Advanced Software Engineering with C++ Templates

Lecture 1: Administrative Issues and Introduction

Thomas Gschwind <thg$_{at}$zurich$_{dot}$ibm$_{dot}$com>

Agenda

- Administrative Issues
  - The name of the game
  - Schedule
  - Lectures, Exercises, Exam
- C++ Highlights
  - Templates
  - User-Defined Types
- The Standard Library
  - Basic Input and Output
  - Strings
  - Containers
  - Iterators

Prerequisites

- Programming Languages
  - Some knowledge of C (pointers, memory management), Java
  - Basic knowledge of object-oriented programming
- For students in the third semester or later

The Name of the Game

- We look at the C++ Programming Language
  - The language features and how they can be used
  - The Standard Library and how it uses the features from C++
  - Examples will be small and appear rather “low-level”
- C++ allows developers to use different programming paradigms
  - Procedural programming
  - Object-based programming (templates, static polymorphism)
  - Object-oriented programming (inheritance, dynamic polymorphism)
- Languages Features of Special Interest
  - Typing
    - Strong vs. weak typing
    - Static vs. dynamic typing
  - Templates
  - Exception Handling
  - Standard Library

It's Rude to be Alive When "No One" Wants You

- C++ is a C-based language that is infamous for being cryptic
  - Many C developers find it too high-level and having too much overhead
  - Java/Python/… developers find it too low-level
- Actually there are many Open-Source projects that use C++
  - Tensorflow, Tesseract
  - Mysql
  - Many KDE applications (The widget library (Qt) is implemented in C++)
  - Some Gnome applications
- C++ provides instructions to influence low-level code generation
  - Today's computer consume little power and are blazingly fast
  - Why would we care?

Embedded Devices

- Anything to be executed on “small” microprocessor
  - Arduino UNO: ATmega@16MHz
  - Raspberry PI: ARM@700MHz
  - Many many more

Arduino UNO
SDK uses C++
Provides 32k Flash

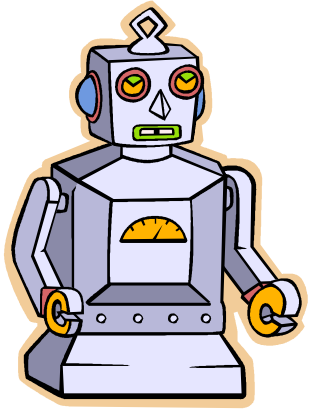
More Embedded Devices

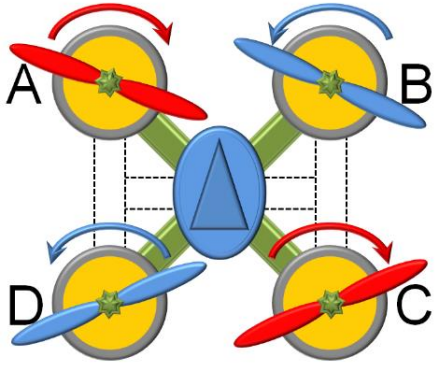
- Anything to be executed on battery-driven processors
  - You programs takes twice the time
  - Your battery lasts half as long max.
  - Power consumption is quadratic to frequency
  - This is why Android's dalvikvm is implemented in C++
  - Apple chose Objective C for everything

- Smartwatches
  - Today they last just a bit more than a day
  - An efficient program really is important
  - Don't want to lug around a watch with extra battery

Even More Embedded Devices

- Anything to be executed on battery-driven processors
  - Now imagine we want to move our embedded system around…
  - Robots, worse, helicopter, quadrocopter, we need to lift our battery as well





Author: Ulrich Heither

Grading

- Exercises handed out after every lecture
  - Two examinations of the exercises
- Exam at the end of the course
  - Probably written (oral if a small number of students)
  - No material is allowed
  - Approx. 50-70% theory, 30-50% practical questions
- Grading
  - Exercises determine 1/3 of the final grade
  - Exam determines 2/3 of the final grade

Organization & Schedule

- Announcements via the course web page
  - http://researcher.watson.ibm.com/researcher/view_group.php?id=5013
  - Urgent announcements by email (mailing list to be announced)
- Approx. 12 lectures, 1.5 hours each
  - Thu. 8:10-9:45 BIN 2.A.10.
  - There is no lecture on Thu. Oct. 27
  - There is no lecture on Thu. Dec. 15
  - For a complete schedule, please check the web
- Exercise submissions and Exam
  - Exercises due:
    - Thu. Nov. 3, 8:00 BIN 2.A.10
    - Thu. Dec. 22, 8:00 BIN 2.A.10
  - Exam on Thu. Jan. 12, 8:15 BIN 2.A.10

Q & A?

- Course material?
  - Slides
- Books?
  - Bjarne Stroustrup.
    The C++ Programming Language (3rd Ed.).
    Addison-Wesley.
  - Stanley B. Lippman, Josée Lajoie, Barbara E. Moo.
    C++ Primer (5th Ed.).
    Addison-Wesley.
  - Scott Meyers.
    Effective C++: 55 Specific Ways to Improve Your Programs and Designs
    Addison-Wesley
  - Check out your local book store
- Which operating system/C++ compiler?
  - Unix/Windows/OSX
  - GNU C++ (strongly recommended)

A Request from the Lecturer

- If anything is unclear
  - Please ask a question
  - This makes the lecture more lively and interesting
- If you find mistakes in the slides
  - Please point them out; if you are unsure, privately after the lecture
  - Improves the quality of the lecture
- I will compare with other programming languages
  - Of course, I will stress the advantages of C++, and is the focus
  - Yes, C++ has its downsides, will address them as well
  - After all, any language can be used to write good and bad programs

Agenda

- Administrative Issues
  - The name of the game
  - Schedule
  - Lectures, Exercises, Exam
- C++ Highlights
  - Templates
  - User-Defined Types
- The Standard Library
  - Basic Input and Output
  - Strings
  - Containers
  - Iterators

Templates – Why?

- Writing gcm, lcm, and swap for all kinds of types is tedious
- We want to define the function once and use it for all types possible
  - Sort of like Lisp, Smalltalk, Python, Ruby, you name it…
  - Just more efficiently

Templates – How?

- Types become parameters
- Support generic programming
  - Many functions are the same independently of the data type
- Types need to implement those routines necessary for the template
- Function calls can be resolved during compilation time

```
template <class T>
T gcf(T a, T b) {
  if (a<b) swap(a, b);
  while (b!=0) { a=a-b; if (a<b) swap(a, b); }
  return a;
}
template <class T>
inline T lcm(T a, T b) {
  return (a/gcf(a, b))*b;
}
```

Using Templates

- Are are invoked like any other function (mostly)
  - int n=lcm(3,7);
- Rarely, they need to be qualified
  - double d=3;
  - int n=lcm(d,7); // ERROR
  - Int n=lcm<int>(d,7); // now it is clear which lcm is meant
- More about this in the next lecture
  - Yes, the same can be done with classes
  - Yes it works in combination with overloading
  - Yes, it is not trivial, but indefinitely powerful

User-Defined Types

- Definition of types which behave similar to built-in types
- Example
  - Rational numbers that can be used like the **`int`** data type
  - Differences between Java and C++

User-Defined Fraction Type in C++ and Java

```
class fraction { // type definition
private:
  int cntr; int denom;
public:
  fraction(int c=0, int d=1)
    : cntr(c), denom(d) {}
  int get_counter() {
    return cntr; }
  void set_counter(int c) {
    cntr=c; }
  int get_denominator() {
    return denom; }
  …
};
```

indicates default parameters

// “same” as { cntr=c; denom=d; }

```
class Fraction { // type definition
  private int cntr;
  private int denom;

  public Fraction(int c, int d) {
    cntr=c; denom=d; }
  public int getCounter() {
    return cntr; }
  public void setCounter(int c) {
    cntr=c; }
  public int getDenominator() {
    return denom; }
  …
}
```

Using the Fraction Data Type

- In Java

```
void main(String[] args) {
  Fraction f=new Fraction(Integer.parseInt(args[1]),
                          Integer.parseInt(args[2]));
  System.out.println(f.getCounter()+":"+f.getDenominator());
}
```

- In C++

```
void main(int argc, char *argv[]) {
  fraction f(atoi(argv[1]), atoi(argv[2]));
  cout << f.get_counter() << ":" << f.get_denominator() << endl;
}
```

Sidebar: Memory “Management” in C++

- In Java
  - Built-in types are stored on the stack
  - Objects are stored on the heap
    (references to objects are stored on the stack)
  - Only references to objects
  - Memory is automatically freed by the garbage collector
- In C++
  - Built-in types and objects both can be stored on the stack and the heap
  - C++ not only supports references but also pointers:
    both can refer/point to objects and built-in types

Operators

- We would like to do calculations with our Fraction type
- How do we define arithmetic operations such as +, -, … in Java?
- How are these operations invoked in C++ and Java?
- Readability of the two approaches?

Operators(?) in Java

```
class Fraction {                        // type definition
  private int c;
  private int d;
  public Fraction(int cntr, int denom) { c=cntr; d=denom; }
  public Fraction mul(Fraction b) {
    int f1=gcf(this.c, b.d), f2=gcf(b.c, this.d);
    return new Fraction((this.c/f1)*(b.c/f2), (this.d/f2)*(b.d/f1));
  }
  public Fraction div(Fraction b) {
    return mul(new Fraction(b.d, b.c));
  }
}
```

```
Fraction f = new Fraction(1, 2);
Fraction g = new Fraction(3, 1);
Fraction h = f.mul(g);
Fraction i = h.add(g.mul(h)).mul(h);
```

Operators as Stand-Alone Functions

- C++ supports user-defined operators

```
fraction operator*(fraction a, fraction b) {
  int f1 = gcf(this.c, b.d), f2 = gcf(b.c, this.d);
  return fraction((a.get_counter()/f1) * (b.get_counter()/f2),
                  (a.get_denominator()/f2) * (b.get_denominator()/f1));
}
fraction operator/(fraction a, fraction b) {
  return a*fraction(b.get_denominator(), b.get_counter());
}
```

- Operator invocation

```
fraction f(1, 2);
fraction g(3);
fraction h = f*g;
fraction i = (h+g*h)*h;
// if you like, this works too
// fraction h2 = operator*(f, g);
```

```
Fraction f = new Fraction(1, 2);
Fraction g = new Fraction(3, 1);
Fraction h = f.mul(g);
Fraction i = h.add(g.mul(h)).mul(h);
```

Operators as Member Functions

- Operators may also be defined as member functions
- There is no difference for its invocation
- The left argument implicitly becomes the this argument

```
class fraction {  // type definition
private:
  int c; int d;
public:
  fraction(int cntr=0, int denom=1) : c(cntr), d(denom) {}
  fraction operator*(fraction b) {
    int f1=gcf(c, b.d), f2=gcf(b.c, d);
    return fraction((c/f1)*(b.c/f2), (d/f2)*(b.d/f1));
  }
  fraction operator/(fraction b) {
    return (*this)*fraction(b.d, b.c);
  }
}
```

Guidelines for User-Defined Operators

- Adhere to mathematical properties
  - foo==foo Reflexivity
  - foo==bar $\Leftrightarrow$ bar==foo Symetry
  - foo==bar $\wedge$ bar==foobar $\Rightarrow$ foo==foobar Transitivity
  - foo!=bar $\Leftrightarrow$!(foo==bar)
- Consider mathematical laws
  - Associativity
  - Commutativity
- Consider typical shortcut operations
  - foo+=bar $\Leftrightarrow$ foo=foo+bar

Agenda

- Administrative Issues
  - The name of the game
  - Schedule
  - Lectures, Exercises, Exam
- C++ Highlights
  - Templates
  - User-Defined Types
- The Standard Library
  - Basic Input and Output
  - Strings
  - Containers
  - Iterators

The Standard Library

- Basic Input & Output
- Strings
- Containers
- Iterators
- A good overview of the classes and its API can be found here: http://www.cppreference.com/

No significant program is written in just a bare programming language.

Bjarne Stroustrup

Basic Input & Output

- File streams <fstream>
  - ifstream, ofstream, fstream
- String streams <sstream>
  - istringstream, ostringstream, stringstream
- Functions
  - >>, get, getline, ignore, read, gcount
  - <<, write

In- & Output (cont‘d)

```
int main(int argc, char *argv[]) {
  char first[256], mi, street[256];
  string last, city;

  // read a word terminated by whitespace
  cout << "First Name? "; cin >> first;          // suspect
  cout << "Middle Initial? "; cin >> mi;         // ok
  cout << "Last Name? "; cin >> last;          // ok

  // read a line terminated by newline
  cin.ignore(256,'\n');                        // ok
  cout << "Address? "; cin.get(street,256);    // suspect
  cout << "City? "; getline(cin,city,'\n');    // ok

  cout << "Hello, " << first << "!" << endl;
  cout << "I like " << city << "!" << endl;
}
```

Unix to DOS Converter

- This code converts Unix newline convention (LF) to DOS conventions (CR,LF)
- The program exhibits a common oversight
- In the output file, all space are missing
- `operator>>` by default skips whitespaces
- Typically, this is what is wanted

```
#include <iostream>
#include <fstream>

int main(int argc, char *argv[])
{
  ifstream ifs(argv[1]);
  ofstream ofs(argv[2]);
  char c;

  for (;;) {
    ifs >> c;
    if (!ifs.good()) break;
    if (c=='\n') ofs << "\r\n";
    else ofs << c;
  }
}
```

Unix to DOS Converter II

- The `istream::get()` member functions reads the next character without skipping whitespaces

```
#include <iostream>
#include <fstream>

int main(int argc, char *argv[])
{
  ifstream ifs(argv[1]);
  ofstream ofs(argv[2]);
  char c;

  for (;;) {
    ifs.get(c);
    if (!ifs.good()) break;
    if (c=='\n') ofs << "\r\n";
    else ofs << c;
  }
}
```

String Functions

```
#include <string>

…

int main(int argc, char *argv[]) {
  string h("Hello"), w("World"), s;
  w.append("!");
  cout << w[5] << w.at(5) << endl;
  w.insert(w.begin(),
           h.rbegin(), h.rend());
  cout << w << endl;
  w.replace(4,2,"H W");
  cout << w << endl;
  cout << "Enter your name: ";
  getline(cin, s);
  cout << h << " " << s << endl;
  return 0;
}
```

- s1.append(s2)
  Append s2 to s1
- s1.insert(pos, it1, it2)
  Insert sequence from it1 to it2 in string s1 at pos
- s1.replace(idx, len, s2)
  Replace len characters in s1 starting from idx with s2
- getline(is, s)
  Read a line from input stream is into string s

Container

- Provide different data structures
- Abstraction of a memory area
- User-defined allocators
- Can be accessed using iterators

Containers (Overview)

- Sequence Containers
  - `vector<T>, list<T>, forward_list<T>, deque<T>`
- Associative Containers
  - `map<K,V>, multimap<K,V>`
  - `set<K>, multiset<K>`
  - `unordered_map<K,V>, unordered_multimap<K,V>`
  - `unordered_set<K>, unordered_multiset<K>`
- Sequence Container Adapters
  - `stack<T,C>`
  - `queue<T,C>, priority_queue<T,C>`

Complexity Guarantees

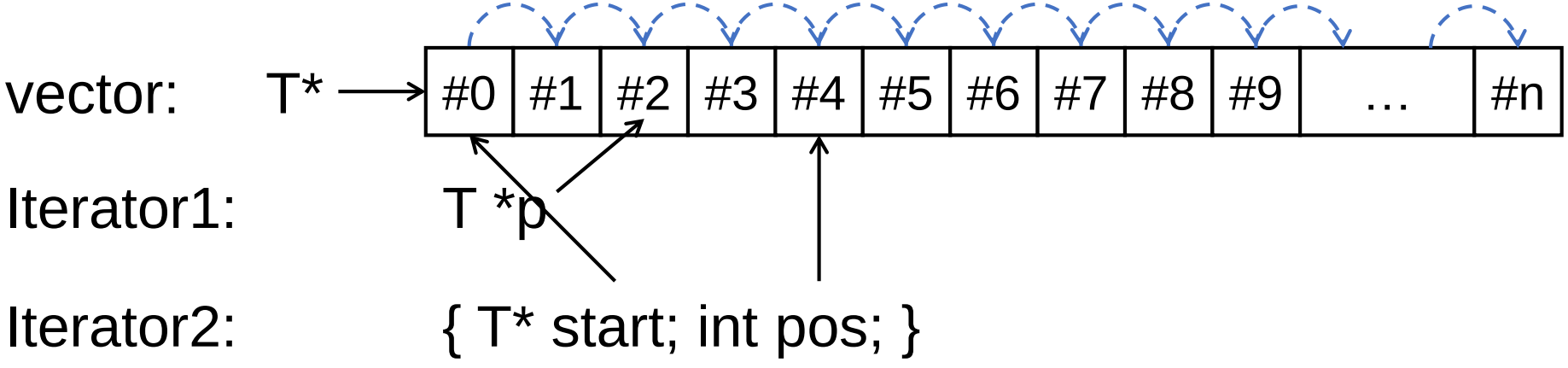
	[ ]	Insert/Remove Elements			Iterator
		Begin	Middle	End	
vector	O(1)		O(n)+	O(1)+	Ran
list		O(1)	O(1)	O(1)	Bi
forward_list		O(1)	O(1)		For
deque	O(1)	O(1)	O(n)	O(1)	Ran
map	O(log(n))		O(log(n))+		Bi
unordered_map	O(1)+		O(1)+		For
string	O(1)	O(n)+	O(n)+	O(n)	Ran

A Simple vector Example

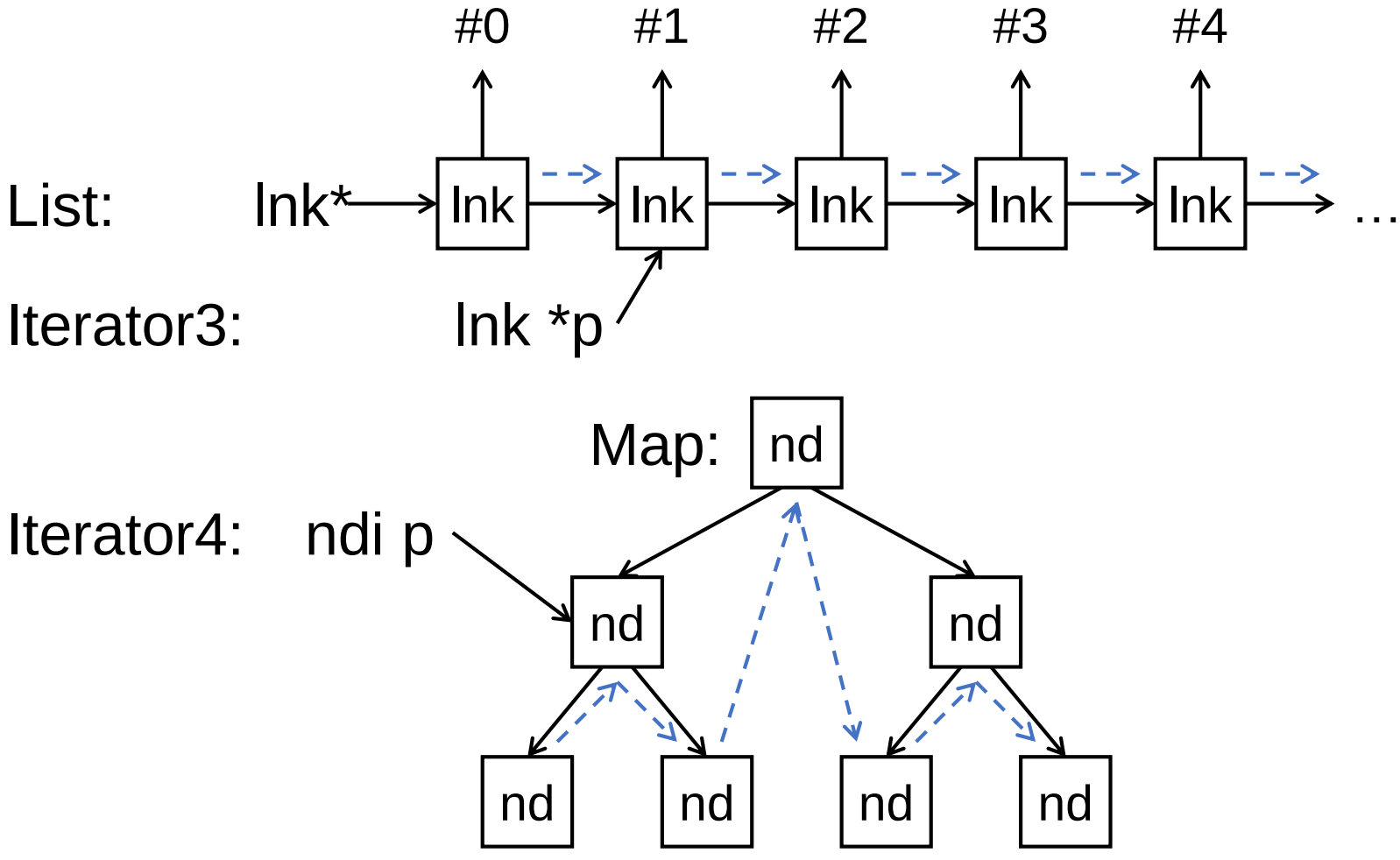
```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

int main(int argc, char *argv[]) {
  vector<string> playlist;
  playlist.push_back("1. Have A Nice Day");
  playlist.push_back("2. I Want To Be Loved");
  playlist.push_back("3. Welcome To Wherever You Are");
  playlist.push_back("4. Who Says You Can't Go Home");
  playlist.push_back("5. Last Man Standing");

  ...
```

Iterators

- Define a sequence over the elements in a container



More on Iterators (cont'd)



Java Trivia

```
... // store playlist in ArrayList playlist
BufferedReader in=new BufferedReader(new InputStreamReader(
            System.in));
ListIterator i=playlist.listIterator();
String song="none";
for (;;) {
  System.out.println("playing "+song);
  String cmd=in.readLine();
  if (cmd.equals("quit")) break;
  if (cmd.equals("next") && i.hasNext()) {
    song=(String)i.next();
  } else if (cmd.equals("prev") && i.hasPrevious()) {
    song=(String)i.previous();
  } else {
    System.out.println("unknown command "+cmd);
  }
}
```

Java Trivia

```
... // store playlist in ArrayList playlist
BufferedReader in=new BufferedReader(new InputStreamReader(
            System.in));
ListIterator i=playlist.listIterator();
String song="none";
for (;;) {
  System.out.println("playing "+song);
  String cmd=in.readLine();
  if (cmd.equals("quit")) break;
  if (cmd.equals("next") && i.hasNext()) {
    song=(String)i.next();
  } else if (cmd.equals("prev") && i.hasPrevious()) {
    song=(String)i.previous();
  } else {
    System.out.println("unknown command "+cmd);
  }
}
```

Constantly going back and forth in the playlist will repeatedly play the same song

Iterators the C++ Way

```
vector<string> playlist;
... // store playlist in ArrayList playlist
vector<string>::iterator fst=playlist.begin(),
                         cur=fst,
                         lst=playlist.end();
string cmd;
for (;;) {
  if (cur==lst) cout << "playing none" << endl;
  else cout << "playing " << *cur << endl;
  cin >> cmd;
  if (cmd=="quit") break;
  if (cmd=="next" && cur!=lst) ++cur;
  else if (cmd=="prev" && cur !=fst) --cur;
  else cout << "unknown command " << cmd << endl;
}
```

Different Types of Iterators

Category	output	input	forward	bi-directional	random-access
Abbrev.	Out	In	For	Bi	Ran
Read		=*p	=*p	=*p	=*p
Access		->	->	->	-> []
Write	*p=		*p=	*p=	*p=
Iteration	++	++	++	++ --	++ + += -- - -=
Compare		== !=	== !=	== !=	== < <= != > >=

Iterators (Java vs. C++)

Java

- Iterator obtained using iterator(), a ListIterator using listIterator()
- Iterator points between elements
  => current element cannot be accessed multiple times
- Iterator knows about first and last position

  => inflexible since routines cannot be easily changed to iterate over a part of the container
- If bounds are exceeded, an exception is raised

C++

- Iterator returned through begin() and end() or rbegin() and rend()
- Iterator points to the element
  => current element can be returned multiple times
- Iterator does not know about begin and end; second iterator is used for this purpose
  => flexibility as side effect; routine can iterate over any segment of the container
- If bounds are exceeded, undefined behavior, probably a “segmentation violation”

Iterators (Java vs. C++)

C++

- Iterator points to the elements (green)
- Repeatedly going forward and backward gives alternating elements

Java

- Iterator points between the elements (red)
- Repeatedly going forward and backward gives the same element

Summary

- Administrative Issues
  - The name of the game
  - Schedule
  - Lectures, Exercises, Exam
- C++ Highlights
  - Templates
  - User-Defined Types
- The Standard Library
  - Basic Input and Output
  - Strings
  - Containers
  - Iterators

Exercise 0 – Some Pointers

- First install the C++ Compiler
  - Install the following packages from www.cygwin.com:
    shells/bash, devel/gcc-g++, devel/make
- Eclipse Users
  - Go to www.eclipse.org and download
    Eclipse IDE for C/C++ Developers
  - In the past, eclipse/eclipsec should have been run from your cygwin shell
    (to ensure that paths are set up correctly)

Exercise 1

- Implement and run a “Hello World”

Exercise 2

- Implement the fraction type
  - Implement the +, -, *, / operators
  - Implement the << and >> operators
  - Provide two test drivers
    - One that checks based on a few samples that your code is correct
    - One that interactively lets a user invoke some operations with fraction numbers

Next Lecture

- Classes
- Templates

Happy Coding and See You Next Thursday…