601.220 Intermediate Programming

Polymorphism

Polymorphism: “many forms”

```
int main() {
    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));

    cout << my_accounts.back().type() << 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

```
#include <string>
class Account {
public:
    Account() : balance(0.0) { }
    Account(double initial) : balance(initial) { }

    void credit(double amt)    { balance += amt; }
    void debit(double amt)     { balance -= amt; }
    double get_balance() const { return balance; }
    std::string type() const { return "Account"; }
private:
    double balance;
};

class CheckingAccount : public Account {
public:
    CheckingAccount(double initial, double atm) :
        Account(initial), total_fees(0.0),
            atm_fee(atm) { }
    void cash_withdrawal(double amt) {
        total_fees += atm_fee;
        debit(amt + atm_fee);
    }
    double get_total_fees() const {
        return total_fees;
    }
    std::string type() const {
        return "CheckingAccount";
    }

private:
    double total_fees;
    double atm_fee;
};

class SavingsAccount : public Account {
public:
    SavingsAccount(double initial, double rate) :
        Account(initial), annual_rate(rate) { }

    //Not shown here; usual compound interest calc
    double total_after_years(int years);

    std::string type() const {
        return "SavingsAccount";
    }

private:
    double annual_rate;
};
```

Polymorphism

```cpp
// account_main3.cpp:
#include <iostream>
#include "account2.h"

using std::cout;  using std::endl;

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

int main() {
    Account acct(1000.0);
    CheckingAccount checking(1000.0, 2.00);
    SavingsAccount saving(1000.0, 0.05);

    print_account_type(acct);
    print_account_type(checking);
    print_account_type(saving);

    return 0;
}
```

Polymorphism

Note the types:

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

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

Polymorphism

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

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

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

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

Polymorphism

```
$ g++ -c account_main3.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o account_main3 account_main3.o
$ ./account_main3
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`)?

Polymorphism

This requires *dynamic binding*

To use it, we declare relevant member functions as *virtual*:

Polymorphism

```
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

```
#include <string>
class Account {
public:
    Account() : balance(0.0) { }
    Account(double initial) : balance(initial) { }

    void credit(double amt)    { balance += amt; }
    void debit(double amt)     { balance -= amt; }
    double get_balance() const { return balance; }
    virtual std::string type() const {
        return "Account";
    }
private:
    double balance;
};

class CheckingAccount : public Account {
public:
    CheckingAccount(double initial, double atm) :
        Account(initial), total_fees(0.0),
            atm_fee(atm) { }
    void cash_withdrawal(double amt) {
        total_fees += atm_fee;
        debit(amt + atm_fee);
    }
    double get_total_fees() const {
        return total_fees;
    }
    virtual std::string type() const {
        return "CheckingAccount";
    }

private:
    double total_fees;
    double atm_fee;
};

class SavingsAccount : public Account {
public:
    SavingsAccount(double initial, double rate) :
        Account(initial), annual_rate(rate) { }

    //Not shown here; usual compound interest calc
    double total_after_years(int years);

    virtual std::string type() const {
        return "SavingsAccount";
    }
private:
    double annual_rate;
};
```

Polymorphism

```cpp
// account_main4.cpp:
#include <iostream>
#include "account3.h"  // now with *virtual* `type` member functions
using std::cout;
using std::endl;

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

int main() {
    Account acct(1000.0);
    CheckingAccount checking(1000.0, 2.00);
    SavingsAccount saving(1000.0, 0.05);

    print_account_type(acct);
    print_account_type(checking);
    print_account_type(saving);

    return 0;
}
```

Polymorphism

```
$ g++ -c account_main4.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o account_main4 account_main4.o
$ ./account_main4
Account
CheckingAccount
SavingsAccount
```

Quiz!

What output is printed by the following program?

```
#include <iostream>
using std::cout;

class X {
public:
  void p() { cout << "X::p "; }
  virtual void q()
    { cout << "X::q "; }
};
class Y : public X {
public:
  void p() { cout << "Y::p "; }
  virtual void q()
    { cout << "Y::q "; }
};

void f(X &obj) { obj.p(); obj.q(); }

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

Very similar but not identical quiz!

What output is printed by the following program?

```
#include <iostream>
using std::cout;

class X {
public:
  void p() { cout << "X::p "; }
  virtual void q()
    { cout << "X::q "; }
};
class Y : public X {
public:
  void p() { cout << "Y::p "; }
  virtual void q()
    { cout << "Y::q "; }
};

void f(X obj) { obj.p(); obj.q(); }

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