Sari Abu-Hamad

CS31

David A Smallberg

2. The program does not originally correctly set the value of arr[1] to 20 as it references \*ptr + 1 which is not a variable as it is adding 1 to the value of ptr currently and since it is a value itself (and not a variable), it cannot be assigned a value. In order to solve this, ptr is incremented and thus it is referencing the value at arr[1] when called as \*ptr. This same strategy is repeated to access arr[2]. Another bug is in the display as the value of ptr is subtracted before displaying its value and thus displays the value at arr[1] first; this can be solved by subtracting it after displaying the value at ptr.

{

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

int\* ptr = arr;

\*ptr = 10; // set arr[0] to 10

ptr++;

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

ptr++;

\*ptr = 30; // set arr[2] to 30

while (ptr >= arr)

{

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

ptr--;

}

cout << endl;

}

1. The function doesn’t work because the pointer p is pass by value so its value in the main function doesn’t change. To fix this, it must be converted to pass by reference by adding a “&” symbol in the function definition.

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 = nullptr;

}

1. The variable p is uninitialized so the pointer is set as a null pointer when passed to the function. It must be given a memory location and thus initialized to a specific variable for its value to be changed.

int main()

{

double b = 0;

double\* p = &b;

hypotenuse(1.5, 2.0, p);

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

}

1. The problem is that the function immediately returns false on the first iteration of the while loop since you cannot compare the values of each using “!=” and should use the strcmp function if they were to do that, or directly compare character by character as I implement below.

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

{

int n = 0;

while (str1[n] != '\0' && str2[n] != '\0') // zero bytes at ends

{

if (str1[n] != str2[n]) // compare corresponding characters

return false;

n++; // advance to the next character

}

return true; // both ended at same time?

}

1. The array within the function is a local variable so when the pointer is returned, it is referencing an array that does not exist in the int main and thus the memory is not allocated for it.

**2.**

1. string\* fp;
2. string fish[5];
3. fp = &fish[4];
4. \*fp = "yellowtail";
5. \*(fish+3)="salmon";
6. fp=fp-3;
7. fp[1]="carp";
8. fp[0]="smelt";
9. bool d=(fp==fish);
10. bool b=(\*fp==\*(fp+1));

**3.**

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

{

const double\* ptr = scores;

double tot = 0;

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

{

tot += \*(ptr+i);

}

return tot/nScores;

}



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;

}



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

{

while((\*str)!='\0')

{

if ((\*str) == chr)

{

return (str);

}

str++;

}

return nullptr;

}

**4.**

The array starts off with the values 5, 3, 4, 17, 22, 19.

int\* ptr = minimart(array, &array[2]); passes the pointers pointing to 5 and 4 in the array to the function minimart which returns the pointer pointing to the smaller value thus it returns the pointer pointing to array[2] since 4 < 5. ptr now points to array[2].

Now ptr[0] points to 4 which is at position arr[2] due to this function and the next line, ptr[1] = 9; sets the value 1 position past that of ptr[0] (which is now at the address of 4 in the array) to 9. Thus the array now holds: 5, 3, 4, 9, 22, 19

Next, ptr += 2; moves the position being referenced in the array to 2 ahead of the current position (ptr[0] or arr[3]) and thus it is referencing arr[4] and sets the value at that position to -1 with \*ptr = -1;

Thus the array now holds: 5, 3, 4, 9, -1, 19

\*(array+1) = 79; sets the value of the second element in the array to 79

Thus the array now holds: 5, 79, 4, 9, -1, 19

cout << "diff=" << &array[5] - ptr << endl; creates the first line of output and displays diff=1. This value is found by subtracting ptr (which is currently equivalent to a pointer pointing to the memory location of array[4]) from &array[5] (which is currently equivalent to a pointer pointing to the location of array[5]). The difference between the memory locations of array[5] and array[4] is 4 and since memory is stored in bits, when being displayed they are displayed as bytes and 4 bits are equal to 1 byte thus 1 is displayed.

swap1(&array[0], &array[1]); does nothing to change the values as the memory locations are being swapped and those being swapped are local to the swap1 function and thus does not affect the values of anything in the array. Thus the values of the array are still: 5, 79, 4, 9, -1, 19

swap2(array, &array[2]); this function swaps the values of the two positions in the array and array references a pointer that points to the memory location of array[0] and &array[2] references a pointer that points to the memory location of array[2]. This function works because it swaps the values and not the memory locations which would be local to the function.

Thus the values of the array are currently: 4, 79, 5, 9, -1, 19.

The for loop that is next outputs the array value by value line by line so the overall output of the whole program including the diff output earlier is finally:

diff=1

4

79

5

9

-1

19

**5.**

void deleteG(char\* cstr){

char \*tempPointer;

while(\*cstr != '\0')

{

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

{

cstr++;

}

else

{

tempPointer = cstr;

while(\*tempPointer != '\0')

{

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

tempPointer++;

}

}

}

}