Name: Morgan Lim

UID: 105 168 668

1a.

#include <iostream>

using namespace std;

int main()

{

int arr[4] = { 0, 1, 2, 3 };

int\* ptr = arr;

\*ptr = arr[1]; // set arr[0] to 1

\*(ptr + 1) = arr[0] \* 10; // set arr[1] to 10

ptr[2] = arr[1] \* 10; // set arr[2] to 100

ptr[3] = 1000; // set arr[3] to 1000

ptr += 4;

while (ptr > arr)

{

ptr--;

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

}

cout << endl;

return(0);

}

1b.

The function did not work because ptr was not being passed by reference to the function.

Even though pointers are a reference to location in memory, their value is simply this location using a pointer as an argument passes the value of the location, but not the pointer itself. To fix this, I simply made the second parameter pass the pointer by reference.

#include <iostream>

using namespace std;

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

{

p = nullptr; /// default value if there isn't a 0 in the array at all

for (int k = n - 1; k >= 0; k--)

{

if (arr[k] == 0) // found an element whose value is 0

{

p = arr + k; // change the value of p

break; // stop looping and return

}

}

}

int main()

{

int nums[6] = { 10, 20, 0, 40, 30, 50 };

int\* ptr = nullptr;

findLastZero(nums, 6, ptr);

if (ptr == nullptr)

{

cout << "The array doesn't have any zeros inside it." << endl;

}

else

{

cout << "The last zero is at address " << ptr << endl;

cout << "It's at index " << ptr - nums << endl;

cout << "The item's value is " << \*ptr << " which is zero!" << endl;

}

return(0);

}

1c

The function does not work because the pointer p is declared but not initialized. To fix this, I simply initialized p.

/\*#include <iostream>

using namespace std;

void biggest(int value1, int value2, int \* resultPtr)

{

if (value1 > value2)

{

\*resultPtr = value1;

}

else

{

\*resultPtr = value2;

}

}

int main()

{

int x;

int\* p = &x;

biggest(15, 20, p);

cout << "The biggest value is " << \*p << endl;

return(0);

}

\*/

1d

The problem with the function is that c strings cannot be compared directly. To fix this problem, I used pointers to point to each char in the c string to be able to compare them.

//return true if two C strings are equal

#include <iostream>

using namespace std;

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

{

const char\* ptr1 = str1;

const char\* ptr2 = str2;

bool result = true;

while (\*ptr1 != 0 && \*ptr2 != 0) // zero bytes at ends

{

if (\*ptr1 != \*ptr2) // compare corresponding characters

{

result = false;

break;

}

ptr1++; // advance to the next character

ptr2++;

}

if (result)

{

result = (\*ptr1 == \*ptr2); // both ended at same time?

}

return(result);

}

int main()

{

char a[10] = "pointy";

char b[10] = "pointless";

if (match(a, b))

{

cout << "They're the same!" << endl;

}

}

1e: When the pointer is declared and initialized it is essentially being set to the value array[0], which is 1. The program is trying to find the location of the first index of the array and to print from there, but what is actually happening is computeFibonacciaSequence is returning an int value rather than where array[0] is.

2 1)f 2)g 3)a 4)b 5)d 6)c 7)b 8)e 9)h

3.

#include <iostream>

using namespace std;

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

{

if (\*a < \*b) //1. the values \*a = 5 and \*b = 4 are compared and since this //statement is not true &array[2] is returned

return a;

else

return b;

}

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

{

int\* temp = a; //2. here a and b are new pointers that have been initialized for //this function, since they are not pass by reference. The values get swapped but do not

//affect array in main since the values of where the pointers were passed to this //function, not the actual pointers

a = b;

b = temp;

}

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

{

int temp = \*a; //3. here, the values actually get swapped, since, by using the

//dereferencing operator, the actual location and value of a and b are being accessed and

//changed.

\*a = \*b;

\*b = temp;

}

int main()

{

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

int\* ptr = minimart(array, &array[2]); //1. minimart is called &array[2] is //returned so ptr is at array[2]

ptr[1] = 9; //the value of ptr[1] which was 17 is now 9; {5,3,4,9,22,19}

ptr += 2; //ptr is not pointing at array[4] which has the value of 22

\*ptr = -1; //the value of array[4] now equals -1; {5,3,4,9,-1,19}

\*(array + 1) = 79; //the value of array[1] now equals 79; {5,79,4,9,-1,19}

cout << "diff=" << &array[5] - ptr << endl; //printing the difference between the //value of the location of array[5] and the value of printer which is 4 bytes which is //just 1 since array is of type int

//output: diff=1

swap1(&array[0], &array[1]); //2. swap1() is called but no change is made since

//what is passed is just the value of where array[0] and array[1] are

swap2(array, &array[2]); //3. swap2() is called and the values of array[0] and //array[2] are switched; {4,79,5,9,-1,19}

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

cout << array[i] << endl;

//output:

//4

//79

//5

//9

//-1

//19

return(0);

}

4.

#include <iostream>

using namespace std;

void deleteDigits(char\* c)

{

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

{

if (\*c >= '0' && \*c <= '9')

{

char \*temp = c;

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

{

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

c++;

}

c = temp;

}

else

{

c++;

}

}

}