Topic 7

1. Object oriented programming
2. Implementing a simple class
3. Specifying the public interface
4. Designing the data representation
5. Member functions
6. Constructors
7. Problem solving: tracing objects
8. Problem solving: discovering classes
9. Separate compilation
10. Pointers to objects
11. Problem solving: patterns for object data

Tracing Objects

Recall how you hand traced code
to help you understand functions.

Adapting tracing for objects
will help you understand objects.

Grab some index cards, one for each object in the program.

Tracing Objects(2)

You know that the **`public:`** section is for others.
That's where you'll write methods for their use.
That will be the front of the card.

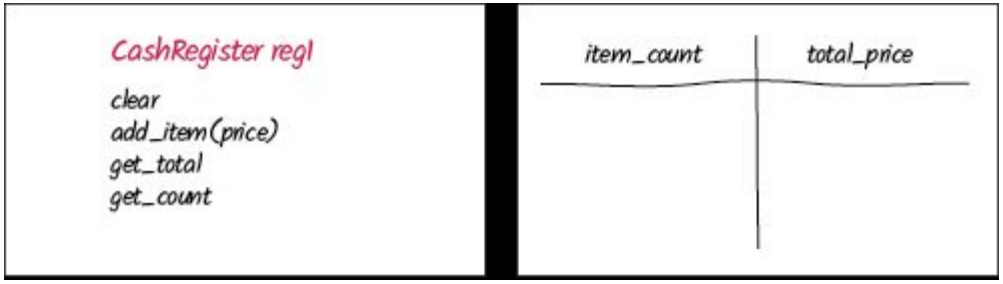
```
class CashRegister
{
public:
   void clear();
   void add_item(double price);
   double get_total() const;
   int get_count() const;
private:
   int item_count;
   double total_price;
};

...

CashRegister reg1;
```

Tracing Objects(3)

You know that the **`private:`** section is for your data – they are not allowed to mess with it except through the public methods you provide. That will be the back of the card.

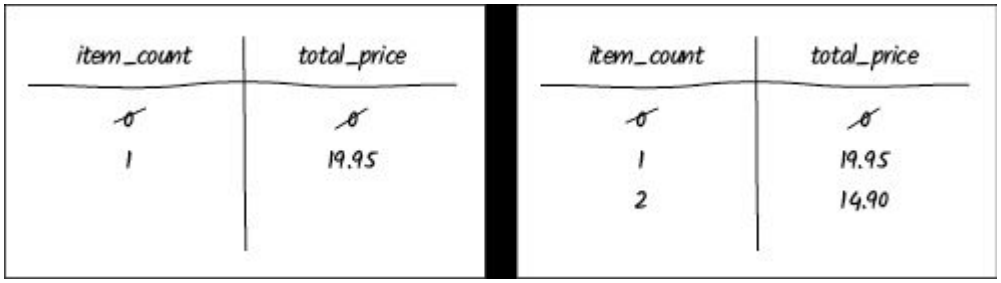


Tracing Objects(4)

```
CashRegister reg1;
CashRegister reg2;
reg1.addItem(19.95);
```

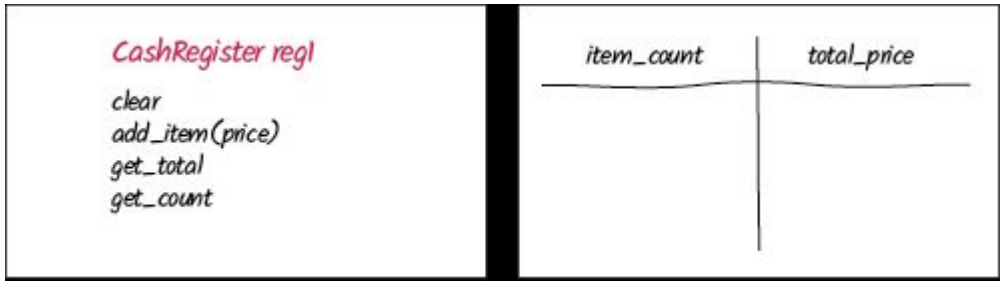
When each object is constructed, fill in the first line on the card table with the initial values.

Then, whenever a member function is called, cross out the old values and write in the new one values on that object's card...



Practice It: Tracing Objects

- Grab some paper or index cards, and trace through the following program, using the `CashRegister class` implementation we have seen so far. Also record the outputs from the `display()` function:




```
int main()
{
   CashRegister register1;
   register1.clear();
   CashRegister reg2;
   reg2.clear();
   register1.add_item(1.95);
   reg2.add_item(0.95);
   display(register1);
   register1.add_item(3.95);
   reg2.add_item(0.55);
   register1.clear();
   display(reg2);
   return 0;
}
```

How to 9.1: Implementing a Class

- Follow these steps to create a class, given a problem description:

1. Get an informal list of the responsibilities of your objects.
2. Specify the public interface.
   - Write the `public:` part of the `class{}` defintion
3. Document the public interface.
   - Add comments describing the class and each function: parameters, return value
4. Determine data members.
5. Implement constructors and member functions.
6. Test your class.
   - Write a `main()` that creates 2 objects and calls all class functions.

WORKED EXAMPLE 9.1: Bank Account Class

- **Problem Statement:** Write a class for a bank account. Customers can deposit and withdraw, but if the balance falls <0, a $10 overdraft penalty is charged. At the end of the month, interest is added to the account. The interest rate can vary every month.
- We'll follow the 6-step method from the preceding slide

1. Get an informal list of the responsibilities of your objects.
   - Deposit funds.
   - Withdraw funds.
   - Add interest.
   - There is a hidden responsibility as well. We need to be able to find out how much money is in the account
     - Get balance.

Bank Account Class(2)

2. Specify the public interface. (constructors and functions listed above)

```
class BankAccount
{
public:
   BankAccount();
   BankAccount(double initial_balance);
   void deposit(double amount);
   void withdraw(double amount);
   void add_interest(double rate);
   double get_balance() const;
private:
   ...
};
```

Bank Account Class(3)

3. Document the public interface (partial list shown below):

```
/**
   A bank account whose balance can be changed by deposits
and withdrawals.
*/
class BankAccount
{
public:
   /**
      Constructs a bank account with zero balance.
   */
   BankAccount();

   /**
      Constructs a bank account with a given balance.
      @param initial_balance the initial balance
   */
   BankAccount(double initial_balance);
```

Bank Account Class(4)

4. Determine data members. Clearly we need to store the bank balance:

```
class BankAccount
{
   ...
private:
   double balance;
};
```

Do we need to store the interest rate? No—it varies every month, and is supplied as an argument to `add_interest`.

Bank Account Class(5)

5. Implement constructors and member functions.

```
BankAccount::BankAccount()
{
   balance = 0;
}
BankAccount::BankAccount(double initial_balance)
{
   balance = initial_balance;
}
double BankAccount::get_balance() const
{
   return balance;
}
void BankAccount::deposit(double amount)
{
   balance = balance + amount;
}

// etc, etc...
```

Bank Account Class(6)

6. Test your class with a main() that calls all functions:

```
int main()
{
   BankAccount harrys_account(1000);
   harrys_account.deposit(500); // Balance $1500
   harrys_account.withdraw(2000); // Balance $1490
   harrys_account.add_interest(1);
              // Balance $1490 + 14.90
   cout << fixed << setprecision(2)
      << harrys_account.get_balance() << endl;
   return 0;
}
```

Program Run

1504.90