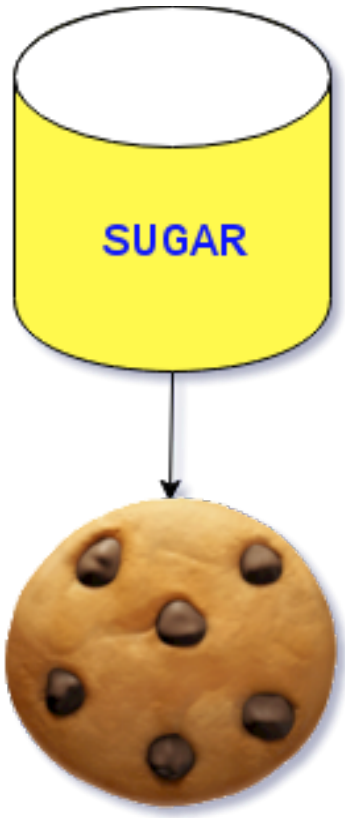
Syntactic Sugar



Syntactic Sugar is a computer science term used to denote when an expression has been made easier to convey and read versus its counterpart. This leads to a "tastier" programming language. Two of the most major forms of syntactic sugar are when dealing with arrays and dynamically allocated classes/data structures.

Arrays decay into a pointer to the first element. The standard syntax for an array is **array[4]**. Have you thought about what the **[]** operator does for us? This is a more aesthetically pleasing variant of ***(array + 4)**. Here, what is happening is pointer arithmetic and dereferencing all in one. If you were to output these statements, the content in the fifth element of the array would be displayed. You can read this as dereferencing the fifth element from the beginning of the array. Since addition is commutative, **4[array]** is valid syntax. Without the sugar, we would have to write ***(4 + array)**, which is the same as above, just switched around. If you output ***array** this is equivalent to dereferencing the beginning of the array or **array[0]**. Taking a look at two-dimensional arrays, the same form applies, except you dereference twice to obtain the value you desire. Dereferencing once will land you a pointer (address), because a 2D-array is an array of pointers each pointing to an array of a type (**int**, **char**, etc). One example is ***(*(array+4)+1)**. The equivalent syntactic sugar is **array[4][1]**.

Another variant of syntactic sugar is the **->** operator. This is used during dynamic allocation of classes and data structures. For example, if you want to make a pointer to the class you just made, you would allocate memory such as **myClass *Object = new myClass();** As you should know, a pointer must be dereferenced before using the contents it points to. The dot operator is used when accessing class member functions via a class object such as **Object.classFunction();** However, **Object** is a pointer and therefore any call to a member function should look like ***(Object).classFunction();** This is not pleasing to the eye. Luckily the **->** operator is here to make our experience sweeter. Now, you can conveniently omit the dereferencing and dot operator to make a visual-appeasing call to a member function. **Object->classFunction();** This can be thought of as "**Object** is pointing to the member function it wants to use." Data structures utilize the same syntax, for example **list<int> *myList = new list<int>(10); cout << myList->size();**