Arrays

Basic Concept

- Many applications require multiple data items that have common characteristics.
  - In mathematics, we often express such groups of data items in indexed form:
    - $x_1, x_2, x_3, \ldots, x_n$
- Why are arrays essential for some applications?
  - Take an example.
  - Finding the **minimum** of a set of numbers.

3 numbers

```
if   ((a <= b) && (a <= c))
   min = a;
else
   if   (b <= c)
        min = b;
   else
        min = c;
```

4 numbers

```
if   ((a <= b) && (a <= c) && (a <= d))
   min = a;
else
   if   ((b <= c) && (b <= d))
        min = b;
   else
       if  (c <= d)
           min = c;
       else
           min = d;
```

The Problem

- Suppose we have 10 numbers to handle.
- Or 20.
- Or 100.

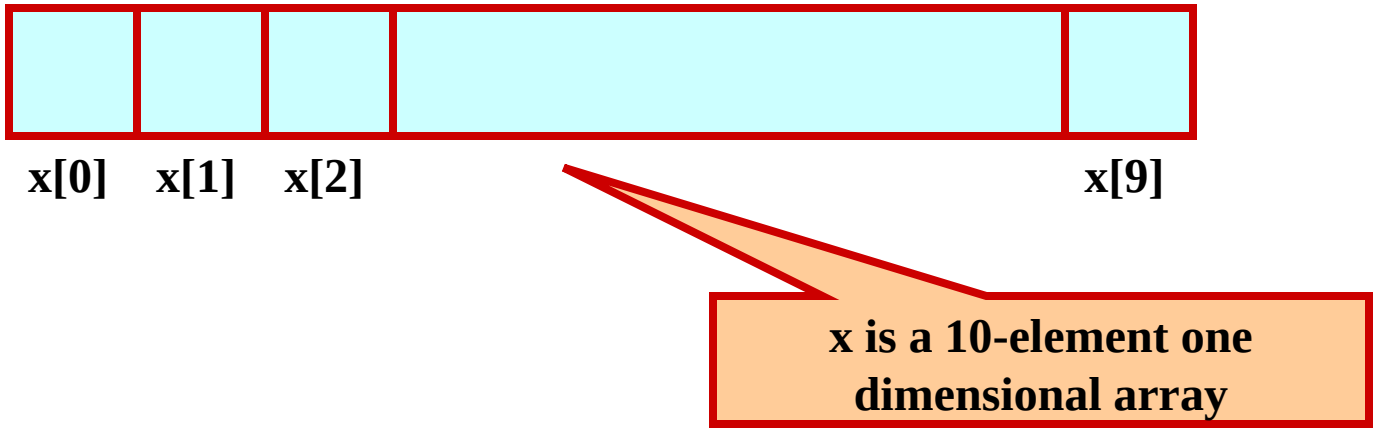
- How to tackle this problem?
- Solution:
  - Use arrays.

Using Arrays

- All the data items constituting the group share the same name.

```
int  x[10];
```

- Individual elements are accessed by specifying the index.



Declaring Arrays

- Like variables, the arrays that are used in a program must be declared before they are used.
- General syntax:

  type array-name [size];

  – type specifies the type of element that will be contained in the array (int, float, char, etc.)

  – size is an integer constant which indicates the maximum number of elements that can be stored inside the array.

int marks[5];

- marks is an array containing a maximum of 5 integers.

- Examples:

```
int  x[10];
char  line[80];
float  points[150];
char  name[35];
```

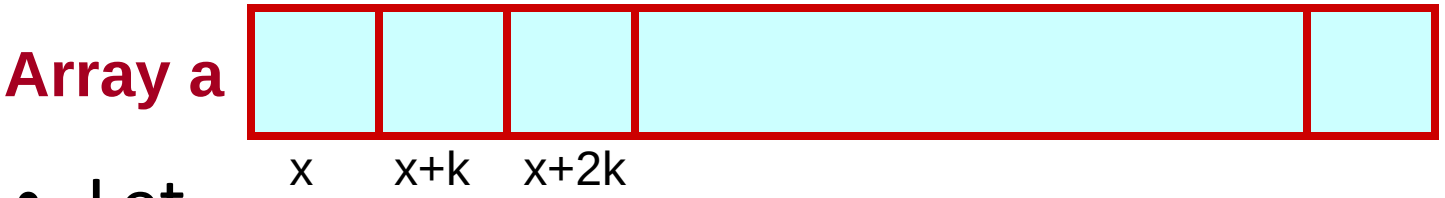
- If we are not sure of the exact size of the array, we can define an array of a large size.

```
int   marks[50];
```

  though in a particular run we may only be using, say, 10 elements.

How an array is stored in memory?

- Starting from a given memory location, the successive array elements are allocated space in consecutive memory locations.



- Let

  x: starting address of the array in memory

  k: number of bytes allocated per array element

  – Element a[i] :: allocated memory location at address x + i*k
    - First array index assumed to start at zero.

Accessing Array Elements

- A particular element of the array can be accessed by specifying two things:
  - Name of the array.
  - Index (relative position) of the element in the array.
- In C, the index of an array starts from zero.
- Example:
  - An array is defined as `int x[10];`
  - The first element of the array x can be accessed as x[0], fourth element as x[3], tenth element as x[9], etc.

Contd.

- The array index must evaluate to an integer between 0 and n-1 where n is the number of elements in the array.

```
a[x+2] = 25;
b[3*x-y] = a[10-x] + 5;
```

A Warning

- In C, while accessing array elements, array bounds are not checked.
- Example:

```
int   marks[5];
:
:
marks[8] = 75;
```

  - The above assignment would not necessarily cause an error.
  - Rather, it may result in unpredictable program results.

Initialization of Arrays

- General form:

  type array_name[size] = { list of values };

- Examples:

  int marks[5] = {72, 83, 65, 80, 76};

  char name[4] = {'A', 'm', 'i', 't'};

- Some special cases:
  - If the number of values in the list is less than the number of elements, the remaining elements are automatically set to zero.

    float total[5] = {24.2, -12.5, 35.1};

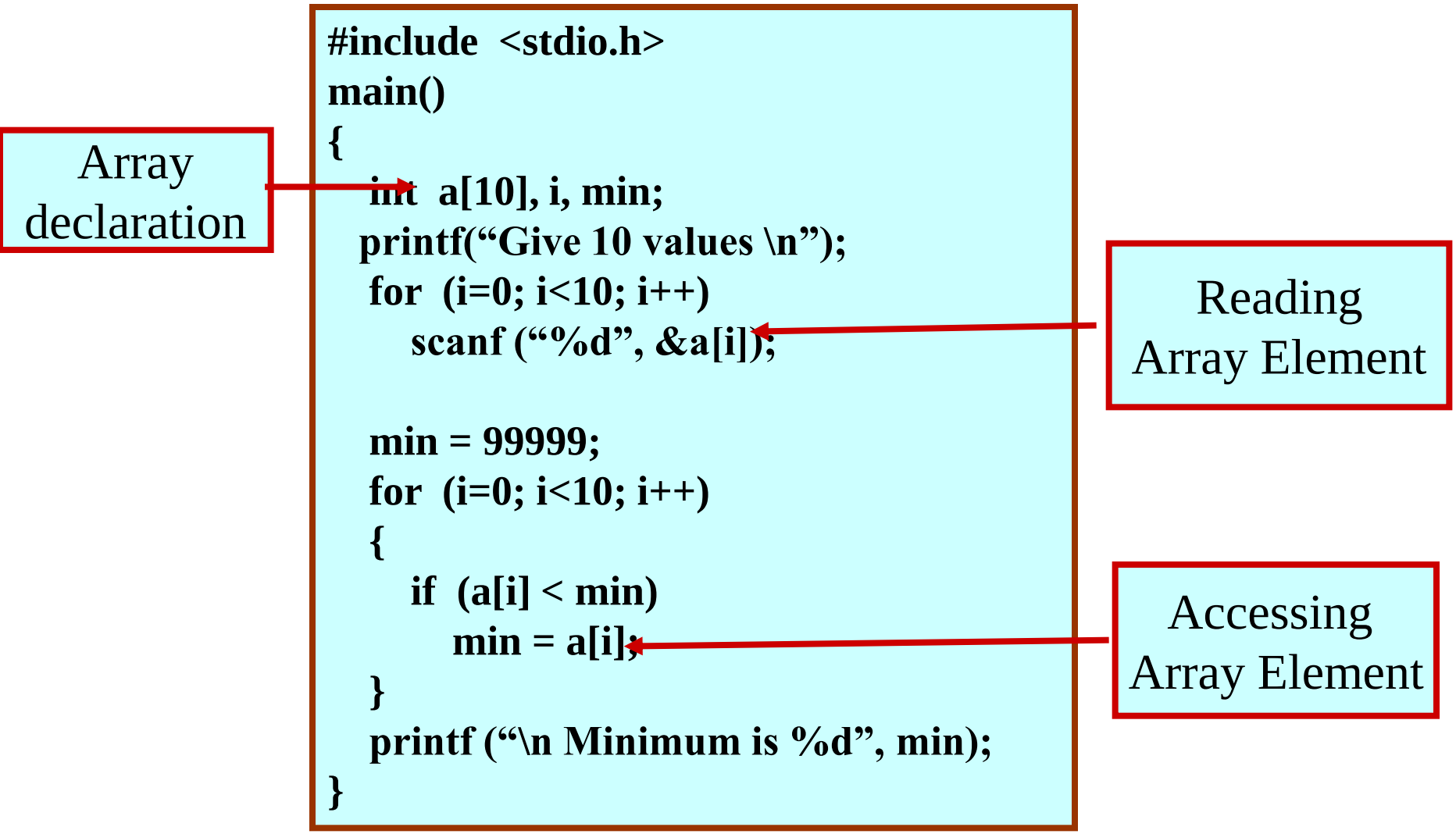
    ➔ total[0]=24.2, total[1]=-12.5, total[2]=35.1, total[3]=0,
    total[4]=0

Contd.

– The size may be omitted. In such cases the compiler automatically allocates enough space for all initialized elements.

```
int   flag[] = {1, 1, 1, 0};
char  name[] = {‘A’, ‘m’, ‘i’, ‘t’};
```

Example 1: Find the minimum of a set of 10 numbers



Alternate Version 1

Change only one line to change the problem size

```
#include  <stdio.h>
#define   size   10

main()
{
   int  a[size], i, min;
   printf("Give 10 values \n");
   for  (i=0; i<size; i++)
      scanf ("%d", &a[i]);

   min = 99999;
   for  (i=0; i<size; i++)
   {
      if  (a[i] < min)
         min = a[i];
   }
   printf ("\n Minimum is %d", min);
}
```

Alternate Version 2

Define an array of large size and use only the required number of elements

```
#include  <stdio.h>

main()
{
   int  a[100], i, min, n;

   printf("Give number of elements (n) \n");
   scanf ("%d", &n);  /* Number of elements */

  printf("Input all n integers \n");
   for  (i=0; i<n; i++)
      scanf ("%d", &a[i]);

   min = 99999;
   for  (i=0; i<n; i++)
   {
      if  (a[i] < min)
         min = a[i];
   }
   printf ("\n Minimum is %d", min);
}
```

Example 2: Computing gpa

Handling two arrays at the same time

```
#include  <stdio.h>
#define  nsub  6

main()
{
   int  grade_pt[nsub], cred[nsub], i,
        gp_sum=0, cred_sum=0, gpa;

printf("Input gr. points and credits for six subjects \n");
   for  (i=0; i<nsub; i++)
      scanf ("%d %d", &grade_pt[i], &cred[i]);

   for  (i=0; i<nsub; i++)
   {
      gp_sum += grade_pt[i] * cred[i];
      cred_sum += cred[i];
   }
   gpa = gp_sum / cred_sum;
   printf ("\n Grade point average:  is %d", gpa);
}
```

Things you cannot do

- You cannot
  - use = to assign one array variable to another

    ```
    a = b;   /* a and b are arrays */
    ```

  - use == to directly compare array variables

    ```
    if  (a = = b)  ……..
    ```

  - directly scanf or printf arrays

    ```
    printf (“……”, a);
    ```

How to copy the elements of one array to another?

- By copying individual elements

```
int a[25],b[25];
 for  (j=0; j<25; j++)
        a[j] = b[j];
```

How to read the elements of an array?

- By reading them one element at a time

```
int a[25];
for  (j=0; j<25; j++)
        scanf  (“%f”, &a[j]);
```

- The ampersand (&) is necessary.
- The elements can be entered all in one line or in different lines.

How to print the elements of an array?

- By printing them one element at a time.

```
for  (j=0; j<25; j++)
    printf  ("\n %f", a[j]);
```

  - The elements are printed one per line.

```
printf  ("\n");
for  (j=0; j<25; j++)
    printf (" %f", a[j]);
```

  - The elements are printed all in one line (starting with a new line).

Example: Matrix Addition

```
#include  <stdio.h>

main()
{
   int  a[100][100], b[100][100],
            c[100][100], p, q, m, n;

   scanf (“%d %d”, &m, &n);

   for  (p=0; p<m; p++)
      for  (q=0; q<n; q++)
         scanf (“%d”, &a[p][q]);

   for  (p=0; p<m; p++)
      for  (q=0; q<n; q++)
         scanf (“%d”, &b[p][q]);
```

```
   for  (p=0; p<m; p++)
      for  (q=0; q<n; q++)
         c[p]q] = a[p][q] + b[p][q];

   for  (p=0; p<m; p++)
   {
      printf  (“\n”);
      for  (q=0; q<n; q++)
         printf (“%f   ”, a[p][q]);
    }
}
```

Passing Arrays to a Function

- An array name can be used as an argument to a function.
  - Permits the entire array to be passed to the function.
  - Array name is passed as the parameter, which is effectively the address of the first element.
- Rules:
  - The array name must appear by itself as argument, without brackets or subscripts.
  - The corresponding formal argument is written in the same manner.
    - Declared by writing the array name with a pair of empty brackets.
    - Dimension or required number of elements to be passed as a separate parameter.

Example: Average of numbers

```
#include <stdio.h>

float avg(float [], int );

main()
{
  float a[]={4.0, 5.0, 6.0, 7.0};

  printf("%f \n", avg(a,4) );
}
```

prototype

Array name passed

Array as parameter

```
float  avg (float x[], int n)
{
 float sum=0;
 int i;

 for(i=0; i<n; i++)
    sum+=x[i];
 return(sum/(float) n);
}
```

Number of Elements used

5.5000

The Actual Mechanism

- When an array is passed to a function, the values of the array elements are not passed to the function.
  - The array name is interpreted as the address of the first array element.
  - The formal argument therefore becomes a pointer to the first array element.
  - When an array element is accessed inside the function, the address is calculated using the formula stated before.
  - Changes made inside the function are thus also reflected in the calling program.

Contd.

- Passing parameters in this way is called

  call-by-reference.
- Normally parameters are passed in C using

  call-by-value.
- Basically what it means?
  - If a function changes the values of array elements, then these changes will be made to the original array that is passed to the function.
  - This does not apply when an individual element is passed on as argument.

Example: Minimum of a set of numbers

```
#include  <stdio.h>

main()
{
   int  a[100], i, n;

   scanf ("%d", &n);
   for  (i=0; i<n; i++)
      scanf ("%d", &a[i]);

   printf ("\n Minimum is %d",
                              minimum (a, n));
}
```

```
int  minimum (x, size)
int  x[], size;
{
   int  i, min = 99999;

   for  (i=0; i<size; i++)
      if  (min < a[i])
         min = a[i];

   return (min);
}
```

Some Exercise Problems to Try Out

- Find the mean and standard deviation of a set of n numbers.
- A shop stores n different types of items. Given the number of items of each type sold during a given month, and the corresponding unit prices, compute the total monthly sales.