Name: Ojas Bardiya

UID: 505145284

1. (a)

#include <iostream>  
    using namespace std;  
      
    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[0] = arr[ 1 ] \* 10;        // set arr[2] to 100  
        ptr[1] = 1000;                 // set arr[3] to 1000

      ptr += 1;

while (ptr >= arr)  
        {  
            cout << " " << \*ptr;    // print values  
        ptr--;

}  
        cout << endl;   
        return( 0 );   
    }

In my solution above, I have:

* increased the pointer value by 1 before the while loop so as to start from the last element of the array
* put the cout statement before the decrement operator to make sure that the last element is printed first before moving on to the ones before it

(b)

In the original function, p is a pass by value parameter. Hence, its value does not get changed even after implementing the function. Thus, an & needs to be placed before it to make it a pass by reference parameter so that any change to the within the actual function itself to the pointer is done to the actual pointer. The change to be made is:

void findLastZero(int arr[], int n, 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  
            }   
        }  
    }

(c)

The reason that the main function doesn’t work is because though the pointer is declared, it is never initialized and doesn’t point to anything particularly in the memory. Fixing it simply requires creating a variable and making the pointer point to the address of that variable in the main routine.

int main()  
    {  
        int b = 0;

int\* p = &b;  
        biggest(15, 20, p);   
        cout << "The biggest value is " << \*p << endl;  
        return( 0 );  
    }

(d)

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

{

bool result = true;

while (\*str1 != 0 && \*str2 != 0) // zero bytes at ends

{

if (strcmp(str1, str2)) // compare corresponding charactersstr

{

result = false;

break;

}

str1++; // advance to the next character

str2++;

}

return( result );

}

* The main problem with this function was that it compared the base addresses of the arrays rather than each element of the arrays ,making it always return falseas the two would invariably be different.
* Another error with the original function was the presence of an unnecessary if statement at the end which was:

if (result)

{

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

}

This was wrong as the final result can’t be returned as a comparison and the if statement is redundant anyway. Thus, I removed the if statement as my first step.

* Secondly, I implemented the strcmp function in order to compare the two C-strings, as strcmp compares each character of the string unlike comparison operators like ‘==’ and ‘!=’ .
* I also added the \* before str1 and str2 in the while loop in order to dereference the pointers and determine if their values point to the zero byte.

(e)

In this particular program, the int\* computeFibonacciSequence(int& n) function returns the wrong value, as it basically returns the memory address associated with the variable instead of returning the value stored in that particular array index.

2)

1) – f) declares a pointer variable to point at a variable of type string

2) – g) declares a five element array of string

3) – a) sets a pointer variable to the last element of an array of five strings

4) – b) sets the string pointed to by a pointer variable to the value "salmon"

5) – d) moves the pointer fp back three strings in the array it points to

6) -   b) sets the string pointed to by a pointer variable to the value "salmon"

7) – c) sets the fourth element of an array pointed to by the variable fp to the value "salmon"

8) – e) initializes a boolean to true if the pointer variable fp points to the string at the start of the fish array, false otherwise

9) – 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.

3)

The output is:

diff=1

4

79

5

9

-1

19

#include <iostream>  
    using namespace std;   
  
    int\* minimart(int\* a, int\* b)  
    {  
        if (\*a < \*b) // basically returns the smaller pointer position  
            return a;  
        else   
            return b;  
    }

    void swap1(int\* a, int \*b) // swaps the addresses of the values but does not change output.  
    {  
        int\* temp = a;  
        a = b;   
        b = temp;  
    }

    void swap2(int\* a, int \*b). // swaps the values  
    {  
        int temp = \*a;  
        \*a = \*b;   
        \*b = temp;  
    }

    int main()  
    {  
        int array[6] = { 5, 3, 4, 17, 22, 19 }; //initializes an array of size 6.

        int\* ptr = minimart(array, & array[2]); //compares the value of 5 and 4 according to the pointer position and sets the pointer position to base address of 4 (which is &array[2]) as 4<5.  
        ptr[1] = 9; // makes the value of &array[3] to 9.  
        ptr += 2; // pointer is moved up to position &array[4]  
        \*ptr = -1; // assigns array[4] the value -1.  
        \*(array+1) = 79; // assigns array[1] the value 79.

        cout << "diff=" << &array[5] - ptr << endl; //prints out diff=1

        swap1(&array[0], &array[1]);  
        swap2(array, &array[2]); //swaps the vales of array[0] and array[2]

        for (int i = 0; i < 6; i++)  
            cout << array[i] << endl; //prints out values of the array

        return( 0 );  
    }

At the end, the array is {4. 79, 5, 9, -1, 19).

4)

The function is as follows:

void deleteDigits(char c[])

{

while (\*c != 0)

{

if(isdigit(\*c))

{

\*c = '\b';

}

c++;

}

}