

C Programming vs. Java Programming

Thing	C	Java
type of language	function oriented	object oriented
basic programming unit	function	class = ADT
portability of source code	possible with discipline	yes
portability of compiled code	no, recompile for each architecture	yes, bytecode is "write once, run anywhere"
security	limited	built-in to language
compilation	<code>gcc hello.c</code> creates machine language code	<code>javac Hello.java</code> creates Java virtual machine language bytecode
linking in the Math library	<code>gcc -lm calculate.c</code>	no special flags needed
joint compilation	<code>gcc main.c helper1.c helper2.c</code>	<code>javac Main.java</code> - any dependent files are automatically re-compiled if needed
execution	<code>a.out</code> loads and executes program	<code>java Hello</code> interprets byte code
hello, world	<pre>#include<stdio.h> int main(void) { printf("Hi\n"); return 0; }</pre>	<pre>public class HelloWorld { public static void main(String[] args) { System.out.println("Hi"); } }</pre>
integer types	<code>int</code> usually 32 bit 2's complement; <code>long</code> usually 32 bit 2's complement	<code>int</code> is 32 bit 2's complement; <code>long</code> is 64 bit 2's complement
floating point types	<code>float</code> usually 32 bit; <code>double</code> usually 64 bit	<code>float</code> is 32 bit IEEE 754 binary floating point; <code>double</code> is 64 bit IEEE 754
boolean type	use <code>int</code> : 0 for false, nonzero for true	<code>boolean</code> is its own type - stores value <code>true</code> or <code>false</code>

character type	char is usually 8 bit ASCII	char is 16 bit UNICODE
for loops	<code>for (i = 0; i < N; i++)</code>	<code>for (int i = 0; i < N; i++)</code>
array declarations	<code>int *a = malloc(N * sizeof(*a));</code>	<code>int[] a = new int[N];</code>
array size	arrays don't know their own size	<code>a.length</code>
strings	'\0'-terminated character array	built-in immutable String data type
accessing a library	<code>#include <stdio.h></code>	<code>import java.io.File;</code>
accessing a library function	<code>#include "math.h"</code> <code>x = sqrt(2.2);</code> all function and variables names are global	<code>x = Math.sqrt(2.2);</code> functions have different namespaces
printing to standard output	<code>printf("sum = %d", x);</code>	<code>System.out.println("sum = " + x);</code>
formatted printing	<code>printf("avg = %3.2f", avg);</code>	<code>System.out.printf("avg = %3.2f", avg)</code>
reading from stdin	<code>scanf("%d", &x);</code>	Java library support, but easier to use our library <code>int x = StdIn.readInt();</code>
memory address	pointer	reference
manipulating pointers	<code>*, &, +</code>	no direct manipulation permitted
functions	<code>int max(int a, int b)</code>	<code>public static int max(int a, int b)</code>
pass-by-value	primitive data types, structs, and pointers are passed by value; array decays to pointer	all primitive data types and references (which includes arrays), are passed by value
defining a data structure	<code>struct</code>	class - key difference is language support for defining methods to manipulate data
accessing a data structure	<code>a.numerator</code> for elements	<code>a.numerator</code> for instance variables, <code>c = a.plus(b)</code> for methods

pointer chasing	<code>x->left->right</code>	<code>x.left.right</code>
allocating memory	<code>malloc</code>	<code>new</code>
de-allocating memory	<code>free</code>	automatic garbage collection
memory allocation of data structures and arrays	heap, stack, data, or bss	heap
buffer overflow	segmentation fault, core dump, unpredictable program	checked run-time error exception
declaring constants	<code>const</code> and <code>#define</code>	<code>final</code>
variable auto-initialization	not guaranteed	instance variables (and array elements) initialized to 0, <code>null</code> , or <code>false</code> , compile-time error to access uninitialized variables
data hiding	opaque pointers and <code>static</code>	<code>private</code>
interface method	non-static function	<code>public</code> method
data type for generic item	<code>void *</code>	<code>Object</code>
casting	anything goes	checked exception at run-time or compile-time
demotions	automatic, but might lose precision	must explicitly cast, e.g., to convert from <code>long</code> to <code>int</code>
polymorphism	<code>union</code>	inheritence
overloading	no	yes for methods, no for operators
graphics	use external libraries	Java library support, use our standard drawing library
null	<code>NULL</code>	<code>null</code>
enumeration	<code>enum</code>	typesafe <code>enum</code>
preprocessor	yes	no

variable declaration	at beginning of a block	before you use it
variable naming conventions	<code>sum_of_squares</code>	<code>sumOfSquares</code>
commenting	<code>/* */</code>	<code>/* */</code> or <code>//</code>
file naming conventions	<code>stack.c</code> , <code>stack.h</code>	<code>Stack.java</code> - file name matches name of class
callbacks	pointers to global functions	use interfaces for command dispatching
variable number of arguments	<code>varargs</code>	<code>String ...</code>
assertions	<code>assert</code>	<code>assert</code>
exit and return value to OS	<code>exit(1)</code>	<code>System.exit(1)</code>