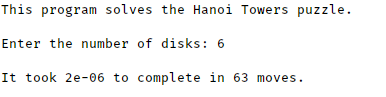
**

*/\*--- hanoi.cpp ----------------------------------------------------------*

*Program* *to* *solve* *the* *Towers* *of* *Hanoi* *puzzle* *recursively.*

*Input:* *numDisks,* *the* *number* *of* *disks* *to* *be* *moved*

*Output:* *A* *sequence* *of* *moves* *that* *solve* *the* *puzzle*

*Written* *by:* *Larry* *R.* *Nyhoff*

*Written* *for:* *Lab* *Manual* *for* *ADTs,* *Data* *Structures,* *and* *Problem*

*Solving* *with* *C++,* *2E*

*Lab* *#9.1*

*Johnathan* *Lee* *CSCI* *1107*

*Lab* *13* *Due* *04/24/18*

*-------------------------------------------------------------------------\*/*

#include <iomanip>

#include <iostream>

#include "Timer.h"

*using* *namespace* std;

void **move**(unsigned n, unsigned& moveNumber, char source, char destination,

char spare);

int **main**() {

*const* char PEG1 = 'A', *//* *the* *three* *pegs*

PEG2 = 'B', PEG3 = 'C';

unsigned moveNumber = 0; *//* *counts* *the* *moves*

cout << "This program solves the Hanoi Towers puzzle.\n\n";

cout << "Enter the number of disks: ";

unsigned numDisks; *//* *the* *number* *of* *disks* *to* *be* *moved*

cin >> numDisks;

cout << endl;

Timer t;

t.start();

move(numDisks, *moveNumber*, PEG1, PEG3, PEG2); *//* *the* *solution*

t.stop();

cout << "It took " << t.seconds() << " to complete in " << moveNumber

<< " moves.\n";

}

void **move**(unsigned n, unsigned& moveNumber, char source, char destination,

char spare)

*/\*-------------------------------------------------------------------------*

*Recursive* *function* *to* *solve* *the* *Towers* *of* *Hanoi* *puzzle* *with* *n* *disks.*

*Precondition:* *None.*

*Postcondition:* *A* *message* *describing* *the* *move* *is* *output* *to* *cout.*

*------------------------------------------------------------------------\*/*

{

*if* (n == 1) *//* *anchor*

{

moveNumber++;

*/\*cout* *<<* *setw(3)* *<<* *moveNumber*

*<<* *".* *Move* *the* *top* *disk* *from* *"* *<<* *source*

*<<* *"* *to* *"* *<<* *destination* *<<* *endl;\*/*

} *else* { *//* *inductive* *case*

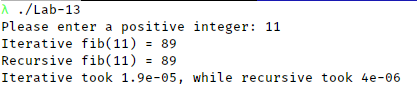
move(n - 1, *moveNumber*, source, spare, destination);

move(1, *moveNumber*, source, destination, spare);

move(n - 1, *moveNumber*, spare, destination, source);

}

}

**

*/\*---* *fibonacci.cpp* *--------------------------------------------------------*

*Program* *to* *compute* *the* *Fibonacci* *function* *both*

*iteratively* *and* *recursively.*

*Input:* *a* *positive* *integer* *x*

*Output:* *the* *x-th* *Fibonacci* *number*

*Written* *by:* *Larry* *R.* *Nyhoff*

*Written* *for:* *Lab* *Manual* *for* *ADTs,* *Data* *Structures,* *and* *Problem*

*Solving* *with* *C++,* *2E*

*Lab* *#9.1*

*Johnathan* *Lee* *CSCI* *1107*

*Lab* *13* *Due* *04/24/18*

*-------------------------------------------------------------------------\*/*

#include <iostream>

#include "Timer.h"

*using* *namespace* std;

unsigned **iterFibonacci**(unsigned n);

*/\*---------------------------------------------------------*

*An* *iterative* *(nonrecursive)* *Fibonacci* *number* *calculator*

*Precondition:* *n* *>=* *1.*

*Postcondition:* *n-th* *Fibonacci* *number* *is* *returned.*

*------------------------------------------------------------\*/*

unsigned **recFibonacci**(unsigned n);

*/\*---------------------------------------------------------*

*A* *recursive)* *Fibonacci* *number* *calculator*

*Precondition:* *n* *>=* *1.*

*Postcondition:* *n-th* *Fibonacci* *number* *is* *returned.*

*------------------------------------------------------------\*/*

int **main**() {

Timer recTimer, iterTimer;

unsigned x;

cout << "Please enter a positive integer: ";

cin >> x;

*//* *Now* *print* *the* *result* *of* *the* *iterative* *version* *of* *the* *function*

iterTimer.start();

cout << "Iterative fib(" << x << ") = " << iterFibonacci(x) << endl;

iterTimer.stop();

*//* *Now* *print* *the* *result* *of* *the* *recursive* *version* *of* *the* *function*

recTimer.start();

cout << "Recursive fib(" << x << ") = " << recFibonacci(x) << endl;

recTimer.stop();

cout << "Iterative took " << iterTimer.seconds() << ", while recursive took "

<< recTimer.seconds() << endl;

}

*//---* *Definition* *of* *iterFibonacci()*

unsigned **iterFibonacci**(unsigned n) {

int nextFib = 1, *//* *the* *next* *Fibonacci* *number* *to* *be* *calculated*

previousFib = 1, *//* *the* *Fibonacci* *number* *before* *it*

beforePreviousFib; *//* *the* *Fibonacci* *number* *before* *that* *one*

*for* (int i = 3; i <= n; i++) {

*//* *First,* *update* *the* *previous* *and* *before* *previous* *values*

beforePreviousFib = previousFib;

previousFib = nextFib;

*//* *Then* *compute* *the* *next* *Fibonacci* *value*

nextFib = previousFib + beforePreviousFib;

}

*return* nextFib;

}

*//---* *Definition* *of* *recFibonacci()*

unsigned **recFibonacci**(unsigned n) {

*if* (n <= 2)

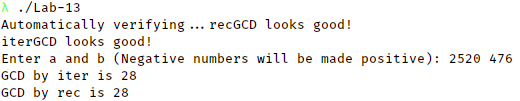
*return* 1;

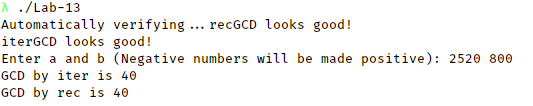
*else*

*return* recFibonacci(n - 1) + recFibonacci(n - 2);

}

Sample executions for GCD functions:





/// \file gcd.cpp

/// \author Johnathan Lee for CSCI 1107, Lab 13

/// \brief Euclidean function for greatest common divisor.

#include <cassert>

#include <iostream>

using namespace std;

/// \brief Recursively calculate the GCD of 2 numbers

/// \param a The first number

/// \param b The second number

/// \returns The GCD of a and b

int **recGCD**(int a, int b);

/// \brief Iteratively calculate the GCD of 2 numbers

/// \param a The first number

/// \param b The second number

/// \returns The GCD of a and b

int **iterGCD**(int a, int b);

int **main**() {

cout << "Automatically verifying...";

assert(recGCD(2520, 476) == 28);

cout << "recGCD looks good!\n";

assert(recGCD(2520, 476) == 28);

cout << "iterGCD looks good!\n";

int a, b;

cout << "Enter a and b (Negative numbers will be made positive): ";

cin >> a >> b;

// Take absolutes.

if (a < 0)

a = -a;

if (b < 0)

b = -b;

cout << "GCD by iter is " << iterGCD(a, b) << endl;

cout << "GCD by rec is " << recGCD(a, b) << endl;

}

int **recGCD**(int a, int b) {

int r = a % b;

if (r == 0 || recGCD(b, r) == 0) // Base

return r;

else // Inductive

return recGCD(b, r);

}

int **iterGCD**(int a, int b) {

int lastR, r = 1;

while (r != 0) {

lastR = r;

r = a % b;

a = b;

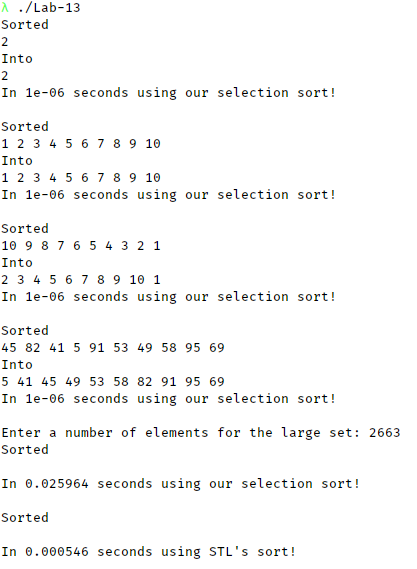
b = r;

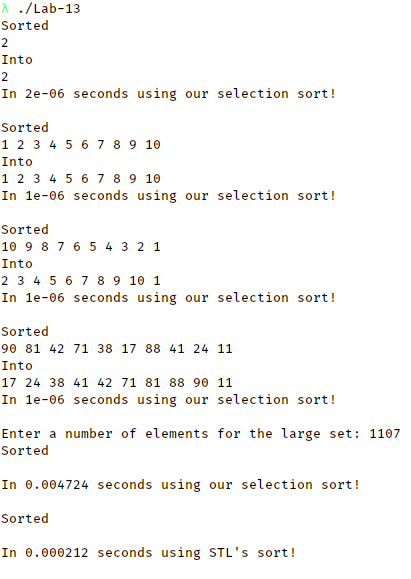
}

return lastR;

}

Sample executions of sorting functions:





/// \file SortDriver.cpp

/// \author Johnathan Lee for CSCI 1107, Lab 13

/// \brief Sorting algorithm and test driver.

#include <algorithm>

#include <iostream>

#include <vector>

#include "RandomInt.h"

#include "Timer.h"

using namespace std;

/// \brief Sort an object that provides subscript access.

/// \tparam T The type of the object we're sorting. MUST PROVIDE SUBSCRIPT

/// OPERATOR.

/// \param vec The object we're sorting

/// \param n The number of elements to sort.

/// \post vec has been sorted.

template <typename T>

void selectionSort(T& vec, int n);

/// \brief Helper function to sort array by either selectionSort or STL's sort,

/// timing it.

/// \param vec The vector to sort.

/// \param stl Should we use STL's sort?

/// \param displayContents Should we display pre-sorted and post-sorted stuff?

/// \note Probably don't do displayContents on a 30000 element list... just

/// maybe..

/// \note Pass by value to preserve old contents for next sort.

void timeSort(vector<int> vec, bool stl, bool displayContents) {

Timer timer;

cout << "Sorted \n";

if (displayContents) {

for (int& el : vec)

cout << el << " ";

cout << "\nInto \n";

}

if (!stl) {

timer.start();

selectionSort(vec, vec.size());

timer.stop();

} else {

timer.start();

sort(vec.begin(), vec.end());

timer.stop();

}

if (displayContents) {

for (int& el : vec)

cout << el << " ";

}

cout << "\nIn " << timer.seconds() << " seconds using "

<< (stl ? "STL's sort" : "our selection sort") << "!\n\n";

}

int main() {

// Initial testing for selection sort.

vector<int> singleEl = {2}, inOrder = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10},

reverseOrder = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1}, random;

RandomInt r(0, 100);

for (int i = 0; i < 10; i++)

random.push\_back(r.generate()); // from 1-100

timeSort(singleEl, false, true);

timeSort(inOrder, false, true);

timeSort(reverseOrder, false, true);

timeSort(random, false, true);

int numEls = 0;

vector<int> largeSet;

cout << "Enter a number of elements for the large set: ";

cin >> numEls;

for (int i = 0; i < numEls; i++)

largeSet.push\_back(r.generate());

// Definitely don't want to display all 25000 elements...

timeSort(largeSet, false, false);

timeSort(largeSet, true, false);

return 0;

}

// Using a simple T for the entire type as opposed to vector<T> or other such

// stuff lets it be used on BOTH arrays AND vectors, and passing in n manually

// allows sorting over a range.

template <typename T>

void **selectionSort**(**T**& **vec**, int **n**) {

for (int i = 0; i < n - 2; i++) {

int smallPos = i;

// Skipping smallest in favor of indexing.

for (int j = i + 1; j < n - 1; j++) {

if (vec[j] < vec[smallPos])

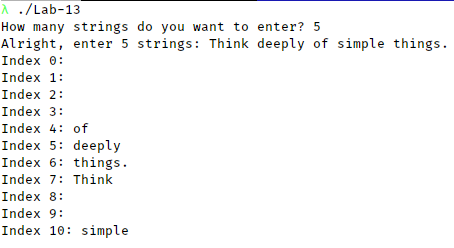
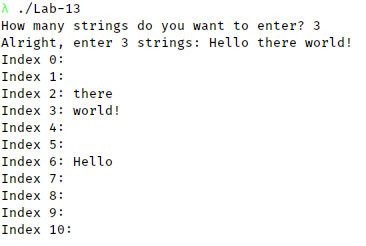
smallPos = j;

}

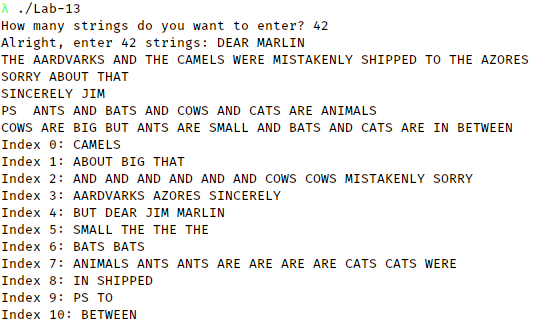
swap(*vec*[*smallPos*], *vec*[*i*]);

}

}

Test runs for HashTable stuff:

Run with the test text:



/// \file HashTable.h

/// \author Johnathan Lee for CSCI 1107, Lab 13

/// \brief Defines a HashTable class which stores strings.

#ifndef HASHTABLE\_H

#define HASHTABLE\_H

#include <iostream>

#include <string>

#include "LinkedList.h"

/// \class HashTable

/// \brief Stores strings in a hash indexed lookup table.

class HashTable {

public:

/// \brief Default constructor

/// \param size The size of the table to use.

**HashTable**(unsigned size = 11);

/// \brief Destructor

/// \note Deallocates memory AND explicitly calls LinkedList destructor.

~**HashTable**();

/// \brief Insert into the HashTable

/// \param str The string to insert.

/// \post str Is inserted into the table at the index of its hash.

void **insert**(string str);

/// \brief Display the entire HashTable

/// \param out The stream to display to.

/// \post \*this is written to out in the format:

/// Index <index>: <All items with this hash>

/// ...

void **display**(ostream& out);

private:

/// \brief The actual hashing function

/// \param str The string to hash.

/// \returns A hash based on the mod of the sum of the first 3 characters and

/// the table size.

int **hash**(const string& str);

unsigned tableSize; ///>! Size of this table. Also used for hash

LinkedList<string>\* table; ///>! Storage for the elements.

};

#endif

/// \file HashTable.cpp

/// \author Johnathan Lee for CSCI 1107, Lab 13

/// \brief HashTable implementation.

#include "HashTable.h"

HashTable::HashTable(unsigned size) {

tableSize = size;

table = new LinkedList<string>[tableSize];

}

HashTable::~**HashTable**() {

for (int i = 0; i < tableSize; i++)

table[i].~LinkedList(); // Make sure LinkedList destructor is called.

delete[] table;

table = NULL; // Because Visual studio is somehow still in use.

}

void HashTable::insert(string str) {

int strHash = hash(str);

table[strHash].insert(str);

}

void HashTable::display(ostream& out) {

for (int i = 0; i < tableSize; i++) {

cout << "Index " << i << ": " << table[i] << endl;

}

}

int HashTable::hash(const string& str) {

// Max out at 3 characters, otherwise use what's in the string.

int stopping = (str.length() > 3 ? 3 : str.length());

int total = 0;

for (int i = 0; i < 3; i++)

total += str[i];

return total % tableSize;

}

/// \file Project-5.2.cpp

/// \author Johnathan Lee for CSCI 1107, Lab 13

/// \brief Test driver for HashTable class.

#include <iostream>

#include <sstream>

#include "HashTable.h"

using namespace std;

int **main**() {

string testText =

// With full formatting!

"DEAR MARLIN\n"

"THE AARDVARKS AND THE CAMELS WERE MISTAKENLY SHIPPED TO "

"THE AZORES\n"

"SORRY ABOUT THAT\n"

"SINCERELY JIM\n"

"PS ANTS AND BATS AND COWS AND CATS ARE ANIMALS\n"

"COWS ARE BIG BUT ANTS ARE SMALL AND BATS AND CATS ARE IN BETWEEN";

HashTable table(11);

string buff;

unsigned numStrings = 0;

cout << "How many strings do you want to enter? ";

cin >> numStrings;

cout << "Alright, enter " << numStrings << " strings: ";

for (int i = 0; i < numStrings; i++) {

cin >> buff;

table.insert(buff);

}

table.display(cout);

return 0;

}