CISP 440

Austin Smothers

Homework 8

/\*

Homemade relational database.

Implements Select, Project, Join and Union operations.

Austin Smothers

Orginal Mar 2013

Updated Mar 28 2017

This program builds on code provided by Professor Ross

\*/

#include <iostream>

#include <iomanip>

#include <string.h>

#include <fstream>

#pragma warning( disable : 4267)

#pragma warning( disable : 4996)

using namespace std;

// \*\*\*\* THE TABLES \*\*\*\*

// Use 800kB global tables cuz RAM is cheap and labor is not!

// Seriously, a more efficient memory implementation would

// use pointers and dynamic memory (using new and/or malloc)

// but that often requires tricky pointer arithmetic

char T1[100][100][80];

char T2[100][100][80];

/\*

Prints out a table

\*\*\*\*\*\*\* HINTS!! \*\*\*\*\*\*\*

Does anybody read this stuff???!

See this function right here?

It has a basic table looping structure that you may find helpful

in building the other functions you need. Just squeeze in some

logic here and there with maybe strcmp and strcpy.

\*/

void printTable(char T[100][100][80])

{

int i = 0, j = 0;

while (T[i][j][0]) { // look for null char at T[i][j][0]

while (T[i][j][0]){

cout << left << setw(20) << T[i][j];// string at T[i][j]

j++;

}

cout << endl;

i++; j = 0;

}

cout << endl;

}

/\*

Erases a table

\*/

void eraseTable(char T[100][100][80])

{

// erase the destination array

for (int i = 0; i < 100; i++) // rows

for (int j = 0; j < 100; j++) // cols

for (int k = 0; k < 80; k++) // chars

T[i][j][k] = 0;

}

void eraseArray(int A[], int size)

{

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

A[i] = 0;

}

/\*

Reads a file into a table

\*/

void filltable(char filename[80], char Table[100][100][80])

{

// open source file

ifstream fin(filename);

if (!fin) { cerr << "Input file could not be opened\n"; exit(1); }

char line[80];

char buf[80];

// table rows and cols

int row = 0; int col = 0;

// Copy file into table

fin.getline(line, 80);

while (line[0]){

col = 0; // reset col

//cout << line << endl << endl;

// parse this line

int i = 0; int j = 0;

int len = strlen(line);

for (i = 0; i <= len; i++){

if ((line[i] == ',') || (line[i] == 0)){ // delimiters

strncpy(buf, line + j, i - j);

buf[i - j] = 0; // null terminator

j = i + 1; // scoot up j

// copy buffer to table array

strcpy(Table[row][col], buf);

col++;

//cout << "Hey: " << buf << endl;

}

}

// get another line

fin.getline(line, 80);

row++;

}

}

/\*

Performs a select operation on a table.

Receives a table.

Returns a table consisting of only the rows which have

the specified 'value' in the specified 'col'

\*/

void select(char Tout[100][100][80], char Tin[100][100][80], int col, char \* value)

{

// erase the destination array

eraseTable(Tout) ;

int i = 0, j = 0;

int k = 0, l = 0; //k and l will control the copied array's elements

while (Tin[i][j][0]) { // look for null char at T[i][j][0]

while (Tin[i][j][0]){

if (strcmp(Tin[i][col], value) == 0)

{ //copy all of the contents of a line

strcpy(Tout[k][l], Tin[i][j]);

/\*for (int k = 0; k < 81; k++)

Tout[i][j][k] = Tin[i][j][k];\*/

l++;

}

j++;

}

if (strcmp(Tin[i][col], value) == 0)

k++, l = 0;

i++; j = 0;

}

}

/\*

Performs a project operation on a table.

Receives a table.

Returns a table consisting of the specified cols.

The 'cols' parameter is a set of boolean flags where

true means we want this row in the resulting table

\*/

void project(char Tout[100][100][80], char Tin[100][100][80], int cols[100])

{

// erase the destination array

eraseTable(Tout);

int i = 0, j = 0;

int k = 0, l = 0; //k and l will control the copied array's elements

while (Tin[i][j][0]) { // look for null char at T[i][j][0]

while (Tin[i][j][0]){

if (cols[j] == 1){

strcpy(Tout[k][l], Tin[i][j]);

l++;

}

j++;

}

if (cols[j] == 1)

k++;

i++; j = 0;

}

}

/\*

Performs a join operation on a table.

Receives 2 tables and joins them.

Returns only the rows where the value in table1's T1col

matches the value in table2's T2col

\*/

void join(char Tout[100][100][80], char T1[100][100][80], char T2[100][100][80], int T1col, int T2col)

{

// erase the destination array

eraseTable(Tout);

int i = 0, j = 0;

T1col = 1, T2col = 1;

int k = 0, l = 0; //k and l will control the copied array's elements

while (T2[i][j][0]) { // look for null char at T[i][j][0]

while (T2[i][j][0]){

if (strcmp(T2[i][T2col], T1[1][T1col]) == 0){

while (j < 3){

strcpy(Tout[i][j], T2[i][j]);

j++;

}

strcpy(Tout[i][3], T1[1][0]);

}

else if (strcmp(T2[i][T2col], T1[0][T1col]) == 0){

while (j < 3){

strcpy(Tout[i][j], T2[i][j]);

j++;

}

strcpy(Tout[i][3], T1[0][0]);

}

i++, j = 0;

}

}

}

/\*

Makes a 3rd table containing all the stuff in 2 other tables.

The 2 input tables must have the same schema.

\*/

void Union(char Tout[100][100][80], char T2[100][100][80], char T3[100][100][80])

{

// erase the destination array

eraseTable(Tout);

int i = 0, j = 0;

while (T2[i][j][0]) { // look for null char at T[i][j][0]

while (T2[i][j][0]){

for (j = 0; j < 3; j++){

strcpy(T2[i + 5][j], T3[i][j]);

}

i++, j = 0;

}

for (i = 0; i < 7; i++){

for (j = 0; j < 3; j++) {

strcpy(Tout[i][j], T2[i][j]);

}

j = 0;

}

}

}

/\*

Reads in some tables and does operations on them

\*/

void main(void)

{

filltable("Professors.txt", T1);

filltable("Students.txt", T2);

cout << "Original Professors table:\n";

printTable(T1);

cout << "Original Students table:\n";

printTable(T2);

// select

// locals are created in the stack (except statics)

// so use a static to prevent stack overflow

static char Temp1[100][100][80];

cout << "\nSELECT TEST: Temp1 = Students[Address = 555 Riley]\n";

select(Temp1, T2, 1, "555 Riley");

printTable(Temp1);

// project

cout << "\nPROJECT TEST: Temp2 = Students[Name, Phone]\n";

static char Temp2[100][100][80];

// dynamic arrays are also easy on the stack

// cuz they point to the heap

int \* cols = new int[100]; eraseArray(cols, 100);

cols[0] = cols[2] = 1;

project(Temp2, T2, cols);

printTable(Temp2);

// join

cout << "\nJOIN TEST: Temp3 = Students[address = address]Professors\n";

static char Temp3[100][100][80];

join(Temp3, T1, T2, 1, 1);

printTable(Temp3);

// union

cout << "\nUNION TEST: Temp4 = Students UNION MoreStudents\n";

static char T3[100][100][80];

filltable("MoreStudents.txt", T3);

static char Temp4[100][100][80];

Union(Temp4, T2, T3);

printTable(Temp4);

}

