C/C++ Basics

Basic Concepts

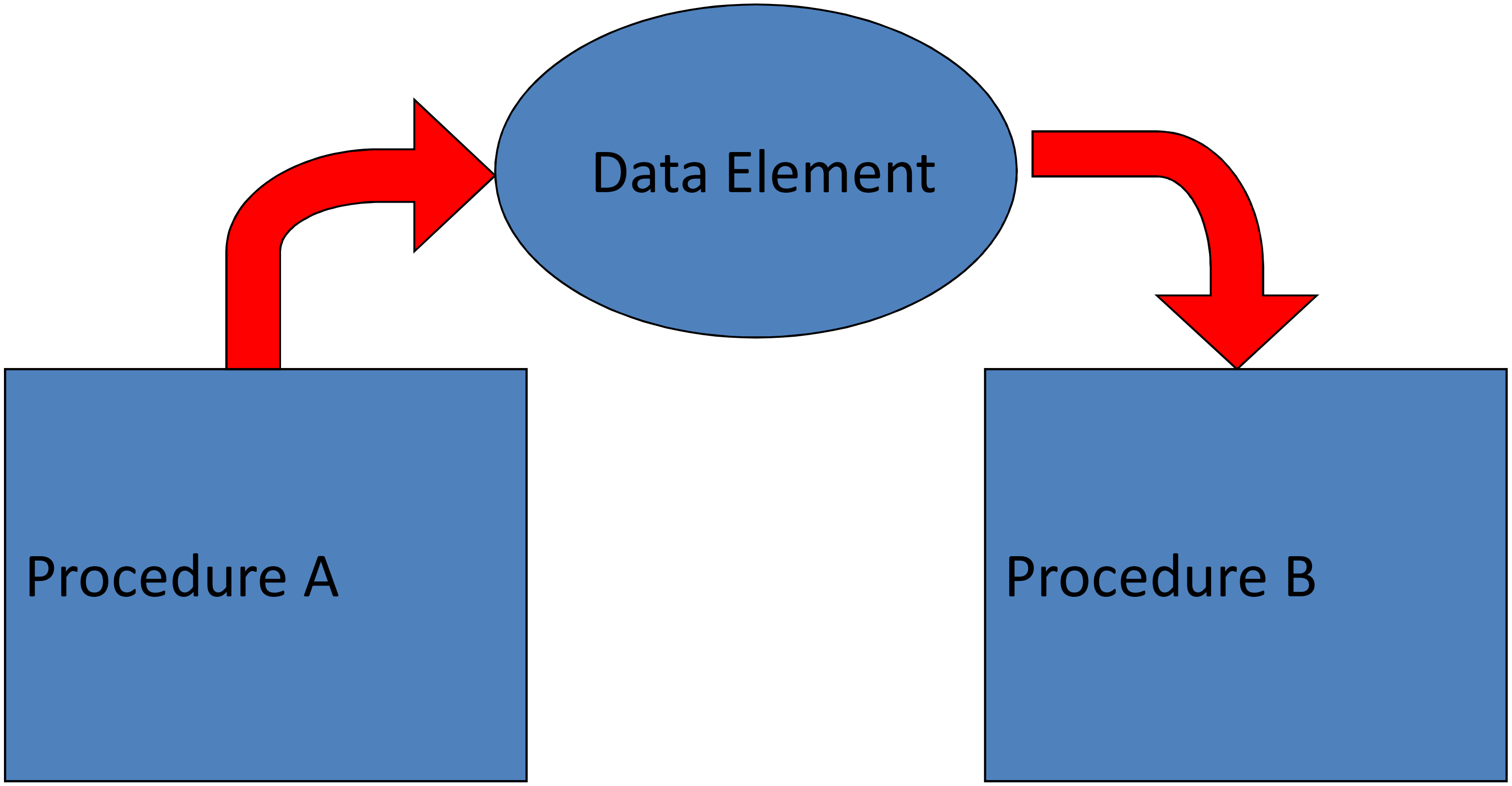
- **Basic functions of each language:** Input, output, math, decision, repetition
- **Types of errors:** Syntax errors, logic errors, runtime errors.
- Debugging
- Machine language, assembly language, high level languages

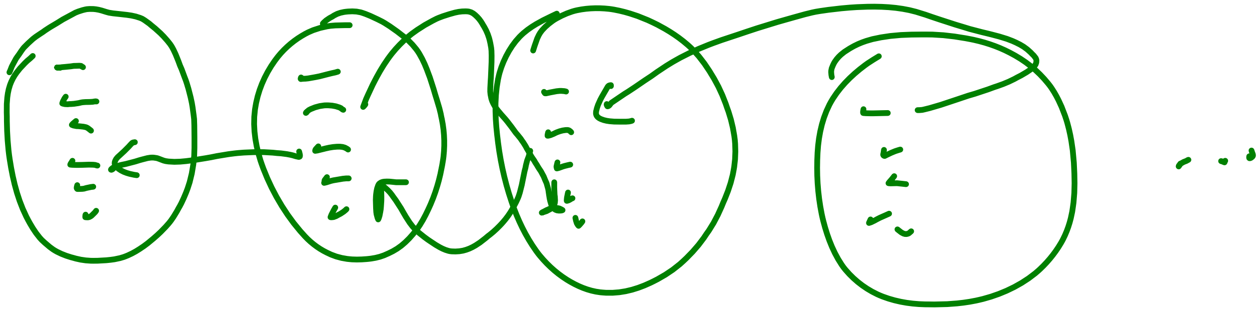
```
for(int i ; i < numbers.length; i++)
{
        numbers[i]= keyboard.nextInt();

}
```

vs.

```
for ( int item : numbers)
{
        if (item == value)


}
```

Procedural Programming



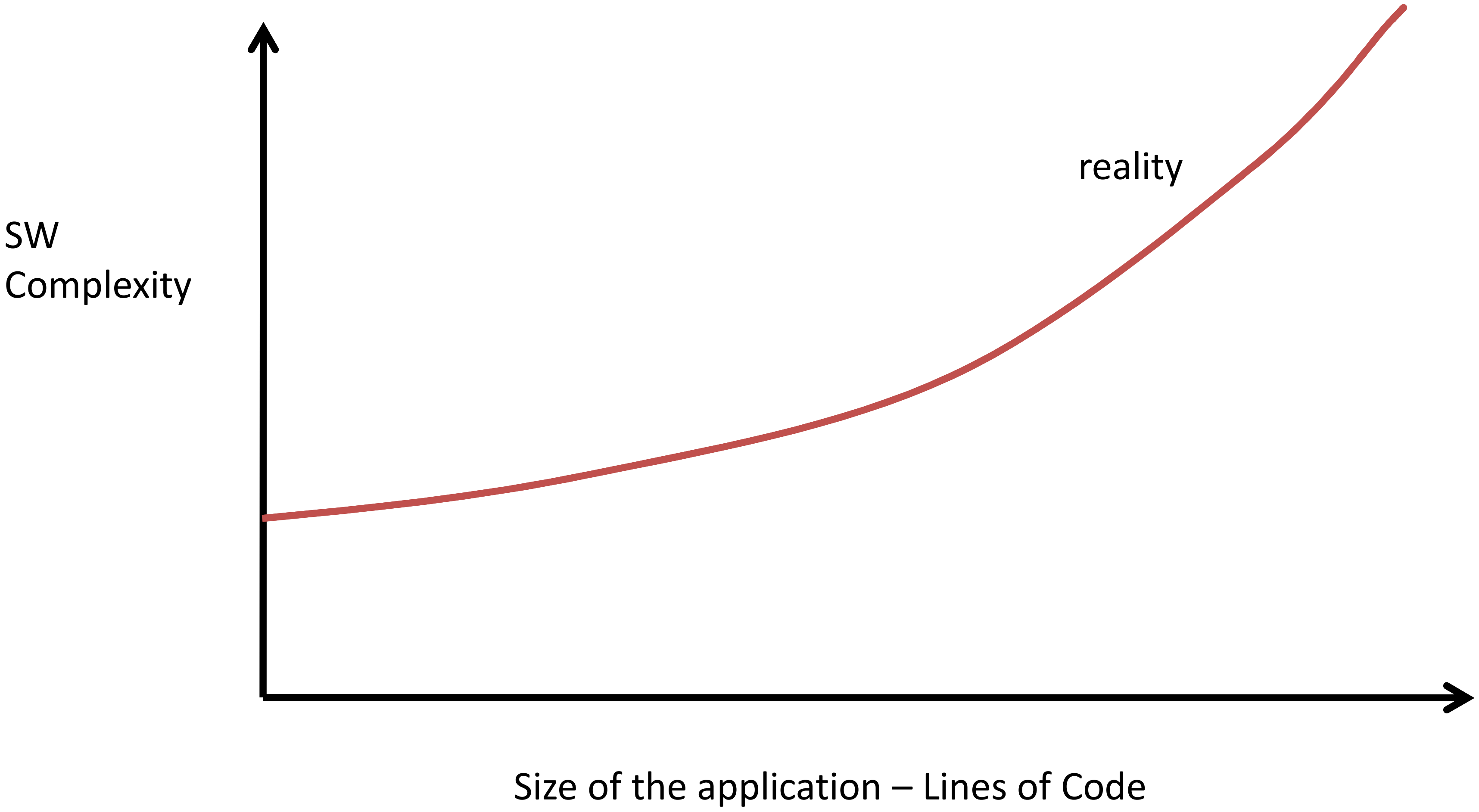


Procedural languages …

- Global variables
- Primitive data types, Arrays, Records/Structures
- lots of functions/procedures/modules
- Uniqueness of “names” requirement

Struct + protection + methods ~= class (OOP)

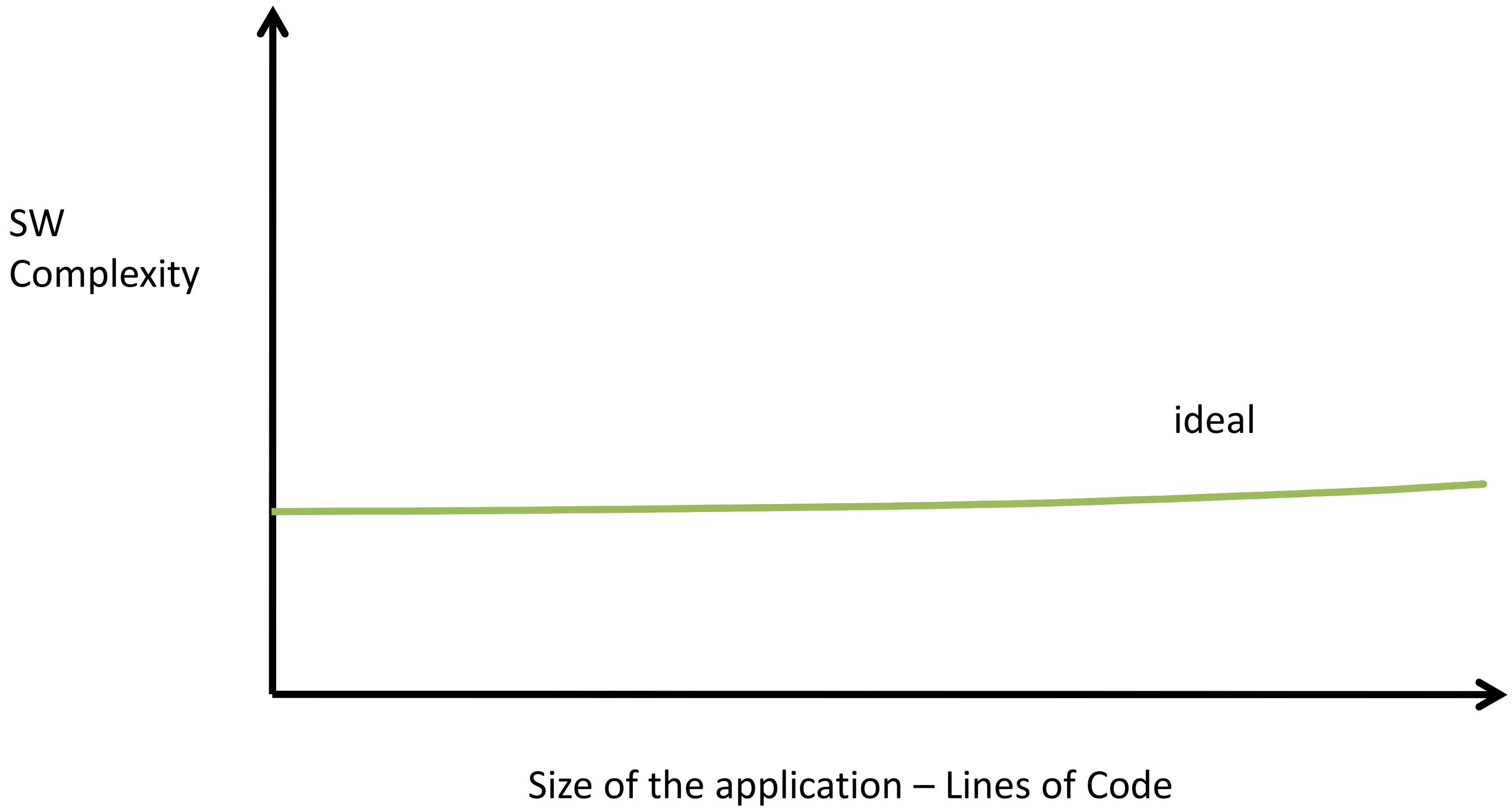
SW Complexity



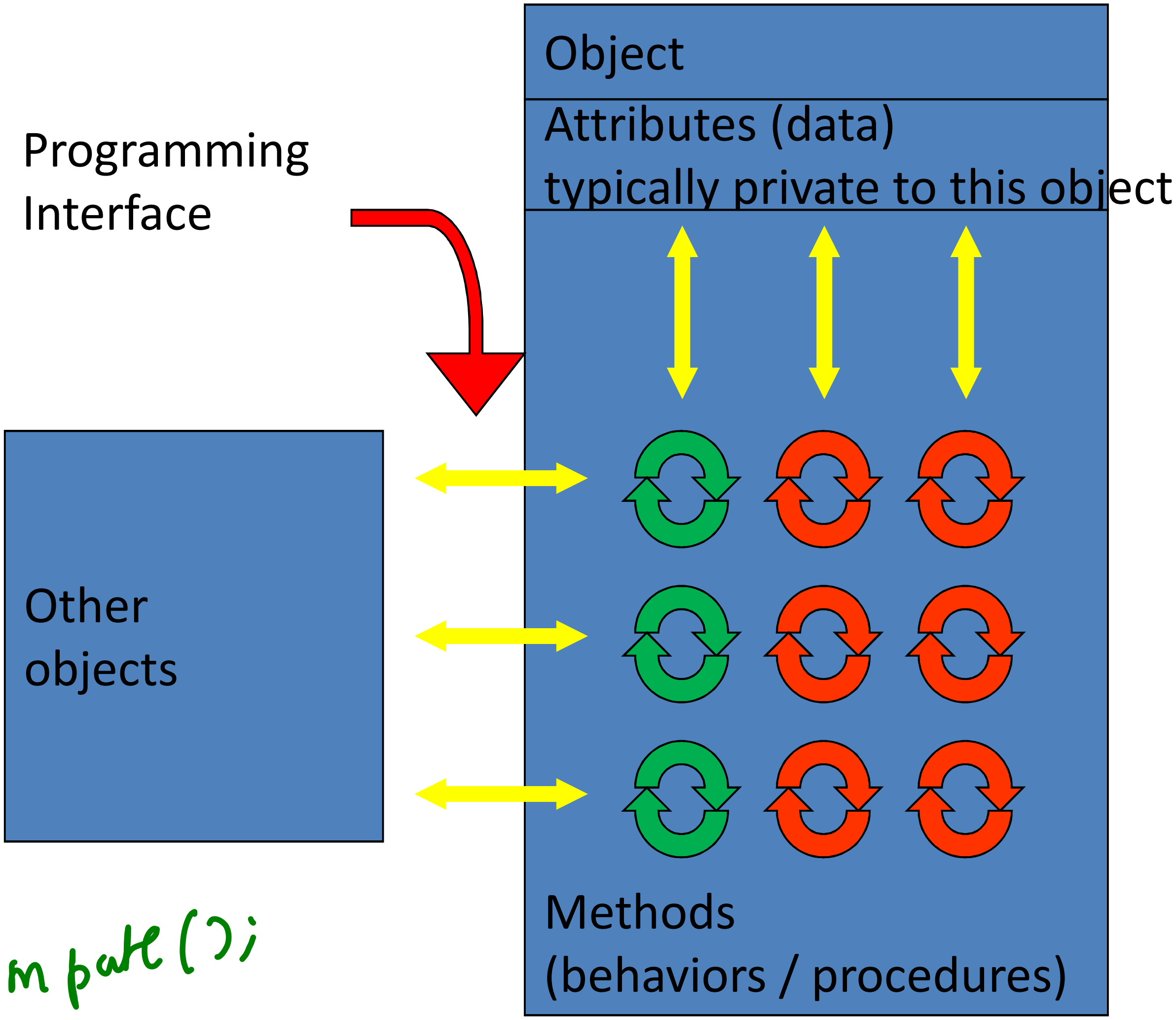
SW complexity?

- Lots of lines in code – hard to understand
- Not many meaningful comments
- Too many variables – not named well
- Complex functions – too nested, too lengthy, too many if conditions
- Inter-dependancies – changes that have to be done together in multiple files
- When you fix a bug, you make a few more.

SW Complexity



Object-Oriented Programming



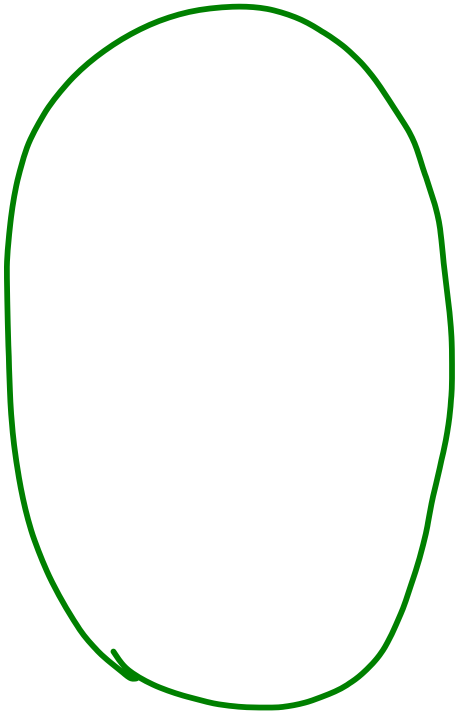
Programming Interface

Object

Attributes (data)
typically private to this object

Other objects

Methods
(behaviors / procedures)

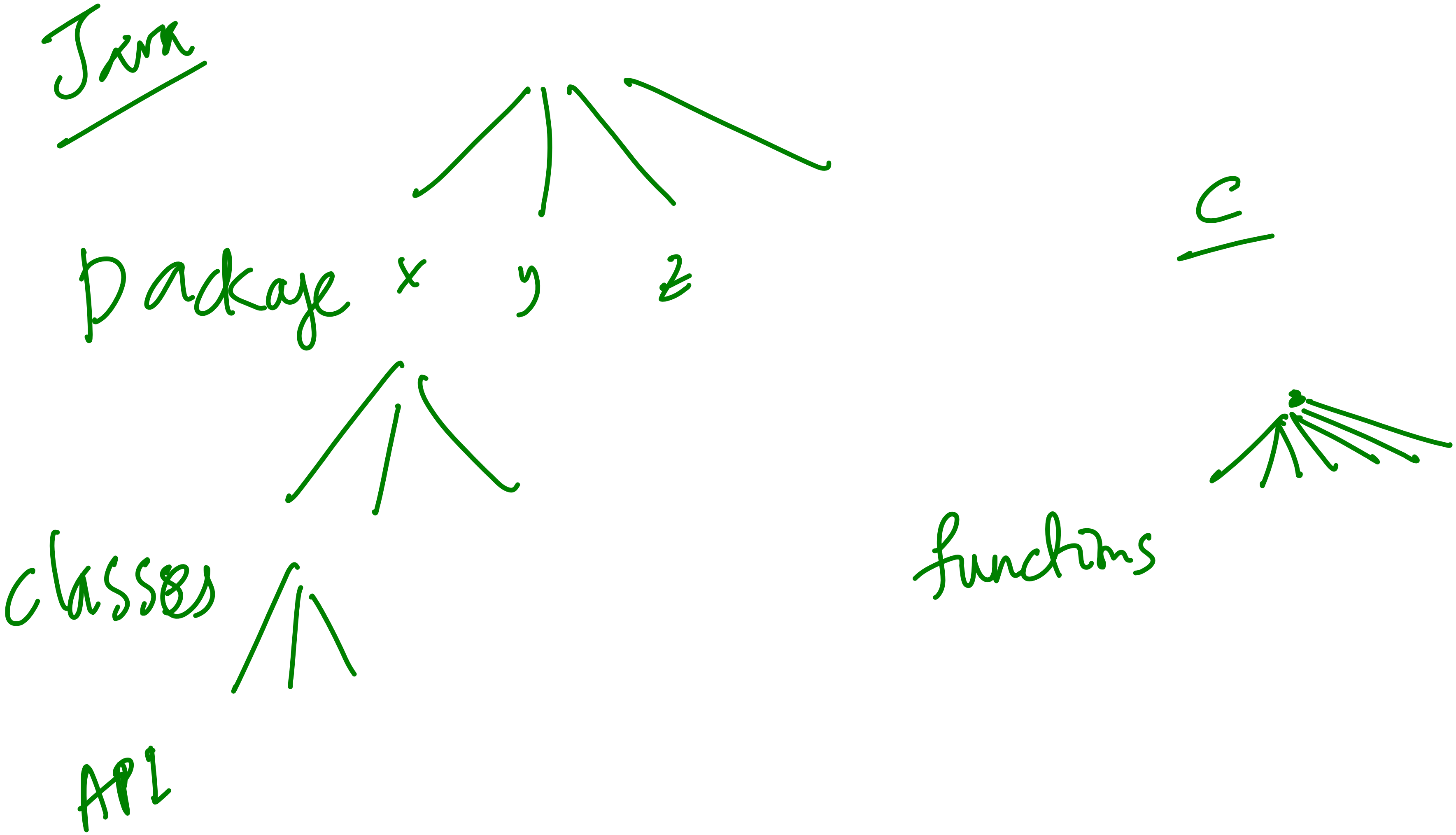


B y;

x.Compute();

y.compute();

A x;

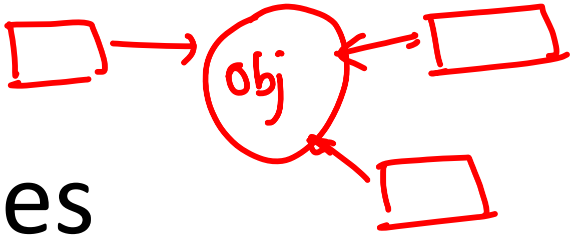

Java
package x y z
classes
API
C
functions

C vs. C++

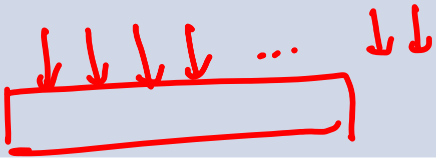
- C++ is backward compatible with C
- C++ supports OOP
- C++ standard library contains lot more functionality
- Improved I/O mechanism
- Lot of new header files
- C is very efficient, C++ is close too.
- …

C++ vs. Java: Similarities

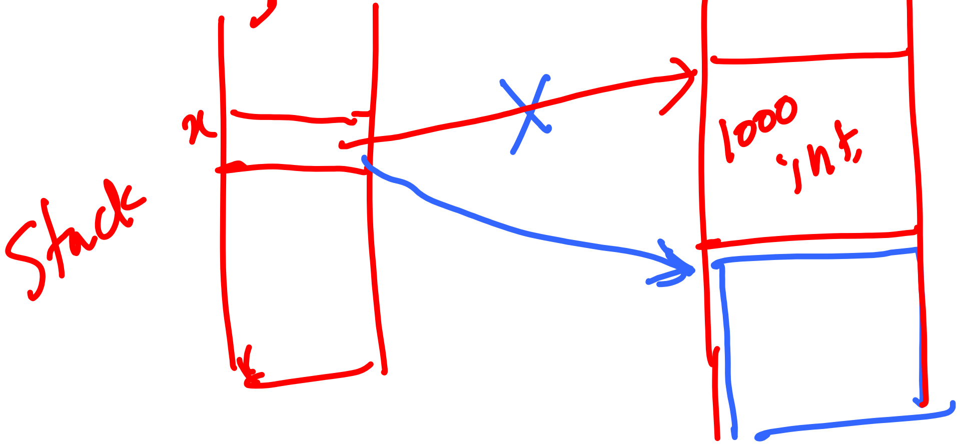
- Both support OOP. Most OOP library contents are similar, however Java continues to grow.
- Syntax is very close – Java has strong influence of C/C++. Easy to learn the other language when you know one of these.

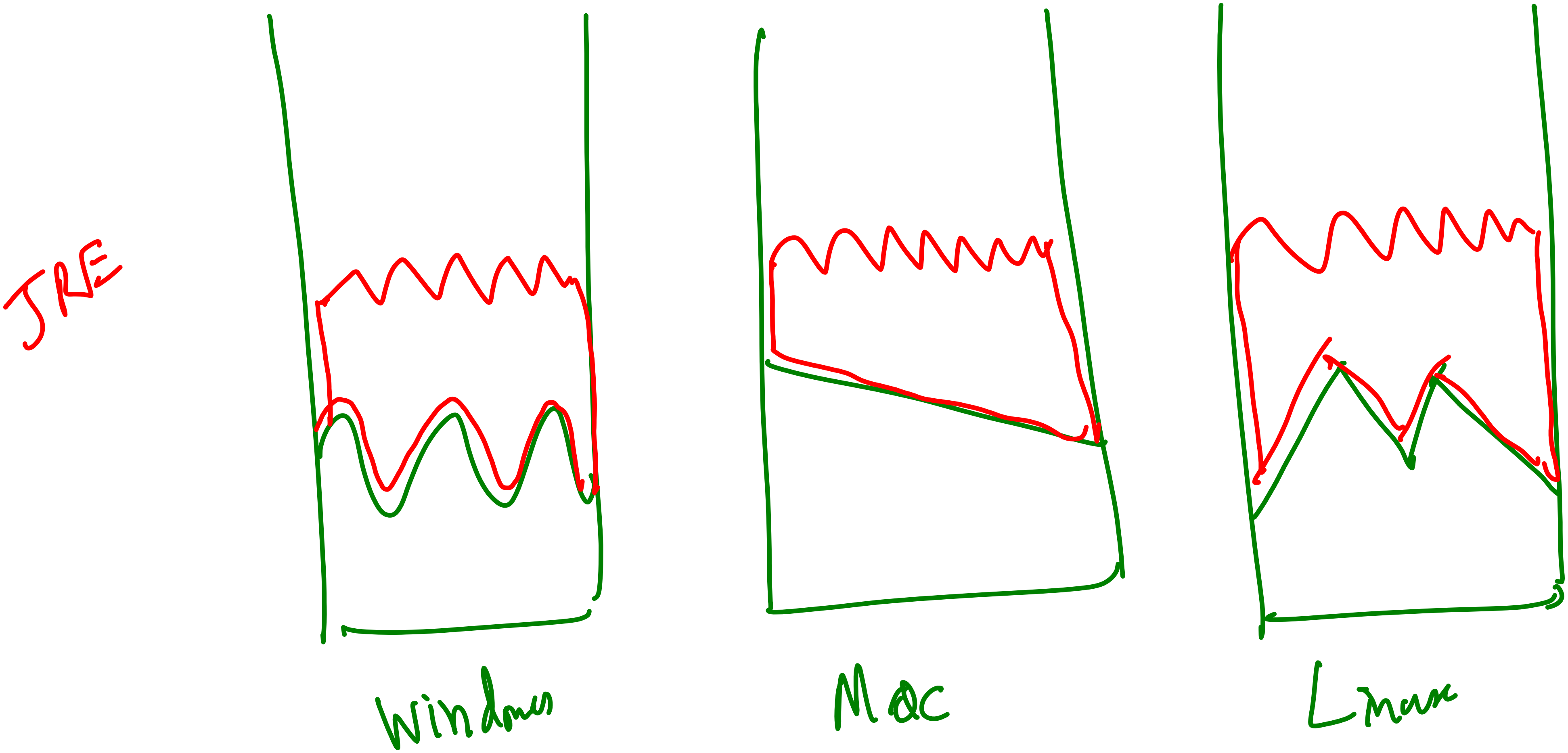


C++ vs. Java: differences

C++	Java
Write once, compile everywhere → unique executable for each target	Compile ~~Write~~ once, run anywhere → same class files will run above all target-specific JREs.
No strict relationship between class names and filenames. Typically, a header file and implementation file are used for each class.	Strict relationship is enforced, e.g. source code for class PayRoll has to be in PayRoll.java
I/O statements use cin and cout, e.g. cin >> x; cout << y; 	I/O input mechanism is bit more complex, since default mechanism reads one byte at a time (System.in). Output is easy, e.g. System.out.println(x); Scanner
Pointers, References, and pass by value are supported. **No array bound checking.**	Primitive data types always passed by value. Objects are passed by reference. Array bounds are always checked.
Explicit memory management. Supports destructors.	Automatic Garbage Collection.
Supports operator overloading.	Specifically operator overloading was left out.

```
        while(true) {
Java:       int x[] = new int[1000];
            x[i] = 1;
        }   vs.

C++:    while(true) {
            int *x = new int[1000];
            x[i] = 1;
        }
```




JRE
Windows
Mac
Linux

C standard library

- http://en.wikipedia.org/wiki/C_standard_library
- stdio.h: scanf(), printf(), getchar(), putchar(), gets(), puts(), …
- stdlib.h: atof(), atoi(), rand(), srand(), malloc(), free(), …
- math.h: sin(), cos(), sqrt(), …
- string.h: strcpy(), strcmp(), …
- ctype: isdigit(), isalpha(), tolower(), toupper()
- …

Sample C programs: prog1.c
These are in ~veerasam/students/basics

```
#include <stdio.h>

main()
{
    int in1, out;
    double in2;

    puts("Enter values:");
    scanf("%d%lf", &in1, &in2); // & means "address of"
        // %d int, %f float, %lf double
    out = in1 + in2;
    printf("Output is %d\n", out);
}
```

Sample C programs: prog2.c

```c
#include <stdio.h>
#include <stdlib.h>

main()
{
    char line[10];
    int out;

    gets(line);
    puts(line);
    out = 2 * atoi(line);
    printf("%d\n", out);
}
```

Sample C programs: prog3.c

```
#include <stdio.h>
#include <stdlib.h>

main()
{
    srand(getpid());
    printf("%d\n", rand());
    printf("%d\n", rand());
    printf("%d\n", rand());
}
```

Sample C programs: prog4.c

```c
#include <stdio.h>

int add(int x, int y)
{
    return x + y;
}

main()
{
    int in1, in2, out;

    puts("Enter values:");
    scanf("%d%d", &in1, &in2);
    out = add(in1, in2);
    printf("Output is %d\n", out);
}
```

Sample C programs: simple.c

```
#include <stdio.h>

main()
{
     int i, arr[10];

     puts("Let me init the array contents.");
     for( i=0 ; i<20 ; i++)
          arr[i] = i;
     puts("Well, I survived!");
     return 0;
}
```

Sample C programs: simple.c

```
#include <stdio.h>

main()
{
     int i, arr[10];

     puts("Let me init the array contents.");
     for( i=0 ; i<20 ; i++)
          arr[i] = i;
     puts("Well, I survived!");
     return 0;
}
```

Sample C programs: simple1.c

```
#include <stdio.h>

int i, array[10], array2[10];

main()
{
    puts("\nArray's contents are");
    for( i=0 ; i<10 ; i++)
        printf("%d\n",array[i]);
    puts("\nArray2's contents are");
    for( i=0 ; i<10 ; i++)
        printf("%d\n",array2[i]);

    puts("Let me init the array contents.");
    for( i=-10 ; i<20 ; i++)
        array[i] = i;
    puts("\nArray's contents are");
    for( i=0 ; i<10 ; i++)
        printf("%d\n",array[i]);
    puts("\nArray2's contents are");
    for( i=0 ; i<10 ; i++)
        printf("%d\n",array2[i]);
    puts("\nWell, I survived!");
    return 0;
}
```

Sample C programs: simple2.c

```
#include <stdio.h>

int i, array[10], array2[10];

printArray(int *arr)
{
    int i;

    for(i=0 ; i<10 ; i++)
        printf("%d\n",arr[i]);
}
```

```
main()
{

    puts("\nArray's contents are");
    printArray(array);
    puts("\nArray2's contents are");
    printArray(array2);

    puts("Let me init array contents.");
    for( i=-10 ; i<20 ; i++)
        array[i] = i;
    puts("\nArray's contents are");
    printArray(array);
    puts("\nArray2's contents are");
    printArray(array2);
    puts("\nWell, I survived!");
    return 0;
}
```

Sample C programs: simple3.c

```
#include <stdio.h>

printArray(int *arr)
{
    int i;

    for(i=0 ; i<10 ; i++)
        printf("%d\n",arr[i]);
}
```

```
main()
{
    int i, array[10], array2[10];

    puts("\nArray's contents are");
    printArray(array);
    puts("\nArray2's contents are");
    printArray(array2);

    puts("Let me init array contents.");
    for( i=-10 ; i<20 ; i++)
        array[i] = i;
    puts("\nArray's contents are");
    printArray(array);
    puts("\nArray2's contents are");
    printArray(array2);
    puts("\nWell, I survived!");
    return 0;
}
```

Sample C programs: simple4.c

```
#include <stdio.h>

printArray(int *arr)
{
    int i;

    for(i=0 ; i<10 ; i++)
        printf("%d\n",arr[i]);
}

main()
{
    second();
    puts("\nWell, I survived!");
    return 0;
}
```

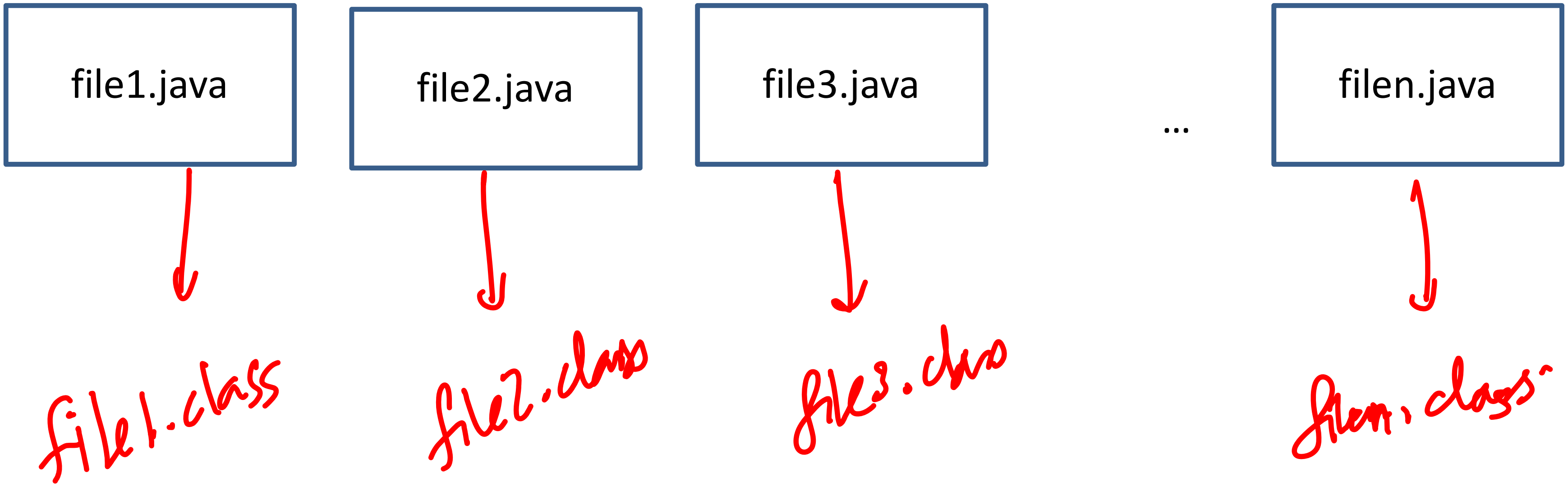
```
second()
{
    int i, array[10], array2[10];

    puts("\nArray's contents are");
    printArray(array);
    puts("\nArray2's contents are");
    printArray(array2);

    puts("Let me init array contents.");
    for( i=-10 ; i<20 ; i++)
        array[i] = i;
    puts("\nArray's contents are");
    printArray(array);
    puts("\nArray2's contents are");
    printArray(array2);
}
```

Things to observe

- Memory allocation for arrays
- Array length is not stored with arrays
- Potentional issues with scanf() and printf()
- Different behavior when overshooting arrays in heap vs. in stack

Sample C program: multiple files

Compare with Java



file1.java

file2.java

file3.java

…

filen.java

file1.class

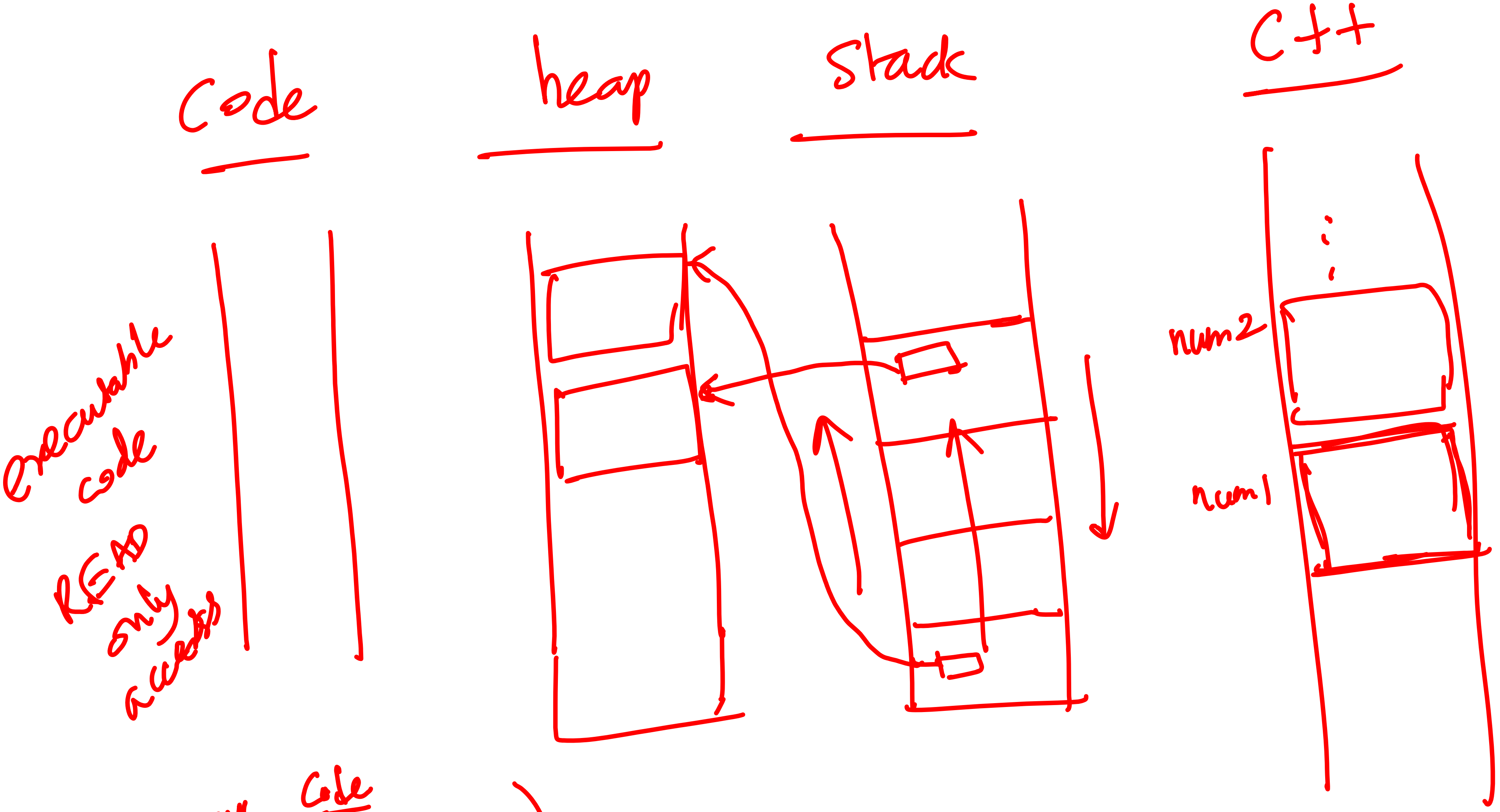
file2.class

file3.class

filen.class

Class vs. Object

Types of memory



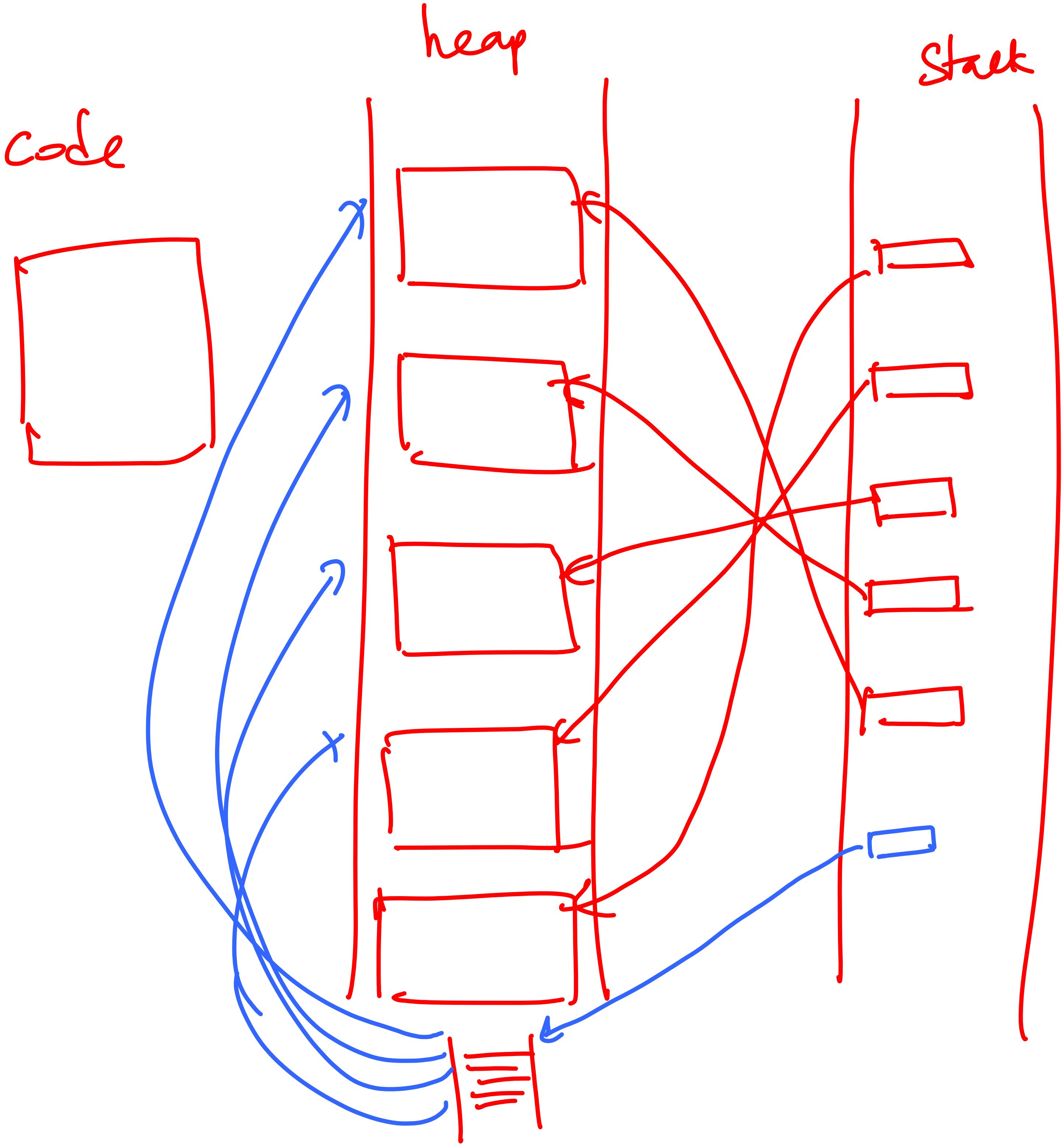
complexType code 150 bytes

data ~~8~~ bytes

16

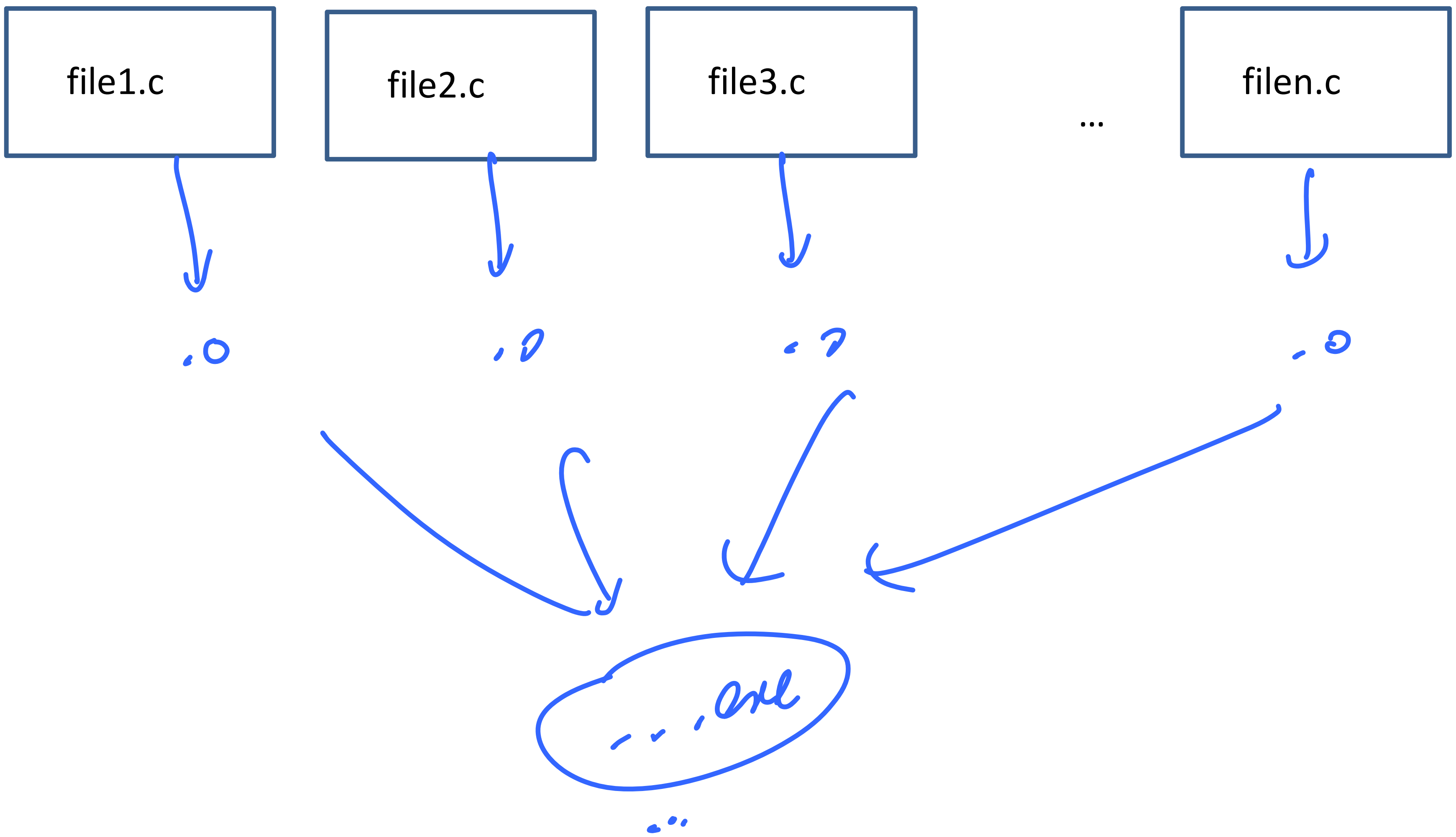
→ 100 objects

Total memory = $\underbrace{16\times100}_{heap} + \underbrace{150}_{code} + \underbrace{100\times8}_{stack}$



code
heap
Stack

C files & runtime environment



file1.c

file2.c

file3.c

…

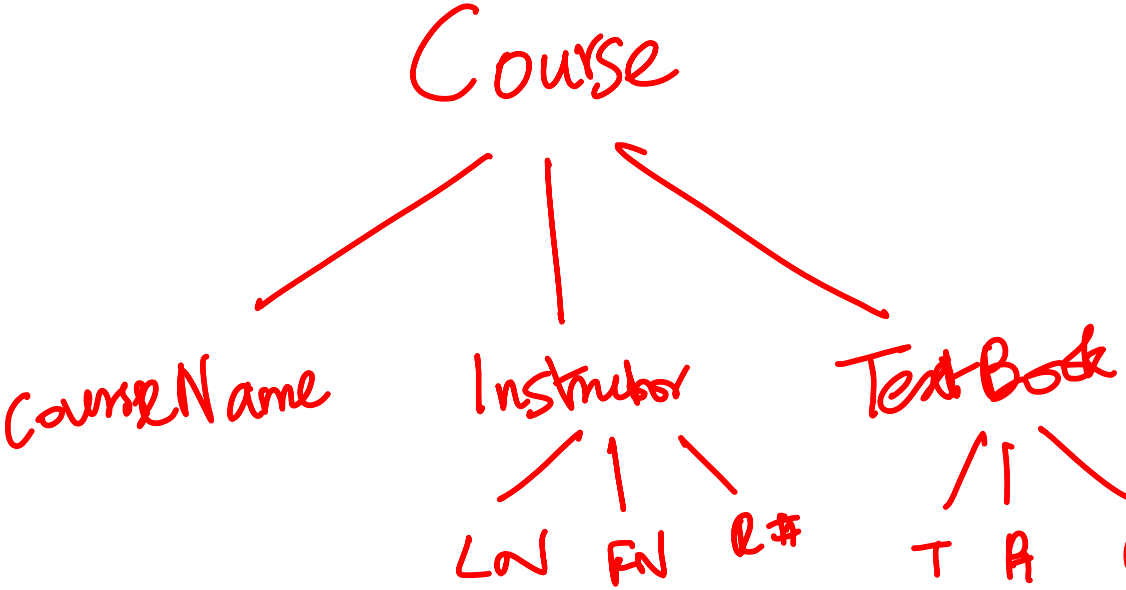
filen.c

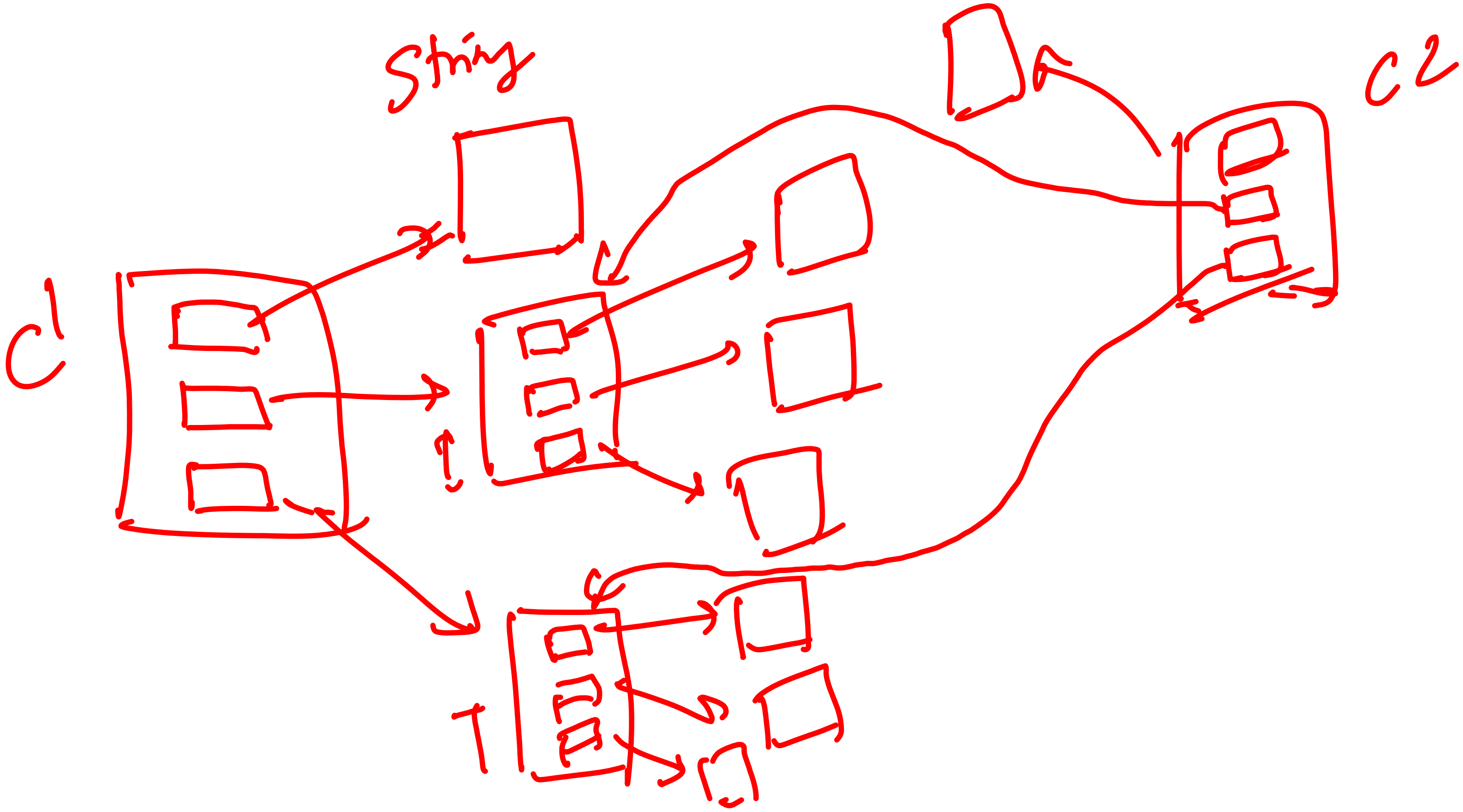
Debugging : gdb

Compile with –g option to use the debugger.

Most used commands:

- run
- list [method name]
- break [line_number]
- step
- next
- continue
- print [variable]


Course
CourseName
Instructor
TextBook
LN
FN
R#
T
A
P


String
C1
C2
I
T