1:

a.

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

ptr = arr; // point back to the first element of array

while (ptr < arr + 3)

{

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

ptr++;

}

}

b.

the function actually creates a copy of the pointer that we pass to it, just like normal variable. To use the original pointer, we must use reference of the pointer when we define the function

void findMax(int arr[], int n, int \*&pToMax) //use reference to pointer

{

if (n <= 0)

return; // no items, no maximum!

pToMax = arr;

for (int i = 1; i < n; i++) //find the maximum integer

{

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;

}

c.

the computeCube function creates a copy of a pointer that we pass to it. Thus, the original ptr pointer will point to nothing. Instead of passing a pointer, we could just declare a variable and pass its address to the function.

void computeCube(int n, int\* ncubed)

{

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

}

int main()

{

int ptr; //use normal integer variable

computeCube(5, &ptr); //pass the address to the function

cout << "Five cubed is " << ptr << endl; //ptr represents the value

}

d.

when we compare corresponding characters, we cannot just compare the address by using str1 != str2. Instead, we should compare the value using \*str1 and \*str2.

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] = "Shen";

char b[15] = "Shi";

if (strequal(a, b))

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

}

e.

in the main function, it first creates an integer n and a pointer ptr to an integer. It calls the function getPtrToArray and pass n to it, and assign the returned pointer to ptr. In getPtrToArray function, it creates a new array anArray which stores 100 integers, and uses a loop to assign 100 to 1 to each element, and finally returns the pointer to first element of the array. Back to the main function, since anArray is declared inside another function, it only has the scope in that function. Just returning back the pointer to the first element of it actually leaves all elements in ptr uninitialized. Thus, it prints random number.

2.

a. double\* cat;

b. double mouse[5];

c. cat = mouse + 4;

d. \*cat = 25;

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

f. cat -= 3;

g. cat[1] = 42;

h. cat[0] = 27;

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

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

3.

a.

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

{

double tot = 0;

int num = 0; /\*use an integer to denote the address that pointer points to\*/

while (num != numScores)

{

tot += \*(scores + num); //\*(scores + num) equals to scores[num]

num++;

}

return tot / numScores;

}

b.

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

{

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

if (\*(str + k) == chr) //use str + k to move the pointer

return (str + k);

return nullptr;

}

c.

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

{

for (str; \*str != 0; str ++) /\*str itself can be changed forward using ++\*/

if (\*(str) == chr)

return str;

return nullptr;

}

4.

3

4

79

-1

9

22

19

The maxwell function return pointer to the larger value of two parameters pointers point to, then assigned to ptr. Now ptr points to the first element of array. Move the ptr two element ahead, now pointing to array[2]. Assign ptr[1], which is now array[3], to 9. \*(array + 1) points to array[1], so it is now 79. So far, the array is {-1, 79, 4, 9, 22, 19}. Since ptr now points to the third element in array, and &array[5] is the address of sixth element in array, their difference is 3. In Swap1 function, parameters are references to pointers. The function receives two addresses of pointers and swap these addresses, which is not allowed and has no effect on original elements in array. Swap2 swaps the value that the pointers point to. Now the array becomes {4, 19, -1, 9, 22, 19}.

5.

#include <iostream>

Using namespace std;

void removeS(char\* arr);

int main()

{

char msg[50] = "She'll be a massless princess.";

removeS(msg);

cout << msg; // prints he'll be a male prince.

}

void removeS(char\* arr)

{

for (arr; \*(arr) != 0; arr++) /\*using pointer to visit every element\*/

{ /\*in the array\*/

while (\*(arr) == 'S' || \*(arr) == 's') /\*while the character is\*/

{ /\*either case of s\*/

\*(arr) = ' '; /\*assign it to empty\*/

for (char \*ptr = arr; \*(ptr) != 0; ptr++) /\*move every\*/

\*ptr = \*(ptr + 1); /\*character to its left\*/

}

}

}