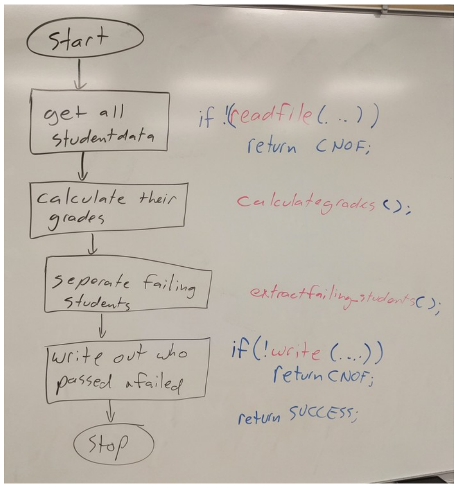

```
if !(readfile(...))
    return CNOF;

calculategrades();

extractfailing_students();

if(!write(...))
    return CNOF;

return SUCCESS;
```

Utility projects follow this pattern
open and read file
process contents
write output

Vector Student Grades
Create projects
add a source folder (not a regular folder) ensures compilation

includes
    constants.h
    utilities.h
utilities
    utilities.cpp
have 4_vector_list.cpp in regular src folder

have TestData.txt already there

do includes

then pop in source (utilities.cpp and 4_vector_list.cpp)
make sure your declaration agrees with definition (ie if & in h also in cpp)

```
//main see //TODO
	//practice some sorting
	sortArray(NAME);

	//why might this one be especially useful?
	sortArray(FINAL_GRADE);


//utilities.h
//sorts studentdata based on SORT_TYPE
enum SORT_TYPE{NAME,FINAL_GRADE};
bool sortArray(SORT_TYPE st);


//utilities.cpp
bool comp_name(const studentData &s1, const studentData &s2){
	return s1.name<s2.name;
}
//any advantage to sorting high to low verses low to high?
bool comp_classgrade(const studentData &s1, const studentData &s2){
	return s1.classgrade>s2.classgrade;
}
bool sortArray(SORT_TYPE st) {
	switch (st){
	case NAME:
		std::sort(allstudentData.begin(), allstudentData.end(), comp_name );
		break;
	case FINAL_GRADE:
		std::sort(allstudentData.begin(), allstudentData.end(), comp_classgrade);
		break;
	default:
		//raise an error here
		return false;
	}

	return true;
}
```

```cpp
#include "../includes/constants.h"
#include "../includes/utilities.h"
using namespace std;

int process_Data(const std::string &infile, const std::string
                    &Passfile, const std::string &Failfile) {
     ifstream myInFile;

     //open file
     myInFile.open(infile.c_str());
     if (!myInFile.is_open())
          return COULD_NOT_OPEN_FILE;

     //read file into vector, calculate final grade
     if (!readFileIntoVector(myInFile))
          return COULD_NOT_READ_FILE_INTO_VECTOR;

     //close file
     if (myInFile.is_open())
          myInFile.close();

     //calculate final grade
     calculateFinalGrade();

     //practice some sorting
     sortArray(NAME);

     //why might this one be especially useful?
     sortArray(FINAL_GRADE);

     //strip out failing students and add to fail.txt
     extractFailingStudents();

     //save failing students to other file
     if (!writeDataToFile(PASS,Passfile))
          return COULD_NOT_WRITE_VECTOR_TO_FILE;

     if (!writeDataToFile(FAIL,Failfile))
          return COULD_NOT_WRITE_VECTOR_TO_FILE;

     return SUCCESS;
}
```

```
/*
 * utilities.h
 *
 *  Created on: Sep 17, 2013
 *      Author: lynn
 */

#ifndef UTILITIES_H_
#define UTILITIES_H_

#include <string>
#include "../includes/constants.h"

const double UNINITIALIZED = -1.0;
struct studentData{
     std::string name;
     double midterm,final;
     double classgrade;
     void clear()
{name.clear();midterm=final=classgrade=UNINITIALIZED;}
};

enum ranking {PASS,FAIL};
enum SORT_TYPE{NAME,FINAL_GRADE};

bool readFileIntoVector(std::ifstream &file, char
char_to_search_for=CHAR_TO_SEARCH_FOR);
void calculateFinalGrade();
void extractFailingStudents(double failgrade = FAILGRADE);

bool writeDataToFile(ranking r, const std::string &filename);

//sorts studentdata based on SORT_TYPE
bool sortArray(SORT_TYPE st);

//if myString does not contain a string rep of number returns o
//if int not large enough has undefined behaviour, very fragile
int stringToInt(const char *myString);
std::string DoubleToString ( double Number );

#endif /* UTILITIES_H_ */
```

```
int main() {
      string infile = ALL_FILE;
    string Passfile = PASS_FILE;
    string Failfile = FAIL_FILE;

    return process_Data(infile,Passfile, Failfile);
}

//ALTERNATIVE- pass the files as arguments
//const int FAIL_WRONG_NUMBER_ARGS = -5;
//const int EXPECTED_NUMBER_ARGUMENTS =4;
//const string WRONG_NUMB_ARGS = "This program expects 3 arguments, infile
passfile failfile";

//int main( int argc, char *argv[] )  {
//    //argc = how many arguments passed in (including this program)
//    //char *argv[] char array of those arguments
//
//    //expect progname infile passfile failfile   //program and 3 arguments,
argc=4
//   if( argc != EXPECTED_NUMBER_ARGUMENTS ) {
//       cout<< WRONG_NUMB_ARGS <<endl;
//       return FAIL_WRONG_NUMBER_ARGS;
//   }
//   string infile = argv[1];
//   string Passfile = argv[2];
//   string Failfile = argv[3];
//
//   return process_Data(infile,Passfile, Failfile);
//}
```