CSCI 2132: Software Development

C vs Java

Norbert Zeh

Faculty of Computer Science
Dalhousie University

Winter 2019

Comparing C to Java

Assumption: You know Java well.

Focus on differences between C and Java.

Arithmetic Operators

Most operators are the same: `+`, `-`, `*`, `/`, `%`, `++`, `--`, `=`, `+=`, ...

Some differences:

- `%` cannot be applied to floating point numbers.
- Integer division (`/`) has implementation-defined behaviour for negative numbers in earlier C standards.
- **C99** defines that integer division rounds towards 0.

Concept: Implementation-defined behaviour

Expression Evaluation

Order of evaluation:

- Java: left-to-right
- C: Unspecified

Example:

```
a = 5;
c = (b = a + 2) - (a = 1);
```

Result:

- Java: 6
- C: 6

Logical Expressions

Operators as in Java:

- Comparison: `<`, `>`, `<=`, `>=`, `==`, `!=`
- Logical operators: `!`, `&&`, `||`
- Logical operators short-circuited in both languages

Representation of Boolean values:

- Java: `boolean`
- C: `int` (C99 has a `bool` type but `int` is still in use, `bool` not mandatory)
- `int` as Boolean: `0` = false, anything else = true

`int` as Boolean

Allows convenient compact notation:

```
int f = 1, i = n;
while (--i) f *= i + 1;
```

But beware:

```
if (a < i < b) { ... }
```

An extremely common mistake the compiler won’t catch:

```
if (x = a + b) { ... }
```

Short-Circuit Evaluation

Applies to `&&` and `||`, as in Java

Example:

```
if (a != 0 && b/a > 2) { ... }
```

Control Structures

- `if`, `switch`, `while`, `do-while`, and `for` work as in Java
- `break` works as in Java but does not accept a label
- To continue to the next iteration of a loop: `continue`
- Return from a function: `return`

Only in C:

- `goto label`: jump to `label` (within the same function)
- `label:` define a label
- Exit the program:
  - `exit()` function defined in `stdlib.h`
  - `return` from `main` function

Variable Declaration in `for`-Loop

Java allows

```
for (int i = 0; i < 10; i++) ...
```

- Not allowed in C before C99
- Allowed in C99

The Comma Operator

```
x = (a = 3, b = 4, c = 5);
```

- Expressions can be sequenced with `,`
- Value of the whole expression is the value of the last subexpression

- Useful in `for`-loops:

```
for (i = 0, j = 0; i < 10; ++i)
  if (a[i] != 0) b[j++] = a[i];
```

goto Statement

```
#include <stdio.h>

int main() {
  int i = 1;

  loop: printf(“%d\n”, i);

  ++i;
  if (i <= 10) goto loop;

  return 0;
}
```

Some Notes about `goto`

`goto` mirrors how your CPU implements loops and conditionals.

Basic and FORTRAN were not as structured as C and used `goto` as their main looping and branching construct.

Use of `goto` is discouraged in structured programming:

- Most control flows can be implemented without `goto`.
- Excessive use of `goto` leads to “spaghetti code”, hard to read.

Typical Uses of `goto`

- Machine-generated code
- In place of labelled `break`:

```
while (...) {
  for (...) {
    ...
    if (...) goto loop_done;
    ...
  }
}

loop_done: ...
```

Null Statement

- Does nothing
- Simply put a semicolon (`;`)
- Often used in `for`-loops:

```
for (d = 2; d < n && n % d != 0; ++d);
if (d < n)
  printf(“%d is not a prime number\n”, n);
```