C and Java
some big differences

Some C

August 30, 2016

- object-oriented vs procedural
- non-interpreted vs interpreted
- memory management
- references vs. free, unrestricted pointers
- error handling

A Very Simple Program

Java
1 public class Welcome {
2  public static void main(String args[]) {
3   System.out.println("Welcome to CIS 2107");
4 
5 }
}

C
1 #include <stdio.h>
2 int main(int argc, char **argv) {
3   printf("Welcome to CIS 2107\n");
4   return 0;
5 }
6
C's main()
- starting point of the program
- returns int, not void
- return status 0 → OK
- int argc, char **argv same as Java's String args
A Very Simple Program

```java
public class Welcome {
    public static void main(String args[]) {
        System.out.println("Welcome to CIS 2107");
    }
}
```

```c
#include <stdio.h>

int main(int argc, char **argv) {
    printf("Welcome to CIS 2107\n");
    return 0;
}
```

#include

- idea not unlike Java import. different mechanism
- preprocessor. text mangling

Compiling and Running in C

... but it's really not that bad

```
gcc -o executable_file_name source_file_name
```

formal
but it’s really not that bad

```bash
# formal
gcc -o executable_file_name source_file_name

# example
gcc -o stuff stuff.c
```

and to execute

```
./stuff
```

Java Data Types

- all types
- primitives
- objects

Java Primitives

- integer types
  - byte
  - short
  - int
  - long
- floating-point types
  - float
  - double
- characters
  - char
- boolean
  - boolean
### Java Primitives vs. C Types

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>integer types</strong></td>
<td><strong>integer types</strong></td>
</tr>
<tr>
<td>byte</td>
<td>char</td>
</tr>
<tr>
<td>short</td>
<td>short</td>
</tr>
<tr>
<td>int</td>
<td>int</td>
</tr>
<tr>
<td>long</td>
<td>long</td>
</tr>
<tr>
<td><strong>floating-point types</strong></td>
<td><strong>floating-point types</strong></td>
</tr>
<tr>
<td>float</td>
<td>float</td>
</tr>
<tr>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td><strong>characters</strong></td>
<td><strong>characters</strong></td>
</tr>
<tr>
<td>char</td>
<td>char</td>
</tr>
<tr>
<td><strong>boolean</strong></td>
<td><strong>boolean</strong></td>
</tr>
<tr>
<td>boolean</td>
<td>boolean</td>
</tr>
</tbody>
</table>

### Java Primitives Sizes

<table>
<thead>
<tr>
<th>type</th>
<th>size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>1</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>long</td>
<td>8</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>char</td>
<td>2</td>
</tr>
<tr>
<td>boolean</td>
<td>1</td>
</tr>
</tbody>
</table>

### Why Care About Size?

For example, if Java's `byte` and `long` can both represent integers, why use one instead of the other?

*large size* > values you can represent  
*small size* < memory used

More on this later
C Types Sizes

<table>
<thead>
<tr>
<th>type</th>
<th>size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1</td>
</tr>
<tr>
<td>short</td>
<td>?</td>
</tr>
<tr>
<td>int</td>
<td>?</td>
</tr>
<tr>
<td>long</td>
<td>?</td>
</tr>
<tr>
<td>float</td>
<td>?</td>
</tr>
<tr>
<td>double</td>
<td>?</td>
</tr>
</tbody>
</table>

Printing a String

```c
#include <stdio.h>

int main(int argc, char **argv) {
    printf("Welcome to CIS 2107\n");
    return 0;
}
```

Format Strings

```c
#include <stdio.h>

int main(int argc, char **argv) {
    int x=10;
    printf("x is %d\n", x);
    return 0;
}
```

Output

```
x is 10
```

Format Strings

```c
#include <stdio.h>

int main(int argc, char **argv) {
    int x=10;
    printf("x is %d\n", x);
    return 0;
}
```

placeholder
Format Strings

```c
#include <stdio.h>

int main(int argc, char **argv) {
  int x=10;
  printf("x is %d\n", x);
  return 0;
}
```

Output

x is 10

Format Strings. Multiple Placeholders

```c
#include <stdio.h>

int main(int argc, char **argv) {
  int x=10, y=20, z=30;
  printf("x is %d, y=%d, z=%d\n", x, y, z);
  return 0;
}
```

Output

x is 10, y is 20, z is 30

Format Strings. Different Formats

```c
#include <stdio.h>

int main(int argc, char **argv) {
  int x=10, y=20, z=30;
  printf("x is %x, y=%x, z=%x\n", x, y, z);
  return 0;
}
```

Output

x is a, y=14, z=1e
Format Strings. Different Formats

```c
#include <stdio.h>

int main(int argc, char **argv) {
    int x=10, y=20, z=30;
    printf("x is %x, y=%x, z=%x\n", x, y, z);
    return 0;
}
```

Output
```
x is a, y=14, z=1e
```

What Was the Deal with char Again?

```c
#include <stdio.h>

int main(int argc, char **argv) {
    char x=65;
    x++;
    printf("x is %d\n", x);
    return 0;
}
```

Output
```
x is 66
```

Can Also Be Used to Represent Characters

```c
#include <stdio.h>

int main(int argc, char **argv) {
    char x='A';
    printf("x is %c\n", x);
    return 0;
}
```

Output
```
x is A
```
Can Also Be Used to Represent Characters

```c
#include <stdio.h>

int main(int argc, char **argv) {
    char x='A';

    printf("x is %c\n", x);

    return 0;
}
```

**Output**
x is A

### if

```java
class Example {
    public static void main(String[] args) {
        if (some condition)
            statement;
        else
            statement;
    }
}
```

```c
class Example {
    public static void main(String[] args) {
        if (some condition)
            statement;
        else
            statement;
    }
}
```

### if-else

```java
class Example {
    public static void main(String[] args) {
        if (some condition)
            statement_1;
        else
            statement_2;
        if (some other condition) {
            statement_1;
            statement_2;
            ...;
            statement_n;
        }
    }
}
```

```c
class Example {
    public static void main(String[] args) {
        if (some condition)
            statement_1;
        else
            statement_2;
        if (some other condition) {
            statement_1;
            statement_2;
            ...;
            statement_n;
        }
    }
}
```

### for loop

**Also the same as Java**

```c
def for (initial condition; test; update) {
    statement_1;
    statement_2;
    ...;
    statement_n;
}
```

```java
def for (initial condition; test; update) {
    statement_1;
    statement_2;
    ...;
    statement_n;
}
```
for loop

Also the same as Java
generic

1  for (initial condition; test; update)
2       statement;

example

1  for (i=0; i<LEN; i++)
2       A[i] += 2;

another example

1  for (i=0, j=LEN-1; i<j; i++, j--)
2       swap(A, i, j);

Loop Control Variables

Java

```java
public static void main(String args[]) {
    for (int i=0; i<MAX; i++) {
    }
    for (int i=MAX; i>0; i--) {
    }
    for (int i=0; i<thresh; i++) {
    }
}
```

C

```c
int main(int argc, char **argv) {
    int i;
    for (i=0; i<MAX; i++) {
    }
    for (i=MAX; i>0; i--) {
    }
    for (i=0; i<thresh; i++) {
    }
}
```

<table>
<thead>
<tr>
<th>expression</th>
<th>Java result</th>
<th>C result</th>
</tr>
</thead>
<tbody>
<tr>
<td>10&lt;20</td>
<td>true</td>
<td>1</td>
</tr>
<tr>
<td>10&gt;20</td>
<td>false</td>
<td>0</td>
</tr>
</tbody>
</table>
means you can have things like

```c
if (1) {
    /* always runs */
}

if (0) {
    /* never runs */
}
```

```c
int i=50;
while (i) {
    i--;
}
```

but also means...

```c
if (x=y) {
    printf("equal\n");
} else {
    printf("not equal\n");
}
```

```c
int x=10, y=20;
if (x=y) {
    printf("equal\n");
} else {
    printf("not equal\n");
}
```

Output

equal

but also means...

```c
printf("x=%d, y=%d\n", x, y);
```
so then you get confused, angry and add

```
int x=10, y=20;
if (x=y) {
  printf("equal
");
} else {
  printf("not equal
");
}
printf("x=%d, y=%d
", x, y);
```

Output
equal
x=20, y=20

What's Happening?
▶ assignment then test
▶ no compiler error

Things that are almost completely the same in C and Java

▶ if, if-else
▶ for, while, do-while
▶ switch though not Strings.
▶ operators +, -, mostly
▶ comments: mostly
  ▶ /* supported everywhere */
  ▶ // mostly supported

Arrays
OK
▶ int A[5];
▶ int A[]={10,20,30,40,50};
Arrays

OK
- int A[5];
- int A[] = {10, 20, 30, 40, 50};

Not OK
- int A[];
- int []A = {10, 20, 30, 40, 50};

Legal but will get you in trouble
- A[-3] = 5;

Arrays

What do we pass when we pass an array in Java?
- size
  - no .length field
  - pass length with array