International University for Science & Technology (IUST)

College of Information Technology

Assignment 4

Student Name	
Year/Group	2009/2010

Assignment title:	**Introduction to variables**

Unit title:	Computer Skills II (C Language)	**Subject Tutor:**	Mr. mhd. Mazen al-Mustafa
Start:		**Coordinator** Mr. mhd. Mazen al-Mustafa	
Submission:			
Actual Submission date:		**Grading.**	

Learning Outcomes covered:

- **Array**
- **Passing array to Function**

Resources:

- C How To Program fifth Edition, by H.M. Deitel & P.J. Deitel,
- Teacher's handouts
- internet

True/False

Consider the following declarations as you determine if each of assignment statements 1 - 7 is valid. Answer True if the statement is valid, False otherwise.

```
#define HALF_CENT 50
#define A_SIZE 26

char a_list[HALF_CENT], b_list[HALF_CENT], a_char = 'v';
int  nums[A_SIZE], vals[A_SIZE], i = 1;
```

1. `nums[0] = nums[25];` [True]

2. `nums = vals + 1;` [False]

3. `a_list[50] = 'd';` [False]

4. `b_list[30] = 0.37 * vals[1];` [False]

5. `a_list = b_list;` [False]

6. `nums[5] = (int)a_char;` [True]

7.
```
for  (i = 1;  i <= A_SIZE;  ++i)
    nums[A_SIZE - i] = i;
```
[True]

8. If b is an array with type int elements, then the statement

```
b += 5;
```
adds five to each element of b. [False]

9. If b is an array with type int elements and the value of b[4] is 3, then the statement

```
printf("%d\n", b[b[4] - 1]);
```

displays one less than the value of b[3]. [False]

10. If b is an array of integer elements, then the statement

```
b[3] *= 2;
```

doubles the value of b[3]. [True]

Multiple Choice

1. What value is returned by function result?

```
int     result(const int a[], int n)
{
    int i, r;

    r = 0;
    for  (i = 1;  i < n;  ++i)
        if (a[i] > a[r])
            r = i;

    return r;
}
```

*a. The subscript of the largest of the first n elements of array a.
b. The value of the largest of the first n elements of array a.
c. The subscript of the smallest of the first n elements of array a.
d. The value of the smallest of the first n elements of array a.
e. The subscript of the last element greater than its predecessor within the first n elements of array a.

For Questions 2 - 6 assume the following environment.

```
#define MAX 50
int a[MAX], i, j, temp;
```

2. What is the effect of this program segment?

```
for  (i = 0;  i < MAX / 2;  ++i) {
    temp = a[i];
    a[i] = a[MAX - i - 1];
    a[MAX - i - 1] = temp;
}
```

a. Arranges the elements of array a in ascending order.
b. Counts the number of elements of a greater than its first element.
*c. Reverses the numbers stored in the array.
d. Puts the largest value in the last array position.
e. None of the above.

3. What is the effect of the following program segment?

```
for  (i = 0;  i < MAX - 1;  ++i)
    if (a[i] > a[i + 1]) {
         temp = a[i];
         a[i] = a[i + 1];
         a[i + 1] = temp;
    }
```

a. Arranges the elements of array a in ascending order.
b. Counts the number of elements of a greater than its first element.
c. Reverses the numbers stored in the array.
*d. Puts the largest value in the last array position.
e. None of the above.

4. What is the effect of the following program segment?

```
temp = 0;
for  (i = 1;  i < MAX;  ++i)
    if (a[i] > a[0])
         ++temp;
```

a. Arranges the elements of array a in ascending order.
*b. Counts the number of elements of array a greater than its initial element.
c. Reverses the numbers stored in the array.
d. Puts the largest value in the last array position.
e. None of the above.

5. What is the maximum valid subscript value for array a?

a. 0
*b. 49
c. 50
d. a[50]
e. none of the above

6. What is the minimum valid subscript value for array a?
*a. 0
b. 1
c. any negative number
d. There is no minimum.
e. none of the above

7. When a C program passes an array as a function argument,

 a. the value of the initial element of the array is actually passed.
 *b. the address of the initial element of the array is actually passed.
 c. the entire array is copied into the function's data area.
 d. the addresses of the initial and final elements of the array are copied into the function's data area.
 e. none of the above.

8. How would you best describe the purpose of the following code?

```
f = 0;
for  (i = 1;  i < N;  ++i)
    if (a[i] >= a[f])
        f = i;
```

 a. Rearrange the first N components of array a in descending order.
 b. Rearrange the first N components of array a in ascending order.
 c. Place the largest component of array a in position N.
 d. Compute the value of the largest component in array a.
 *e. Determine the subscript of the last occurrence of the largest of the first N components of array a.

9. How many numbers can be stored in the array declared below?

```
double arr[10][5][6];
```

 a. 21
 b. 90
 c. 180
 *d. 300
 e. none of the above

10. Which one of the conditions that follows will be false (value of 0) after execution of the program segment below?

```
int v[5] = {0, 0, 0, 0, 1};
int k, j;
for  (j = 3;  j >= 0;  --j)
    for  (k = j;  k < 4; ++k)
        v[k] += v[k + 1];
```

```
  a. v[0] == v[4]
  b. v[1] == v[3]
  c. v[0] < v[1]
  d. v[1] < v[2]
 *e. v[2] < v[3]
```

Short Answer

1. What will be the values of k[1] and k[3] after execution of the code segment below using the data shown?

```
int k[6] = {0, 0, 0, 0, 0, 0};              Data:  2  0  1
int i, n;

for  (i = 3;  i < 6;  ++i) {
    scanf("%d", &n);
    k[n] = i;
}
```

```
[Answer:  k[1] is 5;  k[3] is 0 ]
```

2. Processing array elements in order from first to last is called [sequential] access. When the order in which array elements are accessed is not predictable, one is using [random] access.

For Questions 3 - 9, refer to the declarations and initializations below. Indicate whether each of statements 3 - 7 is valid. If the statement is valid, indicate what value is displayed or assigned. If the statement is invalid, explain why.

```
            double x[8] = {16.0, 12.0, 6.0, 8.0, 2.5, 12.0, 14.0, -
54.5};
            int    j = 5;
```

3. `printf("%.2f\n", x[j] + 1);[Answer:  Valid; 13.00 is displayed.]`

4. `printf("%.2f\n", x[j + 1]);[Answer:  Valid; 14.00 is displayed.]`

5. `printf("%.2f\n", x[j * j]);[Answer:  Invalid; subscript out of range.]`

6. `printf("%.2f\n", x[(int)x[j - 1]]);[Answer:  Valid; 6.00 is displayed.]`

7. `x[2 * j - 3] = x[j / 6];[Answer:  Valid;  x[7] is assigned 16.0.]`

8. `What is the data type of one element of array x?[Answer:  double]`

9. `List all the values that are valid subscripts of array x.`

   `[Answer:  0, 1, 2, 3, 4, 5, 6, 7]`

10. Write the prototype of a void function named array_fixer that takes four parameters:
    out_list - a one-dimensional array output parameter; array values are type double
    in_list - a one-dimensional array input parameter; array values are type double
    n - an integer input parameter
    flagp - an integer output parameter

```
[Answer:
        void array_fixer(double out_list[], const double in_list[],
int n, int *flagp);
```

Test your self

1.Write a function named "`ReadArray`" that takes as its arguments the following:

- an array of floating point values;
- an integer that tells how many floating point values are in the array.

The function must read the elements of the array.
Thus, for example, if the array that's passed to the function looks like this:

```
0      1      2      3      4
?? |   ?? |   ?? |   ?? |   ??
```

then when the function returns, the array will have been modified so that it looks like this:

```
0      1      2      3      4
7.1 | 3.4 | 9.0 | 2.6 | 5.8
```

The function should not return any value.

2.Write a function named "`WriteArray`" that takes as its arguments the following:

- an array of floating point values;
- an integer that tells how many floating point values are in the array.

The function must print the elements of the array. The function should not return any value

3.Write a function named "`SortArray`" that takes as its arguments the following:

- an array of floating point values;
- an integer that tells how many floating point values are in the array.

The function must sort the values in the array.
Thus, for example, if the array that's passed to the function looks like this:

```
0     1     2     3     4
5.8 | 2.6 | 9.0 | 3.4 | 7.1
```

then when the function returns, the array will have been modified so that it looks like this:

```
0     1     2     3     4
2.6 | 3.4 | 5.8 | 7.1 | 9.0
```

The function should not return any value.

4.Write a function named "`linearSearchArray`" that takes as its arguments the following:

- an array of floating point values;
- an integer that tells how many floating point values are in the array.
- The value (key) that this function must find it in the array.

The function must use the `linear Search` method to find the key in the array.

Thus, for example, if the array that's passed to the function looks like this:

```
0     1     2     3     4
5.8 | 2.6 | 9.0 | 3.4 | 7.1
```

and the key is 9.0 then the function returns 2, otherwise the function print the value is not found.

5.Write a function named "`BinarySearchArray`" that takes as its arguments the following:

- an array of floating point values;
- an integer that tells how many floating point values are in the array.
- The value (key) that this function must find it in the array.

The function must use the `Binary Search` to find the key in the array.

Thus, for example, if the array that's passed to the function looks like this:

```
0     1     2     3     4
5.8 | 2.6 | 9.0 | 3.4 | 7.1
```

and the key is 9.0 then the function returns 2, otherwise the function print the value is not found.
Note: use the previous function (`SortArray`) to sort your array before using `Binary Search` method

6.Write a function named "`MultiplicationBetweenToArrays`" that takes as its arguments the following:

- an first array of floating point values;
- an second array of floating point values
- an empty third array
- an integer that tells how many floating point values are in each array (consider that the three arrays have the same size).

The function must multiply first array by second array and assign the result to third array. Thus, for example, if the first array looks like this:

```
0     1     2     3     4
5.8 | 2.6 | 9.0 | 3.4 | 7.1
```

the first array looks like this:

```
0     1     2     3     4
1.0 | 2.0 | 3.0 | 4.0 | 5.0
```

then when the function returns, the array will have been modified so that it looks like this:

```
0      1      2      3       4
27.9 | 55.8 | 83.7 | 111.6 | 139.5
```

The function should not return any value.

7.Write a function named "`sum`" that takes as its arguments the following:

- an array of floating point values;
- an integer that tells how many floating point values are in the array.

The function should return as its value the sum of the floating point values in the array. Thus, for example, if the array that's passed to the function looks like this:

```
0     1     2     3     4
5.8 | 2.6 | 9.0 | 3.4 | 7.1
```

then the function should return the value 27.9 as its value.

8.Write a function named "`Average`" that takes as its arguments the following:

- an array of floating point values;
- an integer that tells how many floating point values are in the array.

The function should return as its value the average of the floating point values in the array. Thus, for example, if the array that's passed to the function looks like this:

```
0     1     2     3     4
5.8 | 2.6 | 9.0 | 3.4 | 7.1
```

then the function should return the value 5.58 as its value.

9.Write a function named "Maximum" that takes as its arguments the following:

- an array of floating point values;
- an integer that tells how many floating point values are in the array.

The function should return as its value the maximum element in the array. Thus, for example, if the array that's passed to the function looks like this:

```
0     1     2     3     4
```

```
5.8 | 2.6 | 9.0 | 3.4 | 7.1
```
then the function should return the value 9.0 as its value.

10. Write a function named "`Minimum`" that takes as its arguments the following:
- an array of floating point values;
- an integer that tells how many floating point values are in the array.

The function should return as its value the minimum element in the array. Thus, for example, if the array that's passed to the function looks like this:
```
0     1     2     3     4
5.8 | 2.6 | 9.0 | 3.4 | 7.1
```

11. Write a function named "`reverse`" that takes as its arguments the following:
- an array of floating point values;
- an integer that tells how many floating point values are in the array.

The function must reverse the order of the values in the array. Thus, for example, if the array that's passed to the function looks like this:
```
0     1     2     3     4
5.8 | 2.6 | 9.0 | 3.4 | 7.1
```
then when the function returns, the array will have been modified so that it looks like this:
```
0     1     2     3     4
7.1 | 3.4 | 9.0 | 2.6 | 5.8
```
The function should not return any value.

then the function should return the value 2.6 as its value.
```
void reverse (float a[], int n)
{
int i = 0, j = n - 1;
while (i < j)
swap_floats (a[i++], a[j--]);
}
```

12. Write a function named "`location_of_largest`" that takes as its arguments the following:
- an array of integer values;
- an integer that tells how many integer values are in the array.

The function should return as its value the subscript of the cell containing the largest of the values in the array.
Thus, for example, if the array that's passed to the function looks like this:
```
0     1     2     3     4
58  | 26  | 90  | 34  | 71
```
then the function should return the integer 2 as its value. If there is more than one cell containing the largest of the values in the array, then the function should return the *smallest* of the subscripts of the cells containing the largest values. For example, if the array that's passed to the function is
```
0     1     2     3     4     5     6
```

```
58 | 26 |   91 |  34 |  70 |  91 |  88
```

then the largest value occurs in cells 2 and 5 , so the function should return the integer value 2 .

13. Write a function named "`location_of_target`" that takes as its arguments the following:
- an array of integer values;
- an integer that tells how many integer values are in the array;
- an integer "target value".

The function should determine whether the given target value occurs in any of the cells of the array, and if it does, the function should return the subscript of the cell containing the target value. If more than one of the cells contains the target value, then the function should return the largest subscript of the cells that contain the target value. If the target value does not occur in any of the cells, then the function should return the sentinel value –1.
Thus, for example, if the target value that's passed to the function is 34 and the array that's passed to the function looks like this:

```
0    1     2     3     4     5     6
58 | 26 |   91 |  34 |  70 |  34 |  88
```

then the target value occurs in cells 3 and 5 , so the function should return the integer value 5 .

14. Write a function named "`rotate_right`" that takes as its arguments the following:
- an array of floating point values;
- an integer that tells the number of cells in the array;

The function should shift the contents of each cell one place to the right, except for the contents of the last cell, which should be moved into the cell with subscript 0 . Thus, for example, if the array passed to the function looks like this:

```
0     1     2     3     4
5.8 | 2.6 | 9.1 | 3.4 | 7.0
```

then when the function returns, the array will have been changed so that it looks like this:

```
0     1     2     3     4
7.0 | 5.8 | 2.6 | 9.1 | 3.4
```

The function should not return a value.

```
void rotate_right (float a[], int n)
{
float temp = a[n-1]; // Hold the contents of the last cell.
int i;
for (i = n - 1; i >= 1; --i)
a[i] = a[i-1];
a[0] = temp;
}
```

15. Write a function named "`shift_right`" that takes as its arguments the following:
- an array of floating point values;
- an integer, call it "`left`", that tells the leftmost cell of the part of the array to be shifted;

- an integer, call it "`right`", that tells the rightmost cell of the part of the array to be shifted;
- a positive integer, call it "`distance`" that tells how many cells to shift by.

The function should make sure that `left` is less than or equal to `right`, and that `distance` is greater than zero. If either of these conditions fails, the function should return the value 1 to indicate an error. Otherwise it should shift by `distance` cells the contents of the array cells with subscripts running from `left` to `right`. Thus, for example, if the array passed to the function looks like this:

```
0     1      2     3     4     5     6     7     8      9     10    ....
5.8 | 2.6 | 9.1 | 3.4 | 7.0 | 5.1 | 8.8 | 0.3 | -4.1 | 8.0 | 2.7 | etc.
```

```
int shift_right (float a[], int left, int right, int distance)
{
if (left > right || distance <= 0)
return 1;
int i = right, // points to the cell to be shifted
j = i + distance; // points to the receiving cell
while (i >= left)
{ a[j] = a[i];
--i;
--j;
}
return 0;
}
```

```
int shift_right (float a[], int left, int right, int distance)
{
if (left > right || distance <= 0)
return 1;
int i;
for (i = right; i >= left; --i)
a[i + distance] = a[i];
return 0;
}
```

16. Write a function named "`subtotal`" takes as its arguments the following:

- an array of floating point values;
- an integer that tells the number of cells in the array.

The function should replace the contents of each cell with the sum of the contents of all the cells in the original array from the left end to the cell in question. Thus, for example, if the array passed to the function looks like this:

```
0      1     2     3     4
5.8 | 2.6 | 9.1 | 3.4 | 7.0
```

then when the function returns, the array will have been changed so that it looks like this:

```
0      1     2     3     4
5.8 | 8.4 | 17.5 | 20.9 | 27.9
```

because 5.8 + 2.6 = 8.4 and 5.8 + 2.6 + 9.1 = 17.5 and so on. Note that the contents of cell 0 are not changed. The function should not return a value.

```
void subtotal (float a[], int n)
{
int i;
for (i = 1; i < n; ++i)
a[i] += a[i-1];
}
```

17. Write a function named "`concatenate`" that copies the cells of one array into a larger array, and then copies the cells of another array into the larger array just beyond the contents of the first array. The contents of the cells will be integers.

The arguments will be as follows:

- the first array that will be copied;
- the number of cells that will be copied from the first array;
- the second array that will be copied;
- the number of cells that will be copied from the second array;
- the large array into which all copying will be performed;
- the number of cells available in the large array.

If the function discovers that the number of cells in the large array is not large enough to hold all the numbers to be copied into it, then the function should return 0 to indicate failure. Otherwise it should return 1.

The function should not alter the contents of the first two arrays. To take an example, if the first two arrays passed to the function look like this:

```
0     1     2     3     4     5     6                    0     1     2     3
58  | 26 |  91 |  34 |  70 |  34 |  88         and      29 |  41 |  10 |  66
```

then, provided the size of the large array is at least 11, the large array should look like this when the function returns:

```
0     1     2     3     4     5     6     7     8     9     10
58 |  26 |  91 |  34 |  70 |  34 |  88 |  29 |  41 |  10 |  66
```

```
int concatenate (const int a[], int m,
const int b[], int n,
int c[], int p)
{
if (m + n > p)
return 1;
int i, j;
for (i = 0; i < m; ++i)
c[i] = a[i];
for (j = 0; j < n; ++j)
c[i++] = b[j]; // Increment i and j after each
assignment
return 0;
}
```

18. Write a function named "`number_of_matches`" that compares the initial parts of two character arrays to see how many pairs of cells match before a difference occurs. For example, if the arrays are

```
0    1    2    3    4    5                        0    1    2    3    4
```

Error! and **Error!**

then the function should return the value 3 because only the first three pairs of cells in the arrays match (cell 0 matches cell 0, cell 1 matches cell 1, and cell 2 matches cell 2). Each of the character arrays will end with the character whose ASCII value is zero; this character, called NUL, is denoted by `'\0'` in the C programming languages (see the two arrays shown above). The pairwise cell comparisons should not go beyond the end of either array. If the two arrays are identical all the way to their terminating NUL characters, return the number of non-NUL characters. The function should take only two parameters, namely the two character arrays to be compared.

```
int number_of_matches (const char a[], const char b[])
{
const char NUL = '\0';
int i = 0; // i keeps track of how many pairs of cells match
while (a[i] != NUL && b[i] != NUL && a[i] == b[i])
++i;
return i;
}
```

Student Declaration: **I certify that the work contained in this assignment was researched and prepared by me:**

Signature: ____________________________________ Date: / /

Remark: separate feedback sheet will be returned to you after your work has been marked

Feedback Sheet

Student's Notes:
Assistant's Feed Back:

Assistant: ____________________________________

Signature: ____________________________________ Date: ____________________________