**Introduction**

This project is designed to help you master pointers.  To that end, you'll get the most out of it by working through the problems by hand.  Only after that should you resort to running the programs (and stepping through them with the debugger) to check your understanding.  Remember, on the final exam you'll have to be able to do problems like this by hand.

**Your task**

This "project" is more like one of the accompanying homeworks.  There are five problems.  In the problems that ask you to change code, make the few changes necessary to fix the code without changing its overall approach.  For example, don't fix the program in problem 1a by changing it to

int main()  
{  
     cout << " 1000 100 10 1" <<  endl;   
}

**Problem 1:** This problem involves errors related to the use of pointers.

a.  This program is supposed to write      1000 100 10 1      , but it doesn't.  Find all of the bugs and show a fixed version of the program: 

    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

**//priority of \* is higher than that of arithmetic operators. Therefore, \*ptr + 1 with no parentheses will evaluate \*ptr first, and the add 1 to the value of that element. To fix, we simply add parentheses, \*(ptr+1).**   
        ptr += 2;  
        ptr[0] = arr[ 1 ] \* 10;        // set arr[2] to 100  
        ptr[1] = 1000;                 // set arr[3] to 1000

**//The pointer is currently pointing to arr[3], so when it enters the while loop, the element will be hidden from the while loop. To fix, we need to increment the pointer to the next position, ptr = ptr + 1.**

        while (ptr >= arr)  
        {  
            ptr--;

**//ptr--; will hide the last element of the ptr from the scope of the while loop since it comes before the cout statement. Therefore, it should come after cout.**   
            cout << " " << \*ptr;    // print values  
        }  
        cout << endl;  
        return( 0 );  
    }

**FIXED VERSION of 1a:**

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

    ptr += 2;

    \*ptr = arr[ 1 ] \* 10;        // set arr[2] to 100

    \*(ptr+1) = 1000;                 // set arr[3] to 1000

    ptr = ptr + 1;

    while (ptr >=  arr)

    {

        cout << " " << \*ptr;    // print values

        ptr--;

    }

    cout << endl;

    return( 0 );

}

b.  The findLastZero function is supposed to find the last element in an array whose value is zero, and sets the parameter p to point to that element, so the caller can know the location of that element holding the value zero.  Explain why this function won't do that, and show how to fix it.  Your fix must be to the function only; you must not change the main routine below in any way.  As a result of your fixing the function, the main routine below needs to work correctly.

    void findLastZero(int arr[], int n, int\* p)

**//int\* p needs to be passed by reference in order to be seen in the main function. To fix, add &, so it would be int\* &p.**

    {  
        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  
            }  
        }  
    }

    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 );  
    }

**FIXED VERSION of 1b:**

void findLastZero(int arr[], int n, int\* &p)    **//need to add &!**

{

    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 biggest function is correct, but the main function has a problem.  Explain why it may not work, and show a way to fix it.  Your fix must be to the main function only; you must not change the biggest function in any way.

    #include <iostream>  
    using namespace std;

    void biggest(int value1, int value2, int \* resultPtr)  
    {  
        if( value1 > value2 )  
        {  
               \*resultPtr = value1;  
        }  
        else  
        {  
               \*resultPtr = value2;  
        }  
    }

    int main()  
    {  
        int\* p;

**//This is known as a dangling pointer. At this point, pointer p is not pointing at anything. We need to create a variable that is passed by reference for it to point at, and it needs to be able to be manipulated outside of the main function (in this case, by the void function biggest).**   
        biggest(15, 20, p);  
        cout << "The biggest value is " << \*p << endl;  
        return( 0 );  
    }

**FIXED VERSION of 1c:**

int main()

{

    int num;

    int\* p = &num;

    biggest(15, 20, p);

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

    return( 0 );

}

d.  The match function is supposed to return true if and only if its two C-string arguments have exactly same text.  Explain what the problems with the implementation of the function are, and show a way to fix them.

    // 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

**//To loop until the C-Strings end, we need to search for null character ‘\0’.**

**//Also, with this syntax, the while loop does not search the C-string by element, and therefore we cannot make this comparison. To fix, a declared variable with brackets is needed to represent the element of the array in consideration.**    
        {  
            if (str1 != str2)  // compare corresponding characters

**//This does not compare the actual dereferenced value, rather it compares the addresses. To fix, we need an \* to access dereferenced values.**   
            {  
                result = false;  
                break;  
            }  
            str1++;            // advance to the next character  
            str2++;  
        }  
        if (result)   
        {  
            result = (str1 == str2);    // both ended at same time?

**//We need to compare the addresses of elements of the C-Strings to see if the end at the same time. To do this, we need to declared variable.**   
        }  
        return( result );  
    }

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

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

**FIXED VERSION of 1d:**

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

{

    bool result = true;

    int i = 0 ;             //declared an i to refer element

    while (str1[i] != '\0'  &&  str2[i] != '\0')  // zero bytes at ends     //C-string, needs null '\0'

    {

        if (\*str1 != \*str2)  // compare corresponding characters    //need the value of the address of what's being pointed at

        {

            result = false;

            break;

        }

        str1++;            // advance to the next character

        str2++;

    }

    if (result)

    {

        result = (str1[i] == str2[i]);    // both ended at same time?

    }

    return( result );

}

e.   This program is supposed to write      1 1 2 3 5 8 13 21       but it probably does not.  What is the program doing that is incorrect?  (We're not asking you explain why the incorrect action leads to the particular outcome it does, and we're not asking you to propose a fix to the problem.)

    #include <iostream>  
    using namespace std;

    int fibonacci( int n )   
    {   
        int tmp;  
        int a = 1;  
        int b = 1;

        for (int i = 0; i < n-2; i++)   
        {  
            tmp = a+b;  
            a = b;  
            b = tmp;  
        }  
        return b;  
   }

   int\* computeFibonacciSequence(int& n)  
   {  
        int arr[8];  
        n = 8;  
        for (int k = 0; k < n; k++)  
        {  
            arr[k] = fibonacci( k );  
        }  
        return arr;

**//return arr; This cannot be done. C++ cannot handle returning an array. Therefore, this causes undefined behavior and will throw garbage values.**

    }

    int main()  
    {  
        int m;  
        int\* ptr = computeFibonacciSequence(m);  
        for (int i = 0; i < m; i++)  
        {  
            cout << ptr[i] << ' ';  
        }  
        return( 0 );  
    }

**Problem 2:**  Match each of the following statements with the explanation of what the statement does.  (NOTE: One of the descriptions listed below actually matches two statements)

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

1. string \* fp;           //f.

2. string fish[ 5 ];      //g.

3. fp = &fish[ 4 ];       //a.

4. \*fp = "salmon";        //b.

5. fp -= 3;               //d.

6.  \*(fp + 3) = "salmon"  //c.

7. fp[ 0 ] = "salmon";    //b.

8. bool b = (fp == fish); //e.

9. bool b = (\*fp == \*(fp + 1));   //h.

**Problem 3:**  What does the following program print and why?  Be sure to explain why each line of output prints the way it does to get full credit.   
int\* minimart(int\* a, int\* b)

{

    if (\*a < \*b)  **//if the dereferenced VALUE OF pointer a is less than dereferenced VALUE OF pointer b, return ADDRESS of a**

        return a;

    else

        return b;   **//otherwise, return ADDRESS of b**

}

void swap1(int\* a, int \*b)  //this is a swap function

{

    int\* temp = a;  **//creates temporary variable, and sets pointer a to it**

    a = b;          **//stores pointer b to pointer a**

    b = temp;       **//makes temporary variable (which previously was pointer a) to pointer b**

}

void swap2(int\* a, int \*b)  //second swap function

{

    int temp = \*a; **//creates temporary variable, and sets to VALUE OF what pointer a is pointing to to temp**

    \*a = \*b;    **//stores VALUE OF what pointer b is pointing at to the VALUE OF what pointer a is pointing at.**

    \*b = temp;  **//sets temp variable (which was previously pointer VALUE OF what pointer a is pointing to) to the VALUE OF what pointer b is pointing at**

}

int main()

{

    int array[6] = { 5, 3, 4, 17, 22, 19 }; **//define array**

    int\* ptr = minimart(array, & array[2]); **//accesses minimart function, feeds addresses of the designated positions of the array to minimart function, which dereferences (\*array = 5, \*array = 4), then returns the address of array[2], because \*array[2] is less than \*array (5>4, return b).**

    ptr[1] = 9; **//equivalent to \*(ptr+1) = 9. previously, ptr was pointing at array[2], so now it points to array[3], which held the value 17, but this line overwrites this value and sets it to 9.**

    ptr += 2;   **//move pointer by 2 positions, which makes it point to array[5], which has value 22.**

    \*ptr = -1;  **//at array[5], store VALUE -1, replacing the value of 22.**

    \*(array+1) = 79;    **//referring to array[0], move one position, which is array[1], and overwrite the current value 3 and set it to 79.**

    cout << "diff=" << &array[5] - ptr << endl; **//the difference in space between the address of &array[5] and address of what ptr was last pointing at, which is &array[4], is 1.**

    swap1(&array[0], &array[1]);        **//swap the addresses of the array[0] and array[1]**

    swap2(array, &array[2]);            **//swap the actual dereferenced value of array[2] with value of array[0].This makes array now read as {4, 79, 5...} whereas previously it was {5, 79, 4...}. The only element that was in original array that was not affect was array[6], which holds the value 19.**

    for (int i = 0; i < 6; i++) **//prints content of the array, which now should be {4, 79, 5, 9, -1, 19}**

        cout << array[i] << endl;

    return( 0 );

}

**Problem 4:**  Write a function named deleteCapitals that accepts one character pointer as a parameter and returns no value. The parameter must be a C-string.  This function must remove all of the upper case letters from the string.  The resulting string must be a valid C-string.

Your function must declare no more than one local variable in addition to the parameter; that additional variable must be of a pointer type.  Your function must not use any square brackets and must not use the strlen or strcpy library functions.

    int main()  
    {  
        char msg[100] = "Happy Days Are Here Again!";  
        deleteCapitals(msg);  
        cout << msg << endl;        // prints:   appy ays re ere gain!  
    }

**MY FUNCTION deleteCapitals**

void deleteCapitals(char \* arr)

{

    char\* ptr = arr;

    for(ptr=arr; \*ptr != '\0'; arr++)     //while the ptr is accessing things that can be accessed (no negative or out of range), and incrementing arr..

    {

        if(!isupper(\*arr))  //if dereferenced index in arr is a NOT uppercase character, set the pointer value equal to value of arr at that index.

        {

            \*ptr=\*arr;

            ptr++;  //then move pointer to next position.

        }

    }

}