1a)

int main()

{

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

int\* ptr = arr;

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

ptr++;

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

ptr++;

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

while (ptr >= arr)

{

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

ptr--;

}

}

1b)

The pToMax parameter is pass-by-value function as it is written right now, so ptr will never be assigned an actual address in memory. We have to pass the pointer as a reference.

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)

In the original code, ptr does not point to a location in memory, so the computeCube function has nowhere to store the result of n\*n\*n.

void computeCube(int n, int\* ncubed)

{

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

}

int main()

{

int cube = 0;

int\* ptr = &cube;

computeCube(5, ptr);

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

}

1d)

As it is written, the function compares the memory addresses of the two C-strings instead of their corresponding characters. Also, the while loop terminates only when the pointers’ memory addresses, rather than the character values that they point to, are both 0.

// 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] = "Chen";

char b[15] = "Cheng";

if (strequal(a,b))

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

}

1e)

The function getPtrToArray tries to return the memory address of the first element to anArray. However, anArray is a local variable. Consequently, the pointer loses its scope outside of the function itself, meaning that in the main routine, the ptr variable invokes unpredictable behavior.

2a) double\* cat;

b) double mouse[5];

c) cat = &mouse[4];

d) \*cat = 25;

e) \*(mouse + 3) = 54;

f) cat -= 3;

g) cat[1] = 27;

h) cat[0] = 42;

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

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

3a)

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

{

double tot = 0;

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

tot += scores[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)

{

const char\* ptr = str;

while(\*ptr != 0)

{

if(\*ptr == chr)

return ptr;

ptr++;

}

return nullptr;

}

4)

Output (each new line is a new line in the output):

3

4

79

-1

9

22

19

Explanation of the main routine:

1. The call to the maxwell function will return the pointer to the first element in the array since 5 > 3.
2. The next 4 lines set the first element’s value to -1, move the pointer to the 3rd element’s memory address, set the 4th element’s value to 9, and set the second element’s value to 79. The array now looks like this: [-1 79 4 9 22 19]
3. The first line in the output will be the difference between the last element’s address and the pointer’s address (which is to the 3rd element). Therefore, this expression yields 6 minus 3, which is 3.
4. The call to the swap1 function does nothing to the array because the function does operations on only the values of the pointers themselves.
5. The call to the swap2 function swaps the values to which the pointers point. Thus, the value of the first element in the array is switched with the value of the third element in the array. The array looks like this: [4 79 -1 9 22 19]
6. The for-loop prints out the final values of the array.

5)

void removeS(char\* str)

{

char\* fast = str;

while(\*fast != 0) {

if(\*fast != 's' && \*fast != 'S') {

\*str = \*fast;

str++;

}

fast++;

}

\*str = 0;

}