University of Glasgow

电子科技大学 University of Electronic Science and Technology of China

UESTC1005 - Introductory Programming

One Dimension Arrays

Week 10 | Lecture 7

Dr Ahmed Zoha

Lecturer in Communication Systems

School of Engineering

University of Glasgow

Email: ahmed.zoha@glasgow.ac.uk

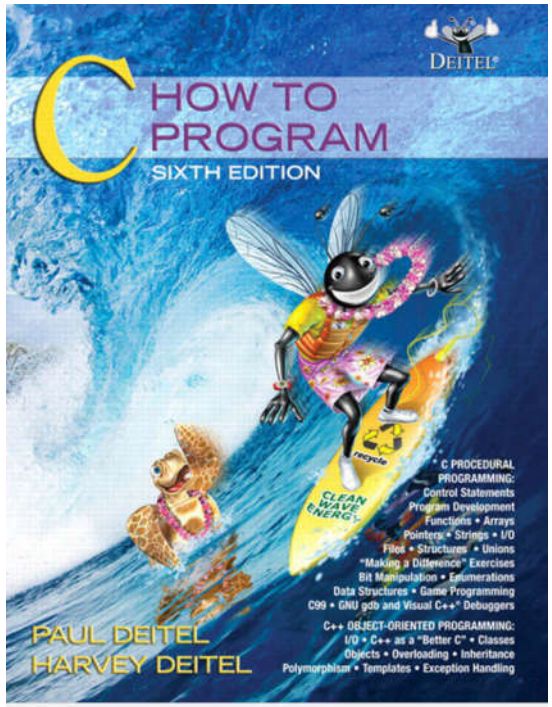
Topics Covered in Week 8/9

- Operators and Program Control
- Loops
- Functions

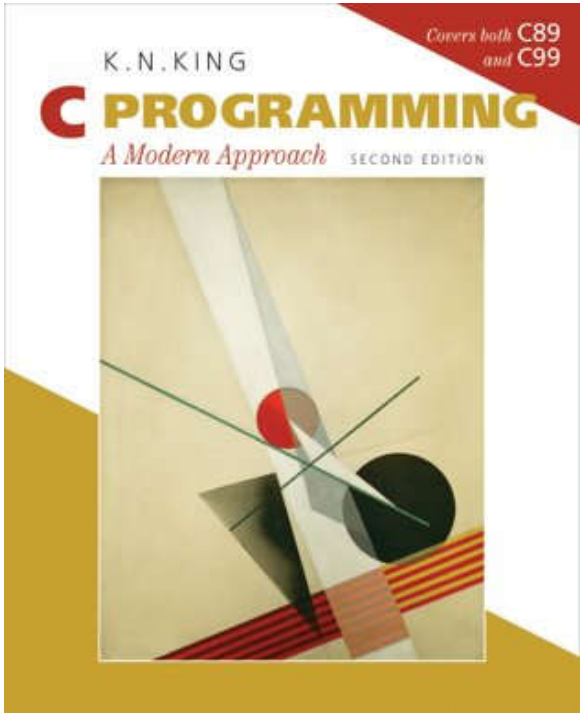
Topics to be Covered in Week 10

- 1D Arrays
- Multi-dimensional Arrays
- Strings

Reading Exercise for Week 10



C How to Program (DETEL)
Chapter 6
Chapter 8
and
Do Exercise



KING C Programming
Chapter 8
Chapter 13

Outline - 1D Array

- What is array
  - Array is a collection of data of a specified type
- Why do we need array
  - Arrays allows you to group values under one variable. You do not need separate variables for each data item

Motivation: Why we need Arrays? 1/2

- Up to now when we have dealt with variables that have all been single values or single characters. What if you want a range of values that are associated?

- Imagine you are required to store the assignment marks for a group of students. From the knowledge you have already gained on this module you would probably tackle the problem in the following way:

```
int score1, score2, ……, scoreN;
printf("Enter the student scores in turn: ");
scanf("%d",&score1);
    .
    .
    .
scanf("%d",&scoreN);
```

- where **N** is the number of students in the group

Motivation: Why we need Arrays? 2/2

- This seems to be a reasonable way of solving the problem until you consider sorting the data that has been entered based upon the marks achieved. How can this be done?

`score1` could be compared with each score entered and a decision made as to if it is the largest value or not

This would take '*N*' comparisons, and if *N=240* then I'm sure you can see that this is becoming a lengthy procedure that will only deal with the first score!

A more easy and efficient way to tackle this problem is via using **Arrays**

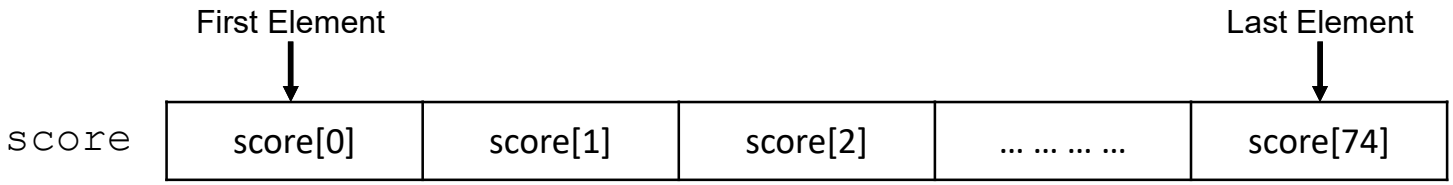
Using Arrays to Group Assignment Scores

- Instead of declaring individual variables, such as `score0, score1, ..., and score74`, you declare one array variable such as:

Array name

Array type → `float score[75];` ← Array size

- and use `score[0], score[1], ... , score[74]` to represent individual variables
- Each element in an array is accessed by an index
- Index start from 0



Compare: Variable/Array/function

Definition of Array

Array name

Array type

```
float score[75];
```

Array size

Difference:
[]
()

- Definition of Variable

Variable name

Variable type

```
float score;
```

Definition of Function

Function name

Function parameters

```
float f_score(par1);
```

Return Type

University of Glasgow

Accessing the Elements of Arrays

- Array
  - Group of consecutive memory locations
  - Same name and type
- To refer to an element, specify
  - First part : Array name
  - Second part: [Position number/index/subscript]
- Format:

  arrayname **[** *position number* **]**

  - First element at position **0**
  - **n** element array named **c**:

  c[0], c[1]…c[n - 1]

Name of array (Note that all elements of this array have the same name, **c**)

c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	78

Position number of the element within array **c**

Common Error while Accessing the elements of Array

- Common Error
  - It is a common error to assume that arrays starts from 1
  - Such error is also referred to as off-by-one error
  - Remember Arrays are Zero-based
  - score[3] access the 4th element of the array.

Accessing Arrays using Expressions

- Array elements are like normal variables

```
c[ 0 ] =  3;
printf( "%d", c[ 0 ] );
```

Or

```
X = c[6]/2;
```

  – Perform operations in subscript. If **x** equals **3**

```
c[ 5 - 2 ]
is equivalent to
c[ 3 ]
is equivalent to
c[ x ]
```

All same !!!

- If b = [2, 4, 1], x =2, b[b[x]] = ?

Properties of Arrays

- Structures of related data items
- Static entity – same size throughout program
- In C an array is a data structure containing a number of items of the same type
- Used to store a collection of data with a common name, but it is often more useful to think of an array as a collection of variables of the same type
  - E.g., int, double, char, …
- Array name can be a mixture of letters, digits and underscores, but can't begin with a digit

Naming of Arrays

Valid	Invalid
cnt	won@last
_myPtr	3_times
go4it	char
testCase	first name
_very_nice	

Variable name is case sensitive: testCase ≠ testcase
The longest variable name can have 31 characters, after that will be omitted by the compiler
e.g.,
my_name_is_lei_zhang_and_i_am_from_glasgow
= my_name_is_lei_zhang_and_i_am_f

Value Assignment in Arrays

Square braces

Curly braces

- Initializers

```
int n[ 5 ] = { 10, 12, 19, 241, 32 }; → n[0]=10; n[1]=12;
  n[2] =19; n[3] = 241; n[4] = 32;
```

  – If not enough initializers, rightmost elements become **0**

```
int n[ 5 ] = { 4 }; → n[0]=4; n[1]=0; n[2] =0; n[3] = 0; n[4]
  = 0;
```

    - The first element is initialized as zero
    - The rest of the elements are initialized as zeros as well

  – If too many a syntax error is produced
  – E.g, `int n[ 5 ] = {10, 12, 19, 241, 32, 43};` --Syntax error

- If size omitted, initializers determine it

```
int n[ ] = { 1, 2, 3, 4, 5 };  -- good style!
```

  – 5 initializers, therefore 5 element array

Example 1: Initialization of Array Using Loops

```c
/* Author: Ahmed Zoha
   Purpose: The purpose of this program is to show how to initialize an array with loops
*/

#include <stdio.h>
#include <stdlib.h>


int main()
{
    int n[10], i; // Declaring an array n and an int variable i
    printf("%s%13s\n","Element","Value");
    for (i = 0; i <= 9; i++)  // Remember array's are zero based
    {
        n[i]= 0;
        printf("%7d%13d\n",i,n[i]);

    }

    return 0;
}
```

```
Element        Value
      0            0
      1            0
      2            0
      3            0
      4            0
      5            0
      6            0
      7            0
      8            0
      9            0

Process returned 0 (0x0)   execution time : 0.197 s
Press any key to continue.
```

Example 2: Value Assignment of Array using lists

```c
/* Author: Ahmed Zoha
   Purpose: The purpose of this program is to show how to initialize an array with lists
*/

#include <stdio.h>
#include <stdlib.h>


int main()
{
    int n[10] = {0,1,2,3,4,5,6,7,8,9}; // Initialization with the list
    int i;
    printf("%s%13s\n","Element","Value");
    for (i = 0; i <= 9; i++)  // Remember array's are zero based
    {
        printf("%7d%13d\n",i,n[i]);

    }

    return 0;
}

```

```
Element        Value
      0            0
      1            0
      2            0
      3            0
      4            0
      5            0
      6            0
      7            0
      8            0
      9            0

Process returned 0 (0x0)   execution time : 0.107 s
Press any key to continue.
```

Designated Initializers

- C99 added a new feature designated initializers
  - Allows you to pick and chose which elements to initialize
- Enclosing an element number in pair of brackets, specific array elements can be initialized in any order

```
float sample_data[500]= {[2] = 500.5, [1]= 300.0, [0]= 100.0};
```

This initializes the **sample_data array** 100.0, 500.5, 300.0 for the first three values

```
int arr[6] = {[5] = 212}; // Initializing arr[5] = 212
```

Example 3: Averaging Scores input by the User

- Requirements:
  - Ask the user to enter 10 scores
  - The code should stores these scores in an array, sum them and displays the average

main.c *main.c

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int scores[10];  // You initialize the score array by specifyin
    int count =10;   // Total number of values input by the user
    long sum = 0;    // Initialize the sum variable
    float average = 0.0f;   // Initialize the average variable

    printf("Enter the 10 scores: \n");

    int i;
    for (i=0; i < count; i++)
    {
        printf("%2u>:",i+1);
        scanf("%d",&scores[i]);  // Asking user to input the score
        sum += scores[i];        // This is equivalent to sum = sum + score[i]
    }
    average = (float)sum/count;  // Casting the float type to sum
    printf("Average of the 10 scores entered is: %.2f\n",average);  // Displaying the
    return 0;
```

C:\Zoha\Work\Teaching\2019\Course_IP_Sep2019_Dec2019\Code\Exampl

```
Enter the 10 scores:
 1>:10
 2>:10
 3>:10
 4>:10
 5>:10
 6>:10
 7>:10
 8>:10
 9>:10
10>:10
Average of the 10 scores entered is: 10.00

Process returned 0 (0x0)   execution time : 17.447 s
Press any key to continue.
```


University of Glasgow

Example 3: Out of Bound Error

- Try changing the < to <= in the for loop of the previous program (although the program compiles but it outputs garbage results)

```
int i;
for (i=0; i <= count; i++)
{
    printf("%2u>:",i+1);
    scanf("%d",&scores[i]);   // Asking use
    sum += scores[i];         // This is eq
}
```

C:\Zoha\Work\Teaching\2019\Course_IP_Sep2019_Dec2019\Code\Exam

```
Enter the 10 scores:
 1>:10
 2>:10
 3>:10
 4>:10
 5>:10
 6>:10
 7>:10
 8>:10
 9>:10
10>:10
11>:10
Average of the 10 scores entered is: 11.00

Process returned 0 (0x0)   execution time : 10.764 s
Press any key to continue.
```

Example 4: Summarizing Poll Results

Forty students were asked to rate the quality of the food in the student cafeteria on a scale of 1 to 10 (1 means awful and 10 means excellent). Place the 40 responses in an integer array and summarize the results of the poll.

```
#include <stdio.h>
#define RESPONSE_SIZE 40
#define FREQUENCY_SIZE 11

int main()
{
   int answer, rating, frequency[ FREQUEN
   int responses[ RESPONSE_SIZE ] =
      { 1, 2, 6, 4, 8, 5, 9, 7, 8, 10
        1, 6, 3, 8, 6, 10, 3, 8, 2
        6, 5, 7, 6, 8, 6, 7, 5
        5, 6, 7, 5, 6, 4, 8
                                  , 10 };

   for ( answer = 0; answer <= RESPONSE_SIZE - 1; a
      ++frequency[ responses [ answer ] ];

   printf( "%s%17s\n", "Rating", "Frequency" );

   for ( rating = 1; rating <= FREQUENCY_SIZE - 1;
      printf( "%6d%17d\n", rating, frequency[ ratin

   return 0;
}
```

```
int a;
a = responses [answer];
 frequency
```

"C:\Users\lz21g\Desktop\Hello world\

Rating	Frequency
1	2
2	2
3	2
4	2
5	5
6	11
7	5
8	7
9	1
10	3

Passing Arrays to Functions

- Passing arrays
  - To pass an array argument to a function, specify the name of the array without any brackets

```
int myArray[ 24 ];

myFunction( myArray, 24 );
```

    - Array size usually passed to function
  - Arrays passed call-by-reference
  - Name of array is address of first element
  - Function knows where the array is stored
    - Modifies original memory locations
- Passing array elements
  - Passed by call-by-value
  - Pass subscripted name (i.e., **`myArray[ 3 ]`**) to function

Passing Arrays to Functions

- Function prototype

```
void modifyArray( int b[], int arraySize );
```

  - Parameter names optional in prototype
    - **`int b[]`** could be written **`int []`**
    - **`int arraySize`** could be simply **`int`**

Example - Passing Arrays to Functions

- Passing arrays and individual array elements to functions

```
#include <stdio.h>
#define SIZE 5

void modifyArray( int [], int );  /* appears strange */
void modifyElement( int );

int main(){
    int a[ SIZE ] = { 0, 1, 2, 3, 4 }, i;

    printf( "Effects of passing entire array call ”
            "by reference:\n\nThe values of the ”
            "original array are:\n" );

    for ( i = 0; i <= SIZE - 1; i++ )
        printf( "%3d", a[ i ] );

    modifyArray( a, SIZE );  /* passed call by reference */

    printf( "The values of the modified array are:\n" );
    for ( i = 0; i <= SIZE - 1; i++ )
        printf( "%3d", a[ i ] );
```

Entire arrays passed call-by-reference, and can be modified

Example - Passing Arrays to Functions

Call-by-value, element value will be not changed

```
    printf( "\n\n\nEffects of passing array el
            "by value:\n\nThe value of a[3] is %d\n", a[ 3 ] );

    modifyElement( a[ 3 ] );
    printf( "The value of a[ 3 ] is %d\n", a[ 3 ] );
    return 0;
}

void modifyArray( int b[], int size )
{
    int j;
    for ( j = 0; j <= size - 1; j++ )
        b[ j ] *= 2;
}

void modifyElement( int e )
{
    printf( "Value in modifyElement is %d\n", e *= 2 );
}
```

Example - Passing Arrays to Functions

- When the above code is compiled and executed, it produces the following result:

```
Effects of passing entire array call by reference:

The values of the original array are:
  0  1  2  3  4
The values of the modified array are:
  0  2  4  6  8

Effects of passing array element call by value:


The value of a[3] is 6
Value in modifyElement is 12
The value of a[3] is 6
```

Sorting (排序) Arrays

- Sorting data
  - Important computing application
  - Virtually every organization must sort some data
- Bubble sort (sinking sort)
  - Several passes through the array
  - Successive pairs of elements are compared
    - If increasing order (or identical), no change
    - If decreasing order, elements exchanged
  - Repeat

The bubble Sort

- The bubble sort works as described in the following instructions:

    1) start at the top of the array
    2) look at the first and second elements
    3) are they the wrong way round? if yes, swap them around.
    4) look at the second and third elements
    5) are they the wrong way round? if yes, swap them around.
    6) repeat until you reach the end of the array
    7) go back to step one.

- The procedure is finished when the whole array has been gone through without a single change being made

- This is all very well, but how do we implement the instructions within a C program?

The bubble Sort

0	1	2	3	4	5	6	7	8
23	17	5	90	12	44	38	84	77

exchange

17	23	5	90	12	44	38	84	77

exchange

17	5	23	90	12	44	38	84	77

ok exchange

17	5	23	12	90	44	38	84	77

exchange

17	5	23	12	44	90	38	84	77

exchange

17	5	23	12	44	38	90	84	77

exchange

17	5	23	12	44	38	84	90	77

exchange

17	5	23	12	44	38	84	77	90

The largest value 90 is at the end of the list.

Animated Example of Bubble Sort

6 5 3 1 8 7 2 4

The bubble Sort

- Try compiling and running this program, it does work.
- There are three main areas of the program, reading the data, sorting the data and displaying the data.
- Note the use of the variable *swap* to indicate when a swap has occurred.
- Also note the use of the variable *temp* to temporarily store a score while they are swapped around.
- It is also worth noting that in the sort loop we only loop up to 9 not 10. This is because we are comparing a value with the one after it, so when we reach the second to last value we have finished the loop.

```c
#include <stdio.h>

#define number_of_students 10

int main()
{
    int score[number_of_students];
    int i,swaps,temp;

    printf("Student mark sorter\n");

    for(i=0;i<number_of_students;i++)
        {
            printf("Score for student %d :",i+1);
            scanf("%d",&score[i]);
        }

    printf("\nNow sorting data!");

    do
        {
            swaps=0;
            for(i=0;i<9;i++)
                {
                    if(score[i]>score[i+1])
                    {
                        temp=score[i+1];
                        score[i+1]=score[i];
                        score[i]=temp;
                        swaps=1;  /*a swap has taken place*/
                    }
                }
        }
    while(swaps !=0);
    printf("\nThe sorted student scores are:\n\n");

    for(i=0;i<number_of_students;i++)
        {
            printf("%d\n",score[i]);
        }

    return 0;
}
```

Reading

Sorting

Displaying

University of Glasgow

The bubble Sort

- When the above code is compiled and executed, it produces the following result:

```
Student mark sorter
Score for student 1 :67
Score for student 2 :58
Score for student 3 :40
Score for student 4 :25
Score for student 5 :86
Score for student 6 :77
Score for student 7 :92
Score for student 8 :14
Score for student 9 :9
Score for student 10 :42

Now sorting data!
The sorted student scores are:

9
14
25
40
42
58
67
77
86
92
```

The bubble Sort (Live Demo)

- Using For loop

```c
#define SIZE 10

int main()
{
   int a[ SIZE ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
   int i, pass, hold;

   printf( "Data items in original order\n" );

   for ( i = 0; i <= SIZE - 1; i++ )
      printf( "%4d", a[ i ] );

   for ( pass = 1; pass <= SIZE - 1; pass++ )  /* passes */

      for ( i = 0; i <= SIZE - 2; i++ )       /* one pass */

         if ( a[ i ] > a[ i + 1 ] ) {  /* one comparison */
            hold = a[ i ];                  /* one swap */
            a[ i ] = a[ i + 1 ];
            a[ i + 1 ] = hold;
         }

   printf( "\nData items in ascending order\n" );

   for ( i = 0; i <= SIZE - 1; i++ )
      printf( "%4d", a[ i ] );

   printf( "\n" );

   return 0;
}
```