Jingru Dou

CS 241

HMWK #2

**From Chapter 7 of the textbook**  
**7.7 Answer each of the following:**  
(a) The &(address) operator returns its operand’s location in memory.  
(b) The \*(dereference) operator returns the value of the object to which its operand points.  
(c) To accomplish pass-by-reference when passing a nonarray variable to a function, it’s necessary to pass the address of the variable to the function.

**Exercise 7.8 State whether the following are true or false. If false, explain why.**  
(a) Two pointers that point to different arrays cannot be compared meaningfully. True  
(b) Because the name of an array is a pointer to the first element of the array, array names may be manipulated in precisely the same manner as pointers. False

The name of array can not change its value because is constant. So it is not the array names are not the same manner as pointers.

**Exercise 7.9 Answer each of the following. Assume that integers are stored in four bytes and that the starting address of the array is at location 1002500 in memory.**  
(a) Define a five-element int array values, and initialize the elements to the even integers from 2 to 10. Assume the symbolic constant SIZE is defined as 5.

unsigned int value[SIZE] = { 2, 4, 6, 8, 10 };

(b) Define a pointer vPtr that points to an object of type int.

unsigned int \*vPtr;

(c) Print the elements of array values using array subscript notation. Use a for statement and assume integer control variable i has been defined.

for (i = 0; i < SIZE; i++) {

printf("%d ", value[i]);

} // 2 4 6 8 10

(d) Give two separate statements that assign the starting address of array values to pointer variable vPtr.

vPtr = &value[0];

printf("%d\n", \*vPtr); //2

vPtr = value;

printf("%d\n", \*vPtr);//2

(e) Print the elements of array values using pointer/offset notation.

for (i = 0; i < SIZE; i++) {

printf("%d ", \*(vPtr + i));

} //2 4 6 8 10

(f) Print the elements of array values using pointer/offset notation with the array name as the pointer.

for (i = 0; i < SIZE; i++) {

printf("%d ", \*(value+ i));

} //2 4 6 8 10

(g) Print the elements of array values by subscripting the pointer to the array.

for (i = 0; i < SIZE; i++) {

printf("%d ", vPtr[i]);

} //2 4 6 8 10

(h) Refer to element 4 of values using array subscript notation, pointer/offset notation via the array name, pointer subscript notation, and pointer/offset notation.

printf("Element 4 are:\n"); //Element 4 are:

printf("%d\n", value[3]); //10

printf("%d\n", \*(value + 3)); //10

printf("%d\n", vPtr[3]); //10

printf("%d\n", \*(vPtr + 3)); //10

(i) What address is referenced by vPtr+3? What value is stored at that location?

0 1 2 3 4

2 4 6 8 10

The address referenced to vPtr+3 is 1002500+3=1002503. The value stored at that location is 8.

(j) Assuming vPtr points to values[4] address, what address is referenced by vPtr-=4? What value is stored at that location?

0 1 2 3 4

2 4 6 8 10

vPtr-=4 means subtract 4, but value[4] which moved the pointer to 4th place, the address referenced to vPtr-=4 is 1002504-4=1002500. The value stored at that location will be no change, the value is 2.

**7.10 For each of the following, write a single statement that performs the indicated task. Assume that long integer variables value1 and value2 have been defined and that value1 has been initialized to 200000.**  
(a) Define the variable 1Ptr to be a pointer to an object of type long.

long \*1Ptr;

(b) Assign the address of variable value1 to pointer variable 1Ptr.

1Ptr = &value1;

(c) Print the value of the object pointed to by 1Ptr.

printf("%d\n", \*1Ptr); // 200000

(d) Assign the value of the object pointed to by 1Ptr to variable value2.

value2 = \*lPtr;

(e) Print the value of value2.

printf(%d\n", value2); //200000

(f) Print the address of value1.

printf("%d\n", &value1); //14155160

(g) Print the address stored in 1Ptr. Is the value the same as the address of value1?

printf("%d\n", 1Ptr); //14155160

Yes, the value is the same as the address of value1.

**7.11 Do each of the following:**  
(a) Write the function header for function zero, which takes a long integer array parameter bigIntegers and does not return a value.

void zero(long \*bigIntegers)

(b) Write the function prototype for the function in part(a).

void zero(long\*);

(c) Write the function header for function add1AndSum, which takes an integer array parameter oneTooSmall and returns an integer.

int add1AndSum(int \*oneTooSmall);

(d) Write the function prototype for the function described in part(c).

int add1AndSum(int\*);

**7.19 Find the error in each of the following program segments. If the error can be corrected, explain how.**(a) int \*number;

printf(“%d\n”, \*number);

The pointer have to match with a corresponding value, if it does not have any values assigned, then it can not able to dereference it.

(b) float \*realPtr;

long \*integerPtr;

integerPtr=realPtr;

It has an error, while passing one pointer’s address to another, they have to be the same data type, they can’t be one float and one long data type.

(c) int \* x, y;

x=y;

It has two errors, not only there should be a \* before x, there also should have \* before y. Second, there should be no space between \* and variable name.

(d) char s[]=”this is a character array”;

int count;

for (; \*s != ; ++s){

printf(“%c ”, \*s);

}

It has two errors, s is char therefore in the for loop, s should not be \*s, secondly, s has to be assigned something after !=.

(e) short \*numPtr, result;

void \*genericPtr=numPtr;

result=\*genericPtr+7;

It has an error, genericPtr doesn’t have a data type, therefore we should assigned a data type to genericPtr before using void.

(f) float x=19.34;

float xPtr=&x;

printf(“%f\n”, xPtr);

It has two errors here, there should be \* before xPtr to point the values. Missing two \* before xPtr in 2nd line and in printf statement.

**2. From Chapter 6 of the textbook**  
**6.7 State which of the following are true and which are false. If false explain why.**(a) To refer to a particular location or element within an array, specify the array’s name and the value of the particular element.

False. To refer a particular location or element within an array, concluding the index is correct and concluding the value is incorrect.

(b) An array definition reserves space for the array.

True

(c) To indicate that 100 locations should be reserved for integer array p, write p[100].

False. Data type have to be specified before as int for integer array p, which will be int p[100];

(d) A program that initializes a 15-element array’s elements to zeros must contain a for statement.

False. There are other ways to initializes a 15-element array element to 0. So the for statement isn’t the only way to do it, therefore it doesn’t have to contain a for statement. For example, we can do “int list[15] = { 0 };”

(e) A program that totals the elements of a two-dimensional array must contain nested for statements.

False. To accomplish this, it is not necessary of containing a nested for loop, one for loop will also work for this question, such like this:

for(int i = 0; i < arr; i++) {

total += list[i][0] + list[i][1] + ... + list[i][j - 1];

}

**6.8 Write statements to accomplish each of the following:**

(a) Display the value of the seventh element of character array f:

printf("%c\n", f[6]);

(b) Input a value into element 4 of one-dimensional floating-point array b.

scanf("%.1f", &b[3]);

(c) Initialize each of the five elements of one-dimensional integer array g to 8.

int g[5] = { 8, 8, 8, 8, 8 };

(d) Total the elements of floating-point 100-element array c.

float total = 0;

for (int i = 0; i <1000; i++){

total += c[i];

}

(e) Copy array a into the first portion of array b. Assume a has 11 elements, b has 34 elements, and both arrays have the same element type.

for (int i = 0; i <= 10; i++) {

b[i] = a[i];

}

(f) Determine and print the smallest and largest values contained in 99-element floating-point array w.

float small, large;

small = w[0];

large = w[0];

for (int i = 1; i < 99; i++) {

if (w[i] < small)

small = w[i];

if (w[i] > large)

large = w[i];

}

printf("Smallest value is: %.2f\n”, small)

Printf(“Largest values is: %.2f\n", large);

**6.12 Write loops that perform each of the following one-dimensional array operations:**

(a) Initialize the 10 elements of integer array counts to zeros.

for (int i = 0; i < 10; i++){

count[i] = 0;

}  
(b) Add 1 to each of the 15 elements of integer array bonus.

for (int i = 0; i < 15; i++) {

add[i]++;

}  
(c) Read the 12 values of float array monthlyTemperatures from the keyboard.

for (int i = 0; i < 12; ++i) {

scanf("%.2f", &monthlyTemperatures[i]);

}  
(d) Print the five values of integer array bestScores in column format.

for (int i = 0; i < 5; ++i) {

printf("%d\n", bestScores[i]);

}

**6.30 You may submit a .c file with the program for the Sieve of Eratosthenes. Name the file with Dou\_J\_primes.c**

See attached file.