Arrays in C

Prof. Indranil Sen Gupta
Dept. of Computer Science & Engg.
Indian Institute of Technology
Kharagpur

1

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.**

2

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;
```

3

The Problem

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

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

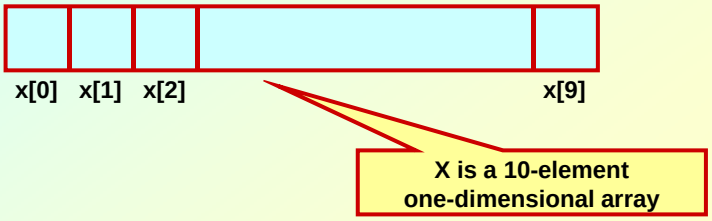
4

Using Arrays

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

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

- Individual elements are accessed by specifying the index.



5

- The name of the array also denotes the starting address of the array in memory.
  - **Example**:

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

`x[0], x[1], x[2], ...` indicates the contents of the successive array locations.

x indicates the starting address in memory for the array.

6

An Example

```
#include <stdio.h>
main()
{
      int x[10];
      x[0] = 15;
      x[1] = x[0] + 5;
      printf ("\n%d %d %d %u \n", x[0], x[1], x[2], x);
}
```

Output:

```
15 20 1107384350 3221224640
```

Garbage

Address

7

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 data 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.
- Example: `int marks[5];`
  - **marks** is an array containing a maximum of 5 integers.

8

- **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.**

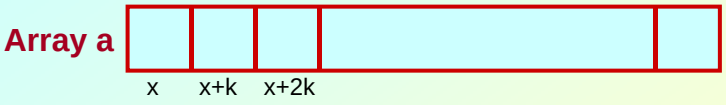
```
int  marks[50];
```

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

9

How an array is stored in memory?

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



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.**

10

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.

11

Contd.

- The array index must evaluate to an ***integer between 0 and n-1*** where n is the number of elements in the array.
- Any integer expression can be given as the index.

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

12

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.

13

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
```

14

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'};
```

15

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

```
#include  <stdio.h>
main()
{
    int  a[10], i, min;

    for  (i=0; i<10; i++)
        scanf ("%d", &a[i]);

    min = 99999;    /* or, min=a[0] */
    for  (i=0; i<10; i++)
    {
        if  (a[i] < min)
            min = a[i];
    }
    printf ("\n Minimum is %d", min);
}
```

16

Alternate Version 1

Change only one line to change the problem size

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

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

    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);
}
```

17

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;

    scanf ("%d", &n);
             /* Number of elements */

    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);
}
```

18

Example 2: Computing gpa

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

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

    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 = (float) gp_sum / cred_sum;
    printf ("\n GPA is: %f", gpa);
}
```

Handling two arrays at the same time

19

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);
    ```

20

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];
```

21

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.

22

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).

23

Passing Arrays to a Function

24

How to pass 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.
  - The way it is passed differs from that for ordinary variables.
- 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.

25

An Example with 1-D Array

We can also write

float x[100];

But the way the function is written makes it general; it works with arrays of any size.

```
main()
{
   int  n;
   float   list[100], avg;
   :
   avg  =  average(n,list);
    :
}

float average(a,x)
int a;
float x[];
{
   :
   sum = sum + x[i];
}
```

26

Same program, with the parameter types specified in the same line as the function definition.

```
main()
{
   int  n;
   float   list[100], avg;
   :
   avg  =  average(n,list);
    :
}

float average(int a, float x[])
{
   :
   sum = sum + x[i];
}
```

27

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.

28

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.

29

Example: Parameter passed as a value

```
#include <stdio.h>

void swap (int a, int b)
{
   int temp;

   temp=a;
   a=b;
   b=temp;
}
```

```
main()
{
   int x,y;

   x=10;  y=15;
   printf("x=%d y=%d \n",x,y);
   swap(x,y);
   printf("x=%d y=%d \n",x,y);
}
```

Output:

```
x=10 y=15
x=10 y=15
```

30

```
main()
{
   int x,y;

   x=10;  y=15;
   printf("x=%d y=%d \n",x,y);
   swap(x,y);
   printf("x=%d y=%d \n",x,y);
}
```

Output:

```
x=10 y=15
x=10 y=15
```

31

Example: Minimum of a set of numbers

```
#include <stdio.h>
int minimum (int x[], int y);

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 > x[i])
        min = x[i];
  return (min);
}
```

Parameter x passed *by reference*, size *by value*.

32

Example: Square each element of array

```
#include <stdio.h>
void square (int a[], int b);

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

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

  square (a, n);

  printf ("\nNew array is: ");
  for (i=0; i<n; i++)
    printf (" %d", a[i]);
}
```

```
void square (x,size)
int x[], size;
{
  int i;

  for (i=0;i<size;i++)
     x[i] = x[i]*x[i];

  return;
}
```

33

Introduction to Pointers

- What is the concept?
  - Pointer is a variable which stores the address in memory location of another variable.
  - When declared, we must specify the data type of the variable being pointed to.
  - Examples:

```
int *p;
float  *x, *y;
char  *flag;
```

34

- **A pointer variable can be assigned the address of another variable.**

```
int  a, *p;
a=10;
p = &a;  /* Address of 'a' assigned to 'p' */
printf ("%d %d", a, *p);
         /* Will print "10 10" */
```

- **Point to note:**
  - **Array name indicates pointer to first array element.**

```
int  num[10], *xyz;
xyz = num;  /* Points to x[0] */
```

35

  - **When an integer expression E is added to or subtracted from a pointer, actually scale factor times E is added/subtracted.**
    - **Scale factor indicates size of the data item being pointed to in number of bytes.**
    - **Scale factor for char is 1, int is 4, float is 4, double is 8, etc.**

```
int a, *p;
p = &a;  /* p is assigned address of a
                              (say, 2500) */
p++;          /* p will become 2504 */
p = p - 10;  /* p will become 2464 */
```

36

- **Consider the declaration:**

```
int x[5] = {1, 2, 3, 4, 5};
int *p;
```

  - Suppose that the base address of x is 2500, and each integer requires 4 bytes.

Element	Value	Address
x[0]	1	2500
x[1]	2	2504
x[2]	3	2508
x[3]	4	2512
x[4]	5	2516

37

Contd.

Both x and `&x[0]` have the value `2500`.

`p = x;` and `p = &x[0];` are equivalent.

- **Relationship between p and x:**

```
p   = &x[0] = 2500
p+1 = &x[1] = 2504
p+2 = &x[2] = 2508
p+3 = &x[3] = 2512
p+4 = &x[4] = 2516
```

*(p+i) gives the value of x[i]

38

- **An example:**

```
int  x[ ] = {1,2,3,4,5,6,7,8,9,10};
int  *p;

p = x + 3;  /* Point to fourth element of x */

printf ("%d", *p);  /* Will print 4 */

printf ("%d", *(p+5));
                    /* Will print 9 */

printf ("%d %d", p[3], p[-1]);
                    /* Will print 7 and 3 */
```

39

Example: function to find average

```
#include <stdio.h>
main()
{
  int x[100], k, n;

  scanf ("%d", &n);

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

  printf  ("\nAverage is %f",
               avg (x, n));
}
```

```
float avg (array, size)
int array[], size;
{
  int  *p, i , sum = 0;

  p = array;

  for (i=0; i<size; i++)
      sum = sum + *(p+i);

  return ((float) sum / size);
}
```

40

Example: SWAP revisited

```
#include <stdio.h>

void swap (int *a, int *b)
{
   int temp;

   temp = *a;
   *a = *b;
   *b = temp;
}
```

```
main()
{
   int x,y;

   x=10;  y=15;
   printf("x=%d y=%d \n",x,y);
   swap(&x, &y);
   printf("x=%d y=%d \n",x,y);
}
```

Output:

```
x=10 y=15
x=15 y=10
```

41