Further Properties of Arrays & Pointers

This lesson explains the concept of multidimensional arrays and pointers pointing to other pointers

We'll cover the following

- Multi-dimensional arrays
- Pointer to a pointer
- Linked List

Multi-dimensional arrays

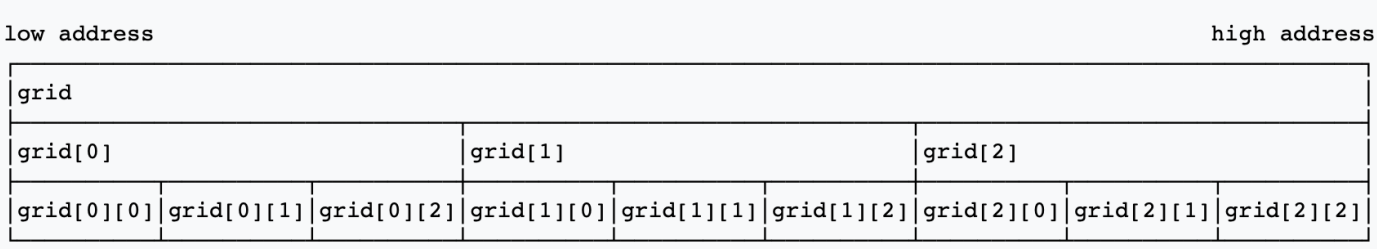
A *multidimensional* array allows **nesting** arrays:

```
int grid[3][3];
```

This allocates 3*3 elements in **one** memory block.

> **Note:** Even though **arrays** behave similarly to *pointers*, a *multidimensional array* is not a **pointer-to-a-pointer**.

Here is a visual representation of the *multi-dimensional* array `grid[3][3]`:



- The objects `grid`, `grid[0]` and `grid[0][0]` are always at the **same** location (but different types).
- The objects of *variable* `pptr` and *pointer* `*pptr`, `**pptr` are at *different* locations.

When evaluating `grid[0][0]`

- The array *grid* (which is an `int[3][3]`) is first converted to a pointer of type `int( * )[3]`.
- Then taking the element at **offset 0** yields an object of `int[3]`.
- Then it is converted to `int*` again and the element at **offset 0** is taken, generating an object of type `int&`.

Down below is an example code illustrating how to work with *multi-dimensional* arrays.

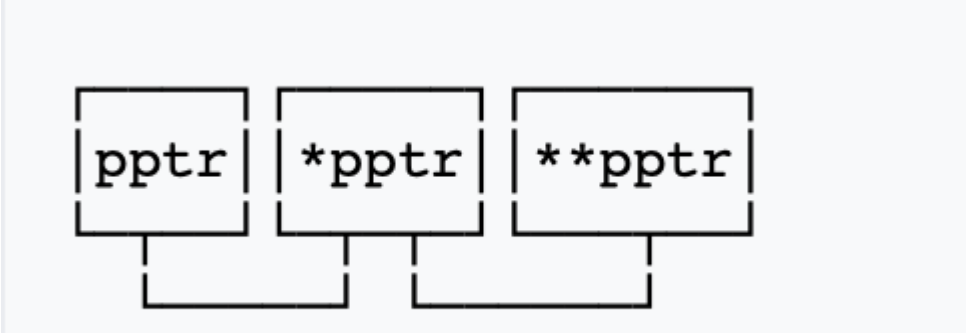
```
#include <iostream>
using namespace std;

int main () {
   // an array with 3 rows and 3 columns.
   int grid[3][3];

   // setting value of each array element
   for ( int i = 0; i < 3; i++ ){
      for ( int j = 0; j < 3; j++ ) {
        grid[i][j] = i+j;
        cout << "grid[" << i << "][" << j << "]: " <<grid[i][j]<< endl;
      }
   }

   return 0;
}
```

Pointer to a pointer

A *pointer* contains a *reference* to another variable. It may also point to a *pointer*:

```
int **pptr;
```

Down below is an *illustration* demonstrating the concept.



For `pptr[0][0]`, the *address* stored in `pptr` is taken and the *address* stored in that *address* is taken, and it is the result of the *expression*.

Take a look at the *example* below to understand this concept better:

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

int main() {

  int x=1;
  int *ptr1;
  int **ptr2;

  ptr1 = &x;    //getting address of x
  ptr2 = &ptr1;  //getting address of ptr1

  cout << "Value of x is: "<<x<<endl;

  //let's print the value being pointed to by ptr1
  cout << "The value being pointed to by ptr1 is: "<<*ptr1<<endl;

  //let's print the address being pointed to by ptr2
  cout << "The address being pointed to by ptr2 is: "<<*ptr2<<endl;

  //let's print the value being pointed by ptr2
  cout << "The value being pointed to by ptr2 is: "<<**ptr2<<endl;

  return 0;
}
```

- In the example above `*ptr2` will give the value of the address at which our number is stored, this will also be the address of `ptr1`.
- `**ptr2` then further *dereferences* and gives the value stored at that *address* which is **1**.

Linked List

This allows for the implementation of a linked list:

```
class LinkedListOfIntsNode
{
    int value;
    LinkedListOfIntsNode *next_node;
};
```

Think of a chain of **ten** `LinkedListOfIntsNode`, each *pointing* to its neighbor to the **right**. You can traverse the *list* using `next_node`.

In the next lesson, we will discuss how to pass *pointers* to *functions*.