C Basics 1: Overview & C vs. Java

1. **Overview of C and C vs. Java**
   - **history of C**
   - **C standards**
   - **C vs. Java characteristics**
2. C Program Elements
3. The Preprocessor
4. Identifiers and Data Types
5. Pointers and Arrays
6. Strings
7. Library I/O

C Basics 1: Overview & C vs. Java

8. Errors and Error Handling
9. Functions and Parameter Passing
10. Dynamic and Static Memory
11. Multidimensional and String Arrays
12. Structs (Structures)
13. Higher-Order Functions
14. Variadic Functions
15. Miscellaneous

Overview of C

C was developed by Dennis Ritchie at AT&T's Bell Labs, starting in 1969.

A key goal was to provide a "high-level" programming language to support development of **portable**, **cross-platform** programs (up to that point, many programs were being written in machine-specific **assembly language**).

Richie also co-developed **UNIX** (along with Ken Thompson and others) starting around the same time.

By 1973, UNIX had been implemented primarily in C, becoming the first (largely) *portable* operating system.

Overview of C (contd.)

While C is much more abstract than assembly language, it lacks features found in modern "high-level" programming languages that are designed to support programming.

C programs are intended to be able to be made nearly as efficient as assembly language programs.

It is often said that a (properly designed) C program will be about as fast as is possible.

With a C program, what occurs at runtime is largely only what the program indicates will happen.

By contrast, in a *Java* program, every array access will be *bounds checked*, *garbage collection* routines will be run periodically, etc.

Overview of C (contd.)

While programmers new to C are often dismayed by the language failing to "hold their hands" and help discover their bugs, if a program is properly designed, automatic operations such as array bounds checking are simply a waste of CPU cycles.

C also differs from most modern high-level languages in allowing the user to directly access and manipulate memory, via **pointers**.

C remains the most commonly used language for programming with **system calls** and for **system programming** (coding of *operating systems* and **embedded systems** software).

C is also one of the most widely available languages, with *free-of-cost* compilers for virtually every computer architecture.

C Language Standards

In 1978, Brian Kernighan and Dennis Ritchie wrote the first book describing C: "**The C Programming Language**."

This book became the *de facto* language standard, often referred to as **K&R C**.

The first standardized versions of C were ratified by ANSI in 1989 and ISO in 1990, commonly referred to as **C89** or **C90**.

A revised version of the ISO C standard was published in 1999, commonly referred to as **C99** (its formal name is **ISO/IEC 9899:1999**).

Among the features added in C99 were: variable-length arrays (array size determined at runtime), intermingling of declarations and code, allowing index variable declarations in `for` statements, new types (`long long int`, `_Bool`, complex), and inline functions.

C Language Standards (contd.)

The latest C standard was ratified by ISO in 2011: **ISO/IEC 9899:2011**.

The *draft* standard had been known as **C1X**, but the ratified standard is referred to as **C11**.

Key extensions of general interest include: better Unicode support, generic function selector (`_Generic`), and language support for **multithreaded programs**.

The multithreading support involves several elements:
an uninterruptible data type (`_Atomic`), thread specific/local storage, and library functions that provide thread operations (creation, mutexes, condition variables, etc.).

This standard also addresses special uses such as embedded processors and better defines various aspects of implementations.

C Language Standards (contd.)

Most C compilers, e.g., **GCC**, now support all C99 features.

However, C99 features may *not all be enabled by default*, requiring options to compile code that uses them.

E.g., with GCC: **-std=c99**

C11 support is still incomplete in most current C compilers.

GCC 4.9 is supposed to support virtually all of the C11 standard when it is released (sometime in 2014).

C vs. Java

Java is a "**C family**" language and so shares much syntax with C, such as declarations, control constructs, operators, etc.

C is a purely **procedural language**, however, while Java is a purely **object-oriented language**.

C does *not have any support* for **OOP** concepts like **classes**.

In a procedural language like C, programs are structured in terms of **functions** (**subroutines**) operating on **data** held in **variables**.

The focus is on the *actions* that need to be taken to accomplish the goals of the program.

With OOP, programs are structured in terms of classes of objects and the operations that can be done to them.

C vs. Java (contd.)

	C	Java
Programming Paradigm	pure procedural	pure OOP
Composite Data Types	structs	classes
Inheritance	no	single: subclass `extends`
Encapsulation Mechanism	files	classes
Compilation Modularity	files	classes (files)
Memory Management (heap)	manual `malloc()`, `realloc()`, `free()`, etc.	garbage collection `new`

C vs. Java (contd.)

	C	Java

Pointers	explicit, manipulable:
	defined: *basetype*`*`
	`int *ip, i;`
	`ip = &i + 1;`
		implicit, not manipulable
		called *references*
		all classes

Invalid Pointer	`NULL` `(void *)0`	`null`
Numeric Types	integer: `char`, `short`, `int`, `long`, `long long` real: `float`, `double`, `long double` (C99) other: `_Bool` (C99) sign: `signed`, `unsigned` minimum sizes only	integer: `byte`, `short`, `int`, `long` real: `float`, `double` (all integers signed) defined sizes

C vs. Java (contd.)

	C	Java
Boolean Type	<C99 no, use `int`'s C99: `_Bool` (0/1 `int`) 0 is false 1 is standard true, but any $\neq 0$ considered true	yes: `false`, `true`
Characters	ASCII encoding single byte integer constants: `'a'`, `'\141'` C99: wide char's C11: Unicode	Unicode encoding two bytes constants: `'a'`, `'\u09AF'`
Strings	non-builtin type: type `char*` (`char` array, requires `'\0'` sentinel) constants: `"abc"`	builtin reference type: `String` constants: `"abc"`
String Comparisons	string library functions e.g., `strcmp()`	String class methods e.g., `.equals()`

C vs. Java (contd.)

	C	Java
Implicit Type Conversion	extensive: *promotion & demotion* may not preserve values: `unsigned int`→`signed int`	limited: promotion
Type Casting	yes: (*type*) *var/expr*	same

Arrays	access: `arr[i]`
	declaration:
	`int iarr[`*constant*`]`
	(C99 allows *expr*)
	static size
	no bounds checking
		access: `arr[i]`
		declaration:
		`int[] iarr =`
		`new int[`*expr*`]`
		static size (vs. vectors)
		auto bounds checking

C vs. Java (contd.)

	C	Java
Main	`int main(int argc, char *argv[])` `int main()`	`public static void main(String[] args)`
Operators	assignment: `=`, `+=`, etc. arithmetic: `+`,`++`, etc. relational: `==`, `!=`, etc. logical/bitwise: `&&`, `&`, etc.	same same same similar
Selection Constructs	`if`, `else`, `switch`	same
Looping Constructs	`for`, `while`, `do-while`	same
Global Variables	yes (external definitions)	no, but: `public static` members
Automatic Variable Initialization	generally no: global and `static` only	yes (to 0 or null)

C vs. Java (contd.)

	C	Java
Subroutines	functions	class methods
Subroutine Values	any type plus void	same
Parameter Passing	call-by-value: but pointer params act like call-by-ref	call-by-value: but reference params act like call-by-ref

Generic Functions	use untyped pointers:
	`void*`
	C11: generic selector
	`_Generic`
		generics system
		(SE 5.0+)

Function Overloading	no	yes (methods)
Operator Overloading	no	no

C vs. Java (contd.)

	C	Java
Function Arguments (Callbacks)	yes: `hofunc(void (*h)(int));`	SE 8.0+: lambdas (previously only messy alternatives to achieve effect)
Variadic Functions	yes: (stdarg.h) `printf(char *fmt, ...);`	SE 5.0+
Optional or Default Parameters	no	no
Preprocessor	yes	no
Header Files	yes	no
Namespaces	no, single space (for *ordinary identifiers*)	yes: `package`, `import`
Exception Mechanism	no	yes: `try`, `catch`

C vs. Java (contd.)

	C	Java
Concurrency: Processes	via system calls (`fork()`, etc.)	limited: `ProcessBuilder` class
Concurrency: Threads	via system calls (Pthreads) C11: yes	yes: `Thread` class `java.util.concurrent` package