization list is shorter than the number of the elements, the remaining elements are set to 0.
   - It is illegal to be either empty or longer

   For example, with the declaration:

```
int arr[10] = {10, 20};
```

the value of arr[0] is initialized to 10
the value of arr[1] is initialized to 20
and the values of all the rest elements
(i.e., arr[2], arr[3], ... , arr[9]) are initialized to 0

# Array Initialization (3/3)

3. If the array's length is omitted, the compiler will create an array with length equal to the number of the values in the list

   For example, with the declaration:

```
int arr[] = {10, 20, 30, 40};
```

the compiler creates an array of four integers and assigns the values 10, 20, 30 and 40 to its elements

- Specifically:
  the value of arr[0] is initialized to 10
  the value of arr[1] is initialized to 20
  the value of arr[2] is initialized to 30
  the value of arr[3] is initialized to 40

## Initialization Examples

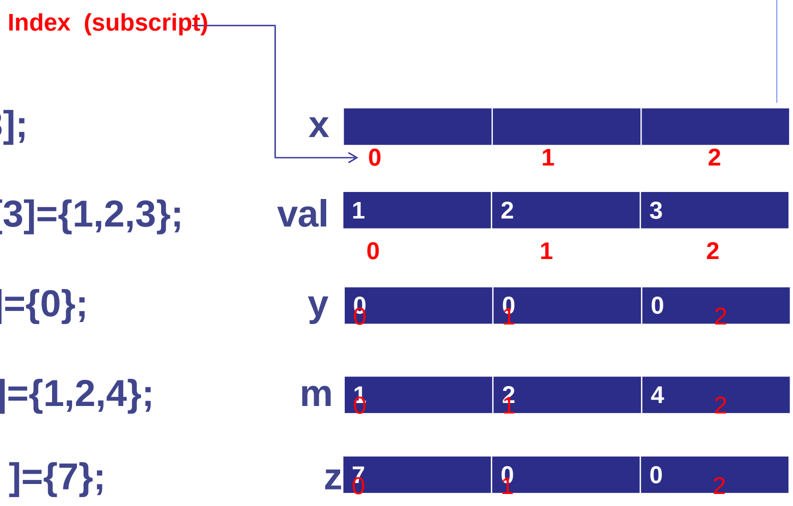
int x [3];

int val[3]={1,2,3};

int y[3]={0};

int m[ ]={1,2,4};

int z[3 ]={7};



## Example Array (1)

```
/*  prints the days for each month */

#include <stdio.h>
#define MONTHS 12

int main(void)
{
int days[MONTHS] = {31,28,31,30,31,30,31,31,30,31,30,31};
int index;
for (index = 0; index < MONTHS; index++)
   printf("Month %d has %2d days.\n", index +1,days[index]);
return 0;
}
```

```
Month 1 has 31 days.
Month 2 has 28 days.
Month 3 has 31 days.
Month 4 has 30 days.
Month 5 has 31 days.
Month 6 has 30 days.
Month 7 has 31 days.
Month 8 has 31 days.
Month 9 has 30 days.
Month 10 has 31 days.
Month 11 has 30 days.
Month 12 has 31 days.
```

# Examples (2)

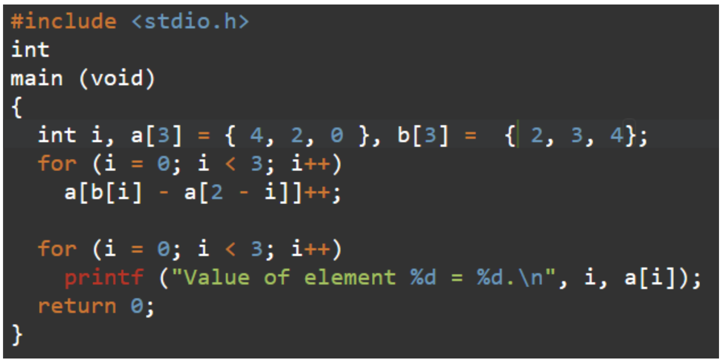
- What is the output of the following program ???

```
#include <stdio.h>
 int main(void)
{
        int i,arr[] = {10,20,30,40,50};

        for(i = 0; i < sizeof(arr)/sizeof(int); i++)
              printf("%d\n",arr[i]);
        return 0;
}
```

```
Output:  10
         20
         30
         40
         50
```

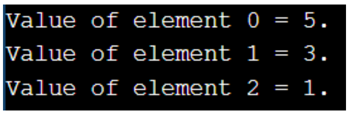
# Examples (3)

- What would be the values of the array a in the following program?



```
#include <stdio.h>
int
main (void)
{
  int i, a[3] = { 4, 2, 0 }, b[3] =  { 2, 3, 4};
  for (i = 0; i < 3; i++)
    a[b[i] - a[2 - i]]++;

  for (i = 0; i < 3; i++)
    printf ("Value of element %d = %d.\n", i, a[i]);
  return 0;
}
```

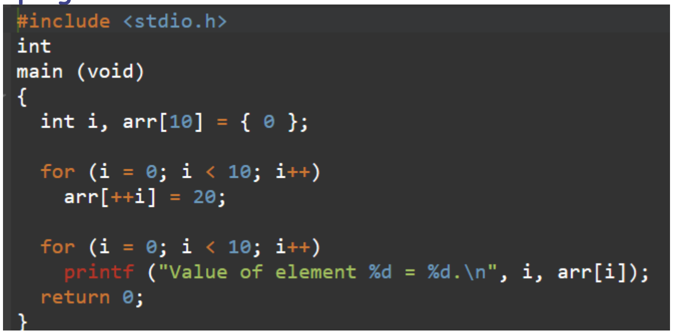
`a[0]` , `a[1]`, and `a[2]` become `5`, `3`, and `1`, respectively



```
Value of element 0 = 5.
Value of element 1 = 3.
Value of element 2 = 1.
```

# Examples (4)

- What would be the values of the array arr in the following program ???

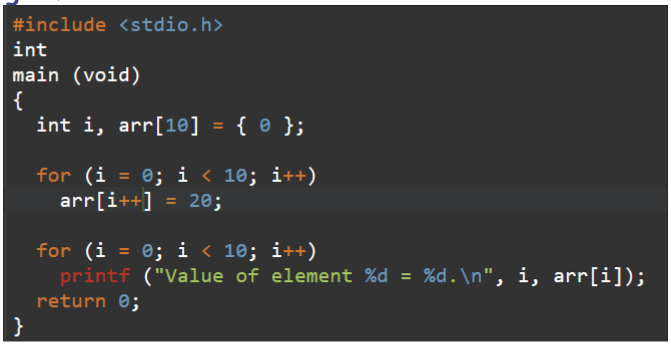


```
#include <stdio.h>
int
main (void)
{
  int i, arr[10] = { 0 };

  for (i = 0; i < 10; i++)
    arr[++i] = 20;

  for (i = 0; i < 10; i++)
    printf ("Value of element %d = %d.\n", i, arr[i]);
  return 0;
}
```

The value of each element with “even index”, i.e.:
arr[0], arr[2], arr[4], arr[6], arr[8] become 0
The value of each element with “odd index”, i.e.:
arr[1], arr[3], arr[5], arr[7], arr[9] become 20

# Examples (5)

- What would be the values of the array arr in the following program ???

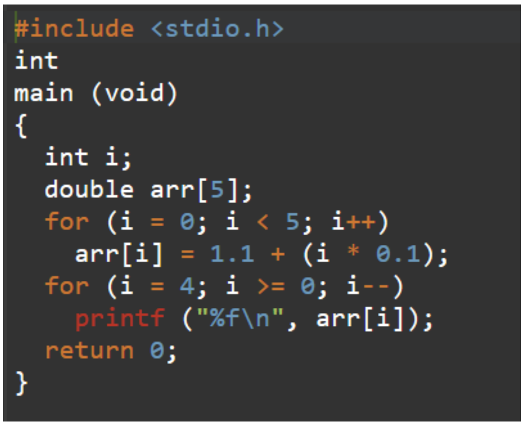


```
#include <stdio.h>
int
main (void)
{
  int i, arr[10] = { 0 };

  for (i = 0; i < 10; i++)
    arr[i++] = 20;

  for (i = 0; i < 10; i++)
    printf ("Value of element %d = %d.\n", i, arr[i]);
  return 0;
}
```

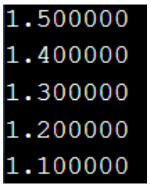
The value of each element with “even index”, i.e.:
arr[0], arr[2], arr[4], arr[6], arr[8] become 20
The value of each element with “odd index”, i.e.:
arr[1], arr[3], arr[5], arr[7], arr[9] become 0

# Examples (6)

- Write a program that declares an array of 5 elements and uses a for loop to assign the values 1.1, 1.2, 1.3, 1.4, and 1.5 to them. Then, the program should display the array's elements in reverse order.



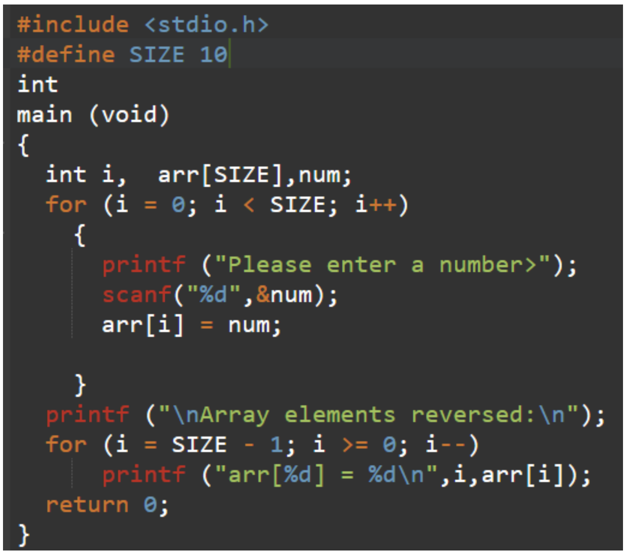
```
#include <stdio.h>
int
main (void)
{
  int i;
  double arr[5];
  for (i = 0; i < 5; i++)
    arr[i] = 1.1 + (i * 0.1);
  for (i = 4; i >= 0; i--)
    printf ("%f\n", arr[i]);
  return 0;
}
```



```
1.500000
1.400000
1.300000
1.200000
1.100000
```

# Examples (7)

- Write a program that reads 10 integers, stores them in an array and displays array elements in reverse order.



```
#include <stdio.h>
#define SIZE 10
int
main (void)
{
  int i,  arr[SIZE],num;
  for (i = 0; i < SIZE; i++)
    {
      printf ("Please enter a number>");
      scanf("%d",&num);
      arr[i] = num;

    }
  printf ("\nArray elements reversed:\n");
  for (i = SIZE - 1; i >= 0; i--)
      printf ("arr[%d] = %d\n",i,arr[i]);
  return 0;
}
```

# Examples (8)

- Write a program that reads 10 integers and stores each one of them in an array, only if it has not already been stored again. It means, that the array elements should have different values between each other.

```
#include <stdio.h>

#define SIZE 10

int main(void)
{
	int i,j,num,found,arr[SIZE];
	i = 0;
	while(i < SIZE)
	{
		printf("Enter number: ");
		scanf("%d",&num);

		found = 0;
		/* The variable i indicates how many integers have
been stored in the array. The for loop checks if the inserted
integer exists in the array. If it does, the variable found
becomes 1 and the for loop terminates. */
```

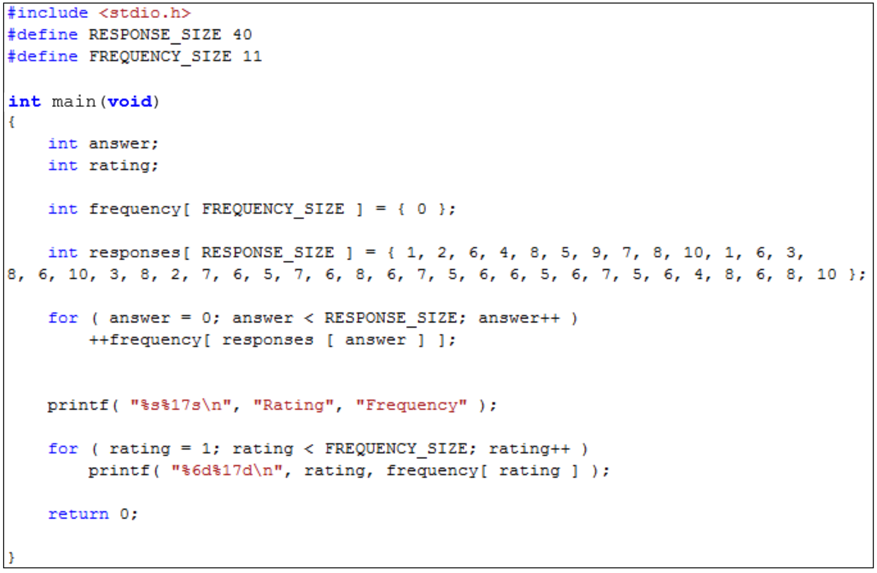
(continued... →)

# Examples (8)

```
		for(j = 0; j < i; j++)
		{
			if(arr[j] == num)
			{
				printf("Error: Number %d exists. ",num);
				found = 1;
				break; /* Termination of for loop. */
			}
		}
		/* If the number does not exist in the array, it is
stored and the index position is increased by one. */
		if(found == 0)
		{
			arr[i] = num;
			i++;
		}
	}
	printf("\nArray elements: ");
	for(i = 0; i < SIZE; i++)
		printf("%d ",arr[i]);

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

# Examples (9)

- What is the output of the following program ???



```c
#include <stdio.h>
#define RESPONSE_SIZE 40
#define FREQUENCY_SIZE 11

int main(void)
{
    int answer;
    int rating;

    int frequency[ FREQUENCY_SIZE ] = { 0 };

    int responses[ RESPONSE_SIZE ] = { 1, 2, 6, 4, 8, 5, 9, 7, 8, 10, 1, 6, 3,
8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7, 5, 6, 6, 5, 6, 7, 5, 6, 4, 8, 6, 8, 10 };

    for ( answer = 0; answer < RESPONSE_SIZE; answer++ )
        ++frequency[ responses [ answer ] ];


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

    for ( rating = 1; rating < FREQUENCY_SIZE; rating++ )
        printf( "%6d%17d\n", rating, frequency[ rating ] );

    return 0;

}
```

# Examples (9)

Output:

```
Rating        Frequency
     1                2
     2                2
     3                2
     4                2
     5                5
     6               11
     7                5
     8                7
     9                1
    10                3
```

# Array Elements as Parameters

- Individual array elements can be used as parameters, *just like other simple variables*. Examples:
- printf( “Last two are %f, %f”, rain[5], rain[6] ) ;
- draw_house( color[i], x[i], y[i], windows[i] ) ;
- scanf( “%lf”,  &rain[0] ) ;
- swap( &rain[i], &rain[i+1] ) ;

# Whole Arrays as Parameters

- Array parameters (entire arrays) work differently:
  - An array is never copied (no call by value)
  - The array name is *always treated as a pointer parameter*
  - The & and * operators are not used
- Programming issue: in C, arrays do not contain information about their size, so the size often needs to be passed as an additional parameter.

## Array Parameter Example

```
#define ARRAY_SIZE  200
```

- In the caller function (for example main):

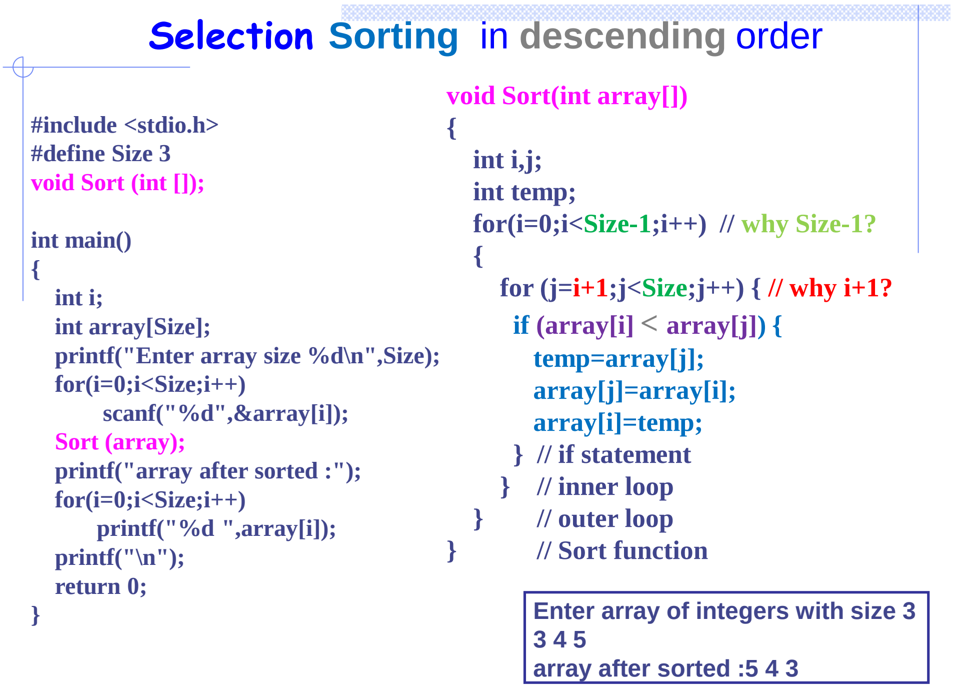
```
int x[ARRAY_SIZE] ;
// here fill the array
int x_avg = average ( x ) ;

// function definition after main function
double average ( int a[ARRAY_SIZE] ) {
 int i, total = 0 ;
 for ( i = 0 ;  i < ARRAY_SIZE ;  i = i + 1 )
    total = total + a[i]  ;
 return ((double) total / ARRAY_SIZE) ;
}
```

0-31

## Fill and print array using function & reverse

```
#define size 5
void fillArray (int[],int);
void printArray (int[],int);
void printArrayInreverse(int[],int);
int main ()    {
   int n[ size];
  fillArray(n,size);
  printArray(n,size);
  printArrayInreverse(n,size);
   return 0;
}

void fillArray (int myArray[],int s) {
   int i;
   for (i=0;i<s;i++)
   {
      printf ("myArray[%d]= ",i);
      scanf("%d",&myArray[i]);
      printf("\n");
   }
}
```

```
void printArray (int myArray[],int s)
{
    int i;
    for (i=0;i<s;i++){
        printf ("myArray[%d]= ",i);
        printf("%d",myArray[i]);
        printf("\n");

    }
}
  void printArrayInreverse(int
  myArray[],int s)
  {   int i;
     for (i=s-1; i>=0; i--){
        printf ("myArray[%d]= ",i);
        printf("%d",myArray[i]);
        printf("\n");
     }
  }
```

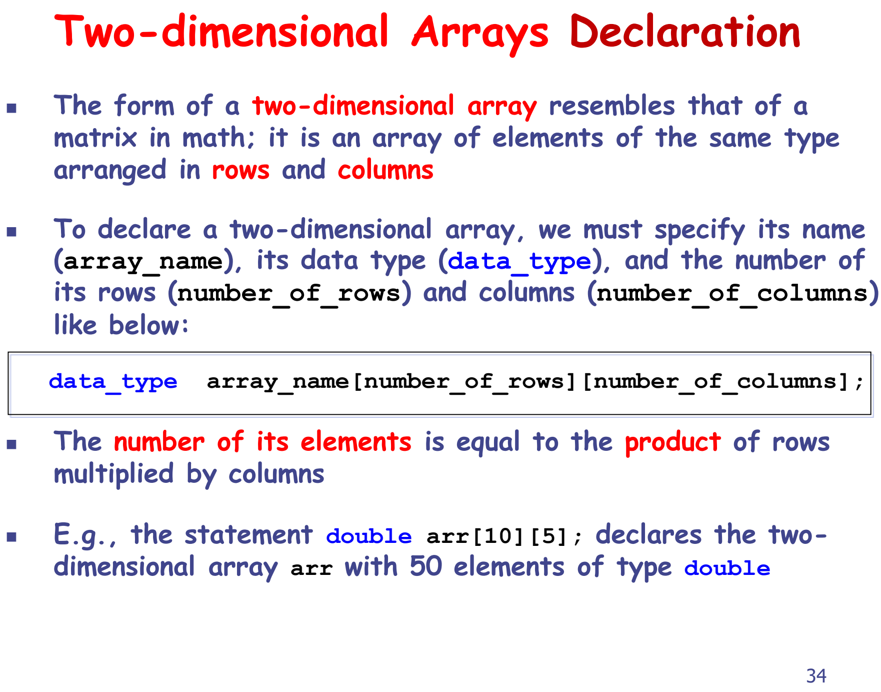


## Selection Sorting in descending order

```
#include <stdio.h>
#define Size 3
void Sort (int []);

int main()
{
   int i;
   int array[Size];
   printf("Enter array size %d\n",Size);
   for(i=0;i<Size;i++)
         scanf("%d",&array[i]);
   Sort (array);
   printf("array after sorted :");
   for(i=0;i<Size;i++)
        printf("%d ",array[i]);
   printf("\n");
   return 0;
}
```

```
void Sort(int array[])
{
   int i,j;
   int temp;
   for(i=0;i<Size-1;i++)  // why Size-1?
   {
      for (j=i+1;j<Size;j++) { // why i+1?
       if (array[i] < array[j]) {
         temp=array[j];
         array[j]=array[i];
         array[i]=temp;
       }  // if statement
      }   // inner loop
   }      // outer loop
}         // Sort function
```

```
Enter array of integers with size 3
3 4 5
array after sorted :5 4 3
```



## Two-dimensional Arrays Declaration

- The form of a two-dimensional array resembles that of a matrix in math; it is an array of elements of the same type arranged in rows and columns
- To declare a two-dimensional array, we must specify its name (`array_name`), its data type (`data_type`), and the number of its rows (`number_of_rows`) and columns (`number_of_columns`), like below:

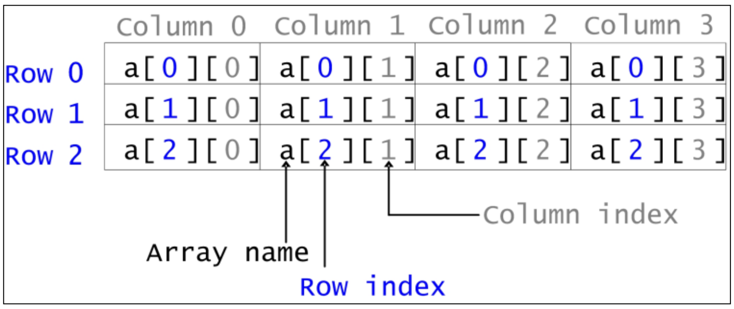
```
data_type  array_name[number_of_rows][number_of_columns];
```

- The number of its elements is equal to the product of rows multiplied by columns
- E.g., the statement `double arr[10][5];` declares the two-dimensional array `arr` with 50 elements of type `double`

## Accessing the Elements of a Two Dimensional Array

- To access an element, we write the name of the array followed by the element’s row index and column index enclosed in double brackets `[][]`
- As with one-dimensional arrays, the indexing of rows and columns starts from `0`
- E.g., the statement: `int a[3][4];`

  declares a two-dimensional array whose elements are the `a[0][0]`, `a[0][1]`, …, `a[2][3]`, as depicted in the figure below



|  | Column 0 | Column 1 | Column 2 | Column 3 |
|---|---|---|---|---|
| Row 0 | a[0][0] | a[0][1] | a[0][2] | a[0][3] |
| Row 1 | a[1][0] | a[1][1] | a[1][2] | a[1][3] |
| Row 2 | a[2][0] | a[2][1] | a[2][2] | a[2][3] |

Column index

Array name

Row index

## Two-dimensional Arrays and Memory

- As with one-dimensional arrays, when a two-dimensional array is declared, the compiler allocates a memory block in the stack to store the values of its elements
- E.g., with the statement:

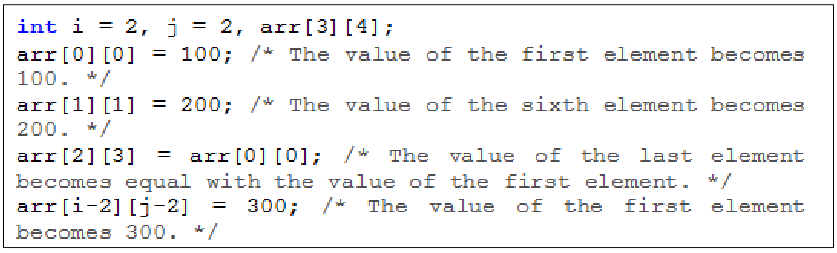
```
int array[10][5];
```

the compiler allocates a block of 200 bytes to store the values of its 50 elements (since each array element requires 4 bytes)

## Accessing two-dimensional Array Elements

- To access an element of a two-dimensional array we must specify its row index and its column index

- E.g.,



```
int i = 2, j = 2, arr[3][4];
arr[0][0] = 100; /* The value of the first element becomes
100. */
arr[1][1] = 200; /* The value of the sixth element becomes
200. */
arr[2][3]  = arr[0][0];  /*  The  value  of  the  last  element
becomes equal with the value of the first element. */
arr[i-2][j-2]  =  300;  /*  The  value  of  the  first  element
becomes 300. */
```

⚠ As with one-dimensional arrays, special care is required not exceed the bounds of any dimension

## Row, Column Indices

|   | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 78 | 83 | 82 |
| 1 | 90 | 88 | 94 |
| 2 | 71 | 73 | 78 |
| 3 | 97 | 96 | 95 |
| 4 | 89 | 93 | 90 |

**Give both the ROW and COLUMN indices to pick out an individual element.**

***The fourth student's third test score is at ROW 3, COLUMN 2***

**int a[5][3];**
**a[3][2];**

# Example

```
int a[2][4];

a[1][0]=9;

a[0][3]=5;

a[0][1]=a[0][3]+ a[1][0];
```

|   | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 |   | 14 |   | 5 |
| 1 | 9 |   |   |   |

## Two-Dimensional Array Initialization (1/5)

- A two-dimensional array can be initialized when declared, just like a one-dimensional array.
  1. The initialization values are assigned in row order, starting from row 0, then row 1, and so on, e.g.,

```
int arr[3][3] =  {{10,20,30},
                   {40,50,60},
                   {70,80,90}};
```

In this example:

the value of `arr[0][0]` is initialized to `10`

the value of `arr[0][1]` is initialized to `20`

the value of `arr[0][2]` is initialized to `30`

the value of `arr[1][0]` is initialized to `40` and so on...

Alternatively, we can omit the inner braces and write:

```
int arr[3][3] = {10,20,30,40,50,60,70,80,90};
```

since, once the compiler fills one row, it continues with the next one

## Two-Dimensional Array Initialization (2/5)

2. When using braces, if the initialization list is shorter than the row's elements, the compiler assigns the value 0 to the remaining elements in that row. If it is larger it is illegal.

   E.g.,

```
int arr[3][3] = {{10,20},
                 {40,50},
                 {70}};
```

   <u>In this example:</u>

   the value of `arr[0][0]` is initialized to 10

   the value of `arr[0][1]` is initialized to 20

   the value of `arr[1][0]` is initialized to 40

   the value of `arr[1][1]` is initialized to 50

   the value of `arr[2][0]` is initialized to 70

   while, the values of `arr[0][2]`, `arr[1][2]`, `arr[2][1]` and `arr[2][2]` are initialized to 0

## Two-Dimensional Array Initialization (3/5)

3. If we omit the initialization of a row, the compiler initializes its elements to 0

   E.g.,

```
int arr[3][3] = {{10,20,30}};
```

   <u>In this example:</u>

   The elements of the second and third row are set to 0

4. If the inner braces {} are omitted and the initialization list is shorter than the number of the array elements, the remaining elements are set to 0

   E.g.,

```
int arr[3][3] = {10,20};
```

   <u>In this example:</u>

   The value of `arr[0][0]` becomes 10, `arr[0][1]` becomes 20 and the remaining elements are all set to 0

## Two-Dimensional Array Initialization (4/5)

5. When a two-dimensional array is declared, the number of columns must be specified

   However, the number of rows is optional

   If it is not specified, the compiler will create a two-dimensional array based the initialization list

   E.g.,

```
int arr[][3] = {10,20,30,40,50,60};
```

   In this example:

   Since the array **arr** has three columns and the initialization values are six, the compiler creates a two-dimensional array of two rows and three columns

   Specifically:

   the value of **arr[0][0]** is initialized to 10

   the value of **arr[0][1]** is initialized to 20

   the value of **arr[0][2]** is initialized to 30

   the value of **arr[1][0]** is initialized to 40 and so on...

## Two-Dimensional Array Initialization (5/5)

6. If the initialization value is the same or it can be easily produced a pair of nested loops is the typical choice

   E.g.,

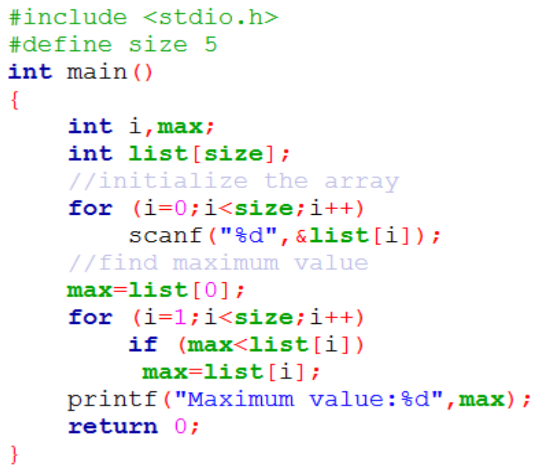
```
#include <stdio.h>
int main(void)
{
        int i, j, arr[50][100];

        for(i = 0; i < 50; i++)
                for(j = 0; j < 100; j++)
                        arr[i][j] = 1;
        return 0;
}
```

## Linear Search

```
#include <stdio.h>
#define size 7
int main() {
 int myArray[size]={29,99,87,34,97,54,66};
   int target;       // input - value searched for
   int location;   //  index of the target
   int found = 0;
   int i=0;
   printf("please enter a target: ");
   scanf("%d",&target);

   while (i<size)   {
         if (target==myArray[i])  {
             location=i;  //update location
             found=1;    //Matching target
             break;
          }   // end if
         i++;
    }   // end while
   if (found==1)
        printf("location is %d\n",location);
     else
         printf("Not found\n");
    return 0;
  }  // main
```

45

## Example: Finding the Maximum

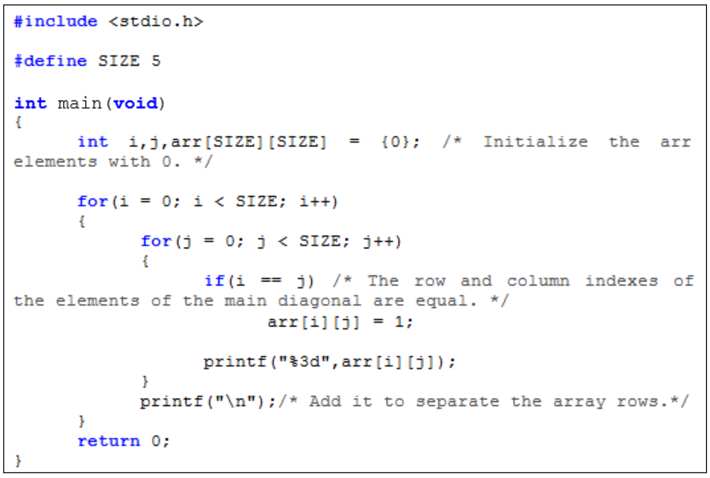


```
#include <stdio.h>
#define size 5
int main()
{
     int i,max;
     int list[size];
     //initialize the array
     for  (i=0;i<size;i++)
          scanf("%d",&list[i]);
     //find maximum value
     max=list[0];
     for  (i=1;i<size;i++)
          if  (max<list[i])
           max=list[i];
     printf("Maximum value:%d",max);
     return  0;
}
```

# Examples (I)

- Write a program that creates an identity 5×5 array and displays its elements as an identity 5×5 matrix in algebra form.

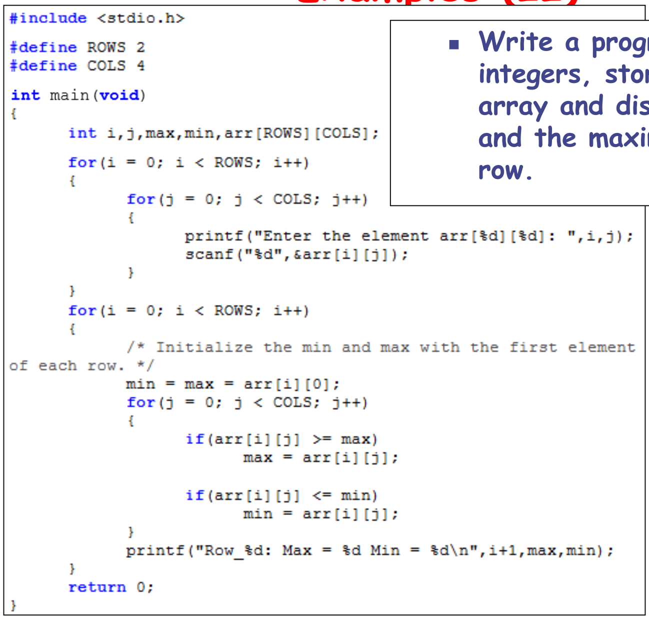
  *Note:* In math, an identity matrix has 1's on the main diagonal's elements and 0's everywhere else.



```
#include <stdio.h>

#define SIZE 5

int main(void)
{
	int  i,j,arr[SIZE][SIZE]  =  {0};  /*  Initialize  the  arr
elements with 0. */

	for(i = 0; i < SIZE; i++)
	{
		for(j = 0; j < SIZE; j++)
		{
			if(i == j)  /* The  row  and  column  indexes  of
the elements of the main diagonal are equal. */
				arr[i][j] = 1;

			printf("%3d",arr[i][j]);
		}
		printf("\n");/* Add it to separate the array rows.*/
	}
	return 0;
}
```

# Examples (II)

- Write a program that reads integers, stores them in an 2×4 array and displays the minimum and the maximum value of each row.



```
#include <stdio.h>

#define ROWS 2
#define COLS 4

int main(void)
{
	int i,j,max,min,arr[ROWS][COLS];

	for(i = 0; i < ROWS; i++)
	{
		for(j = 0; j < COLS; j++)
		{
			printf("Enter the element arr[%d][%d]: ",i,j);
			scanf("%d",&arr[i][j]);
		}
	}
	for(i = 0; i < ROWS; i++)
	{
		/* Initialize the min and max with the first element
of each row. */
		min = max = arr[i][0];
		for(j = 0; j < COLS; j++)
		{
			if(arr[i][j] >= max)
				max = arr[i][j];

			if(arr[i][j] <= min)
				min = arr[i][j];
		}
		printf("Row_%d: Max = %d Min = %d\n",i+1,max,min);
	}
	return 0;
}
```