C++ Introductory Tutorial

Based on Borland's World of C++ Video Course

16/10/95

Contents

- WWW Usefull Entry Points
- Bibliography
- Slides from a C++ course at SSC Laboratory by I.Sheer
- Course exercise book

WWW usefull entry points

AIPS++ Home Page (Astronomical Information...)

Astronomical Information Processing System C++ software to process (primarily radio) astronomical data, developed at an international consortium of observatories;

http://info.cv.nrao.edu/aips++/docs/html/aips++.html

CASE TOOL PAGE

This page is devoted to Freeware or Shareware CASE tools available on the Web for IBM PCs running MS Windows.

http://osiris.sunderland.ac.uk/sst/casehome.html

WWW Courses and slides:

C++ Annotations

WWW course on C++ for C user's
Frank B. Brokken and Karel Kubat (frank@icce.rug.nl, karel@icce.rug.nl)
ICCE, State University of Groningen
Westerhaven 16, 9718 AW Groningen
Netherlands

http://www.icce.rug.nl/docs/cplusplus/cplusplus.html

C++ Programming Language Tutorial Handouts

Pointer to handouts for C++ coureses.
Also other interesting pointers to OO places.

http://www.cs.wustl.edu/~schmidt/C++/index.html

LEARN C/C++ TODAY

Listing of available C/C++ tutorials and book on C++.
You find here almost everything usefull available on

the WWW.

http://vinny.csd.mu.edu/learn.html

Learning C++ (21-Nov-1994)

Learning C++ page from the C++ Virtual Library.

http://info.desy.de/user/projects/C++/Learning.html

CERN STING information service

Software Technology Interest Group home page at CERN

http://dxsting.cern.ch/sting/sting.html

Mike Stump's Home Page at Cygnus

Hot news about C++ ANSI standard and GNU g++

http://www.cygnus.com/~mrs/

OMG Home Page

The Object Management Group (OMG) is a non-profit consortium dedicated to promoting the theory and practice of object technology (OT) for the development of distributed computing systems.

http://www.omg.org/

Object-Oriented Information Sources

Search engine with predefined queries an a lot of usefull pointers maintained by the
University of Geneva (Ch)

http://cuiwww.unige.ch/ooinfo?info-page

Object Port Transport

Main pointer for OO links

http://www.s2k.com/trans.htm

OSE - Home Page

OSE is a collection of programming tools and class libraries targeted mainly at C++ programmers.
There are three major components in OSE. These are the C++ class libraries (OTCLIB/OUXLIB/OTKLIB), a build environment (makeit) and a set of documentation extraction tools (class2man/class2mml).
http://www.telstra.com.au/docs/ose/doc/ose-home.html

Quadralay C++ Archive

One of the best sources of pointers on OO
http://www.quadralay.com:80/www/CCForum/CCForum.html

SIGS Publications Home Page

SIGS in one of the major publishers in the OO worls.
http://www.sigs.com/cgi-bin/imagemap/menumap?219,46

The C++ Virtual Library

One of the main entry points for OO links
http://info.desy.de/user/projects/C++.html

The Object-Oriented Page

You will find here a lot of links to object-oriented info, as well as comments on object-oriented books, products, object databases, articles, etc.
http://www.einet.net/galaxy/Engineering-and-Technology/
Computer-Technology/Object-Oriented-Systems/
ricardo-devis/oo.html

WWW Virtual Library - Software Engineering

Main entry point on OO links, handled by NASA.
A lot of infos on conferences, organizations, standards.
http://rbse.jsc.nasa.gov/virt-lib/soft-eng.html

Welcome to Rational Software Corporation

Rational is where Booch and Rumbaugh work.

They produce Rose and many other products.

http://www.rational.com/

Bibliography

To learn C++:

C++ Inside Out
Eckel, B. - McGrawHill, 1995

C++ Primer, 2nd edition
Lippman S., Addison Wesley, 1991

C++ Reference books/manuals

The C++ programming Language
Stroustrup B., Addison Wesley, 1991

The annotated C++ reference manual
Ellis M. Stroustrup B., Addison Wesley, 1990

C++ ANSI Standard (Draft focument)
ftp://research.att.com/dist/stdc++/WP/

C++ Advanced books

The Design and Evolution of C++
Stroustrup B., Addison Wesley, 1994

C++ Programming Styles and Idions
Coplien J., Addison Wesley, 1992

Object Oriented Design

Object Oriented Analysis and Design with Applications
G.Booch, Addison Wesley, 1994

Object-Oriented Modeling and Design With Applications
Rumbaugh J. et al., Prentice Hall, 1992

Objectifying Real-Time Systems
Ellis J.R., SIGS Books, 1994
Object Oriented Software Construction
Meyer B., Prentice Hall, 1989

Lesson 17

```
// CAR.CPP : basic composition
// From Borland's The World of C++, Lesson 17
// (C) Copyright 1991 by Borland International
class wheel {
  int diameter;
public:
  wheel(int d) { diameter = d; }
};

class engine {
  int horsepower;
public:
  engine(int hp) { horsepower = hp; }
};

class car {
  wheel lf, rf, lr, rr;
  engine e;
  int passengers;
public:
  car(int dia, int hp, int pas);
};

car::car(int dia, int hp, int pas) :
  lf(dia), rf(dia), lr(dia), rr(dia), e(hp) {
  passengers = pas;
}

main() {
}
```

```
// CAR2.CPP : pseudo constructor calls in the constructor
// initializer list for car.
// From Borland's The World of C++, Lesson 17
// (C) Copyright 1991 by Borland International

class wheel {
  int diameter;
public:
  wheel(int d) { diameter = d; }
};

class engine {
  int horsepower;
public:
  engine(int hp) { horsepower = hp; }
};

class car {
  wheel lf, rf, lr, rr;
  engine e;
  int passengers;
public:
  car(int dia, int hp, int pas);
};

car::car(int dia, int hp, int pas) :
  lf(dia), rf(dia), lr(dia), rr(dia), e(hp),
  passengers(pas) {
}
```

Lesson 18

```
// DOG.CPP
// From Borland's The World of C++, Lesson 18
// (C) Copyright 1991 by Borland International

#include <stdio.h>

class dog {
public:
  void speak() { puts("woof!"); }
};

void talk(dog * d) {
  d->speak();
}

main() {
  dog d;
  talk(&d);
}
```

```
// PET.CPP
// From Borland's The World of C++, Lesson 18
// (C) Copyright 1991 by Borland International
#include <stdio.h>

class pet {
public:
  void speak() {}
};

class dog : public pet {
public:
  void speak() { puts("woof!"); }
};

class cat : public pet {
public:
  void speak() { puts("murroww!"); }
};

class bird : public pet {
public:
  void speak() { puts("cheep!"); }
};

void talk(dog * d) {
  d->speak();
}

main() {
  dog d;
  talk(&d);
}
```

```
// PET2.CPP : don't write C++ programs this way!
// From Borland's The World of C++, Lesson 18
// (C) Copyright 1991 by Borland International

#include <stdio.h>

enum pet_type { DOG, CAT, BIRD };

class pet {
  pet_type ptype;
public:
  pet(pet_type pt) : ptype(pt) {}
  void speak() {}
  // ask it what type it is:
  pet_type type() { return ptype; }
};

class dog : public pet {
public:
  dog() : pet(DOG) {}
  void speak() { puts("woof!"); }
};

class cat : public pet {
public:
  cat() : pet(CAT) {}
  void speak() { puts("murroww!"); }
};

class bird : public pet {
public:
  bird() : pet(BIRD) {}
  void speak() { puts("cheep!"); }
```

```
};

void talk(pet * p) {
  switch(p->type()) {
    case DOG  : ((dog*)p)->speak();  break;
    case CAT  : ((cat*)p)->speak();  break;
    case BIRD : ((bird*)p)->speak(); break;
    default   : puts("oops!");   break;
  }
}

main() {
  dog d;
  cat c;
  bird b;
  talk(&d);
  talk(&c);
  talk(&b);
}
```

Lesson 19

```
// VPET.CPP : pets speak properly with late binding
// From Borland's The World of C++, Lesson 19
// (C) Copyright 1991 by Borland International

#include <stdio.h>

class pet {
public:
  // virtual causes late binding:
  virtual void speak() {}
  virtual void sit() {}
};

// talk uses the common interface to pet:
void talk(pet * p) {
  p->speak();
}

class dog : public pet {
public:
  void speak() { puts("woof!"); }
};

class cat : public pet {
public:
  void speak() { puts("murroww!"); }
};

class bird : public pet {
public:
```

```
  void speak() { puts("cheep!"); }
};

main() {
  dog d;
  talk(&d);  // This still works...
}
```

```
// VPET2.CPP : A menagerie of pets
// From Borland’s The World of C++, Lesson 19
// (C) Copyright 1991 by Borland International

#include <stdio.h>

class pet {
public:
  virtual void speak() {}  // virtual causes late binding
  virtual void sit() {}
};

// talk uses the common interface to pet:
void talk(pet * p) {
  p->speak();
}

// can also use references:
void talk2(pet& p) {
  p.speak();
}

class dog : public pet {
public:
  void speak() { puts(“woof!”); }
};

class cat : public pet {
public:
  void speak() { puts(“murroww!”); }
};

class bird : public pet {
```

```
public:
  void speak() { puts("cheep!"); }
};

main() {
  pet* menagerie[] = {
    new dog, new cat, new bird
  };
  const sz =
    sizeof(menagerie)/sizeof(menagerie[0]);
  for(int i = 0; i < sz; i++) {
    talk(menagerie[i]);
    talk2(*menagerie[i]);
  }
}
```

```
// SIZES.CPP : sizes of objects with virtual functions
// From Borland's The World of C++, Lesson 19
// (C) Copyright 1991 by Borland International

#include <stdio.h>

class no_virtual {
  int i;
public:
  void f();
  void g();
};

class simple_base : public no_virtual {};

class one_virtual {
  int i;
public:
  virtual void f();
  void g();
};

class virtual_in_base :
  public one_virtual {};

class two_virtuals {
  int i;
public:
  virtual void f();
  virtual void g();
};

class virtual_in_base2 :
```

```
  public two_virtuals {};

#define SIZE(arg) \
  printf("sizeof(" #arg ")= %d\n",sizeof(arg))

main() {
  SIZE(no_virtual);
  SIZE(simple_base);
  SIZE(one_virtual);
  SIZE(virtual_in_base);
  SIZE(two_virtuals);
  SIZE(virtual_in_base2);
  void * voidpointer;
  SIZE(voidpointer);
}
```

Lesson 20

```
// VPET2.CPP : A menagerie of pets
// From Borland's The World of C++, Lesson 20
// (C) Copyright 1991 by Borland International

#include <stdio.h>

class pet {
public:
  // virtual causes late binding:
  virtual void speak() {}
};

class dog : public pet {
public:
  void speak() { puts("woof!"); }
};

class cat : public pet {
public:
  void speak() { puts("murroww!"); }
};

class bird : public pet {
public:
  void speak() { puts("cheep!"); }
};

// talk uses the common interface to pet:
void talk(pet * p) {
  p->speak();
```

```
}

// can also use references:
void talk2(pet& p) {
  p.speak();
}

main() {
  pet* menagerie[] = { new dog, new cat, new bird };
  const sz = sizeof(menagerie)/sizeof(menagerie[0]);
  for(int i = 0; i < sz; i++) {
    talk(menagerie[i]);
    talk2(*menagerie[i]);
  }
}
```

```
// VPET3.CPP : pets with an abstract base class
// From Borland's The World of C++, Lesson 20
// (C) Copyright 1991 by Borland International

#include <stdio.h>

class pet {
public:
  // pure virtual function:
  virtual void speak() = 0;
};

class dog : public pet {
public:
  void speak() { puts("woof!"); }
};

class cat : public pet {
public:
  void speak() { puts("murroww!"); }
};

class bird : public pet {
public:
  void speak() { puts("cheep!"); }
};

// talk uses the common interface to pet:
void talk(pet * p) {
  p->speak();
}

// can also use references:
```

```
void talk2(pet& p) {
  p.speak();
}

main() {
  pet* menagerie[] = { new dog, new cat, new bird };
  const sz = sizeof(menagerie)/sizeof(menagerie[0]);
  for(int i = 0; i < sz; i++) {
    talk(menagerie[i]);
    talk2(*menagerie[i]);
  }
}
```

```
// VPET4.CPP : Extending the pet system
// From Borland's The World of C++, Lesson 20
// (C) Copyright 1991 by Borland International

#include <stdio.h>

class pet {
public:
  // pure virtual function:
  virtual void speak() = 0;
};

class dog : public pet {
public:
  void speak() { puts("woof!"); }
};

class cat : public pet {
public:
  void speak() { puts("murroww!"); }
};

class bird : public pet {
public:
  void speak() { puts("cheep!"); }
};

// Extending the system by adding a new class:
class hamster : public pet {
public:
  void speak() { puts("squeak!"); }
};
```

```
// talk uses the common interface to pet:
void talk(pet * p) {
  p->speak();
}

// can also use references:
void talk2(pet& p) {
  p.speak();
}

main() {
  pet* menagerie[] = { new dog, new cat, new bird, new hamster };
  const sz = sizeof(menagerie)/sizeof(menagerie[0]);
  for(int i = 0; i < sz; i++) {
    talk(menagerie[i]);
    talk2(*menagerie[i]);
  }
}
```

Lesson 21

```
// OP_OVL.CPP : Simple operator overloading in C++
// From Borland's The World of C++, Lesson 20
// (C) Copyright 1991 by Borland International

class point {  // a point in 2-D space
  int x, y;
public:
  point(int X, int Y) { x = X; y = Y; }
  point operator=(point rv) {
    x = rv.x;
    y = rv.y;
    return *this; // a copy of this object
  }
  point operator+(point arg) {
    return point(x + arg.x, y + arg.y);
  }
};

main() {
  point A(9,12), B(47,14), C(0,0);
  C = A + B;
}
```

```
// MULTIPLE.CPP: simple multiple inheritance
// From Borland's The World of C++, Lesson 21
// (C) Copyright 1991 by Borland International

class storable { // stores itself on disk
public:
  storable(char* filename);
  virtual void store();
};

class elevator {
  int floor_selected;
  int floor_number;
public:
  elevator(int starting_floor);
  void go(int floor);
};

class SElevator  // storable elevator
        : public storable, public elevator {
public:
  SElevator(char * fn, int sfloor) :
    storable(fn), elevator(sfloor) {}
  // The meaning of store() probably changes:
  void store();
};
```