C++, Cmake

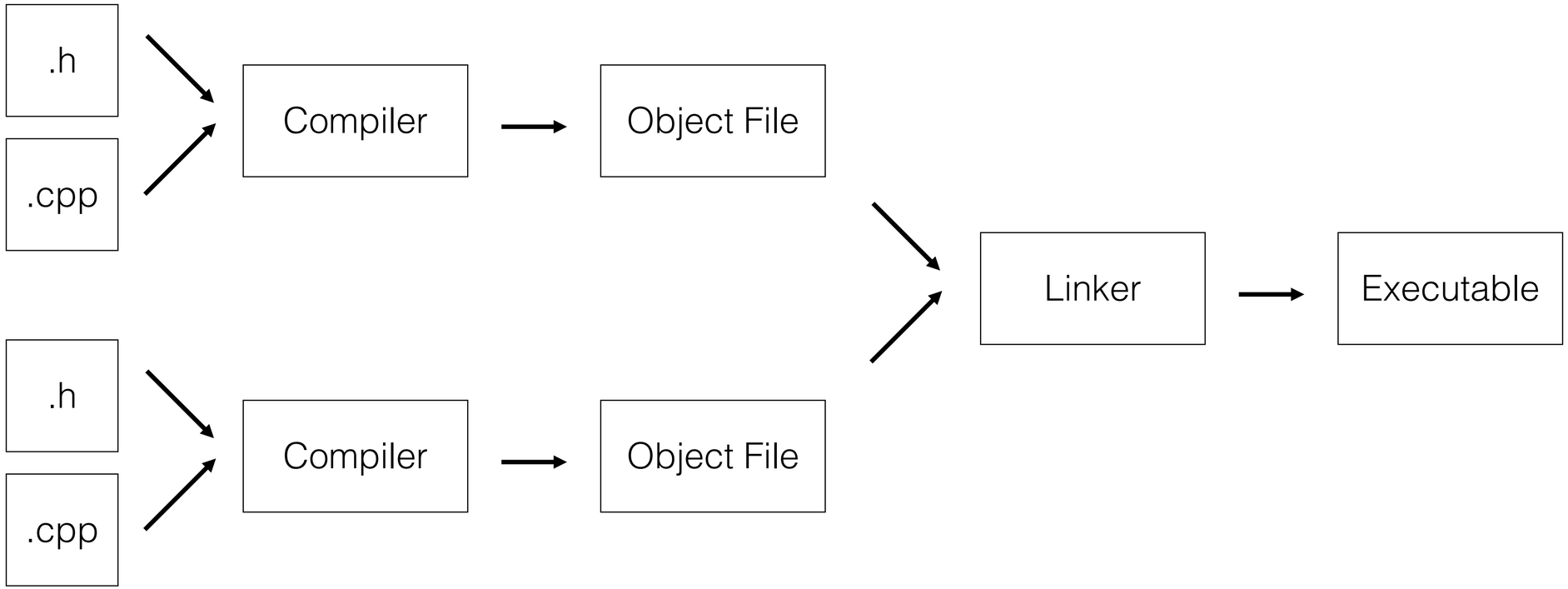
Acknowledgements: Daniele Panozzo

Why C++?

- It is the industry standard for Computer Graphics
- It allows to write highly efficient code in a convenient way
- I will give an overview of its main features, if you never used it before you will have to study it on your own
- The quality of the code will not be evaluated in the assignments. However, if you learn how to write good C++ code it will greatly simplify the homework

What is C++?

- It is a compiled language:



http://www2.hawaii.edu/~takebaya/ics111/process_of_programming/process_of_programming.html

CMAKE and Makefile

- When some files are changed, the compiler needs to be called again
- A “make system” takes care of this for you
- We use CMAKE, it is straightforward to use and you can just take a look at the provided files and do cut&paste
- Important: you need to first launch CMAKE to generate a project file, then use the platform dependent make system to compile

CMAKE

- If your project is in a folder Assignment_1, you need to:
  - mkdir build; cd build
  - cmake ../
- This will create the project. To compile it:
  - make (macosx/linux)
  - Open the project with visual studio 2015 on windows
- As an alternative, you can use CLion that does all of this for you!

Basic CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8.12)
project(Assignment1)

### Add src to the include directories
include_directories("${CMAKE_CURRENT_SOURCE_DIR}/src")

### Include Eigen for linear algebra
include_directories("${CMAKE_CURRENT_SOURCE_DIR}/../ext/eigen")

### Compile all the cpp files in src
file(GLOB SOURCES
"${CMAKE_CURRENT_SOURCE_DIR}/src/*.cpp"
)

add_executable(${PROJECT_NAME}_bin ${SOURCES})
```

https://cmake.org/cmake-tutorial/

C++

- It is flexible, and many of the features are optional:
  - it can be used as an extension of C, with no objects
  - it can be used as a fully object oriented language
  - it has many advanced features such as “templates” that are very useful to write efficient code that *is also readable*

Comparison with Java

- **Java**: Everything must be placed in a class. **C++**: We can define functions and variables outside a class, using the same syntax used in ANSI C. The ”main“ function must be defined outside a class.
- **Java**: All user-defined types are classes. **C++**: We can define C types (enum,struct,array).
- **Java**: Single Inheritance. **C++**: Multiple Inheritance

Comparison with Java

- **Java**: No explicit pointers. **C++**: Explicit pointers and ”safe pointers“ (called Reference) are available.
- **Java**: Automatic memory management. **C++**: Manual memory management (like C) or semi-automatic management (shared pointers, http://en.cppreference.com/w/cpp/memory/shared_ptr).
- **Java**: All objects are allocated on the heap. **C++**: An object can be allocated on the heap or on the stack.

C Pointers

- **int*** p; ← p is not initialized here, it contains a random value

It defines a pointer ***p*** to an integer

- **int** x; p = &x;

It defines a variable ***x*** and assigns the address of ***x*** to ***p***.

- (*p) = 15;

It assigns the value 15 to the location pointed by ***p***. It changes the value of ***x***

Reference Type Modifier &

- **int**& p;

Error! A reference must always be initialized.

- **int** x; **int**& p = x;

Define a variable x and a reference to it called p. P behaves like a variable, it does not use the notation used for pointers.

- p = 15;

Assign the value 15 to the reference p. The value of x is now 15.

Reference Type Modifier &

- Reference types are safe, they behave like pointers, but they cannot contain a dangling value
- Use them extensively and avoid to use pointers at all costs! Pointers are only needed in very few cases and only for performance reasons

https://en.wikipedia.org/wiki/Reference_(C++)

Example of using &

```
void swap(int& a, int& b)
{
 int s = b;
 b = a;
 a = s;
}
```


Classes - Java

```
class point
{
  public float x;
  public float y;
  public void print()
  {
    System.out.println("Point(" + x + "," + y + “)");
 }
  public void set(float x0, float y0)
  {
      x = x0;
      y = y0;
  }
  public point(float x0, float y0)
  {
    set(x0,y0);
  }
};
```

Classes - C++

```
   File - point.h
class point {
  public:
    float x; float y;
    void print();
    void set(float x0, float y0);
    point(float x0, float y0);
}; <------- NOTE: Remember the ";"

   File - point.cpp
#include "point.h"
void point::print()
{ printf("Point %f,%f", x, y); }
void point::set(float x0, float y0)
{ x = x0; y = y0; }
point::point(float x0, float y0)
{ set(x0,y0); }
```

Visibility

- C++ supports 3 types of visibility associated to a method or a variable inside a class:
  - **Public**: the function or variable can be used in any functions
  - **Private**: the function or variable can be used only in function defined inside the same class. It is not possible to use it in any derived class
  - **Protected**: the function or variable can be used in functions defined in the class or in derived classes

Method body in .h or .cpp?

- **Usually**, a .h file must contain only prototypes of functions and classes. Since they are included in other files, a big .h file will slow down the compilation of all the source files that include it

- On the other hand, **C++ allows the definition of the body of a method directly in the definition of the class**

- When this is done, the body of the method is compiled in every source file that include its definition, and the compiler can perform optimization on it (inline methods). This is very useful for small methods that are called millions of times. In this case the run-time performance gain justify the increment of compilation time and binary size

http://www.cplusplus.com/articles/2LywvCM9/

Example

```
class point
{
  public:
    point(float x0, float y0)
    {
      x = x0;
      y = y0;
    }
    float& x() { return x; }
    float& y() { return y; }
  private:
    float x; float y;
};
```

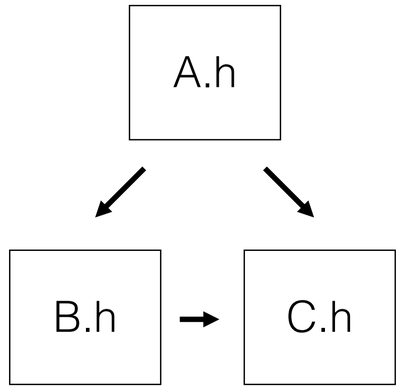
How to write a header (.h) file

- A single .h can be included by multiple files. Suppose that file A.h includes both B.h and C.h. File B.h includes C.h too.
- Every file that includes A.h will contain two versions of C.h inside and it will inevitably lead to compilation error due to the repeated definition of the same classes/functions.
- To avoid it we tell the compiler to include each .h once per source file using the #ifndef directive.

```
#ifndef FILENAME_H_
#define FILENAME_H_

code

#endif
```



Dynamic and Static Binding

- All methods in java are subject to dynamic binding
- The C++ methods are not subject to dynamic bindings. It is possible to enable dynamic binding for a particular method if the method prototype is preceded by virtual.
  - Example: **virtual void** someMethod(**int** a);

Every time a virtual function is called, the run-time environment must check the dynamic type of the interested object and find the correct method to call. With static binding the compiler can decide the method to call at compile time.

Creating Objects - Stack

- A new object can be created in the same way as a variable
- In this case the object is allocated on the stack
- Suppose that we defined a class Point, with two standard constructors. The first is the default constructor that does not accept any parameter, the second accepts a pair of float numbers. An object can be created with:
  - Point p; <— No parameters, no parenthesis (Point p(); is wrong!)
  - Point p(2.0,3.0); <— The parenthesis are required if the constructor has one or more parameters

The object will be destroyed when the variable p goes out of scope.

Creating Objects - Heap

- A new object can be created with the operator **new**
- In this case the object is allocated on the heap. Suppose that we defined a class Point, with two standard constructors. The first is the default constructor that does not accept any parameter, the second accepts a pair of float numbers. An object can be created with:
  - Point* p = **new** Point; <— No parameters, parenthesis optional (Point* p = **new** Point(); is ok)
  - Point* p = **new** Point(2.0,3.0);

The object will **NOT** be destroyed when the variable p goes out of scope. It must be manually deallocated using the operator **delete**.

Heap vs Stack

Stack

- The deallocation is automatic, cannot be used if the object must survive outside the scope of the variable.
- Memory leaks cannot happen
- ”Fast“, to be used when it is possible.

Heap

- The deallocation is manual, the user has control over the deallocation of the object.
- Memory leaks happen in this way!
- ”Slow“, to be used with care, in particular if you allocate a lot of small objects.

Constructor and Destructor

- As in Java every class must have a constructor. It is defined in the same way as a method, but with the name equal to the class name and no return type

- It is possible to provide more than one constructor for a single class, but their prototypes must be different

- A C++ class must also have a destructor, that it is called when the class is destroyed. This can happens when an object on stack goes out of scope, or when an object on the heap is deleted with the **delete** keyword

Example - Don’t do it!

```
class PointPair
{
public:
  PointPair() { init(); }
  PointPair(Point* xp, Point* yp) {
    p1=xp; p2=yp; }
  virtual ~PointPair() { delete p1; delete p2; }

  void init()
  { p1 = new Point; p2 = new Point; }

Point* p1;
Point* p2;
};
```

Example - Fixed

```
class PointPair
{
public:
  PointPair() { init(); }
  PointPair(Point* xp, Point* yp) {
    init(); *p1=*xp; *p2=*yp; }
  virtual ~PointPair() { delete p1; delete p2; }

  void init()
  { p1 = new Point; p2 = new Point; }

Point* p1;
Point* p2;
};
```

Public and Private Inheritance

```
class point3D : public point
{
  public:
    float z;
    point3D(float x0, float y0, float z0) : point(x0,y0)
    {  z = z0;  }
};
```

- C++ uses ":“ to separate the name of the class from the list of its parent classes
- The same syntax is used to call the constructor of the parent class (line 5 and 6)
- The public keyword (line 1) before the class point, indicates that the (public) methods of point will become public in point3D
- It is possible to reduce the visibility of the inherited methods/variables changing public in private

C++ Templates

- C++ supports **generics**. The syntax is similar to the one used in java generics but they are implemented differently

- In java a generic class is compiled independently of the parameters, and the run-time environment takes care of the required type conversions at run time

- In C++ for every set of parameters, the class is recompiled. The gain in speed achieved by the C++ generics is due to the reduction of work for the runtime environment. The drawback is the increase in size of the final executable.

C++ Templates

- Both the prototype and the implementation of a template class must be placed inside the header file! Elsewhere the compiler is not able to find the source code necessary to recompile the class at every instantiation.

```
template<class T>
class Pair
{
  private:
    T f; T s;
  public:
    inline T& first()
    { return f;  }

    inline T& second()
    { return s; }
};
------------------
Pair<double> p1;
p1.first() = 5.6;
double d = p1.first();
```


Operator Overloading

- The operator overloading is a technique that allows to call a predefined function using the notation assigned to a standard operator.
- In C++ it is only possible to overload the operators already available (+, -, *, =, etc.), it is not possible to define new operators
- Eigen uses operator overloading extensively!

Standard Template Library

- STL is a library included in the C++ standard.
- Check http://www.cppreference.com for a manual.
- We will look only at the basic dynamic data structures, please take a look at the documentation to avoid to waste time rewriting stuff already available.
- Most of the classes in STL are based on templates to specify the type of managed data.

Iterators

- Similarly to java, the containers in STL are based on the concept of an iterator.
- An iterator behave exactly as a pointer of the same type of the underlining container.
- Suppose we defined a list of integer (class list<int> as we will see later), the corresponding iterator type will be list<int>::iterator.
- An iterator must be created with the methods begin() and rbegin() present in every container. They return an iterator pointing the first and the last element, respectively.
- There are two special iterators, container.end() and container.rend(), that points to an element after the last and an element before the first, respectively.
- They are used to check when we reach the end or the beginning of our container.

Iterator's Operators

- An iterator it implements the following operators:
  - ++it ti++ (what to use in a context where their behavior is the same?)
  - --it it-- (not supported by all iterators)
  - *it (read or edit the pointed item)
  - it->method() (equivalent to (*it).method())
  - it == it2 (check if the two iterators point to the same element)
  - it != it2 (check if the two iterators point to different elements)
  - it = it + x (x is an integer, move the iterator forward of x elements)
  - it = it - x (x is an integer, move the iterator backward of x elements)

Vectors

- A vector is a dynamic array. Use it in any case you need an array, the overhead in respect to a c array is negligible. The vector class is a template, it must be initialized with the type of the contained elements.
  - #include <vector> *<— header file containing the definition*
  - using namespace std; *<— vector is in the std namespace vector<int>*
  - vector<int> v; *<— it creates a container of integers*
  - vector<int*> v; *<— it creates a container of pointers to integers*
- Useful methods and operators, we assume that vec is a vector:
  - vec.push_back(item) - *it inserts item at the end of the vector, if no space is available the entire array is reallocated and expanded to make room for the new item*
  - vec.erase(iterator) *- it remove the item pointed by iterator (warning, the array must be reallocated if the elements removed is not the last)*
  - vec[i] - *ith element (starts from 0!)*
  - vec.resize(i) *- it resizes the vector to size i*

Lists

- A C++ list is a double linked list. The list class is a template, it must be initialized with the type of the contained elements.
  - #include <list> *<— header file containing the definition*
  - using namespace std; *<— list is in the std namespace*
  - list<int> l; *<— it creates a container of integers*
  - list<int*> l; *<— it creates a container of pointers to integers*
- Useful methods and operators, we assume that lis is a list:
  - lis.push_back(item) *- it inserts item at the end*
  - lis.push_front(item) *- it inserts item at the beginning*
  - lis.erase(iterator) *- it removes the item pointed by iterator*
  - list.front() *- first element*
  - list.back() *- last element*
  - lis.clear() *- it removes all elements*

Example

```
#include <list>
list<int> li;
list<int>::iterator it;

li.push_back(5); li.push_back(6); li.push_back(7); ...

for(it = li.begin(); it != li.end(); ++li)
{
  int& temp = *it;
  // do something with temp
}
```

Assignment 1

- Let's take a look at the provided code and at the assignment description

References

- **Thinking in C++ second edition** - Bruce Eckel
  http://www.mindview.net/Books/TICPP/ThinkingInCPP2e.html
- **Wikipedia** http://en.wikipedia.org/wiki/C%2B%2B
- **Cpp Reference** http://www.cppreference.com
- **Cmake** https://cmake.org