Pointers

In this lesson, we will discuss the functionality of pointers.

WE'LL COVER THE FOLLOWING

All the data created in a C++ program has an **address** in memory where it is stored. Each new data object is assigned a different address.

> A **pointer** holds the address of a value. It does not have a value of its own. Instead, it simply "points" to a value.

For each value of type `T`, there exists a pointer to that value.

A pointer can be created using the `*` operator and the address of value can be accessed using the `&` operator. Let's make all of this clearer with an example:

```
#include <iostream>

int main() {
  int i = 20; // A variable containing an integer
  int* iptr = &i; // A pointer that points to 'i'
  std::cout << iptr << std::endl; // Accessing the address stored in the pointer
  std::cout << *iptr << std::endl; // Accessing the value that the pointer points to
  i = 30;
  std::cout << *iptr << std::endl; // derefrencing the pointer reflects the change made in th
  *iptr = 50;  // pointer variables can be used to change the value
  std::cout << *iptr << std::endl; // Accessing the changed value
}
```

As we can see above, the value being pointed to can be accessed using the `*`

operator. This process is known as **pointer dereferencing**.

Pointer arithmetic

In C++, we can access a particular value in an array by using its index. Well, it turns out that indexing is just pointer arithmetic. The variable holding the array is actually a pointer pointing to the array.

The code,

```
intArray[i]
```

is equivalent to,

```
*(intArray + i)
```

`intArray` itself holds the address for the zeroth index of the array. Hence, `intArray + i` gives us the pointer to the i-th index. Then, the pointer is simply dereferenced to obtain the value. All of this happens when we use the `[]` brackets for indexing.

```
#include <iostream>

int main() {
  int intArray[] = {15, 30, 45, 60};
  std::cout << intArray[2] << std::endl;
  std::cout << *(intArray + 2) << std::endl;
}
```

Dynamic memory

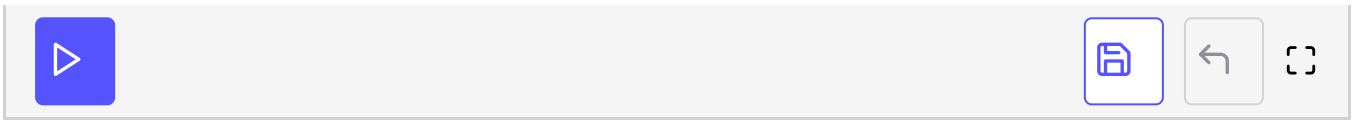
Pointers don't always have to point to an existing value. They can be used to create values and arrays in dynamic memory, which is really helpful during runtime.

To create dynamic data using pointers, we must use the `new` keyword. The type of the pointer and the value must be the same.

```
#include <iostream>
```

```
int main(){
  float* fltPtr = new float(50.505); // A float has been created in dynamic memory
  std::cout << *fltPtr << std::endl;

  int* intArray = new int[10]; // A dynamic array of size 10 has been created
  intArray[0] = 20;
  std::cout << intArray[0] << std::endl;
}
```



In the next lesson, we will study the null pointer.