

601.220 Intermediate Programming

Polymorphism

Polymorphism: “many forms”

```
int main() {  
    std::vector<Account> my_accounts;  
  
    // this is sort of OK, since CheckingAccount is  
    // derived from Account (except: "slicing" will  
    // occur)  
    my_accounts.push_back(CheckingAccount(2000.0));  
  
    std::cout << my_accounts.back().type() << std::endl;  
    return 0;  
}
```

Polymorphism: “many forms”

```
class Account {
public:
    ...
    std::string type() const { return "Account"; }
    ...
};

class CheckingAccount : public Account {
public:
    ...
    std::string type() const { return "CheckingAccount"; }
    ...
};

class SavingsAccount : public Account {
public:
    ...
    std::string type() const { return "SavingsAccount"; }
    ...
};
```

Polymorphism: “many forms”

// account.h

```
1  #include <string>
2  class Account {
3  public:
4      Account() : balance(0.0) { }
5      Account(double initial) : balance(initial) { }
6
7      void credit(double amt)    { balance += amt; }
8      void debit(double amt)     { balance -= amt; }
9      double get_balance() const { return balance; }
10     std::string type() const { return "Account"; }
11 private:
12     double balance;
13 };
14
15 class CheckingAccount : public Account {
16 public:
17     CheckingAccount(double initial, double atm) :
18         Account(initial), total_fees(0.0),
19         atm_fee(atm) { }
20     void cash_withdrawal(double amt) {
21         total_fees += atm_fee;
22         debit(amt + atm_fee);
23     }
24     double get_total_fees() const {
25         return total_fees;
26     }
```

```
27     std::string type() const {
28         return "CheckingAccount";
29     }
30
31 private:
32     double total_fees;
33     double atm_fee;
34 };
35
36 class SavingsAccount : public Account {
37 public:
38     SavingsAccount(double initial, double rate) :
39         Account(initial), annual_rate(rate) { }
40
41     //Not shown here; usual compound interest calc
42     double total_after_years(int years);
43
44     std::string type() const {
45         return "SavingsAccount";
46     }
47
48 private:
49     double annual_rate;
50 };
```

Polymorphism: “many forms”

```
// account_main.cpp

1  #include <iostream>
2  #include "account.h"
3
4  void print_account_type(const Account& acct) {
5      std::cout << acct.type() << std::endl;
6  }
7
8  int main() {
9      Account acct(1000.0);
10     CheckingAccount checking(1000.0, 2.00);
11     SavingsAccount saving(1000.0, 0.05);
12
13     print_account_type(acct);
14     print_account_type(checking);
15     print_account_type(saving);
16
17     return 0;
18 }
```

Polymorphism: “many forms”

Note the types:

```
void print_account_type(const Account& acct) {  
    std::cout << acct.type() << std::endl;  
}
```

```
int main() {  
    ...  
    CheckingAccount checking(1000.0, 2.00);  
    ...  
    print_account_type(checking);  
    ...  
}
```

Polymorphism: “many forms”

In main, `checking_acct` has type `CheckingAccount`

Passed to `print_account_type` as `const Account&`

- This is allowed; `CheckingAccount` is derived from `Account`

Usually, you may use a variable of a derived type **as though it has the base type**

- Makes logical sense; `CheckingAccount` **is-an** `Account`

Polymorphism: “many forms”

```
void print_account_type(const Account& acct) {  
    std::cout << acct.type() << std::endl;  
}
```

```
int main() {  
    ...  
    CheckingAccount checking(1000.0, 2.00);  
    ...  
    print_account_type(checking);  
    ...  
}
```

Does `acct.type()` call `Account::type()` (matching parameter's type) or `CheckingAccount::type()` (matching the original declared type)?

Polymorphism: “many forms”

```
$ g++ -o account_main account_main.cpp -std=c++11 -pedantic -Wall -Wextra
```

```
$ ./account_main
```

```
Account
```

```
Account
```

```
Account
```

It calls `Account::type()`

Can we force `print_account_type` to call the function corresponding to the **actual** type (`CheckingAccount`) rather than the locally declared base type (`Account`)?

This requires **dynamic binding**

To use it, we declare relevant member functions as `virtual`

Polymorphism: dynamic binding

```
class Account {
public:
    ...
    virtual std::string type() const { return "Account"; }
    ...
};

class CheckingAccount : public Account {
public:
    ...
    virtual std::string type() const { return "CheckingAccount"; }
    ...
};

class SavingsAccount : public Account {
public:
    ...
    virtual std::string type() const { return "SavingsAccount"; }
    ...
};
```

Polymorphism: dynamic binding

// account2.h

```
1  #include <string>
2  class Account {
3  public:
4      Account() : balance(0.0) { }
5      Account(double initial) : balance(initial) { }
6
7      void credit(double amt)    { balance += amt; }
8      void debit(double amt)     { balance -= amt; }
9      double get_balance() const { return balance; }
10     virtual std::string type() const {
11         return "Account";
12     }
13 private:
14     double balance;
15 };
16
17 class CheckingAccount : public Account {
18 public:
19     CheckingAccount(double initial, double atm) :
20         Account(initial), total_fees(0.0),
21         atm_fee(atm) { }
22     void cash_withdrawal(double amt) {
23         total_fees += atm_fee;
24         debit(amt + atm_fee);
25     }
26     double get_total_fees() const {
27         return total_fees;
28     }
29
30     virtual std::string type() const {
31         return "CheckingAccount";
32     }
33
34 private:
35     double total_fees;
36     double atm_fee;
37 };
38
39 class SavingsAccount : public Account {
40 public:
41     SavingsAccount(double initial, double rate) :
42         Account(initial), annual_rate(rate) { }
43
44     //Not shown here; usual compound interest calc
45     double total_after_years(int years);
46
47     virtual std::string type() const {
48         return "SavingsAccount";
49     }
50
51 private:
52     double annual_rate;
53 };
```

Polymorphism: dynamic binding

```
// account_main2.cpp

1  #include <iostream>
2  #include "account2.h"
3
4  void print_account_type(const Account& acct) {
5      std::cout << acct.type() << std::endl;
6  }
7
8  int main() {
9      Account acct(1000.0);
10     CheckingAccount checking(1000.0, 2.00);
11     SavingsAccount saving(1000.0, 0.05);
12
13     print_account_type(acct);
14     print_account_type(checking);
15     print_account_type(saving);
16
17     return 0;
18 }
```

Polymorphism: dynamic binding

```
$ g++ -o account_main2 account_main2.cpp -std=c++11 -pedantic -Wall -Wextra
```

```
$ ./account_main2
```

```
Account
```

```
CheckingAccount
```

```
SavingsAccount
```

Quiz!

What output is printed by the following program?

// quiz1.cpp

```
1  #include <iostream>
2
3  class X {
4  public:
5      void p() { std::cout << "X::p "; }
6      virtual void q()
7          { std::cout << "X::q "; }
8  };
9  class Y : public X {
10 public:
11     void p() { std::cout << "Y::p "; }
12     virtual void q()
13         { std::cout << "Y::q "; }
14 };
15
16 void f(X &obj) { obj.p(); obj.q(); }
17
18 int main() { Y myObj; f(myObj); }
```

- A. X::p X::q
- B. X::p Y::q
- C. Y::p X::q
- D. Y::p Y::q
- E. Some other output is printed

Quiz - answers

What output is printed by the following program?

```
// quiz1.cpp
```

```
1  #include <iostream>
2
3  class X {
4  public:
5      void p() { std::cout << "X::p "; }
6      virtual void q()
7          { std::cout << "X::q "; }
8  };
9  class Y : public X {
10 public:
11     void p() { std::cout << "Y::p "; }
12     virtual void q()
13         { std::cout << "Y::q "; }
14 };
15
16 void f(X &obj) { obj.p(); obj.q(); }
17
18 int main() { Y myObj; f(myObj); }
```

At line 16:

Symbols(Scope-Type)	Values
myObj(main-Y), obj(f-X)	-
(obj.p())(f)	"X::p "
(obj.q())(f)	"Y::q "

A very similar quiz!

What output is printed by the following program?

// quiz2.cpp

```
1  #include <iostream>
2
3  class X {
4  public:
5      void p() { std::cout << "X::p "; }
6      virtual void q()
7          { std::cout << "X::q "; }
8  };
9  class Y : public X {
10 public:
11     void p() { std::cout << "Y::p "; }
12     virtual void q()
13         { std::cout << "Y::q "; }
14 };
15
16 void f(X obj) { obj.p(); obj.q(); }
17
18 int main() { Y myObj; f(myObj); }
```

- A. X::p X::q
- B. X::p Y::q
- C. Y::p X::q
- D. Y::p Y::q
- E. Some other output is printed

A very similar quiz - answers

What output is printed by the following program?

// quiz2.cpp

```
1  #include <iostream>
2
3  class X {
4  public:
5      void p() { std::cout << "X::p "; }
6      virtual void q()
7          { std::cout << "X::q "; }
8  };
9  class Y : public X {
10 public:
11     void p() { std::cout << "Y::p "; }
12     virtual void q()
13         { std::cout << "Y::q "; }
14 };
15
16 void f(X obj) { obj.p(); obj.q(); }
17
18 int main() { Y myObj; f(myObj); }
```

At line 16:

Symbols(Scope-Type)	Values
myObj(main-Y)	-
obj(f-X)	-
(obj.p())(f)	"X::p "
(obj.q())(f)	"X::q "