1a.

\*ptr + 1 = 20; is a bug, this does not set arr[1] to 20

You need to change that line to \*(ptr + 1) = 20, which means \*(&arr[1]) = 20, which means arr[1] = 20, which successfully sets the integer at index 1 to a value of 20.

Also, change while loop to a for loop so you can print out the array in ascending index order.

Revised Code:

int main()

{

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

int\* ptr = arr;

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

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

ptr += 2;

ptr[0] = 10; // set arr[2] to 10

for(ptr = &arr[0]; ptr < &arr[3]; ptr++)

{

cout << \*ptr << endl;

}

}

1b.

This function does not work correctly because outside of findMax, the pointer initialized in the main method does not end up pointing to the maximum item in the array. To fix this problem, just change the last argument in findMax to int&\* pToMax so that ptr in the main method points to the maximum item. pToMax becomes another name for ptr, so they both point at the same place.

Revised Code:

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

{

if (n <= 0)

return; // no items, no maximum!

pToMax = arr;

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

{

if (arr[i] > \*pToMax)

pToMax = arr + i;

}

}

int main()

{

int nums[4] = { 5, 3, 15, 6 };

int\* ptr;

findMax(nums, 4, ptr);

cout << "The maximum is at address " << ptr << endl;

cout << "It's at position " << ptr - nums << endl;

cout << "Its value is " << \*ptr << endl;

}

1c.

The pointer ptr is unitialized, so to fix this, initialize an int, and then initialize pointer ptr to point at the address of the int.

Revised Code:

void computeCube(int n, int\* ncubed)

{

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

}

int main()

{

int i;

int\* ptr = &i;

computeCube(5, ptr);

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

}

1d.

The const char str1 and str2 are pointers, so instead of directly comparing pointers to pointers, you need to compare the value that the first pointer points to to the value that the second pointer points to. In other words, you need to dereference the pointers, so that the values that the pointers point to are actually being compared. Change all str1 to \*str1 and str2 to \*str2 when you want to compare values. Leave str1++ and str2++ because you want the pointer to point at the next value of the c string.

Revised Code:

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

}

int main()

{

char a[15] = "Noor";

char b[15] = "Noor";

if (strequal(a,b))

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

}

1e.

The storage for the local variable anArray goes away after the function getPtrToArray returns. However, getPtrToArray returns a pointer to that storage. Attempting to follow that pointer in the main routine (when ptr[i] is called) yields undefined behavior.

2

1. double\* cat;
2. double mouse[5];
3. cat = mouse + 4;
4. \*cat = 42;
5. \*(mouse + 3) = 25;
6. cat -= 3;
7. cat[2] = 17;
8. cat[0] = 54;
9. bool b = (\*cat == \*(cat + 1));
10. bool d = (cat == mouse);

3a.

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

{

int i = 0;

double tot = 0;

while (i != numScores)

{

tot += \*(scores + i);

i++;

}

return tot/numScores;

}

3b.

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

{

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

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

return (str + k);

return nullptr;

}

3c.

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

{

for ( ; str != 0; str++)

if (\*str == chr)

return str;

return nullptr;

}

4.

3: &array[5] generates the address of the sixth integer of the array, and ptr is an address of the third integer of the array, so &array[5] – ptr equals address of 6th integer minus address of 3rd integer which equals 3. So the first cout statement prints out 3.

4: Program prints out number at index 0. The call to swap2 swaps the two values that the pointers point at, so the value at index 0 of the array gets swapped with the value at index 2, so old array is { -1, 79, 4, 9, 22, 19 } and new array is: { 4, 79, -1, 9, 22, 19 }.

79: Program prints out number at index 1. \*(array + 1) = 79 means that the value at index 1 is set to 79.

-1: Program prints out number at index 2. -1 was set to the value at index 0 with the statement \*ptr = -1. Then, the swap2 call swapped the values at index 0 and index 2, so value at index 2 now holds the -1 value.

9: Program prints out number at index 3. Pointer ptr initially pointed at index 0, but after the ptr += 2 call, it now points at index 2. Then, ptr[1] refers to the index one after index 2, so ptr[1] = 9 is the the same thing as saying the value at index 3 is set to 9.

22: Program prints out number at index 4. This value was unaffected by the program, so the value at index 4 is still 22.

19: Program prints out number at index 5. This value was unaffected by the program, so the value at index 5 is still 19.

5.

void removeS(char\* c)

{

char\* final = c;

for ( ; \*c != ‘\0’; c++)

{

if (\*c != ‘s’ && \*c != ‘S’)

{

\*final = \*c;

final++;

}

}

\*final = ‘\0’;

}