Why learn C (after Java)?

- Both high-level and low-level language
- Better control of low-level mechanisms
- Performance better than Java (Unix, NT !)
- Java hides many details needed for writing OS code
  
  But,….

- Memory management responsibility
- Explicit initialization and error detection
- More room for mistakes
Simple Example

```c
#include <stdio.h>
void main(void) {
    /* print out a message */
    printf("Hello World. \n \t and you ! \n ");
    return;
}
```

- `#include <stdio.h>` = include header file `stdio.h`
  - No semicolon at end
  - Small letters only – C is case-sensitive

- `void main(void){ ... }` is the only code executed

- `printf(" /* message you want printed */ ");`

- `
  \n  \t = newline
  \t = tab`
## Simple Data Types

<table>
<thead>
<tr>
<th>Data type</th>
<th>#bytes</th>
<th>Values</th>
<th>Short-hand</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>4</td>
<td>-2,147,483,648 to 2,147,483,647</td>
<td>%d</td>
</tr>
<tr>
<td>char</td>
<td>1</td>
<td>-128 to 127</td>
<td>%c</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
<td>3.4E+/-38 (7 digits)</td>
<td>%f</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
<td>1.7E+/-308 (15 digits long)</td>
<td>%lf</td>
</tr>
<tr>
<td>long</td>
<td>4</td>
<td>-2,147,483,648 to 2,147,483,647</td>
<td>%l</td>
</tr>
<tr>
<td>Short</td>
<td>2</td>
<td>-32,768 to 32,767</td>
<td></td>
</tr>
<tr>
<td>long long int</td>
<td>8</td>
<td>Very long ;)</td>
<td>%lld</td>
</tr>
</tbody>
</table>
Like Java, like C

- Operators same as Java:
  - Arithmetic
    - `int i = i+1; i++; i--; i *= 2;`
    - `+, -, *, /, %`
  - Relational and Logical
    - `<, >, <=, >=, ==, !=`
    - `&&, ||, &, |, !`
- Syntax same as in Java:
  - `if ( ) { } else { }`
  - `while ( ) { }`
  - `do { } while ( );`
  - `for(i=1; i <= 100; i++) { }`
  - `switch ( ) {case 1: ... }`
  - `continue; break;`
Like Java, somewhat like C

- Type conversions
  - but you can typecast from any type to any type
    - `c = (char) some_int;`
  - So be careful!

- Arrays
  - Always initialize before use
    - `int number[12];`
    - `printf("%d", number[20]);`
    - produces undefined output, may terminate,
      Strings are terminated by `\0` character
    - `char name[6] = {'C','S','4','1','4','\0'};`
      /* `\0` = end of string */
    - `printf("%s", name); /* print until `\0` */`
Pointers made easy - 1

- Pointer = variable containing address of another variable

```c
float f; /* data variable */
float *f_addr; /* pointer variable */
```

- `f_addr = &f; /* & = address operator */`

```
4300
4304
```

```plaintext
f
f_addr

? ?
4300 4304

any float
any address
```

```plaintext
f
f_addr

? → 4300
4300 4304
```
Pointers made easy - 2

\[ *f\_addr = 3.2; \quad /* \text{indirection operator} */ \]

\[ f = 1.3; \]

float \( g = *f\_addr; \quad /* \text{indirection:} g \text{ is now 3.2} */ \]
#include <stdio.h>

void main(void) {
    int j;
    int *ptr;

    ptr=&j;  /* initialize ptr before using it */
    /* *ptr=4 does NOT initialize ptr */

    *ptr=4;  /* j <- 4 */

    j=*ptr;  /* j <- ??? */
}
#include <stdio.h>
void swap(int, int);
main() {
    int num1 = 5, num2 = 10;
    swap(num1, num2);
    printf("num1 = %d and num2 = %d\n", num1, num2);
}
void swap(int n1, int n2) { /* passed by value */
    int temp;

    temp = n1;
    n1 = n2;
    n2 = temp;
}
Why pointer arguments? This is why

```c
#include <stdio.h>

void swap(int *, int *);

main() {
    int num1 = 5, num2 = 10;
    swap(&num1, &num2);
    printf("num1 = %d and num2 = %d\n", num1, num2);
}

void swap(int *n1, int *n2) {
    /* passed and returned by pointer */
    int temp;

    temp = *n1;
    *n1 = *n2;
    *n2 = temp;
}
```
Structures

• Equivalent of Java’s classes with only data (no methods)

```c
#include <stdio.h>

struct birthday{
    int month;
    int day;
    int year;
};

main() {
    struct birthday mybday; /* - no 'new' needed ! */
    /* then, it's just like Java ! */
    mybday.day=1; mybday.month=1; mybday.year=1977;
    printf("I was born on \%d/\%d/\%d", birth.day, 
            birth.month, birth.year);
}
```
More on Structures

```c
struct person{
    char name[41];
    int age;
    float height;
    struct { /* embedded structure */
        int month;
        int day;
        int year;
    } birth;
};

struct person me;
me.birth.year=1977;........

struct person class[60];
/* array of info about everyone in class */
class[0].name="Gun"; class[0].birth.year=1971;......
```
enum - enumerated data types

#include <stdio.h>
enum month{
    JANUARY,    /* like #define JANUARY 0 */
    FEBRUARY,   /* like #define FEBRUARY 1 */
    MARCH       /* ... */
};

/* JANUARY is the same as month.JANUARY */

/* alternatively, ... */

enum month{
    JANUARY=1,    /* like #define JANUARY 1 */
    FEBRUARY,     /* like #define FEBRUARY 2 */
    MARCH         /* ... */
};
typedef int Employees;

Employees my_company;  /* same as int my_company; */

typedef struct person Person;

Person me;            /* same as struct person me; */

typedef struct person *Personptr;

Personptr ptrtome;    /* same as struct person *ptrtome; */

• Easier to remember
• Clean code
# Externs

```c
#include <stdio.h>

extern char user2line[20];    /* global variable defined in another file */
char user1line[30];           /* global for this file */
void dummy(void);

void main(void) {
    char user1line[20];       /* different from earlier user1line[30] */
    . . .                       /* restricted to this func */
}

void dummy() {
    extern char user1line[];  /* the global user1line[30] */
    . . .
}
```