Project 6

Jingjing Nie UID: 304567417

**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

ptr += 2;

ptr[0] = arr[ 1 ] \* 10; // set arr[2] to 100

ptr[1] = 1000; // set arr[3] to 1000

ptr ++;

while (ptr >= arr && ptr < &arr[4])

{

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

ptr--; // switch their orders

}

cout << endl;

return( 0 );

}

b)

In this code, since it does not use pass by reference, the address of pointer p is unchanged in the main function. Thus, the function won’t tell the caller the location of the last zero.

To fix it, we should add a ‘&’ to the function:

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

}

}

}

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)

In the main function, the declared pointer ptr does not have an initialization, which prevents the function from being built successfully. Thus, in order to solve it, we should use an int a to initialize it first.

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)

The problem with this function is that it will always return false even if the two strings are indeed the same. This is because the function is comparing the addresses of these two strings and they have different addresses.

Instead, what it should do is to compare the actual character, which means the content of the two.

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); // both ended at same time?

}

return( result );

}

int main()

{

char a[10] = "pointy";

char b[10] = "pointless";

if (match(a,b))

{

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

}

}

e)

The int array[8] was a local array since it is created in the function. Thus, the array values are lost outside the function and errors are generated in the main function.

The way the function is trying to return an array is conceptually wrong, too, because the size of the array is unknown to the driver code when it is returned.

Also, int m in the main function is uninitialized, so it passes an uninitialized value to the computeFibonacciSequence function.

Moreover, in fibonacci function, instead of letting int i < n-2, we should change it to i < n-1. Otherwise, the Fibonacci sequence would be generated to be 1 1 1 2 3 5 8 13, which is wrong.

**Problem 2.**

1 f ; 2 g ; 3 a ; 4 b ; 5 d ; 6 c ; 7 b ; 8 e ; 9 h ;

**Problem 3.**

(The explanations are presented by comments in the code)

//pointers pointing to array[0] and array [2] are set to this function

int\* minimart(int\* a, int\* b)

{

if (\*a < \*b)

return a;

else //\*a=5, \*b=4, so return b, which points to array[2]

return b;

}

//does not use pass by reference, so the addresses of the pointers do not change in the driver code

void swap1(int\* a, int \* b)

{

int\* temp = a;

a = b;

b = temp;

}

//swap the values the pointers point at using a temp

void swap2(int\* a, int \*b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int main()

{

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

int\* ptr = minimart(array, & array[2]); //let ptr point to array[2]

ptr[1] = 9;

//ptr[1] points to array[3], so it changes array[3] from 17 to 9

ptr += 2;

//shift ptr to point array[4]

\*ptr = -1;

//change array[4] from 22 to -1

\*(array+1) = 79;

//array is a constant pointer which always points to array[0], so \*(array+1) refers to the value it points at array[1], so it changes array[1] from 3 to 79

cout << "diff=" << &array[5] - ptr << endl;

//position difference is 1, since pointer now points to array[4]

swap1(&array[0], &array[1]);

//didn't do anything because array is fixed

swap2(array, &array[2]);

//swap values at array[0] and array[2], which are 5 and 4, so array[0]=4 and array[2]=5

for (int i = 0; i < 6; i++)

cout << array[i] << endl;

//print the numbers in the array: 4 79 5 9 -1 19

return( 0 );

}

**Problem 4.**

void deleteCapitals (char \* ptr) //function with a CString parameter

{

char \* p = ptr;

// while the pointer is pointing to some characters, which means within the string

while (\*ptr != '\0')

{

// if the character the pointer points at is upper case

if (isupper (\*ptr))

p = ptr;

// let the pointer p point to the same adress as pointer

while (\*p != '\0')

{

\*p = \* (p +1);

p ++;

//assign the later characters to the former ones/ shift the whole string to the left

}

if (!isupper (\*ptr))

ptr ++;

}

}

int main()

{

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

deleteCapitals(msg);

cout << msg << endl; // prints: appy ays re ere gain!

}