1a).

int main()

{

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

int\* ptr = &arr[2];

\*(ptr-2) = 30; // set arr[0] to 30

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

\*ptr = 10; // set arr[2] to 10

ptr -= 2;

/\*cerr << \*ptr << endl;

ptr++;

cerr << \*ptr << endl;

ptr++;

cerr << \*ptr << endl;\*/

while (ptr <= &arr[2])

{

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

ptr++;

}

}

1b).

The function will not produce correct results the way it is because the pointer pToMax must be passed by reference. If it is not passed by reference, the function will simply move pToMax to the correct position but not the original pointer in the main function. If not passed by reference the program will simply output the position of the uninitialized pointer ptr.

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 problem with the program is that pointer ptr is not initialized to anything, therefore is could hold any random address. To fix this problem, you must initialize it so that it is pointing to some specific space in memory.

void computeCube(int n, int\* ncubed)

{

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

}

int main()

{

int i = 0;

int\* ptr = &i;

computeCube(9, ptr);

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

}

1d). The problem with this program is that you want the compare the values of each of the characters not their addresses. In order to do this properly, asterisks must be added into the strequal function.

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

{

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

{

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

return false;

str1++;

str2++;

}

return \*str1 == \*str2;

}

int main()

{

char a[15] = "Tang";

char b[15] = "Zhang";

if (strequal(a,b))

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

}

1e). The problem is that ptr in the main function points to an element in anArray. However, anArray only exists in the function getPrtToArray so you cannot (technically you can cause it will compile but give the wrong answer) dereference ptr in the main function. In getPrtToArray you are only returning the address of the first thing in anArray! (not the array itself).After you leave getPtrToArray the array effectively disappears.

2.

a). double\* cat;

b). double mouse[5] ;

c). cat = mouse + 4;

d). \*cat = 42;

e). \*(mouse + 3) = 17;

f). cat -= 3;

g). cat[1] = 25;

h). cat[0] = 54; //check if it is cat[1]

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

j). bool d = (cat == mouse);

3a).

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;

}

3b).

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

{

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

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

return ptr+k;

return nullptr;

}

3c).

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

{

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

{

if(\*ptr == chr)

return ptr;

}

return nullptr;

}

4). The first line of output is the number 3. This is because the address that ptr holds is subtracted from the address of the final element in the array. At this point, ptr holds the address of the third element of the array at position 2. Subtracting these two things gives the distance between the two. In this case that distance is 3. The next line of output is the number 4. This is because previously in the main function, the value of the number at the 0th position of the array was made to be -1 and the value of the number at the 2nd position of the array was made to be 4. When function swap 2 is called, it swaps the values at these positions making it so that the value at the 0th position is now 4 and the value at the 2nd position is now -1. Also, previously in the main function, the element at the 1st position of the array way made to be 79 by the line \*(array+1) = 79. The element at the third position of the array was initially 17, but it got changed to 9 because after ptr moved to point to the 2nd position of the array, ptr[1] is changed to 9. This means that the first element after where the pointer currently is gets its value changed. In reality the value of the element at position 3 of the array is what gets changed.

The elements in the array after these remain unchanged and therefore when the for loop at the end of the main function prints out each of the numbers in the array, the following appears…

**3**

**4**

**79**

**-1**

**9**

**22 //these were initially in the array and remain unchanged**

**19 //**

Note that the first three that is printed out is not from the array.

5).

void removeS(char MSG[])

{

char\* i = MSG;

while(\*MSG != 0)

{

if(\*MSG != 'S' && \*MSG != 's')

{

\*i = \*MSG;

i++;

}

MSG++;

}

\*i = 0;

}