Joseph Picchi

UID: 605-124-511

Project 6 Solutions

1a) The original code is shown below with the bugs indicated in boldface:

#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

**while (ptr >= arr)**

*//ptr points to index 2 at this point, when it should point to index 3 in order to*

*//print out all values in the array.*

{

**ptr--;**

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

*//if ptr-- occurs before cout, the program accesses a value outside the*

*//bounds of the array*

}

cout << endl;

return( 0 );

}

A fixed version of the program is shown below:

#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 );

}

1b) The problem is that the pointer parameter in the findLastZero function is pass-by-value and not pass-by-reference, so the line p = arr + k; will not change the address to which the pointer points outside of the function scope. You would fix this by making the function pass-by-reference for the pointer parameter, so the function header should be changed from

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

to

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

1c) The program may not work because when the pointer is created in the line int\* p; , it is uninitialized, so it doesn’t point to anything. When p is passed into the biggest function, the value at the memory address to which it points cannot be changed because p doesn’t point to any anything. To fix this, the main function could be rewritten as follows:

int main()

{

int n;

int\* p = &n;

biggest(15, 20, p);

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

return( 0 );

}

1d) The function is shown below with the bugs in boldface and explained in comments:

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

{

bool result = true;

while (**str1 != 0 && str2 != 0**)

**//The while statement currently checks the addresses of the pointers, when**

**//it should be checking if the values they point to are equivalent to the null**

**//character**

{

if (**str1 != str2**)

**//The if statement is currently executed if str1 and str2 point to**

**//different addresses, but it should be executed only if the values**

**//pointed to by str1 and str2 are different**

{

result = false;

break;

}

str1++;

str2++;

}

if (result)

{

result = (**str1 == str2**);

**//This expression is currently true only if str1 and str2 point to the**

**//same address, but it should be true if the values pointed to by str1**

**//and str2 are equivalent**

}

return( result );

}

A fixed version of the function is shown below (the main function should be unchanged)::

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

{

bool result = true;

while (\*str1 != '\0' && \*str2 != '\0')

{

if (\*str1 != \*str2)

{

result = false;

break;

}

str1++;

str2++;

}

if (result)

{

result = (\*str1 == \*str2);

}

return( result );

}

1e) The problem is that the line int arr[8]; in the function computeFibonacciSequence creates a local array in the scope of the function and attempts to return a pointer to the first element of this array. However, since the array is local to the function and only exists in stack memory, it is deleted after the function is called, so the values stored in the array can no longer be accessed by the driver code and thus the desired output cannot be printed.

2) The pairings between statements and descriptions would be as follows:

1. F
2. G
3. A
4. C //NOTE: here, and in statement 6, we assume that the “fourth element of the
5. D // array” refers to the element at the fourth index. This assumption is necessary
6. C // in order for the solution to be consistent with the directions for the problem.
7. B
8. E
9. H

3) The program prints the following output:

diff=1

4

79

5

9

-1

19

The first output line occurs because of the following line of logic:

1. int array[6] = {5, 3, 4, 17, 22, 19}; creates an array called array with 6 elements.
2. minimart(array, &array[2]) passes into the function minimart a pointer to the element at index 0 (5) and a pointer to the element at index 2 (4).
3. Because (5 < 4) returns false, the else statement of minimart is initiated, which returns the pointer to the element at index 2 of array.
4. int\* ptr creates a pointer called ptr and initializes it to the pointer returned by minimart(array, &array[2]) , so ptr points to the element at index 2 of array.
5. ptr += 2; makes ptr point to the element at index 2+2 of array. So ptr now points to the element at index 4 (22).
6. The statement &array[5] - ptr returns 1 because the difference (in units of indexes) between a pointer pointing to index 5 of array and a pointer pointing to index 4 of array is 1.
7. Therefore, the line cout << “diff=” << &array[5] - ptr << endl; prints the first line of output, followed by a newline.

The second line of output occurs because of the following:

1. int array[6] = {5, 3, 4, 17, 22, 19}; creates an array called array with 6 elements.
2. The statement swap2(array, &array[2]); passes a pointer to index 0 of array and a pointer to index 2 of array to the function swap2 as a and b, respectively.
3. The function swap2 swaps the values pointed to by array and &array[2], so the values at indexes 0 and 2 are swapped.
4. Now, the value at index 0 is 4, so when the line cout << array[i] << endl; is executed, 4 is printed as the second line of output, followed by a newline.

The third line of output occurs because of the following:

1. int array[6] = {5, 3, 4, 17, 22, 19}; creates an array called array with 6 elements.
2. \*(array+1) = 79; changes the value at index 0+1 of array to 79. So the value at index 1 is 79.
3. When the line cout << array[i] << endl; is executed, 79 is printed as the third line of output, followed by a newline.

The fourth line of output occurs because of the following:

1. Steps 1-3 for the second line of output are initiated.
2. Now, the value at index 2 is 5, so when the line cout << array[i] << endl; is executed, 5 is printed as the fourth line of output, followed by a newline.

The fifth line of output occurs because of the following:

1. Steps 1-4 for the first line of output are initiated, so now ptr points to the element at index 2 of array.
2. ptr[1] = 9; sets the value at index 2+1 equal to 9. So the value at index 3 of array is now 9.
3. When the line cout << array[i] << endl; is executed, 9 is printed as the fifth line of output, followed by a newline.

The sixth line of output occurs because of the following:

1. Steps 1-4 for the first line of output are initiated, so now ptr points to the element at index 2 of array.
2. ptr += 2; makes ptr point to the value at index 2+2 of array. So ptr now points to index 4 of array.
3. \*ptr = -1; changes the value at index 4 of array to -1.
4. When the line cout << array[i] << endl; is executed, -1 is printed as the sixth line of output, followed by a newline.

The seventh line of output occurs because of the following:

1. int array[6] = {5, 3, 4, 17, 22, 19}; creates an array called array with 6 elements.
2. No changes are made to the last element, so when the line cout << array[i] << endl; is executed, 19 is printed as the seventh line of output, followed by a newline.

4) The deleteDigits function would appear as follows:

void deleteDigits(char\* p) {

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

if (\*p >= '0' && \*p <= '9') {

for (char\* p2 = p; \*p2 != '\0'; p2++) {

\*p2 = \*(p2+1);

}

}

else {

p++;

}

}

}