# Polymorphism: a Case Study

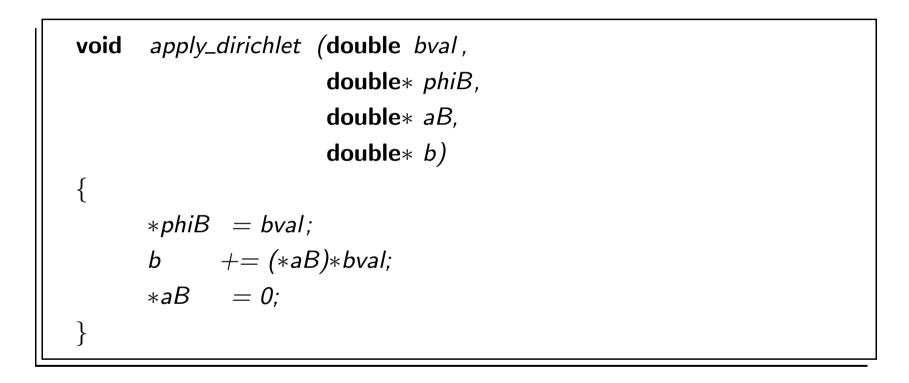
Jan van Dijk

#### Outline

Polymorphism in C++: a case study

- ► In transport equations: boundary conditions
- ► We create code for handling boundary conditions
- ► A *base class* contains the commonalities
- Derived classes implement specialties
- Polymorphism: in the base class we declare an interface (without implementing it). That is enough to use the condition.

## A Dirichlet Functions



### The disadvantages of this approach

Look at the usage:

Using functions:

- In the code we must pass these parameters every time;
- Functions cannot remember their context, because:
- Functions do not have data members;

## A Dirichlet Class

```
class BoundCondDirichlet
 public :
   BoundCondDirichlet(
             double bval,
             double* aB,
             double* b)
       m_bval = bval;
       m_aB = aB;
       m_b = b;
```

```
double Apply(double PhiP)
{
    m_b += (*m_aB)*m_bval;
    *m_aB = 0;
    return m_bval;
}
protected :
    double m_bval;
    double m_bval;
    double m_aB;
    double* m_aB;
```

};

The Neumann class is similar.

#### Discussion of the Dirichlet Class

BoundCondDirichlet\* dirbc = **new** BoundCondDirichlet( /\* params \*/) // ... phiE = dirbc->Apply(phiP)

- Creation is more difficult (one time)
- ► Usage is simple
- ▶ In the usage we hardly see the details ('this is Dirichlet')

Disadvantage: We do not use the similarity of Dirichlet and Neumann. Trick:

- 1. Introduce base class for common code (coefs, ...)
- 2. Declare abstract interface (without implementation)

#### The base class

```
class BoundCond
public :
 BoundCond( double delta,
             double* aP,
             double* aB,
             double* b);
  virtual double Apply(double)=0;
protected :
 double m_delta,
 double* m_aP,
 double* m_aB,
 double* m_b;
```

- Expresses what a BC is (it has a member Apply)
- ► All we need to *use* BC's
- Shared by Dirichlet & Neumann
- Boundary conditions can be derived
- Can use everything in base class
- These must provide an appropriate *Apply*

## Dirichlet, implemented as derived class

```
class BoundCondDirichlet : public BoundCond
 public :
      BoundCondDirichlet(double bval, double delta,
                         double* aP, double* aB, double* b)
       : BoundCond(delta, aP, aB, b) {
              m_bval=bval;
      }
     double Apply(double phiP) {
       m_b += (*m_aB)*m_bval;
       *m_{a}B = 0;
        return m_bval;
 protected :
     double m_bval;
};
```

#### Advantages

Dirichlet and Neumann are both *BoundCond*-like classes. We can write:

BoundCond\* bc = **new** BoundCondDirichlet(...);

(Where we *create* the BC, we must know the type)

BUT:

BoundCond\* bc = CreateWhateverBC(name, ...); phiB = bc->Apply(phiP);

Where we use the condition, we only see the base class.

Implementation of core code: independent of details.

Only the base class interface matters.