Pointers Implementations

This lesson discusses using the new operator with pointers, referencing variables using & operator and use of pointers with arrays,

We'll cover the following

Allocating variables

In C++, a new *object*, *variable* or *array* can be created using the `new` operator, and freed with the `delete` operator.

```
int *ptr = new int;
  /* ... */
delete ptr;
```

The `new` operator allocates an object from the heap and optionally *initializes* it. When you have finished using it, you **must** `delete` it. Otherwise, the pointed memory is *inaccessible*, and the result is **memory leak**.

Let's take a look at an example to understand the concept better.

> Note: Before running the code below, click on the >_**STDIN** button. A field will appear where you can enter the **four** values that you want to `cin` with space in between. The reason for entering four values is because the `for` loop in the code below runs `num` times and `num` is set to `4`.

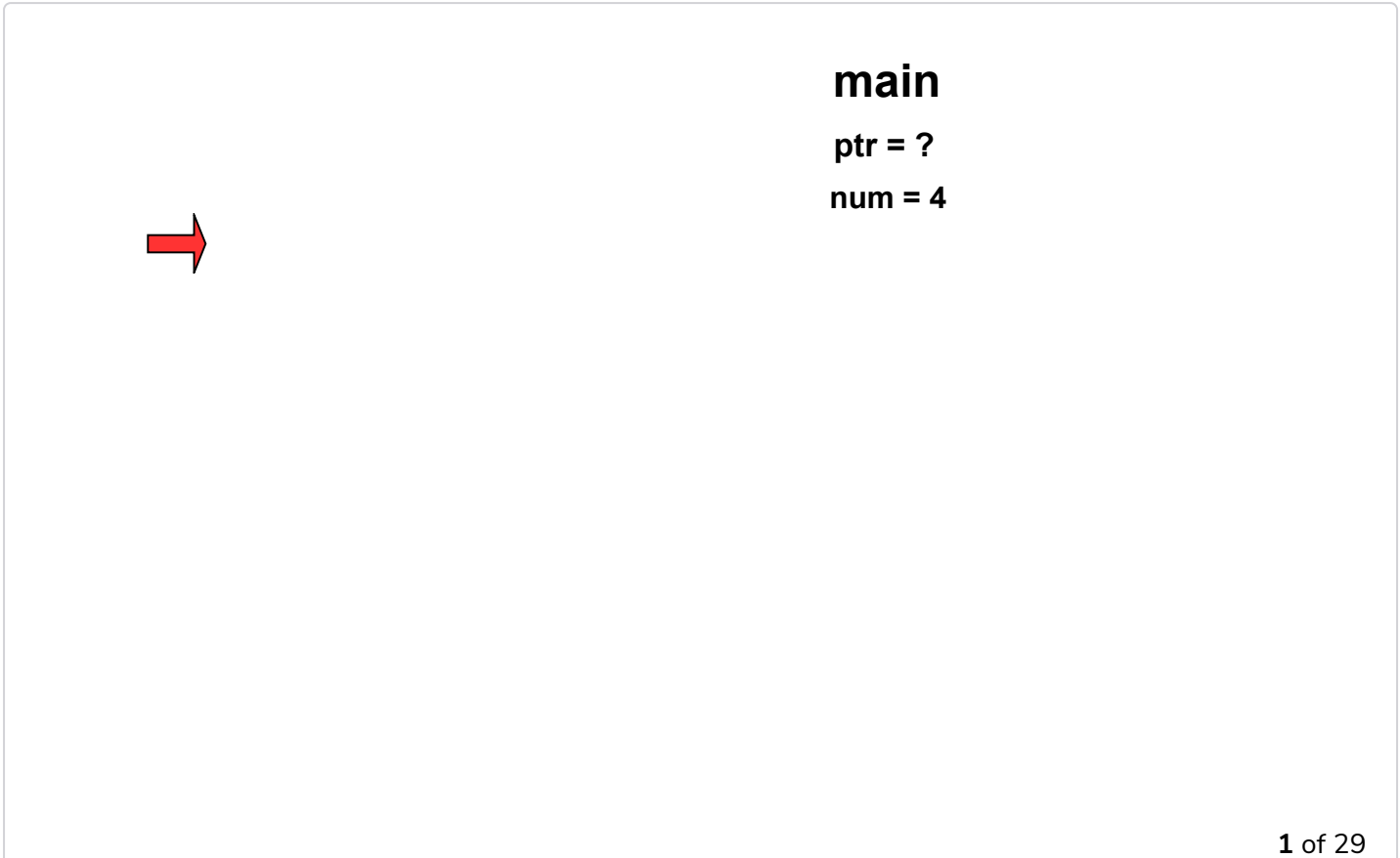
```
#include <iostream>
using namespace std;

int main() {
  int *ptr;      //creating a pointer named ptr
```

```
  int num = 4;    //intilizaing an int type variable named num with value 4
  cout << "The integer input is: " << num <<endl;
  ptr = new int[num];  //using new to initialize the ptr array with size num and dynamically alloc
  cout << "Input " << num << " integers\n";
  for (int i = 0; i < num; i++){
    ptr[i] = i+1;   //assigning value i+1 to the index i of ptr array
    cin >> ptr[i];  // we cin this value
  }
  cout << "Elements entered by you are\n";
  for (int i = 0; i < num; i++){
    cout << ptr[i] << endl;    //displaying the values that we passed through cin above
  }
  delete[] ptr;      // now that we are done using ptr, we delete the array
                    // and free the part of memory which was allocated to it in the heap
  return 0;
}
```

- When we allocate memory using `new`, it remains allocated until the program *exits*, but you can explicitly *deallocate* it with `delete` beforehand.
- The above example contains only **one** *function*, `main`, so memory will be *deallocated* after this program *exits*. However, we have used `delete` in line **18** as it is a good programming practice to *deallocate* memory which isn't required further in the program.

Down below is an illustration demonstrating the code above:



main

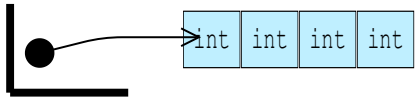
ptr = ?

num = 4

The integer input is: 4

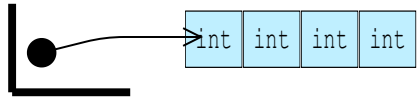
main Heap

ptr



main Heap

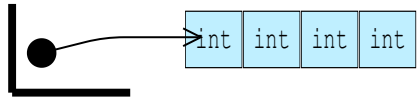
ptr



Input 4 integers

main Heap

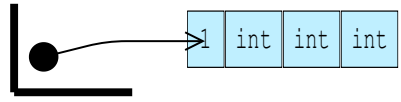
ptr



i = 0

main Heap

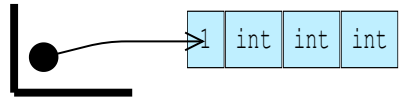
ptr



i = 0

main Heap

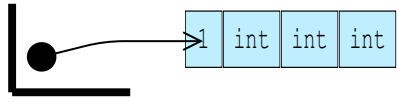
ptr



i = 0

main Heap

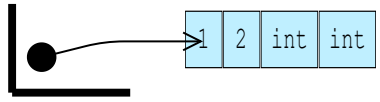
ptr



i = 1

main Heap

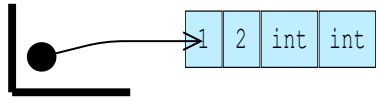
ptr



i = 1

main Heap

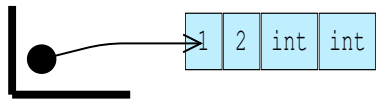
ptr



i = 1

main Heap

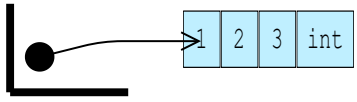
ptr



i = 2

main Heap

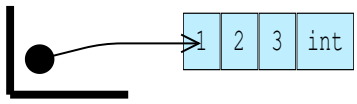
ptr



i = 2

main Heap

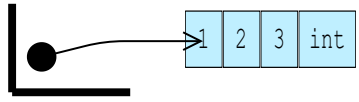
ptr



i = 2

main Heap

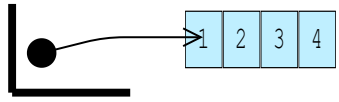
ptr



i = 3

main Heap

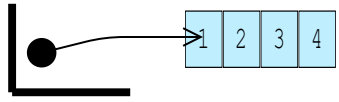
ptr



i = 3

main Heap

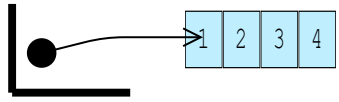
ptr



i = 3

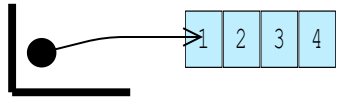
main Heap

ptr



main Heap

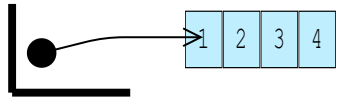
ptr



Elements entered by you are

main Heap

ptr

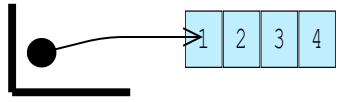


i = 0

Elements entered by you are

main Heap

ptr



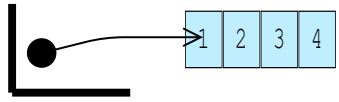
i = 0

Elements entered by you are

1

main Heap

ptr



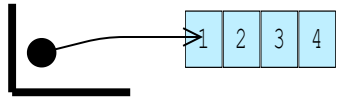
i = 1

Elements entered by you are

1

main Heap

ptr



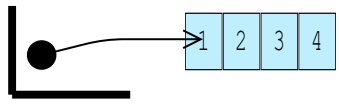
i = 1

Elements entered by you are

1
2

main Heap

ptr



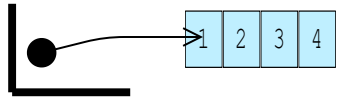
i = 2

Elements entered by you are

1
2

main Heap

ptr



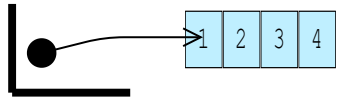
i = 2

Elements entered by you are

1
2
3

main Heap

ptr



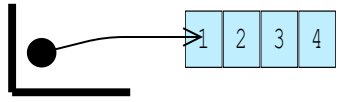
i = 3

Elements entered by you are

1
2
3

main Heap

ptr



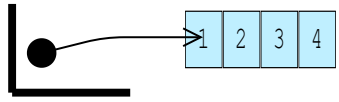
i = 3

Elements entered by you are

1
2
3
4

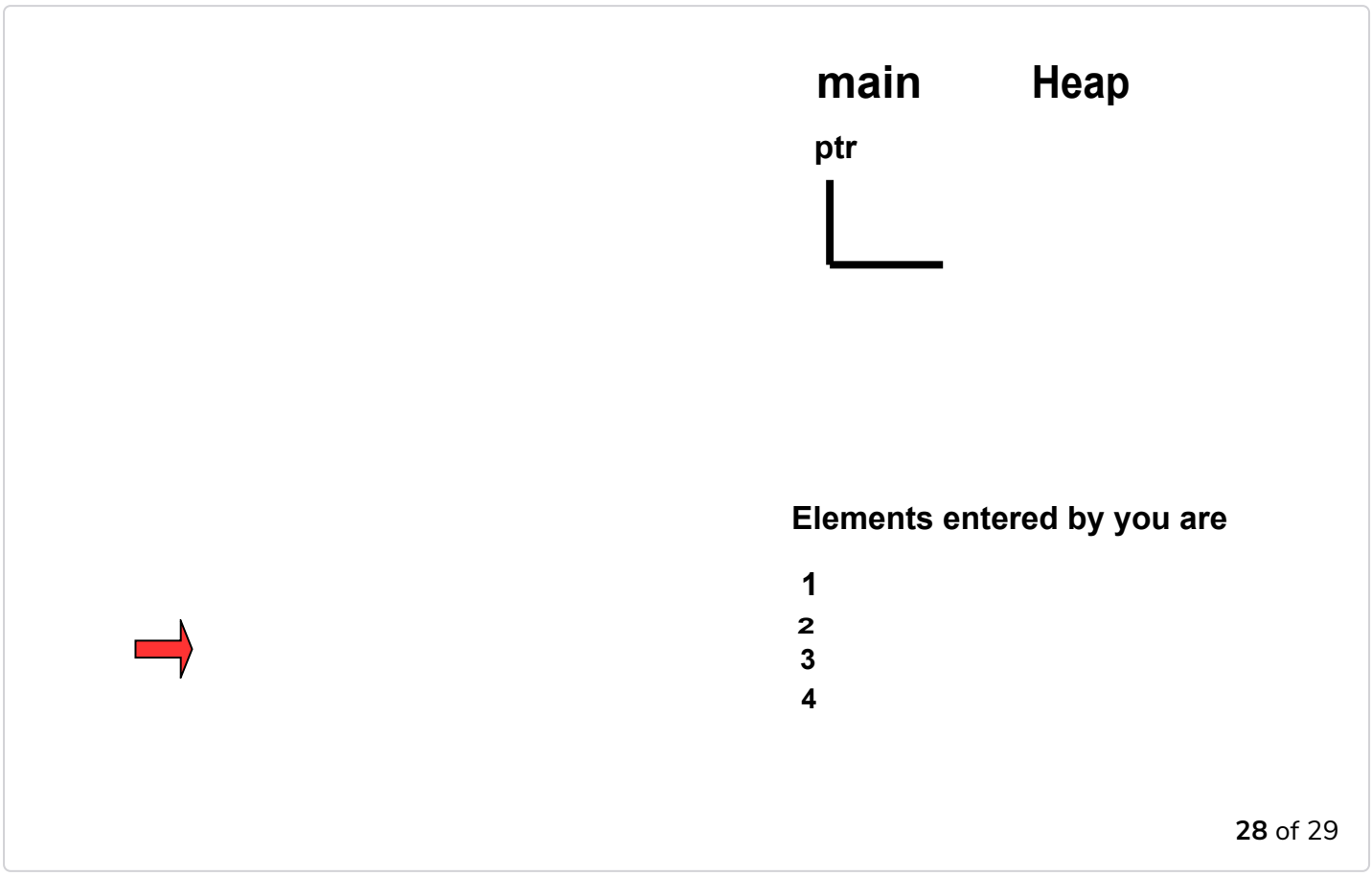
main Heap

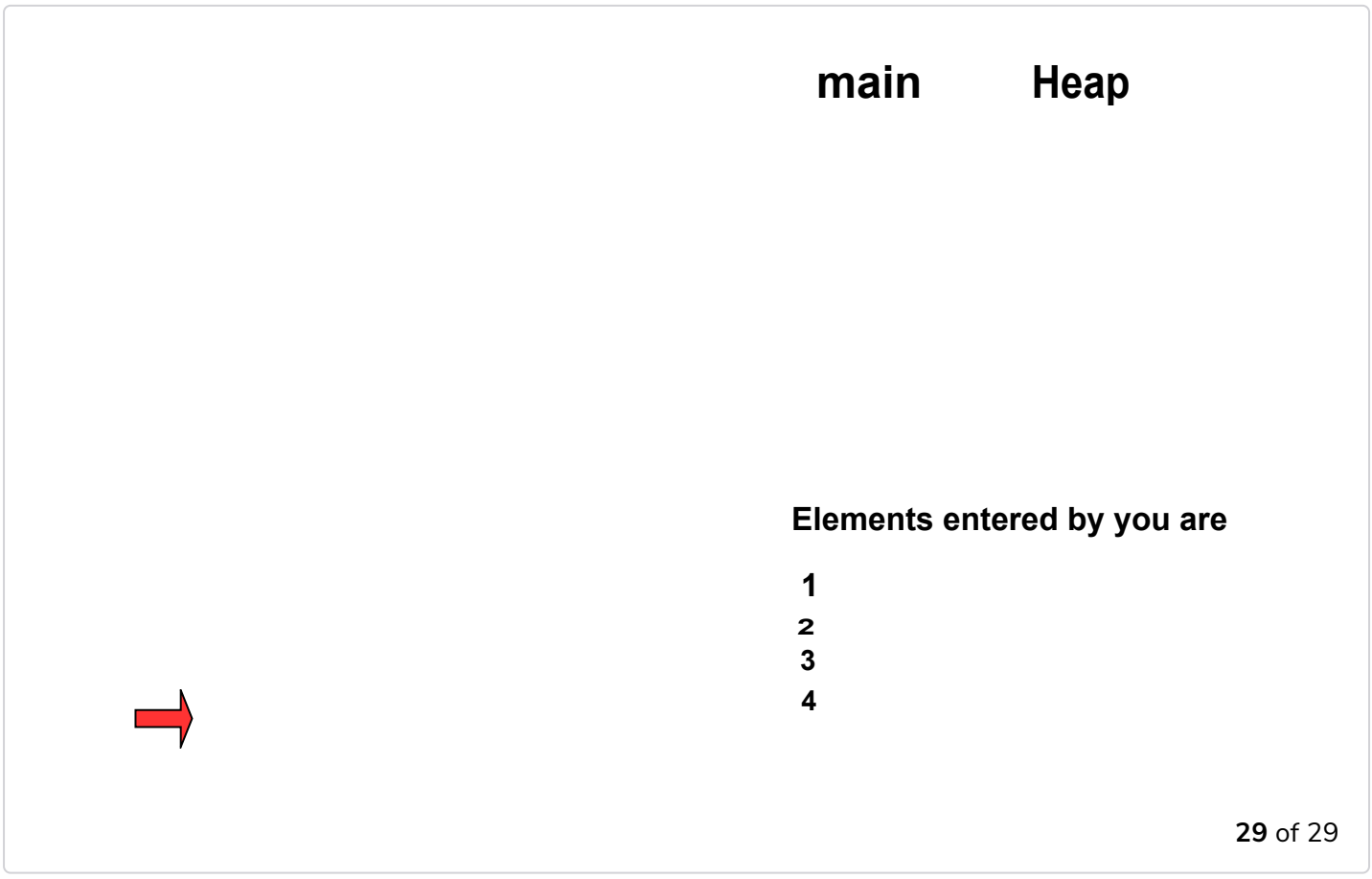
ptr



Elements entered by you are

1
2
3
4





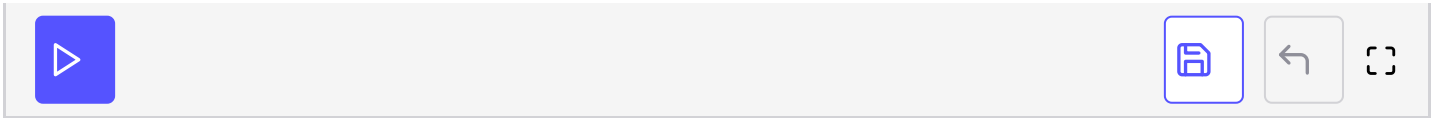
Referencing variables

The `&` operator is used to *reference* an object. When using this *operator* on an object, you are provided with a *pointer* to that object. This new *pointer* can be used as a *parameter* or be *assigned* to a variable.

Let's take a look at an example below:

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

int main() {
  int num;
  int *ptr;    //declaring int type pointer
  ptr = &num;    //setting pointer equal to the address of num
  cout << "Address stored in ptr is: "<< ptr<<endl;
  *ptr = 7;    //setting value of *ptr to 7
  cout << "value of num is: "<< num <<endl;
  return 0;
}
```



Example Explanation

- First, we declare a *pointer* type `ptr`.
- Then we store the address of `num` into the pointer `ptr` as shown in line 7.
- We use the `&` operator to store the *address* of `num` in line 7.
- Then we set the value of `ptr` equal to 7 using the `*` operator as seen in line **9**.
- This value of 7 is stored at the *address* where the *pointer* `ptr` is pointing to (in this case address of `num`).
- Hence, in the output the value of `num` is displayed as 7.

Pointers to Arrays

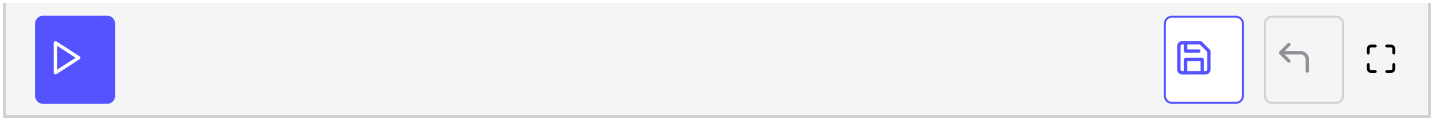
C++ allows us to create *arrays* and then use *pointers* to carry out operations on those *arrays*.

For example, let's step through the example below and take a look at how we can do this.

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

int main() {
  //first we declare an array
  int arr[4];
  //next we declare a pointer
```

```
  //next we declare a pointer
  int *ptr;
  //now we make the pointer ptr point to the first element of the array arr
  ptr = arr;
  //next we set the value of the first element of arr, that is, arr[0] equal to 3
  *ptr = 3;
  //now we increment the pointer ptr to point to second element of the array arr
  ptr++;
  //next we update the value of the second element of arr, that is, arr[1] being pointed at by ptr
  *ptr = 5;
  //to directly store a value at some index in arr, e.g at the 3rd index of array we first get the
  ptr = &arr[3];
  //now storing a value at arr[3] location
  *ptr = 10;
  //moving pointer back to arr[0]
  ptr = arr;
  //storing value at arr[2] now
  *(ptr+2) = 8;
  //now lets display all the values we stored in our array
  for (int i=0; i<4; i++){
    cout<< "value at arr["<<i<<"] is: "<<arr[i]<<endl;
  }
  return 0;
}
```



Going interesting so far? There are more intriguing concepts such as the *multi-dimensional* arrays that we'll discuss in the next lesson.