C vs Java: Portability

Java
- Source and binaries are entirely portable
- Source format is standardized
- Binaries run on a software emulation of a standardized processor
- Execution slow

C
- Binaries are not usually portable from one platform to another
- Use the platform's native, hardware processor directly
- Runs faster
- Can be portable with little modification if it adheres to the ANSI C standard

Subject to the available libraries
Java vs C: Speed and Ease of Development

- **Java programs may be easier to develop**
  - Dynamic memory management is largely automatic
  - Diagnostic exceptions are thrown for illegal operations

- **C programs will usually run faster**
  - Dynamic memory management is fully under the programmer's control
  - There are no checks for illegal operations
  - Require greater responsibility from the programmer
```c
#include <stdio.h> /* include information about standard library */

int main() /* define a function called main that receives */
/* no argument value and returns an integer */
{
    /* statements of main are enclosed in braces */
    printf("Hello World!\n"); /* main calls library function printf to a sequence of */
    /* characters. \n represents the newline character */
    return 0; /* function main returns an integer of value 0 to the */
    /* operating system. */
} /* main ends */
```
Primitive Types

- In C, the primitive types are referred to using a combination of the keywords:
  - char, int, float, double, signed, unsigned, long, short and void.
  - Meanings depend on the compiler and platform in use, unlike Java.

- **char**: A single byte, capable of holding one character in the local character set
- **int**: An integer, typically reflecting the natural size of integers on the host machine
- **float**: A single precision floating-point type, correct to 6 decimal places.
- **double**: A double precision floating-point type, correct to 10 decimal places.
Primitive Types

- **short**: often 16 bits
- **long**: often 32 bits
- **unsigned**: always positive or zero
- **long double**: An extended double precision floating-point type.
- **void**: An empty type, has no value, cannot be accessed.
  - As in Java, C functions with no return value are defined to return **void**.
  - Unlike Java, a function with no parameters has **void** in its parameter list.
- There is no **boolean** type.
  - Use integer expressions with a boolean interpretation: zero means false, non-zero means true.
  - Relational operators (==, !=, <=, >=, < and >) and logical operators return 0 for false and 1 for true.
Variable Storage

- **Format**: [Storage class] type variable_name

- **auto**: Most common, the keyword auto is seldom used
  - When the block is entered, memory is allocated for the automatic variables
  - When the block is exited, the memory set aside for the automatic variable will be released.

- **static**: Allow a local variable to retain its previous value when the function is reentered

- **register**: Improve execution speed by putting the data in a register rather than a memory location

- **extern**: inform the compiler to look for declaration of the variable elsewhere
  - Either within the same, file or in another file
  - No new variable is created
const

C uses the keyword const with an object declaration to indicate a constant object that can (and must) be initialized, but cannot subsequently be assigned to — it is not a variable, but it still has an address and a size.

Like final in Java

double sin(double); /* declaration of mathematical function sine */

const double pi = 3.14159;
double val;

val = sin(pi); /* legal expression */
pi = 3.0;      /* illegal; not a modifiable lvalue */
printf: print output onto the screen

```c
#include <stdio.h>
int main()
{
    int a = 72;
    char b = 'A';
    printf("a equals %d \n", a);
    printf("a equals %c \n", a);
    printf("b equals %d %c\n", b, b);
}
```

Output on screen:
a equals 72
a equals H
b equals 65 A

<table>
<thead>
<tr>
<th>Format specifier</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>%d or %i</td>
<td>int</td>
</tr>
<tr>
<td>%c</td>
<td>char</td>
</tr>
<tr>
<td>%f</td>
<td>float/double</td>
</tr>
<tr>
<td>%s</td>
<td>string</td>
</tr>
</tbody>
</table>
6 Bitwise Operators

- Perform calculations using binary digits
- Can be applied to integral operands only
- char, int, short, long
- Need to know how to convert to and from decimal and binary
- Don't confuse the bitwise operators with the logical ones!
- Logical AND &&, OR ||, NEGATION !
- Operands may be expressed in hexadecimal numbers (0x) for implementation
Bitwise Operators

& : bitwise AND, align the right most bits
   1000 & 1101 = 1000
   00100 & 10111 = 00100
   01111 & 11100 = 01100

| : bitwise inclusive OR
   1000|1101=1101
   00100|10111=10111
   01111 | 11100 = 11111

^ : bitwise exclusive OR
   1000 ^ 1101 = 0101
   00100 ^ 10111 = 10011
Bitwise Operators

\(<<\) : left shift, multiply an integer by a power of 2
5 \(<<\) 3 = 5 \times 8 = 40 (101 becomes 101000)
1 \(<<\) 2 = 1 \times 4 = 4 (1 becomes 100)

\(>>\) : right shift (takes away bits on the right)
8 \(>>\) 2. 8 is 1000 in binary. Performing a right shift of 2 involves knocking the last 2 bits off, which leaves us with 10, i.e. 2.

\(\sim\) : one’s complement, unary, bitwise NOT, convert each 1 into 0 and vice versa
Suppose we assign 9 to an unsigned short int declared variable x.
x is 0000 0000 0000 1001 (0x9)
\(\sim x\) is 1111 1111 1111 0110 (0xffff6)
Comments

- Java allows the use of these forms of comment:

  /*This is OK in C */

  /* This is a valid C comment,
     a multiline comment */

  // a single line comment like this is not valid in C

- No nested comments in C
C structure
- like a Java class that only contains public data members
- there must be no functions
- all parts are visible to any code that knows the declaration

This declares a type called `struct point`
- `struct` is part of the name
- `point` is known as the structure type's tag

```c
struct point {
    int x, y;
};
```

Members of a C structure are accessed using the . operator, as class members can be in Java:

```java
struct point location;

location.x = 10;
location.y = 13;
```
A structure object may be initialized where it is defined:

```c
struct point location = { 10, 13 };
/* okay; initialization (part of definition) */
location = { 4, 5 }; /* illegal assignment (not part of definition) */
```

Unlike Java, where class variables are references to objects,
C structure variables are the objects themselves,
Assigning one to another causes copying of the members

```c
struct point a = { 1, 2 };
struct point b;
b = a;   /* copies a.x to b.x, and a.y to b.y */
b.x = 10; /* does not affect a.x */
```
Enumerations

- An enumeration defines several symbolic integer constants with unique values.

- The following declares a new type enum light, and defines the symbols RED for 0, BLUE for 1, GREEN for 2, and AMBER for 3:
  ```
  enum light { RED, BLUE, GREEN, AMBER };
  ```

- The first symbol is assigned the value 0, and each subsequent symbol is assigned the next integer.

- If a new type is not required, the tag can be omitted:
  ```
  enum { RED, BLUE, GREEN, AMBER };
  ```

- The symbols can be used in any expression, and may be assigned to any integral type, not just the enum type. For this reason, the tag is rarely used.
C allows an area of memory to be occupied by data of several types, but it can only store information in one field at any one time.

Unions are syntactically similar to structures:

```c
union number {
  char c;
  int i;
  float f;
  double d;
};
union number n;
int j;

n.i = 10;
j = n.i;
```

- This declares a type called `union number`
- `union` is part of the name
- `number` is known as the union's tag

Members of a C union are accessed using the . operator, just as structure members are accessed:
Union

- Only the member to which a value was last assigned contains valid information to be read
- No way to determine that member implicitly
- Programmer must take steps to identify it (by using a separate variable to indicate the type)
- Java does not have unions

```cpp
union number n;
enum { CHAR, INT, FLOAT, DOUBLE } nt;
n.i = 10;
nt = INT;

switch (nt) {
  case CHAR:
    /* access n.c */
    break;
  case INT:
    /* access n.i */
    break;
  case FLOAT:
    /* access n.f */
    break;
  case DOUBLE:
    /* access n.d */
    break;
}
```
Namespace

- In C, all functions are global, and must share a single namespace (i.e. one per program).
- Global variables can also be declared and defined, and they also share that namespace.
- Other namespaces exist in C:
  - A single namespace is shared by the tags of all structures, unions and enumerations.
  - Each structure and union holds a unique namespace for its members.
  - Each block statement {...} holds a namespace for local variables.
Name Overloading and Alias

- No function name overloading
  - In Java, two functions in the same namespace may share the same name if their arguments are sufficiently different.
  - In C, all function names must be unique.

- New names or aliases for existing types may be created using typedef. Eg.

  ```
typedef int int32_t;
  ```

  - This allows int32_t to be used anywhere in place of int,
  - Often used to hide implementation- or platform-specific details
  - Allow the choice of a widely-used type to be changed easily.
  - There is no equivalent of type aliasing in Java.
Declarations and Definitions

- C compilers read through source files sequentially, looking for names of types, objects and functions being referred to by other types, objects and functions.

- A declaration of a type, object or function tells the compiler that a name exists and how it may be used, and so may be referred to later in the file.

- If the compiler encounters a name that does not have a preceding declaration, it may generate an error or a warning because it does not understand how the name is to be used.

- A definition of an object or function tells the compiler which module the object or function is in (Program modularity). For an object, the definition may also indicate its initial value. For a function, the definition gives the function's behaviour.
In Java, the use of a function may appear earlier than its definition.
In C, all functions being used in a source file should be declared somewhere earlier than their invocations in that file.
A function declaration (or prototype) looks like a function definition, but its body (the code between the \{ statements \}) is replaced by a semicolon.
If the compiler finds a function invocation before any declaration, it will try to infer a declaration from the invocation, and this may not match the true definition.

/* a declaration; parameter names may be omitted */
int power(int base, int exponent);

/* From here until the end of the file, we can make calls to power(), even though the definition hasn't been encountered. */

/* a definition; parameter names do not need to match declaration */
int power(int b, int e) {
    int r = 1;
    while (e-- > 0)
        r *= b;
    return r;
}
C Compiler in UNIX

```
gcc world.c -o world
```

- gcc ran the gcc compiler.
- The filename after gcc is the file to be compiled, which is "world.c" in this case.
- The filename after -o is the output file, which is "world".
- The output filename is the word you type to run the program.
- If "-o world" is missing, a file a.out will be produced.

- Compiler is a program that translates source code into machine code.
- The LINKER is a program that is run after compilation. Its job is to piece together various parts of a program, in order to produce the final executable.
main() function

int main(int argc, char **argv);

- The parameters represent an array of character strings that form the command that ran the program. argv[0] is usually the name of the program, argv[1] is the first argument, ..., argv[argc - 1] is the last, and argv[argc] is a null pointer. For example, the command in UNIX

  myprog x1 y2

cause main to be invoked as if by:

  char a1[] = "myprog";
  char a2[] = "x1";
  char a3[] = "y2";
  char *argv[4] = { a1, a2, a3, NULL };
  main(3, argv);

The parameters are optional (you can replace them with a single void), but main always returns int in any portable program. Returning 0 tells the environment that the program completed successfully. Other values (implementation-defined) indicate some sort of failure.