

# 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	gcc hello.c creates machine language code	javac Hello.java creates Java virtual machine language bytecode
linking in the Math library	gcc -lm calculate.c	no special flags needed
joint compilation	gcc main.c helper1.c helper2.c	javac Main.java - any dependent files are automatically re-compiled if needed
execution	a.out loads and executes program	java Hello interprets byte code
hello, world	#include<stdio.h> int main(void) { printf("Hi\n"); return 0; }	public class HelloWorld { public static void main(String[] args) { System.out.println("Hi"); } }
integer types	int usually 32 bit 2's complement; long usually 32 bit 2's complement	int is 32 bit 2's complement; long is 64 bit 2's complement
floating point types	float usually 32 bit; double usually 64 bit	float is 32 bit IEEE 754 binary floating point; double is 64 bit IEEE 754
boolean type	use int: 0 for false, nonzero for true	boolean is its own type - stores value true or false

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

pointer chasing	<code>x-&gt;left-&gt;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, unpredicatable 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, null, 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	sum_of_squares	sumOfSquares
commenting	/* */	/* */ or //
file naming conventions	stack.c, stack.h	Stack.java - file name matches name of class
callbacks	pointers to global functions	use interfaces for <a href="#">command dispatching</a>
variable number of arguments	varargs	String ...
assertions	assert	assert
exit and return value to OS	exit(1)	System.exit(1)