Alex Chen

CS 31

Project 6

1. 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 ++;

}

}

The findMax function would not allow pToMax to point to the max element in the array because pToMax will not keep this value permanently. ptr is passed as a pointer, but it needs to be passed as by reference so that the address of the max value in array will be memorized, allowing the main function to be able to access this value.

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;

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;

}



The main function has a problem because the ncubed parameter of the computeCube function is a pointer. Therefore, passing a reference is meaningful, rather than another pointer. The reference can refer to an address in the memory where ncubed can be stored. A pointer would not be able to do so.

Thus, so solve this issue, in the main function, ptr can be declared as an int, and then passed as a reference when computeCube is called. In the third line of the main function, ptr can be used to cout the value calculated in computeCube. \*&ptr can also be used.

void computeCube(int n, int\* ncubed)

{

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

}

int main()

{

int ptr;

computeCube(5, &ptr);

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

}

The problem with the implementation of strequal is that the actual characters are not being compared – the addresses are being compared. Str1 and str2 refer to the actual cstring, rather than the characters. To actually refer to the characters, dereferencing needs to occur, by adding a \* before str1 and str2 in the loops and final return statement.

// return true if two C strings are equal

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?

}

int main()

{

char a[15] = "Chen";

char b[15] = "Cheng";

if (strequal(a,b))

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

}



The function probably does not return 100 99 98 3 2 1 because when getPtrToArray is called, the created array, anArray, cannot be accessed outside of the function. Thus, when ptr[i] is called, the values cannot be accessed, causing the function to output garbage values.

1. 1. double\* cat;
   2. double mouse[5];
   3. cat = mouse + 4;
   4. \*cat = 25;
   5. \*(mouse+3) = 54;
   6. cat -= 3;
   7. cat[1] = 27;
   8. cat[0] = 42;
   9. bool b = (\*cat == \*cat[1]);
   10. bool d = (cat == &mouse[0]);

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

{

const double\* ptr = scores;

int i = 0;

double tot = 0;

while (ptr + i != scores + numScores)

{

tot += \*(ptr+i);

i++;

}

return tot/numScores;

}



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

{

int k = 0;

while (\*str != ‘\0’)

{

str++;

k++;

}

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

{

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

{

return str+i;

}

}

return nullptr;

}



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

{

while (\*str != ‘\0’)

{

if (\*str == chr)

{

return str;

}

str++;

}

return nullptr;

}

1. 1. The output of the program is
      1. 3, 4, 79, -1, 9, 22, 19. The numbers separated by commas are each outputted on a new line.
      2. 3 is outputted from cout << &array[5] - ptr << endl; because &array[5] references the address of the 5th element of array, while ptr points to the address of the 2nd element of array, which happened with the code “ptr += 2”. Thus, 5-2 is equal to 3.
      3. swap1(&array[0], &array[1]); will cause &array[0] to point to &array[1] and &array[1] to point to @array[0]. However, these do not permanently save. So after swap1 is called, essentially nothing has changed. Thus, when swap2 is called, in which the first and third element of array are swapped permanently, due to dereferencing, rather than just referencing the address, the first and third element are swapped, and the next six elements printed out are the elements of array, which are -1, 79, 4, 9, 22, and then 19.

void removeS(char\* msg)

{

For (char\* p = msg; \*p; )

{

While (\*p == ‘s’ || \*p == ‘S’)

{

++p;

}

\*msg++ = \*p++;

}

\*msg = ‘\0’;

}