1. A) 30

20

10

int main()

{

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

int\* ptr = arr;

\*ptr= 10;

ptr++;

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

ptr++;

\*ptr = 30;// set arr[0] to 30

while (ptr >= arr)

{

cout << \*ptr << endl;

ptr--;// print values

}

}

1. B) This function does not change the value of the pointer of because it falls out of scope. Since the pointer that is part of the function call is uninitialized a reference to a pointer should be used in the function declaration.

void findMax(int arr[], int n, int\*& pToMax)

{

if (n <= 0)

return; // no items, no maximum!

pToMax = arr; // location of first index of array

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

{

if (arr[i]>\*pToMax)

{

pToMax=arr+i;

}

}

}

1. C) This function does not work because the pointer is uninitialized when it is called. In addition, the output shows the location of the pointer in memory as opposed to the actual value of the pointer, which is desired.

void computeCube(int n, int\* ncubed)

{

\*ncubed = n \* n \* n;

}

int main()

{

int a=1;

int\* ptr=&a;

computeCube(5, ptr);

cout << "Five cubed is " << \*ptr << endl;

}

1. D) The condition of the loop checks to see if the locations of the strings are not 0, which is not what you want to do. Instead you should check to see if the specific characters are the 0 byte to go until the end of the string by using the \*symbol. Another problem is that the part of the code comparing characters looks at their locations as opposed to their specific character values, so a star should be added to do so. Lastly the function should return true after it exits the loop because all the values are the same instead of checking if the last locations are the same.

// return true if two C strings are equal

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

{

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

{

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

return false;

str1++; // advance to the next character

str2++;

}

return true; // both ended at same time?

}

int main()

{

char a[15] = "Noor";

char b[15] = "Noah";

if (strequal(a,b))

cout << "They're the same person!\n";

}

1. E) The scope of array anArray[] is only defined in the function so, when the function returns anArray[] that array is no longer usable by the main function and this results in undefined behavior.

This program is supposed to write 100 99 98 3 2 1, but it probably does not. What is the program doing that is incorrect? (We're not asking you explain why the incorrect action leads to the particular outcome it does, and we're not asking you to propose a fix to the problem.)

#include <iostream>

using namespace std;

int\* getPtrToArray(int& m)

{

int anArray[100];

for (int j = 0; j < 100; j++)

anArray[j] = 100-j;

m = 100;

return anArray;

}

void f()

{

int junk[100];

for (int k = 0; k < 100; k++)

junk[k] = 123400000 + k;

}

int main()

{

int n;

int\* ptr = getPtrToArray(n);

f();

for (int i = 0; i < 3; i++)

cout << ptr[i] << ' ';

for (int i = n-3; i < n; i++)

cout << ptr[i] << ' ';

cout << endl;

}

double\* cat;

double mouse[5];

cat=mouse+4;

\*cat=42;

\*(mouse+3)=25;

cat-=3;

cat[1]=17;

cat[0]=54;

bool b=(\*cat== \*(cat+1));

bool d=(cat==mouse);

3) a.

double mean(const double\* scores, int numScores)

{

const double\* ptr = scores;

double tot = 0;

int i=0;

while (i!=numScores)

{

tot += \*(ptr+i);

i++;

}

return tot/numScores;

}

B)

// This function searches through str for the character chr.

// If the chr is found, it returns a pointer into str where

// the character was first found, otherwise nullptr (not found).

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

{

const char\* ptr=str;

for (int k = 0; \*(ptr+k)!=0; k++)

{

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

return &\*(ptr+k);

}

return nullptr;

}

C)

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

{

while (\*str!=0)

{

if (\*str == chr)

return str;

str++;

}

return nullptr;

}

4.

The program prints out

3

4

79

-1

9

22

19

The first line of code in the main sets a pointer to an integer to array because the maxWell function takes in the pointer to the 0 element and the 2nd element and returns a pointer to the larger one, which in this case in array element 0. Then the value of the the 1st element is set to -1.Then the element in the array that the pointer is pointing is incremented by 2 to element 2. Then element 3 of the array is set to 9. Then the position in the array that the pointer is pointing, the 0 element, is incremented by one and the value of the element is set to 79.The array at this point has a value {-1,79,4,9,22,19} Then we are subtracting the same array at different elements and i<j the difference (3) is the first output. The swap one method does not change the actual values of the elements but instead changes the locations of a and b in memory, which does not change the actual integer that the pointer points to. While on the other hand the swap2 function actually changes the values that the pointer points. Therefore the array that is printed out with each element followed by a newline is {4,79,-1,9,22,19}.

#include <iostream>

using namespace std;

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

{

if (\*a > \*b)

return a;

else

return b;

}

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

{

int\* temp = a;

a = b;

b = 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 = maxwell(array, &array[2]);

\*ptr = -1;

ptr += 2;

ptr[1] = 9;

{-1,79,4,9,22,19}

\*(array+1) = 79;

cout << &array[5] - ptr << endl;

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

swap2(array, &array[2]);

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

cout << array[i] << endl;

}

5.

void removeS( char\* c)

{

while(\*c!=0)

{

if (\*c=='s' || \*c=='S')

{

for (char\* ptr=c;\*ptr!=0; ptr++)

{

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

}

}

else

{

c++;

}

}

}

int main()

{

char msg[50] = "She'll be a massless princess.";

removeS(msg);

cout << msg; // prints he'll be a male prince.

}