**Dereference operator (\*).**

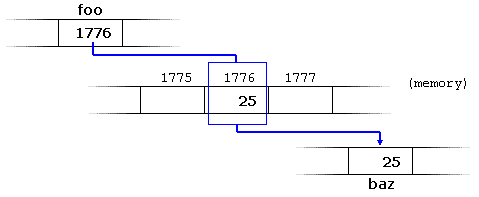
As just seen, a variable which stores the address of another variable is called a *pointer*. Pointers are said to "point to" the variable whose address they store.

An interesting property of pointers is that they can be used to access the variable they point to directly. This is done by preceding the pointer name with the *dereference operator* (\*). The operator itself can be read as "value pointed to by".

Therefore, following with the values of the previous example, the following statement:

|  |  |  |
| --- | --- | --- |
|  | baz = \*foo; |  |

This could be read as: "*baz* equal to value pointed to by *foo*", and the statement would actually assign the value 25 to *baz*, since *foo* is 1776, and the value pointed to by 1776 (following the example above) would be 25.



It is important to clearly differentiate that *foo* refers to the value 1776, while *\*foo* (with an asterisk \* preceding the identifier) refers to the value stored at address 1776, which in this case is 25. Notice the difference of including or not including the *dereference operator* (I have added an explanatory comment of how each of these two expressions could be read):

|  |  |  |
| --- | --- | --- |
| 1 2 | baz = foo; // baz equal to foo (1776)  baz = \*foo; // baz equal to value pointed to by foo (25) |  |

The reference and dereference operators are thus complementary:

* & is the *reference operator*, and can be read simply as "address of"
* \* is the *dereference operator*, and can be read as "value pointed to by"

Thus, they have sort of opposite meanings: An address obtained with & can be dereferenced with \*.

Earlier, we performed the following two assignment operations:

|  |  |  |
| --- | --- | --- |
| 1 2 | myvar = 25;  foo = &myvar; |  |

Right after these two statements, all of the following expressions would give true as result:

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 | myvar == 25  &myvar == 1776  foo == 1776  \*foo == 25 |  |

The first expression is quite clear, considering that the assignment operation performed on *myvar* was *myvar*=25. The second one uses the address-of operator (&), which returns the address of *myvar*, which we assumed it to have a value of 1776. The third one is somewhat obvious, since the second expression was true and the assignment operation performed on *foo* was *foo=&myvar*. The fourth expression uses the *dereference operator* (\*) that can be read as "value pointed to by", and the value pointed to by *foo* is indeed 25.

So, after all that, you may also infer that for as long as the address pointed to by *foo* remains unchanged, the following expression will also be true:

|  |  |
| --- | --- |
|  | \*foo == myvar |

Aforementioned operations:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12  13  14  15  16  17 | #include "iostream"  using namespace std;  int main()  {  auto myvar = 25;  auto foo = &myvar;  decltype(myvar) bar = myvar;  auto baz = \*foo;  decltype(foo) var = foo;  cout << myvar << endl;  cout << foo << endl;  cout << bar << endl;  cout << baz << endl;  cout << var << endl;  cout << \*foo << endl;  return 0;  } | 25  004FFE18  25  25  004FFE18  25 | [Edit & Run](https://www32.cplusplus.com/doc/tutorial/ntcs/) |

Also pay attention to the below codes:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12  13  14  15  16  17  18 | #include "iostream"  using namespace std;  int main()  {  auto myvar = 25;  auto foo = &myvar;  decltype(myvar) bar = myvar;  \*foo = 10;  auto baz = \*foo;  decltype(foo) var = foo;  cout << myvar << endl;  cout << foo << endl;  cout << bar << endl;  cout << baz << endl;  cout << var << endl;  cout << \*foo << endl;  return 0;  } | 10  0078FBAC  25  10  0078FBAC  10 | [Edit & Run](https://www32.cplusplus.com/doc/tutorial/ntcs/) |

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12  13  14  15  16  17  18 | #include "iostream"  using namespace std;  int main()  {  auto myvar = 25;  auto foo = &myvar;  decltype(myvar) bar = myvar;  auto baz = \*foo;  \*foo = 10;  decltype(foo) var = foo;  cout << myvar << endl;  cout << foo << endl;  cout << bar << endl;  cout << baz << endl;  cout << var << endl;  cout << \*foo << endl;  return 0;  } | 10  0097FA84  25  25  0097FA84  10 | [Edit & Run](https://www32.cplusplus.com/doc/tutorial/ntcs/) |