

Advanced Topics

01 - C++ Intro

Goals

- Be familiar with C++ main syntax
- Know the differences from Java and the relevant substitutes
- Be able to program in C++, avoiding C++ common bugs and mistakes

About me

Lecturer

Academic College of Tel-Aviv-Yaffo
Tel-Aviv University

C++ Development Roles

Member of the Israeli ISO C++ NB

Co-Organizer of the **CoreCpp**
conference and meetup group



Why C++

- Direct Memory Access: Drivers, Low Level
- Performance:
 - Throughput
 - Latency
 - Deterministic behavior (no GC)
- OS Specific needs
- (Legacy Code)

(Why Not?)

- Memory Management (it's a Plus and Minus)
- Lack of standard libraries (it's changing!)
- Compilation per OS (again, Plus and Minus)
- Less "friendly"

C++ Usage Examples

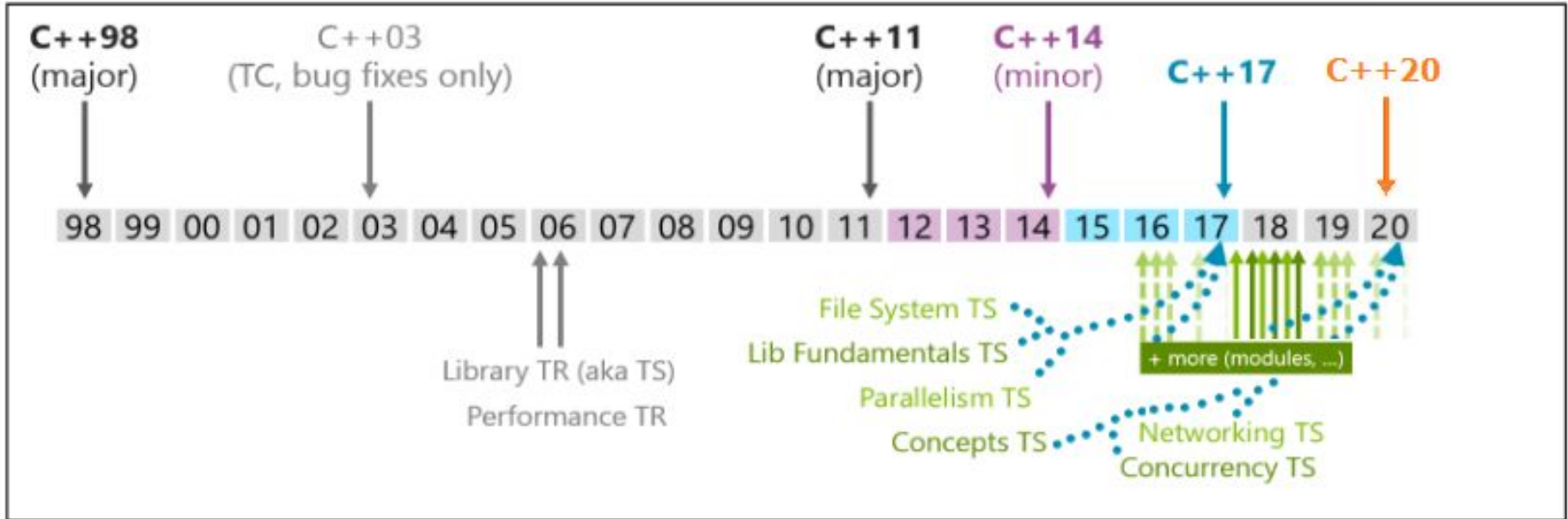
Games and 3D engines, Multimedia Engines
(audio and video streaming, image processing)

Networking and Telecom (Switches, **Routers**,
SIP servers, **Firewalls**, Load Balancers)

Military Systems (e.g. **Iron Dome**), Avionics,
Automotive, Trading, Embedded Software,

Compilers (see: <http://www.lextrait.com/vincent/implementations.html>)

C++ History and Future (old chart)



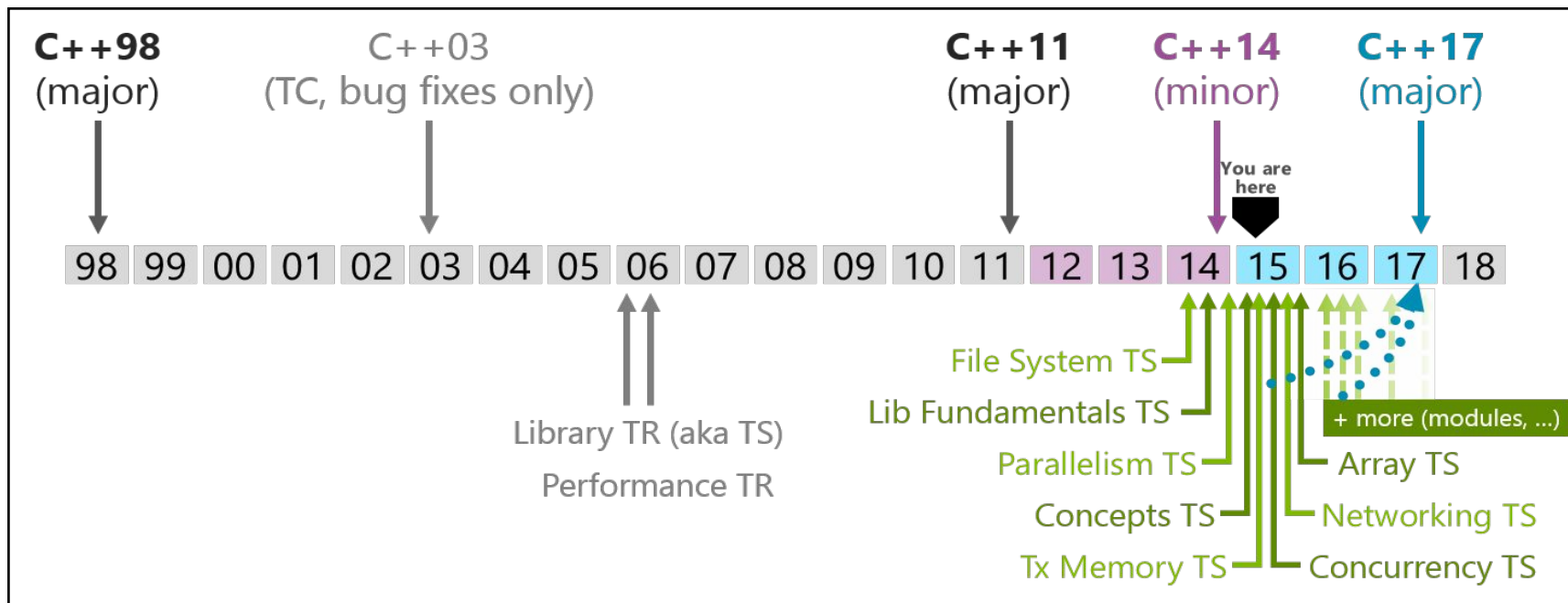
see: <https://isocpp.org/std/status>

TS = Technical Specification

TR = Technical Report

TC = Technical Committee

C++11 History (older chart)



source: <https://isocpp.org/std/status>

TS = Technical Specification

TR = Technical Report

TC = Technical Committee

Let's Begin

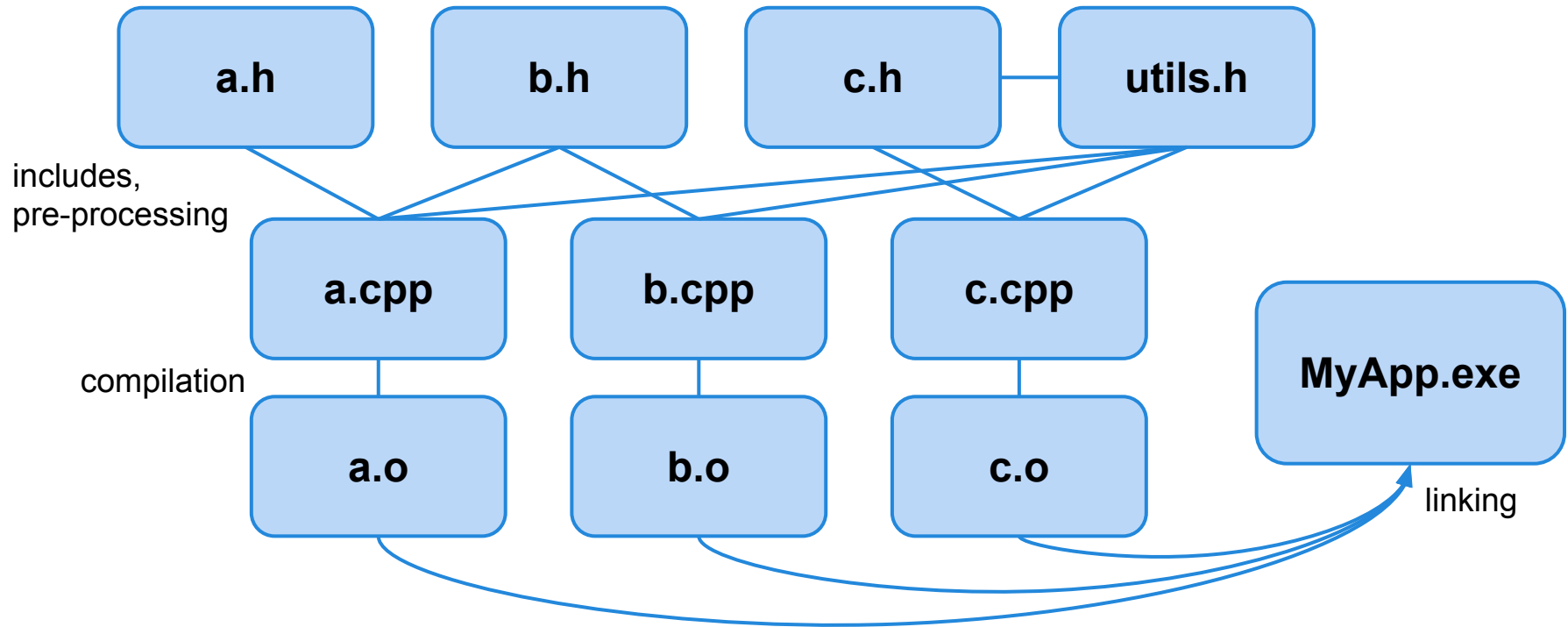
Compilation Process, Basic Syntax,
Pointers, References, new, delete, Overloading,
Classes, constructors, destructors, Init List,
copy constructor, operators, static, const, mutable,
explicit, **Inheritance, Polymorphism,**
Templates ...

Our next item... Compilation

⇒ .h and .cpp files

⇒ compilation and linking

C/C++ Compilation Process



Pre-Processing

#define

#ifdef #ifndef

#include

etc.



Always use #ifndef to prevent multiple inclusion of .h files

Compiler

⇒ source-code to binary (machine code)

⇒ auto-casting, inlining, optimizations

⇒ symbols for linker



Compilation error is better than any other error

prototypes and implicit-casting

Example 1:

```
int i = pow(3, 2);
```

Example 2:

```
int i = someGlobalVarFromAnotherFile + 1;
```

forward declaration

```
class Gardener;
```

```
class Garden {  
    Gardener* myGardener;  
};
```

```
class Garden;
```

```
class Gardener {  
    Garden* myGarden;  
};
```

Linker

- ⇒ **find symbols and embed their address**
(this includes external libraries)
- ⇒ **create the final artifact (exe, lib)**



Linker errors are nasty but they usually say that something was declared but not defined, or defined more than once

Overloading and extern "C"

- a word on Functions Overloading
- C++ name mangling
- `nm main.o | c++filt` (or: `nm main.o -C`)
- The need for extern "C"

Intermediate Summary - Compilation

- ✓ **.h and .cpp files**
- ✓ **compilation and linking**

Our next item... Memory

⇒ Stack Variables

⇒ Pointers and References

⇒ new and delete

Stack Variables - 1

```
int main() {  
    Person p;  
    p.name = "momo";  
    Person p2 = p;  
    p2.name = "koko";  
    Person p3;  
    p3 = p2;  
    p3.name = "george";  
}
```

??

What's the name of each person at the end of each line?

Stack Variables - 2

```
// a global function
// for demo only
void setName(Person p) {
    p.name = "momo";
}

int main() {
    Person p;
    setName(p);
}
```

??

Does 'p' in main get the name "momo"?

Stack Variables - 3

```
// a bad function
// for demo only
char* getName() {
    char name[10];
    cin >> name;
    return name;
}
// above is BAD
```

??

What's the problem?

Stack Variables - 4

```
// not so good
// for demo only
Person getPerson() {
    Person p;
    p.name = "momo";
    return p;
}

// was considered as
// not so efficient
// till C++11
```

??

What happens here?

Pointers - 1

```
int main() {  
    int i = 3;  
    int* ptr = &i;  
    *ptr = 5;  
    cout << i;  
}
```


Pointers - 2

```
int main() {  
    Person p;  
    Person* pPtr = &p;  
    pPtr->name = "momo";  
    cout << p.name;  
}
```

Pointers - 3

```
int main() {  
    Person p[10];  
    for(int i=0; i<10; ++i) {  
        (p+i)->age = i;  
        p[i].height = 45+i*10;  
    }  
}
```

Pointers - 4

```
int main() {  
    Person p1, p2, *ptr;  
    ptr = new Person("momo");  
    ptr->print();  
    delete ptr;  
    ptr = &p1;  
    ptr->print();  
    // delete ptr; // NO!  
    // ... =>
```

```
    // ... continues  
    ptr = &p2;  
    p2.name = "koko";  
    ptr->print();  
    p1.print();  
    ptr = new Person[10];  
    ptr[0].print();  
    delete []ptr;  
}
```

Pointers - 5

```
void setName(Person* p) {  
    p->name = "jasmin";  
}  
  
int main() {  
    Person p;  
    setName(&p);  
    printName(&p);  
}
```

```
void printName(const Person* p)  
{  
    p->print();  
}  
  
// global functions aren't nice  
// this is just for demo
```

Exercise



1. Write a "swap" method for ints using pointers.
2. Write a "swap" method for char* using pointers (i.e. char**).

References - 1

```
int main() {  
    int i = 3, k = 5;  
    int& j = i;  
    j = 12;  
    cout << i;  
    j = k;  
    k = 231;  
    cout << i;  
}
```

References - 2

```
void setName(Person& p) {  
    p.name = "jasmin";  
}  
  
int main() {  
    Person p;  
    setName(p);  
    printName(p);  
}
```

```
void printName(const Person& p)  
{  
    p.print();  
}  
  
// global functions aren't nice  
// this is just for demo
```

References - 3

```
int& find(int* arr, int size,
          int& lookFor) {
    for(int i=0; i<size; ++i) {
        if(arr[i]==lookFor) {
            return arr[i];
        }
    }
    return lookFor;
}
```

```
int main() {
    int arr[]={1,2,3};
    int num = 2;
    int size =
        sizeof(arr)/
        sizeof(arr[0]);
    find(arr, size, num)
        = 15;
}
```


Exercise



1. Write a "swap" method for ints using reference.
2. Write a "swap" method for char* using reference.

Exercise



Write a "max" method that returns the maximum value in int array, ByRef.

Call this function and use the return value as Lvalue (i.e. assign new value into it).

Exercise



Allocate array of ints with user input size, then get user input for populating the array.

Print the array and finish.

Intermediate Summary - Memory

- ✓ **Stack Variables**
- ✓ **Pointers and References**
- ✓ **new and delete**

Our next item... Basics

⇒ C++ Functions Overloading

⇒ C++ Default Parameters

⇒ C/C++ enums

⇒ C/C++ define - macros and consts

Functions Overloading

```
void drawRect(int x1, int y1, int x2, int y2);  
void drawRect(const Point& p1, const Point& p2);
```

Also:

```
T& Array::operator[] (int index);  
T Array::operator[] (int index) const;
```

But NOT:

```
void doIt(const Person& p);  
void doIt(Person& p);
```

Default Parameters

```
void message(const char* msg, const Point& p=Point(1,1));  
void message(const char* msg, int x1, int y1);
```

Note:

- Defaults must start from the end and backward
(i.e. you can't have default for the 1st parameter and not for the 2nd)

enums - 1

Just for defining constants:

```
enum{LEFT, RIGHT, UP, DOWN};  
int direction = UP;
```

Define a new type:

```
enum DayOfWeek {Sun=1, Mon, Tue, Wed, Thu, Fri, Sat};  
DayOfWeek day = Sun;
```

Or: define enum with a context...

enums - 2

enum with a context (easier to use):

```
struct DayOfWeek {  
    enum value {Sun=1, Mon, Tue, Wed, Thu, Fri, Sat};  
};
```

```
DayOfWeek::value day = DayOfWeek::Sun;
```

or, C++11:

```
enum class DayOfWeek {Sun=1, Mon, Tue, Wed, Thu, Fri, Sat};  
DayOfWeek day = DayOfWeek::Sun;
```

enums - 3

For bitwise flags:

```
struct DoIt {  
    enum DoitFlag {FO=0x01, MO=0x02, DO=0x04, JO=0x08};  
};  
  
DoIt::DoitFlag flag =  
    (DoIt::DoitFlag) (DoIt::FO | DoIt::MO);
```

#define (that's C!)

For constants:

```
#define SIZE 10
```

For simple replacement:

```
#define signals public
```

As a macro:

```
#define SQR(num) (num) * (num)
```

Intermediate Summary - Basics

- ✓ **C++ Functions Overloading**
- ✓ **C++ Default Parameters**
- ✓ **C/C++ enums**
- ✓ **C/C++ define - macros and consts**

Our next item... Classes

- ⇒ **C++ Class Syntax**
- ⇒ **C'tor, D'tor, Copy C'tor, Casting w/ C'tor**
- ⇒ **Operators (Assignment and others)**
- ⇒ **static, const, mutable, friend**
- ⇒ **Nested classes, Namespaces**

Classes - First Example

```
class Person {  
    string name;  
    const int id;  
public:  
    Person(const string& nm, int id1): name(nm), id(id1) {}  
    const string& getName()const {return name;}  
    int getId()const {return id;}  
    void setName(const string& nm) {name = nm;}  
};
```



Don't forget the semicolon!

Classes - .h and .cpp

.h file:

```
class Person {  
    string name;  
    const int id;  
public:  
    Person(const string& nm, int id1);  
    const string& getName() const;  
    int getId() const;  
    void setName(const string& nm);  
};
```

Classes - .h and .cpp

.cpp file:

```
Person::Person(const string& nm, int id1)
    : name(nm), id(id1) {}

const string& Person::getName() const {
    return name;
}

int Person::getId() const {
    return id;
}

void Person::setName(const string& nm) {
    name = nm;
}
```


Constructors

Same as Java:

- No c'tor = there is empty by default
- No empty = must pass parameters
- Can overload c'tors
- C++11: Can call other c'tor (not with “this” though, through init list call)

Not the same as Java:

- Can use default parameters
(a single c'tor can get parameters and still be empty c'tor)
- Init list - as seen in the previous example
used for initialization of members as well as base class(es)

Destructor

An important “creature” in C++!

- Takes no arguments, thus there is only one per class:
`~<ClassName>(); // e.g. for class A: ~A();`
- Guaranteed to be called immediately when object "dies"*
(if process is not terminated)
- Usually used for resource de-allocations (but can actually do anything)

* When object dies?

- Stack object - at the matching closing curly brackets
- Heap object allocated with 'new' - when deleted with 'delete'
- Global or Static object - at the end of the process
- Temporary object - by end of the line

`message("hello", Point(10,10));`

Exercise



Implement your own String class, with relevant constructor and destructor.

Make sure you allow the creation of an “empty” string, like this: `String s;`

As well as: `String s("hello");`

(What happens with: `String s="hello";` ?)

Copy C'tor

An important “creature” in C++!

- Signature:

`A::A(const A& a) ;`

- Used when creating a copy on your own
- Called automatically when passing objects of this class By Value
- If you don't implement your own - you get a “free” one automatically!

??

**What happens if this is
my copy c'tor signature:**

`A::A(A a) ;`

C'tor used for Casting

```
class A {  
    int i;  
public:  
    A(int i1):i(i1){}  
};  
  
void f(const A& a);  
  
// auto casting works  
// only for 'const ref'  
// or for byval  
// but not for byref
```

```
int main() {  
    A a1(1);  
    A a2 = 2;  
    f(A(1)); // works  
    f((A)1); // works  
    f(1);    // works!  
    a1 = 3;  // works!  
}
```

explicit

```
class A {  
    int i;  
public:  
    explicit A(int i1):i(i1){}  
};  
  
void f(const A& a);
```

```
int main() {  
    A a1(1);    // ok  
    // A a2=2;  // can't...  
    f(A(1));    // ok  
    f((A)1);    // ok  
    // f(1);    // can't...  
    // a1 = 3;   // can't...  
    a1 = A(3);  // ok  
}
```

Operators

In C++ you can teach your class to “understand” operators!

- **There are rules on how-to, what is OK and what's NOT, etc. (see for example: More Effective C++ / Scott Meyers)**
- **There are important operators for external libs (such as < for sorting)**
- **Most operators can be defined as member or as global**
- **Except for assignment (=) all others are NOT pre-defined, i.e. if you need it, you must define it (for example: there is no default ==)**
- **Assignment operator (=) is important, the default is not always good...**

Exercise



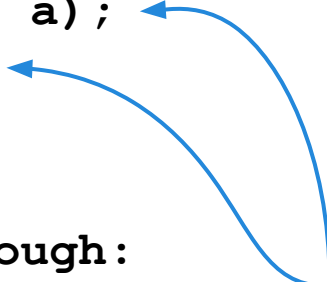
1. Add to your `String` class `operator==` and `operator[]`
2. Does this work with your string:
`String s="!!";`
Just for the sport: block it!

Casting Operator

A special operator

```
class A {  
    int i;  
public:  
    A(int i1):i(i1){}  
    operator int()const {  
        return i;  
    }  
};
```

```
int main() {  
    A a = 1;  
    int i = 3 + a;  
    i = pow(a, a);  
    a = 12.5;  
}  
  
// can go through:  
// 1 user casting + 1 c casting  
// (in any order)
```



Assignment Operator

An important “creature” in C++!

- Signature:

`A& A::operator=(const A& a) ;`

- Used when creating a copy on your own
- Don't confuse with Copy C'tor! They are very similar but not the same
- If you don't implement your own - you get a “free” one automatically!

??

Can we implement assignment as a global function:

`A& operator=(A& a1, const A& a2) ;`

Blocking Copy and Assignment

If the default Copy and Assignment are not OK
But, you do not need your own “deep-copy”

⇒ Simply Block them by defining them as private with no implementation

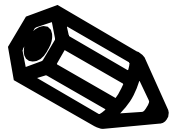
```
class A {  
    A(const A&) ;  
    A& operator=(const A&) ;  
public:  
    // ...  
};
```

Blocking Copy and Assignment

C++11 syntax:

```
class A {  
public:  
    A(const A&) = delete;  
    A& operator=(const A&) = delete;  
};
```

Exercise



1. **Block in your String class the default copy c'tor and assignment operator. Check they are indeed disabled!**
2. **Now remove the block and implement copy c'tor and assignment operator.**
3. **Add to your String class an operator for casting to const char***

static members and methods

Same as in Java...

Except for syntax:

```
A::static_method_call();  
A::static_member;
```

Also note:


- static field must be declared in .h and defined in .cpp
- the modifier 'static' appears only in .h file and not in .cpp

const methods - example 1

```
class MyString {  
    char* str;  
public:  
    // ...  
    void print() const {  
        std::cout << str;  
    }  
};  
  
// main  
const MyString s = "hello"; // assume there is a proper ctor  
s.print(); // OK - print is a const method
```

const methods - example 2

```
class Person {  
    std::string name;  
    long id;  
    Widget* widget;  
public:  
    // ...  
  
    const std::string& getName() const {  
        return name;  
    }  
    long getId() const {  
        return id;  
    }  
    const Widget* getWidget() const {  
        return widget;  
    }  
};
```



const + mutable members

```
class Array {  
    int arr[SIZE]{};  
    mutable int sum = 0;  
    mutable bool isSumUpdated = true;  
    void calcSum()const;  
public:  
    Array() {}  
    // cont' in next page
```

const + mutable - cont'

```
int& operator[](int index) {  
    isSumUpdated = false;  
    return arr[index];  
}  
int getSum()const {  
    if(!isSumUpdated) {  
        calcSum();  
    }  
    return sum;  
}  
void print()const;  
};
```

const + mutable - cont'

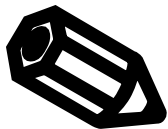
```
void Array::calcSum()const {
    sum = 0;
    for(int i=0; i<SIZE; ++i) {
        sum += arr[i];
    }
    isSumUpdated = true;
}

void Array::print()const {
    for(int i=0; i<SIZE; ++i) {
        cout << "arr[" << i << "] = " << arr[i] << endl;
    }
    cout << "Sum: " << getSum() << endl;
}
```

Code:

<http://coliru.stacked-crooked.com/a/186ace4664482b73>

Exercise



Improve the code in previous slides so that if an item is retrieved from the array for **reading** the sum would not be invalidated (it would be invalidated only if an item is retrieved for **writing**).

Solution (don't peek): <http://coliru.stacked-crooked.com/a/d81519b5722ba153>
using proxy class ^

Templated version: <http://coliru.stacked-crooked.com/a/dee8717f20ba8e8b>

friend

- **friend functions and friend classes can access private members of the one who declared them as friend**
- **friend functions are usually* global functions! even if implemented inside a class - yes, you can see a global function implemented in class**
 - * **(you can declare a member function of another class as friend of your class, but that's a very rare usage)**

The main usage of friend function is to declare a function in the context of a class, even though for technical reasons it have to be global.

Nested Classes

Same as nested static class in Java *

To use them, need to use full name:

`OuterClass::InnerClass myObj;`

*** (C++ doesn't have nested non-static class as in Java).**

Namespaces

Same as 'packages' in Java

C++

using namespace

Java

import

== ~

python

from <class> import *

Namespace declaration in C++ (== code *belongs to namespace*)
use curly brackets block for opening and closing

Intermediate Summary - Classes

- ✓ **C++ Class Syntax**
- ✓ **C'tor, D'tor, Copy C'tor, Casting w/ C'tor**
- ✓ **Operators (Assignment and others)**
- ✓ **static, const, mutable, friend**
- ✓ **Nested classes, Namespaces**

Our next item... Inheritance

- ⇒ **Inheritance Syntax**
- ⇒ **[private and protected inheritance]**
- ⇒ **Multiple Inheritance and “Interfaces”**
- ⇒ **Polymorphism and virtual functions**
- ⇒ **Abstract Classes**

Inheritance - First Example

```
class Person {  
    // ...  
};  
  
class Student: public Person {  
    vector<int> grades;  
public:  
    Student(const string& name):Person(name) {}  
  
};
```

private and protected inheritance

```
class B: private A {  
    // ...  
};
```

```
class C: protected A {  
    // ...  
};
```

```
// B and C are NOT a "type" of A  
// B and C hide the public interface of A  
// not so common  
// can be replaced easily with Containing A
```

Multiple Inheritance

```
class C: public A, public B {  
    // ...  
};  
  
// C is "type" of A and also "type" of B  
// must init both parents if they don't have an empty c'tor  
// not so common (brings all sort of complexities)  
// except for imitating Java interface  
// (i.e. A and/or B are "interfaces")
```

Multiple Virtual Inheritance

```
class A: virtual public Ancestor {};  
class B: virtual public Ancestor {};  
  
class C: public A, public B {  
    // ...  
};  
  
// C gets only one copy of Ancestor  
// C shall init Ancestor if it doesn't have an empty c'tor  
// even less common... (brings all sort of complexities)
```

Polymorphism - 1

```
class A {  
public:  
    void doIt();  
};  
  
class B: public A {  
public:  
    void doIt();  
};  
  
A* pa = new B();  
pa->doIt();
```

Polymorphism - 2

```
class A {  
public:  
    virtual void doIt();  
};  
  
class B: public A {  
public:  
    void doIt() override; // override contextual keyword added in C++11  
};  
  
A* pa = new B();  
pa->doIt();
```

Virtual D'tor

```
class A {  
public:  
    virtual ~A(){}  
};  
  
class B: public A {  
    Godzilla* godzil;  
public:  
    B():godzil(new Godzilla){}  
    ~B(){delete godzil;}  
    // ...  
};
```

```
int main() {  
    A* pa = new B();  
    // ...  
    delete pa;  
}
```

**?? What would happen
without virtual d'tor?**

Stack and BetterStack Example - 1

```
class Stack {
    int size, pos;
    int* arr;
public:
    Stack(int size1)
        : size(size1), pos(0), arr(new int[size1]) {}
    Stack(const Stack& s){arr=NULL; size=0; *this=s;};
    virtual ~Stack(){delete []arr;}
    virtual void push(int num){arr[pos++]=num;}
    virtual int pop(){return arr[--pos];}
    Stack& operator=(const Stack& s);
};
```

Stack and BetterStack Example - 2

```
Stack&
Stack::operator=
(const Stack& s) {
    if(&s != this) {
        if(size < s.pos) {
// we may prefer to be always exact
// same size, in which case shall
// check size != s.size
            delete []arr;
            size = s.size;
            arr = new int[size];
        }
        // cont =>
```

```
// do the copy
pos = s.pos;
for(int i=0; i<pos; ++i){
    arr[i] = s.arr[i];
}
return *this;
}
```

Stack and BetterStack Example - 3

```
class BetterStack: public Stack {
    int sum;
public:
    BetterStack(int size):Stack(size), sum(0){}
    void push(int num){Stack::push(num); sum+=num;}
    int pop(){int num=Stack::pop(); sum-=num; return num;}
};
```

Exercise - 1



We want to allow for the presented **Stack** and **BetterStack** the following assignments:

```
BetterStack bs;  
Stack s;  
bs = s;  
s = bs;
```

```
Stack* pStack = &bs;  
(*pStack) = s;  
BetterStack bs2;  
bs2 = *pStack;
```

Which assignment are already well supported?
Implement wisely the additional required code!

Exercise - 2



Implement another BetterStack that holds its own array with values of num^2 (in addition to the array of num values in the base class).

Which changes are required in the base class?

What are the differences from the BetterStack presented above?

Pure Virtual =0 and Abstract Classes

```
class Shape {
    Color fillColor, lineColor;
public:
    Shape(); // default colors
    Shape(Color fillColor1, Color lineColor1);
    virtual bool isPointInside(const Point& p) const=0;
    virtual void draw() const=0;
    virtual void move(const Point& src, const Point& dst)=0;
};
```

Pure Virtual =0 and Abstract Classes

```
class Circle: public Shape {
    Point center;
    int radius;
public:
    // ...
    virtual bool isPointInside(const Point& p) const {
        return p.distance(center) <= radius;
    }
};
```

main

```
// Shape s; // not ok
Shape* s = new Circle(); // ok
```

Exercise



Implement the required classes for the following main:

```
Expression* e = new Sum(  
    new Exponent(new Number(2),  
        new Factorial(  
            new Number(3))),  
    new Number(-2));  
cout << *e << "=" << e->eval() << endl;  
delete e;
```


Intermediate Summary - Inheritance

- ✓ **Inheritance Syntax**
- ✓ **[private and protected inheritance]**
- ✓ **Multiple Inheritance and “Interfaces”**
- ✓ **Polymorphism and virtual functions**
- ✓ **Abstract Classes**

Our next item... Templates

⇒ Templates is more than Generics

⇒ Template Functions

⇒ Template Classes

Template Stack

```
template<class T> class Stack {
    int size, pos;
    T* arr;
public:
    Stack(int size1)
        : size(size1), pos(0), arr(new T[size1]) {}
    Stack(const Stack& s){arr=NULL; size=0; *this=s;};
    virtual ~Stack(){delete []arr;}
    virtual void push(const T& val){arr[pos++]=val;}
    virtual T pop(){return arr[--pos];}
    Stack& operator=(const Stack& s);
};
```

Template Stack - Usage

```
int main() {  
    Stack<Person> persons(10);  
    persons.push(Person("joko"));  
    persons.push(Person("jasmin"));  
    persons.push(Person("joshua"));  
    persons.pop().print();  
    persons.pop().print();  
    persons.pop().print();  
};
```

Template Swap

```
template<class T>
void swap(T& lhs, T& rhs) {
    T temp = lhs; // with C++11 there is a better way
    lhs = rhs;
    rhs = temp;
}

int main() {
    Person p1("dudu"), p2("mika");
    swap(p1, p2);
}
```

Intermediate Summary - Templates

- ✓ **Templates is more than Generics**
- ✓ **Template Functions**
- ✓ **Template Classes**

A few words on Exceptions

- **Similar to Java (almost...)**
- **When/Why to avoid (Performance Price)**

see also: <http://google-styleguide.googlecode.com/svn/trunk/cppguide.html#Exceptions>

- **Alternatives**
- **There is no finally in C++ -- use D'tors**

End of 01 - C++ Intro



Questions?