Comparing Programming Languages: C# vs Object Pascal

Sam Huffer, 19 October 2017

Programming is multilingual

When communicating with other people, there are many languages one might use. You could choose to communicate in English, or Spanish, or Russian, or even a language that no one really uses any more if you wanted to be really creative. Languages are continually evolving, and in a few centuries, people might be speaking a language that does not even exist yet.

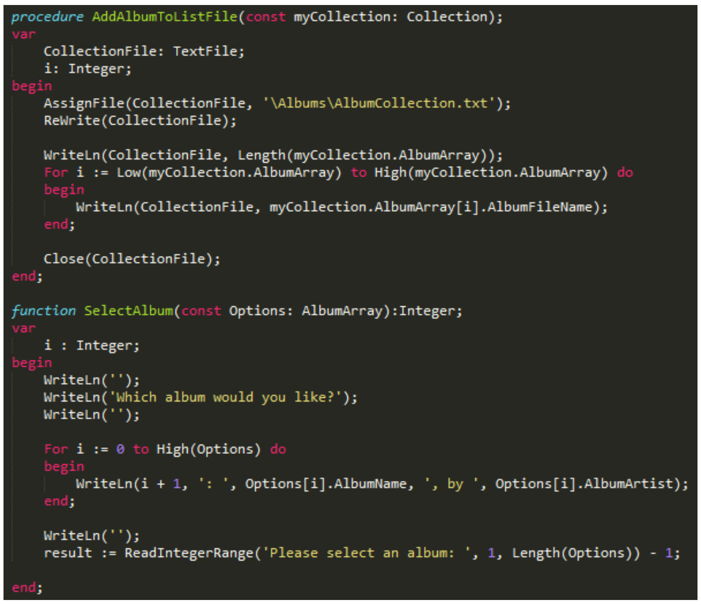
Programming is very similar. Like spoken languages, there are different types of programming languages. There are procedural languages where programs are basically just a sequence of procedures and functions that can do stuff for you, and variables that hold all of your data. There are also object-oriented languages that allow programmers to define and employ objects that encapsulate data and functionality into a single entity. Some languages and variants of languages are somewhere in between. New languages are constantly being created and updated, while others fall out of use as they are surpassed.

In this essay, I will be discussing and comparing two languages: C#, an object-oriented programming language; and Object Pascal, an extension of the procedural programming language Pascal that provides it with object-oriented functionality. I will discuss their similarities, in some instances contrasting with third languages, and then I shall explore their differences.

Apples and carrots are plants

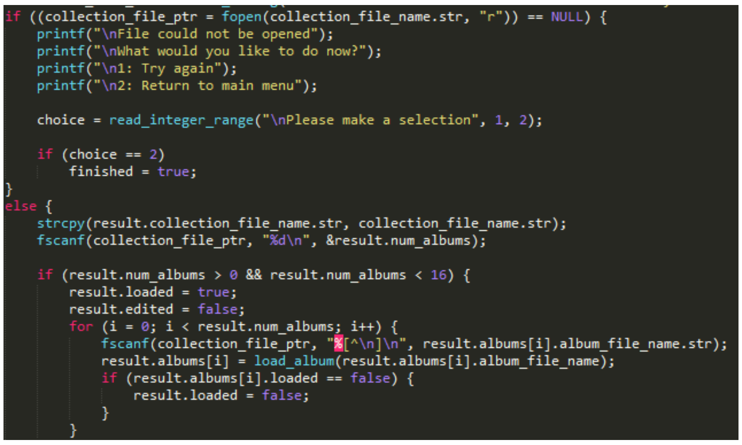
Like apples and carrots, C# and Pascal have a number of similarities. Both are programming languages that allow the use of objects and object-oriented programming; C# was designed as an OOP language, whereas Pascal affords OO functionality through its Object Pascal variants, including Turbo Pascal, Free Pascal, and Delphi.

Both C# and Pascal are also easy languages to read (see figure 1). Many of their keywords and inbuilt functionality are more or less presented to users in regular English, or in a format easily expressed verbally in English, with sensible and understandable terminology. This commonality can be contrasted with other languages that are



```
procedure AddAlbumToListFile(const myCollection: Collection);
var
    CollectionFile: TextFile;
    i: Integer;
begin
    AssignFile(CollectionFile, '\Albums\AlbumCollection.txt');
    ReWrite(CollectionFile);

    WriteLn(CollectionFile, Length(myCollection.AlbumArray));
    For i := Low(myCollection.AlbumArray) to High(myCollection.AlbumArray) do
    begin
        WriteLn(CollectionFile, myCollection.AlbumArray[i].AlbumFileName);
    end;

    Close(CollectionFile);
end;

function SelectAlbum(const Options: AlbumArray):Integer;
var
    i : Integer;
begin
    WriteLn('');
    WriteLn('Which album would you like?');
    WriteLn('');

    For i := 0 to High(Options) do
    begin
        WriteLn(i + 1, ': ', Options[i].AlbumName, ', by ', Options[i].AlbumArtist);
    end;

    WriteLn('');
    result := ReadIntegerRange('Please select an album: ', 1, Length(Options)) - 1;

end;
```

Figure 1: An excerpt of code in Pascal, illustrating it's easy to read nature.



```
if ((collection_file_ptr = fopen(collection_file_name.str, "r")) == NULL) {
    printf("\nFile could not be opened");
    printf("\nWhat would you like to do now?");
    printf("\n1: Try again");
    printf("\n2: Return to main menu");

    choice = read_integer_range("\nPlease make a selection", 1, 2);

    if (choice == 2)
        finished = true;
}
else {
    strcpy(result.collection_file_name.str, collection_file_name.str);
    fscanf(collection_file_ptr, "%d\n", &result.num_albums);

    if (result.num_albums > 0 && result.num_albums < 16) {
        result.loaded = true;
        result.edited = false;
        for (i = 0; i < result.num_albums; i++) {
            fscanf(collection_file_ptr, "%[^\n]\n", result.albums[i].album_file_name.str);
            result.albums[i] = load_album(result.albums[i].album_file_name);
            if (result.albums[i].loaded == false) {
                result.loaded = false;
            }
        }
```

Figure 2: An excerpt of code in C, illustrating it's relatively clunky and ugly appearance when compared to other languages.

comparatively ugly and harder to understand, such as C, with its less attractive naming conventions, less intuitive names for inbuilt procedures and functions (e.g. fscanf), and incomprehensible parameters (e.g. the weird character combinations in the middle of fscanf's parameters; see figure 2). While those aspects might be more understandable to programmers more experienced with C, they are not conducive to learning C and raise its barrier to entry, as opposed to languages like Pascal and C# which are much plainer and easier to learn.

Another feature shared by C# and Pascal that lowers their complexity and makes them easier to learn is that there is less onus on the user to allocate and reallocate memory. For example, Pascal users can employ the SetLength function to change the length of dynamic arrays as necessary, and C# users can store variables in lists and use the list object's inbuilt Add and Remove methods to manage the list's contents. As such, users are not forced to manually manage memory themselves if they don't want to have to worry about it, as opposed to languages like C (this is my last tirade about C being awful, I promise) where you might have malloc (i.e. memory allocate) and realloc (i.e. re-allocate memory) functions to manipulate variables in memory, but would then still have to manually check that everything went ok, and that you're not accidentally going to read or damage data from another program. The fact that Pascal and C# hide the internal workings of memory allocation from you and give you generic functions that can handle the specifics for you make both conducive to learning programming.

Carrots don't go in a fruit salad

While C# and Pascal have similarities that make them conducive to learning and ease of comprehension, they also have differences. The most far-reaching difference is the type of language that each is: C# was designed as an object-oriented programming language, whereas Pascal was designed as a procedural programming language, with object-oriented functionality provided by variants of Object Pascal. While both allow users to define classes and create objects, the fact that they are different types of languages results in a difference in the ease with which this can be done, and the ease of comprehension that each language affords when doing so.

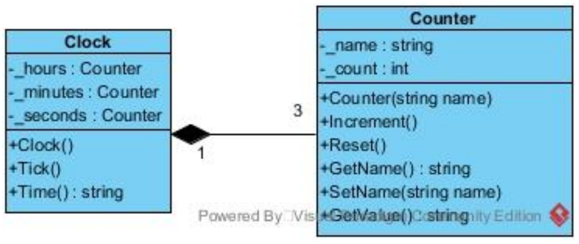
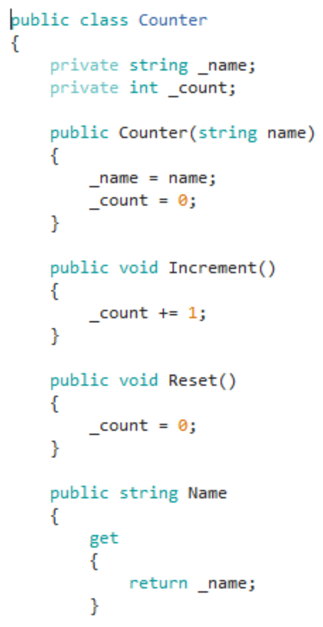


Figure 3: A UML Class diagram for a clock program

Perhaps the most obvious indicator of this is the structure of code produced in each language when defining and implementing objects. Since C# was designed to use objects right from the start, it's file layout for classes inherently aids comprehension. For example, in figure 4, which illustrates how C# classes are defined, you can see how the class declaration at the top of the page encapsulates all of the fields and methods that belong to the class within an opening brace and a closing brace (the



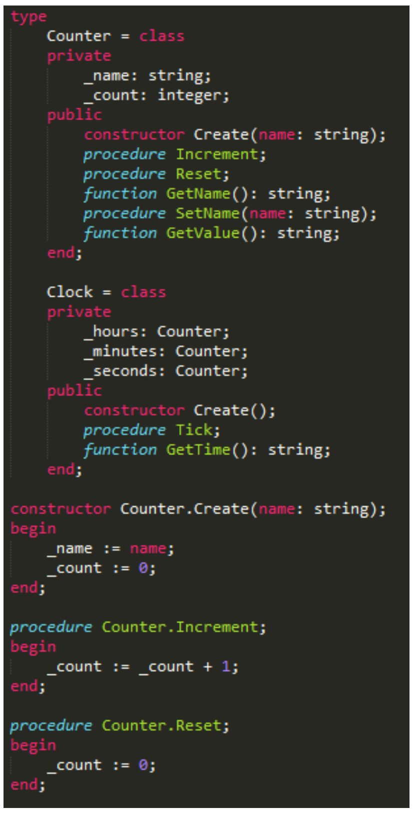
```csharp
public class Counter
{
    private string _name;
    private int _count;

    public Counter(string name)
    {
        _name = name;
        _count = 0;
    }

    public void Increment()
    {
        _count += 1;
    }

    public void Reset()
    {
        _count = 0;
    }

    public string Name
    {
        get
        {
            return _name;
        }
```

Figure 4: C# code for a clock program

latter omitted from the image) in much the same way that the methods encapsulate all of the lines of code that comprise them within braces. As such, much as the braces clearly illustrate to programmers that a section of code belongs to a method, the braces that encapsulate the class's fields and methods clearly convey that those fields and methods belong to said class, resulting in a representation that is fairly close in appearance to that of a UML class diagram (such as the one in figure 3).

In contrast to this, the organisation of Object Pascal's objects and the methods belonging to those objects is not nearly as orderly. Since Pascal was designed as a procedural programming language that only needed to worry about procedures, functions, and records encapsulating data and functionality separately, the organisation of classes would not have been an important concern. So, when its Object Pascal extensions and variants were added, they were not well equipped to convey that a class encapsulated particular fields and methods in exactly the same way as dedicated object-oriented languages like C#. As you can see in figure 5, programs written in Object Pascal that involve the use of objects require programmers to outline the fields and methods that comprise those objects in a manner almost identical to how one would outline what the contents of a record would be.

However, these object definitions, like classes in a UML diagram, only list what fields and methods comprise the object; they do not provide the details of the internal workings of the methods themselves. To provide these details, each method belonging to an object needs to be declared the same ways as the program's non-object-related procedures and functions, tagged as belonging to a specific class of objects, and then filled in like any other procedure/function. Because Object Pascal does not visually encapsulate methods within classes as clearly as C#, it requires programmers to pay more attention to which class a method is tagged as belonging to, as they are otherwise indistinguishable from methods belonging to a completely different class and procedures and functions not belonging to any class at all. Furthermore, this lack of clear, visual encapsulation also affords the possibility that programmers with poor skills in laying out sections of code within a program might not lay out methods from the same class consecutively. Instead, they could very well list a method belonging to one class, then another method from a different class, then another method from the first class, and then a method from a third class, then the first class again, putting no thought into the organisation of their methods, and then asking an object to use a method that they mistakenly thought belonged to it. To summarise, it's just too easy in Object Pascal to make a confusing mess of your objects and their methods.



```pascal
type
    Counter = class
    private
        _name: string;
        _count: integer;
    public
        constructor Create(name: string);
        procedure Increment;
        procedure Reset;
        function GetName(): string;
        procedure SetName(name: string);
        function GetValue(): string;
    end;

    Clock = class
    private
        _hours: Counter;
        _minutes: Counter;
        _seconds: Counter;
    public
        constructor Create();
        procedure Tick;
        function GetTime(): string;
    end;

constructor Counter.Create(name: string);
begin
    _name := name;
    _count := 0;
end;

procedure Counter.Increment;
begin
    _count := _count + 1;
end;

procedure Counter.Reset;
begin
    _count := 0;
end;
```

Figure 5: Object Pascal code for a clock program

Of course, it is entirely possible to mitigate this problem using multiple files. In my own experience, it is common when using C# to implement the code for each class in its own file. Such a strategy could feasibly be employed when programming in Object Pascal to mimic C#'s inherent in-code encapsulation of the contents of objects. However, it should also be noted that even with such an approach, Object Pascal would still require users to explicitly list all the methods contained within a class, rather than just needing to declare the method like any other function to have it included within the class. Another drawback of this approach would also force users to use more includes statements at the start of each file than would otherwise be necessary. Ultimately, C# is much more suited for easy-to-read and well organised object-oriented programming than Object Pascal.

In the End

You start with one language when learning programming, and then another, and another. For object-oriented programming, C# is a good place to start. It's easy to read, and its layout emphasises how objects encapsulate data and functionality into a single entity. Though technically possible, you wouldn't use it as an introduction to procedural programming; Pascal is better suited to that. Conversely, if you were learning object-oriented programming, Object Pascal (or at least the variant used in this essay's clock example) would not be a good place to start, as it has a higher barrier-to-entry in terms of comprehension complexity, and it would be too easy for the inexperienced programmer to confuse themselves with poor organising. Both have their pros and cons, and different cases where one would be more suitable for a task than the other. Regardless of their differences, one key thing to remember is that, in terms of clarity, ease of use, and suitability for learning (yes, I said I wouldn't mention it again, and no, I'm not sorry), they are both far better than C.