One-Dimensional Arrays

Random access lists of elements

CS10003 PROGRAMMING AND DATA STRUCTURES

Array

Many applications require multiple data items that have common characteristics

- In mathematics, we often express such groups of data items in indexed form:
  - $x_1, x_2, x_3, \ldots, x_n$

Array is a data structure which can represent a collection of data items which have the same data type (float/int/char/…)

Example: Printing Numbers in Reverse

3 numbers

```
int a, b, c;
scanf("%d", &a);
scanf("%d", &b);
scanf("%d", &c);
printf("%d  ", c);
printf("%d  ", b);
printf("%d  \n", a);
```

4 numbers

```
int a, b, c, d;
scanf("%d", &a);
scanf("%d", &b);
scanf("%d", &c);
scanf("%d", &d);
printf("%d  ", d);
printf("%d  ", c);
printf("%d  ", b);
printf("%d  \n", a);
```

Suppose we have 100 numbers to handle

Where to store the numbers? Use 100 variables?

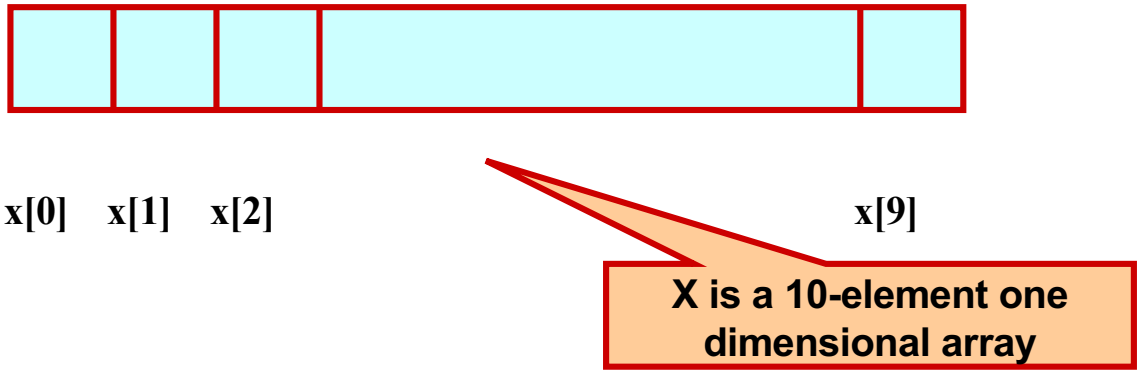
Solution: Use arrays

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 used in a program must be declared before they are used

General syntax:

```
type   array-name [size];
```

- **type** specifies the type of elements 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 that can store a maximum of 5 integers

Array Declarations: examples

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 that we will need, we can define an array of a large enough size

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

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

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

Important to remember: In C, the index of an array starts from 0, not 1

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.

A First Example

Array size should be a constant

```
int main()
{
  int i;
  int data[10];
  for (i=0; i<10; i++)
       data[i]= i;
  i=0;
  while (i<10)
  {
     printf("Data[%d] = %d\n", i, data[i]);
     i++;
  }
  return 0;
}
```

"data" is a block of 10 integer variables:
data[0], data[1], …, data[9]

```
Data[0] = 0
Data[1] = 1
Data[2] = 2
Data[3] = 3
Data[4] = 4
Data[5] = 5
Data[6] = 6
Data[7] = 7
Data[8] = 8
Data[9] = 9
```

Output

How is an array stored in memory?

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

Array A

- Let x: starting address of the array in memory
- k: number of bytes allocated per array element (e.g., 4 for each int, 1 for each char)

- The array element A[i] is allocated memory location at address $x + i*k$

A Special Operator: AddressOf (&)

Remember that each variable is stored at a memory location with a unique address

Putting & before a variable name gives the starting address of the variable in the memory (where it is stored, not the value)

Can be put before any variable (with no blank in between)

```
int a =10;
printf("Value of a is %d, and address of a is %d\n", a, &a);
```

Similarly, if we have an array data

Memory address of the first array element is &data[0]

Memory address of the second array element is &data[1]

Memory address of the third array element is &data[2]

Name of an array is interpreted as the address to the first element

If we have an array data
data is interpreted as &data[0]

Example

```
int main()
{
  int i;
  int Data[10];
  for(i=0; i<10; i++)
  printf("&Data[%d] = %u\n", i, &Data[i]);
  return 0;
}
```

Note: memory addresses are being printed as unsigned integers using %u in printf

Output

```
&Data[0] = 322122480
&Data[1] = 322122484
&Data[2] = 322122488
&Data[3] = 322122492
&Data[4] = 3221224496
&Data[5] = 3221224500
&Data[6] = 3221224504
&Data[7] = 3221224508
&Data[8] = 3221224512
&Data[9] = 3221224516
```

How to read the elements of an array?

By reading them one element at a time

Suppose we have declared an array: float a[25];

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

Note the ampersand (&) in scanf

The elements can be entered all in one line or in different lines

Reading into an array: example

```
int main()  {
    const int MAX_SIZE = 100;
    int i, size;
    float marks[MAX_SIZE];
    float total;
    scanf("%d", &size);
    for (i=0, total=0; i<size; i++)
    {
        scanf("%f", &marks[i]);
        total = total + marks[i];
     }
   printf("Total = %f \n Avg = %f\n", total,
total/size);
   return 0;
}
```

Input a list of marks from the user and compute their total and average

Output

```
4
2.5
3.5
4.5
5
Total = 15.500000
 Avg = 3.875000
```

Printing in Reverse Using Arrays

```
int main()
{
    int n, A[100], i;
    printf("How many numbers to read? ");
    scanf("%d", &n);
    for (i = 0; i < n; ++i)
                    scanf("%d", &A[i]);         // input the i-th array element
    for (i = n -1; i >= 0; --i)                 // note – loop counts downward
                    printf("%d  ", A[i]);       // output the i-th array element
    printf("\n");
    return 0;
}
```

Indexes into Arrays

The array index can be any expression that evaluates to an integer between 0 and n-1 where n is the maximum number of elements possible in the array

```
a[x+2] = 25;

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

Remember that each array element is a variable in itself, and can be used anywhere a variable can be used (in expressions, assignments, conditions,…)

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'};
```

The size may be omitted if all initializers are specified. 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'};
```

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, which are very hard to debug

How to copy the elements of one array to another?

By copying individual elements

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

The element assignments will follow the rules of assignment expressions

Destination array must have sufficient size

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);
```

Example: Find the minimum of a set of 10 numbers

```
int main()
{
   int  a[10], i, min;

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

   min = a[0];
   for  (i=1; i<10; i++)
   {
      if  (a[i] < min)
         min = a[i];
   }
   printf ("\n Minimum is %d", min);
   return 0;
}
```

Alternate Version 1

Change only one line to change the problem size

```
const int size = 10;

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

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

   min = a[0];
   for  (i=1; i<size; i++)
   {
      if  (a[i] < min)
         min = a[i];
   }
   printf ("\n Minimum is %d", min);
   return 0;
}
```

Alternate Version 2

Change only one line to change the problem size

Used #define macro

```
#define size 10

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

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

   min = a[0];
   for  (i=1; i<size; i++)
   {
      if  (a[i] < min)
         min = a[i];
   }
   printf ("\n Minimum is %d", min);
   return 0;
}
```

Alternate Version 3

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

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

   scanf ("%d", &n);  /* Number of elements */
   for  (i=0; i<n; i++)
      scanf ("%d", &a[i]);

   min = a[0];
   for  (i=1; i<n; i++)
   {
      if  (a[i] < min)
         min = a[i];
   }
   printf ("\n Minimum is %d", min);
   return 0;
}
```

Example: Computing Grade Point Average

Handling two arrays at the same time

cred[j] stores credit of subject j

grade_pt[j] stores grade point obtained by a student in subject j

```
const int  nsub = 6;

int main()
{
   int  grade_pt[nsub], cred[nsub], i, gp_sum=0,
cred_sum=0;
   double 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 Grade point average:  is %f", gpa);
   return 0;
}
```

Passing Arrays to a Function

How to pass arrays to a function?

An array name can be used as an argument in a function.

- Permits the entire array to be passed to the function.
- The way it is passed differs from that for ordinary variables.

Rules:

- Function call: the array name must appear by itself as argument, without brackets or subscripts.
- Function definition: corresponding formal argument is declared by writing the array name with a pair of empty brackets.

We can also write

float x[100]

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

Note: the number of valid elements in the array is passed to the function as another integer argument (len)

Why is this needed?

```
main()
{
    int  n;
    float   list[100], avg;

    // input n values into array
    avg  =  average(n, list);
     :
}

float average(int len, float x[])
{
    float sum = 0.0; int j;
    for(j=0;j<len;j++)
        sum = sum + x[i];

    return (sum/len);
}
```

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(int x[], int size)
{
  int i, min = x[0];

  for (i=1;i<size;i++)
     if (x[i] < min)
        min = x[i];

  return (min);
}
```

Note: When a function takes an array as argument, it needs to know how many valid elements the array contains.

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 argument in the function therefore gets the address of the first array element.
- When an array element is accessed inside the function, the address is calculated using the formula stated before.
- Changes made to the array inside the called function are also reflected in the calling function.

Basically what it means:

- If a function changes the elements of an array that is passed as argument, these changes will be made to the original array that is passed to the function.
- This does not apply when an individual element of an array is passed as argument.

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 (int x[], int size)
{
  int i;

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

  return;
}
```

More Examples

Reversing an Array: Find the mistake, reason for it and correct it

Correctly Reversing A = { 1, 2, 0, 5, 3 } results is A = { 3, 5, 0, 2, 1 }

The program below is wrong.

```
#include<stdio.h>
main()
{
  int A[20], n,k,temp;
  scanf("%d", &n);
  printf("n= %d \n", n);
  for (k=0; k<n; k++) scanf("%d", &A[k]);
  printf("numbers read are: ");
  for (k=0; k<n; k++) printf("%d  ", A[k]);
  printf("\n");
  for(k=0; k<n; k++){
     temp = A[k];
     A[k]=A[n-k-1];
     A[n-k-1] = temp;
  }
  for (k=0; k<n; k++) printf("%d  ", A[k]);
  printf("\n");
  }
```

Finding the largest contiguous sequence of equal numbers

```
#include<stdio.h>
main() {
int i, n, A[20], k = 0, maxbegin = 0, maxcount = 1, ssbegin, count;
scanf("%d\n", &n); for(i=0; i<n; i++) scanf("%d", &A[i]);
printf("A = "); for(i=0; i<n; i++) printf("%d, ", A[i]); printf("\n");
while(k < n) {
   ssbegin = k; count = 1;
   while(A[k] == A[k+1]) {
      k++; count++;
      if (k == n-1) break;
}
  if (count > maxcount) { maxbegin = ssbegin; maxcount = count; }
  k++;
}
printf("Sequence starting from A[%d] =  of Length = %d, Value = %d \n", maxbegin, maxcount,
A[maxbegin]);
}
```

```
10
1 2 2 2 3 2 2 2 2 7
A = 1, 2, 2, 2, 3, 2, 2, 2, 2, 7,
Sequence starting from A[5] =  of Length = 4, Value = 2
```

Practice Problems

1. Read in an integer n (n < 25). Read n integers in an array A. Then do the following (write separate programs for each, only the reading part is common).
   a) Find the sum of the absolute values of the integers.
   b) Copy the positive and negative integers in the array into two additional arrays B and C respectively. Print A, B, and C.
   c) Exchange the values of every pair of values from the start (so exchange A[0] and A[1], A[2] and A[3] and so on). If the number of elements is odd, the last value should stay the same.

2. Read in two integers n and m (n, m < 50). Read n integers in an array A. Read m integers in an array B. Then do the following (write separate programs for each part, only the reading part is common).
   a) Find if there are any two elements x, y in A and an element z in B, such that x + y = z
   b) Copy in another array C all elements that are in both A and B (intersection)
   c) Copy in another array C all elements that are in either A and B (union)
   d) Copy in another array C all elements that are in A but not in B (difference)