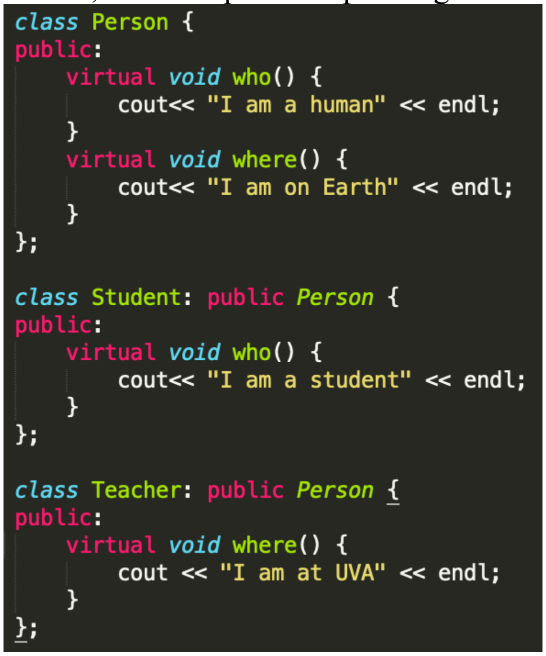
Yubin Lim yl2vv
inlab9.pdf
April 11, 2019
CS2150-103

Dynamic Dispatch

For dynamic dispatch the decision on which member function to invoke is made using run-time type of an object. This is other type of dispatch is static dispatch, where the decision on which member function to invoke is made using compile-time type of an object. A difference between Java and C++ is that Java uses dynamic dispatch by default, that is, everything is virtual, while C++ supports both types of dispatch. Virtual methods are only needed under certain circumstances, specifically when you want to use polymorphism. The syntax use in C++ for dynamic dispatch is the "virtual" keyword. By using "virtual", dynamic dispatch is implemented with the use of virtual method tables. The C++ compiler inserts a virtual method table for every class having virtual function or class inherited from the class that has a virtual function. In essence, each object has a pointer to the virtual method table. That table has the addresses of the methods. Any virtual method must follow the pointer to the object, then follow the virtual method table pointer. It proceeds to look up the method pointer and jump to that method.

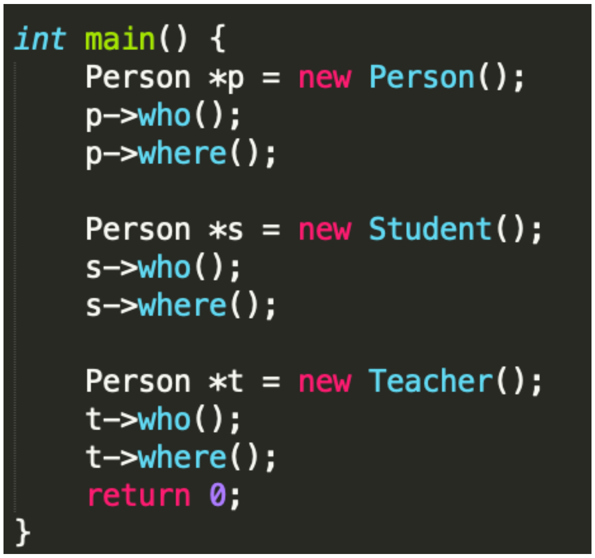
As an example, I created three classes: Person superclass, and two subclasses called Student and Teacher (as shown by the picture below). I used the virtual keyword to make the functions use dynamic dispatch. In the main method, I created a Person pointer p pointing to Person, a Person pointer s pointing to Student, and a Person pointer t pointing to Teacher.



```
class Person {
public:
    virtual void who() {
        cout<< "I am a human" << endl;
    }
    virtual void where() {
        cout<< "I am on Earth" << endl;
    }
};

class Student: public Person {
public:
    virtual void who() {
        cout<< "I am a student" << endl;
    }
};

class Teacher: public Person {
public:
    virtual void where() {
        cout << "I am at UVA" << endl;
    }
};
```

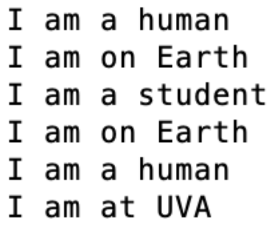


```
int main() {
    Person *p = new Person();
    p->who();
    p->where();

    Person *s = new Student();
    s->who();
    s->where();

    Person *t = new Teacher();
    t->who();
    t->where();
    return 0;
}
```

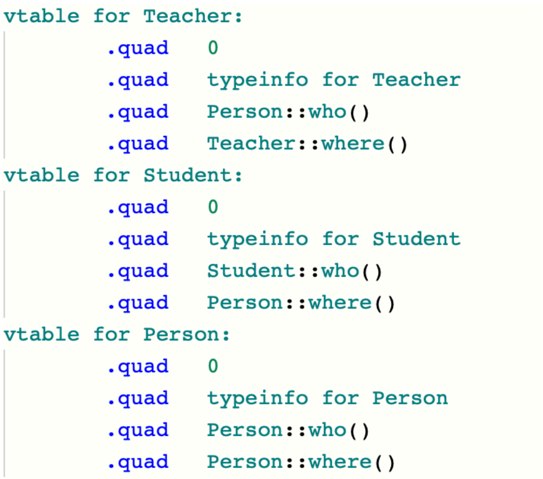
Per expected, the code printed the following:



```
I am a human
I am on Earth
I am a student
I am on Earth
I am a human
I am at UVA
```

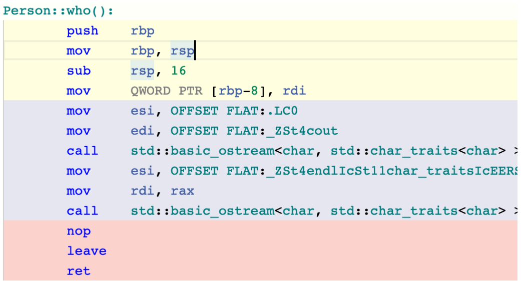
The code was able to decide what member function to call during run time. For example, as the Student Class inherited the Person Class. The vtable of Student Class contains pointer to the who() function of Student class (“I am a student”), but the pointer to where() is of base class, the Person class (“I am on Earth”).

I used godbolt.org to convert my C++ code into assembly. The assembly gave a vtable, which is the virtual table that contains the pointers. The vtable contains the addresses of the appropriate methods as well as the pointers to metadata. As shown in the table, in the virtual table for Teacher, the function who() is called from the Person class while the function where() is called from the Teacher class. In the virtual table for Student, the function who() is called from the Student class while the function where() is called from the Person class.

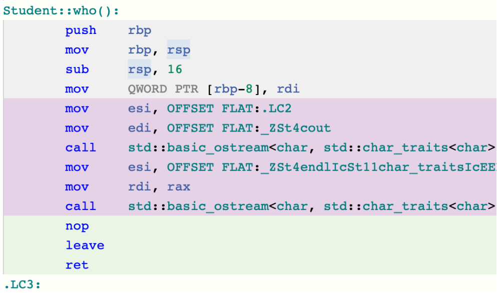


```
vtable for Teacher:
        .quad   0
        .quad   typeinfo for Teacher
        .quad   Person::who()
        .quad   Teacher::where()
vtable for Student:
        .quad   0
        .quad   typeinfo for Student
        .quad   Student::who()
        .quad   Person::where()
vtable for Person:
        .quad   0
        .quad   typeinfo for Person
        .quad   Person::who()
        .quad   Person::where()
```

The assembly code for the Person::who() and the Student::who() looks almost identical, it is the difference in the virtual table. QWORD PTR shows the pointer in dynamic dispatch.

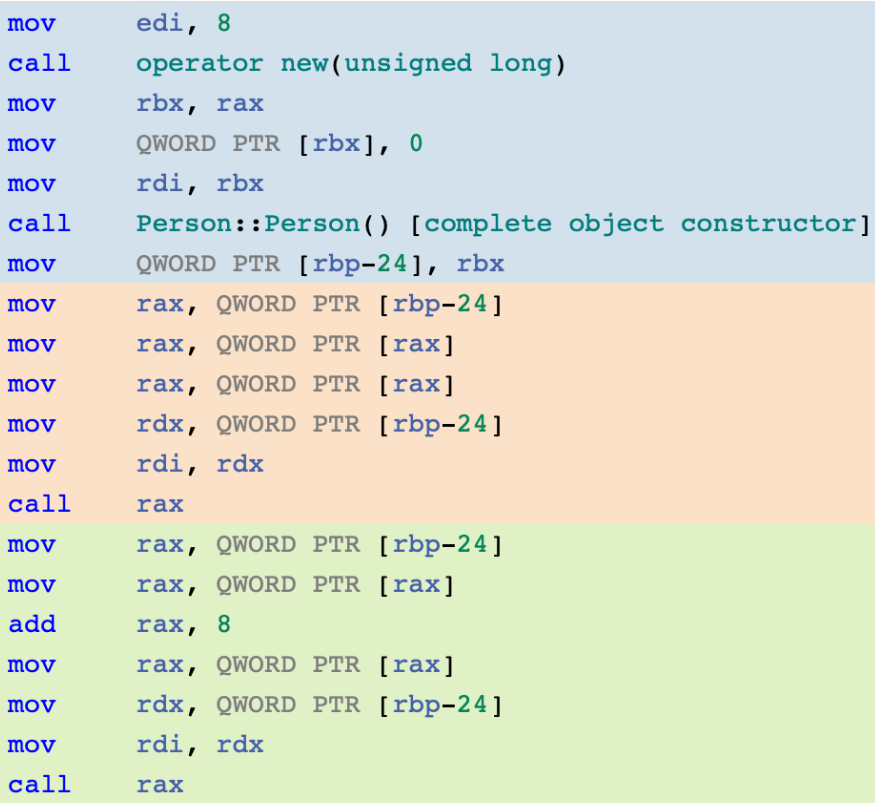


```
Person::who():
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     esi, OFFSET FLAT:.LC0
        mov     edi, OFFSET FLAT:_ZSt4cout
        call    std::basic_ostream<char, std::char_traits<char> >
        mov     esi, OFFSET FLAT:_ZSt4endlIcSt11char_traitsIcEERS
        mov     rdi, rax
        call    std::basic_ostream<char, std::char_traits<char> >
        nop
        leave
        ret
```

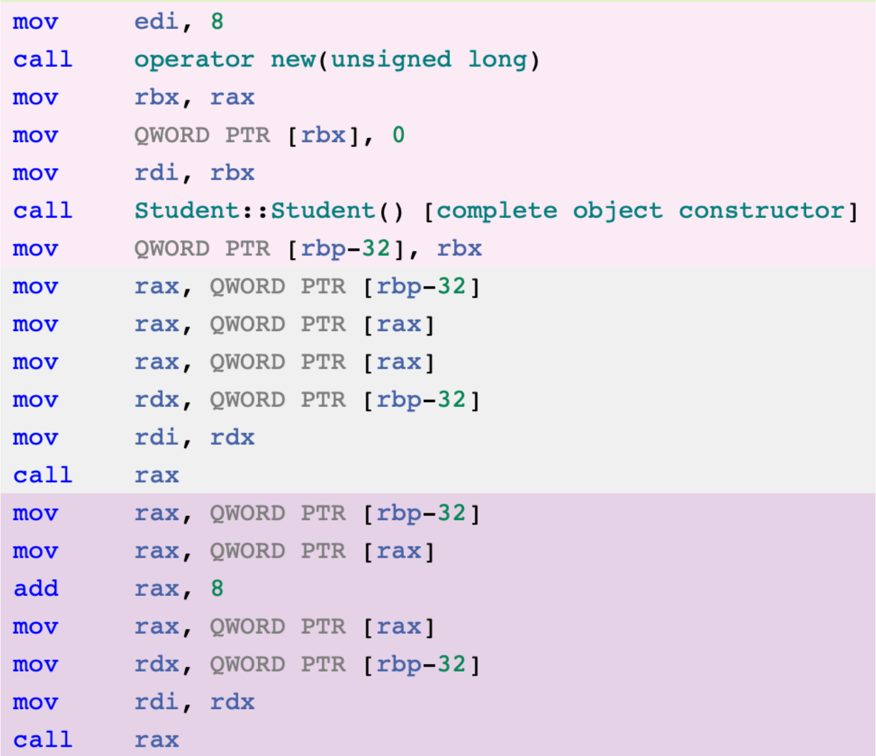


```
Student::who():
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     esi, OFFSET FLAT:.LC2
        mov     edi, OFFSET FLAT:_ZSt4cout
        call    std::basic_ostream<char, std::char_traits<char>
        mov     esi, OFFSET FLAT:_ZSt4endlIcSt11char_traitsIcEE
        mov     rdi, rax
        call    std::basic_ostream<char, std::char_traits<char>
        nop
        leave
        ret
.LC3:
```

The assembly code from the main method is shown below. The first picture is from the pointer p and second picture is the pointer s. The call to the first function, who() has the virtual pointer to the appropriate function. The call to the second function where() has a line that wasn't in the who() function. It is the assembly line “add rax, 8” which shows that since we are calling the second overridden method, we access an offset of the virtual method pointer by 8 bytes.



```
mov     edi, 8
call    operator new(unsigned long)
mov     rbx, rax
mov     QWORD PTR [rbx], 0
mov     rdi, rbx
call    Person::Person() [complete object constructor]
mov     QWORD PTR [rbp-24], rbx
mov     rax, QWORD PTR [rbp-24]
mov     rax, QWORD PTR [rax]
mov     rax, QWORD PTR [rax]
mov     rdx, QWORD PTR [rbp-24]
mov     rdi, rdx
call    rax
mov     rax, QWORD PTR [rbp-24]
mov     rax, QWORD PTR [rax]
add     rax, 8
mov     rax, QWORD PTR [rax]
mov     rdx, QWORD PTR [rbp-24]
mov     rdi, rdx
call    rax
```



```
mov     edi, 8
call    operator new(unsigned long)
mov     rbx, rax
mov     QWORD PTR [rbx], 0
mov     rdi, rbx
call    Student::Student() [complete object constructor]
mov     QWORD PTR [rbp-32], rbx
mov     rax, QWORD PTR [rbp-32]
mov     rax, QWORD PTR [rax]
mov     rax, QWORD PTR [rax]
mov     rdx, QWORD PTR [rbp-32]
mov     rdi, rdx
call    rax
mov     rax, QWORD PTR [rbp-32]
mov     rax, QWORD PTR [rax]
add     rax, 8
mov     rax, QWORD PTR [rax]
mov     rdx, QWORD PTR [rbp-32]
mov     rdi, rdx
call    rax
```