Classes/Encapsulation (+STL, maps, iterators, different versions of C++)

STL (Standard Template Library):

- A collection made up of: algorithms, containers, functions and iterators

- We have already used one of the containers: vector
  - Containers can hold different types and objects
  - Notice that we have been able to use vectors with different types specified in < >
  - There are other containers that we can use (look up STL containers)

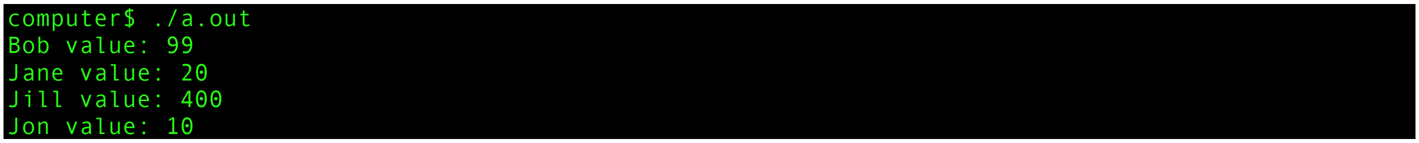
Note: We will talk more in depth about what an iterator and a template actually are in a future lecture. For today, we will just be using them.

Example 1: maps

- A map is another type of container in STL (Standard Template Library)
  - Remember vector is also a container in the STL

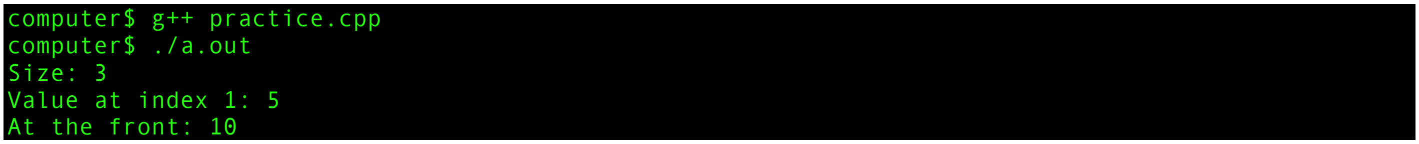
- A map works by storing values with a key
  - We then access our values using this key

- We can use something called an iterator to move to each element in our map
  - Note we can also use an iterator on a vector
    - Next lecture, I will show an example of this
  - Think of it like a pointer
    - Remember the notation -> from pointers means *x.member



```
computer$ ./a.out
Bob value: 99
Jane value: 20
Jill value: 400
Jon value: 10
```

```
#include <iostream>
#include <string>
#include <map>

int main (int argc, char **argv)
{
  //creating a map
  std::map<std::string,int> map_example = {{ "Jon", 100 }, { "Jane", 90 },{ "Jill", 50 }};

  map_example.at("Jon") = 20; //assign value (20) at this key (Jon)
  map_example.at("Jane") = 20;

 map_example["Jill"]=400; //can also give value at key like this-similar to vector v[1] for example except
 map_example.insert({"Bob", 99});
 map_example.insert({"Jon", 99}); //won't get inserted because we already have a key Jon

  //Notice when declaring our iterator we say:  map<std::string, int>::iterator.  Iterator is a type inside our map class so we access it using ::
  //begin() returns an iterator to the first element, end() returns an iterator to the last element
 for (std::map<std::string, int>::iterator it=map_example.begin(); it!=map_example.end(); it++)
 {
  std::cout << it->first << " value: " << it->second << std::endl; //first is the key, second is the value
//Note: our map class has a type called value_type.  This is defined as a pair<key, value> (pair is a class template).  First accesses the key and second accesses the value.  Our iterator is pointing at a pair, so when we dereference, we need to specify which in the pair we are looking at.
 }

}
```



```
computer$ g++ practice.cpp
computer$ ./a.out
Size: 3
Value at index 1: 5
At the front: 10
```

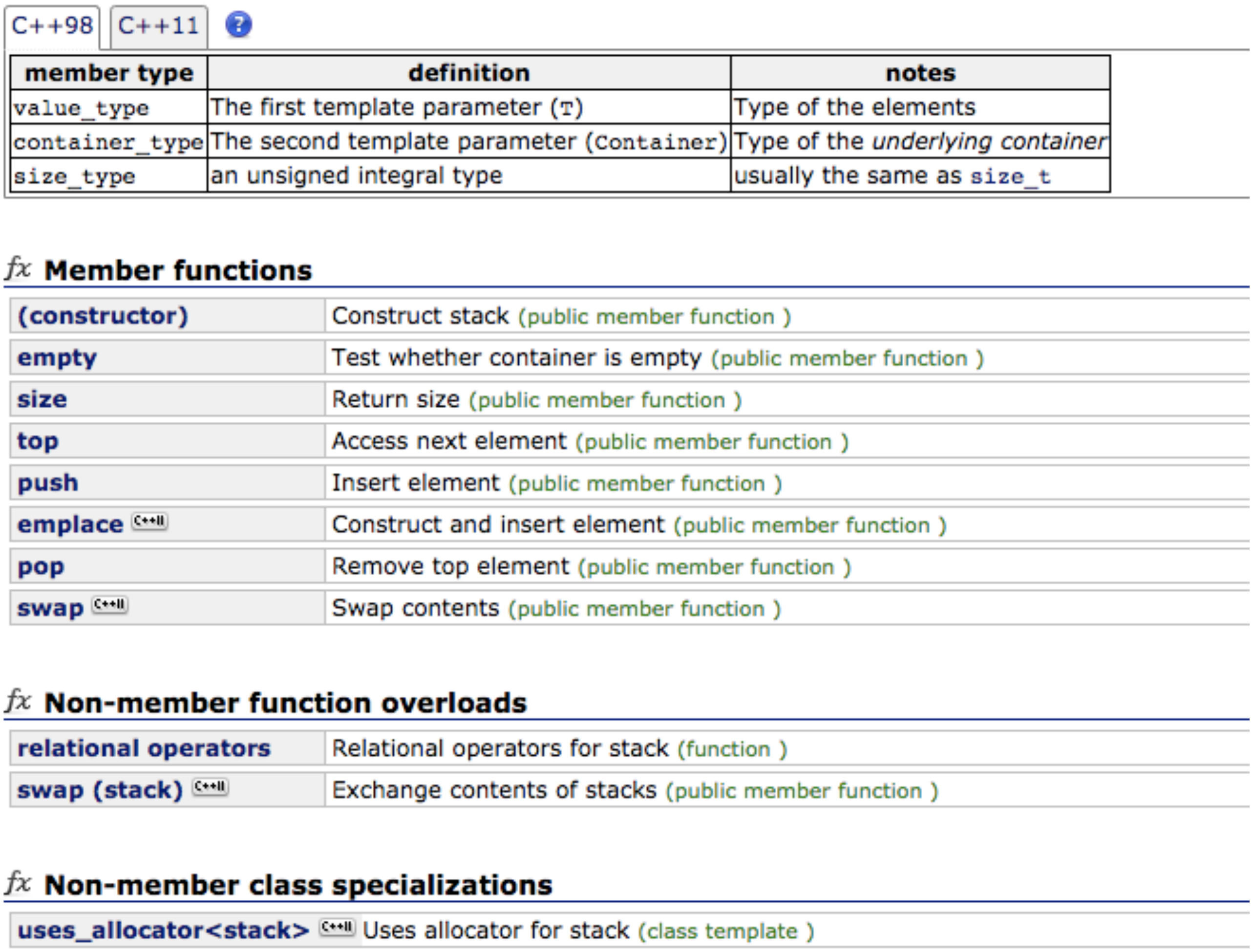
Example 2: dequeue

```
#include <iostream>
#include <deque> //you can look up the deque class online.  It is a container class like vector

using namespace std;

int main(int argc, char **argv)
{
   deque <int> deque_example; //specifying it to use ints (looks like vector)
   deque_example.push_back(5); //push to the back
   deque_example.push_front(10); //push to the front
   deque_example.push_back(15); //push to the back, after 5

   //order: 10,5,15

   cout << "Size: " << deque_example.size();

   cout << "\nValue at index 1: " << deque_example.at(1);
   cout << "\nAt the front: " << deque_example.front();

}
```

Different Versions of C++:

- Just like a spoken language evolves over time, programming languages do also
  - For example, we don’t speak English the same way it was spoken 100 years ago

- When a programming language evolves, it is usually to add in helpful features that make coding in the language easier
    - Notice below that some of the functions are only available in C++11 (a later variation of C++)



C++98 | C++11 | ?

member type	definition	notes
value_type	The first template parameter (T)	Type of the elements
container_type	The second template parameter (Container)	Type of the *underlying container*
size_type	an unsigned integral type	usually the same as size_t

fx Member functions

(constructor)	Construct stack (public member function)
empty	Test whether container is empty (public member function)
size	Return size (public member function)
top	Access next element (public member function)
push	Insert element (public member function)
emplace C++11	Construct and insert element (public member function)
pop	Remove top element (public member function)
swap C++11	Swap contents (public member function)

fx Non-member function overloads

relational operators	Relational operators for stack (function)
swap (stack) C++11	Exchange contents of stacks (public member function)

fx Non-member class specializations

uses_allocator<stack> C++11	Uses allocator for stack (class template)

http://www.cplusplus.com/reference/stack/stack/

- Different compilers handle different versions of the language
    - They were written to handle the language at the time
        - They can also usually handle earlier versions

    - The C++ compiler on Omega is old and cannot handle C++11 or any evolution after, for example
        - The compiler on the older virtual machine link given can handle up to C++14
        - The compiler on the newer virtual machine can handle up C++17

- You can specify the version of C++ you want to use
    - Compilers can usually handle up to a specific language version

- More info: http://www.stroustrup.com/C++.html#standard

The compiler on my computer can handle up to c++14 (I get an error if I try to compile using C++17):

```
computer$ g++ -std=c++11 practice.cpp
computer$ g++ -std=c++14 practice.cpp
computer$ g++ -std=c++17 practice.cpp
```

```
error: invalid value 'c++17' in '-std=c++17'
```

I can see the default version of my compiler using the following program:

```
#include <iostream>

int main(int argc, char **argv)
{
 std::cout << __cplusplus << std::endl;
}
```

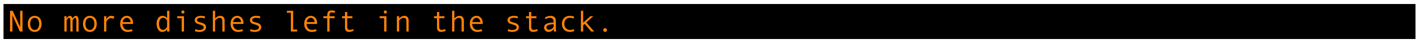
```
computer$ g++ practice.cpp
computer$ ./a.out
199711  //default is C++98
computer$ g++ -std=c++11 practice.cpp
Computers-MacBook-Air:C++ computer$ ./a.out
201103
```

***Note**: I will be showing you guys how to use different containers, but make sure to know how to use maps and vectors (part of the course objectives)*

Program 1:

Create a stack of dishes. Dishes fall off the stack if someone clumsy comes by.

```
computer$ g++ -std=c++11 practice.cpp
computer$ ./a.out
1
Pressy added a dish.

Number of dishes in the stack: 2
3
Vicki added a dish.

Number of dishes in the stack: 3
2
!!!Crash!! A plate is broken because anonymous is clumsy :(

Number of dishes in the stack: 2
2
!!!Crash!! A plate is broken because anonymous is clumsy :(

Number of dishes in the stack: 1
2
!!!Crash!! A bowl is broken because anonymous is clumsy :(

Number of dishes in the stack: 0
```



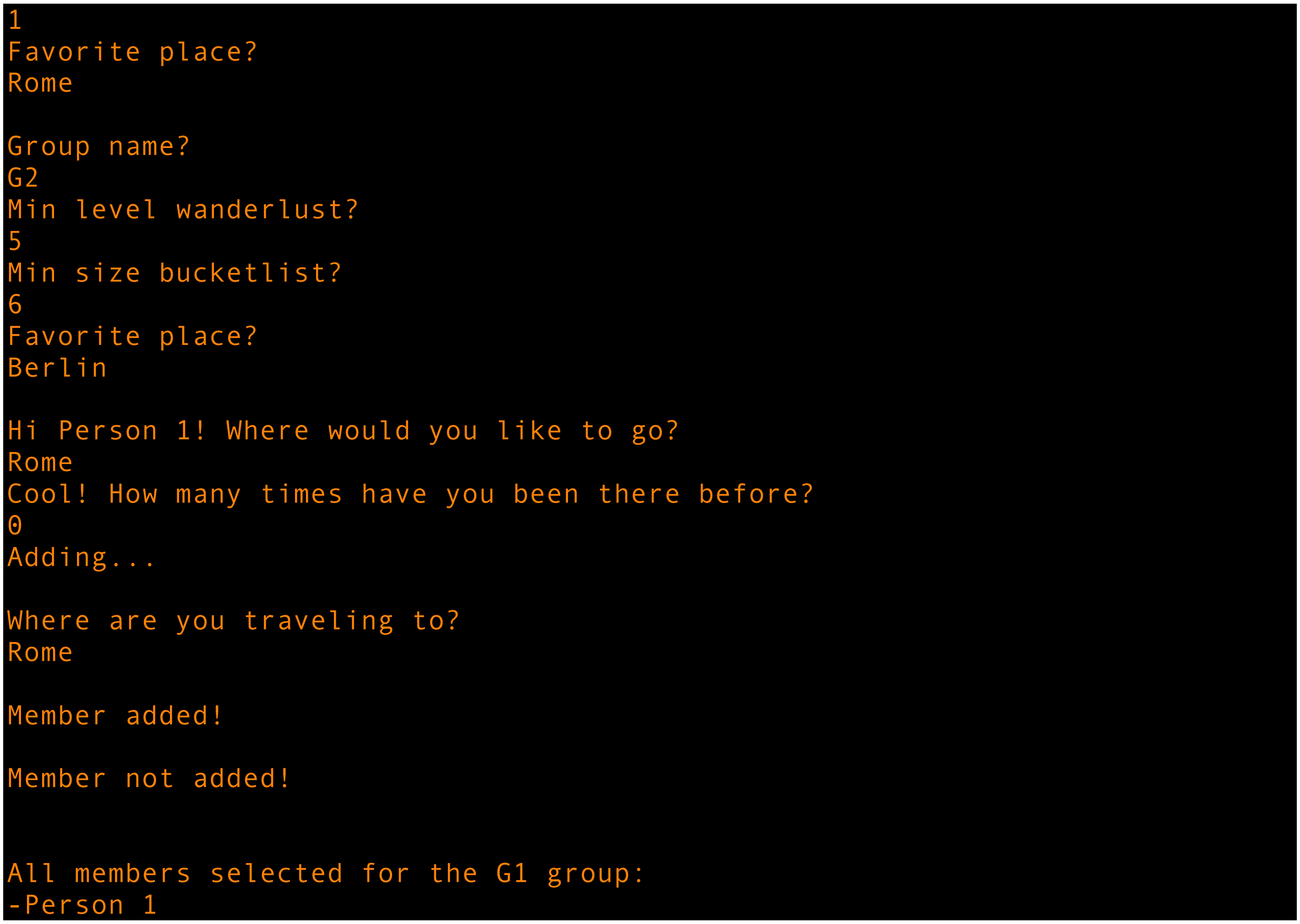
```
No more dishes left in the stack.
```

```
#include <iostream>
#include <stack>
#include <vector>
#include <string>

class Person{

public:
    std::string name="anonymous"; //you can do this in c++11 and up
    bool clumsy=true; //you can do this in c++11 and up…true means clumsy, false means not

};

class Dish{

    std::string type;
    float price; //I don’t end up using price in this program, but keeping it here

public:
    Dish(std::string dish_type, float dish_price)
    {
        type=dish_type;
        price=dish_price;
    }

    void fall_off(Person p)
    {
        std::cout<<"!!!Crash!! A "<< type <<" is broken because "<<p.name<<" is clumsy :("<<std::endl;


    }
};

int main(int argc, char ** argv)
{
    int answer;
    Person p1;
    Person p2;
    Person p3;

    p1.name="Pressy";
    p1.clumsy=false;

    p3.name="Vicki";
    p3.clumsy=false;
    Dish d1("bowl", 3.44);

    std::vector<Person> all_people={p1,p2,p3}; //initialize vector (c++11)
    std::stack<Dish> all_dishes; //create a stack of Dish objects
    all_dishes.push(d1); //add a dish to the stack

    while(!all_dishes.empty()) //keeps going for as long as there are dishes in the stack
    {
        std::cin>>answer;

        //pass in person at index (minus 1).  Note this program does not check to make sure the
number entered is a valid index
        if(all_people.at(answer-1).clumsy)
        {
            all_dishes.top().fall_off(all_people.at(answer-1));
            all_dishes.pop();
            std::cout<<"\nNumber of dishes in the stack: "<<all_dishes.size()<<std::endl;
        }

        else
        {
            Dish d("plate",2.99);
            all_dishes.push(d);
            std::cout<<all_people.at(answer-1).name<<" added a dish."<<std::endl;
            std::cout<<"\nNumber of dishes in the stack: "<<all_dishes.size()<<std::endl;
        }

    }

    std::cout<<"\nNo more dishes left in the stack."<<std::endl;
}
```

Program 2:

Create a program of travelers joining a travel group.

```
computer$ g++ -std=c++11 practice.cpp
computer$ ./a.out

Group name?
G1
Min level wanderlust?
1
Min size bucketlist?
```



```
1
Favorite place?
Rome

Group name?
G2
Min level wanderlust?
5
Min size bucketlist?
6
Favorite place?
Berlin

Hi Person 1! Where would you like to go?
Rome
Cool! How many times have you been there before?
0
Adding...

Where are you traveling to?
Rome

Member added!

Member not added!


All members selected for the G1 group:
-Person 1
```

```
#include <iostream>
#include <map>
#include <string>
#include <vector>

class Traveler{

        std::string name;
        int level_wanderlust;

public:
        std::map<std::string, int> bucket_list; //place, number of times visited

        Traveler(std::string n, int level)
        {
                name=n;
                level_wanderlust=level;
        }

        //won't add duplicates.  Well discuss in a future lecture how to indicate something was a duplicate
        void add_new_place()
```

```
    {
        std::string answer;
        int number_times;
        std::cout<<"\nHi "<<name<<"! Where would you like to go?"<<std::endl;
        getline(std::cin,answer);
        std::cout<<"Cool! How many times have you been there before?"<<std::endl;
        std::cin>>number_times;
        std::cin.ignore();
        std::cout<<"Adding...\n"<<std::endl;
        bucket_list.insert({answer,number_times});
    }

    //assume that the user always enters a place already in the list-well discuss in a future lecture how to check if a key actually exists or not
    void travel()
    {
        std::string answer;
        std::cout<<"Where are you traveling to?"<<std::endl;
        getline(std::cin, answer);
        bucket_list.at(answer)++;

    }

    //getter
    int get_wanderlust_level()
    {
        return level_wanderlust;
    }

    //setter
    void set_wanderlust_level()
    {
        std::cout<<"How do you currently feel about traveling?  Level 0-5: "<<std::endl;
        std::cin>>level_wanderlust;
    }

    std::string get_name()
    {
        return name;
    }


};

class Travel_group
{
    std::string group_name;
    std::vector<Traveler> all_members;
    int min_level_wanderlust;
    int min_size_bucketlist;
    std::string favorite_place;

public:
    Travel_group()
    {
        std::cout<<"\nGroup name?"<<std::endl;
        std::cin>>group_name;

        std::cout<<"Min level wanderlust?"<<std::endl;
        std::cin>>min_level_wanderlust;

        std::cout<<"Min size bucketlist?"<<std::endl;
        std::cin>>min_size_bucketlist;

        std::cout<<"Favorite place?"<<std::endl;
        std::cin>>favorite_place;

        std::cin.ignore();
    }


    bool add_member(Traveler t) //true means accept, false means reject
    {
        bool ret;
        //check if wanderlust level is greater than or equal to min level required by group AND
number of places on bucketlist exceeds number given
        //bucket list is public, so we can directly access it
        if(min_level_wanderlust<=t.get_wanderlust_level() &&
min_size_bucketlist<=t.bucket_list.size())
        {
            ret=true;
            all_members.push_back(t);
        }

        //at least one of them is true (wanderlust level OR min size of bucket list) remember if both
were true, we would have caught it with the first if statement
        else if(min_level_wanderlust<=t.get_wanderlust_level() ||
min_size_bucketlist<=t.bucket_list.size())
        {
            ret=false;
            //willing to accept if the traveler has been the favorite place at least 1 time.
            //using an iterator to go through list (we could def do the other way shown before)
            //also note you could declare the iterator outside of the for loop then use it
            for (std::map<std::string, int>::iterator it=t.bucket_list.begin(); it!=t.bucket_list.end();
it++)
            {
                if(it->first==favorite_place && 1<=it->second) //found matching favorite
matching place and check if number of times is at least once
```

```
                {
                    ret=true; //only becomes true if this occurs
                    all_members.push_back(t);
                }
            }
        }

        else
        {
            ret=false;
        }

        return ret;
    }

    void show_all_members()
    {
        std::cout<<"\n\nAll members selected for the "<<group_name<<" group:"<<std::endl;

        //remember you can also do this without an iterator (by using [] or at)-just showing you guys
        for (std::vector<Traveler>::iterator it=all_members.begin(); it!=all_members.end(); it++)
        {
             std::cout<<"-"<<it->get_name()<<std::endl;
        }
    }

};


int main(int argc, char ** argv)
{
    Travel_group g1;
    Travel_group g2;

    Traveler t1("Person 1",3);
    Traveler t2("Person 2",4);

    t1.add_new_place();
    t1.travel();



    bool b=g1.add_member(t1);

    if(b)
    {
        std::cout<<"\nMember added!"<<std::endl;
    }
    else
    {
        std::cout<<"\nMember not added!"<<std::endl;
    }


    b=g1.add_member(t2);

    if(b)
    {
        std::cout<<"\nMember added!"<<std::endl;
    }

    else
    {
        std::cout<<"\nMember not added!"<<std::endl;
    }

    g1.show_all_members();


}
```