Calvin Liu

804182525

1a)This program is supposed to write **30 20 10**, one per line. Find all of the bugs and show a fixed version of the program:

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;

while (ptr <= (arr + 2))

{

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

ptr++;

}

}

1b)

void findDisorder(int arr[], int n, int\*& p)

{

for (int k = 1; k < n; k++)

{

if (arr[k] < arr[k-1])

{

p = arr + k;

return;

}

}

p = NULL;

}

You have to pass the pointer parameter by reference, not by value

1c)

The problem is that the pointer p is uninitialized and dereferencing the resultPtr could lead to undefined behavior. One way to fix it is:

int main()

{

int k;

double\* p = &k;

hypotenuse(1.5, 2.0, p);

cout << "The hypotenuse is " << \*p << endl;

}

1d)

The test str1 != 0 is asking if the str1 pointer itself has a value different from the null pointer.

The test we want, though, is to see if the character *pointed to* by str1 is different from the zero byte that marks the end of a C string. (The same applies to str2.)

Similarly, the test str1 != str2 is asking whether those two pointers have different values (i.e., they point to different places). But what should be tested is whether the characters they point to have different values. (The same applies to str1 == str2.)

The while loop must be changed to:

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

The statement

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

return false;

should be changed to

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

return false;

return str1 == str2 should be changed to

return \*str1 == \*str2

1e)

The array arr will go away once you leave the function, but will return a pointer to the beginning of the arr. Attempting to follow the pointer will yield undefined behavior.

2a) string\* fp;

2b) string fish[5];

2c) fp = fish + 4;

2d) \*fp = “salmon”;

2e) \*(fish + 3) = “yellowtail”;

2f) fp -= 3;

2g) fp[1] = “eel”;

2h) fp[0] = “tuna”;

2i) bool b = fp == fish

2j) bool d = \*fp == \*(fp+1);

3a)

double computeAverage(const double\* scores, int nScores)

{

int k = 0;

double tot = 0;

while (k != nScores)

{

tot += \*(scores + k);

k++;

}

return tot/nScores;

}

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 NULL;

}

3c)

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

{

if (\*a < \*b)

return a; //returns a pointer based on what

else //the address contains

return b;

}

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

{

int\* temp = a; //does nothing really, pass by value

a = b;

b = temp;

}

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

{

int temp = \*a; //swap the actual value

\*a = \*b; //in spots [0] and [2]

\*b = temp; //because deref array is pass by ref

}

int main()

{

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

int\* ptr = minimart(array, &array[2]); //ptr now points to index 2

ptr[1] = 9; //17 now becomes 9

ptr += 2; //ptr now points to index 4

\*ptr = -1; //22 becomes -1

\*(array+1) = 79; //3 becomes 79

cout << "diff=" << &array[5] - ptr << endl; //will cout diff=1 because ptr 5 - ptr 4

swap1(&array[0], &array[1]); // swaps [0] and [1]’s copy’s addresses but the function ends and nothing really happens because it isn’t pass by reference

swap2(array, &array[2]);

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

cout << array[i] << endl; //print out the array values in order which is (diff=1, 4, 79, 5, 9, -1, 19 with endl’s)

}

void deleteG(char\* cstr)

{

char\* ptr = cstr;

for(; \*cstr != '\0'; cstr++)

{

if(\*cstr != 'g' && \*cstr != 'G')

{

\*ptr = \*cstr;

ptr++;

}

}

\*ptr = ‘\0’;

}