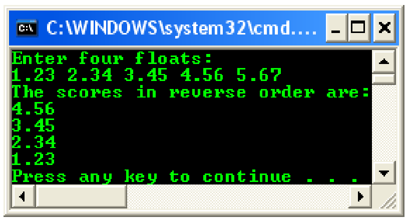
To:
Tenouk

C LAB WORKSHEET 9
C & C++ Array Data Type Part 1

1. The C & C++ array data type.
2. Array declaration and initialization.
3. Tutorial references that should be used together with this worksheet are array part 1 and array part 2.

- Array is another C/C++ data type. It is an aggregated same data type. Let start with a program example that doesn't use loop to see array usefulness.
- Create a project named **myarray** and add a source file named **myarraysrc**. Set your project to **Compile as C Code**. Build and run the following sample program and that follow.

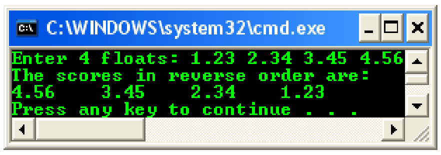
```
#include <stdio.h>

void main()
{
      float score1, score2, score3, score4;
      printf("Enter four floats: \n");
      scanf_s("%f", &score1);
      scanf_s("%f"", &score2);
      scanf_s("%f", &score3);
      scanf_s("%f", &score4);
      printf("The scores in reverse order are: \n");
      printf("%.2f\n", score4);
      printf("%.2f\n", score3);
      printf("%.2f\n", score2);
      printf("%.2f\n", score1);
}
```



```
Enter four floats:
1.23 2.34 3.45 4.56 5.67
The scores in reverse order are:
4.56
3.45
2.34
1.23
Press any key to continue . . .
```

- In this program, game scores for four individuals are read in and they are printed in reverse order. Imagine how much more we have to repeat the monotonous steps if scores are in thousands. Just reading and writing like that would take thousand lines let alone doing other processing, such as finding the highest score.
- Instead, consider the solution using arrays shown below. Notice that although the looping structures are slightly more complicated to code, there are fewer lines of coding. Also, if we wanted to extend the program to handle 1000 scores, then only the #define statement would need to be changed, only one statement. The saving in the number of programming lines would also be evident every time we need to process the scores any further.

```
#include <stdio.h>
#define SIZE 4  // the array's size

void main()
{
      float score[SIZE];
      int i;
       // loop to read in the scores into the array
      printf("Enter %d floats: ", SIZE);
      for(i = 0; i <= (SIZE - 1); i = i + 1)
            scanf_s("%f", &score[i]);
      // loop to write out the scores from the array
      printf("The scores in reverse order are: \n");
      for(i = SIZE - 1; i >= 0; i = i - 1)
            printf("%.2f\t", score[i]);
      printf("\n");
}
```



```
Enter 4 floats: 1.23 2.34 3.45 4.56
The scores in reverse order are:
4.56    3.45    2.34    1.23
Press any key to continue . . .
```

- Arrays are identified easily by the **brackets** following their **names**, both in their declaration and in their usage. Here, score[] is an array. The variable score[] has brackets after it in its declaration. The number in the brackets is 4 because the value of SIZE is 4. This means that score has four elements associated with it. We can call them slots. Furthermore, since the array is declared as a float, score can store floating points in each of these slots. Take note that, with strings a null character at the end signals where the last character is stored. In the previous example you can see that a variable (SIZE) may be used to hold the number of elements present in the array. Take note also about the counting of the elements in arrays start at 0 and not 1. For example:

  score[4];

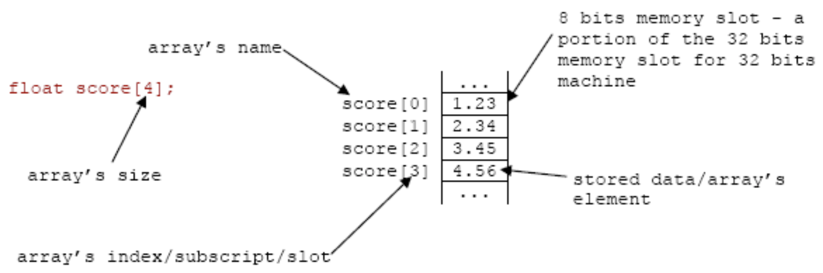
- The counting of the element, the index started at 0. So the score[4] array variable will be something like the following:

  score[0], score[1], score[2], score[3]

- But in typical array representation, the following array statements look similar, so be careful. It can be the array size (normally in array declaration) or array index (when used in program). The first one is the declaration of an array named score with size of 2 and the second one is the array element with index number 3.

  int score[2] and score[2]

- Then you have to be careful with this issue. You must differentiate between **array index**, that start from 0 and the **array size** that is the real array size. The formal name for a **slot number** is **index**. Its plural is either **indices** or **indexes**. Some books use **subscript**. The highest index used with an array should always be one less than its size. For the previous example, the highest index used with score should be 3, which is one less than its size (4 – 1 = 3). All the information of an array for the previous example can be depicted in the following simplified Figure.

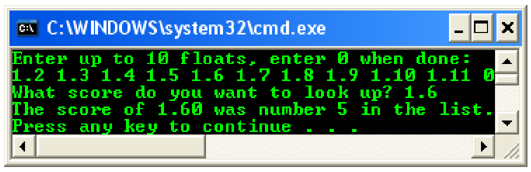

float score[4];
array's name
array's size
score[0]
score[1]
score[2]
score[3]
...
1.23
2.34
3.45
4.56
...
8 bits memory slot - a portion of the 32 bits memory slot for 32 bits machine
stored data/array's element
array's index/subscript/slot

Searching An Array Element

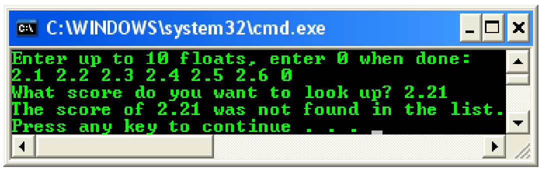
- Let take a look at another example of array program, searching an array element. Build and run the following program.

```
#include <stdio.h>

void main()
{
     // an array to store a list of scores
     float score[12],
     // the array is searched for this score
     lookup;
     // used as an index to step through the array
     int i,
     // the index of the last score stored in the array
     max;
     // read in the scores into the array
     printf("Enter up to 10 floats, enter 0 when done: \n");
     scanf_s("%f", &score[0]);
     for(i = 0; score[i] != 0; i = i + 1)
          scanf_s("%f", &score[i + 1]);
     max = i - 1;
     // read a score to be searched in the array
     printf("What score do you want to look up? ");
     scanf_s("%f", &lookup);
     // search the array for this score
     for(i = 0; i <= max; i = i + 1)
          if(score[i] == lookup)
          // abandon the loop if element is found
          break;
     // if it was found, then a break was executed and "i" was <= "max"
     if(i <= max)
          printf("The score of %.2f was number %d in the list.\n", lookup, i + 1);
     // otherwise, "i" went past the value of "max".
     else
          printf("The score of %.2f was not found in the list.\n", lookup);
}
```

A sample input and output.



```
Enter up to 10 floats, enter 0 when done:
1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 1.10 1.11 0
What score do you want to look up? 1.6
The score of 1.60 was number 5 in the list.
Press any key to continue . . .
```



```
Enter up to 10 floats, enter 0 when done:
2.1 2.2 2.3 2.4 2.5 2.6 0
What score do you want to look up? 2.21
The score of 2.21 was not found in the list.
Press any key to continue . . .
```

- In this example, the array score[ ] has 12 slots/index (0 – 11), which assigned values from user input until 0 is read in. Therefore the loop to read data into the array is different from the previous example. There we had a fixed number of elements to read. Here we stop reading data into the array when we read in a zero.
- The program continues by asking the user what number he/she wants to look up. If the number is found, then it will print where the number was located. If the number isn't found, it will notify the user. The method of doing this search is called a **linear search**, where the number is searched starting from the beginning of the array to the end of it. If the number doesn't appear until the end of the array, this may take some time. With a lot of elements, this is not an efficient one. A more efficient method of searching is called a **binary search** introduced later in this worksheet.

Array Initialization

- Some examples of the array initialization are given below.

```
char myname[ ] = "Mike";
char yourname[5] = {'M', 'i', 'k', 'e', '\0'};
int score[5] = {4, 2, 654, 0, 78};
```

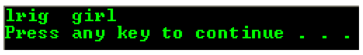
- Individual elements can be specified using braces and commas, as with score[5] and yourname[5]. Two strings, yourname and myname are initialized to the same value. When initializing arrays as above, one can omit the array size given in the brackets, as in myname[ ] (unsized array). This is because the size of the array can be determined by the number of initial values. Also remember that a string such as "Mike" actually contains five characters because the **null character** is used to **terminate a string** and this is specific to string only.

More Practice

- Arrays allow programmers to **group related items of the same data type in one variable**. However, when referring to an array, one has to specify not only the array or variable name but also the index number of interest. Let us first look at strings that are special types of arrays, since we are more familiar with them. Build and run the following program then try answering the questions that follow.

```
#include <stdio.h>

void main()
{
     char a[11] = "Boring";
     printf("Index 0 has %c\n", a[0]);
     printf("Index 1 has %c\n", a[1]);
     printf("Index 2 has %c\n", a[2]);
     printf("Index 3 has %c\n", a[3]);
     printf("Index 4 has %c\n", a[4]);
     printf("Index 5 has %c\n", a[5]);
     printf("Index 6 has %c\n", a[6]);
     printf("Index 7 has %c\n", a[7]);
     printf("Numerically, the a[6] is %d\n", a[6]);
}
```

a. How many slots/indexes does this array have?
b. How many characters, counting the null character at the end, does this array hold?
c. The number of an index is also called an index. What is the lowest index?
d. What is the highest index for this array?
e. Is the number for the highest index the same as that for the number of indexes?
f. What is stored in the last index as a character, that is, in an index number 5?
g. What is stored in the last index numerically? All **strings** should have this null character stored in its last index to signal the end of the string.

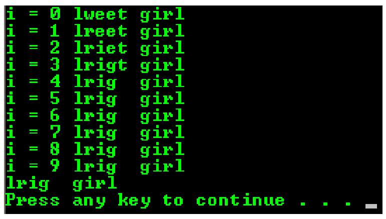


```
1rig  girl
Press any key to continue . . .
```

a. 11.
b. 7 characters including the null character.
c. 0.
d. 10.
e. No. The number of indexes are 11 but the highest index is 10 because the index counting starts from 0 instead of 1.
f. A 'g' character.
g. A null character.

```
#include <stdio.h>

void main()
{
     char a[11] = "sweet girl";
     int i;
     for(i = 0; i <= 10; i = i + 1)
          printf("Index %d has %c\n",  i, a[i]);
}
```

a. When we wanted to print the index or the number of the index, did we print i or a[i]?
b. When we wanted to print what was stored in an index, did we print i or a[i]?
c. Are i and a[i] the same?
d. Which of the above is the value stored in an index?
e. Which is the index number itself?



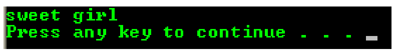
```
i = 0 1weet girl
i = 1 1reet girl
i = 2 1riet girl
i = 3 1rigt girl
i = 4 1rig  girl
i = 5 1rig  girl
i = 6 1rig  girl
i = 7 1rig  girl
i = 8 1rig  girl
i = 9 1rig  girl
1rig  girl
Press any key to continue . . .
```

a. We print i.
b. a[i] in this case.
c. No. i is the index but a[i] represent the data stored.
d. a[i].
e. i.

- In the following example, the loop goes up to only "i = 9".

```
#include <stdio.h>

void main()
{
     char a[11] = "sweet girl";
     int i;
     for(i = 0; i <= 9; i = i + 1)
          printf("%c", a[i]);
     printf("\n");
}
```

a. Is the "\n" printed inside the loop or after it? Why?
b. If it is printed inside the loop, how would you put it after?
c. If it is printed after the loop, how would you put it inside?
d. Can you remember an easy way to write out a string using one printf() and no loop?



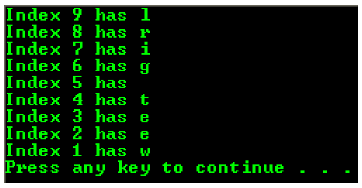
```
sweet girl
Press any key to continue . . .
```

a. After or outside the loop because the for body doesn't have curly braces, { }.
b. No. We can't put it after the loop to print it inside the loop.
c. No. We can't put it inside the loop to print it after the loop.
d. Just put the string as a literal sting or as an argument in the printf() such as printf("sweet girl");

- Complete the for loop in the following program so that the characters of the array are printed in reverse order, starting from index number 10 down to slot number 0?

```
#include <stdio.h>

void main()
{
     char a[11] = "sweet girl";
     int i;

     for(_____; __________________; ________)
          printf("Index %d has %c\n", i, a[i]);
}
```

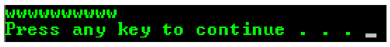
```
#include <stdio.h>

void main()
{
     char a[11] = "sweet girl";
     int i;

     for(i = 9; i > 0; i = i - 1)
        printf("Index %d has %c\n", i, a[i]);
}
```


```
Index 9 has l
Index 8 has r
Index 7 has i
Index 6 has g
Index 5 has
Index 4 has t
Index 3 has e
Index 2 has e
Index 1 has w
Press any key to continue . . .
```

- Remember that in an assignment statement, the **value on the right side of the equal sign is stored in the variable shown on its left side**. Build and run the following program.

```
#include <stdio.h>

void main()
{
      char a[11] = "sweet girl";
      int i;

      for(i = 0; i <= 9; i = i + 1)
            a[i] = a[1]; // notice the 'i' and '1'
      printf("%s\n", a);
}
```

  a. What was stored in each index of the array?
  b. If the array were initialized to “good boy”, what would have been printed?

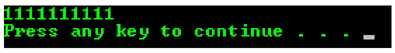

```
wwwwwwwwww
Press any key to continue . . .
```

a. A 'w' character.
b. A 'o' character.

- Next, try the following program.

```
#include <stdio.h>

void main()
{
      char a[11] = "sweet girl";
      int i;

      for(i = 0; i <= 9; i = i + 1)
            a[i] = '1';
      printf("%s\n", a);
}
```

  a. What was stored in each index of the array?
  b. If the array were initialized to “good boy”, what would have been printed?

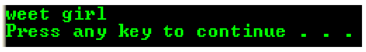

```
1111111111
Press any key to continue . . .
```

a. A '1' character.
b. A '1' character.

- First run the following program with the first printf() is commented out. Then answer the questions. Finally check your answer by uncomment the first printf() and rerun the program.

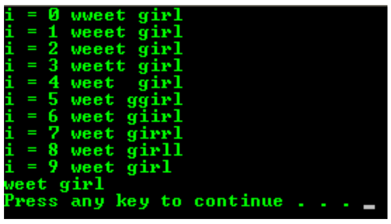
```
#include <stdio.h>

void main()
{
      char a[11] = "sweet girl";
      int i;
      for(i = 0; i <= 9; i = i + 1)
      {
            a[i] = a[i + 1];
            // printf("i = %d %s\n", i, a);
      }
      printf("%s\n", a);
}
```

  a. When i was 0, which index of a was changed? To which value?
  b. When i was 1, which index of a was changed? To which value?
  c. When i was 2, which index of a was changed? To which value?
  d. When i was 9, which index of a was changed? To which value?


```
weet girl
Press any key to continue . . .
```

a. Index 0 was changed to 1. a[0+1] = a[1]. When index is 0, assign a[1] to a [0].
b. Index 1 was changed to 2. a[1+1] = a[2]. When index is 1, assign a[2] to a [1].
c. Index 2 was changed to 3. a[2+1] = a[3]. When index is 2, assign a[3] to a [2].
d. Index 9 was changed to 10. a[9+1] = a[10]. When index is 9, assign a[10] to a[9].

When the first printf() uncommented the following is the output.


```
i = 0 wweet girl
i = 1 weeet girl
i = 2 weeet girl
i = 3 weett girl
i = 4 weet  girl
i = 5 weet ggirl
i = 6 weet giirl
i = 7 weet girrl
i = 8 weet girll
i = 9 weet girl
weet girl
Press any key to continue . . .
```