

C++ vs. Java

control flow syntax mostly same as Java 1.4

if (x=7) OK in C++

int type cast to bool automatically ($\neq 0 = \text{true}$
 $0 = \text{false}$)

primitive types same (int, double, bool, char, float)

↓
Java: 32 bits

C++: platform specific

↓
C++: 8 bits

w_char: 16 bits

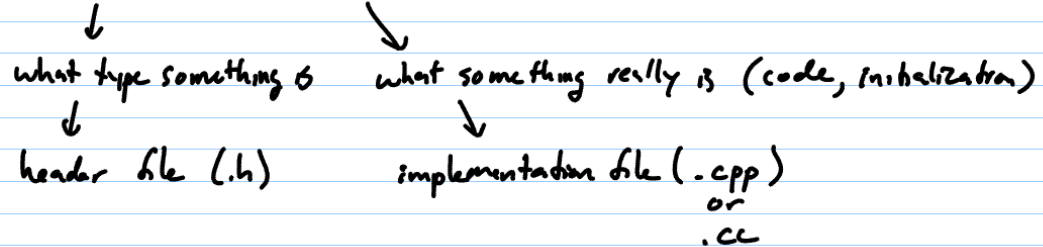
signed/unsigned can be applied to integral types

unsigned int x; x is between 0... 4 billion (assuming
32-bit ints)

organization

can have fns outside classes (Ex: main)

split declaration/definition in 2 files



to use a class / group of related global fns

include for header file (also in implementation file)

specify all implementation files when compiling

```
public class X { public void foo() { bar(); }  
                public void bar() { foo(); } }
```

```
header: class X
{
public:
    void foo();
    void bar();
};
```

```
public class t {
    ...
    X x = new X();
    x.foo();
}
```

```
implementation: #include "x.h"
int X::foo()
{
    bar();
}
int X::bar()
{
    foo();
}
```

```
use: #include "x.h"
X x;
```

```
x.foo();  
;  
;
```

arrays

allocated statically or dynamically

```
int arr[5][3];
```

5 row, 3 col
array of ints

arr[0][0] is upper left of array delete[] arr;

no length field!
(need separate variables)

```
int total_array(int x[], int size)
```

no bounds checks!

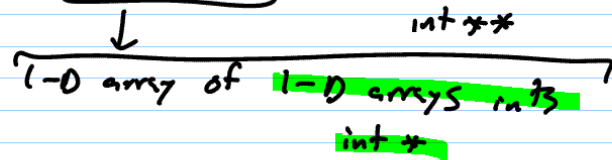
type for array =
element-type *

↓ ↓

```
int *arr;  
arr = new int[5];
```

```
arr[0] = 4;
```

dynamically allocated 2-D array



```
int ** arr2D;
```

```
arr2D = new int*[5];
```

↳ new array of 5 arrays of ints

```
for (int r = 0; r < 5; r++)
```

```
    arr2D[r] = new int[3];
```

```
    arr2D[0][2] = 5;
```

```
    ;
```

```
for (int r = 0; r < 5; r++)
```

```
delete[] arr2D[r];
```

```
delete[] arr2D;
```

strings : C vs C++

array of chars
w/ null (ASCII 0)
at the end

```
#include <cstring>
```

```
strcpy(dest, src)
```

```
char s[] = "hello";
```

```
char copy[6];
```

```
strcpy(copy, s);
```

```
copy[0] = 'j';
```

```
std::cout << copy << std::endl;
```

objects

```
#include <string>
```

```
std::string s = "hello";
```

```
std::cout << s << std::endl;
```

```
s[0] = 'j';
```

STL (Standard Template Library) \approx JCF (Java Collections Framework)

```
#include <vector>
```

```
⋮
```

```
std::vector<std::string> names;
```

add \rightarrow push-back

```
names.push-back("Jim Glenn");
```

```
names.push-back("Brady Anderson");
```

get \rightarrow []

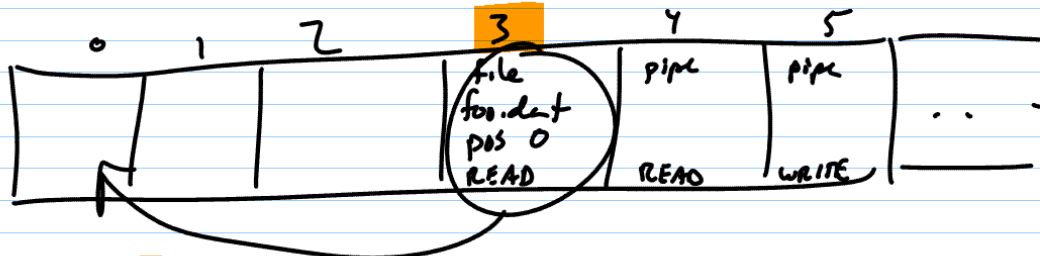
```
std::cout << names[0];
```

```
cat scores | grep Villanova | wc
```

```
commands[0] commands[1][0] commands[2]
```

commands [1][1]

File Descriptor Table



```
int fd = open("foo.dat", O_RDONLY);
```

0 = standard input (System.in, std::cin, stdin)

1 = standard output (System.out, std::cout, stdout)

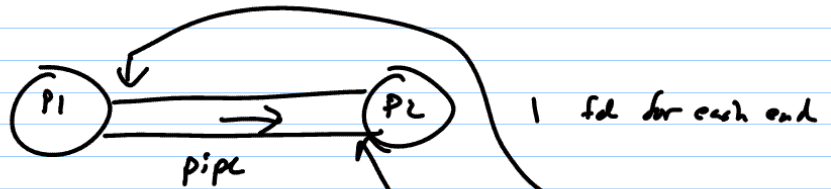
libraries translate System.out.println("...")
or std::cout << "..."

to write(1, "...")

dup2 copies FDT entries

```
dup2(fd, 0);
```

now read(0, ...) reads from file, not keyboard



```
int pipe_fd[2];
```

```
pipe(pipe_fd)
```

pipe fd

4	5
---	---

P1 writes to pipe_fd[1]

P2 reads that data from pipe_fd[0]