**Recitation 4**

**Topics**

* Classes - design and implementation
* Review of Vectors

**The problem:**

We have just been hired by a sandwich making company to write modeling code for them.  
Specifically they are interested in the order taking, sandwich making, delivery truck loading and delivery and return of the delivery trucks. We'll do this in steps, of course, as we always think incremental development.  
  
Our job is to create software tools that the company's programmers can use.  
In testing our code, we'll be writing code a lot like what their programmers might write but our job is really only to provide the things they will need.

**Task ONE: just the sandwich - where else should we start?**

This company makes only one kind of sandwich: cheese sandwich.  
There's no choice of what kind of cheese but a sandwich does consist of:

* two slices of bread (no choice of kind here either)
* one slice of cheese (only one kind of cheese: unspecified)
* mayonnaise (the usual amount is 1 ounce)
* 2 slices of tomato
* mustard (the usual amount is .05 oz.)

**Q:** (open for group discussion) How many things is a sandwich?  
**Q:** When we deliver a sandwich should we first send out the bread truck, then send out the cheese truck, then the mayo truck, then the tomato truck and finally the mustard truck?  
**Q:** What are we getting at with the two previous questions?

What to write for Task One:

* a CLASS that will represent the ONE thing in this Task: a sandwich - or any sandwich this company makes. The TYPE that will be the best thing their programmers can have as a software model of the company's sandwich.

**Q:** What's a good name for the class?  
**Q:** What style should you use in this course for class names?

Note that we are not worrying about anything but the "shape" of the sandwich object - the data members.

Notice also that we don't care at this point what's actually stored in a sandwich object. We should *correctly* see garbage values in each field. We are only defining the type to hold the data; nothing more.

**Q:** Should these data members be public: or private:?  
**Q:** What is the only difference - *in the language* - between a struct and a class?  
**Q:** What is the typical usage of a struct and a class?  
**Q:** Should there ever be any public: data members in a class?

Define the class for a sandwich.  
Test your class by creating an object of your type in main.  
Start the debugger and look (in RAM) into the internal structure of the object you have created.

You will need to show that you can use the debugger with classes and your lab worker will ask you to demonstrate that you can. If you don't know how, your lab worker will show you.

**Task TWO: display method**

Most classes are given a "display" method that is used at least during the development phase so do that in this Task.  
The actual format of what a sandwich's display looks like is just a UI issue. You'd normally consult with the company and the programmers involved there but for now, just print the amounts of each ingredient on a separate line, well labeled, of course.

**Q:** Should a display method be able to accidentally or deliberately change the data members?

Make *sure* your display method asks the compiler to check for this.

Write the display method for this class.  
Test your object created in main from the previous Task.  
Should be the same garbage - good.

**Task THREE: initialization of an object - 1st thought**

Do we really like garbage?  
No. We often initialize variables so that we are in control of their initial values.

Set the mayo value to 1 by what looks like the normal initialization process:

class Sandwich {

...

double mayo( 1 ); // or: mayo = 1;

...

**Prior** to C++11, this did **not** work.  If you are using a C++11 compliant compiler, then it should.

**Task FOUR: initialization of an object - the constructor and the member initialization list  
customizeable initialization of an object**

**Q:** What is the purpose of a constructor?  
**Q**: Does the constructor really construct?  
**Q:** What are the (3) rules for a constructor? (What makes them different than any other method?)  
**Q:** Should the constructor be public ? What if it's not?  
**Q:** How do we "call" a constructor when we define a variable of class type?  
**Q:** What is the member initialization list for?  
**Q:** Do we have to use the member initialization list here?

* For this Task, consider only the mayo data member.
* Write the constructor and *initialization list* so that the mayo amount is initialized to 0.1 ounces.
* To test your code, create an instance of a Sandwich - (what is an instance?). Examine the mayo amount in the debugger *and* using your display method.
* Show, using the debugger, that the constructor is actually called. How?

You could have used either the assignment to member variables in the constructor's body or the member initialization list for this Task. Eventually you will be required (by C++) to use the initialization list for certain things so we suggest getting used to it but **if** it has not been covered in class at this point, you may do the initialization either way.

**Task FIVE: initialization of an object by the definer of the object  
customizeable objects without specifying some parameters**

Sometimes in this company, someone who calls in and orders a sandwich says things like: "I want extra mayo" or "Lots of mustard, please".  
We need to be able to create a sandwich that allows this non-normal sandwich - without first building a normal sandwich and then adding or, worse, scraping off ingredients.  
The company did some research and determined that this is the order of most often requested changes from the normal sandwich configuration:

|  |  |
| --- | --- |
| **ingredient** | **how often a change from the normal is requested** |
| tomatoes | most often |
| cheese | less often than tomatoes |
| mustard | less often than tomatoes or cheese |
| mayonnaise | hardly ever changed at all |

This order should be set up for creating sandwiches so that most of the time, creating a sandwich uses the normal values.   
  
**Q:** What is the relationship between the parameters in the constructor and the data members?  
**Q:**How do we "call" a constructor when we define a variable of class type?  
**Q:** Can parameters have default values?  
**Q:** What are the rules about default parameter values?  
Q: If all parameters have default values, what kind of constructor do we then have?  
**Q:** In what order should parameters with default values be placed?

Notice that we are providing a model of a sandwich that can be created with various ingredient configurations.  
Our job here is to make the sandwich object 'makeable' with various ingredient configurations but allow the normal sandwich to be easily built (without having to specify those values).

Modify the constructor to allow this sort of initialization.

**Task SIX: finding out the values individual ingredient amounts**

When we get to the code for modeling a loaded delivery truck being sent off into the night to make deliveries, the company wants to be able to have a check on what's on the truck. That's pretty easy, we'll just write code to display each sandwich in the truck. But the company wants even more checking: they want the ability to look at a specific sandwich's mayo content. They will ask their workers to manually check their order forms against what a sandwich actually has. They want this for every ingredient. We must provide a way for a programmer to get at the mayo (and every other) member in a sandwich.

**Q:** What is an accessor method?  
**Q:** How many accessor methods should there be in "an object"?  
**Q:** ...in this object?  
**Q:** Should accessor methods be allowed to change the data they are giving to their callers?

Write accessor methods for each ingredient and test the code.  
Show your lab worker that they work with your one sandwich in main.

**Task SEVEN: changing individual ingredient amounts**

Sometimes after the sandwich has been ordered and built, the customer changes his mind: "Oh, sorry. I meant to say 'hold the mayo and hold the mustard.'" Of course the company obliges. It's easy because the mayo is on one piece of bread and the mustard on the other so they just split the sandwich apart and scrape off whatever needs to be removed and put on fresh tomatoes if needed. Sometimes the customer's change is for extra helpings of ingredients. The company wants to please so our software model of their sandwich must be able to do this changing after creation.

**Q:** What is a mutator?  
**Q:** How many mutators should there be in "an object"?  
**Q:** ...in this object?  
**Q:** What about the bread?  
**Q:** Should mutator methods be allowed to change the data in the object?  
**Q:** Shouldn't the order taking code handle all changes before we get to the build a sandwich code?

Yes, that would be better but it's not the way this company works.  
We have to model the way this company works and they build first and then change if needed.  
So we must provide those tools.

Write mutator methods for each ingredient and test the code.  
Show your lab worker that they work with your one sandwich in main.

**Task EIGHT: what about the bread?**

**Q:** Does the number of bread slices ever change in this company's sandwiches?  
**Q:** Can a data member be const?  
**Q:** How can a const data member be initialized?  
**Q:** Should there / could there be a mutator for the bread amount?  
**Q:** Should there even be a parameter in the constructor for the bread amount?  
**Q:** Should there still be an accessor since "*we*" know that it is always two slices?  
Q: What does a vector really do when you do a push\_back?

Make the bread's number of slices const.   
What problems will this const data member introduce?  
You might need to reconsider having a mutator method for slices.  
After creating a sandwich object to test your const data member, try putting one into your vector.  
What's going on? Shouldn't the copy semantics of push\_back cause an compiler error - assigning to a const data member?

**Task TEN: delivering the sandwiches - the truck**

Q: How should be model the delivery truck?  
Q: How about a vector< Sandwich > member variable  
Q: How can we have more than one truck?  
Q: Isn't a truck a thing so we should create a class for it and create separate objects of that type when ?  
Q: Again what about this copying of sandwiches?

Recall the company wants us to be able to show all the ingredients in all the sandwiches that have been loaded on the truck just before we send it out to make deliveries. Be sure to consider that. Is checkBeforeLeaving a thing the truck needs to know how to do or should that be code in main?

And how did those sandwiches get into the that truck in the first place? Should there be an insertSandwich method in a truck class?

What other things should a programmer who wants to use this truck class be able to do?  
Are those the methods you must write?

Create the class SandwichTruck.  
Show your worker that it works.

**Task ELEVEN: use the tools you've created to model the sandwich making world**

Now you'll pretend that you are one of the programmers for the company and use your code like they would in modeling their sandwich and order-taking and delivery truck worlds.

We have scenarios that are things that might happen in the sandwich world and we need to write code, in main, to model this sequence of events.

Here is a scenario:

customer 1 orders a regular sandwich with no changes.   
customer 2 orders a regular sandwich then changes her mind to "no mustard".  
customer 3 orders a sandwich with extra cheese (extra means one more slice than regular) and has no changes.  
customer 4 orders a regular sandwich with no changes.  
the delivery truck is checked out before it leaves  
the mustard amount in customer 2's sandwich is checked

Use the tools you've created and write the code to model this scenario.  
Where will you write this code?

[Home](http://cis.poly.edu/cs1124)

Top of Form

Bottom of Form