**Project 6**

**PROBLEM 1A**

**int** main()

{

**int** arr[3] = { 5, 10, 15 };

**int**\* ptr = arr;

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

\*(ptr + 1) = 20; // set arr[1] to 20

// Use parentheses to dereference entire expression, or else it is just the expression “10 + 1 = 20”

ptr += 2;

ptr[0] = 30; // set arr[2] to 30

**while** (ptr >= arr)

{

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

ptr--;

// Move variable update to after print statement, or it will skip printing arr[2]

}

cout << endl;

}

**PROBLEM 1B**

This function doesn’t work because the pointer is a pass-by-value parameter, which only stores a local copy of the argument that will not be updated outside of the function. This can be fixed by using a reference operator (&) to turn the pointer parameter into a pass-by-reference parameter.

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

{

for (int k = 1; k < n; k++)

{

if (arr[k] < arr[k-1])

{

p = arr + k;

return;

}

}

p = nullptr;

}

**PROBLEM 1C**

The main function may not work because pointer p is uninitialized, so function hypotenuse will be dereferencing an unknown address. This can be fixed by initializing the pointer before running, which I did using a random double.

int main()

{

double\* p;

double someDouble = 0;

p = &someDouble;

hypotenuse(1.5, 2.0, p);

cout << "The hypotenuse is " << \*p << endl;

}

**PROBLEM 1D**

This function implementation is incorrectly comparing the memory locations of the pointer variables str1 and str2 instead of comparing the values to which they point. This can be fixed by using the dereference operator to retrieve the data to which the pointer variables str1 and str2 point.

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

{

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

{

if (\*str1 != \*str2) // compare corresponding characters

return false;

str1++; // advance to the next character

str2++;

}

return \*str1 == \*str2; // both ended at same time?

}

**PROBLEM 1E**

This program initializes arr[10] as a local variable.

**PROBLEM 2**

1. **string \*fp;**

1. **string fish[5];**



1. **\*fp = "yellowtail";**
2. **\*(fish + 3) = "salmon";**

1. **fp -= 3;**
2. **fp[1] = "pike";**

1. **fp[0] = "basa";**

1. **bool d = (fp == &fish[0]);**

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

**PROBLEM 3A**

double computeAverage(const double \*scores, int nScores)

{

double tot = 0;

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

{

tot += \*(scores + i);

}

return tot/nScores;

}

**PROBLEM 3B**

const char\* findTheChar(const char \*str, char chr)

{

int k = 0;

while (\*(str + k) != 0)

{

if (\*(str+k) == chr)

{

str += k;

return str;

}

++k;

}

str = nullptr;

return str;

}

**PROBLEM 3C**

const char\* findTheChar(const char str[], char chr)

{

while (\*str != 0)

{

if (\*str == chr)

return str;

else

++str;

}

str = nullptr;

return str;

}

**PROBLEM 4**

This program modifies the elements of integer array array[], and the final for loop of this program prints out all 6 elements in array from indexes 0 to 5, separated by newlines.

These elements are { 4, 79, 5, 9, -1, 19, }

OUTPUT:

**4**

**79**

**5**

**9**

**-1**

**19**

OUTPUT EXPLANATION:

|  |  |
| --- | --- |
| **4** | swap2(array, &array[2])  // swaps the values of the variables being points to at array[0] and array[2],  so array[2] = 5 and array[0] = 4 |
| **79** | \*(array+1) = 79;  // Sets the value of array[1] equal to 79 by dereferencing array + 1 |
| **5** | swap2(array, &array[2])  // swaps the values of the variables being points to at array[0] and array[2],  so array[2] = 5 and array[0] = 4 |
| **9** | ptr[1] = 9;  // Sets the value of the first element of ptr (the 4th element of array) equal to 9 by dereferencing ptr + 1 |
| **-1** | **int**\* ptr = minimart(array, &array[2]);  // Initializing pointer ptr to point to the variable with the lowest value between array[0] and array[2] -- 4 < 5, so it points to array[2];  ptr += 2;  // Moves the pointer ptr forward by two array elements (ptr = array[4]) by adding 2 to its address  \*ptr = -1;  // Sets the value of array[4] equal to -1 by dereferencing ptr |
| **19** | // array[5] maintains its original value of 19 because it is untouched in the routine |

**LINE-BY-LINE EXPLANATION:**

|  |
| --- |
| **#include <iostream>**  **using namespace std;**  **int\* minimart(int\* a, int\* b) // Initializing a function that returns an integer pointer**  **{**  **if (\*a < \*b) // If the value that a points to is less than the value that b points to, return pointer a; else, return pointer b**  **return a;**  **else**  **return b;**  **}**  **void swap1(int\* a, int \*b) // fails to swap the addresses of its arguments because its parameters are**  **pass by value, creating local variables that don’t impact the main routine.**  **{**  **int\* temp = a;**  **a = b;**  **b = temp;**  **}**  **void swap2(int\* a, int \*b) // Initializing a function that swaps the values of the variables that a and b point to**  **{**  **int temp = \*a; // Setting an integer variable equal to the value of the variable that pointer a points to**  **\*a = \*b; // Setting the value that pointer a points to equal to the value that pointer b points to**  **\*b = temp; // Setting the value that pointer b points to equal to the value that pointer a originally pointed to**  **}**  **int main()**  **{**  **int array[6] = { 5, 3, 4, 17, 22, 19 };**  **int\* ptr = minimart(array, &array[2]); // Initializing pointer ptr to point to the variable with the lowest value between array[0] and array[2] -- 4 < 5, so it points to array[2];**    **ptr[1] = 9; // Sets the value of the first element of ptr (the 4th element of array) equal to 9 by dereferencing ptr + 1**  **ptr += 2; // Moves the pointer ptr forward by two array elements (ptr = array[4]) by adding 2 to its address**  **\*ptr = -1; // Sets the value of array[4] equal to -1 by dereferencing ptr**  **\*(array+1) = 79; // Sets the value of array[1] equal to 79 by dereferencing array + 1**  **cout << "diff=" << &array[5] - ptr << endl; // Outputs diff = (the address of array[5] - the address of ptr (array[4]) = 1) because array[5] and array[4] are one contiguous memory location apart**  **cout << &array[0] << endl;**  **cout << &array[1] << endl;**  **swap1(&array[0], &array[1]); // Swaps the memory addresses of array[0] and array[1]**  **cout << "swap" << endl;**  **cout << &array[0] << endl;**  **cout << &array[1] << endl;**  **swap2(array, &array[2]); // Swaps the values of the variables being pointed to at array[0] and array[2]), so array[2] = 5 and array[0] = 4**  **for (int i = 0; i < 6; i++) // Prints out all 6 elements in array[] in order separated by endlines**  **cout << array[i] << endl;**  **}** |

**PROBLEM 5**

**void** deleteG (**char** \*ptr)

{

**char**\* ptr2 = ptr;

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

{

**if** (\*ptr == 'g' || \*ptr == 'G' )

{

ptr2 = ptr;

**while** (\*ptr2 != '\0')

{

\*ptr2 = \*(ptr2+1);

++ptr2;

}

}

**else**

++ptr;

}

}