1.

a.

The code line “\*ptr + 1 = 20” is supposed to set arr[1] to 20, however, it resulted in a compiler error, what it’s supposed to be is “\*(ptr+1) = 20”. Another bug was that the “ptr--" came before the cout line, which printed starting from the address of arr[1].

int main()

{

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

int\* ptr = arr;

\*ptr = 10; // set arr[0] to 10

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

ptr += 2;

ptr[0] = 30; // set arr[2] to 30

while (ptr >= arr)

{

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

ptr--;

}

cout << endl;

}

b. Pointers are passed by value, and thus the original function did not change the value of ptr. This made value of ptr to always point at the first element. By doing “int\*& p”, the function works properly.

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

{

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

{

if (arr[k] < arr[k - 1])

{

p = arr + k;

return;

}

}

p = nullptr;

}

c. The pointer p does not point to anything, and thus has no place to store the value of the hypotenuse. It could be solved by creating a double variable x for pointer p to point to.

int main()

{

double x;

double\* p = &x;

hypotenuse(1.5, 2.0, p);

cout << "The hypotenuse is " << \*p << endl;

}

d. The function calls str1 and str2’s memory address and not to the character in the each array. By adding a \* operand for every time we want to compare the str1 and str2’s characters, we can solve the issue.

// return true if two c strings are equal

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

{

while (\*str1 != 0 && \*str2 != 0) // zero bytes at ends

{

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

return false;

str1++; // advance to the next character

str2++;

}

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

}

e. The program only contains the array in the computeSquares function, and not in ptr. Pointer ptr points to addresses not of m.

2.

int main()

{

string\* fp;

string fish[5] = {"smelt", "test", "test", "test", "test"};

fp = &fish[4];

\*fp = "yellowtail";

\*(fish + 3) = "salmon";

fp -= 3;

fp[1] = "carp";

fp[0] = "smelt";

bool d;

if (fp == &fish[0])

{

d = true;

}

else

{

d = false;

}

bool b;

if (\*fp == \*(fp + 1))

{

b = true;

}

else

{

b = false;

}

}

3.

a.

double computeaverage(const double\* scores, int nscores)

{

double tot = 0;

int i = 0;

while (i < nscores)

{

tot += \*scores;

\*scores++;

i++;

}

return tot / nscores;

}

b.

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;

}

c.

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

{

while (\*str != 0)

{

if (\*str == chr)

{

return (str);

}

str++;

}

return nullptr;

}

4.

* array[2] > array[0], thus minimart returns ptr to address of array[2]
  + **ptr == array[2]**
* ptr[1] = 9
  + ptr[0] == &array[2]
  + ptr [1] -> &array[2 + 1]
  + ptr[1] -> &array[3]
  + **array[3] = 9**
    - {5, 3, 4, 9, 22, 19}
* ptr+=2
  + ptr -> &array[2]
  + ptr+=2 >> ptr -> &array[2 + 2]
  + **ptr -> &array[4]**
* \*ptr = -1
  + ptr -> &array[4]
  + **array[4] = -1**
    - {5, 3, 4, 9, -1, 19}
* \*(array + 1) = 79
  + array[0 + 1] = 79
  + **array[1] = 79**
    - {5, 79, 4, 9, -1, 19}
* &array[5] – ptr
  + ptr -> &array[4]
  + **&array[5] - &array[4] = 1**
* no change, pointers are passed by value, so the addresses of &array[0] and &array[1] remain the same
* value at array[2] is copied to array[0], and value at array[0] is copied to array[2]
  + **array[0] == 4, array[2] == 5**
    - {4, 79, 5, 9, -1, 19}

output:

diff= 1

4

79

5

9

-1

19

5.

void deleteG(char\* str)

{

char\* p = str;

while (\*str != 0)

{

if (tolower(\*p) == 'g')

{

p++;

}

\*str = \*p;

str++;

p++;

}

str++;

\*str = 0;

}