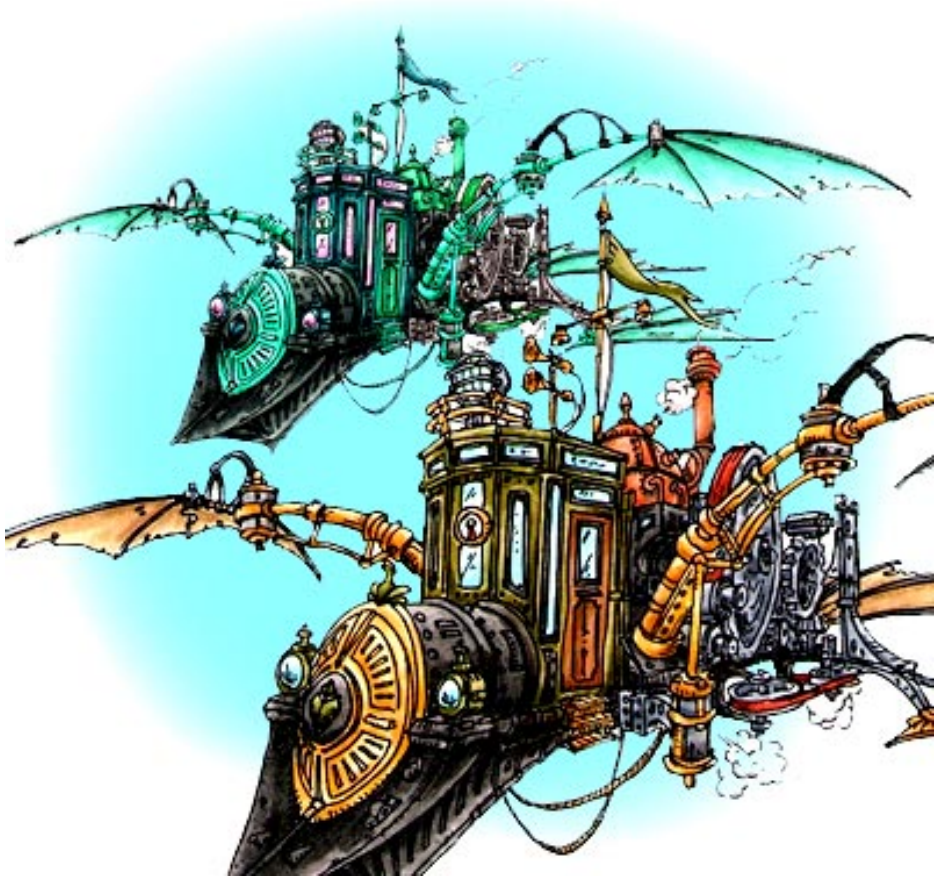
8. The C++ language, 2



Programming and Algorithms II

Degree in Bioinformatics

Fall 2018

(Linked) Lists

Bad at:

- Access to a random position list[i]: O(size)

Good at:

- Inserting where you have a finger O(1)
- Deleting where you have a finger O(1)
- Extracting a sublist marked by two fingers O(1)

We will see next day how to use them.

Implementation next course (uses pointers and dynamic memory)

Linked lists

list<string> lst; // creates an empty list

list<string>::iterator it1 = lst.begin();

list<string>::iterator it2 = lst.end();

Yes:

- it = lst.begin(); it = lst.end();
- it1 = it2;
- it1 == it2, it1 != it2
- *it : element “under” it // error if it == lst.end()
- ++it , --it // except if at l.end() and l.begin, respectively
- (think of as “move forward” and “move back”, not “increment” and “decrement)

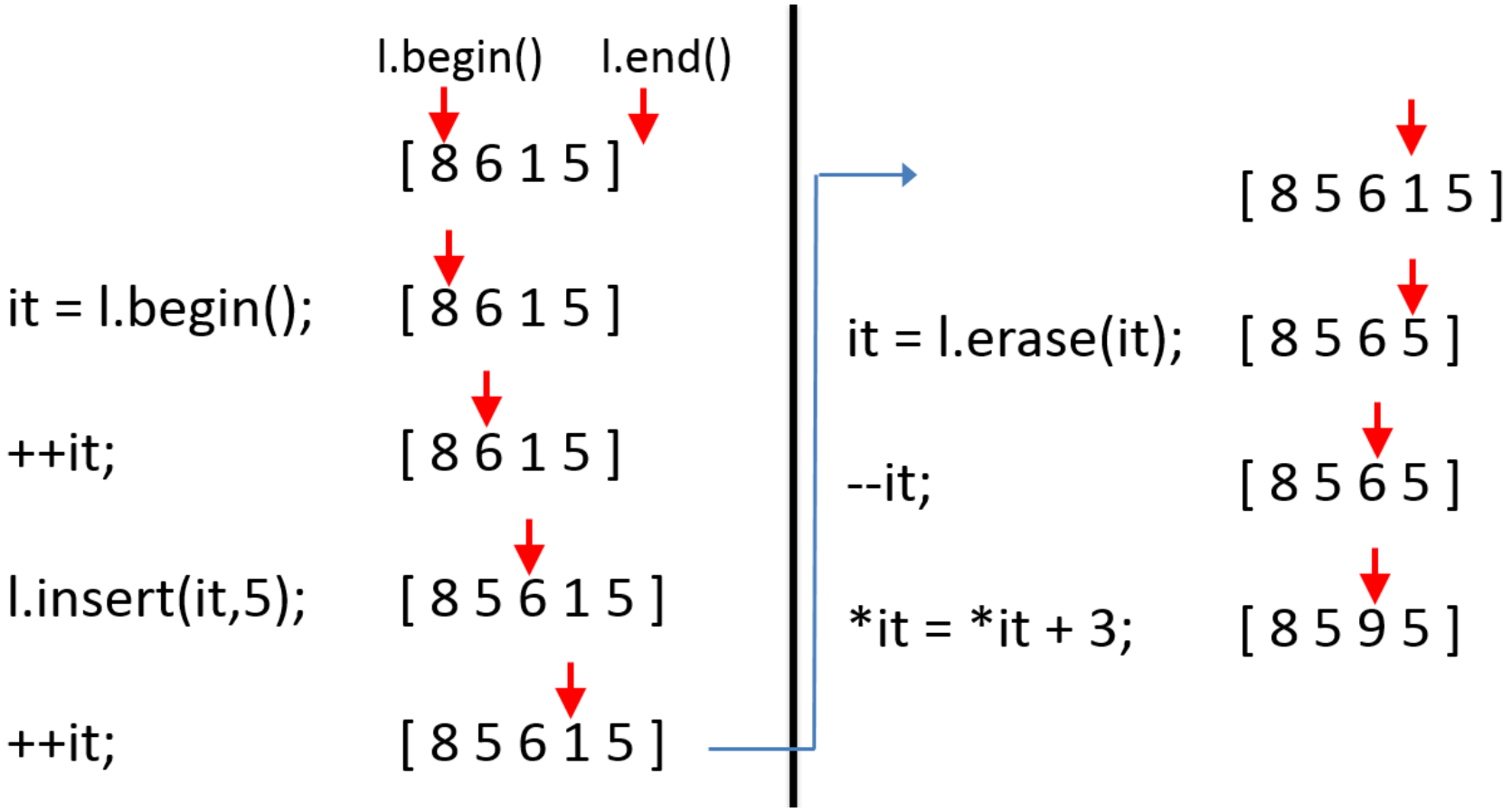
Linked lists

```
list<string> lst;  // creates an empty list
list<string>::iterator it1 = lst.begin();
list<string>::iterator it2 = lst.end();
```

NO:

- lst[index]
- it = it + 5; it = it - 1;
- (because ++it looks like it = it+1, but it is “advance it”)
- if (it1 < it2) ...

Linked lists



Linked lists: noob mistake

```
float sum = 0;
for (int i = 0; i < lst.size(); ++i)
       sum += get(lst,i);

int get(const list<float>& lst, int pos) {
      list<float>::iterator it = lst.begin();
      for (int i = 0; i < pos; ++i) ++it;
      return *it;
}
```

Note:

list<T>::iterator: allows changing list… *it = …

list<T>::const_iterator: only "… = *it"

Linked lists: usual scheme

```
float sum = 0;
list<T>::iterator it = l.begin();
while (it != l.end()) {
      sum += *it;
      ++it;
}
```

Usual search scheme:

```
list<T> lst;
list<T>::iterator it = l.begin();
while (it != l.end() and not (condition on *it)) {
      … access *it …
      ++it;
}
```

Is this list a palindrome?

```
bool is_palindrome(const list<int>& lst) {
    if (lst.empty()) return true;
    list::const_iterator it1 = l.begin();
    list::const_iterator it2 = l.end();
    --it2;
    while (it1 != it2) {
        if (*it1 != *it2) return false;
        ++it1;
        if (it1 == it2) return true;
        --it2;
    }
    return true;
}
```

Maps and sets

Dictionaries are called “maps” in C++

Unordered maps (hashing):

- O(1) access time
- get all keys or values: in random order

Ordered maps (balanced trees):

- O(log(size)) access
- Get all keys or values in order, O(size) time

map<key,info>:

- can only map objects of type key to objects of type info
- Key needs to be hashable
- It has iterators

Maps and sets

```
#include <map>
map<string,Point> gpsinfo;

# usually we give write gps info as (latitude,longitude)
# but Point(…) takes first x(=longitude) then y(=latitude)

gpsinfo[“Barcelona”] = Point(2.15,41.39);
gpsinfo[“New York”] = Point(-73.93,40.73);
gpsinfo[“Buenos Aires”] = Point(-58.38,-34.60);
…
if (gpsinfo[c].get_y() < 0)
    cout << c << “ is in Southern hemisphere)” << endl;
…
map::const_iterator it = gpsinfo.find(c)
while (it != gpsinfo.end())
   cout << it->first() << “ is at (”
       << it->second.get_y() << “,“
       << it->second.get_x() << “)“ << endl;
```

Classes

Implemented in two files: a header file and an implementation file

Here is what I would like:

my_class.h: All that you need to use the class

class name, *declaration* of public operations (header, not the body)

including a creator operation and a destroyer operation

my_class.cc (or .cpp):

definition of attributes,

definition of private auxiliary operations: header and body

body of public ops, using the attributes and private ops

You give your teammates my_class.h and my_class.o

They do #include “my_class.h” and link with my_class.o

Classes

Implemented in two files: a header file and an implementation file

Here is how it really is:

my_class.h: Contains a public part and a private part:

public: declaration of public operations and (if any) public attributes

private: declaration of private attributes and private operations

my_class.cc (or .cpp):

body (implementation) of public and private operations

Note:

- Private attributes and private operations can only be accessed from within the operations that are members of the class. Error otherwise.
- In my_class.cc we are outside the class. We need to tell the compiler “we are implementing the operations that we declared in my_class.h”

Classes: Example

File Point.h:

```
class Point {

public:
  Point();    // returns a new point containing (0,0)
  Point(float x, float y);   // returns a new point containing (x,y)
  void set_x(float x);
  void set_y(float y);
  float get_x();
  float get_y();
  float get_distance(const Point& another_point);
  ~Point();  // destructor operation

private:
  float x;
  float y;

};   // yes, here we have ; after the }
```

Classes: Example

File Point.cc:

```
#include “Point.h”

Point::Point() { x = 0; y = 0; }
Point::Point(float x, float y) {
      this->x = x; this->y = y;
      // “this” is like “self”. use -> instead of .
      // unlike Python, not needed if there is no confusion with parameters
}
void Point::set_x(float x) { this->x = x; }
void Point::set_y(float y) { this->y = y; }
float Point::get_x(); { return x; }  // see? No “this”
float Point::get_y(); { return y; }  // see? No “this”
float Point::get_distance(const Point& another_point) {
      float diffx = another_point.x – x;
      float diffy = another_point.y – y;
      return sqrt(diffx*diffx – diffy*diffy);
}
Point::~Point() {}   // nothing special to be done when a Point is destroyed
```

More on classes

no “self” in parameter list

“this” can be used optionally

Necesary if there is a collision of names

We will deal with C++ pointers next course. Preview:

“this” is actually a pointer to the implicit parameter

- *this is the real object
- a->b is a short hand for (*a).b
- so this->method(…) and this->attribute are the method and attribute of the implicit parameter (object)

More on classes

classname::something means

"that something that was defined inside class classname"

Frequent error: in Point.cc, writing

```
float get_y() { return y; }
```

instead of

```
float Point::get_y() { return y; }
```

you get error "y is undefined"

Class attributes and method

Static in front of an attribute or method indicates that it belongs to the class, not objects

You can't use "this" inside a static method, or call non-static methods from there

Person.h:

```
class Person {
 public:
       Person(string name, int age);
       string get_name();
       int get_age();
       static bool exists(string name);
       static int people_created();
   private
       static int how_many = 0;
       static set<string> names;
       string name;
       int age;
};
```

Person.cc

```
Person::Person(string name, int age) {
    if (not Person.exists(name))
        ++how_many;
    this->name = name;
    this->age = age;
}
static bool Person::exists(string
name) {
    return names.count(name) > 0;
}
static int Person::people_created() {
   return how_many;
}
```

Class attributes and methods

Static in front of an attribute or method indicates that it belongs to the class, not objects

You can't use "this" inside a static method, or call non-static methods from there

Person.h:

```
class Person {
   public:
       Person(string name, int age);
       string get_name();
       int get_age();
       static bool exists(string name);
       static int people_created();
   private
       static int how_many = 0;
       static set<string> names;
       string name;
       int age;
};
```

```
string name;
cin >> name;
while (Person.exists(name)) {
    cout << "someone called "
    << name
    << " exists; choose another name "
    << endl;
    cin >> name;
}
int age;
cin >> age;
Person p = Person(name,age);
```

Inheritance: an intro

```
class A {...};
class B {...};
class C: public A, public B { ... };
```

C “is derived” or “inherits” from A and B

`public` means that all public attributes and method of A and B are available and public in C objects

If you change to `private`, they are there but can only be accessed from inside C methods

If you change to `protected`, they are there but can only be accessed from C methods and methods in any subclass of C

It gets complicated. `friend` classes can access private methods and attributes

Summary

- Lists are different from Python lists. They are great for inserting and deleting, bad for random access by position
- Maps, sets are similar to Python dictionaries, sets
- But they can only contain values of one type
- Accessed via iterators

- Classes: Distinction public / private by syntax

What we expect you to know about C++

Well:

- Variables, expressions, function calls
- Loops
- Multi-file programs, compiling, linking
- Vectors. Difference with arrays
- Translate Python programs with loops and lists to C++ programs with loops and vectors

A bit (I can learn more if needed):

- C++ lists and how they differ from Python lists
- Maps and sets
- OOP. I can modify existing simple classes
- That inheritance exists also in C++