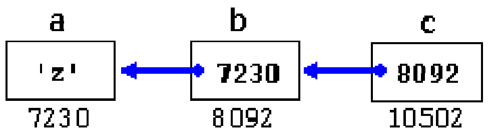
Pointers to pointers

C++ allows the use of pointers that point to pointers, that these, in its turn, point to data (or even to other pointers). In order to do that, we only need to add an asterisk (`*`) for each level of reference in their declarations:

```
char a;
char * b;
char ** c;
a = 'z';
b = &a;
c = &b;
```

This, supposing the randomly chosen memory locations for each variable of `7230`, `8092` and `10502`, could be represented as:



The value of each variable is written inside each cell; under the cells are their respective addresses in memory.

The new thing in this example is variable `c`, which can be used in three different levels of indirection, each one of them would correspond to a different value:

- `c` has type `char**` and a value of `8092`
- `*c` has type `char*` and a value of `7230`
- `**c` has type `char` and a value of `'z'`

void pointers

The `void` type of pointer is a special type of pointer. In C++, `void` represents the absence of type, so void pointers are pointers that point to a value that has no type (and thus also an undetermined length and undetermined dereference properties).

This allows void pointers to point to any data type, from an integer value or a float to a string of characters. But in exchange they have a great limitation: the data pointed by them cannot be directly dereferenced (which is logical, since we have no type to dereference to), and for that reason we will always have to cast the address in the void pointer to some other pointer type that points to a concrete data type before dereferencing it.

One of its uses may be to pass generic parameters to a function:

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

void increase (void* data, int psize)
{
  if ( psize == sizeof(char) )
  { char* pchar; pchar=(char*)data; ++(*pchar); }
  else if (psize == sizeof(int) )
  { int* pint; pint=(int*)data; ++(*pint); }
}

int main ()
{
  char a = 'x';
  int b = 1602;
  increase (&a,sizeof(a));
  increase (&b,sizeof(b));
  cout << a << ", " << b << endl;
  return 0;
}
```

```
y, 1603
```

`sizeof` is an operator integrated in the C++ language that returns the size in bytes of its parameter. For non-dynamic data types this value is a constant. Therefore, for example, `sizeof(char)` is `1`, because `char` type is one byte long.

Null pointer

A null pointer is a regular pointer of any pointer type which has a special value that indicates that it is not pointing to any valid reference or memory address. This value is the result of type-casting the integer value zero to any pointer type.

```
int * p;
p = 0;     // p has a null pointer value
```

Do not confuse null pointers with void pointers. A null pointer is a value that any pointer may take to represent that it is pointing to "nowhere", while a void pointer is a special type of pointer that can point to somewhere without a specific type. One refers to the value stored in the pointer itself and the other to the type of data it points to.

Pointers to functions

C++ allows operations with pointers to functions. The typical use of this is for passing a function as an argument to another function, since these cannot be passed dereferenced. In order to declare a pointer to a function we have to declare it like the prototype of the function except that the name of the function is enclosed between parentheses `()` and an asterisk (`*`) is inserted before the name:

```
// pointer to functions
#include <iostream>
using namespace std;

int addition (int a, int b)
{ return (a+b); }

int subtraction (int a, int b)
{ return (a-b); }

int operation (int x, int y, int
(*functocall)(int,int))
{
  int g;
  g = (*functocall)(x,y);
  return (g);
}

int main ()
{
  int m,n;
  int (*minus)(int,int) = subtraction;

  m = operation (7, 5, addition);
  n = operation (20, m, minus);
  cout <<n;
  return 0;
}
```

```
8
```

In the example, `minus` is a pointer to a function that has two parameters of type `int`. It is immediately assigned to point to the function `subtraction`, all in a single line:

```
int (* minus)(int,int) = subtraction;
```