

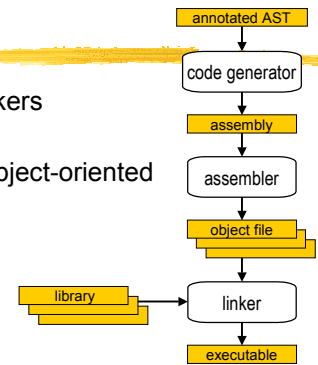
Compiler construction in4020 – lecture 10

Koen Langendoen

**Delft University of Technology
The Netherlands**

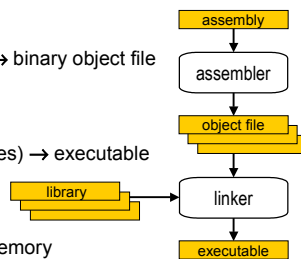
Overview

- assemblers & linkers
- imperative and object-oriented programming
 - context handling
 - code generation



Assemblers, linkers & loaders

- assembler
symbolic object code → binary object file
- linker
multiple objects (libraries) → executable
- loader
load executable into memory



Assemblers

- symbolic code → binary machine code

`addl %edx, %ecx` 0000 0001 11 010 001
 opcode addl reg-reg edx ecx

- handle addresses
 - internal: resolve
 - external: store in relocation table

Handling internal addresses

- two passes
 - compute addresses
 - generate object file
- one pass
 - backpatch list

```

.data
    .align 8
var1:
    .long 666
...
.text
main:
    addl var1, %eax
    ...
    jmp L1;
    ...
L1:
    
```

Handling external addresses

- relocation information
 - entry points (text + data)
 - unresolved references

```

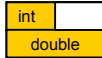
char fmt[] = "x=%d, y=%d\n";
int x = 11;
extern int y;

int main()
{
    printf(fmt, x, y);
    return globj;
}
    
```

symbol	type	address	
_fmt	entry	000000	data
_x	entry	00000c	data
_y	D ref	00000f	text
_main	entry	000000	text
_printf	T ref	00001f	text

Source language data representation and handling

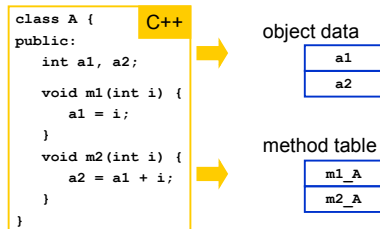
- basic types – map to target hardware
- enumeration types – map to integers
- set types – map to bitset
- record types – field alignment
- union types – union tag?
- array types – descriptors + precomputation
- pointer types – check scope rules
- **object types**
- **routine types**



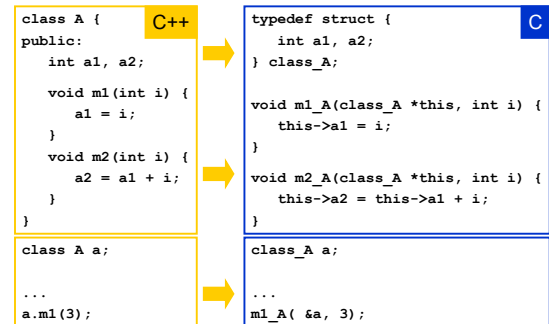
Object types

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

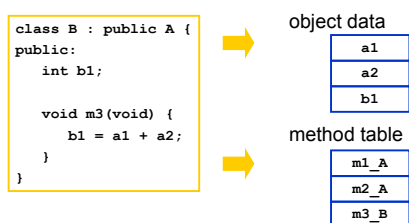
Object representation



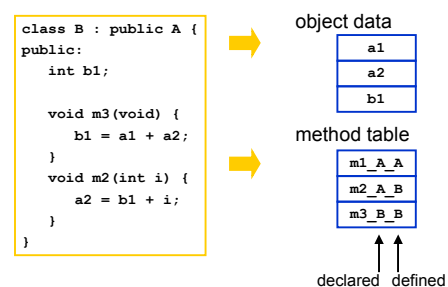
Object representation



Inheritance



Method overriding



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;}
}
```

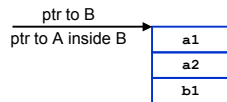
Answers

Polymorphism

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

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

- implementation requires pointer supertyping



Dynamic typing

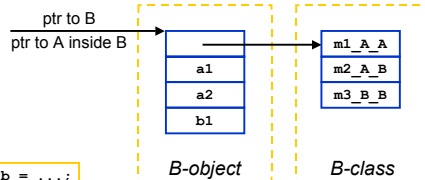
- which method to invoke on overloaded polymorphic types?

```
class B *b = ...;
class A *a = b;
a->m2(3);
```

m2_A_A(a, 3); static
m2_A_B(a, 3); dynamic

Dynamic typing

- implementation: dispatch tables



```
class B *b = ...;
class 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;
}
```

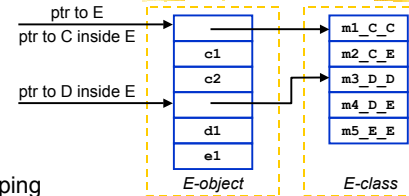
Multiple inheritance

```
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() {...}
}

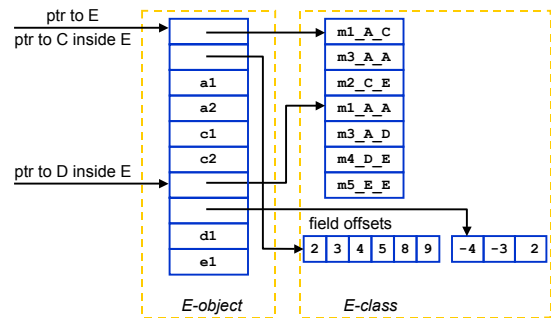
class D: public A {
public:
    int d1;
    void m3() {...}
    void m4() {...}
}

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`?

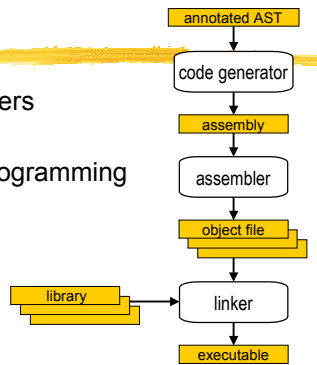
- 1: **dependent** inheritance
- 2: **independent** inheritance

Dependent multiple inheritance



Summary

- assemblers & linkers
- object-oriented programming
 - translation to C
 - method tables



Homework

- study sections:
 - 6.2.6 Array types
- assignment 2:
 - make Asterix OO
 - deadline June 4 08:59
- print handout for next week [blackboard]