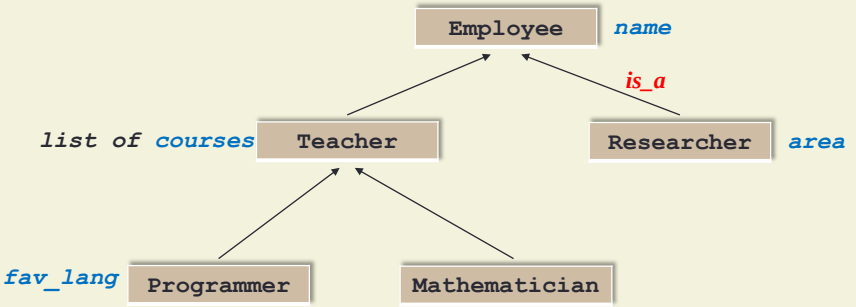
Lecture 10

- **Inheritance** *(arv)* [sec. 9]
  - Base class, derived class (*härledda classer*) [sec 9.1]
- Constructors and inheritance [sec. 9.2]
- Destructors and inheritance [sec. 9.3]
- Members accessibility [sec. 9.4]
  - **`private`**, **`public`**, **`protected`**
- Example
  - **`Employee`**'s hierarchy of classes

Inheritance

- To implement **`is_a`** relationship
- Used in software engineering to create systems that can be easily extended



See **`arv1.cpp`**

Base class and derived class

```
class Employee
{
  public:
     explicit Employee(string s = "");
     void set_name(const string s);
     void display() const;
  private:
     string name;
};
```

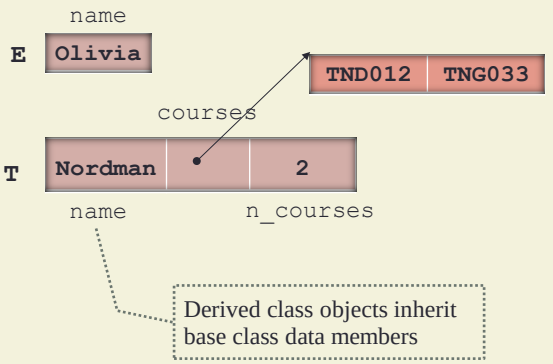
Base class
Super class

```
class Teacher : public Employee
{
  public:
    Teacher(string s = "", string *c = nullptr, int n = 0);
    bool gives_course(string code) const;
    void display() const;
    …
  private:
     string *courses; //an array with course codes
     int n_courses;
};
```

Derived class of **Employee**
Subclass **Teacher**
Teacher is_a Employee

Objects of base/derived class

```
Employee E("Olivia");
string course_codes[] = {"TND012", "TNG033"};
Teacher T("Nordman", course_codes, 2);
```



Derived class objects inherit base class data members

Derived class member functions

```
Employee E("Olivia");
string course_codes[] = {"TND012", "TNG033"};
Teacher T("Nordman", course_codes, 2);
```

```
E.display();
if ( T.gives_course("TNG033") ) …;
T.set_name("Vitória");
T.display();
```

Member function of **Teacher**

Member function inherited from base class (**Employee**)

Defined in class **Employee**

Redefined in class **Teacher**

Hides base class member function (**Employee::display()**)

Derived class member functions

```
void Teacher::display() const
{
   cout << "Name: " << name << endl;
   cout << "Courses: ";
   //display course codes
   …
}
```

Compilation error:
if **name** is a private data member of the base class (**Employee**)

```
void Teacher::display() const
{
   //call display function from class Employee
   Employee::display();
   cout << "Courses: ";
   for(int i = 0; i < n_courses; i++)
          cout << courses[i] << " ";
}
```

Access to the hidden member function of the base class

Derived class member functions

- A derived class (e.g. **`Teacher`**) inherites members from the base class
  - Private data (function) members are not accessible
    - E.g. **`name`** is not accessible in class **`Teacher`**
  - **`set_name()`** inherited member function available for class **`Teacher`**
  - A member function of the derived class can have te same name as a member function of the base class
    - E.g. **`Teacher::display()`**
- Constructors, destructors, **`operator=`**, and **`friend`**s are not inherited

Members accessibility

- Control accessibility of class members outside the class
  - **`public`**, **`private`**, **`protected`**

```
class Employee {
  public:
     …
     void display() const;
  protected:
     string name;
};
```

Can be accessed by any member function of a derived class

```
class Teacher: public Employee
{
  public:
     …
     void display() const;
  protected:
     string *courses;
     int n_courses;
};
```

Members accessibility

```
void Teacher::display() const
{
   //Employee::display();
   cout << "Teacher: " << name << endl;
   cout << "Courses: ";
   //display the courses
}
```

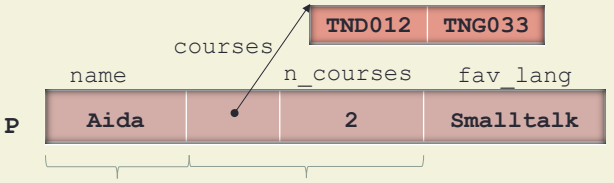


```
class Teacher : public Employee
{
  public:
    Teacher(string s = "", string *c = nullptr, int n = 0);
    bool gives_course(string code);
    void display() const;
    …
  protected:
     string *courses;
     int n_courses;
};
```

```
class Programmer : public Teacher
{
  public:
    Programmer(string s = "", string *c = nullptr,
               int n = 0, string f = "C++");
    void display() const;
  protected:
     string fav_lang; //favorite prog. language
};
```

Derived class **Programmer**

```
string course_codes[] = {"TND012", "TNG033"};
Programmer P("Aida", course_codes, 2, "Smalltalk");
```



inherited from **Employee**

inherited from **Teacher**

Redefined in class **Programmer**

```
P.display();
if ( P.gives_course("TNG33") ) …;
P.set_name("Alex");
```

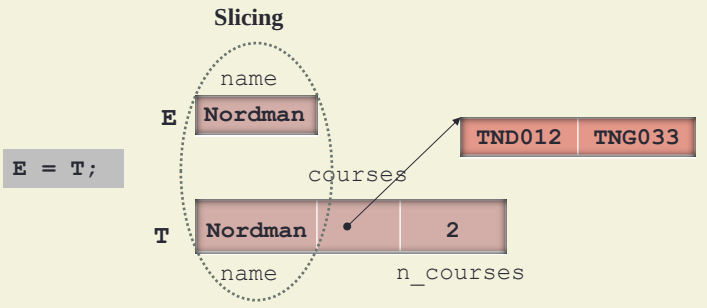
Inherited from class **Teacher**

Compiler climbs up the hierarchy …

Inherited from class **Employee**

Derived-class and base-class convertions

```
string course_codes[] = {"TND012", "TNG033"};
Employee E;
Teacher T("Nordman", course_codes, 2);
```



Derived-class and base-class conversions

```
string course_codes[] = {"TND012", "TNG033"};
Teacher T("Nordman", course_codes, 2);
Programmer P("Aida", course_codes, 2, "Smalltalk");
```

```
Employee *ptr_E = nullptr;
Teacher *ptr_T = &T;
```

```
ptr_E = ptr_T;
ptr_E = &P;
```

Pointer to the base class can point to an object of a derived class

`(D*)` is automatically converted to **`(B*)`**

```
ptr_E->display();
```

Calls **`Employee::display()`**

It's the type of **`ptr_E`** that decides which member function to call

Unless … more later…

Derived-class and base-class conversions

```
string course_codes[] = {"TND012", "TNG033"};
Teacher T("Nordman", course_codes, 2);
```

```
Employee *ptr_E = &T;  //points to a Teacher
Teacher *ptr_T = nullptr;
```

```
ptr_T = ptr_E;
```

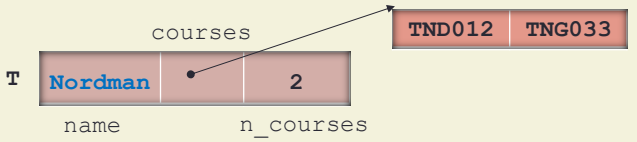
Pointer to the base class cannot be automatically converted to a pointer to an object of the derived class

`(B*)` is NOT automatically converted to **`(D*)`**

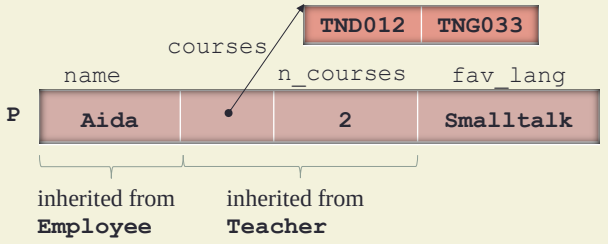
Compilation error!!

Constructors and inheritance

```
Teacher T("Nordman", course_codes, 2);
```




```
string course_codes[] = {"TND012", "TNG033"};
Programmer P("Aida", course_codes, 2, "Smalltalk");
```



Constructors and inheritance

```
class B {
  public:
    B(parameters);
    …
};
```

```
class D : public B {
  public:
    D(parameters);
    …
  protected:
    int n;
    …
};
```

```
D::D(parameters)
  : n(0), B(parameter)
{
   statements;
}
```

1. Call base class constructor
2. Perform other initializations in the initialization list
3. Execute `{ statements; }`

No call to base class constructor `B::B()`

Constructors and inheritance

```
class B {
  public:
    B( );
    …
};
```

Default constructor

```
class D : public B {
  public:
    D(parameter);
    …
  protected:
    int n;
    …
};
```

```
D::D(parameter)
{
  statements;
}
```

Default base class constructor **B::B()** is called first automatically, i.e. before *statements* are executed

Important: a default constructor **B::B()** must be provided!!

Read sec. 9.2

Constructors and inheritance

```
class B {
  public:
    B(parameter);
    …
};
```

No default constructor **B::B()** available

```
class D : public B {
  public:
    D(parameter);
    …
  protected:
    int n;
    …
};
```

```
D::D(parameter)
{
  statements;
}
```

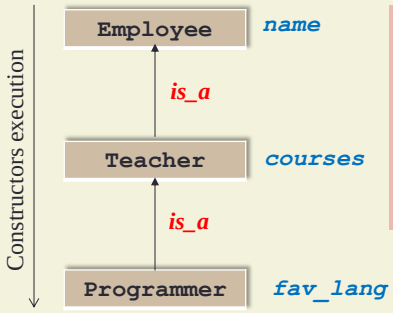
Compilation error!!

```
int main()
{
  D d(…);
  …
}
```

See **test.cpp**

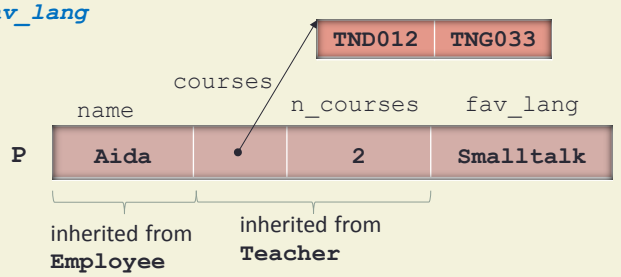
Important: a default constructor **B::B()** must be provided!!

Constructors and inheritance



1. Execute constructor for **Employee**
   - **name** is initialized
2. Execute constructor for **Teacher**
   - **courses** and **n_courses** are initialized
3. Execute constructor code for **Programmer**
   - **fav_lang** is initialized

Constructors are NOT inherited by derived classes



Copy constructor and inheritance

```
class D : public B {
  public:
    …
  protected:
    …;
};
```

Copy constructor of base class is never inherited by derived classes

Call copy constructor of base class to initialize the base class' data members

```
D::D(const D &d) : B(d)
{
  initialize D's specific data members;
}
```

Copy constructor generated by the compiler

```
D::D(const D &d) : B(d)
{  //shallow copy }
```

operator= and inheritance

```
class D : public B {
  public:
    …
  protected:
    …;
};
```

operator= of base class is never inherited by derived classes

Default **operator=** provided by the compiler

```
const D& operator=(const D &d)
{
   B::operator=(d);
   shallow copy of d's specific data members;
}
```

If the programmer defines an assignment operator then **operator=** of base class is NOT called automatically

Destructors and inheritance

```
//destructor
Teacher::~Teacher()
{
    delete [] courses;
   //Employee destructor is automatically called last
}
```

Last *step* in a derived class destructor **D::~D()** is to call the base class destructor **B::~B()**

Destructors and inheritance

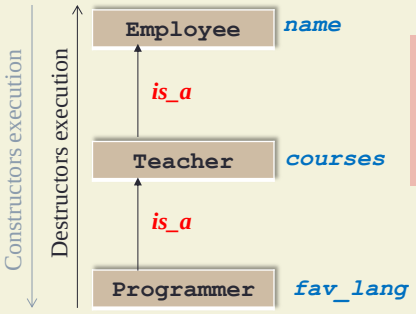
```
class B {
  public:
    ~B( );
    …
};
```

```
class D : public B {
  public:
    ~D( );
    …
  protected:
    …
};
```

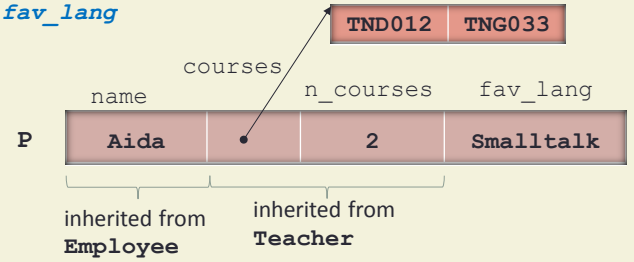
```
D::~D()
{
  destroy object;
}
```

Compiler always calls automatically **B::~B()** as last step of a derived class destructor

Destructors and inheritance



1. Execute destructor for **Programmer**
2. Execute destructor for **Teacher**
   - **courses** array is deallocated
3. Execute destructor for **Employee**



Read sec. 9.3

Next …

- Important to study the example in **`arv1.cpp`**
- **Fö 11**
  - Polymorphism and dynamic binding [sec. 9.6]
    - Virtual member functions
  - Virtual destructors [sec. 9.9]
  - Abstract classes [sec. 9.10]
- **Lab 3**
  - Hierarchy of **`Expression`**s
  - Polymorphism, dynamic binding, abstract classes