Brian Han-Wei Chang

UID: 405-142-813

CS31 Project 6

**Problem 1(a)**

**Original:**

#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; //We should omit this entirely, such that the while-loop will run starting from the first element of the array.

ptr[2] = arr[ 1 ] \* 10; // set arr[2] to 100 //Should be ptr[2] instead of ptr[0] in order to set the value of the 3rd element of the array.

ptr[3] = 1000; // set arr[3] to 1000 //Should be ptr[3] instead of ptr[1] to set the 4th element of the array.

while (ptr <= &arr[3]) // loop over the whole array

{

Ptr++; //Should increment the counter at the end of the loop body, not the start.

cout << " " << \*ptr; // print a single value

}

cout << endl;

return( 0 );

}

**Problem 1(a):**

**Fixed:**

#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;** //Commenting out to show omission.

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

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

while (ptr <= &arr[3]) // loop over the whole array

{

cout << " " << \*ptr; // print a single value

**ptr++;**

}

cout << endl;

return( 0 );

}

**Problem 1(b)**

The function is currently passing in pointer ‘ptr’ in a pass-by-value format, such that the function creates and manipulates a copy of the pointer instead of the real pointer. To fix this, we should make it a pass-by-reference argument which allows us to manipulate the actual pointer ‘ptr’ within the function ‘findFirstZero’.

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

By changing this one line of code the function should run normally.

**Problem 1(c)**

The problem with 1(c) is that the pointer ‘p’ is not initialised at declaration. We should initialise the pointer with an address such that it isn’t empty.

**Fixed Code:**

#include <iostream>

using namespace std;

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

{

if( value1 < value2 )

{

\*resultPtr = value1;

}

else

{

\*resultPtr = value2;

}

}

int main()

{

**int tempvar = 0;** //declared temporary variable, tempvar.

**int\* p = &tempvar;** //Stored tempvar’s address in point p at initialisation.

smallest(15, 20, p);

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

return( 0 );

}

**Problem 1(d):**

The main problem of this code is that it is missing the dereferencing operator ( \* ) to call the actual characters in the string. The code is currently comparing the addresses of the characters instead of the characters themselves. Furthermore, we need to declare ‘#include <iostream>’ and ‘using namespace std;’ in order to use ‘cout’. Fixed code below.

#include <iostream>

using namespace std;

// return true if two C strings are equal

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

{

bool result = true;

**while (\*str1 != 0 && \*str2 != 0)** //Added dereferencing operator.

{

**if (\*str1 != \*str2)** ///Added dereferencing operator.

{

result = false;

break;

}

str1++; // advance to the next character

str2++;

}

if (result)

{

**result = (\*str1 == \*str2);** //Added dereferencing operator.

}

return( result );

}

int main()

{

char a[10] = "pointy";

char b[10] = "pointless";

if (match(a,b))

{

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

}

}

**Problem 1(e):**

Because ‘int array[8];’ is initialised within the function, it only exists within the function itself. Once the function is run the array “dies” with it.

When the function ‘computeFibonacciSequence(m)’ returns, the pointer ‘ptr’ will still point to the memory address where the first element of the array was stored, but since the array itself no longer exists in memory, we get garbage values when we try to access it.

**Problem 2:** *What does the following program print and why? Comment each line of code to explain what it is doing.*

The program prints ‘5 4 3 2 1’

The program creates an integer array with 5 elements, {1,2,3,4,5}. The program then swaps the first and last value, and the 2nd and 2nd last value using a for-loop. Such that the array is modified to be {5,4,3,2,1} instead of {1,2,3,4,5}. Then the program uses a second for-loop to print the elements of the array one by one.

#include <iostream> //including library function used for input/output functions like cin and cout.

using namespace std; //letting the compiler know that we are using the standard namespace.

int main( ) //main function.

{

int x[ 5 ] = { 1, 2, 3, 4, 5 }; // initialising integer array with 5 elements. 1 to 5.

int \* p = x; // initialising a pointer that points to the memory address of the first element in the array.

int i; // declaring integer variable i.

//This for loop swaps first and last values from the ends of the array to the middle.

//Such that, array[0] swaps with array[4] and array[1] swaps with array[3].

//array[2] is unchanged since the loop only runs twice. Which makes sense because it's the middle element of an array with an odd number of elements.

for (i = 0; i < 2; i++) //creating a for loop that runs twice. i=0 & i=1.

{

int temp = \*(p + i); //initialising integer variable temp with the dereferenced (\*) value of (p + i)

\*(p + i) = \*(p + 4 - i); //Assigning the dereferenced value of (p+4-i) as the dereferenced value of (p+i)

\*(p +4 - i) = temp; //assigning the value stored in temp as the dereferenced value of (p+4-i)

}

for (i = 0; i < 5; i++) //creating a for loop that runs 5 times. From i=0 to i=4, incrementing i by 1 per loop.

{

cout << x[i] << " "; //we're printing the i-th element of the array.

}

cout << endl; //ending the line that we printed the elements of the array on.

return( 0 ); //ending the program with exit code 0.

**Problem 3:** *What does the following program print and why? Comment each line of code to explain what it is doing. Explain why the second \* is required on line A below.*

The program prints ‘5.999 5.999 5.9995.999’

The first time is printing the value of x. The second time prints the value of dereferenced memory address of x. The third time prints the value of the dereferenced pointer y. The fourth time prints the value of the dereferenced pointer z. There is a space between each printed value except between the third and fourth printed value.

#include <iostream>

using namespace std;

int main()

{

double x = 5.999; //initialising a double variable called x with the value 5.999

double \*y, \*z; //initialising two double pointer variables, y and z.

//The second \* is necessary to ensure that z is initialised as a pointer as well.

y = &x; //assigning the memory address of x to pointer y.

z = y; //assigning the memory address STORED in pointer y to pointer z

//printing the value of x. ( x )

//printing one space.

//printing the dereferenced memory address of x. ( \*(&x) )

//printing one space.

//dereferencing pointer y and printing the value (\*y)

//dereferencing pointer z and printing the value (\*z)

cout << x << " " << \*(&x) << " " << \*y << \*z << endl;

return( 0 ); //ending the program with exit code 0.

}

**Problem 4:**  *What does the following program print and why? Comment each line of code to explain what it is doing.*

The program prints ‘10 50 40 0’.

The program creates an integer array called track with the values {10,20,30,40} and a pointer called pointer that initially points to the memory address of the first element of the array. The program then increments the value of the second element of the array by 30. Prints the first element of the array through a dereferenced pointer. Decrements the value of the first element of the array by 10. Increments the pointer by 1, such that it points to the memory address of the second element in the array. Prints the second element via its dereferenced pointer. Increments the pointer by 2, such that it points to the fourth element of the array. Prints the fourth element via its dereferenced pointer. Then prints the first element of the array.

#include <iostream>

using namespace std;

int main()

{

int track[ ] = { 10, 20, 30, 40 }; //initialising an array with values {10,20,30,40}

int \* ptr; //declaring a pointer, ptr.

ptr = track; //assigning the memory address of the first value of the array to pointer ptr.

track[1] += 30; //incrementing the value of the second element of the array by 30, such that track[1] = 50.

cout << \* ptr << " "; //printing the value of the first element of the array. (printing '10')

\*ptr -= 10; //decrementing the value of the first element of the array by 10, such that track[0] = 0.

ptr++; //incrementing the pointer ptr by 1, such that it now holds to the memory address of the second element of track.

cout << \* ptr << " "; //printing the value of the second element in the array. (printing '50')

ptr += 2; //incrementing the pointer ptr by 2 such that it now holds to the memory address of the fourth element of track.

cout << \* ptr << " "; //printing the value of the fourth element of the array. (printing '40')

cout << track[ 0 ] << endl; //printing the value of the first element in the array. (printing '0')

return( 0 ); //ending program with exit code 0.

}

**Problem 5:**  *Rewrite this program and remove the variable declared at line A below. Make your revised program generate the exact same output as the original without using the variable declared at line A.*

#include <iostream>

using namespace std;

int main()

{

int num[ 5 ];

\*num = 100;

\*(&num[0+1]) = 90;

\*(&num[2]) = 80;

\*(&num[0+3]) = 70;

\*(&num[0+4]) = 60;

for (int i = 0; i < 5; i++)

cout << num[ i ] << " ";

cout << endl;

return( 0 );

}

**Problem 6:** *Write a function named deleteDigits 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 digit character 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.*

#include <iostream>

using namespace std;

void deleteDigits(char\* str)

{

char\* ptr = str; //initialising a second pointer, ptr, that points to the memory address of the first element of the char array.

while (\*ptr != '\0') //creating a while loop that parses the char array one char at a time.

{

if (\*ptr >= '0' && \*ptr <= '9') //creating an if condition that runs if the loop finds a digit.

{

ptr++; //incrementing pointer ptr. str pointer remains at digit index.

}

else //creating an else condition that will run as long as the char is not a digit.

{

\*str = \*ptr; //replacing the value at the most recent integer index with the first non-integer value it finds.

str++; //incrementing pointer str.

ptr++; //incrementing pointer ptr.

}

}

\*str = '\0'; //Terminating the string.

}

int main()

{

char msg[100] = "Happy 2019!";

deleteDigits(msg);

cout << msg << endl; // prints: Happy !

}