SimCenter NHERI
Center for Computational Modeling and Simulation

Programming Bootcamp

Abstraction & Object Oriented Programming (OOP)

Frank McKenna
University of California at Berkeley

NSF award: CMMI 1612843

Abstraction

The goal of "**abstraction**" is to reduce complexity by removing unnecessary information.

float
integer
double
char

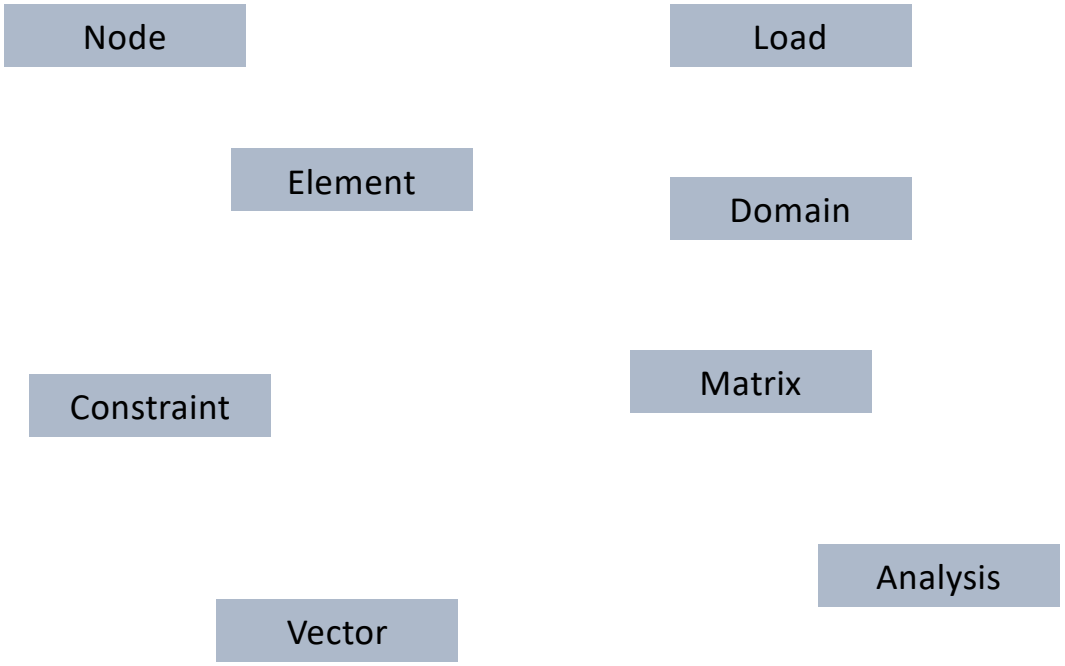
Point
Node
Element
…

Abstraction

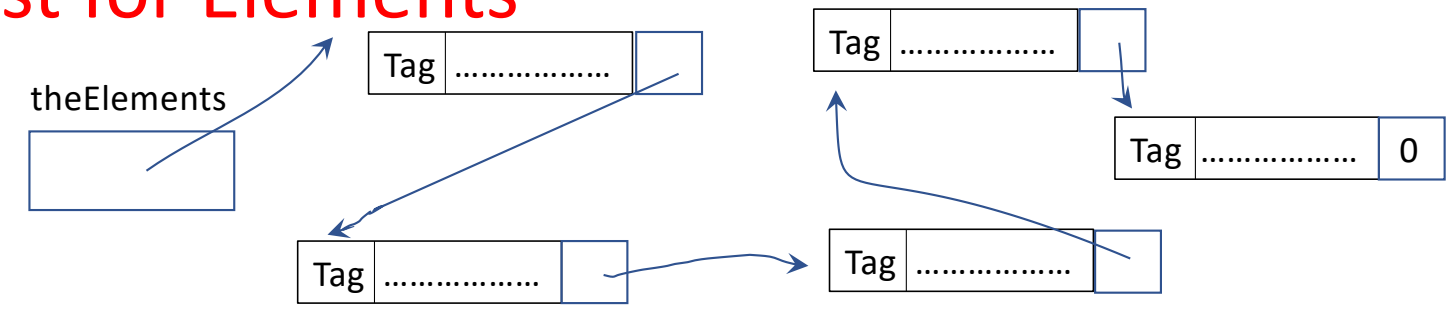
- Abstraction simplifies many aspects of computing and makes it possible to build complex systems. We would not be where we are today if all programmers were still writing code in 0's and 1's

- Computing Languages Provide Programmers the Ability to create abstractions. Higher Level Languages provide more abstraction capabilities (albeit at expense of performance) than lower level languages.

What about Abstractions for a Finite Element Application?



Linked List for Elements



theElements – pointer to an Element *,
each Element has a pointer to another Element or NULL if last

c/fem/domain2.h

```
#include “domain.h”
#include ”node.h”
typedef (int)(*elementStateFunc)(Domain *theDomain, double *k, double *P);

typedef struct element {
  int tag;
  int nProps, nHistory;
  int *nodeTags;
  double *paramaters;
  double *history;
  elementStateFunc eleState;
  struct element *next;
} Element;
```

We are starting to use a struct to hold both the data and a list of function pointers to point to the relevant functions for that data.

Each struct now holds the data for the object and the functions that work on the data.

We are beginning to program in an object oriented paradigm within C.

Approaches to Building Manageable Programs

PROCEDURAL DECOMPOSITION

Divides the problem into more easily handled subtasks, until the functional modules (procedures) can be coded

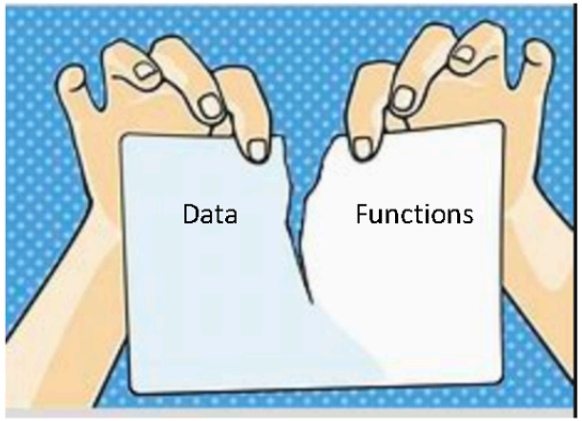
FOCUS ON: procedures

OBJECT-ORIENTED DESIGN

Identifies various objects composed of data and operations, that can be used together to solve the problem

FOCUS ON: data objects

Difficulty Programming in OOP with C



Tying object data and object functions together is a chore

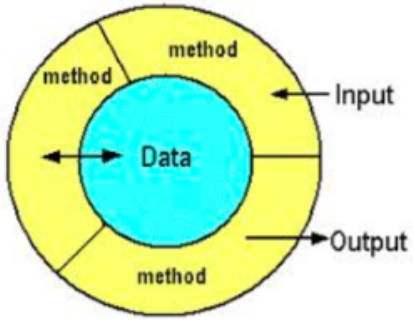
Solution C++

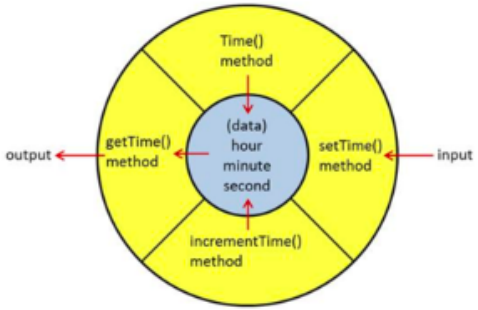
The C++ Programming Language

- Developed by Bjourne Stroustroup working at Bell Labs (again) in 1979. Originally “C With Classes” it was renamed C++ in 1983.
- A general purpose programming language providing both functional and object-oriented features.
- As an **incremental upgrade** to C, it is both strongly typed and a compiled language.
- The updates include:
  - Object-Oriented Capabilities (**encapsulation, inheritance, and polymorphism**)
  - Standard Template Libraries
  - Additional Features to make C Programming easier!

Encapsulation

Encapsulation performs the operation of data hiding. Data and functions are bundled together into what Is called a class. The class provides an interface to the outside world. It hides the data and implementation from the outside world. If written correctly only the class can access the data. The functions or other class objects in the program can only query the functions presented through the interface, the methods.

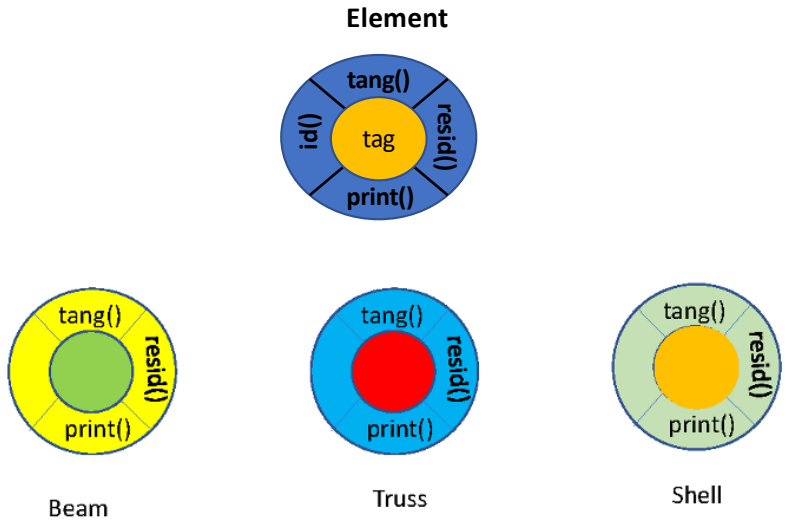




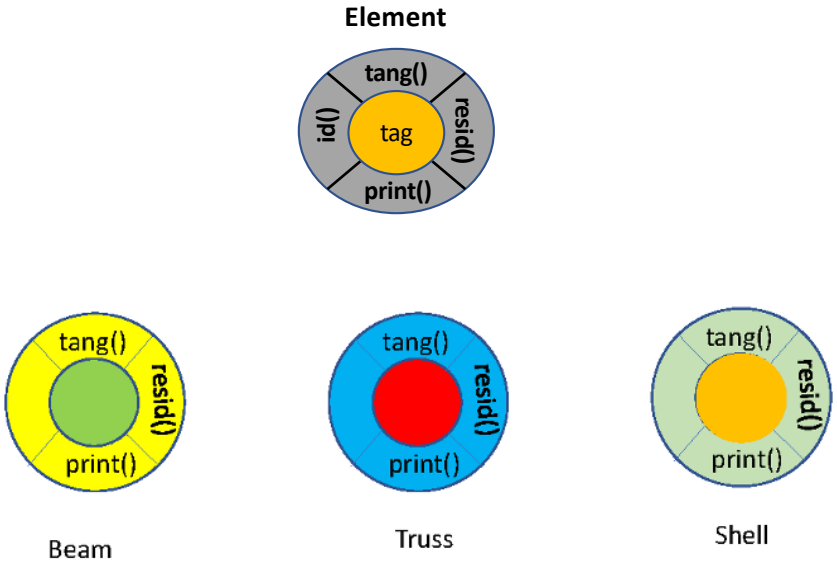
Inheritance & Polymorphism

Polymorphism is the ability to treat objects of different types in a similar manner, e.g. an Analysis object looping over elements to form a stiffness matrix and call the method tang() without knowing or caring what the element actually is, i.e. Beam, Truss, and Shell elements are all the same to it.

Inheritance occurs when a class derives the methods and data of a parent class. The derived class or subclass can overwrite the parent classes methods and add new data, thus enabling polymorphism.



Terminology



Element is Parent Class

Truss, Beam, Shell are Child or subclasses Classes of Element

Truss, Beam, Shell are derived from Element

Truss, Beam, Shell override methods tang(), print() and resid() methods of Element

Truss, Beam, Shell inherit interface, data, e.g. tag, and methods, e.g. id() of Element

Class refers to the implementation

In the running program Objects are created of a specific class type.

SimCenter NHERI
Center for Computational Modeling and Simulation

Programming Bootcamp

C++ Language

Frank McKenna
University of California at Berkeley

NSF award: CMMI 1612843

C++ Program Structure

A ~~C~~ C++ Program consists of the following parts:

- Preprocessor Commands
- Functions
- Variables
- Statements & Expressions
- Comments
- **Classes**

Hello World in C++

Code/C++/hell01.cpp

```
#include <iostream>
int main(int argc, char **argv) {
  // my first C++ program
  std::cout << "Hello World! \n";
  return 0;
}
```

Code/C/hell01.c

```
#include <stdio.h>
int main(int argc, char **argv) {
   /* my first program in C */
   printf("Hello World! \n)
   return 0;
}
```

pointers, new() and delete()

Code/C++/memory1.cpp

```
#include <iostream>
int main(int argc, char **argv) {

  int n;
  double *array1, *array2, *array3;
  std::cout << "enter n: ";
  std::cin >> n;

  // allocate memory & set the data
  array1 = new double[n];
  for (int i=0; i<n; i++) {
    array1[i] = 0.5*i;
  }
  array2 = array1;
  array3 = &array1[0];
  for (int i=0; i<n; i++, array3++) {
    double value1 = array1[i];
    double value2 = *array2++;
    double value3 = *array3;
    printf("%.4f %.4f %.4f\n", value1, value2, value3);
  }
  // free the array
  delete [] array1;
}
```

Gone is the cast

strings

Code/C++/string1.cpp

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

int main(int argv, char **argc) {
  std::string pName = argc[0];
  std::string str;
  std::cout << "Enter Name: ";
  std::cin >> str;


  if (pName == "./a.out")
    str += " the lazy sod";

  str += " says ";
  str = str + "HELLO World";
  std::cout << str << "\n";

  return 0;
}
```

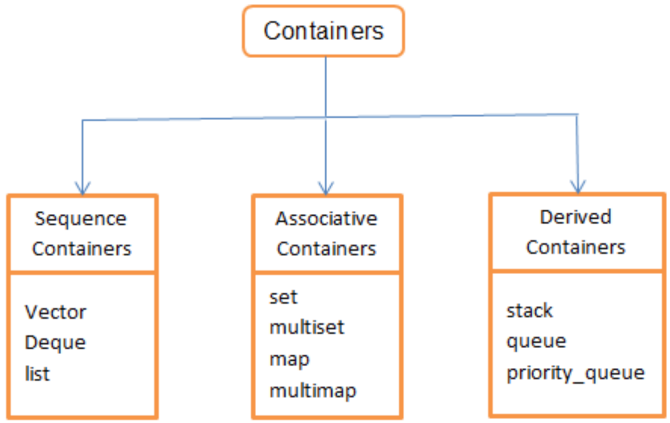
Pass by reference

Code/C++/ref1.cpp

```
#include <iostream>

void sum1(int a, int b, int *c);
void sum2(int a, int b, int &c);


int main(int argc, char **argv) {
  int x = 10;
  int y = 20;
  int z;
  sum1(x,y, &z);
  std::cout << x << " + " << y << " = " << z << "\n";

  x=20;
  sum2(x, y, z);
  std::cout << x << " + " << y << " = " << z << "\n";
}
// c by value
void sum1(int a, int b, int *c) {
  *c = a+b;
}


// c by ref
void sum2(int a, int b, int &c) {
  c = a + b;
}
```

STL Library

The Standard Template Library (STL) is a set of C++ template classes to provide common programming data structures and functions such as lists, stacks, arrays, etc. It is a library of container classes, algorithms, and iterators.



Will hold off on an example for now

Class

A class in C++ is the programming code that defines the methods (defines the api) in the class interface and the code that implements the methods. For classes to be used by other classes and in other programs, these classes will have the **interface in a .h** file and the **implementation in a .cpp** (.cc, .cxx) file

SimCenter NHERI
Center for Computational Modeling and Simulation

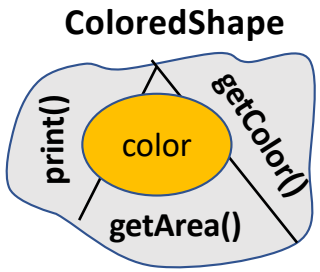
Programming Bootcamp

C++ Class Example

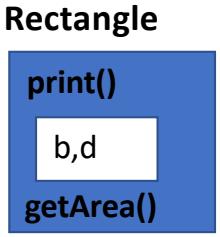
Frank McKenna
University of California at Berkeley

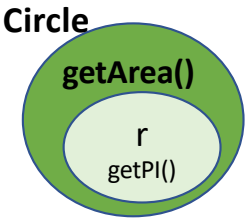
NSF award: CMMI 1612843

Class Example for Colored Shapes



Circle will be parent class of all shapes
It will also be an "ABSTRACT" class, i.e. no objects of it can be instantiated





Circle will be a subclass of ColoredShape
Circle will inherit print() and getColor() Methods
Circle will override getArea() method
Private data r, private method getR()

Programming Classes – header file (ColoredShape.h)

```
#ifndef _COLORED_SHAPE
#defne _COLORED_SHAPE

#include <string>
using std::string;

class ColoredShape {
  public:
    ColoredShape(string color);
    virtual ~ColoredShape();
    virtual double getArea(void) =0;
    virtual void print(ostream &s);
    const string &getColor(void);

  private:
      string color;
};

#endif // _COLORED_SHAPE
```

keyword **class** defines this as a class, **ColoredShape** is the name of the class

Classes can have 3 sections:

Public: objects of all other classes and program functions can invoke this method **on the object or** ~~**see this data**~~

Protected: only objects of subclasses of this class can invoke this method or see this data.

Private: only objects of this specific class can invoke the method.

virtual double getArea(void) = 0 , the =0;makes this an abstract class. (It cannot be instantiated.) It says the class does not provide code for this method. A subclass must provide the implementation.

virtual void print(ostream &s) the class provides an implementation of the method, the **virtual** a subclass may also provide an implementation.

Const string &getColor() is a method provided to return color. Not virtual so no subclass can override.

virtual ~Shape() is the destructor. This is method called when the object goes away either through a delete or falling out of scope.

Rectangle.h (in blue)

```
class ColoredShape {
   public:
    ColoredShape(string color);
    virtual ~ColoredShape();
    virtual double getArea(void) =0;
    virtual void print(ostream &s);
};
```

```
class Rectangle: public ColoredShape {
 public:
  Rectangle(string color, double w, double h);
  ~Rectangle();
  double getArea(void);
  void print(std::ostream &s);

 protected:

 private:
  double width, height;
  static int numRect;
};
```

- **class Rectangle: public Shape** defines this as a class, **Rectangle** which is a subclass of the class Shape.
- It has 3 sections, public, protected, and private.
- It has a constructor **Rectangle(string color, double w, double h)** which states that class takes 3 args (color, w and h) when creating an object of this type.
- It also provides the methods double getArea(void) and void
- printArea(ostream &s); Neither are virtual which means no subclass can provide an implementation of these methods.
- In the private area, the class has 3 variables. Width and height are unique to each Rectangle num rect is static, which means it is shared amongst all objects of type Rectangle.

Programming Classes – source file (ColoredShape.cpp)

```
#include <ColoredShape.h>
ColoredShape::ColoredShape(string col)
:color(col)
{
}

ColoredShapeShape::~ColoredShape() {
  std::cout << "Shape Destructor\n";
}
const string &
ColoredShape::getColor(void)[
   return color;
}
void
Shape::print(std::ostream &s) {
  s << "UNKOWN shape: color ” << color
<< “, area: " << this->getArea() << "\n";
}
```

- Source file contains the implementation of the class.
- 4 methods provided. The constructor Shape(), the destructor ~Shape(), getColor() and the print() method. A definition is provided for each method not designated pure virtual that is defined in the header file.
- The Destructor just sends a string to cout.
- The print() methods prints out the color and area. It obtains the area by invoking the **this** pointer.
- **This pointer is not defined in the .h file or .cpp file anywhere as a variable. It is a default pointer always available to the programmer. It is a pointer pointing to the object itself.**

Rectangle.cpp

```
int Rectangle::numRect = 0;

Rectangle::~Rectangle() {
  numRect--;
  std::cout << "Rectangle Destructor\n";
}

Rectangle::Rectangle(string color, double w, double d)
  :ColoredShape(color), width(w), height(d)
{
  numRect++;
}

double
Rectangle::getArea(void) {
  return width*height;
}
void
Rectangle::print(std::ostream &s) {
  s << "Rectangle: color " this->getColor() << width * height
     <<       " numRect: " << numRect << "\n";
}
```

- **int Rectangle::numRect = 0** creates the memory location for the classes static variable numRect.
- The **Rectangle::Rectangle(double w, double d) is the class constructor taking 2 args.**
- the line **:Shape(), width(w), height(d)** is the first code exe. It calls the base class constructor and then sets it's 2 private variables.
- The constructor also increments the static variable in **numRect++;** That variable is decremented in the **destructor.**
- **The GetArea()** method, which computes the area can access the private data variables height and width

Main Program (main1.cpp)

C++/shape/main1.cpp

```
# #include "Rectangle.h"
#include "Circle.h"

int main(int argc, char **argv) {
  Rectangle s1(1.0, 2.0, “red”);
  ColoredShape *s2 = new Rectangle(3.0,1.0,”blue”);

  s1.printArea(std::cout);
  s2->printArea(std::cout);

  delete s2;
  return 0;
}
```

When we run it, results should be as you expected. Notice the destructors for s1 is not called. It was not created using new but the destructor was invoked when went out of scope (which is why printed after s2 destructor).
The destructor for s2 is explicitly invoked in the delete s2 statement.
Note that the base class destructor is invoked before the parent class constructor.

```
build >make
[ 66%] Built target ShapeLib
[100%] Built target ex1
build >./ex1
Rectangle: color: red, area: 2 numRect: 2
Rectangle: color: blue, area: 3 numRect: 2
Rectangle Destructor blue
ColoredShape Destructor
Rectangle Destructor red
ColoredShape Destructor
build >
```

Circle.h

```
class ColoredShape {
  public:
    ColoredShape(string color);
    virtual ~ColoredShape();
    virtual double getArea(void) =0;
    virtual void print(ostream &s);
};
```

```
#ifndef _CIRCLE
#define _CIRCLE
class Circle: public ColoredShape {
 public:
  Circle(double r, string color);
  ~Circle();
  double getArea(void);

 private:
  double diameter;
  double getPI(void);
};
#endif // _CIRCLE
```

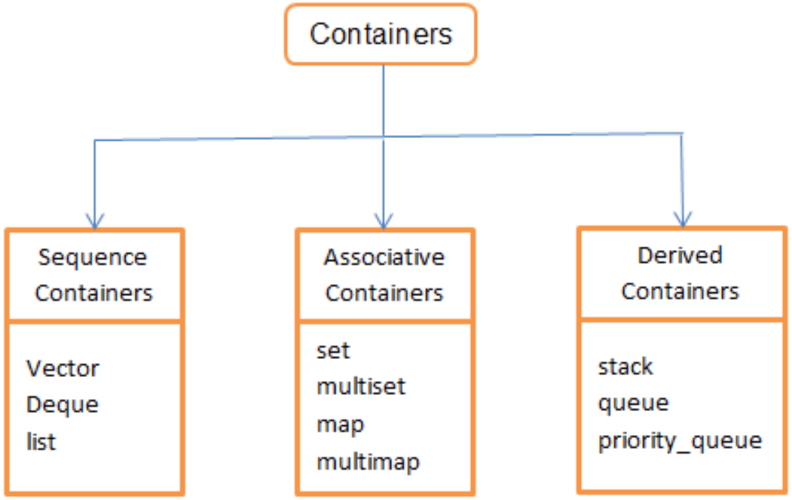
- **class Circle: public Shape** defines this as a class **Circle** which is a subclass of the class Shape.
- It has 2 sections, public and private.
- It has a constructor **Circle(double d)** which states that class takes 1 arg d when creating an object of this type.
- It also provides the method **double getArea(void).**
- **There is no print() method, meaning this class relies on the base class implementation.**
- In the private area, the class has 1 variable and defines a private method, GetPI(). Only objects of type Circle can invoke this method.

Circle.cpp

```
#include <Circle.h>

Circle::Circle(string color, float radius)
:ColoredShape(color), r(radius) {
 }

Circle::~Circle() {
  std::cout << "Circle Destructor\n";
}

double
Circle::getArea(void) {
  return this->getPI() * r * r;
}

double
Circle::getPI(void) {
  return 3.14159;
}
```

STL Library

The Standard Template Library (STL) is a set of C++ template classes to provide common programming data structures and functions such as lists, stacks, arrays, etc. It is a library of **container classes**, algorithms, and **iterators**.



Main Program with STL Container

C++/shape/main3.cpp

```
#include "Rectangle.h"
#include "Circle.h"
#include <list>
int main(int argc, char **argv) {
  std::list<ColoredShape*> theShapes;

  Circle s1(2.0, “red”);
  ColoredShape *s2 = new Rectangle(1.0, 2.0, “blue");
  ColoredShape *s3 = new Rectangle(3.0,2.0, “green”);

  theShapes.push_front(&s1);
  theShapes.push_front(s2);
  theShapes.push_front(s3);

  std::list<ColoredShape *>::iterator it;
  for (it = theShapes.begin();  it != theShapes.end(); it++) {
    (*it)->PrintArea(std::cout);
  }
  return 0;
}
```

typedef to the rescue

C++/shape/main4.cpp

```
#include "Rectangle.h"
#include "Circle.h"
#include <list>
#include <vector>
typedef std::list<ColoredShape*> Container;
typedef Container::iterator Iter;

int main(int argc, char **argv) {
  Container theShapes;
  Circle s1(2.0, “red”);
  ColoredShape *s2 = new Rectangle(1.0, 2.0, “red”);
  ColoredShape *s3 = new Rectangle(3.0,2.0, “green”);
  theShapes.push_front(&s1);
  theShapes.push_front(s2);
  theShapes.push_front(s3);

  Iter it;
  for (it = theShapes.begin(); it != theShapes.end(); it++) {
    (*it)->PrintArea(std::cout);
  }
  return 0;
}
```