1.a.

{

    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

    // OR: \*ptr = 10;

     ptr-=2;

    while (ptr < arr+3)

    {

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

        ptr++;

    }

}

The program originally printed the elements in reverse order.

1.b.

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;

    }

}

The problem is that the pointer is pass-by-value in the original function (without the ‘&’ sign). Therefore, the function returns a random pointer’s address, index and value that is initialized in the main route.

1. C

int main()

{

    int\* ptr=new int;

    computeCube(5, ptr);

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

}

The initialized pointer has no memory space to store the result of computeCube()

A pointer must point to an existing object or variable. Otherwise, use “new” to create a new memory space

The highlighted line can also be:

int result=0; //initialize result with value 0. Anyway it will always get replaced

int \*ptr=&result;

1.d

bool strequal(char str1[], 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?

}

The problem is:

The pointer str1 and str2, which point to the beginning of the two C-Strings, are passed as const. Therefore, manipulating the pointer(str1++ and str2++) is not allowed and thus it is unable to traverse through the array.

Also, the original comparing line if (str1 != str2) was actually comparing the address of the starting element of two arrays. It will always be false if str1 and str2 are not pointing to the same array even if the arrays are equal.

1. e.

The program creates a dangling pointer “ptr” pointing to the array generated by getPtrToArray. But the array was destroyed as the function call ends. “ptr” ends up pointing to some random junk. Modifying ptr is of no use.

2. a.

double\* cat = new double;

    B.

    double mouse[5];

    C.

    cat = mouse+4;

    D.

    \*cat = 25;

    E.

    \*(mouse+3) = 42;

    F.

    cat -=3;

    G.

    cat[1]=27;

    H.

    cat[0]=54;

    I.

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

    J.

    bool\* d = new bool(cat==mouse);

3.a.

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

{

    int i=0;

    double tot = 0;

    while (i != numScores)

    {

        tot += \*(scores+i);

        i++;

    }

    return tot/numScores;

}

Notice:

1. 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;

}

1. C

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

{

    while(str!=0){ // Same as while(\*str!='\0')

        if (\*str == chr)

            return str;

        str++;

    }

    return nullptr;

}

4.

Output:

**3**

**Pointer was first set by maxwell() to point to array[0]. Adding 2 makes it point to array[2]. Therefore,** &array[5] - ptr **gives the number of separation between the 6th and the 3rd element in array, which is 3.**

**4**

**The swap2() function actually swap the values pointed by array[0] and array[2]. It changes the elements pointed to by array[0] from -1 to 4 and that of array[2] from 4 to -1 separately.**

**The reason why swap1() is not working is that both pointers are passed by value. The function only changed the pointers, both of which are destroyed after the function call.**

**79**

**array+1 points to the second element of array**

\*(array+1) = 79; **changes its value to 79**

**-1**

**swap2 function switched the values of the first element of the array and the third element of the array**

**9**

**ptr was originally pointing to a[0].** ptr += 2; **makes it pointing to the third element in array. ptr[1]=9; assigns the fourth element to 9**

**22**

**Fifth element in array is 22. It was not changed after initialization.**

**19**

**Sixth element in array is 19. It was not changed after initialization.**

**5.**

void removeS(char ch[]){

    int index=0;

    while(\*ch!='\0'){

        if(tolower(\*ch)=='s'){

            index++;

        }

        else{

            \*(ch-index)=\*ch;

        }

        ch++;

    }

    \*(ch-index)='\0';

}