Chapter 3_1

Compiler generated code for C++

Assembly Code Specific to C++

- I know three differences in assembly code generated by C++ compilers, compared with those by C compilers:
  1. “new Constructor()” operation
  2. “this” pointer in instance methods
  3. name mangling
  4. “virtual function tables (VFTs)” to implement polymorphism

© Copyright: All rights reserved. Not to be reproduced without prior written consent.

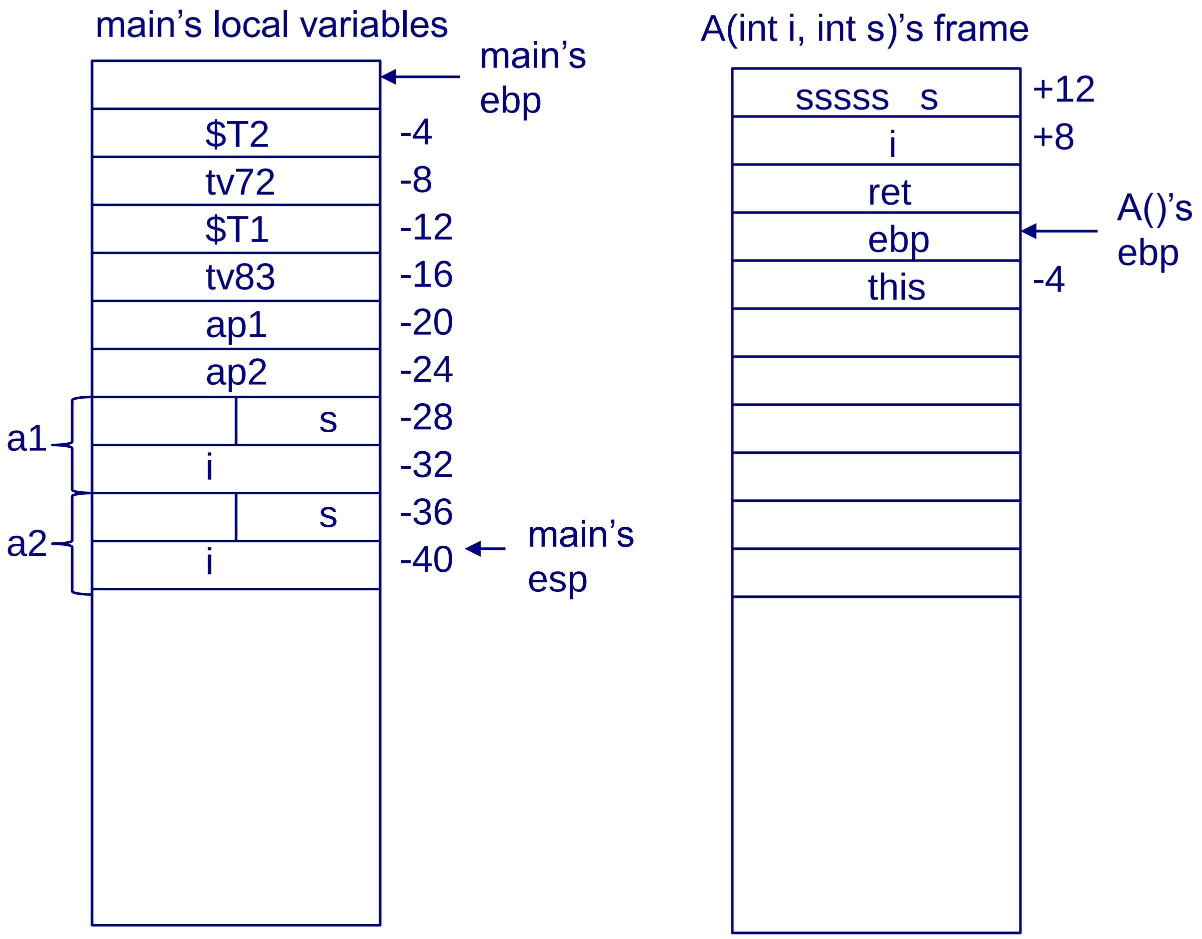
Constructor and “new” operation

- When a “new Constructor()”, say “ap = new A();”, is executed, the compiler generated code, collaborating with run-time functions, does the following:
  1. Allocate area for the object in heap
     - unlike Java or C#, the area is not initialized to 0 by a run-time function
     - the return value of the function is the base (starting) address of the allocated area
  2. Call the constructor with the base address of the area as the “this” parameter
     - push arguments to the constructor if there are arguments
     - In the current version of VC++, “ret X;” is used at the end of the constructor, where X is the number of bytes pushed for the parameters
       - ret X;   => (1) ret (= pop eip), (2) esp += X
     - the “this” pointer (holding the base address of the new object) is returned as the return value of the constructor (more precisely, the return value of the new operation)
  3. The return value of the new operation (= the return value of the constructor is assigned to “ap”.

© Copyright: All rights reserved. Not to be reproduced without prior written consent.

Constructor and “new” operation (cont)

- Refer to “NewOperation” in Chap3 in CoursePrograms



name mangling

- Refer to “NameMangling” in Chap3 in CoursePrograms
- In C++, method name “f” can appear more than once in a program
  - in class A
    - void f();
    - void f(int x);
  - in class B: A
    - void f(int x);

$\left.\begin{array}{l}\text{void f();}\\ \text{void f(int x);}\end{array}\right\}$ method overloading

$\left.\begin{array}{l}\text{void f(int x); (class A)}\\ \text{void f(int x); (class B)}\end{array}\right\}$ method overriding

- In C programs, when a function “f(int x)” is declared, a C compiler usually generates label “_f” to denote the starting address of the compiled code for “f”
- On the other hand, a C++ compiler generates different names for all the overloading/overriding functions of “f” - this is called “name mangling”
- Recall that C compilers do “name mangling” on internal static variables for the same reason.

© Copyright: All rights reserved. Not to be reproduced without prior written consent.

“this” pointer in an instance method

- Refer to “ObjectImplementation” in Chap3 in CoursePrograms
  - In the program, class A is declared as follows:
- When an instance method (void f(int x) in the program) is compiled, the compiler adds one argument, called “this”, to the method
  - The type of “this” is “A *”
- “this” points to the object that the method is applied to
  - when “ap1->f(3);” is executed, “ap1” is passed to “this”
- All accesses to field i in the method are changed to “this->i”

```
class A
{
  public:
    int i1;
    char c1;
    short s1;
    long l1;

    A() {
        i1 = 3;
        c1 = 4;
        s1 = 5;
        l1 = 6;
    }

    void f(int x) {
      i1 += x;
      c1 += x * 2;
      s1 += x + 3;
      l1 += x - 5;
    }
};
```

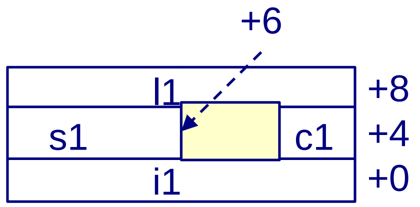
© Copyright: All rights reserved. Not to be reproduced without prior written consent.

“this” pointer in an instance method (cont)

when ap1->f(3) is executed

```
f (A *this, int x) {
   this->i1 +=  x;
   this->c1 += x * 2;
   this->s1 += x + 3;
   this->l1 += x – 5;
}
```

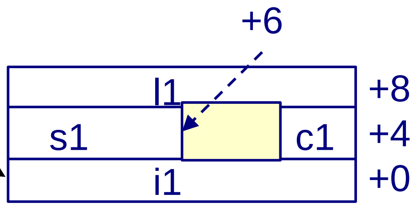
In all the compilers I know of, “this” is passed to the function using the “ecx” register



object pointed to by ap1

when ap2->f(5) is executed

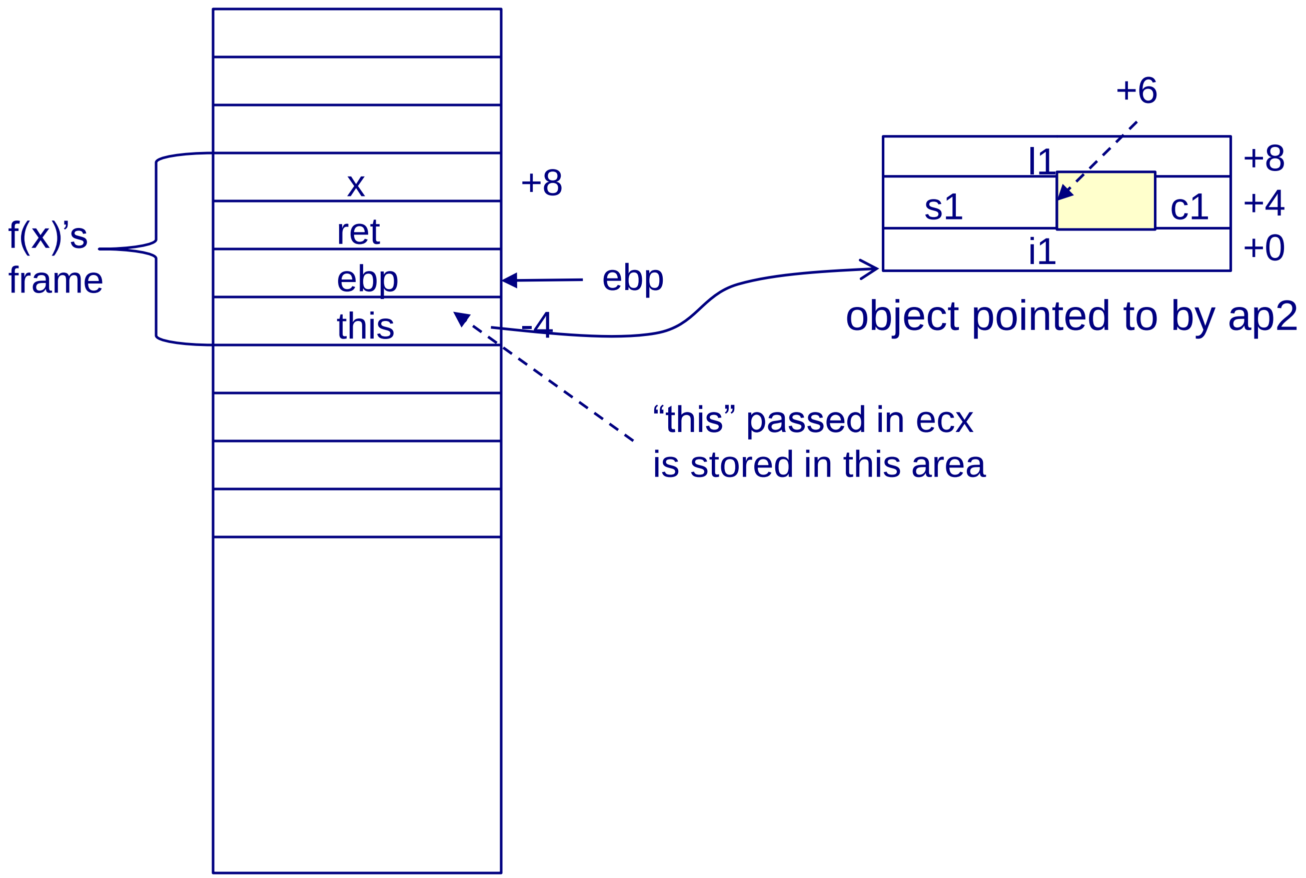
```
f (A *this, int x) {
   this->i1 +=  x;
   this->c1 += x * 2;
   this->s1 += x + 3;
   this->l1 += x – 5;
}
```



object pointed to by ap2

© Copyright: All rights reserved. Not to be reproduced without prior written consent.

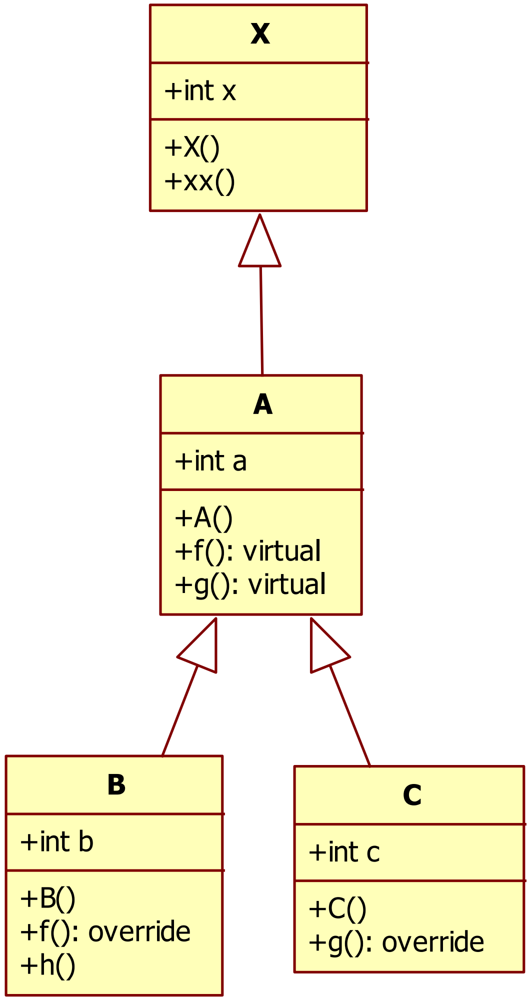
“this” pointer in an instance method (cont)



© Copyright: All rights reserved. Not to be reproduced without prior written consent.

Polymorphism

- Consider the class structure on the right
- For objects of each class, which methods are called is determined at compile time:
  - X'x object:
    - xx( ) : X::xx()
  - A's objects:
    - xx() : X::xx()
    - f( ) : A::f(),   g() : A::g()
  - B's objects:
    - xx() : X::xx()
    - f( ) : B::f(),   g() : A::g(), and
    - h( ) : B::h() through ptr to B
  - C's objects:
    - xx( ) : X::xx( )
    - f( ): A::f(),   g() : C::g()



© Copyright: All rights reserved. Not to be reproduced without prior written consent.