//Shahroz Imtiaz, si6rf, inlab9.pdf, 10/20/2018

1. Dynamic dispatch: Describe how dynamic dispatch is implemented. Note that dynamic dispatch is NOT the same thing as dynamic memory! Show this using a simple class hierarchy that includes virtual functions. Use more than one virtual function per class.
    a. Dynamic dispatch is used when a call to an overridden method is resolved at runtime instead of compile time. Let's look at a simple program I created to illustrate this:

inlab.cpp

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
using namespace std;

class Alexa {
    public:
       void print() {
       cout<<"From class A\n";
   }
};

class Bart : public Alexa {
    public:
       void print(){
        cout<<"From class B\n";
    }
};

int main () {
    srand(time(NULL));
    int boolean = rand() % 2;
    Alexa *test;
    if ( boolean)
        test = new Alexa();
    else
        test = new Bart();
    test->print();
    return 0;
}
```

```
inlab — -bash — 80×24
[Shahrozs-MacBook-Air:inlab shahrozimtiaz$ ./a.out
From class A
[Shahrozs-MacBook-Air:inlab shahrozimtiaz$ ./a.out
From class A
[Shahrozs-MacBook-Air:inlab shahrozimtiaz$ ./a.out
From class A
[Shahrozs-MacBook-Air:inlab shahrozimtiaz$ ./a.out
From class A
[Shahrozs-MacBook-Air:inlab shahrozimtiaz$ ./a.out
From class A
Shahrozs-MacBook-Air:inlab shahrozimtiaz$
```

inlab.cpp

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
using namespace std;

class Alexa {
    public:
    virtual void print() {
       cout<<"From class A\n";
   }
};

class Bart : public Alexa {
    public:
    virtual void print(){
        cout<<"From class B\n";
    }
};

int main () {
    srand(time(NULL));
    int boolean = rand() % 2;
    Alexa *test;
    if ( boolean)
        test = new Alexa();
    else
        test = new Bart();
    test->print();
    return 0;
}
```

```
inlab — -bash — 80×24
[Shahrozs-MacBook-Air:inlab shahrozimtiaz$ ./a.out
From class B
[Shahrozs-MacBook-Air:inlab shahrozimtiaz$ ./a.out
From class A
[Shahrozs-MacBook-Air:inlab shahrozimtiaz$ ./a.out
From class B
[Shahrozs-MacBook-Air:inlab shahrozimtiaz$ ./a.out
From class A
[Shahrozs-MacBook-Air:inlab shahrozimtiaz$ ./a.out
From class A
[Shahrozs-MacBook-Air:inlab shahrozimtiaz$ ./a.out
From class B
[Shahrozs-MacBook-Air:inlab shahrozimtiaz$ ./a.out
From class A
Shahrozs-MacBook-Air:inlab shahrozimtiaz$
```

Assembly code for test->print();

```
mov rdi, qword ptr [rbp - 16]
call Alexa::print()
xor eax, eax
```

Assembly code for test->print();

```
mov rax, qword ptr [rbp - 16]
mov rcx, qword ptr [rax]
mov rdi, rax
call qword ptr [rcx]
xor eax, eax
```

The left-side shows how static dispatching works. The compiler already knows at compile time that it will call the print function from Alexa, thus you can see “call Alexa::print()” in the assembly code. But the right side shows dynamic dispatching because I used the “virtual” keyword. My program randomly makes Boolean true or false and depending on the result of that it determines whether the print function will be called from Alexa or Bart. The compiler can’t possibly account for this and thus has to wait until runtime to figure out which print function to call. Thus, in the assembly code, you can see the compiler accessing the stack to figure this out during runtime, unlike before when it knew to call the print function from Alexa at compile time.