One-Dimensional Arrays and Strings

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 having the same data type (float/int/char/...)

Why do we need arrays?

Printing 3 numbers in reverse

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

Printing 4 numbers in reverse

```
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 10 numbers to handle
  - or 20
  - or 100
- Where do we store the numbers ? Use 100 variables ?

Solution:

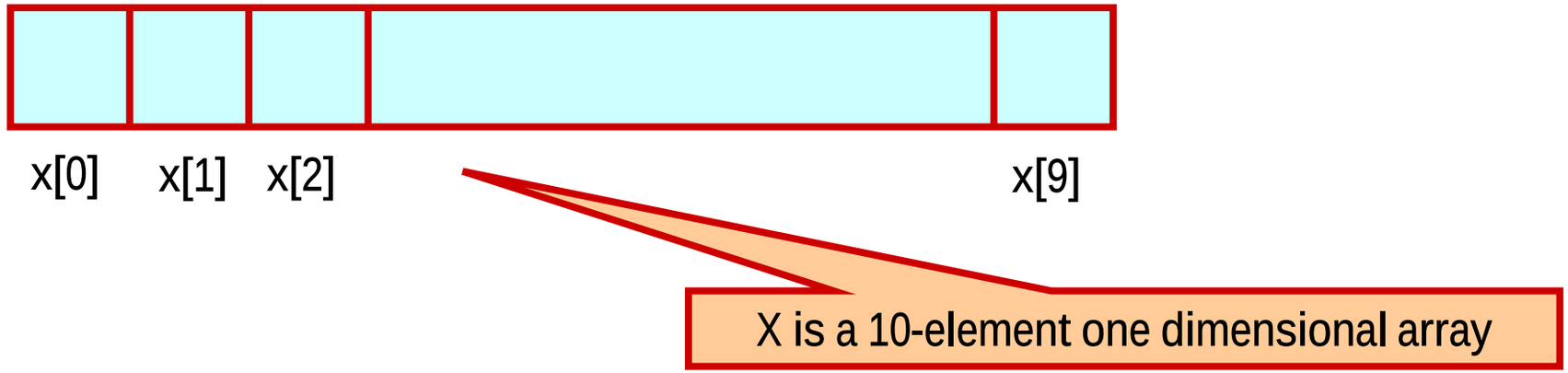
- Use arrays

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 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

- marks is an array that can store a maximum of 5 integers

```
int   marks[5];
```

How is an array 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

- A[i] is allocated memory location at address x + (i * k)

A First Example

```
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;
}
```

Array size should be a constant

data refers to a block of 10 integer variables, namely:

data[0], data[1], …, data[9]

Output:

```
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
```

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]);
    for (i = n – 1; i >= 0; --i)
                        printf(“%d  ”, A[i]);
    printf(“\n”);
    return 0;
}
```

Array Declarations

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 enough size, say

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

  – though in a particular execution 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.

Handling 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 wherever a variable can be used (in expressions, assignments, conditions,…)

A Special Operator: AddressOf (&)

- Remember that each variable is stored at a memory location with an unique address
- Putting & before a variable name gives the starting address of the variable (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);
```

Example

```
int main( )
{
    int i;
    int data[10];

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

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
```

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. 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’ };
```

How to read the elements of an array?

By reading them one element at a time

```
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

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 during compilation
- Rather, it may result in unpredictable program results, which are very hard to debug

Reading into an array

```
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 / OUPUT:

4

2.5

3.5

4.5

5

Total = 15.500000

Avg = 3.875000

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 ] );
```

- Here the elements are printed one per line

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

- Here the elements are printed all in one line (starting with a new line)

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

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;
}
```

We could also use

```
#define size 10
```

instead of

```
const int size = 10;
```

Are they the same?

Instead of dealing with the constant 10 at multiple places, we can define it as a constant

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

   for  (i=0; i<10; 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;
}
```

#define macro

#define X Y

Preprocessor directive

- The #include you have been using is also a preprocessor directive

Compiler will first replace all occurrences of string X with string Y in the program, then compile the program

Similar effect as read-only variables (const), but no storage allocated

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

```
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 %.2lf”, gpa);
       return 0;
}
```

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);

Character Arrays and Strings

```
char C[8] = { 'a', 'b', 'h', 'i', 'j', 'i', 't', '\0' };
```

- C[0] gets the value 'a', C[1] the value 'b', and so on.
- The last (7th) location receives the null character ‘\0’

- Null-terminated character arrays (last character is ‘\0’) are also called null-terminated strings or just strings.
- Strings can be initialized in an alternative way. The last declaration is equivalent to:

```
char C[8] = "abhijit";
```

- The trailing null character is missing here. C automatically puts it at the end if you define it like this
- Note also that for individual characters, C uses single quotes, whereas for strings, it uses double quotes

Reading strings: %s format

```
int main()
{
    char name[25];
    scanf("%s", name);
    printf("Name = %s \n", name);
    return 0;
}
```

Note that there is no & here !! Why?

- %s reads a string into a character array given the array name or start address
- It ends the string with the special “null” character ‘\0’.

Example: Finding length of a string

```
#define SIZE 25
int main()
{
    int i, length=0;
    char name[SIZE];

    scanf("%s", name);
    printf("Name = %s \n", name);

    for (i=0; name[i] != '\0'; i++) length++;
    printf("Length = %d\n", length);
    return 0;
}
```

Note that character strings read in %s format end with '\0'

INPUT / OUPUT:

Satyanarayana

Name = Satyanarayana

Length = 13

Example: Counting the number of a’s

```
#define SIZE 25
int main()
{
    int i, count=0;
    char name[SIZE];

    scanf("%s", name);
    printf("Name = %s \n", name);

    for (i=0; name[i] != '\0'; i++)
            if (name[i] == ‘a’) count++;
    printf(“Count = %d\n", count);
    return 0;
}
```

INPUT / OUPUT:

Satyanarayana

Name = Satyanarayana

Count = 6

Example: Palindrome Checking

```
int main( )
{
        int i, flag, count=0;
        char name[25];
        scanf("%s", name);              /* Read Name */

        for (i=0; name[i]!='\0'; i++);    /* Find Length of String */
        count=i;  flag = 0;

        /* Loop below checks for palindrome  by comparison*/
        for(i=0; i<count; i++)
              if ( name[ i ] != name[count – i – 1] ) flag = 1;

        if (flag ==0) printf ("%s is a Palindrome\n", name);
        else printf("%s is NOT a Palindrome\n", name);
        return 0;
}
```

Is there scope for making it more efficient?

Examples for Discussion in Class

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

```
#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");
 }
```

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

This program is wrong. Why?

Sorting Elements

```
#include<stdio.h>
main( )
{
    int A[20], n, i,j,k, temp;
    scanf("%d", &n); printf("n = %d\n", n);
    for (k=0; k<n;k++) scanf("%d", &A[k]);
    printf("Input: ");  for (k=0;k<n;k++) printf("%d ", A[k]); printf("\n");

    for(i=0; i < n – 1; i++){
            for (j=0; j < n – i – 1; j++) {
                    if(A[j] > A[j+1]) { temp = A[j];  A[j] = A[j+1]; A[j+1] = temp; }
            }
            printf("A at i = %d: ", i); for (k=0;k<n;k++) printf("%d ", A[k]);
            printf("\n");
     }
     printf("Sorted Output: "); for (k=0;k<n;k++) printf("%d ", A[k]); printf("\n");
}
```

```
5
n = 5
5 9 1 7 2
Input: 5 9 1 7 2
A at i = 0: 5 1 7 2 9
A at i = 1: 1 5 2 7 9
A at i = 2: 1 2 5 7 9
A at i = 3: 1 2 5 7 9
Sorted Output: 1 2 5 7 9
```

```
4
n = 4
9 6 4 2
Input: 9 6 4 2
A at i = 0: 6 4 2 9
A at i = 1: 4 2 6 9
A at i = 2: 2 4 6 9
Sorted Output: 2 4 6 9
```

Finding the largest 6 of n numbers using an array of fixed size

```
#include<stdio.h>
main( )
{
    int A[7], n, k, i, j, temp;
    scanf("%d", &n);  printf("n= %d \n", n);
    scanf("%d", &A[0]); printf("0-th Number read = %d\n", A[0]);
    k=1;
    for(i=1;i<n;i++){
            scanf("%d", &A[k]);
            printf("%d-th Number read = %d, k = %d\n", i, A[k], k);
            for (j=k; j>0; j--){
               if (A[j] > A[j-1]) { temp = A[j];  A[j] = A[j-1]; A[j-1] = temp;  }
            }
            if (i > 5) k = 6; else k++;
     }
     printf("Final Top %d are:", k);
     for (j=0; j<k; j++) printf("%d  ", A[j]); printf("\n"); return 0;
}
```

```
9
n= 9
7 3 1 5 7 3 2 9 6
0-th Number read = 7
1-th Number read = 3, k = 1
2-th Number read = 1, k = 2
3-th Number read = 5, k = 3
4-th Number read = 7, k = 4
5-th Number read = 3, k = 5
6-th Number read = 2, k = 6
7-th Number read = 9, k = 6
8-th Number read = 6, k = 6
Final Top 6 are:9  7  7  6  5  3
```

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
```

Extraction of Vowels from a String

```
#include<stdio.h>
main( )
{
     char A[40], B[5] ={'a', 'e', 'i', 'o', 'u'};
     int i, j, len, C[5]= {0,0,0,0,0};
    scanf("%s", A); printf("A = %s \n", A);

    for (i=0; A[i]!='\0'; i++);
    len = i;
    printf("Length = %d\n", len);
    for(i=0; i<len; i++){
                for(j=0; j<5;j++)
                            if(A[i]== B[j]) C[j]++;
    }
   for (j=0;j<5;j++) printf("Number of %c = %d \n", B[j], C[j]);
}
```

```
thequickbrownfoxjumpsoverthelazydog
A = thequickbrownfoxjumpsoverthelazydog
Length = 35
Number of a = 1
Number of e = 3
Number of i  = 1
Number of o = 4
Number of u = 2
```

Handling Strings with Blanks

```
main()
{
    char A[20], B[20], c;
    int len1, len2;
    printf("Enter the First String:");
    scanf("%s", A);  printf("First String is %s\n", A);
    for(len1=0;A[len1]!='\0';len1++);
    printf("Length is %d\n", len1);

    printf("Enter the Second String:");
    len2=0;
    fflush(stdin);
    B[len2] = getchar();
    while(B[len2]!='\n') {len2++; B[len2] = getchar();}
    B[len2] = '\0';
    printf("Second String is %s\n", B);
    printf("Length is %d\n", len2);
}
```

```
Enter the First String:IIT Kharagpur
First String is IIT
Length is 3
Enter the Second String:IIT Kharagpur
Second String is IIT Kharagpur
Length is 13
```

```
Enter the First String:IIT Kharagpur India
First String is IIT
Length is 3
Enter the Second String:IIT Kharagpur India
Second String is IIT Kharagpur India
Length is 19
```

Can also be read using:
scanf("%[^\n]",B); /* read until newline char */

We may also use the library function:
char *gets(char *str) /* to be explained later */

Pattern Matching

```
#include<stdio.h>
main( )
{
    char S[20], P[20];
    int i,j, k, flag;
    printf("Enter String and Pattern:\n");
    scanf("%s%s", S, P);
    printf("S = %s,P = %s \n", S, P);

    k =0;
    for (i=0; S[i] != '\0'; i++){
            flag = 1;
            for(j=0;P[j]!='\0'; j++)
                    if (S[i+j]!= P[j]) {flag = 0; break;}
            if (flag == 1) k++;
    }
    printf("Number of Matches = %d \n", k);
}
```

```
Enter String and Pattern:
abababababab
aba
S = abababababab,P = aba
Number of Matches = 5
```

```
Enter String and Pattern:
abababababab
ababab
S = abababababab,P = ababab
Number of Matches = 4
```

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, 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)
3. Read in two null-terminated strings A and B (using %s. Assume max characters < 25 in each). Create another string C that is the concatenation of A and B (A followed by B). Print A, B, C using %s
4. Read in two null-terminated strings A and B. Check if A is lexicographically smaller, larger, or equal to B and print appropriate messages in each case.