**Lab 1: STL containers**

**Please read this in Chrome so that hints are collapsed by default!**

**1. STL Vector**

In this one, you will use the [STL Vector](http://www.cplusplus.com/reference/vector/vector/) object.

**Starter code**

#include <iostream>

#include <vector>

#include <string>

using namespace std;

void AddIngredients( vector<string>& ingredients )

{

// Use the push\_back function to add "lettuce", "tomato", "mayo", "mustard", and "bread" to the vector.

}

void DisplayIngredients( const vector<string>& ingredients )

{

// Use a for-loop, going from 0 to the ingredient's size(), to display each of the ingredients

}

int main()

{

// 1. Declare a vector of strings named ingredients.

// 2. Call the AddIngredients function, passing in ingredients as an argument.

// 3. Call the DisplayIngredients function, passing in ingredients as an argument.

return 0;

}

**Instructions**

**main()**

In **main**, create a vector of strings called **ingredients**.

Afterward, call the **AddIngredients** function, passing in ingredients as an argument.

Finally, call the **DisplayIngredients** function, passing in ingredients as an argument.

**AddIngredients**

Within this function, use the vector's **push\_back** function to add a series of strings to your ingredients vector, such as *lettuce, tomato, mayo, mustard, and bread*.

**DisplayIngredients**

Within this function, use a *for-loop* to iterate through all the elements of the ingredients vector. You can use the vector's **size** function to get the amount of items in the vector, and iterate from 0 to that item count.

Use **cout** to display the current ingredient. You can access an item at index *i* in the vector with a subscript, just like with an array.

**Solution**

***Solution***

#include <iostream>

#include <vector>

#include <string>

using namespace std;

void AddIngredients( vector<string>& ingredients )

{

// Use the push\_back function to add "lettuce", "tomato", "mayo", "mustard", and "bread" to the vector.

ingredients.push\_back( "lettuce" );

ingredients.push\_back( "tomato" );

ingredients.push\_back( "mayo" );

ingredients.push\_back( "mustard" );

ingredients.push\_back( "bread" );

}

void DisplayIngredients( const vector<string>& ingredients )

{

// Use a for-loop, going from 0 to the ingredient's size(), to display each of the ingredients

for ( unsigned int i = 0; i < ingredients.size(); i++ )

{

cout << ingredients[i] << endl;

}

}

int main()

{

// 1. Declare a vector of strings named ingredients.

vector<string> ingredients;

// 2. Call the AddIngredients function, passing in ingredients as an argument.

AddIngredients( ingredients );

// 3. Call the DisplayIngredients function, passing in ingredients as an argument.

DisplayIngredients( ingredients );

return 0;

}

**2. STL List**

In this one, you will use the [STL List](http://www.cplusplus.com/reference/list/list/) object.

**Starter code**

#include <iostream>

#include <string>

#include <list>

using namespace std;

class Student

{

public:

Student( const string& name, int grade ) : m\_name( name ), m\_grade( grade ) { }

void Display();

bool operator>( const Student& other );

bool operator<( const Student& other );

bool operator>=( const Student& other );

bool operator<=( const Student& other );

bool operator==( const Student& other );

private:

string m\_name;

int m\_grade;

};

int main()

{

// Create a list of Student objects called students

list<Student> students;

// Push multiple students into the list.

// students.push\_back( Student( "Vid", 100 ) );

// Call the sort() function

// Call the reverse() function to make it descending.

// Display all the students

for ( list<Student>::iterator it = students.begin(); it != students.end(); it++ )

{

it->Display();

}

return 0;

}

void Student::Display()

{

cout << m\_grade << "\t" << m\_name << endl;

}

bool Student::operator>( const Student& other )

{

return this->m\_grade > other.m\_grade;

}

bool Student::operator<( const Student& other )

{

return this->m\_grade < other.m\_grade;

}

bool Student::operator>=( const Student& other )

{

return this->m\_grade >= other.m\_grade;

}

bool Student::operator<=( const Student& other )

{

return this->m\_grade <= other.m\_grade;

}

bool Student::operator==( const Student& other )

{

return this->m\_grade == other.m\_grade;

}

**Instructions**

**main()**

In **main**, create a list of Student objects, called **students**.

Use the list's **push\_back** function to add a series of students to the list.

Hint: You can insert a new student without creating a Student variable local to main() by passing in a constructor with its arguments, like this:

blahblahblah( Student( "Vid", 100 ) );

Add several students and grades in random order.

Afterward, call the **sort** and **reverse** functions on the list.

Sort will sort it, ascending, and reverse will change it to descending order.

A for loop with an iterator is already provided - it might look strange! If you haven't seen an iterator before, that's OK. But with lists, we can't iterate through it like a vector; we can't randomly access items with the subscript operator [ ] so we have to use the iterator.

**Solution**

***Solution***

#include <iostream>

#include <string>

#include <list>

using namespace std;

class Student

{

public:

Student( const string& name, int grade ) : m\_name( name ), m\_grade( grade ) { }

void Display();

bool operator>( const Student& other );

bool operator<( const Student& other );

bool operator>=( const Student& other );

bool operator<=( const Student& other );

bool operator==( const Student& other );

private:

string m\_name;

int m\_grade;

};

int main()

{

list<Student> students;

students.push\_back( Student( "Vid", 100 ) );

students.push\_back( Student( "Katarina", 50 ) );

students.push\_back( Student( "Christy", 80 ) );

students.push\_back( Student( "Hakan", 70 ) );

students.push\_back( Student( "Naoki", 90 ) );

students.push\_back( Student( "Thankarat", 75 ) );

students.sort();

students.reverse();

for ( list<Student>::iterator it = students.begin(); it != students.end(); it++ )

{

it->Display();

}

return 0;

}

void Student::Display()

{

cout << m\_grade << "\t" << m\_name << endl;

}

bool Student::operator>( const Student& other )

{

return this->m\_grade > other.m\_grade;

}

bool Student::operator<( const Student& other )

{

return this->m\_grade < other.m\_grade;

}

bool Student::operator>=( const Student& other )

{

return this->m\_grade >= other.m\_grade;

}

bool Student::operator<=( const Student& other )

{

return this->m\_grade <= other.m\_grade;

}

bool Student::operator==( const Student& other )

{

return this->m\_grade == other.m\_grade;

}

**3. STL Queue**

In this one, you will use the [STL Queue](http://www.cplusplus.com/reference/queue/queue/) object.

**Starter code**

#include <iostream>

#include <string>

#include <queue>

using namespace std;

int main()

{

float balance = 0.0;

// Create a queue of floats, named transactions

// Push some + and - floats into the transactions queue.

// Loop through all the transactions - while the queue is empty.

// Within the loop, whatever amount is at the front of the queue,

// add it to the balance and then pop it off the queue.

cout << "Final balance: $" << balance << endl;

return 0;

}

**Instructions**

**main()**

In **main**, create a queue of floats called **transactions**.

Push a series of values into the queue: both positive and negative values. These will be deposits and withdraws into an account.

Create a while loop that will *continue looping while the transactions queue is not empty.* Use the queue's **empty** function to see if the queue is empty or not.

Within the while loop, you will take the front item of the queue (with the **front** function) and add it to the balance. Afterward, you will **pop** the front value off the queue.

At the end, it will display the final balance after all the deposits and withdraws.

**Solution**

***Solution***

#include <iostream>

#include <string>

#include <queue>

using namespace std;

int main()

{

float balance = 0.0;

queue<float> transactions;

transactions.push( 100.00 );

transactions.push( -5.00 );

transactions.push( 50.00 );

transactions.push( -20.00 );

while ( !transactions.empty() )

{

float amount = transactions.front();

cout << amount << " pushed to account" << endl;

balance += amount;

transactions.pop();

}

cout << "Final balance: $" << balance << endl;

return 0;

}