Passing Pointers as Arguments

In this lesson, we'll try passing pointers as input arguments to functions!

WE'LL COVER THE FOLLOWING

Pointer Scope

As we've learned in the Function section, function arguments are usually local in scope. Once we return from the function, all its variables are deleted forever.

One solution was to pass external variables by reference using the `&` operator. This would alter the values of the variables outside the function scope.

```
#include <iostream>
using namespace std;

void square(int &a){ // the function takes an integer and
  //replaces the value with its square
  a = a * a;
}

int main() {
  int a = 5;
  cout << "The value of a before the function call: " << a << endl;
  square(a);
  cout << "The value of a after the function call: " << a << endl;
}
```

All of this becomes simpler with the use of pointers. By definition, pointers store and pass references (addresses) of other variables or objects to the function. If a pointer is passed to a function, the function can directly manipulate the value the pointer points to.

```
#include <iostream>
using namespace std;

void square(int *a){ // the function takes a pointer and replaces the value with its square
  if(a != NULL){
    *a = (*a) * (*a);
  }
}

int main() {
  int *p = new int(5);
  cout << "The value of p before the function call: " << *p << endl;
  square(p);
  cout << "The value of p after the function call: " << *p << endl;
}
```

In this sense, pointers can act as global variables which can be accessed by all functions.

Arrays as Input Arguments

Since arrays are basically pointers to a block of memory, they are also passed by reference to functions. We do not need to use the `&` operator.

```
#include <iostream>
using namespace std;

void doubleValues(int arr[], int size){
// a function which doubles the values of all the elements in an array
  for (int i = 0; i < size; i++){
    arr[i] = arr[i] * 2;
  }
}

int main() {
  const int size = 10;
  int arr[size];

  cout << "Original Values: ";
  for (int i = 0; i < size; i++){
    arr[i] = i;
    cout << arr[i] << ", ";
  }
  cout << endl;

  doubleValues(arr, size);

  cout << "Doubled Values: ";
  for (int i = 0; i < size; i++){
    cout << arr[i] << ", "; // the original array has been changed
  }
}
```

So far, we’ve seen how pointers behave with functions. In the next lesson, we will learn how arithmetic operations are performed on pointers.