**PROBLEM 1**

**a.**

**Bugs:**

* ptr[0] = arr[ 1 ] \* 10; -> This only changes the address, of ptr, not the actual value.
* ptr[1] = 1000; -> same as previous reason.
* There was no change for the ptr to point to the last value in the array, so all the values in the array won’t get printed.

**Fixes (highlighted in red):**

int main()

{

    int arr[4] = {0, 1, 2, 3};

    int\* ptr = arr;

    \*ptr = arr[1];

    \*(ptr + 1) = arr[0] \* 10;

ptr += 2;

    \*ptr = arr[1] \* 10;

    \*(ptr + 1) = 1000;

    ptr += 1;

    while (ptr >= arr)

    {

        cout << \*ptr << “ “;

        ptr--;

    }

    return(0);

}

**b.**

**Bugs:**

* void findLastZero(int arr[], int n, int\* p) – The pointer variable isn’t passed by reference, so it won’t get updated to the address of the last zero

**Fixes (highlighted in red):**

void findLastZero(int arr[], int n, int\*& p)  
    {  
        p = nullptr;       
        for (int k = n - 1; k >= 0; k--)   
        {  
            if (arr[k] == 0)  
            {   
                 p = arr + k;       
                 break;             
            }   
        }  
    }

    int main()  
    {  
        int nums[6] = { 10, 20, 0, 40, 30, 50 };  
        int\* ptr;

        findLastZero(nums, 6, ptr);  
        if (ptr == nullptr)  
        {  
            cout << "The array doesn't have any zeros inside it." << endl;   
        }   
        else  
        {  
            cout << "The last zero is at address " << ptr <<  endl;  
            cout << "It's at index " << ptr - nums << endl;   
            cout << "The item's value is " << \*ptr << " which is zero!" << endl;  
        }   
        return( 0 );   
    }

**c.**

**Bugs:**

* int\* p; -> The pointer p isn’t initialized to the address of an int, so it will be assigned to something random

**Fixes (highlighted in red):**

void biggest(int value1, int value2, int \* resultPtr)

{

    if( value1 > value2 )

    {

        \*resultPtr = value1;

    }

    else

    {

        \*resultPtr = value2;

    }

}

int main()

{

    int a;

    int\* p = &a;

    biggest(15, 20, p);

    cout << "The biggest value is " << \*p << endl;

    return( 0 );

}

**d.**

**Bugs:**

* In the function match, the cstrings are treated like strings, which causes an issue in implementation. Each cstring (str1 & str2) should be tracked with an index & that should be incremented in the while loop.
* For the terminating condition, the last index should equal to ‘\0’, not 0.
* result = (str1 == str2); -> Cstrings can’t be compared this way. Also, this doesn’t consider the case when the strings end at different times

**Fixes (highlighted in red):**

bool match(const char str1[], const char str2[])

{

    bool result = true;

    int i = 0;

    int j = 0;

    while (str1[i] != ‘\0’  ||  str2[j] != ‘\0’)  // zero bytes at ends

    {

        if (str1[i] != str2[j])  // compare corresponding characters

        {

            result = false;

            break;

        }

        i++;

        j++;     // advance to the next character

    }

    return( result );

}

int main()

{

    char a[10] = "pointy";

    char b[10] = "pointless";

    if (match(a, b))

    {

        cout << "They're the same!" << endl;

    }

}

**e.**

* **for (int i = 0; i < n – 2; i**++) -> At this point, when n = 2, the loop will still not be entered and the Fibonacci numbers will not be accurate.
* **return arr;** -> Can’t return an array
* **ptr[i]** -> gives the address, not the value of the array element
* **int \* ptr = computeFibonacciSequence(m);** -> Can’t assign a pointer variable to an array

**PROBLEM 2**

1. string \* fp; 🡪 f. declares a pointer variable to point to a variable of type string
2. string fish[ 5 ]; 🡪 g. declares a five element array of string
3. fp = &fish[ 4 ]; 🡪 a. sets a pointer variable to the last element of an array of five strings
4. \*fp = "salmon"; 🡪 b. sets the string pointed to by a pointer variable to the value "salmon"
5. fp -= 3; 🡪 d. moves the pointer fp back three strings in the array it points to
6. \*(fp + 3) = "salmon"; 🡪 c. sets the fourth element of an array pointed to by the variable fp to the value "salmon"
7. fp[ 0 ] = "salmon"; 🡪 b. sets the string pointed to by a pointer variable to the value "salmon"
8. bool b = (fp == fish); 🡪 e. initializes a boolean to true if the pointer variable fp points to the string at the start of the fisharray, false otherwise
9. bool b = (\*fp == \*(fp + 1)); 🡪 h. initializes a boolean to true if fp points to a string whose value matches the string immediately following the string pointed to by fp, false otherwise

**PROBLEM 3**

**In int main():**

* **int array[6] = { 5, 3, 4, 17, 22, 19 };** -> an array of size 6 & type int is declared & initialized.
* **int\* ptr = minimart(array, & array[2]);** -> A pointer of type int calls the function int\* minimart( int \* a, int \* b).
* **In int\* minimart( int \* a, int \* b):**
  + **if (\*a < \*b)** -> checks if the value of array[0] ( = 5) is greater than the size of array[1] ( array[1] = 4)  
        **return a;  
    else   
      return b;**
  + function minimart returns the address of 4 – ptr points at the position of 4 in array.

**In int main():**

* **ptr[1] = 9;** -> the position following the position of 4 (array[3]) gets updated to 9 but pointer still points to 4
* **ptr += 2;** -> ptr points to 2 positions away from 4 – points at array[4](= 2.
* **\*ptr = -1;** -> The value of array[4] gets updated to -1.
* **\*(array+1) = 79;** -> The value of array[1] gets updated to 79.
* **cout << "diff=" << &array[5] - ptr << endl;** -> The difference between the positions of array[5] and ptr are subtracted and is printed on to the console (5-4=1). So, “diff=1” is printed out.
* **swap1(&array[0], &array[1]);** -> Function void swap1(int \* a, int \* b) is called where a is array[0] (= 5) & b is array[1] (= 79).
* **In swap1(int \* a, int \* b):**
  + **int \* temp = a;** -> a pointer variable of type int is declared & initialized. It points to array[0].
  + **a = b;** -> a points to array[1].
  + **b = temp;** -> b points to array[0].

**In int main():**

* **swap2(array, &array[2]);** -> Function void swap2(int \* a, int \* b) is called where a is array[0] (=5) and b is array[2] (= 4).
* **In swap2(int \* a, int \* b):**
  + **int temp = \*a;** -> A variable of type int is created in which the value of a ( = 5) is stored.
  + **\*a = \*b;** -> The value of a is updated to the value of b. So now, a is 4.
  + **\*b = temp;** -> The value of b is updated to the value of temp (the old value of a). So now, b is 5.

**In int main():**

* **for (int i = 0; i < 6; i++)**

**cout << array[i] << endl;**

This will print out the final values of the elements in array.

So, output for this step:

4

79

5

9

-1

19

**PROBLEM 4**

#include <iostream>

#include <cstring>

using namespace std;

void deleteCapitals(char \* c);

int main()

{

    char msg[100] = "Happy Days Are Here Again";

    deleteCapitals(msg);

    cout << msg << endl;

}

void deleteCapitals(char \* c)

{

    while (\*c != '\0')

    {

        if (isupper(\*c))

        {

            for(char \*p = c; \*p != '\0'; p++)

            {

                \*p = \*(p + 1);

            }

        }

        else

        {

            c++;

        }

    }

}