CMSC 104 - Lecture 20
Park , adapted by C Grasso

Arrays: Part 1 of 2

Arrays, Part 1 of 2

Topics

- Definition of a Data Structure
- Definition of an Array
- Array Declaration, Initialization, and Access
- Program Example Using Arrays

Data Types - Simple

- So far, we have seen only **simple data types**, such as int, float, and char.
- Simple variables can hold only one value at any time during program execution, although that value may change.

Data Types – Complex / Composite

- A **data structure** is a data type that can hold multiple values, in a structured form, at the same time.
  - Synonyms: **complex data type**, **composite data type**

- The **array** is one kind of data structure.

A Motivating Example

- We want to write a program that will accept a collection of numerical grades, and then print out the *mean* grade
  - How do you calculate the mean ?

A Motivating Example

```
#include <stdio.h>

int main() {
    int counter = 0;
    float total = 0.0;

    do {
        scanf(“%d”, &grade);
        if (grade >= 0) {
            total += grade;
            counter++;
        }
    } while (grade >= 0);

    printf(“Mean for %d grades is %f”, counter, total / counter);

    return(0);
}
```

A Motivating Example

- Now, the user wants us to print out the ***median*** grade:
  - What is the median?
  - How do you calculate it from a set of grades?
  - What is needed in order to calculate it?

A Motivating Example

- We don’t know in advance exactly how many grades we will be getting
  - (We can, however, enforce an upper limit on how many we can handle)
- Can we do it “in place”, as with calculating the mean?
  - Unfortunately, NO.
- Can we do it with a collection of simple variables?
  - Again, NO.
- So, we need a special place to save *all* the input values

Arrays

- An array is a group of
  - related data items with the
  - same data type, and share a
  - common name
- Arrays can be of any data type we choose.
- An array's data items are stored contiguously in memory.

Arrays

- Each of the data items is known as an **element** of the array.

- Each element can be accessed individually.

- The maximum number of elements in the array remains the same throughout program execution.
  - We say that the **size** of the array is **static**.

Array Declaration

```
int   numbers[5] ;
```

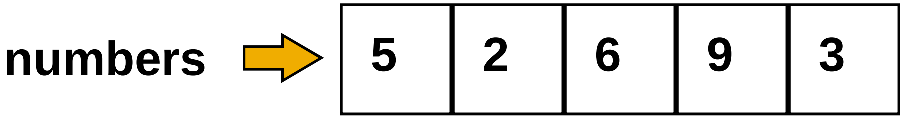
- The name of this example array is “numbers”.
- This declaration sets aside a chunk of memory that is big enough to hold 5 integers.
- It does not initialize those memory locations to 0 or any other value.
  - They contain garbage.

Array Initialization

```
int    numbers[5] ;
```

- Initializing an array may be done with an **array initializer**, as in :

```
int    numbers[5]   =   { 5, 2, 6, 9, 3 }   ;
```

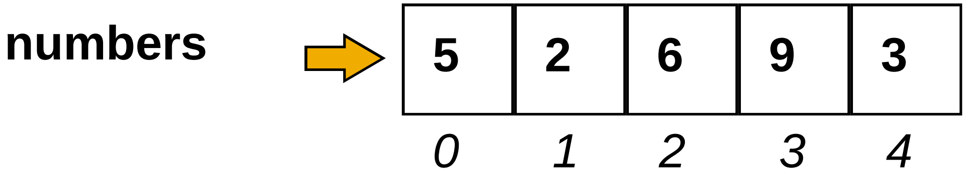


numbers

5	2	6	9	3

Accessing Array Elements

- Each element in an array has a **subscript** (**index**) associated with it.



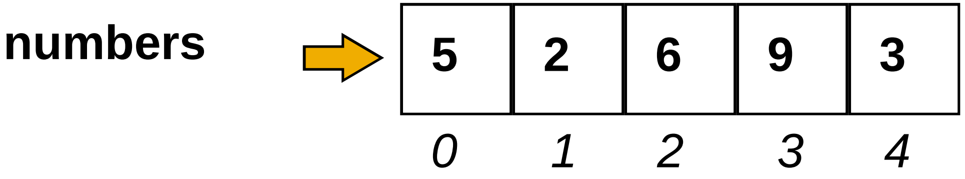
numbers

5	2	6	9	3
0	*1*	*2*	*3*	*4*

- Subscripts are integers and always begin at zero.

Accessing Array Elements

- Values of individual elements can be accessed by **indexing** into the array.



5	2	6	9	3
0	*1*	*2*	*3*	*4*

- For example,

printf(“The third element = %d. \n”, **numbers[2]**) ;

would give the output

The third element = 6.

Accessing Array Elements (con’t)

- A subscript can also be any expression that evaluates to an integer.

```
numbers[(a + b) * 2] ;
```

- Caution! It is a logical error when a subscript evaluates to a value that greater than the number of elements declared in the array.
  - Some systems will handle an **out-of-range error** gracefully and some will not (including ours).

Modifying Elements

- Individual elements of an array can also be modified using subscripts.

```
numbers[4] = 20 ;          // 5th element of the array contains 20
```

- Initial values may be stored in an array using indexing, rather than using an array initializer.

```
numbers[0] = 5 ;
numbers[1] = 2 ;
numbers[2] = 6 ;
numbers[3] = 9 ;
numbers[4] = 3 ;
```

Filling Large Arrays

- Since many arrays are quite large, using an array initializer can be impractical.

- Large arrays are often filled using a for loop.

```
        for ( i = 0; i < 100; i++ )
        {
           values [ i ] = 0 ;
        }
```

  would set every element of the 100 element array "values" to 0.

More Declarations

```
int     score [39] ;
int     gradeCount [5];
```

- Declares two arrays of type int.
- Neither array has been initialized.
- "score" contains 39 elements
  - one for each student in a class
- "gradeCount" contains 5 elements
  - one for each possible grade, A - F

Using #define for Array Sizes

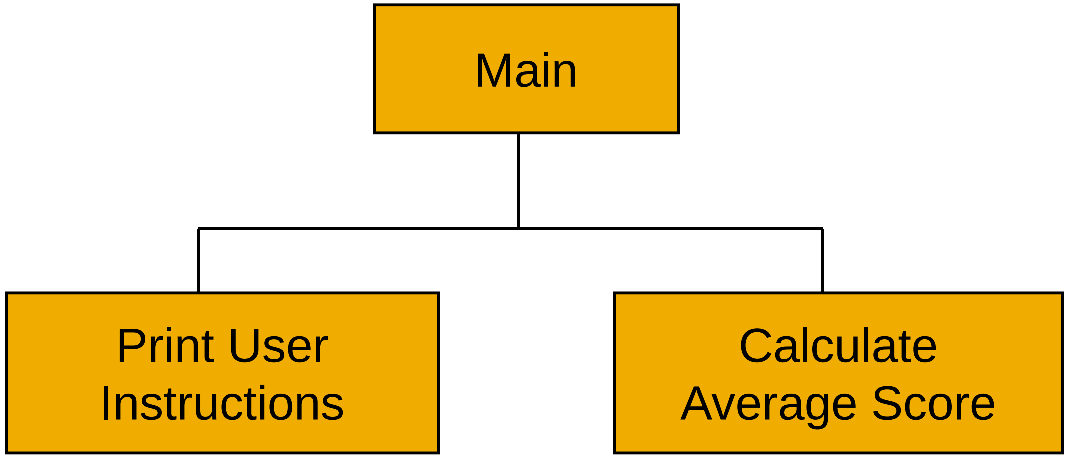
```
#define SIZE        39
#define GRADES  5

int main ( )
{
   int    score [SIZE] ;
   int    gradeCount [GRADES] ;
        ▪
        ▪
        ▪
}
```

Example Using Arrays

Problem: Find the average test score and the number of A's, B's, C's, D's, and F's for a particular class.

Design:



“Clean” Example Using Arrays (con’t)

```
#include <stdio.h>

#define  SIZE      39    /* number of tests                          */
#define  GRADES  5       /* number of different grades: A, B, C, D, F */

void PrintInstructions ( ) ;
double FindAverage (double sum, int quantity) ;
```

“Clean” Example Using Arrays (con’t)

```
int main ( )
{
    int i ;                      /* loop counter                      */
    int total ;                  /* total of all scores               */
    int score [SIZE] ;           /* student scores                    */
    int gradeCount [GRADES] ;    /* count of A’s, B’s, C’s, D’s, F’s  */
    double average ;             /* average score                     */

    /* Print the instructions for the user */
    PrintInstructions ( ) ;
```

“Clean” Example Using Arrays (con’t)

```
/* Initialize grade counts array to zero */

for ( i = 0; i < GRADES; i++ )
{
    gradeCount [ i ] = 0 ;
}


/* Fill score array with scores */

for ( i = 0; i < SIZE; i++ )
{
    printf (“Enter next score: ”) ;

    scanf (“%d “, &score [ i ] ) ;
}
```

“Clean” Example Using Arrays (con’t)

```
/* Calculate score total and count number of each grade */

for ( i = 0; i < SIZE; i++ )
{
    total += score [ i ] ;
    switch ( score [ i ] / 10 )
    {
        case 10 :
        case   9 :  gradeCount [4]++  ;    break ;
        case   8 :  gradeCount [3]++  ;    break ;
        case   7 :  gradeCount [2]++  ;    break ;
        case   6 :  gradeCount [1]++  ;    break ;
        default   :  gradeCount [0]++  ;
    }
}
```

“Clean” Example Using Arrays (con’t)

```
  /* Calculate the average score */

  average = FindAverage (total, SIZE) ;

  /* Print the results */

  printf (“The class average is %.2f\n”, average ) ;
  printf (“There were %2d As\n”, gradeCount [4] ) ;
  printf (“               %2d Bs\n”, gradeCount [3] ) ;
  printf (“               %2d Cs\n”, gradeCount [2] ) ;
  printf (“               %2d Ds\n”, gradeCount [1] ) ;
  printf (“               %2d Fs\n”, gradeCount [0] ) ;

  return 0 ;

} /* end main */
```

“Clean” Example Using Arrays (con’t)

```
/********************************************************************
** PrintInstructions - prints the user instructions
** Inputs:         None
** Outputs:        None
/********************************************************************
void PrintInstructions ( )
{
    printf (“This program calculates the average score\n”) ;
    printf (“for a class of 39 students.  It also reports the\n”) ;
    printf (“number of A’s, B’s, C’s, D’s, and F’s.  You will\n”) ;
    printf (“be asked to enter the individual scores.\n”) ;
}
```

"Clean" Example Using Arrays (con't)

```
/****************************************************************
** FindAverage - calculates an average
** Inputs:  sum - the sum of all values,     num - the number of values
** Outputs:  the computed average
****************************************************************/
double FindAverage (double sum, int num)
{
double average = 0 ;   /* computed average */

    if ( num != 0 ) {
        average = sum / num ;
    }

   return average ;
}
```

Improvements ?

- We're trusting the user to enter valid grades. Let's add input error checking.

- If we aren't handling our array correctly, it's possible that we may be evaluating garbage rather than valid scores. We'll handle this by adding all the cases for F's (0 - 59) to our switch structure and using the default case for reporting errors.

- We still have the "magic numbers" 4, 3, 2, 1, and 0 that are the quality points associated with grades. Let's use symbolic constants for these values.

Improved Program

```
#include <stdio.h>

#define  SIZE       39   /* number of scores                          */
#define GRADES  5        /* number of different grades: A, B, C, D, F */

#define A  4             /* A's position in grade count array         */
#define B  3             /* B's position in grade count array         */
#define C  2             /* C's position in grade count array         */
#define D  1             /* D's position in grade count array         */
#define F  0             /* F's position in grade count array         */

#define MAX 100          /* maximum valid score                       */
#define MIN   0          /* minimum valid score                       */

void PrintInstructions ( ) ;
double FindAverage (double sum, int quantity) ;
```

Improved Program

```
int main ( )
{
   int i ;                          /* loop counter                    */
   int total ;                      /* total of all scores             */
   int score [SIZE] ;               /* student scores                  */
   int gradeCount [GRADES] ;        /* count of A’s, B’s, C’s, D’s, F’s */
   double average ;                 /* average score                   */

   /* Print the instructions for the user */
   PrintInstructions ( ) ;
```

Improved Program (con't)

```
/* Initialize grade counts to zero */

for ( i = 0; i < GRADES; i++ )
{
    gradeCount [ i ] = 0 ;
}
```

Improved Program (con’t)

```
/* Fill array with valid scores */

for ( i = 0; i < SIZE; i++ )
{
    printf (“Enter next score : ”) ;
    scanf (“%d “, &score [ i ] ) ;

    while ( (score [ i ] < MIN)  ||  (score [ i ] > MAX) )
    {
        printf (“Scores must be between”) ;
        printf (“ %d and %d\n”, MIN, MAX) ;

        printf (“Enter next score : ”) ;
        scanf (“%d “, &score [ i ] ) ;
    }
}
```

Improved Program (con’t)

```
/* Calculate score total and count number of each grade */
for ( i = 0 ; i < SIZE ; i++ )
{
    total += score [ i ] ;
    switch ( score [ i ] / 10 )
    {
        case 10 :
        case   9 :  gradeCount [A]++  ;   break ;
        case   8 :  gradeCount [B]++ ;    break ;
        case   7 :  gradeCount [C]++  ;   break ;
        case   6 :  gradeCount [D]++ ;    break ;
        case   5 :
        case   4 :
        case   3 :
        case   2 :
        case   1 :
        case   0 :   gradeCount [F]++ ;   break;;

        default  :  printf(“Error in score.\n”) ;
    }
}
```

Improved Program (con’t)

```
     /* Calculate the average score */

     average = FindAverage (total, SIZE) ;

     /* Print the results */

    printf (“The class average is %.2f\n”, average ) ;
    printf (“There were %2d As\n”, gradeCount [4] ) ;
    printf (“                %2d Bs\n”, gradeCount [3] ) ;
    printf (“                %2d Cs\n”, gradeCount [2] ) ;
    printf (“                %2d Ds\n”, gradeCount [1] ) ;
    printf (“                %2d Fs\n”, gradeCount [0] ) ;

    return 0 ;

} /* end main */
```

Other Improvements?

- Why is main so large?
- Couldn’t we write functions to:
  - Initialize an array to hold all `0`s?
  - Fill an array with values entered by the user?
  - Count the grades and find the class average?
  - Print the results?
- Yes, we can as soon as we learn about passing arrays as parameters to functions in the next lecture.