Classes

C structures with

Methods

Public, private, and protected members

Inheritance

The header file dice.h

```
class Dice
{
  public:
    Dice(int sides);         // constructor
    int Roll();              // return the random roll
    int NumSides() const;    // how many sides
    int NumRolls() const;    // # times this die rolled

  private:
    int myRollCount;         // # times die rolled
    int mySides;             // # sides on die
};
```

Using the Dice class

```
#include <iostream>
using namespace std;
#include ”dice.h”
int main()
{
   Dice cube(6);
   Dice dodeca(12);

   cout << ”cube rolled a ” << cube.Roll() << endl;

   for(int k=0; k < 6; k++)
   {
         cout << ”dodeca just rolled a ” << dodeca.Roll() << endl;
   }
   cout << endl;
   cout << ”cube rolled ” << cube.NumRolls() << ” times” << endl;
   cout << ”dodeca rolled ” << dodeca.NumRolls() << ” times” << endl;
   return 0;
}
```

The implementation file dice.cpp

```
Dice::Dice(int sides)
// postcondition: all private fields initialized
{
    myRollCount = 0;
    mySides = sides;
}

int Dice::NumSides() const
// postcondition: return # of sides of die
{
    return mySides;
}

int Dice::NumSides() const
// postcondition: return # of sides of die
{
    return mySides;
}

int Dice::Roll()
// postcondition: number of rolls updated
//                random 'die' roll returned
{
    RandGen gen;    // random number generator (“randgen.h”)

    myRollCount= myRollCount + 1;    // update # of rolls
    return gen.RandInt(1,mySides);   // in range [1..mySides]
}
```

Constructors and Destructors

Constructors invoked on
- Entry to a scope
- Call to new (by default)

Destructors invoked on
- Exit from scope
- Call to delete (by default)

Simple constructors

```
class IntLink {
public:
  int element;       // Value for this node
  IntLink *next;         // Pointer to next node in list
  IntLink(const int& elemval, IntLink* nextval =NULL)
    { element = elemval;  next = nextval; }
  IntLink(IntLink* nextval =NULL) { next = nextval; }
};
```

Constructors can be used to initialize
Destructors are rarely needed

`new` and `delete`

Used in allocation of pointer-accessed classes

```
P = new IntLink() ;
P = new int[1000] ;
```

Example of a class in need of a destructor

```
class Person
{
    public:
        Person(char const *n, char const *a,
               char const *p);
        ~Person();

        char const *getName() const;
        char const *getAddress() const;
        char const *getPhone() const;

    private:
        // data fields
        char *name;
        char *address;
        char *phone;
};
```

The constructor

```
Person::Person(char const *n, char const *a, char const *p)
{
    name    = strdupnew(n);
    address = strdupnew(a);
    phone   = strdupnew(p);
}
```

The destructor

```
Person::~Person()
{
    delete name;
    delete address;
    delete phone;
}
```

OK in Java, not OK in C++

```
for(i = 0; i < 1000000 ; ++i) {
     char *c = new char[1000] ;
     Do some stuff!
}
```

By the way, in this case you need

```
delete[] c ;
```