**The Differences Between Java, C And C++**

This article aims to set out some of the differences between C, C++ and Java. Whatit does not aim to do is conclude that one language is always the best one to use.Language choice depends upon a range of factors including field of application(operating systems, desktop software, web applications etc), what programmingparadigm suits the application (OOP, procedural, etc), the target platform andavailable programmer expertise. What follows should help you decide where it mightbe suitable to use C, C++ or Java.

**Paradigm**

C is geared towards procedural programming. That is, you write a number of procedures to do certain tasks and build up a program by calling those procedures asneeded.Java, on the other hand, is geared towards OOP (object oriented programming).With OOP, you define classes which represent an entity (for example, a window, abutton, a string of text, a file). From one class many objects may be created, withevery object of a certain class having the fields (places to store data) and methods(named blocks of code associated with the object) as defined by the class.It is possible to write in an object oriented style in C and in a procedural style inJava, but in each case the language will somewhat get in your way. C++ is designedto support both paradigms.

**Preprocessor**

All C and C++ compilers implement a stage of compilation known as thepreprocessor. The preprocessor basically performs an intelligent search and replaceon identifiers that have been declared using the #define or #typedef directives.#define can also be used to declare macros. For example, a macro MAX(x,y) couldbe defined to return whichever of x or y holds the greatest value. This is not likecalling a function as the substitution is done before the code is compiled. Most of thepreprocessor definitions in C and C++ are stored in header files, which complementthe actual source code files.Java does not have a preprocessor. Constant data members are used in place of the#define directive and class definitions are used in lieu of the #typedef directive,however there is no substitute for macros, which can be useful. The Java approachto defining constants and naming types of data structures is probably conceptuallysimpler for the programmer. Additionally, Java programs don't use header files; theJava compiler builds class definitions directly from the source code files, whichcontain both class definitions and method implementations.

**Memory Management**

In C and C++, any memory that is allocated on the heap (e.g. using malloc or new)must be explicitly freed by the programmer (e.g. using free or delete). Forgetting tofree memory leads to memory leaks, and in long-running programs can lead to the

http://htmlimg1.scribdassets.com/7r2tjt53cw58bfx/images/1-d2111a9f08.jpg

memory usage of the program growing very large.Java provides garbage collection, meaning that memory is freed automatically whenit is no longer reachable by any references. This prevents memory leaks, but canlead to pauses in execution while the garbage collector runs. Also, there is nopromise of timely destruction in Java.

**Pointers**

Most developers agree that the misuse of pointers causes the majority of bugs in Cand C++ programs. Put simply, when you have pointers, you have the ability toattempt to access memory that isn't yours and modify memory relating to a differentdata structure than the one you intended by accident. C/C++ programmers regularlyuse complex pointer arithmetic to create and maintain dynamic data structures. It'spowerful, but can lead to a lot of time spent hunting down complex and often subtlebugs that arise as a result of having unguarded memory access.The Java language does not support pointers. Instead, it provides similarfunctionality by making heavy use of references. A reference can be thought of as a"safe pointer" - the programmer can not directly manipulate the memory address.Java passes all arrays and objects by reference. This approach prevents commonerrors due to pointer mismanagement. It also makes programming easier in a lot of ways simply because the correct usage of pointers is easily misunderstood byinexperienced programmers.C++ does provide references too. It considers them as aliases to another variable orobject. They are safer than pointers where they can be used.

**Bounds Checking**

An array in C or C++ is not bounds checked, so attempts to access the sixth elementof a 5-element array will appear to work - that is, no runtime error will occur. Thismeans the programmer needs to code very carefully, especially considering thepotential for buffer overflow attacks.Java will bounds check arrays to prevent this from happening, of course with a littleextra runtime cost.

**Portability And Performance**

C and C++ both compile to native machine code. This means that, with a goodcompiler, programs written in these languages will perform very well. However, italso restricts them to running on the platform they were compiled to run on.Java generally compiles to Java bytecode, which then runs on top of a virtualmachine (the JVM). The JVM has to turn instructions in the bytecode into instructionsthat are understood by the machine that the bytecode is running on. This gives aruntime performance penalty (although this is getting less significant as the JVMimproves and computers get faster). However, now only the virtual machine (andstandard library) have to be ported to different platforms, then the bytecode formany Java programs can be executed on that platform. So bytecode is portableaccross different operating systems and processors.

**Complex Data Types**

There are two types of complex data types in C: structures and unions. C++ addsclasses to this list. Java only implements one of these data types: classes.A structure can be emulated by a class - simply write a class without any methodsand make all the fields public. However, emulating a union is not always possible inJava, and the memory saving advantages unions hold in C may not carry accross.Java presents a simpler model but at the cost of not being able to save a littlememory. For many applications this will be a non-issue.

**Strings**

C has no built-in string data type. The standard technique adopted among Cprogrammers is that of using null-terminated arrays of characters to representstrings. This practice if often seen in C++ programs too.Neither C++ or Java have string as a primitive type, but they do both have stringobjects that are a standard part of the language. In Java this type is called String,and in C++ it is called CString.

**Multiple Inheritance**

Multiple inheritance is a feature of some object oriented languages that allows you toderive a class from multiple parent classes. Although multiple inheritance is indeedpowerful (and sometimes the logical way to define a class hierachy), it is complicatedto use correctly and can create situations where it's uncertain which method will beexecuted. For example, if each of the parent classes provide a method X and thederived class does not, it is unclear which X should be invoked. It is also complicatedto implement from the compiler perspective.C++ supports multiple inheritance. Java provides no direct support for multipleinheritance, but you can implement functionality similar to multiple inheritance byusing interfaces in Java. Java interfaces provide method descriptions but contain noimplementations. Therefore implementations can only be inherited from one class, sothere is no ambiguity over which method to invoke.

**Operator Overloading**

Operator overloading enables a class to define special behaviour for built-inoperators when they are applied to objects of that class. For example, if the \*(multiply) operator was to be used on two objects of type Matrix, then matrixmultiplication could be implemented. This allows object types to feel much moretightly integrated into the language and can deliver much clearer code. However,sometimes it is not clear what a particular operator would sensibly do for a particulartype, whereas a well-named method call would be clear.Operator overloading is considered a prominent feature in C++. It is not supportedin Java, probably in an effort to keep the language as simple as possible and helpensure it is obvious what code does, even though it may take longer to type andread.

**Automatic Coercions**

Automatic coercion refers to the implicit casting of data types that sometimes occursin C and C++. For example, in C++ you can assign a float value to an int variable,which can result in a loss of information, although a compiler warning will be givenabout this. Java does not support C++ style automatic coercions. In Java, if coercionwill result in a loss of data, you must always explicitly cast the data element to thenew type.

**Goto Statement**

The goto statement is rarely used these days in C and C++, but it is a standard partof the language. The goto statement has historically been cited as the cause formessy, difficult to understand, and sometimes near impossible to predict code knownas "spaghetti code." The primary bad usage of the goto statement has merely beenas a convenience to substitute not thinking through an alternative, more structuredbranching technique. Very occasionally, it can lead to clearer code.To avoid the potential for "spaghetti code", Java does not provide a goto statement.The Java language specifies goto as a keyword, but its usage is not supported. Thisis consistent with Java's desire to make programmers write clear, non-messy code.

**Variadic Arguments**

C and C++ let you declare functions, such as printf, that take a variable number of arguments. Although this is a convenient feature, it is impossible for the compiler tothoroughly type check the arguments, which means problems can arise at runtimewithout you knowing. Java doesn't support variable arguments at all, though if it didit would likely be able to handle subsequent runtime problems better than C or C++.

**Command-line Arguments**

The command-line arguments passed from the system into a Java program differ ina couple of ways from the command-line arguments passed into a C++ program.First, the number of parameters passed differs between the two languages.In C and C++, the system passes two arguments to a program: argc and argv. argcspecifies the number of arguments stored in argv. argv is a pointer to an array of characters containing the actual arguments. In Java, the system passes a singlevalue to a program: args. ‘args’ is an array of Strings that contains the command-line arguments.

**Table Comparing C, C++ and Java**

This table is a summary of the differences found in the article.

|  |  |  |  |
| --- | --- | --- | --- |
| Feature | C | C++ | Java |
| Paradigms | Procedural | Procedural, OOP, Generic Programming | OOP, Generic Programming (from Java 5) |
| Form of Compiled Source Code | Executable Native Code | Executable Native Code | Java bytecode |
| Memory management | Manual | Manual | Managed, using a garbage collector |
| Pointers | Yes, very commonly used. | Yes, very commonly used, but some form of references available too. | No pointers; references are used instead. |
| Preprocessor | Yes | Yes | No |
| String Type | Character arrays | Character arrays, objects | Objects |
| Complex Data Types | Structures, unions | Structures, unions, classes | Classes |
| Inheritance | N/A | Multiple class inheritance | Single class inheritance, multiple interface implementation |
| Operator Overloading | N/A | Yes | No |
| Automatic coercions | Yes, with warnings if loss could occur | Yes, with warnings if loss could occur | Not at all if loss could occur; msut cast explicitly |
| Variadic Parameters | Yes | Yes | No |
| Goto Statement | Yes | Yes | No |