David Lane

004638553

CS 31 HW 6

Problem 1:

a. int main()  
    {  
        int arr[4] = { 0, 1, 2, 3 };  
        int\* ptr = arr;

        \*ptr = arr[ 1 ];               // set arr[0] to 1  
        \***(**ptr + 1**)** = arr[ 0 ] \* 10;      // set arr[1] to 10 **Missing ()**  
        ptr += 2;   
        ptr[0] = arr[ 1 ] \* 10;        // set arr[2] to 100  
        ptr[1] = 1000;                 // set arr[3] to 1000

**Ptr += 2; //in order for the while loop to print all 4 array members**

        while (ptr > arr) **//no = sign. = Creates out of bound error.**  
        {  
            ptr--;   
            cout << " " << \*ptr;    // print values  
        }  
        cout << endl;   
        return( 0 );   
    }

b. The function will not succeed in finding the last zero because the pointer value it takes in is called by value and therefore will not be changed by the function. To fix this add an ampersand as follows:

    void findLastZero(int arr[], int n, int\***&** p) **//requires the & to pass //by reference.**  
    {  
        p = nullptr;    /// default value if there isn't a 0 in the array at all  
        for (int k = n - 1; k >= 0; k--)   
        {  
            if (arr[k] == 0)      // found an element whose value is 0  
            {   
                 p = arr + k;     // change the value of p  
                 break;           // stop looping and return  
            }   
        }  
    }

c. the pointer in the main is not initialized so running this driver code will lead to undefined behavior. Even though the function establishes a value for the pointer, it’s not being stored in a variable so after the function call, the pointer again points at nothing. The way to fix this is:

int main()  
    {  
        **int** p; **// make this an integer**        biggest(15, 20, **&**p); **//add an ampersand here to point to int p**  
        cout << "The biggest value is " << **p** << endl;**//output p //instead of the dereferenced pointer**  
        return( 0 );  
    }

now the function call will change the value of the variable being pointed to.

d.     // return true if two C strings are equal   
    bool match(const char str1[], const char str2[])  
    {   
        bool result = true;   
        while (str1 != **‘\0’**  &&  str2 != **‘\0’**)  // **zero bytes at ends**   
        {  
            if (str1 != str2)  // compare corresponding characters  
            {  
                result = false;   
                break;   
            }   
            str1++;            // advance to the next character  
            str2++;  
        }   
        if (result)    
        {   
            result = (str1 == str2);   **// This //line is not possible or //correct**  
        }  
        return( result );  
    }

    int main()  
    {  
        char a[10] = "pointy";  
        char b[10] = "pointless"; 

        if (match(a,b))  
        {  
            cout << "They're the same!" << endl;  
        }  
    }

e. The program is printing out the addresses of certain variables rather than printing the desired values of the Fibonacci sequence, this means that the pointer is pointing to other pointers after the first iteration of the for loop.

Problem 2:

1. f

2. g

3. a

4. b

5. d

6. c

7. b

8. e

9. h

Problem 3: (Prints:)

**diff=1** //Prints 1 because &array[5] points to 6th item in the array and ptr is pointing to the 5th item in the array, 1 integer is between (6 minus 5) them so pointer arithmetic would evaluate to 1.

**4** // function swap2 switches the variables pointed to by the pointers called by the function. These pointers point to the first and third item in the array so the function switches their values. (function swap1 switches the addresses of the two pointers, but since they are call by value parameters, this is only for the duration of the function call, afterwards they go back to as they were before.)

**79** // “\*(array + 1) = 79;” array is a pointer to the first item in the array “array,” adding one points to the second item in the array then dereferencing it and assigning 79 sets the second item equal to 79.

**5** // function swap2 switches the variables pointed to by the pointers called by the function. These pointers point to the first and third item in the array so the function switches their values.

**9** // after minimart is called, the pointer ptr will be initialized to point to the third item in the array because the dereferenced value of the first parameter is greater than that of the second. ptr[1] is a dereferenced pointer to the next item in the array, the 4th item, and the assignment statement changes that integer value to 9.

**-1** // as stated above, the pointer was initialized to point to the third item in the array, then the statement “ptr += 2;” adds two to the pointer so it points to the fifth item in the array. Then the assignment statement dereferences the pointer and assigns the value -1 to the dereferenced integer variable.

**19** // this is simply declared in the array when it is initialized

Problem 4:

void deleteCapitals(char input[]) {

char\* moving = input; //new pointer

while (input != '\0') { //goes until end of string

if (isupper(\*input)) {// if the letter in the string in main isupper

input++;

moving = input; //set the new pointer to address of input++

}

input++;

moving++; //move pointer to next memory address of each

}

input = moving; //after gone through whole string set the pointer parameter to the //new pointer parameter

}