Some C

August 30, 2016

C and Java

some big differences

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

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;
}
```

A Very Simple Program

Java

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

C

```c
#include <stdio.h>

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

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

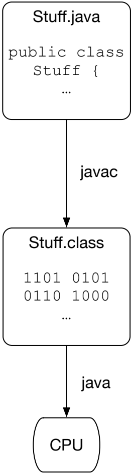
```
#include <stdio.h>

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 Java



Stuff.java

```
public class
Stuff {
...
```

javac

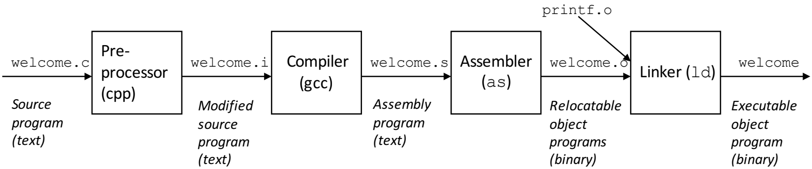
Stuff.class

```
1101 0101
0110 1000
...
```

java

CPU

Compiling and Running in C



. . . but it's really not that bad

formal

```
gcc -o executable_file_name source_file_name
```

. . . but it's really not that bad

formal

```
gcc -o executable_file_name source_file_name
```

example

```
gcc -o stuff stuff.c
```

. . . but it's really not that bad

formal

```
gcc -o executable_file_name source_file_name
```

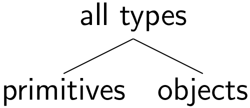
example

```
gcc -o stuff stuff.c
```

and to execute

```
./stuff
```

Java Data Types



Java Primitives

integer types
byte
short
int
long
floating-point types
float
double
characters
char
boolean
boolean

Java Primitives vs. C Types

Java

integer types
byte
short
int
long
floating-point types
float
double
characters
char
boolean
boolean

C

integer types
`char`
short
int
long
floating-point types
float
double
characters
char
boolean
any integer type

Java Primitives Sizes

type	**size (bytes)**
byte	1
short	2
int	4
long	8
float	4
double	8
char	2
boolean	1 *bit*

Why Care About Size?

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

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

type	**size (bytes)**
char	1
short	?
int	?
long	?
float	?
double	?

Printing a String

```
#include <stdio.h>

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

  return 0;
}
```

Format Strings

```
#include <stdio.h>

int main(int argc, char **argv) {
  int x=10;

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

  return 0;
}
```

Format Strings

```
#include <stdio.h>

int main(int argc, char **argv) {
  int x=10;

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

  return 0;
}
```

placeholder

Format Strings

```
#include <stdio.h>

int main(int argc, char **argv) {
  int x=10;

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

  return 0;
}
```

placeholder

Output

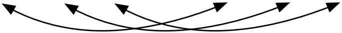
```
x is 10
```

Format Strings. Multiple Placeholders

```
#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;
}
```

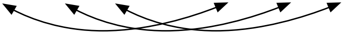
Format Strings. Multiple Placeholders

```
#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

```
#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;
}
```

Format Strings. Different Formats

```
#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?

```
#include <stdio.h>

int main(int argc, char **argv) {
  char x=65;

  x++;
  printf("x is %d\n", x);

  return 0;
}
```

What Was the Deal with char Again?

```
#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

```
#include <stdio.h>

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

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

  return 0;
}
```

Can Also Be Used to Represent Characters

```
#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

```
if (some condition)
  statement;

if (some other condition) {
  statement_1;
  statement_2;
  ...
  statement_n;
}
```

C

```
if (some condition)
  statement;

if (some other condition) {
  statement_1;
  statement_2;
  ...
  statement_n;
}
```

if-else

Java

```
if (some condition)
  statement_1;
else
  statement_2;

if (some condition) {
  statement_1;
  statement_2;
  ...
} else {
  statement_3;
  statement_4;
  ...
}
```

C

```
if (some condition)
  statement_1;
else
  statement_2;

if (some condition) {
  statement_1;
  statement_2;
  ...
} else {
  statement_3;
  statement_4;
  ...
}
```

for loop

Also the same as Java

generic

```
for (initial condition; test; update)
  statement;
```

`for loop`

Also the same as Java

generic

```
for (initial condition; test; update)
  statement;
```

example

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

for loop

Also the same as Java

generic

```
for (initial condition; test; update)
  statement;
```

example

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

another example

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

Loop Control Variables

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

```
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++) {

  }

}
```

boolean

expression	Java result	C result
10<20	true	1
10>20	false	0

means you can have things like

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

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

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

but also means . . .

```c
int x=10, y=20;

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

but also means . . .

```
int x=10, y=20;

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

Output

```
equal
```

so then you get confused, angry and add

```c
int x=10, y=20;

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

printf("x=%d, y=%d\n", x, y);
```

so then you get confused, angry and add

```c
int x=10, y=20;

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

printf("x=%d, y=%d\n", x, y);
```

Output

```
equal

x=20, y=20
```

so then you get confused, angry and add

```
int x=10, y=20;

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

printf("x=%d, y=%d\n", x, y);
```

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};`

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

- `int A[10]; A[15]=555;`
- `A[-3]=5;`

Arrays

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