* 1. 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 (error, not legal syntax)  
 ptr += 2;  
 **ptr[0] = 10;** // set arr[2] to 10 (error, doesn’t set arr[2] to 10)

**while (ptr >= arr)** (error, prints the array in reverse) **{  
 ptr--;  
 cout << \*ptr << endl; // print values  
 }** }

Solution:

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) = 10; //set arr[2] to 10

while (ptr <= (arr+2))

{

cout << \*ptr << endl;

ptr++;

}

}

* 1. The findMax function is that the pointer is passed by value instead of passed by reference. Thus, it creates a copy of the pointer, not the actual pointer itself. To fix this, simply change the pointer to be passed by reference.

void findMax(int arr[], int n, int\* &pToMax)

* 1. The main function’s problem is that the pointer variable is declared but not initialized, so it really isn’t pointing at anything in memory. A simple fix is to initialize a variable to hold the int you want cubed. Then, initialize the pointer to the address of that variable. Then pass the int variable to the computeCube function.

void computeCube(int n, int\* ncubed)

{

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

}

int main()

{

int x = 5;

int\* ptr = &x;

computeCube(x, ptr);

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

}

* 1. The function’s problem is the way the arrays are being compared. The function isn’t actually comparing each individual character, but actually the base addresses of the array in memory, so it will always evaluate to false. The way to fix this problem is to pass in pointers to the arrays. Then, dereference the pointers to check for the zero byte. Then dereference the pointers to compare the values. Finally, dereference the pointers for the final comparison.

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?

}

* 1. The problem with this program is the scope of the anArray variable in the getPtrToArray function. AnArray is a local variable, so the pointer is pointing to the first element of that array, but once the function is done, all the elements within anArray are gone and set to undefined values. The program fails because it relies on undefined behavior and prints out random values.
  2. double\* cat;
  3. double mouse[5];
  4. cat = mouse + 4;
  5. \*cat = 25;
  6. \*(mouse + 3) = 42;
  7. cat -= 3;
  8. cat[1] = 54;
  9. cat[0] = 27;
  10. bool b = (\*cat == \*(cat + 1));
  11. bool d = (cat == &mouse[0]);
  12. double mean(const double\* scores, int numScores)

{

const double\* ptr = scores;

double tot = 0;

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

tot += \*(ptr + x);

return tot/numScores;

}

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

{

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

if (\*str == chr)

return str;

return nullptr;

}

int\* maxwell(int\* a, int\* b) //returns pointer that points to the greater dereference value

{

if (\*a > \*b)

return a;

else

return b;

}

void swap1(int\* a, int\* b) //swaps the addresses of the values but doesn't have an effect on the output

{

int\* temp = a;

a = b;

b = temp;

}

void swap2(int\* a, int\* b) //actually swaps the values the pointers are pointing to

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int main()

{

int array[6] = { 5, 3, 4, 17, 22, 19 }; //initiliazes an array of ints size 6

int\* ptr = maxwell(array, &array[2]); //initiliazes a pointer that points to &array[0]

\*ptr = -1; //sets the value at &array[0] to -1

ptr += 2; //moves the pointer to &array[2]

ptr[1] = 9; //sets the value of &array[3] to 9

\*(array+1) = 79; //sets the value of &array[1] to 79

cout << &array[5] - ptr << endl; //prints the difference of &array[5] and &array[2], which is 3 because the address locations of an array are sequential

swap1(&array[0], &array[1]); //swaps the addresses of &array[0] and &array[1]

swap2(array, &array[2]); //swaps the values of &array[0] and &array[2]

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

cout << array[i] << endl; //prints out the each value of the array, which is now: 4, 79, -1, 9, 22, 19

}

The output is:

3

4

79

-1

9

22

19

void removeS(char\* msg)

{

char\* result = msg;

for (;\*msg != 0; msg++)

{

if (\*msg == 'S' || \*msg == 's') {

continue;

}

else

{

\*result = \*msg;

result++;

}

}

\*result = 0;

}