Practical Programming Methodology
(CMPUT-201)

Michael Buro

Lecture 17

- Virtual Functions Continued
- C++ Library: string, streams
- Casts

Abstract Base-Class Example

```
class Shape {  // abstract base class
public:
  int color;
  virtual void draw(Screen *s) const = 0; //abstract
};  // = 0: derived classes must implement function

class Circle : public Shape {
public:
  Circle() { x = y = r = 0; }
  void draw(Screen *s) const { ... } // implements
                                     // virtual function
private:
  int x,y,r;
}
```


Virtual Function Syntax and Semantics

- Default implementation in base-class:
  `virtual <type> <func>(<params>) { ... }`
- Signals the compiler to create a virtual function table and to add a virtual function pointer to each object that derives from this class
- Abstract virtual function: derived classes must provide implementation
  `virtual <type> <func>(<params>) = 0;`
- The presence of abstract virtual functions marks class as being abstract
- Abstract classes can't be instantiated
  (e.g. `Shape x;` or `new Shape;` is illegal)

Virtual Destructors

```
class X {
public:
  X() { ... }
  ~X() { ... } // should have been virtual!
  virtual void foo() { ... }
};

class Y : public X {
public:
  Y() { ... }
  ~Y() { ... }
  virtual void foo() { ... }
};

X *px = new Y; // calls Y() - which calls X() first - OK
px->foo();     // calls Y::foo() - OK
delete px;     // only calls ~X(), but not ~Y(). PROBLEM!
```

Requires destructors in base-class to be declared virtual!

Example

```
class Shape {
  virtual void draw(Screen *s) = 0;
  virtual ~Shape() { }
};

class Circle    : public Shape { ... };
class Rectangle : public Shape { ... };
class Square    : public Shape { ... };

Shape *a[3];

// allocate shapes (constructor args. omitted)
a[0] = new Circle; a[1] = new Rectangle; a[2] = new Square;

// draw all shapes using their respective draw functions
for (int i=0; i < 3; ++i) a[i]->draw(screen);

// delete all shapes using their respective destructors
for (int i=0; i < 3; ++i) delete a[i];
```


Inheritance Tips

- Declare destructors virtual in the presence of other virtual member functions. g++ will remind you.
- Base-class copy constructors are not automatically called in derived class copy constructors you provide (use `: X(...)`)
- In the derived class assignment operator call base-class X operator explicitely: `X::operator=(source);`
- Virtual function calls in base-class constructors call base-class functions! Derived class functions can't be called this way, because derived class objects haven't been initialized yet.

C++ Library (1): strings

- C++ template class (more on templates later): `typedef basic_string<char> string;`
- Plugin replacement for C-strings
- Can grow/shrink
- Can hold characters with code 0
- Many useful public member functions
- C-strings should be avoided in C++ programs. Use `string` instead!

Examples

```
#include <string>
using namespace std;

string a;
string b("foo");        // foo
string c = b + b;       // foofoo

size_t l = c.length(); // 6
bool   e = c.empty();  // false

char x = s[0];          // element access by index

if (b < c) { ...        // comparison (lexicographic ordering)

c.insert(0, "x");       // xfoofoo
c.erase(0, 4);          // erase interval [begin, end) -> foo
string::size_type ind = c.find("o");
                        // index of first occurrence: 1
                        // string::npos if not found
```

C++ Library (2): Stream Hierarchy

How to design output routines that work on a variety of output media such as files and strings?

Solution: pass a reference or pointer to an output base-class object to functions or operators. Virtual output functions in the derived class object can then be accessed.

Example: C++ standard library output streams

- `class ostream` : output stream base-class
- `class ofstream : public ostream`
  Output is directed to a file
- `class ostringstream : public ostream`
  Output is accumulated in string

Example

```
#include <iostream>
#include <fstream>  // file-stream classes
#include <sstream>  // string-stream classes
using namespace std;

struct Foo
{
  void write(ostream &os = cout) { os << ...; }
};

Foo x;

x.write();      // write to cout
x.write(cout);  // write to cout

ofstream of("file");
if (!of) {      // error ...
x.write(of);    // write to file

ostringstream oss;
x.write(oss);  // write to string-stream

cout << oss.str() << endl; // write string-stream contents to cout
```


C++ Casts

`static_cast`

- Used for standard conversions
- Compile-time operator – no run-time check
- `int i; double d; i = static_cast<int>(d);`
- Do not use for up/down-casts – see `dynamic_cast`

`reinterpret_cast`

- Conversion from one pointer type to any other pointer type
- Can also convert pointers to ints and vice versa
- `int a[100]; char *p = reinterpret_cast<char*>(a);`
- Dangerous! Avoid! Can result in unportable code

const_cast

- `const_cast` changes status: const ↔ non-const
- Used for changing unessential private data members in const functions (can also be accomplished by `mutable`)

```
class X {
public:
  int get_index() const;

private:
#ifndef NDEBUG
  int getn; // counter
#endif
};
```

```
int X::get_index() const {
#ifndef NDEBUG
  // increment counter,
  // preserve constness
 const_cast<X*>(this)->getn++;
#endif

  // get_index body
  ...
}
```

Alternative Implementation

```
class X {
public:
  int get_index() const;

private:
#ifndef NDEBUG
  mutable int getn; // mutable counter
  // "mutable" indicates that this variable
  // can be changed in const functions
#endif
```

```
int X::get_index() const {
#ifndef NDEBUG
  // increment counter
  ++getn;
#endif
  ...
```


dynamic_cast

```
class X { virtual void foo(); };  // VFTP added

class Y : public X { };

void bar(X *p) {
  Y *q = dynamic_cast<Y*>(p);
  assert(q != 0); // q == 0 iff *p is neither
                  // of type Y nor derived from Y
}
```

- Useful for down-casting pointers (trying to treat them as derived class pointer). This usually indicates a broken design! Use virtual functions!
- Only works with polymorphic types (with VFTP)
- Run-time check (slows down program)
- Returns 0 if cast is illegal, and pointer to object of correct type otherwise