Atij Mahesh

Professor Smallberg

Computer Science 31

21 November 2022

Project 6

1a)

| int main() {  int arr[3] = { 5, 10, 15 };  int\* ptr = arr;   \*ptr = 10; // set arr[0] to 10  \*(ptr + 1) = 20; // set arr[1] to 20  ptr += 2;  \*ptr = 30; // set arr[2] to 30    while (ptr >= arr) {  cout << \*ptr << endl;  ptr--;  } } |
| --- |

Bugs: Setting up the array in reverse order to print, putting parentheses around prt+1, putting p-- after the print statement.

1b)

The issue with findMax is that the pointer parameter (pToMax) is not a reference parameter, and thus cannot change the value of the input parameter in main. A way to fix this is to make the pointer parameter a reference parameter in findMax.

| 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;  }  }   int main()  {  int nums[4] = { 5, 3, 15, 6 };  int\* ptr = &nums[0];   findMax(nums, 4, ptr);  cout << "The maximum is at address " << ptr << endl;  cout << "It's at position " << ptr - nums << endl;  cout << "Its value is " << \*ptr << endl;  } |
| --- |

1c)

The issue in main is that the pointer never gets initialized, and thus doesn’t “point” to anything. A way to fix this is to have the pointer point to the number cubed.

| void computeCube(int n, int\* ncubed)  {  \*ncubed = n \* n \* n;  }   int main()  {  int n = 5;  int\* ptr = &n;  computeCube(n, ptr);  cout << "Five cubed is " << \*ptr << endl;  } |
| --- |

1d)

strequal does not work as the function never compares the actual character held in the c-string, but only the addresses of the characters. To fix this, I used pointers in the comparison statements instead of comparing the addresses.

| bool strequal(const char str1[], const char str2[])  {  while (\*str1 != 0 && \*str2 != 0) // zero bytes at ends  {  if (\*str1 != \*str2) // compare corresponding characters  return false;  str1++; // advance to the next character  str2++;  }  return \*str1 == \*str2; // both ended at same time?  }   int main()  {  char a[15] = "Chang";  char b[15] = "Zhang";   if (strequal(a,b))  cout << "They're the same person!\n";  } |
| --- |

1e)

In function getPtrToArray, anArray is not a reference parameter and is local to the function, and when the function ends, the contents and the array itself are deleted from memory. Therefore, once getPtrToArray (and nochange) finish running, anArray is non-existent, and ptr in main only points to the array’s memory address, which doesn’t have anything useful to the program and may even store other data.

2a) double\* cat;

b) double mouse[5];

c) cat = mouse + 4;

d) \*cat = 25;

e) \*(mouse+3) = 54;

f) cat -= 3;

g) cat[1] = 42;

h) cat[0] = 17;

i) bool d = (cat == mouse);

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

3a)

| double mean(const double\* scores, int numScores)  {  double tot = 0;  int i = 0;  while (i<numScores)  {  tot += \*(scores+i);  i++;  }  return tot/numScores;  } |
| --- |

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 nullptr;  } |
| --- |

3c)

| const char\* findTheChar(const char str[], char chr)  {  while(\*str) {  if(\*str == chr) return str;  str++;  }  return nullptr;  } |
| --- |

4)

Code Output:

3

4

79

-1

9

22

19

Upon initializing the array, the following line in main sets ptr to the output of function maxwell. maxwell inputs the 1st and third elements of the array and returns a pointer to the index containing a greater value, which in this case, is index 0. The next line stores -1 “inside” the pointer, which in the main method, is inside the array, and thus stores -1 and index 0 of the array. The pointer then advances two indexes in the array (to index 2), and sets the following index (index 3) to 9. The following line states \*(array+1) = 79, and since “array” points to the 0th index of the array, this line sets the array at index 1 to 79.

Current Array Values: {-1, 79, 4, 9, 22, 19} Pointer ptr points to 2nd index.

The following line is a print statement, which prints &array[5] - ptr. Both of these values are pointers, and only utilize indexes by themselves, so &array[5] = 5, ptr = 2, and thus 5-2 = 3, which will be printed and have a new line added to the end. The next line involves swap1, which intends to swap the elements the pointers point to, but since they are passed by value and swap1 does not contain reference parameters, changes made in swap1 are local to the method and do not affect the values of pointers or the array in main. The call to swap2 in the main does not pass both pointers as value, and successfully swaps elements at index 0 and 2.

Current Array Values: {4, 79, -1, 9, 22, 19}

The final line prints the array, with each element on its own line, which matches up with the current array values presented above.

5)

| void removeS(char\* c) {  while(\*c) {  if(tolower(\*c) == 's') {  char\* temp = c;  while(\*(temp+1)) {  //move elements back  \*temp = \*(temp+1);  temp++;  }  \*temp = 0;  }  else  c++;  } } |
| --- |