Compiler construction in4303 – lecture 10

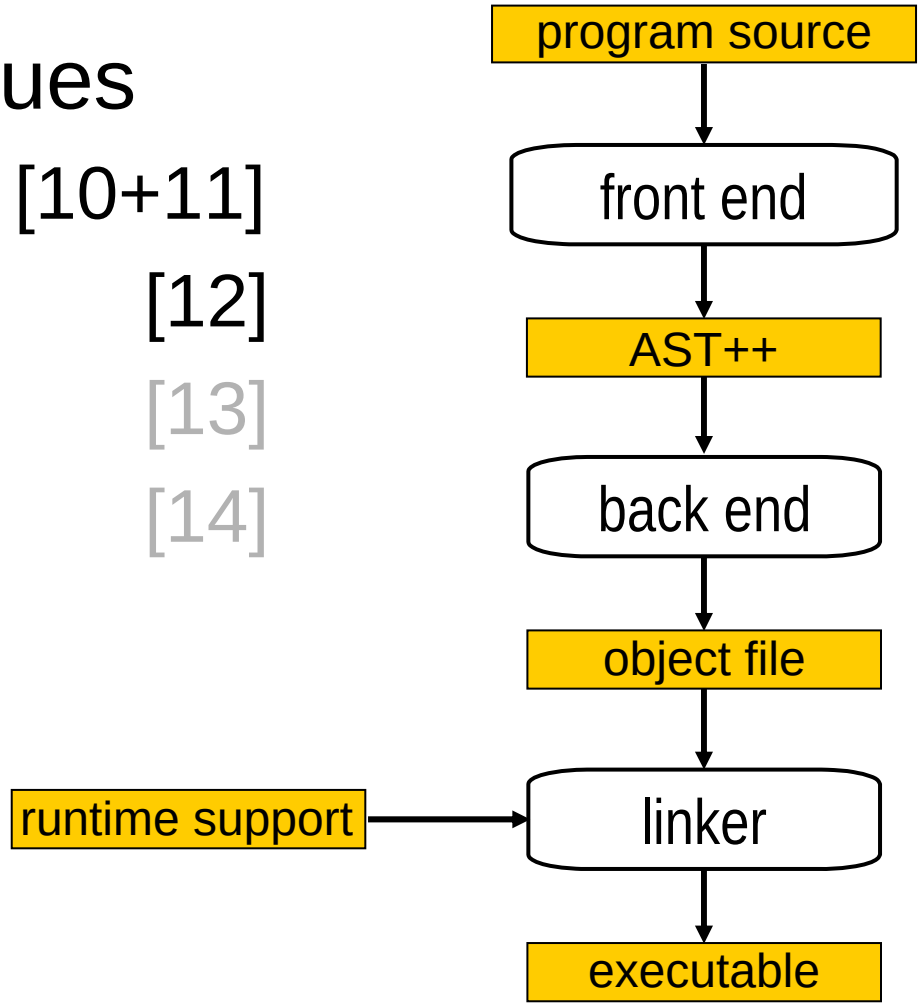
Imperative & OO languages: context handling

Chapter 6.2

Overview – remaining lectures

Language specific issues

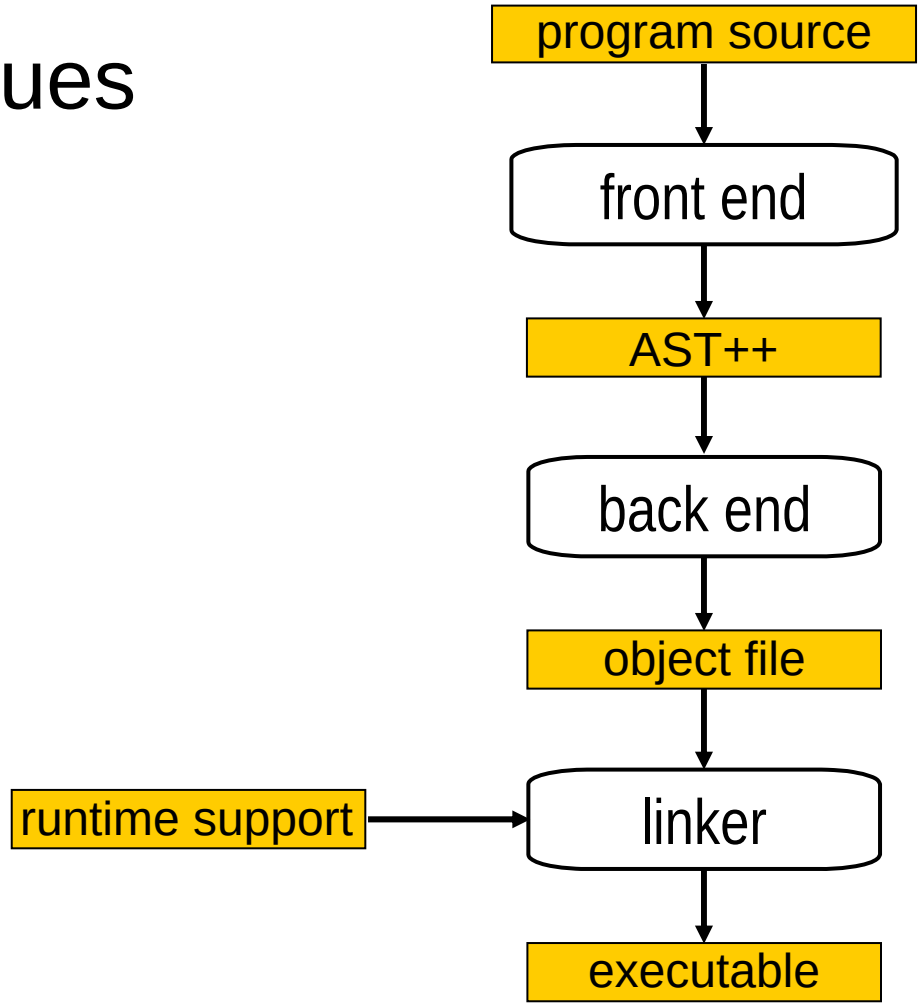
- imperative & OO [10+11]
- memory management [12]
- functional programs [13]
- logic programs [14]



Imperative & OO programs

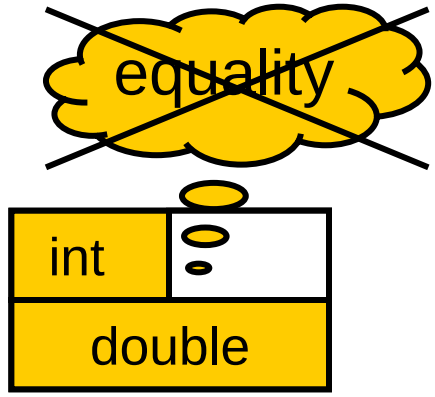
Language specific issues

- context handling
- code generation



Source language data representation and handling

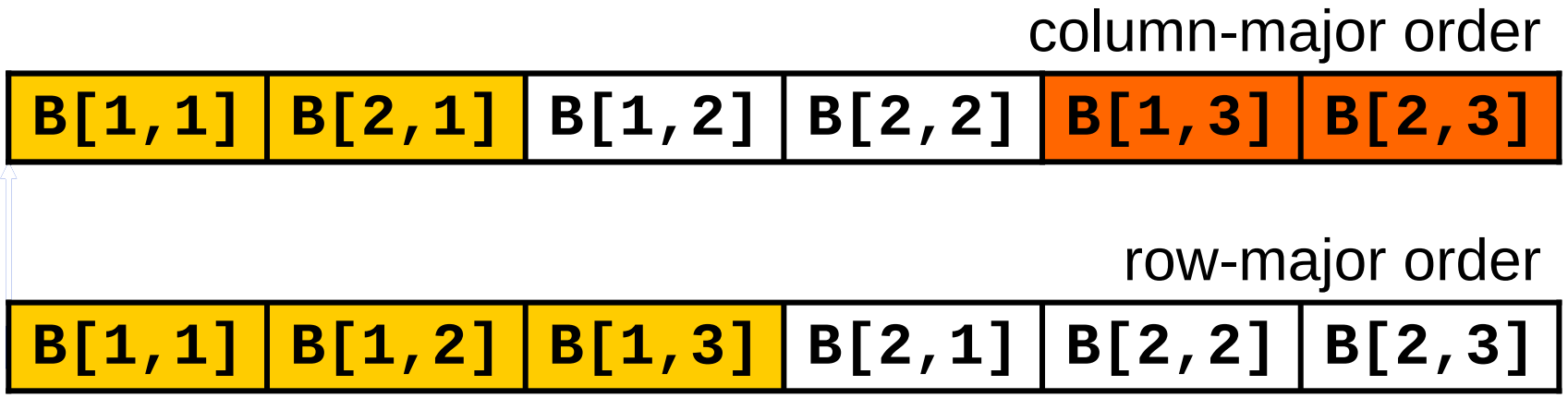
- basic types – map to target hardware
- enumeration types – map to integers
- set types – map to bitset
- record types – field alignment
- union types – union tag?
- pointer types – check scope rules
- array types
- object types
- routine types



Array types – data layout

Multi-dimensional arrays

- vector `A: ARRAY [1..4] OF Integer;`
- matrix `B: ARRAY [1..2, 1..3] OF Integer;`
- ...



Array types – indexing

Location(**B[i,j]**)

= base(**B**) + ((**i**–LB_1)×LEN_2 + **j**–LB_2) × size-of-int

where $LEN_k = UP_k - LB_k + 1$

zeroth element (pre computed)

= base(**B**) – (LB_1×LEN_2 + LB_2) × size-of-int

+ (**i**×LEN_2 + **j**) × size-of-int

B[1,1]	B[1,2]	B[1,3]	B[2,1]	B[2,2]	B[2,3]

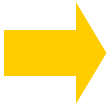
Object types

- data representation
- features
  - inheritance
  - method overriding
  - polymorphism
  - dynamic binding
  - (dependent) multiple inheritance

Object representation

C++

```
class A {
public:
    int a1, a2;

    void m1(int i) {
        a1 = i;
    }
    void m2(int i) {
        a2 = a1 + i;
    }
}
```



object data

a1
a2

method table

m1_A
m2_A

Object representation

C++

```
class A {
public:
   int a1, a2;

   void m1(int i) {
      a1 = i;
   }
   void m2(int i) {
      a2 = a1 + i;
   }
}
```

```
A a;


...
a.m1(3);
```

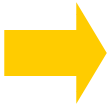
C

```
typedef struct {
   int a1, a2;
} class_A;


void m1_A(class_A *this, int i) {
   this->a1 = i;
}

void m2_A(class_A *this, int i) {
   this->a2 = this->a1 + i;
}
```

```
class_A a;


...
m1_A( &a, 3);
```

Inheritance

```
class B : public A {
public:
   int b1;

   void m3(void) {
      b1 = a1 + a2;
   }
}
```



object data

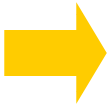
a1
a2
b1

method table

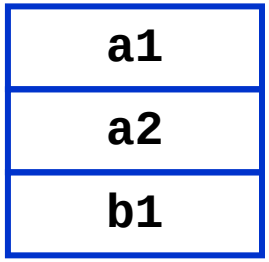
m1_A
m2_A
m3_B

Method overriding

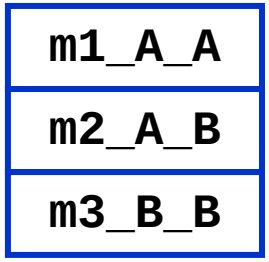
```
class B : public A {
public:
   int b1;

   void m3(void) {
      b1 = a1 + a2;
   }
   void m2(int i) {
      a2 = b1 + i;
   }
}
```

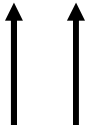


object data



method table





declared defined

Exercise (4 min.)

Give the method tables for **Rectangle** and **Square**

```
abstract class Shape {
   boolean IsShape() {return true;}
   boolean IsRectangle() {return false;}
   boolean IsSquare() {return false;}
   abstract double SurfaceArea();
}
class Rectangle extends Shape {
   double SurfaceArea() { ... }
   boolean IsRectangle() {return true;}
}
class Square extends Rectangle {
   boolean IsSquare() {return true;}
}
```

Answers

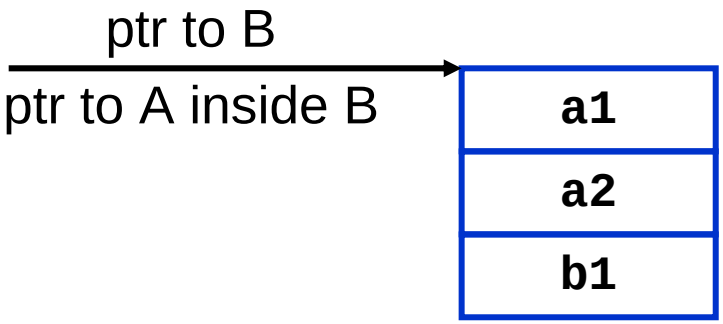
Polymorphism

- a pointer to class C may actually refer to an object of class C or any of its extensions

```
B *b = ...;
A *a = b;
```

```
class_B *b = ...;
class_A *a = convert_ptrB_to_ptrA(b);
```

- implementation requires pointer supertyping



Dynamic typing

- which method to invoke on overloaded polymorphic types?

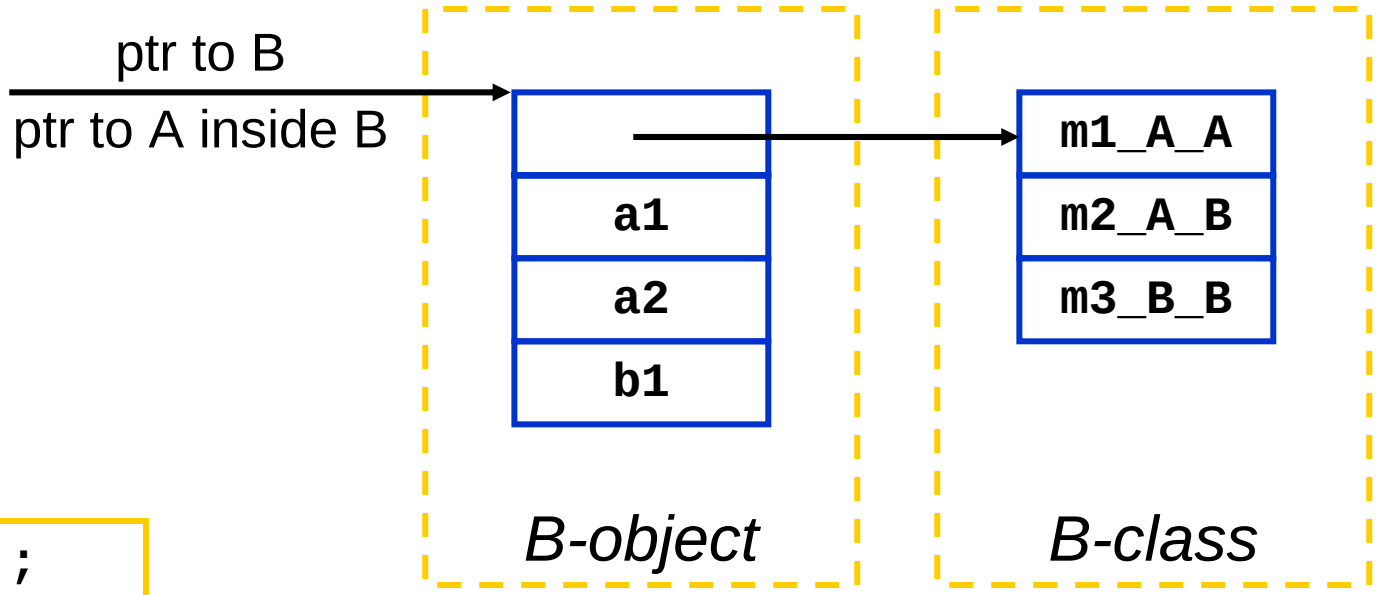
```
B *b = ...;
A *a = b;

a->m2(3);
```

```
m2_A_A(a, 3);
```
static

```
m2_A_B(a, 3);
```
dynamic

Dynamic typing

- implementation: dispatch tables




```
B *b = ...;
A *a = b;

a->m2(3);
```

```
*(a->dispatch_table[1])(a, 3);
```

Dynamic typing

- implementation requires pointer subtyping

```
void m2_A_B(class_A *this_A, int i) {
   class_B *this = convert_ptrA_to_ptrB(this_A);

   this->a2 = this->b1 + i;
}
```

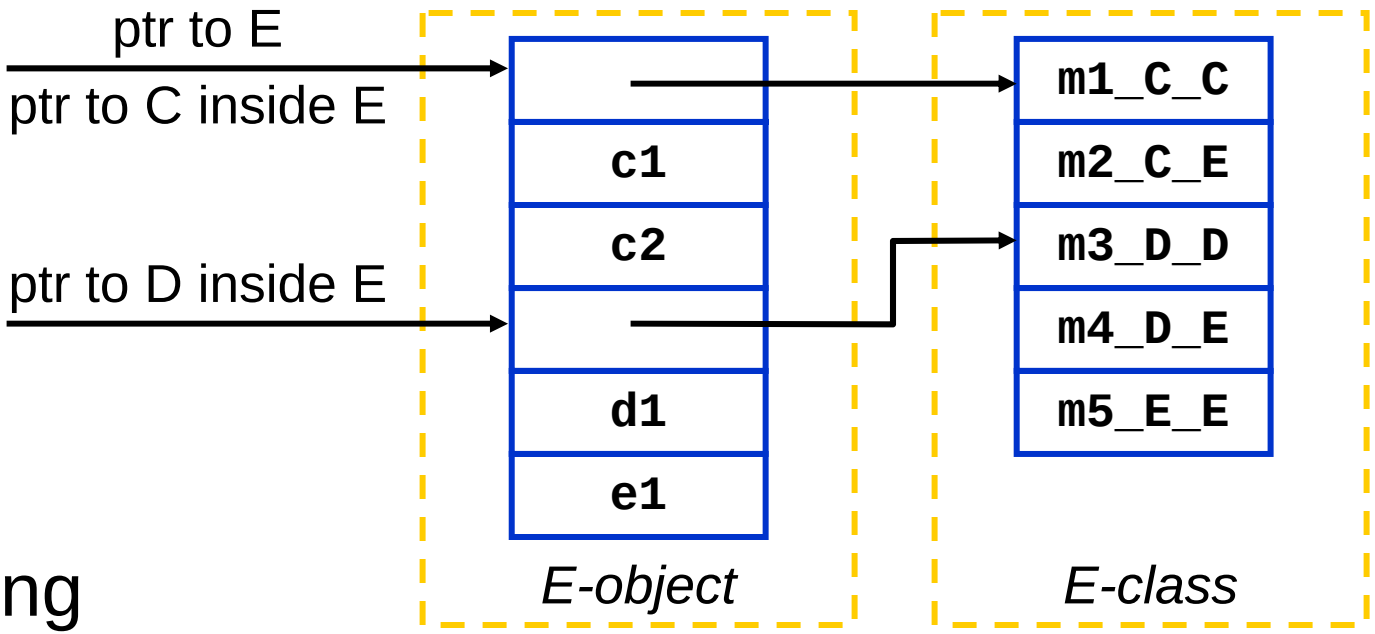
Multiple inheritance

```
class C {
public:
    int c1, c2;
    void m1() {...}
    void m2() {...}
}
```

```
class D {
public:
    int d1;
    void m3() {...}
    void m4() {...}
}
```

```
class E : public C, D {
public:
    int e1;
    void m2() {...}
    void m4() {...}
    void m5() {...}
}
```

Multiple inheritance



- supertyping

```
convert_ptrE_to_ptrC(e) ≈ e
convert_ptrE_to_ptrD(e) ≈ e + sizeof(Class_C)
```

- subtyping

```
convert_ptrC_to_ptrE(c) ≈ c
convert_ptrD_to_ptrE(d) ≈ d - sizeof(Class_C)
```

Exercise (4 min.)

- given an object **e** of class **E**, give the compiled code for the calls

```
e->m1()
e->m3()
e->m4()
```

Answers

Dependent multiple inheritance

```
class A {
public:
    int a1, a2;
    void m1() {...}
    void m3() {...}
}
```

```
class C: public A {
public:
    int c1, c2;
    void m1() {...}
    void m2() {...}
}
```

```
class D: public A {
public:
    int d1;
    void m3() {...}
    void m4() {...}
}
```

```
class E: public C,D {
public:
    int e1;
    void m2() {...}
    void m4() {...}
    void m5() {...}
}
```

Does an object of class E contain 1 or 2 objects of class A?

1: dependent inheritance

2: independent inheritance

Dependent multiple inheritance