**1.**

**a.**

ORIGINAL:

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

//This adding is not correct without parentheses

//This causes a compile-error

ptr += 2;

ptr[0] = 10; // set arr[2] to 10

while (ptr >= arr)

{

ptr--;

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

//(1) This causes an out of range error because after executing two times, ptr equals arr. Therefore, this loop will still execute once. However, ptr[-1](arr[-1]) is undefined.

//(2) Another bug is that if we run the program like this, it will print 10, 20, 30 rather than 30, 20, 10 one per line. Since right before the while loop, ptr points to the value which arr[2] is. And the decrement of ptr will reversely print the elements of the array.

}

}

FIXED VERSION:

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 -= 2;

while (ptr <= arr + 2)

{

cout << \*ptr << endl;

ptr++;

}

}

b.

ORIGINAL:

void calcSum(int arr[], int n, int\* pToSum)

{

pToSum = new int;

(\*pToSum) = 0;

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

{

(\*pToSum) += arr[i];

}

}

int main()

{

int nums[4] = { 5, 3, 15, 6 };

int\* ptr;

calcSum(nums, 4, ptr);

cout << "The sum is " << \*ptr << endl;

}

// This does not work as intended because the pointer type parameter pToSum is a call-by-value parameter. The pointer itself and the variable it points will not change after the function is called.

FIXED VERSION:

void calcSum(int arr[], int n, int\* &pToSum) //The ampersand is the key revision

{

pToSum = new int;

(\*pToSum) = 0;

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

{

(\*pToSum) += arr[i];

}

}

int main()

{

int nums[4] = { 5, 3, 15, 6 };

int\* ptr;

calcSum(nums, 4, ptr);

cout << "The sum is " << \*ptr << endl;

}

c.

ORIGINAL:

void smaller(int val1, int val2, int\* smaller)

{

if( val1 < val2 )

\*smaller = val1;

else

\*smaller = val2;

}

int main()

{

int\* min;

smaller(5, 8, min);

cout << "The min of 5 and 8 is " << \*min << endl;

}

//Although min is declared at the beginning of main(), it is a dangling pointer, thus calling smaller() will not assign the smaller value of two integers to the variable min is pointing(min does not point to any variables). Therefore, it’s proper to initialize the pointer to a dynamic variable at the beginning.

FIXED VERSION:

void smaller(int val1, int val2, int\* smaller)

{

if( val1 < val2 )

\*smaller = val1;

else

\*smaller = val2;

}

int main()

{

int\* min = new int; //key point

smaller(5, 8, min);

cout << "The min of 5 and 8 is " << \*min << endl;

}

d.

ORIGINAL:

// return true if two C strings are equal

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

{

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

//an invalid bool expression that always yields true, resulting in an infinite loop

{

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

//The condition within the if-statement always yields true because str1 and str2 without indices never compare the corresponding characters in each element space

return false;

str1++; // advance to the next character

str2++;

}

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

//The expression after return is not syntax for bool expression

}

//This function definition does not compare the length of two C strings at all

int main()

{

char a[15] = "pointy";

char b[15] = "pointless";

if (areSame(a,b))

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

}

FIXED VERSION:

#include <iostream>

#include <cstring>

using namespace std;

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

{

if (strlen(str1) != strlen(str2))

{

return false;

}

int i = 0;

while (str1[i] != ‘\0’ && str2[i] != ‘\0’)

{

if (str1[i] == str2[i])

{

i++;

}

else

{

return false;

}

}

return true;

}

e.

ORIGINAL:

int\* makeCubes(int& m)

{

int anArray[100];

for (int j = 0; j < 100; j++)

anArray[j] = j \* j \* j;

m = 100;

return anArray;

}

void f()

{

int junk[100];

for (int k = 0; k < 100; k++)

junk[k] = 123400000 + k;

}

int main()

{

int n;

int\* ptr = makeCubes(n);

f();

for (int i = 0; i < 3; i++)

cout << ptr[i] << ' ';

for (int i = n-3; i < n; i++)

cout << ptr[i] << ' ';

cout << endl;

}

//(1) anArray is a local variable to the function, makesCubes(). After the implementation of makeCubes, spaces for anArray[] will be reallocated to variables used in f(). Thus, pointers and the elements that pointers pointing to are not guaranteed to be the original ones.

//(2) The last element in anArray is 99^3, not 100^3 = 1000000

//(3) Suppose without f(), two for-loops in main() will print values of the first three elements: 0, 1, 8, and the values of the last three elements: 97^3, 98^3 and 99^3. This is not the intended order.

2.

a. string\* dessert = new string;

b. string desserts[5];

c. dessert = desserts[4];

d. \* desserts = “ice cream”;

e. \* (desserts + 3) = “cookies”;

f. dessert -= 3;

g. dessert[1] = “pizookie”;

h. dessert[0] = “strawberries”;

i. bool b = ((\* dessert) == (\* (desserts + 1)))

j. bool d = (dessert == desserts[0])

3.

a.

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

{

double tot = 1.0;

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

{

tot \*= \*(ptr + i);

}

return pow(tot, 1.0/numScores);

}

b.

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

{

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

{

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

{

str += k;

return str;

}

}

return nullptr;

}

c.

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

{

while (\*str != 0)

{

if (\*str == chr)

{

return str;

}

else

{

str++;

}

}

return nullptr;

}

4.

#include <iostream>

using namespace std;

int\* largess(int\* a, int\* b)

{

if (\*a > \*b)

return a;

else

return b;

}

void swap1(int\* a, int\* b)

{

int\* temp = a;

a = b;

b = temp;

}

//This function exchanges two pointers a and b but does not exchange the value within the addresses that pointers pointing to

void swap2(int\* a, int\* b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int main()

{

int array[6] = { 5, 3, 4, 17, 22, 19 };

int\* ptr = largess(array, &array[2]);

//define a pointer ptr which points to array[0]

\*ptr = -1;

//array[0] = -1

ptr += 2;

//ptr now points to array[2]

ptr[1] = 9;

//array[3] = 9;

\*(array+1) = 79;

//array[1] = 79

cout << &array[5] - ptr << endl;

//The output here is 3, since the result of the subtraction of two pointers is the indexed elements between them. ptr is a pointer pointing to array[2], so there are 3 elements between array[2] and array[5]

//Another explanation:

ptr += 3 will result in ptr to be a pointer pointing to array[5], thus the result of the subtraction is 3

swap1(&array[0], &array[1]);

//swap1 actually exchanges two pointers, not the values

swap2(array, &array[2]);

//swap2 only swaps the values of elements that two pointers pointing to. array[0] = 4 and array[2] = -1

for (int i = 0; i < 6; i++)

cout << array[i] << endl;

//In conclusion, array[0] = 4, array[1] = 79, array[2] = -1, array[3] = 9, array[4] = 22, array[5] = 19. The output should be:

4

79

-1

9

22

19

}

5.

#include <iostream>

using namespace std;

void removeF(char\* str)

{

char\* charPtr = str;

while (\*charPtr != 0)

{

if (\*charPtr == ‘f’ || \*charPtr == ‘F’)

{

if (\*(charPtr + 1) == 0)

{

\*charPtr = 0;

continue;

}

while (\*(charPtr + 1) != 0)

{

\*charPtr = \*(charPtr + 1);

charPtr++;

}

\*charPtr = 0;

charPtr = str;

continue;

}

charPtr++;

}

}