Pointer Arithmetic

This lesson highlights the different arithmetic operations we can perform on pointers.

WE'LL COVER THE FOLLOWING

Basic Addition and Subtraction

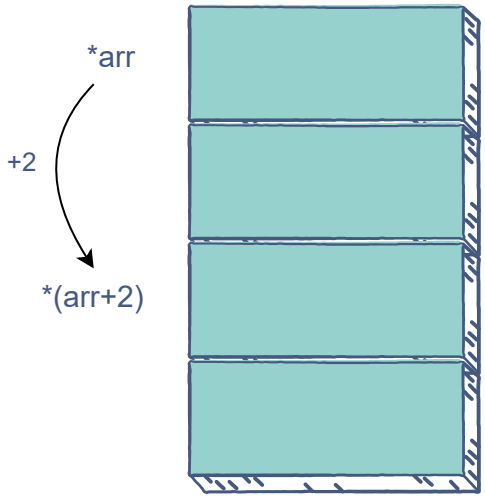
Consider a simple pointer, `p`, which points to a value of 10. What would happen if we increment it by `1`?

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

int main() {
  int *p = new int(10);
  cout << "p: " << p << endl;
  cout << "value of p: " << *p << endl << endl;

  p = p + 1;
  cout << "p: " << p << endl;
  cout << "value of p: " << *p << endl << endl;
}
```

As we can observe, incrementing `p` actually increments its address. Since it is an integer pointer, the address jumps 4 bytes ahead (the size of an integer is 4 bytes).

The value of `p` becomes `0` which represents `null` or an empty space. This can be a dangerous practice since the new space may already be in use by another variable or process. Accessing this memory could cause a crash!

Arrays and Pointer Arithmetic



Arrays can be represented using pointer arithmetic. We discovered that an array variable itself is a pointer to the zeroth index of the array. Hence, for an array named `arr`, `*arr` is equivalent to `arr[0]`.

`*(arr + 1)` corresponds to `arr[1]`.

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

int main() {
  int *arr = new int[10];

  for(int i = 0; i < 10 ; i++){
    arr[i] = i;
  }

  cout << "arr[0]: " << *arr << endl;
  cout << "arr[3]: " << *(arr + 3) << endl;
}
```

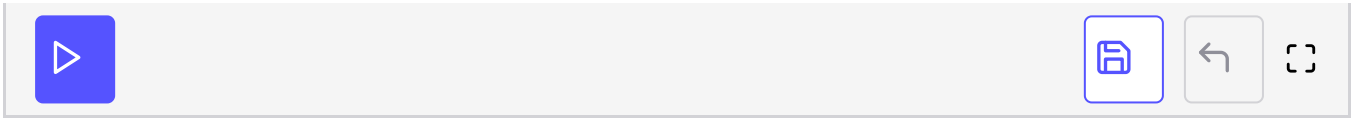
In the code above, **line 5** shows a dynamic array of size 10 being created by the pointer `arr`. A `for` loop is used to assign values at all the indices. In the end, we can see that `arr[0]` is equivalent to `arr* + 0`.

Operators Supported by Pointers

Pointers also support `++`, `--`, `+=` and `-=` operators.

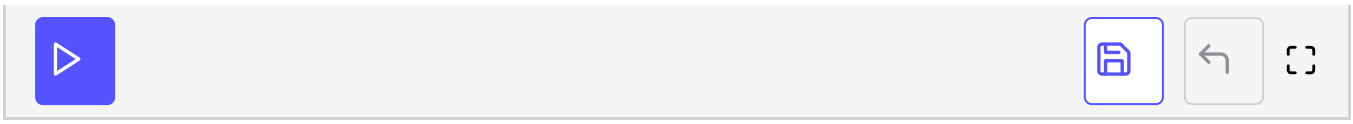
```
#include <iostream>
using namespace std;

int main() {
```

```
  int *p = new int(10);
  cout << p << endl; // The address p points to
  p++;
  cout << p << endl; // The pointer has jumped 4 bytes ahead to a new address
  p--;
  cout << p << endl; // The pointer jumps back to the original address
  p += 1;
  cout << p << endl; // Works the same way as p++
  p -= 1;
  cout << p << endl; // Works the same way as p--
}
```



Just as an integer can be added to a pointer to jump to a new address, two pointers can be added or subtracted with the result being an integer (potentially a long integer).

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

int main() {
  int *p, *q;
  cout << "p: " << p << endl;
  cout << "q: " << q << endl;
  cout << "p - q: " << p - q;
}
```



That ends our discussion on pointers. Experiment with the codes above to learn new and fun things about pointers. For now, we have all we need to move on to the next section.

Before we delve into classes, be sure to try out the quiz and exercises that lie ahead. They will reinforce all the concepts that we have covered in this lesson.