1.

1. #include <iostream>

using namespace std;

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

ptr = arr;

while (ptr <= &arr[2])

{

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

ptr++;

}

}

1. The issue with the function give is that the parameter pToMax is not passed by reference, therefore the variable ptr is never initialized in main, meaning that nothing can be printed and undefined behavior occurs. This is easily fixed by adding an ampersand after the data type int\* in the parameters for the function, and it functions perfectly thereafter.

Fix(to function):

void findMax(int arr[], int n, int\*& pToMax) // ampersand added!

{

if (n <= 0)

return; // no items, no maximum!

pToMax = arr;

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

{

if (arr[i] > \* pToMax)

pToMax = arr + i;

}

}

1. Once again the program does not work because the pointer is not passed by reference, and therefore undefined program behavior will occur because ptr has no value. To fix this without affecting the function in any way, we must send the address of what ptr is pointing to. To do so, ptr must be initialized, which I do by initializing an array a of size 1 and giving ptr the value of the address of a[0]. Then in the call to the function instead of just ptr I send up &\*ptr, the address of what ptr points to, which is a[0]. Therefore when \*ptr is written out as the result, the modified value that is stored and displayed is found in a[0].

Fix(to main only):

int main()

{

int arr[1] = { 2 }; //initialization of int to store value of result

int\* ptr = arr;

computeCube(5, &\*ptr); //send up address of arr[0]

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

}

1. The way I see it, the implementation of the function has 4 main issues. The first is that even if the comparison properly worked, it will return false by default because the supposed return for true is not a valid way to return true. To fix this I simply changed the comparison meant to return true to simply return true. The second issue is that if the function had corrected its format it would return true if both string were equal up until one of them ended. To combat this I added an or within the if statement that if the length of the two strings were not equal, it would return false. The third issue is that the strings are not being compared correct, using a for loop with an initialized value of 0 to represent the current position in the string, I compare each character correctly in the funciton.

Fix(to function):

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

{

for (int a = 0; a < strlen(str1); a++)//advance to the next character

{

if ((str1[a] != str2[a]) || (strlen(str1) != strlen(str2))) // compare corresponding characters

return false; // return false if one ends before or if not equal;

}

return true; // both ended at same time?

}

1. The first issue is that the pointer in the main function ends up containing the address of element 0 of an array that does not exist in the main. This creates undefined behavior.

2.

1. double\* cat;
2. double mouse[5];
3. cat = &mouse[4];
4. \*cat = 25;
5. \*(mouse+3) = 54;
6. cat = cat -3;
7. cat[1] = 27;
8. cat[0] = 42;
9. bool b = (\*cat == \*(cat + 1));
10. bool d = (cat == mouse);

3.

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

{

double tot = 0;

int a = 0;

while (a < numScores)

{

tot += \*(scores+a);

a++;

}

return tot/numScores;

}

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

{

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

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

return &\*(str+k);

return nullptr;

}

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

{

while (\*str != 0)

{

if (\*str == chr)

return &\*str;

str++;

}

return nullptr;

}

4. The program prints the following:

3 //because ptr at the time &array[5]-ptr is printed is pointing at array[2], which means that the result of the subtraction is 3, as there are three spaces of memory required for a double between them

4 //because as a result of the swap2 function, the elements of array[0] and array[2] are switched, giving array[0] the unmodified value of array[2] while array[2] gets the value -1, which is given to array[0] before the swap occurs

79 //because array[1] is given this value and is actually totally unmodified by the swap1 function

-1 //becuase swap2 function gives array[2] the modified value of array[0] as a result of earlier operations in the code

9 //the pointer originally points to array[0], then to array[2], therefore the operation ptr[1] = 9, modifies array[3] to become the integer 9

22 //this element of the integer array is unaffected by any line in the code and is therefore unmodified and is printed as such

19 //much like the prior number printed, this element of the array is unchanged by any operations with the pointers and such and is therefore printed as an unmodified value from the initialization of the array.

5.

void removeS(char\* str)

{

char\* ptr = str;

while (\*ptr != 0)

{

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

{

while (\*ptr != 0)

{

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

ptr++;

}

ptr = str;

}

ptr++;

}

}