1-d Arrays

Array

- Many applications require multiple data items that have common characteristics
 - □ In mathematics, we often express such groups of data items in indexed form:
 - X₁, X₂, X₃, ..., 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);
```

The Problem

- Suppose we have 10 numbers to handle
- Or 20
- Or 100
- Where do we store the numbers? Use 100 variables??
- How to tackle this problem?
- Solution:
 - Use arrays

Printing in Reverse Using Arrays

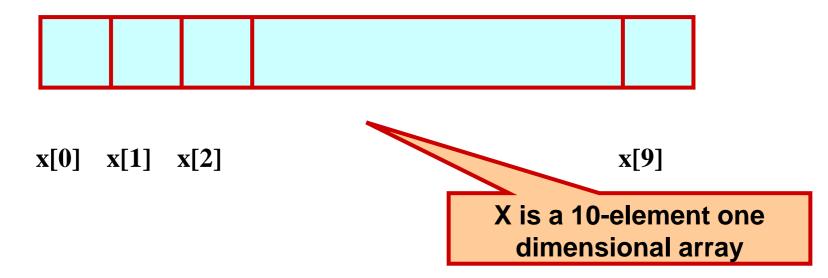
```
int main()
   int n, A[100], i;
   printf("How many numbers to read? ");
   scanf("%d", &n);
   for (i = 0; i < n; ++i)
       scanf("%d", &A[i]);
   for (i = n - 1; i >= 0; --i)
       printf("%d ", A[i]);
   printf("\n");
   return 0;
```

Using Arrays

All the data items constituting the group share the same name

int x[10];

Individual elements are accessed by specifying the index



A first example

```
"data refers to a block of 10
int main()
                                             integer variables, data[0], data[1],
                                             ..., data[9]
 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;
```

The result

```
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

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
```

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

int marks[5];

marks is an array that can store 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

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 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,...)

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

- 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

Storage

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

Output

&Data[0] = 3221224480&Data[1] = 3221224484&Data[2] = 3221224488&Data[3] = 3221224492&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;
```

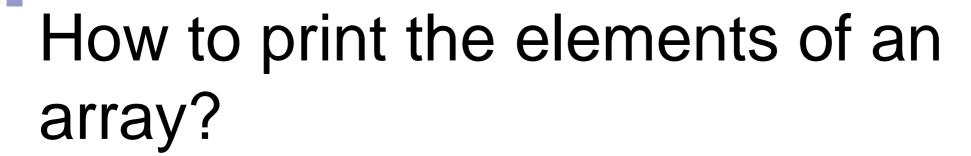
- The above assignment would not necessarily cause an error
- Rather, it may result in unpredictable program results

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 \setminus n Avg = \%f \setminus n", total,
total/size);
  return 0;
```

Output

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



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]);</pre>
```

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 1: 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;
```

#define macro

- #define X Y
- 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
- We prefer you use const instead of #define

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 2: Computing cgpa

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;
```

Example: Binary Search

- Searching for an element k in a sorted array A with n elements
- Idea:
 - Choose the middle element A[n/2]
 - \square If k == A[n/2], we are done
 - \square If k < A[n/2], search for k between A[0] and A[n/2 -1]
 - □ If k > A[n/2], search for k between A[n/2 + 1] and A[n-1]
 - Repeat until either k is found, or no more elements to search
- Requires less number of comparisons than linear search in the worst case (log₂n instead of n)

```
int main() {
  int A[100], n, k, i, mid, low, high;
  scanf("%d %d", &n, &k);
  for (i=0; i<n; ++i) scanf("%d", &A[i]);
  low = 0; high = n - 1; mid = low + (high - low)/2;
  while (high >= low) {
      printf("low = %d, high = %d, mid = %d, A[%d] = %d\n",
  low, high, mid, mid, A[mid]);
      if (A[mid] == k) {
        printf("%d is found\n", k);
         break;
      if (k < A[mid]) high = mid - 1;
      else low = mid + 1;
      mid = low + (high - low)/2;
  if (high < low) printf("%d is not found\n", k);
  return 0;
```

Output

```
8 21

9 11 14 17 19 20 23 27

low = 0, high = 7, mid = 3, A[3] = 17

low = 4, high = 7, mid = 5, A[5] = 20

low = 6, high = 7, mid = 6, A[6] = 23

21 is not found
```

```
8 14

9 11 14 17 19 20 23 27

low = 0, high = 7, mid = 3, A[3] = 17

low = 0, high = 2, mid = 1, A[1] = 11

low = 2, high = 2, mid = 2, A[2] = 14

14 is found
```



- Sort the elements of an array A with n elements in ascending order
- Basic Idea:
 - □ Find the min of the n elements, swap it with A[0] (so min is at A[0] now)
 - Now find the min of the remaining n-1 elements, swap it with A[1] (so 2nd min is at A[1] now)
 - Continue until no more elements left

```
int main() {
   int A[100], n, i, j, k, min, pos, temp;
   scanf("%d", &n);
   for (i=0; i<n; ++i) scanf("%d", &A[i]);
   for (i = 0; i < n - 1; ++i) {
        min = A[i]; pos = i;
        for (j = i + 1; j < n; ++j) {
           if (A[j] < min) {
               min = A[j]; pos = j;
       temp = A[i];
       A[i] = A[pos];
       A[pos] = temp;
       for (k=0; k<n; ++k) printf("%d ", A[k]);
       printf("\n");
   } /* end of outer for */
   return 0;
```

Output

```
6
7 12 5 15 17 9
5 12 7 15 17 9
5 7 12 15 17 9
5 7 9 15 17 12
5 7 9 12 17 15
5 7 9 12 15 17
```

```
98765432
      6 5 4 3 9
 3 7 6 5 4 8 9
 3 4 6 5 7 8 9
 3 4 5 6 7 8 9
 3 4 5 6 7 8
   4
     5
       6 7
           8
```

Things you cannot do

- You cannot
 - □ use = to assign one array variable to anothera = 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 (last character is '\0') character arrays are also called 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;
}
```

%s reads a string into a character array given the array name or start address.

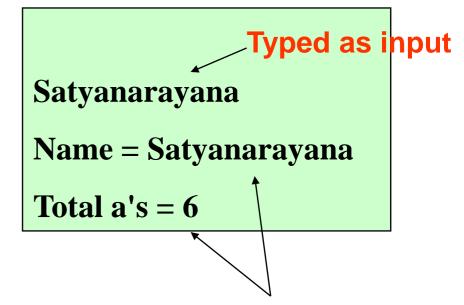
It ends the string with '\0'

An example

```
int main()
 #define SIZE 25
 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("Total a's = %d\n", count);
 return 0;
```

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

Seen on screen



Printed by program

Palindrome Checking

```
int main()
 const int SIZE = 25;
 int i, flag, count=0;
 char name[SIZE];
 scanf("%s", name); /* Read Name */
 for (i=0; name[i]!='\0'; i++); /* Find Length of String */
 printf("Total length = %d\n",i);
 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;
```

Some Exercises

- 1. Write a C program that reads an integer n and stores the first n Fibonacci numbers in an array.
- 2. Write a C program that reads an integer n and uses an array to efficiently find out the first n prime numbers.
- 3. Read in an integer n, read in n integers and print the integer with the highest frequency.
- 4. Read in an integer n, read in n numbers and find out the mean, median and mode.
- 5. Read in two names and compare them and print them in lexicographic (dictionary) order.
- 6. Read in an integer n, read in n names and print the last name when compared in lexicographic order.