Choosing the Right Language for the Job

Sheldon Linker
sol@linker.com
800-315-1174 • www.linkersystems.com

Choosing the Right Language for the Job

…in which everything is listed alphabetically to avoid bias.

Is there a best language overall?

Each computer language has its proponents, some of whom claim that their computer language is the only one worth using. There are generally three reasons why someone will propose that one language is superior to all others. These are:

- The proponent doesn't really know any other computer languages, or has only learned and used one, and find the others difficult.
- There is some ego investment. For instance, the proponent may have designed the language.
- The profit motive: If a particular company has a financial stake in which language you use, you can bet your bottom dollar that they will propose that you go their way.

However, each computer language was developed with the intent of solving some sort of problem or class of problems; and each computer language is very well suited for solving the problem it was intended to solve, and often poorly suited to solving problems vastly different from its design criteria. In the sections below, we will go through various computer languages, and try to give some guidance in how to proceed.

Academia Versus the Real World

Very often, proponents of a certain theory of computing, cognition, or some similar field develops a language or class of languages. When academics design a language, they strongly imbue the language with facilities based on their theories or principals. These languages are almost never tested against real-world usage patterns or constraints.

Companies Versus the Real World

Or, perhaps I should have said "Companies Driving the Real World". Lately, some companies have taken to creating new languages to use against their competition, or drive their own sales, or both, without concern as to whether the language makes sense to its intended audience. Microsoft is a prime example of this. See the description of C# and related languages for more on this topic.

Companies Driven by the Real World

The best example of this is Sun and Java. Rather than abstract theories, or a pure and direct profit motive, Sun was answering the concerns of its customers, potential customers, and users in general when it decided to create a language to solve these problems. See the descriptions of Java and related languages for more information on this topic.

Object-Oriented Versus Older Languages

It seems that currently, there are two major classes of languages and programming style, object-oriented and non-object-oriented languages. First, let me describe object orientation extremely briefly.

An object-oriented program in one in which data-groupings — call them records, structures, objects, or whatever — are treated like objects, in that they have properties and actions. For instance, a property might be its screen position or account balance, and an action might be "display yourself" or "save yourself to disk".

A non-object-oriented program, so-called because they're older than object-oriented programs, and thus did not receive a type-name, are programs in which some procedure not directly connected with the data groupings handle the data.

Proponents of object-oriented programming will tell you that object-oriented programming can only occur in an object-oriented language, and that code is much more easily written in an object-oriented form, and thus all code should be object oriented.

On the other hand, proponents of pure procedural code will tell you that pure procedures are more easily written in pure procedural languages, and that everything is much more straightforward in a pure procedural language, and thus all code should be purely procedural.

Both of these groups are very shortsighted. Within any one program, some things are more easily done procedurally, and some things are more easily done using object-orientation. When you're designing your programs, take the time to figure out which is which. For instance, popping up a general alert message stating that you have not saved your work for half an hour with "Save" and "Wait" buttons is a nice feature. However, it does not need to be written as an object class. It's something more easily done by calling a nice, old-fashioned subroutine. However, when the subroutine wants to save the current work. And if there is more than one kind of possible data grouping to save, telling the object to save itself (object-oriented) is easier than figuring out which kind of save routine to call (procedurally).

Also, keep in mind that by carefully crafting your data structures, you can write object-oriented code in a non-object-oriented language. Similarly, you are still allowed to call normal subroutines in most object-oriented languages.

The morals are:

- Given the choice between an older, non-object-oriented language and a newer object-oriented version of the same language, go with the object-oriented language. (Unless there are execution-speed problems.)
- Evaluate for yourself whether each item in the program should be object-oriented or not. Don't worry about abstract theories or rules.

And on a similar note…

"Go To" Statements

Computers have GO TO instructions down at their lowest levels. Thus, it was inevitable that programming languages would have GO TO statements. Later, some academics and others decided that a GO TO in a program was inherently evil, or at least detrimental, and that new languages should be designed without GO TO statements, thus preventing programmers from committing this evil.

As with all things, GO TOs are sometimes a good thing, and sometimes bad. You just have to figure out for yourself which case you're in, unless, of course, your language of choice precludes it.

The Languages Themselves

Assembly

History: When people started programming computers, they did so in the binary or decimal code native to the computer. (Yes, decimal. The IBM 1620 and others used actual decimal storage locations that could hold numbers between 0 and 9, rather than the modern byte holding 0 to 255.) Assembly language allowed people to program computers using the names of instructions, rather than the numeric values.

Portability: None whatsoever.

Writing, reading, and project management concerns: Assembly is extremely time-consuming to write, and tedious to read. Nobody will ever be able to understand your code unless it is fully commented, module by module and line by line.

Plusses, minuses, and indications: In writing operating systems, there are some pieces of code that must be written in assembly, purely because compilers can't generate the needed code. In other cases, there is a routine, on occasion, which constitutes a major bottleneck, and the programmer can write better assembly code than the compiler can. In these rare instances, use assembly. For instance, the 1.0 version of Animation Stand was 30,000 lines of C code, and 22 lines of assembly code.

C, C++, Objective C, APIs, Foundation classes, and Lint

History: When AT&T (and specifically Kernigan and Ritchie) developed the Unix operating system, they developed a language to go with it. They had a very simple language that was to have almost all the control of Assembly language, and was ideally suited to the DEC PDP-11 instruction set: The A language (No, it was not designed as the universal language.) Later revisions were called the B and C languages, respectively. Unlike other languages around at the time, the C language family made no assumptions about the capabilities of the operating system, and provided no OS features directly. Instead, it had a subroutine package called "stdio" (for "StanDard Input/Output") that handled those functionalities. Such subroutine packages care called "APIs" (for "Application Program Interfaces"). Later, the C language was made available on other computers, and then standardized to ANSI C. When the object-oriented craze struck (having started primarily in Ada and SmallTalk), a new version of C was crated to add direct object features. Although one might have guesses the new language would have been called "D", the name chosen was C++, also known as "C Plus Plus". Some C/C++ programs run as "silent" or "console mode" applications that make no graphical interface calls or other system-dependent calls. These C programs use the "standard" API. Other C programs that make use of the local graphics systems will us the Macintosh, Windows, X, or other APIs, and are not portable. Often, C++ programs will use local "foundation classes", meant to simplify the local API. Another version of object-oriented C is Objective C. Objective C is a superset (to a great extent) of C++, allowing additional interesting forms of message passing, calls into classes not yet instantiated, and other naming and calling conventions. Lastly, there is Lint, which is a debugging tool. Lint "compiles" C/C++ programs, and "links" them, but does not produce a final program. Instead, it produces a report showing possible failure points for the program. Relatively recently, many C/C++/Objective C compilers have directly added man Lint features into the compiler.

Cobol

History: Cobol is a language that followed directly from Assembly. It was the second compiled language. (Fortran was the first). The idea was to have a language that meat the needs of the business applications community as Fortran met the needs of the scientific programming community. The idea was to create a language that anyone could read and write. Thus, it looks a lot like English. COBOL was based on work by Dr. Grace Murray Hopper (who later became one of the first woman Rear Admirals). In 1959, the Conference on Data System Languages (CODASYL) was formed to create the first industry standard business programming language. Sponsored by the U.S. Department of Defense, the creation of COBOL (Common Business Oriented Language) was supported by computer manufactures, large users and several universities. IBM later was the first manufacturer to build a computer designed specifically to run COBOL, the IBM 360.

Portability: Mainframes, PCs, and Unix systems have COBOL. There may be a COBOL for the Macintosh, but I have not seen it.

Writing, reading, and project management concerns: COBOL is tedious to write, and easy to read.

Plusses, minuses, and indications: COBOL is tedious, but has features that most of the others lack, such as breaks in report writing when data changes, sorting of files, copying fields by name, and complete control over how fractional arithmetic works in any instance. Passing arguments to subroutines is possible but painful.

.Net ("Dot Net") Languages, including C#, VB.Net, et Cetera

History: The .Net languages were designed to be web-borne languages similar to the original languages. For instance, C# is similar to C++, and has some extensions. The reason for these languages does not seem to be any lack in any of the existing languages, but just an attempt at supplanting Java, which Microsoft found (by virtue of being on the losing end of a lawsuit) that it could not control. Microsoft already had web delivery of program content via Active-X, but the .Net languages have the marketing flavor of Java.

Portability: Not at all, although Linux versions are in development.

Writing, reading, and project management concerns: There is no significant difference in this regard than from the look-like languages.

Plusses, minuses, and indications: For any intended use of the .Net languages, Active X or Java is a better bet. Use Active X if the program is to run on PCs or Java if it is to be portable.

Fortran

History: Fortran was another of computing's first languages. Fortran stands for FORmula TRANslator. There were additional versions of Fortran, named Fortran II, Fortran IV, Fortran 77, and Fortran 90. The goal of each version of Fortran has been to provide the easiest way to write a program to solve a computational problem.

Portability: Fortran is well standardized, and is completely portable.

Writing, reading, and project management concerns: Fortran programs are relatively easy to write and read, so long as the program is basically a number cruncher. String manipulation and data structures are not easy to deal with.

Plusses, minuses, and indications: Complex arithmetic is built-in, and rows, columns, and arrays can be acted on simultaneously.

Java and JavaScript

History: People using the Internet have different types of computers, with different processors and different operating systems, yet they all wanted to be able to load common programs from web sites, run them, and be able to restrict them to certain types of allowed activities. Sun invented Java to do all of this. They created a language that in many respects was like C++, but was in many ways simpler in its interface to the outside world. JavaScript is a scripting language much like Java in its style.

Portability: Designed with portability in mind, it is the most portable language there is. However, newer versions of Internet Explorer do not answer to Java, and need a separate plug-in.

Writing, reading, and project management concerns: JavaScript is fast and easy. The same concerns hold for Java as do for C++.

Plusses, minuses, and indications: As portable as it comes. Memory deallocation in handled for you. However, it will often happen at inopportune times, causing your program to pause now and then. Because it runs in emulation, it is slow on calculations.

Pascal and Ada

History: Pascal was a language invented by Nicolas Wirth. The language had new constructs, such as named values (later to appear in C and other languages), and a self-determining way of handling sub-records. Dr. Wirth also decided to remove some concepts he didn't like, such as GOTO. Ada is a Pascal-like language, but includes a host of multitasking features, and is object-oriented. The DoD, at one point, made Ada its language of choice, because it purported to solve all of programming's ills. Intel built a computer chip to run Ada (the 80432), but its performance was terrible. Apple made Pascal its language of choice. The Lisa has Pascal as its only programming language, and the Macintosh had Pascal as its first programming language.

Portability: Pascal is very portable, assuming no local APIs are used. Ada is also portable.

Writing, reading, and project management concerns: The same concerns hold for Pascal and Ada as hold for C and C++, respectively. However, the Ada environment has a number of very nice project management features built into it, such as automatic make files and clipping out of code that won't be meaningful.

Plusses, minuses, and indications: Not as portable as C or C++. Handles Pascal strings well (obviously), but does not handle null-terminated strings well.

PL/1

History: IBM built PL/1 to be its "do everything" language. PL/1 has all the features of Fortran, COBOL, and the OS/360/370/390/400 operating systems, and some of the features of APL (for A Programming Language) built right in. It looked a lot like Algol (for ALGOrithemic Language), and C looks a lot like PL/1. Because the language did so much, it was hard to learn all of it. Because of that, few places taught it, and because of that, it is little used today. There are now NT and Unix versions.

Portability: PL/1 is now available on IBM mainframes, NT and Unix.

Writing, reading, and project management concerns: PL/1 programs are easy to write, but can be written in a straightforward manner or as complete spaghetti. Obviously, the well-written programs are the easy ones to read.

Plusses, minuses, and indications: Has the positive features of Algol, C, COBOL, Fortran, and Pascal, plus others. It's hard to staff a project with PL/1 programmers, and the code will not be fully portable. Some of the features of PL/1 rely on the features of IBM mainframes, and thus are not portable.

RPG

History: RPG stands for Report Program Generator. It is logically an outgrowth of wired-in programs, and tells the computer either how to act on a stream of data (as most programming languages do), or on a single record. If the program details how to deal with a single record, the program's method is applied to every record in the input stream.

Portability: RPG runs on IBM mainframes only.

Writing, reading, and project management concerns: RPG programs that don't involve a lot of logic are extremely easy to write and OK to read. However, once the level of algorithmic difficulty goes up, writing and reading difficulty go up with it.

Plusses, minuses, and indications: As much as I'd like to be generous here, I can't imagine why anyone would use RPG for anything (said having used it). Whatever you have in mind, C, COBOL, SQL or PL/1 will be better.

SQL, PL/SQL, and Inline SQL

History: SQL, which stands for Structured Query Language, was first developed not as a programming language, but instead as a means of querying to contents of a database. Rather than write a program to tell the system how to read the file and output the result, the programmer instead defines terms and conditions of what are wanted, and the system does the rest. For writing instance,

```
SELECT MAX(salary) FROM employees
```

Is much easier than writing a program for the same purpose. In this example, the output is written to the terminal. Later, PL/SQL (for Programming Language/SQL) was developed so that SQL programs could be written. Inline SQL, also known as C/SQL (or language of your choice/SQL) or SQL*Plus was developed so that SQL could be used inside a C or other program. For instance, to allow a C program to read the value determined above, use

```
EXEC SQL SELECT MAX(salary) INTO  :myVariable FROM employees;
```

Portability: SQL is extremely portable, but there are minor differences in each manufacturer's version of SQL, even when using ANSI SQL. Typically, when moving an SQL application from one system to another, there will be some 'porting cost.

Writing, reading, and project management concerns: Reading and writing are both easy. Writing is easy enough, through, that people sometimes forget to save their code.

Plusses, minuses, and indications: SQL has the concept of Null values, which mean "no value here" as a value. Here's an example: Let's say you have a database with city names and average rainfall values. If you list a city, but don't know its average rainfall value, you would use Null as the value. Null has some drawbacks, too, as one must be handled carefully in comparisons. For most applications involving large amounts of data or even small amounts of correlation, C/SQL or some similar mix is your best bet.

Visual Basic, Access, and ASP

History: Basic was actually Microsoft's first product, and was completely procedural. Visual Basic, or VB, was designed to be completely visual, in that you design your application's look, and then add code to that. Access came after that, and was Visual Basic combined with a database system. ASP is basically a combination of Visual Basic, a web server, and SQL, allowing the programmer to configure a web page on the fly with the database values.

Portability: PC's only. However, Access can act as a client application and communicate with other applications via SQL. ASP can only be hosted on PC's, but the client software can be any web client.

Writing, reading, and project management concerns: Visual Basic is extremely easy to write, but difficult to read. The only reason it's difficult to read is that the code is hard to find. You basically have to look for code behind every button, rather than find it in one convenient location. This also makes project management difficult, in that it is not straightforward to link code from two programmers together, nor to use any sort of source-code control system. Access exacerbates the problem, in that the code and data are linked, and it's hard to run new code on old data, or to drop new data on old code. Further, the Jet database engine within Access does not do well with multiple users. ASP is easy to read, write, and manage.

Plusses, minuses, and indications: Visual Basic is good for quick prototyping, but you see a lot of projects converting to C++ to get over one limitation or another. Access is good for writing single-user database applications that operate one way, and stay that way, but no easier than C/SQL. ASP is quick and easy, but no quicker and easier than C/SQL as a back-end system running as a CGI program. The big question on ASP is server portability.

Other languages

Don't take this as an exhaustive list. There are many other special purpose languages. Among my favorites are APL, MatLab, and Lisp. Don't try these for a large project, though.

Recommendations

The easiest way to pick a language for a project is to rule things out, and pick from whatever is left.

- For instance, does the project have to run on a particular processor? If so, rule out languages that don't run on that processor.
- Does it have to be portable? If so, rule out languages that are not portable.
- Does it have to run at very high throughput levels or carry out large amounts of calculations? If so, rule out languages that are interpreted or run on an emulated machine.
- Does it do something that draws itself to a particular language? This would include presorting, complex, vector, or array math, or something of that nature. If so, use the language that lends itself best to that.

- Still have more than one language in the list? At this point, run a popularity contest among your programmers. If they like one language better than the others, and all else is equal, go with what they like.

Mixed Language Projects

Keep in mind that a large project can have several languages in use. For instance, a two-tier system, with a server and clients need not use the same language for the server code and the client system. As another example, the software for the SDI missile-tracking system uses C code for the software's backbone and common subroutines, but some of the active calculation tasks are written in Fortran 90.

Need help planning a large project?

Call Linker Systems: 800-315-1174