Chapter 6: Arrays

- In this chapter, you will learn about
  - Introduction to Array
  - Array declaration
  - Array initialization
  - Assigning values to array elements
  - Reading values from array elements
  - Simple Searching
  - Simple Sorting
  - 2 Dimensional arrays
  - Working with matrices

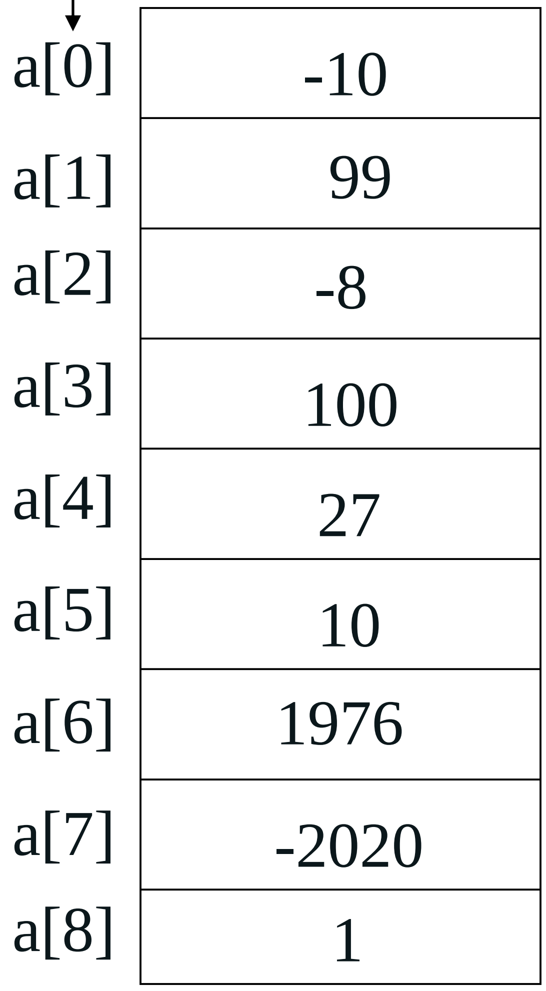
Introduction to Array

- In C, a group of items of the same type can be set up using Array
- An array is a group of consecutive memory locations related by the fact that they all have the same name and the same type.
- The compiler must reserve storage (space) for each element/item of a declared array.
- The size of an array is static (fixed) throughout program execution.
- To refer to a particular location or element in the array, we specify the name of the array and its index (the position number of the particular element in the array).

Let say we have an array called **a.**

Notice that the position starts from 0.

Name of the array



a[0]	-10
a[1]	99
a[2]	-8
a[3]	100
a[4]	27
a[5]	10
a[6]	1976
a[7]	-2020
a[8]	1

The position number within the square brackets is formally called a ***subscript***. A subscript can be an integer or an integer expression. For example if x = 1 and y = 2, then a[x+y] is equal to a[3].

Array Declaration

- Array declaration is made by specifying the **data type**, it's **name** and the **number of space (size)** so that the computer may reserve the appropriate amount of memory.
- General syntax:

  data_type array_name[size];

- Examples:
  - `int my_array[100];`
  - `char name[20];`
  - `double bigval[5*200];`
  - `int a[27], b[10], c[76];`

Array Initialization

- There are 2 ways to initialize an array: during compilation and during execution.
- During compilation:
  - `int arr[ ] = {1, 2, 3, 4, 5};`
    - Unsized array : We can define how many elements that we want since the array size is not given.
  - `int arr[3] = {90, 21, 22};`
    - We can define only 3 elements since the array size is already given.
  - `int arr[5] = {2,4};`
    - Initialize the first two elements to the value of 2 and 4 respectively, while the other elements are initialized to zero.
  - `int arr[5] = {0};`
    - Initialize all array elements to zero.

Array Initialization cont…

- During execution:
  - Using loop to initialize all elements to zero

```
int arr[3], index;
for (index = 0; index < 3; index++)
          arr[index] = 0;
```

  - Using loop and asking the user to specify the value for each element.

```
int arr[3], index;
for (index = 0; index < 3; index++)
{
          printf ("arr[%d]:",index);
          scanf("%d",&arr[index]);
}
```

Assigning value to array element

- We can assign a value to a specific array element by using its index number.
- Example: let’s say we have an array that represents the number of inhabitants in 5 apartment units.

```
int apartment[5]={3,2,6,4,5};
```

- The above initialization indicates that there are 3 people living in apartment 0, 2 people living in apartment 1 and so on.
- Let say that we have a new born in apartment 3, so we need to change the number of inhabitants living in apartment three.

```
apartment[3] = apartment[3] + 1;
```

- Now, we have the following values in our array:

	[0]	[1]	[2]	[3]	[4]
apartment	**3**	**2**	**6**	**5**	**5**

Reading values from array elements

- We can read a value from a specific array element by referring to the index.
- For example, let’s say we want to know how many people leaving in apartment 3, we could simple do this:

```
int apartment[5] = {3,2,6,4,5};
int no_of_people;
no_of_people = apartment[3];
printf(“Apartment 3 has %d people”,no_of_people);
```

- The above C code will produce the following output:

```
Apartment 3 has 4 people
Press any key to continue
```

Remember that array’s index starts at 0!

Example 1: finding total inhabitants

```c
#include <stdio.h>
#define size 5

int main(void)
{
   int apartment[size] = {3,2,6,4,5};
   int index, total = 0;

   for (index = 0; index < size; index++)
   {
       total = total + apartment[index];
   }

   printf("There are total of %d inhabitants",total);
   return(0);
}
```

```
There are total of 20 inhabitants
Press any key to continue
```

Example 2: list down number of inhabitants in each apartment

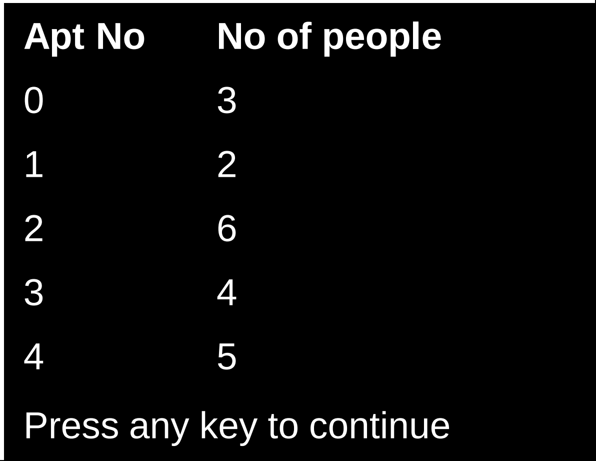
```
#include <stdio.h>

int main(void)
{
   int apartment[5] = {3,2,6,4,5};
   int index, total = 0;

   printf("%s\t%s\n","Apt No", "No of people");

   for (index = 0; index < 5; index++)
   {
       printf("%d\t%d\n",index, apartment[index]);
   }

   return(0);
}
```



```
Apt No    No of people
0         3
1         2
2         6
3         4
4         5
Press any key to continue
```

Exercise 1

- Modify from example 1, write a program that prints the average number of inhabitants.

Exercise 2

- Given the following ordered list of floating point numbers, write a program that determines its median and standard deviation.

- Standard deviation formula, $s = \sqrt{\dfrac{\sum (x - \bar{x})^2}{n - 1}}$

{1.0, 6.0, 18.0, 39.0, 86.0}

Data Searching

- Searching is the process of determining whether an array contains a value that matches a certain *key value/search key*.
- The process of finding a particular element of an array is called *searching*.
- There are more than one algorithms that can be used to do a search.
- The most commonly used searching techniques are linear search and binary search.
- Here, we will discuss how to do searching by using linear search on an array.

Linear Search

- *Search key* is a data element of the same type as the list elements.
  - If search key == list element value, the search is said to be successful.
  - Otherwise, it is unsuccessful.
- Linear search is a simple searching algorithm where:
  - data are stored in an array
  - a search key is compared with each elements in the array starting from the first element.

Example: Linear Search

```
#include <stdio.h>

int main(void)
{
   int list[ ] = {34, 53, 21, 23, 4};
   int i, search_key, found = 0;

   printf(“Enter the number that you want to find: ”);
   scanf(“%d”, &search_key);

   for (i = 0; i < 5; i++)
   {
     if  (list[i] = = search_key)
     {
       found = 1;
       printf(“%d is found at index %d\n”, search_key, i);
     }
   }

   if (found = = 0)
     printf(“%d cannot be found in the list\n”,search_key);

   return(0);
}
```

```
Enter the number that you want to find: 53
53 is found at index 1
Press any key to continue
```

Sorting

- Sorting is the process of placing data into a particular order such as ascending or descending.
- The following example shows the C code for sorting unsorted list to a list sorted in ascending order.
- Explanation for the working program and the concept behind it will be done during lecture hour… (so please attend the class!!!!).

Example: Simple Sort

```
void main(void)
{
  int pivot, checker, temp, list[]={3, 2, 5, 4, 1};
  for (pivot = 0; pivot < (size - 1); pivot++)
  {
    for (checker = (pivot + 1); checker < size; checker++)
    {
      if (list[checker] < list[pivot])
      {
        /* swap the elements */
        temp = list[pivot] ;
        list[pivot] = list[checker];
        list[checker] = temp;
      }
    }
  }

  for(int i=0; i<5; i++)
  printf("%d ", list[i]);
}
```

Two-Dimensional Array

- It is possible to create an array which has more than one dimension.
- For example:
  - 2D array: `int array[4][2];`
- Graphical representation of a 2D array:

```
int myarray[4][2] = {1,2,3,4,5,6,7,8};
```

	Col 1	Col2
Row 1	1	2
Row 2	3	4
Row 3	5	6
Row 4	7	8

This array has 4 rows and 2 columns.

Two-Dimensional Array cont…

- Variable initialization can also be done this way:

```
int myarray[4][2] = {{1,2},{3,4},{5,6},{7,8}};
```

- This method is less confusing since we can see the rows and columns division more clearly.
- To initialize a 2D array during execution, we need to use a nested for loop:

```
for (row = 0; row < 4; row++)
{
    for (column = 0; column < 2; column++)
    {
        myarray[row][column] = 0;
    }
}
```

- Although it is possible to create a multi-dimensional array, arrays above 2-dimensions are rarely used.

Matrices

- A matrix is a set of numbers arranged in a rectangular grid with rows and columns.
- Below is an example of a matrix with four rows and three columns, specified as 4 x 3 matrix

$$\mathbf{A} = \begin{bmatrix} -1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & -2 & 3 \\ 0 & 2 & 1 \end{bmatrix}$$

- Be careful when translating equations in matrix notation into C statements because of the difference in subscripting, i.e. in matrix, row and column numbers begin with 1.

Example

- Write program that perform dot product from the following vectors.

$$\boldsymbol{A} = \begin{bmatrix} 4 & -1 & 3 \end{bmatrix}$$

$$\boldsymbol{B} = \begin{bmatrix} -2 & 5 & 2 \end{bmatrix}$$

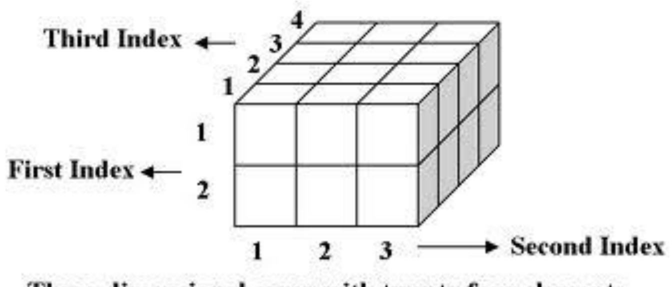
```
int main (void){
        int k, product=0, A[] = {4, -1, 3}, B[] = {-2, 5, 2};
        for (k=0; k<size-1;k++)
                product += A[k]*B[k];
        printf("%d", product);
        return 0;
}
```

Exercise

- Write a program that can perform addition and subtraction of two same size matrices.

Beyond Two-Dimensional Array

- Declaration of 3 dimensional array
  - 3D array: `int array[2][3][4];`



Three-dimensional array with twenty four elements

- Four-dimensional array?

Summary

- In this chapter, we have looked at:
  - Array declaration and initialization
  - Reading and writing from/to array elements
  - Passing array to function
  - Simple search
  - Simple sort
  - 2 dimensional array
  - Working with matrices

Exercise

Assume the following array declaration

```
float number[5] = {2.3, 4.2, 5.0, 7.9, 6.2};
```

What will be the output of the following statement?

```
a) printf(“%f”, number[2+2] );
b) printf(“%f”, number[2]+2 );
c) printf(“%f”, number[1*2] );
d) printf(“%f”, number[1]*2 );
```

Exercise

Assume the following array declaration

```
int result[5] = {56, 69, 89};
int i = 2;
```

What will be the output of the following statement?

```
a) printf("%d", result[1]);
b) printf("%d", result[4]);
c) printf("%d", result[0] + result[1]);
d) printf("%d %d", i, result[i]);
```

Exercise

Assume the following array declaration

```
int result[3*2];
```

a) Write C statements that would read the values for the array element from the user.
b) Write C statements that would list down all the values in the array.
c) Write C statements that would sum up all the values in the array.