Project Name: Project 1: Voting System **Team# 3**

Test Stage: Unit X__ System __ **Test Date: 4/2/2020**

Test Case ID#: stv_candidate_UT008 **Name(s) of Testers: Bryan Baker**

Test Description:
The test verifies that stv candidates can have their ballot lists accessed.

Test File: candidate_UT.cc
Method: TEST_F(STVCandidateTests, RemoveBallotList)

Automated: yes_X__ no ___

Results: Pass __X___ Fail_____

Preconditions for Test:
Create one stv candidate objects, candidate1

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Check the initial number of ballots for candidat1	candidate1	0	0	
2	Try to pull the ballot list from candidate2	candidate2	std::list<Ballot*>{}	std::list<Ballot*>{}	
3	Add ballot1	candidate2, ballot1			
4	Check the number of ballots	candiate2	1	1	
	Check the first item in the ballot list	candidate2	ballot1	ballot1	
	Add ballot2	candidate2, ballot2			
	Check the number of ballots	candidate2	1	1	Since the list was removed it should be zero
	Check the last item in the ballot list	candidate2	ballot2	ballot2	
	Add ballots 1 and ballots 2	candidate2, ballot1, ballot2			
	check the size of the ballot list removed	candidate2	2	2	

Post condition(s) for Test:

It is known that an stv candidate object can have its ballot list removed.

Project Name: Project 1: Voting System **Team# 3**

Test Stage: Unit X__ System __ **Test Date: 4/2/2020**

Test Case ID#: stv_candidate_UT006 **Name(s) of Testers: Bryan Baker**

Test Description:
The test verifies that stv candidates can set a value for the firt ballot number.

Test File: candidate_UT.cc
Method: TEST_F(STVCandidateTests, SetFirstBallotNum)

Automated: yes_X__ no ___

Results: Pass __X___ Fail_____

Preconditions for Test:
Create one stv candidate objects, candidate2

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Check the initial number of ballots for candidate2	candidate2	0	0	
2	Try to set the ballot number to a positive value.	candidate2	Expect no exception	No exception found.	
3	Try to set the ballot number to a negative value.	candidate2	Expect an exception	Exception found.	
4					

Post condition(s) for Test:

It is known that an stv candidate object can have its first ballot number set only to a positive value.

Project Name: Project 1: Voting System **Team# 3**

Test Stage: Unit X__ System __ **Test Date: 4/2/2020**

Test Case ID#: stv_candidate_UT009 **Name(s) of Testers: Bryan Baker**

Test Description:
The test verifies that stv candidates can have their ballot count set to zero. This is the primary method that sets the number of ballots for a candidate to zero when the ballot listt is removed.

Test File: candidate_UT.cc
Method: TEST_F(STVCandidateTests, SetNumBallotZero)

Automated: yes_X__ no ___

Results: Pass __X___ Fail_____

Preconditions for Test:
Create one stv candidate objects, candidate1 and two ballot objects ballot1 and ballot2.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Check the initial number of ballots for candidat2	candidate2	0	0	
2	Add ballot1	candidate2, ballot1			
3	Get the number of ballots	candidate2	1	1	
4	Remove the ballot list fro candidate2	candidate2			
	Get the number of ballots	cnadidate 2	0	0	
	Add ballot 1 and ballot2	candidate2, ballot1, ballot2			
	check the number of ballots	candidate2	2	2	
	Remove the ballot list from candidate2	cnadidae2			
	Get the number of ballots	cnadidater2	0	0	

Post condition(s) for Test:

It is known that an stv candidate object will have its ballot counter set to zero after the remove ballot method is called.

Project Name: Project 1: Voting System **Team#3**

Test Stage: Unit __ System _x_ **Test Date: 3/30/20**

Test Case ID#: STV_Election_Record_Test_ST001 **Name(s) of Testers: Hailin Archer**

Test Description:

Create an STVElectionRecord object

Indicate where are you storing the tests (what file) and the name of the method/functions being used.

Automated: yes__x_ no ___

Results: Pass ___x__ Fail_____

Preconditions for Test: VotingSystem started

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Set up initial values for candidate and ballots				
2	Instantiate an STVElectionRecord object with those initial values		Object created	Object created	
3					
4					

Post condition(s) for Test:

Program exits

Project Name: The project #, name of your system, and the team#

Test Stage: Indicate whether it is a unit test or a system test.

Test Date: The date the test was performed.

Test Case ID#: A unique ID is required. Decide on a naming convention and use numbering. Example: Ballot_Shuffle_1

Name(s) of Testers: List the names of anyone involved in running this test case.

Test Description: Describe briefly the test objective.

Automated: Indicate if the test is completely automated or being checked manually. (If you have methods running the tests and checking results, select "yes". If you are manually checking results, indicate manual by selecting the "no.")

Results: Indicate if the test passed or failed.

Step #: You will be listing the test steps in order. This number is the step number in the process.

Test Step Description: Details of the test step.

Test Data: What the test data will be for this step. Be clear on what the input data will be. If using a specific file, be clear on the name.

Expected Result: What result are you expecting from the program component or system.

Actual Result: What result were returned based on the test.

Post condition for Test: What will be true after the test has been run? Has the state of the system changed in any way?

Notes: Comments and notes for you and your team members.

Project Name: Project 1: Voting System **Team#3**

Test Stage: Unit __ System _x_ **Test Date: 4/2/20**

Test Case ID#: UserInterface_Test_ST001 **Name(s) of Testers: Hailin Archer**

Test Description:

Able to run user interface

Indicate where are you storing the tests (what file) and the name of the method/functions being used.

Automated: yes__x_ no ___

Results: Pass ___x__ Fail_____

Preconditions for Test: VotingSystem started

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Start VotingSystem program				
2	Display User Interface		User Interface displayed	User Interface displayed	
3					
4					

Post condition(s) for Test:

Program exits

Project Name: The project #, name of your system, and the team#

Test Stage: Indicate whether it is a unit test or a system test.

Test Date: The date the test was performed.

Test Case ID#: A unique ID is required. Decide on a naming convention and use numbering. Example: Ballot_Shuffle_1

Name(s) of Testers: List the names of anyone involved in running this test case.

Test Description: Describe briefly the test objective.

Automated: Indicate if the test is completely automated or being checked manually. (If you have methods running the tests and checking results, select "yes". If you are manually checking results, indicate manual by selecting the "no.")

Results: Indicate if the test passed or failed.

Step #: You will be listing the test steps in order. This number is the step number in the process.

Test Step Description: Details of the test step.

Test Data: What the test data will be for this step. Be clear on what the input data will be. If using a specific file, be clear on the name.

Expected Result: What result are you expecting from the program component or system.

Actual Result: What result were returned based on the test.

Post condition for Test: What will be true after the test has been run? Has the state of the system changed in any way?

Notes: Comments and notes for you and your team members.

Project Name: Project 1: Voting System **Team#3**

Test Stage: Unit __ System _x_ **Test Date: 4/2/20**

Test Case ID#: UserInterface_Test_ST002 **Name(s) of Testers: Hailin Archer**

Test Description:

Able to take user inputs of algorithm choice

Indicate where are you storing the tests (what file) and the name of the method/functions being used.

Automated: yes__x_ no ___

Results: Pass ___x__ Fail_____

Preconditions for Test: VotingSystem started

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Start VotingSystem program				
2	Display User Interface				
3	User select from option list		Able to make selection	Able to make selection	
4	User selection is passed into algorithm choice variable		Selection is stored	Selection is stored	

Post condition(s) for Test:

Program exits

Project Name: The project #, name of your system, and the team#

Test Stage: Indicate whether it is a unit test or a system test.

Test Date: The date the test was performed.

Test Case ID#: A unique ID is required. Decide on a naming convention and use numbering. Example: Ballot_Shuffle_1

Name(s) of Testers: List the names of anyone involved in running this test case.

Test Description: Describe briefly the test objective.

Automated: Indicate if the test is completely automated or being checked manually. (If you have methods running the tests and checking results, select "yes". If you are manually checking results, indicate manual by selecting the "no.")

Results: Indicate if the test passed or failed.

Step #: You will be listing the test steps in order. This number is the step number in the process.

Test Step Description: Details of the test step.

Test Data: What the test data will be for this step. Be clear on what the input data will be. If using a specific file, be clear on the name.

Expected Result: What result are you expecting from the program component or system.

Actual Result: What result were returned based on the test.

Post condition for Test: What will be true after the test has been run? Has the state of the system changed in any way?

Notes: Comments and notes for you and your team members.

Project Name: Project 1: Voting System **Team#3**

Test Stage: Unit __ System _x_ **Test Date: 4/2/20**

Test Case ID#: UserInterface_Test_ST003 **Name(s) of Testers: Hailin Archer**

Test Description:

Able to reject invalid input for algorithm selection

Indicate where are you storing the tests (what file) and the name of the method/functions being used.

Automated: yes__x_ no ___

Results: Pass ___x__ Fail_____

Preconditions for Test: VotingSystem started

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Start VotingSystem program				
2	Display User Interface				
3	User input option number	-1	Option is rejected and user is asked to re-enter option choice	Option is rejected and user is asked to to re-enter option choice	

Post condition(s) for Test:

Program exits

Project Name: The project #, name of your system, and the team#

Test Stage: Indicate whether it is a unit test or a system test.

Test Date: The date the test was performed.

Test Case ID#: A unique ID is required. Decide on a naming convention and use numbering. Example: Ballot_Shuffle_1

Name(s) of Testers: List the names of anyone involved in running this test case.

Test Description: Describe briefly the test objective.

Automated: Indicate if the test is completely automated or being checked manually. (If you have methods running the tests and checking results, select "yes". If you are manually checking results, indicate manual by selecting the "no.")

Results: Indicate if the test passed or failed.

Step #: You will be listing the test steps in order. This number is the step number in the process.

Test Step Description: Details of the test step.

Test Data: What the test data will be for this step. Be clear on what the input data will be. If using a specific file, be clear on the name.

Expected Result: What result are you expecting from the program component or system.

Actual Result: What result were returned based on the test.

Post condition for Test: What will be true after the test has been run? Has the state of the system changed in any way?

Notes: Comments and notes for you and your team members.

Project Name: Project 1: Voting System **Team#3**

Test Stage: Unit __ System _x_ **Test Date: 4/2/20**

Test Case ID#: UserInterface_Test_ST004 **Name(s) of Testers: Hailin Archer**

Test Description:

Able to take number of seats input

Indicate where are you storing the tests (what file) and the name of the method/functions being used.

Automated: yes__x_ no ___

Results: Pass ___x__ Fail_____

Preconditions for Test: VotingSystem started

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Start VotingSystem program				
2	Display User Interface				
3	User made algorithm selection	1	1	1	
4	System asks user to input number of seats	1	1	1	

Post condition(s) for Test:

Program exits

Project Name: The project #, name of your system, and the team#

Test Stage: Indicate whether it is a unit test or a system test.

Test Date: The date the test was performed.

Test Case ID#: A unique ID is required. Decide on a naming convention and use numbering. Example: Ballot_Shuffle_1

Name(s) of Testers: List the names of anyone involved in running this test case.

Test Description: Describe briefly the test objective.

Automated: Indicate if the test is completely automated or being checked manually. (If you have methods running the tests and checking results, select "yes". If you are manually checking results, indicate manual by selecting the "no.")

Results: Indicate if the test passed or failed.

Step #: You will be listing the test steps in order. This number is the step number in the process.

Test Step Description: Details of the test step.

Test Data: What the test data will be for this step. Be clear on what the input data will be. If using a specific file, be clear on the name.

Expected Result: What result are you expecting from the program component or system.

Actual Result: What result were returned based on the test.

Post condition for Test: What will be true after the test has been run? Has the state of the system changed in any way?

Notes: Comments and notes for you and your team members.

Project Name: Project 1: Voting System **Team#3**

Test Stage: Unit __ System _x_ **Test Date: 4/2/20**

Test Case ID#: UserInterface_Test_ST005 **Name(s) of Testers: Hailin Archer**

Test Description:

Able to reject invalid seat number input

Indicate where are you storing the tests (what file) and the name of the method/functions being used.

Automated: yes__x_ no ___

Results: Pass ___x__ Fail_____

Preconditions for Test: VotingSystem started

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Start VotingSystem program				
2	Display User Interface				
3	User made algorithm selection	1	1	1	
4	System asks user to input number of seats	0	Input is rejected. System asks user to re-enter number of seats	Input is rejected. System asks user to re-enter number of seats	

Post condition(s) for Test:

Program exits

Project Name: The project #, name of your system, and the team#

Test Stage: Indicate whether it is a unit test or a system test.

Test Date: The date the test was performed.

Test Case ID#: A unique ID is required. Decide on a naming convention and use numbering. Example: Ballot_Shuffle_1

Name(s) of Testers: List the names of anyone involved in running this test case.

Test Description: Describe briefly the test objective.

Automated: Indicate if the test is completely automated or being checked manually. (If you have methods running the tests and checking results, select "yes". If you are manually checking results, indicate manual by selecting the "no.")

Results: Indicate if the test passed or failed.

Step #: You will be listing the test steps in order. This number is the step number in the process.

Test Step Description: Details of the test step.

Test Data: What the test data will be for this step. Be clear on what the input data will be. If using a specific file, be clear on the name.

Expected Result: What result are you expecting from the program component or system.

Actual Result: What result were returned based on the test.

Post condition for Test: What will be true after the test has been run? Has the state of the system changed in any way?

Notes: Comments and notes for you and your team members.

Project Name: Project 1: Voting System **Team#3**

Test Stage: Unit _x_ System __ **Test Date: 4/1/20**

Test Case ID#: Plurality_election_record_UT003 **Name(s) of Testers: Colin Kluegel**

Test Description:
Test distributes the ballots to candidates and then sorts the list by number of ballots, test verifies that candidates are correctly sorted

Test file: plurality_election_record_UT.cc
Method: TEST_F(PluralityElectionRecordTests, SortNonElectedCandidateList)
Indicate where are you storing the tests (what file) and the name of the method/functions being used.

Automated: yes_x__ no ___

Results: Pass ___x__ Fail_____

Preconditions for Test: 5 candidate objects and 5 ballot objects are created and put in candidate lists and ballot lists respectively. A new PluralityElectionRecord object is created with these lists

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Call election_record->DistributeBallots				
2	Call election_record->SortNonelectedCandidateList				
3	Create a list of candidates and set it to what is returned from election_record->GetNonElectedCandidateList()	Candidate list			
4	Check that candidate at front of the list is the correct winner	Candidate list	Candidate with ID 1 at front of list	Candidate with id 1 at front of list Test passed	
5	Remove first candidate from list so we can check 2nd place is correct	Candidate on list	Candidate with ID 2 at front of list	Candidate with id 2 at front of list Test passed	
6					

Post condition(s) for Test:
All ballots have been distributed, nonElectedCandidateList is now sorted with the candidates with more votes at the front of the list.

Project Name: The project #, name of your system, and the team#

Test Stage: Indicate whether it is a unit test or a system test.

Test Date: The date the test was performed.

Test Case ID#: A unique ID is required. Decide on a naming convention and use numbering. Example: Ballot_Shuffle_1

Name(s) of Testers: List the names of anyone involved in running this test case.

Test Description: Describe briefly the test objective.

Automated: Indicate if the test is completely automated or being checked manually. (If you have methods running the tests and checking results, select "yes". If you are manually checking results, indicate manual by selecting the "no.")

Results: Indicate if the test passed or failed.

Step #: You will be listing the test steps in order. This number is the step number in the process.

Test Step Description: Details of the test step.

Test Data: What the test data will be for this step. Be clear on what the input data will be. If using a specific file, be clear on the name.

Expected Result: What result are you expecting from the program component or system.

Actual Result: What result were returned based on the test.

Post condition for Test: What will be true after the test has been run? Has the state of the system changed in any way?

Notes: Comments and notes for you and your team members.

Project Name: Project 1: Voting System **Team#**

Test Stage: Unit x __ System __ **Test Date: 4/1/20**

Test Case ID#: Plurality_election_record_UT004 **Name(s) of Testers: Colin Kluegel**

Test Description: Test distributes the ballots to candidates and then sorts the list by number of ballots, test verifies that candidates are correctly sorted, test is the same as the Plurality_election_record_UT003, but we insert the candidates in original list in a different order and distribute the ballots to different candidates to verify that we just didn't get lucky the for UT003

Test file: plurality_election_record_UT.cc
Method: TEST_F(PluralityElectionRecordTests, SortNonElectedCandidateList_reorder)
Indicate where are you storing the tests (what file) and the name of the method/functions being used.

Automated: yes__x_ no ___

Results: Pass __x___ Fail_____

Preconditions for Test: 5 candidate objects and 5 ballot objects are created and put in candidate lists and ballot lists respectively. A new PluralityElectionRecord object is created with these lists

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Call election_record->DistributeBallots				
2	Call election_record->SortNonelectedCandidateList				
3	Create a list of candidates and set it to what is returned from election_record->GetNonElectedCandidateList()	Candidate list			
4	Check that candidate at front of the list is the correct winner	Candidate on list	Candidate with ID 1 at front of list	Candidate with id 1 at front of list Test passed	
5	Remove first candidate from list so we can check 2nd place is correct	Candidate on lsit	Candidate with ID 2 at front of list	Candidate with id 2 at front of list Test passed	

Post condition(s) for Test:
All ballots have been distributed, nonElectedCandidateList is now sorted with the candidates with more votes at the front of the

list.

Project Name: The project #, name of your system, and the team#

Test Stage: Indicate whether it is a unit test or a system test.

Test Date: The date the test was performed.

Test Case ID#: A unique ID is required. Decide on a naming convention and use numbering. Example: Ballot_Shuffle_1

Name(s) of Testers: List the names of anyone involved in running this test case.

Test Description: Describe briefly the test objective.

Automated: Indicate if the test is completely automated or being checked manually. (If you have methods running the tests and checking results, select "yes". If you are manually checking results, indicate manual by selecting the "no.")

Results: Indicate if the test passed or failed.

Step #: You will be listing the test steps in order. This number is the step number in the process.

Test Step Description: Details of the test step.

Test Data: What the test data will be for this step. Be clear on what the input data will be. If using a specific file, be clear on the name.

Expected Result: What result are you expecting from the program component or system.

Actual Result: What result were returned based on the test.

Post condition for Test: What will be true after the test has been run? Has the state of the system changed in any way?

Notes: Comments and notes for you and your team members.

Project Name: Project 1: Voting System **Team#3**

Test Stage: Unit __ System _x_ **Test Date: 4/2/20**

Test Case ID#: Plurality_System_Test_ST001 **Name(s) of Testers: Colin Kluegel**

Test Description:a plurality election was run on ballot file plurality_ballots.csv

Indicate where are you storing the tests (what file) and the name of the method/functions being used.

Automated: yes__x_ no ___

Results: Pass ___x__ Fail_____

Preconditions for Test: VotingSystem started

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Run election with any number of seats, in this case 2 seats				
2	Use plurality_ballots.csv				
3	Run election		A has 33% of the vote, C has 33% or the vote, F has 33% of the vote, everyone else has 0 votes	A has 33% of the vote, C has 33% or the vote, F has 33% of the vote, everyone else has 0 votes	
4					

Post condition(s) for Test:

Program exits

Project Name: The project #, name of your system, and the team#

Test Stage: Indicate whether it is a unit test or a system test.

Test Date: The date the test was performed.

Test Case ID#: A unique ID is required. Decide on a naming convention and use numbering. Example: Ballot_Shuffle_1

Name(s) of Testers: List the names of anyone involved in running this test case.

Test Description: Describe briefly the test objective.

Automated: Indicate if the test is completely automated or being checked manually. (If you have methods running the tests and checking results, select "yes". If you are manually checking results, indicate manual by selecting the "no.")

Results: Indicate if the test passed or failed.

Step #: You will be listing the test steps in order. This number is the step number in the process.

Test Step Description: Details of the test step.

Test Data: What the test data will be for this step. Be clear on what the input data will be. If using a specific file, be clear on the name.

Expected Result: What result are you expecting from the program component or system.

Actual Result: What result were returned based on the test.

Post condition for Test: What will be true after the test has been run? Has the state of the system changed in any way?

Notes: Comments and notes for you and your team members.

Project Name: Project 1: Voting System **Team# 3**

Test Stage: Unit X__ System __ **Test Date: 4/2/2020**

Test Case ID#: stv_candidate_UT006 **Name(s) of Testers: Bryan Baker**

Test Description:
The test verifies that stv candidate objects can have ballots assigned to them.

Test File: candidate_UT.cc
Method: TEST_F(STVCandidateTests, AddBallot)

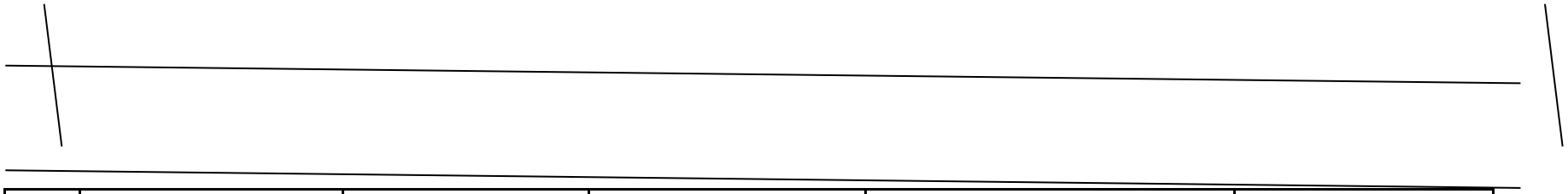
Automated: yes_X__ no ___

Results: Pass __X___ Fail_____

Preconditions for Test:
Create two stv candidate objects candidate1 and candidate2 and two ballot objects ballot1 and ballot2.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Check initial ballot counts candidate1.	candidate1	0	0	
2	Check initial ballot counts candidate2.	candidate2	0	0	
	Check initial first ballot num for candidate1	candidate1	0	0	
	Check initial firstt ballot num for candidate2	candidate2	0	0	
3	Add a ballot to candidate1	candidate1, ballot1			
4	Check ballot counts for candidate1	candidate1	1	1	
5	Check ballot counts for candidate2	candidate2	0	0	checking that candidate2 was not affected.
	Check first ballot num for candidate1	cnadidate1	1	1	
	Check first ballot num for candidate2	candidate2	0	0	
6	Add a ballot to candidate2	candidate2, ballot2			
7	Check ballot counts for candidate1	candidate1	1	1	make sure that candidate1 did not change.
8	Check ballot counts for	candidate2	1	1	

	candidate2				
	Check first ballot number for candidate1	candidate1	1	1	
	Check first ballot numbedr for candidate2	cndidate2	2	2	
9	Add ballot1 to candidate2	candidate2, ballot1			Thinking that this test would be nice to ensure that we can not assign the same ballot to two different candidates.
10	check ballot counts for candidate1	candidate1	1	1	Checking that candidate1 was not affected.
11	Check ballot counts for candidate2	candidate2	1	2	Failed this test.
	Check first ballot number for candidate1	candidate1	1	1	



	Check first ballot numbedr for candidate2	cndidate2	2	2	

Post condition(s) for Test:

Two candidate objects can have ballot objects added to them.

Project Name: Project 1: Voting System **Team# 3**

Test Stage: Unit X__ System __ **Test Date: 4/2/2020**

Test Case ID#: stv_candidate_UT001 **Name(s) of Testers: Bryan Baker**

Test Description:
The test verifies that stv candidates can be created correctly.

Test File: candidate_UT.cc
Method: TEST_F(STVCandidateTests, Constructor)

Automated: yes_X__ no ___

Results: Pass __X___ Fail_____

Preconditions for Test:
Create two stv candidate objects.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create an stv candidate object with a negative candidate id number.	candidate1	Expect an exception	Exception found.	
2	Check the normal creation of stv candidate objects.	candidate1, candidate2	Expect no exception	No exception found.	
3					
4					

Post condition(s) for Test:

Two stv candidate objects can be created and are ready for further processing.

Project Name: Project 1: Voting System **Team# 3**

Test Stage: Unit X__ System __ **Test Date: 4/2/2020**

Test Case ID#: stv_candidate_UT007 **Name(s) of Testers: Bryan Baker**

Test Description:
The test verifies that stv candidates can get and set the first ballot number.

Test File: candidate_UT.cc
Method: TEST_F(STVCandidateTests, GetFirstBallotNum)

Automated: yes_X__ no ___

Results: Pass __X___ Fail_____

Preconditions for Test:
Create one stv candidate objects, candidate2

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Check the initial ballot number.	candidate2	0	0	
2	check the initial ballot count.	candidate2	0	0	
3	Set the first ballot number to 1	candidate2			
4	check the first ballot number.	candidate2	1	1	
5	Set the first ballot number to 200	candidate2			
6	check the first ballot number.	candidate2	200	200	

Post condition(s) for Test:

It is known that an stv candidate object can have its first ballot number set.

Project Name: Project 1: Voting System **Team# 3**

Test Stage: Unit X__ System __ **Test Date: 4/2/2020**

Test Case ID#: stv_candidate_UT002 **Name(s) of Testers: Bryan Baker**

Test Description:
The test verifies that stv candidates have the correct id number.

Test File: candidate_UT.cc
Method: TEST_F(STVCandidateTests, GetID)

Automated: yes_X__ no ___

Results: Pass __X___ Fail_____

Preconditions for Test:
Create two stv candidate objects, candidate1 and candidate2.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Check normal function of GetID for candidate1.	candidate1	1	1	
2	Check normal function of GetID for candidate2.	candidate2	2	2	
3	Create a new stv candidate in candidate2 with an id of 43	candidate2	43	43	
4					

Post condition(s) for Test:

Two stv candidate objects are known to have the correct id numbers assigned to them.

Project Name: Project 1: Voting System **Team# 3**

Test Stage: Unit X__ System __ **Test Date: 4/2/2020**

Test Case ID#: stv_candidate_UT003 **Name(s) of Testers: Bryan Baker**

Test Description:
The test verifies that stv candidates have the correct candidate name.

Test File: candidate_UT.cc
Method: TEST_F(STVCandidateTests, GetName)

Automated: yes_X__ no ___

Results: Pass __X___ Fail_____

Preconditions for Test:
Create two stv candidate objects, candidate1 and candidate2.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Check the normal function of GetName for candidate1	candidate1	Allison	Allison	
2	Check normal function of GetNamefor candidate2.	candidate2	Mark	Mark	
3					
4					

Post condition(s) for Test:

Two stv candidate objects are known to have the correct candidate names assigned to them.

Project Name: Project 1: Voting System **Team# 3**

Test Stage: Unit X__ System __ **Test Date: 4/2/2020**

Test Case ID#: stv_candidate_UT004 **Name(s) of Testers: Bryan Baker**

Test Description:
The test verifies that stv candidates the correct number of ballots initialized.

Test File: candidate_UT.cc
Method: TEST_F(STVCandidateTests, GetNumBallots)

Automated: yes_X__ no ___

Results: Pass __X___ Fail_____

Preconditions for Test:
Create two stv candidate objects, candidate1 and candidate2.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Check the normal function of GetNumBallots for candidate1	candidate1	0	0	
2	Check normal function of GetNumBallots for candidate2.	candidate2	0	0	
3					
4					

Post condition(s) for Test:

Two stv candidate objects are known to have the correct initial number of ballots.

Project Name: Project 1: Voting System **Team# 3**

Test Stage: Unit X__ System __ **Test Date: 4/2/2020**

Test Case ID#: stv_candidate_UT005 **Name(s) of Testers: Bryan Baker**

Test Description:
The test verifies that stv candidates have the number of ballots increased when ballots are added. The IncrementNumBallots method is the method called by add ballot to set the new number of ballots for the candidate.

Test File: candidate_UT.cc
Method: TEST_F(STVCandidateTests, IncrementNumBallots)

Automated: yes_X__ no ___

Results: Pass __X___ Fail_____

Preconditions for Test:
Create one stv candidate objects, candidate1 and two ballot objects ballot1 and ballot2.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Check the initial value of the number of ballots for candidate1	candidate1	0	0	
2	Add ballot 1 to candidate1	candidate1, ballot1			
3	Check the number of ballots for candidate1	candidate1	1	1	
4	Add ballot 2 to candidate1	candidate1, ballot2			
	Check the number of ballots for candidate2	candidate2	2	2	

Post condition(s) for Test:

It is known that an stv candidate object will have its number of ballots incremented when ballots are added to it.

Project Name: Project 1: Voting System **Team# 3**

Test Stage: Unit X__ System __ **Test Date: 4/2/2020**

Test Case ID#: stv_candidate_UT008 **Name(s) of Testers: Bryan Baker**

Test Description:
The test verifies that stv candidates can have their ballot lists accessed.

Test File: candidate_UT.cc
Method: TEST_F(STVCandidateTests, RemoveBallotList)

Automated: yes_X__ no ___

Results: Pass __X___ Fail_____

Preconditions for Test:
Create one stv candidate objects, candidate1

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Check the initial number of ballots for candidat1	candidate1	0	0	
2	Try to pull the ballot list from candidate2	candidate2	std::list<Ballot*>{}	std::list<Ballot*>{}	
3	Add ballot1	candidate2, ballot1			
4	Check the number of ballots	candiate2	1	1	
	Check the first item in the ballot list	candidate2	ballot1	ballot1	
	Add ballot2	candidate2, ballot2			
	Check the number of ballots	candidate2	1	1	Since the list was removed it should be zero
	Check the last item in the ballot list	candidate2	ballot2	ballot2	
	Add ballots 1 and ballots 2	candidate2, ballot1, ballot2			
	check the size of the ballot list removed	candidate2	2	2	

Post condition(s) for Test:

It is known that an stv candidate object can have its ballot list removed.

Project Name: Project 1: Voting System **Team# 3**

Test Stage: Unit X__ System __ **Test Date: 4/2/2020**

Test Case ID#: candidate_UT006 **Name(s) of Testers: Bryan Baker**

Test Description:
The test verifies that the number of ballots increment correctly. This method is implemented in add ballot and is called to change the number of ballots accordingly.

Test File: candidate_UT.cc
Method: TEST_F(CandidateTests, IncrementNumBallots)

Automated: yes_X__ no ___

Results: Pass __X___ Fail_____

Preconditions for Test:
Create one candidate object candidate1 and two ballot objects ballot1 and ballot2.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Verify that the initial ballot count is 0.	candidate1	0	0	
2	Add ballot1 to candidate1	candidate1, ballot1			
3	Check the number of ballots for candidate1.	candidate1	1	1	
4	Add ballot2 to candidate1	candidate1, ballot2			
	Check the number of ballots for candidate1	candidate1	2	2	

Post condition(s) for Test:

A candidate object will correctly increment the number of ballots when adding ballots.

Project Name: Project 1: Voting System **Team# 3**

Test Stage: Unit X__ System __ **Test Date: 4/2/2020**

Test Case ID#: logger_UT001 **Name(s) of Testers: Bryan Baker**

Test Description:
The test verifies that the logger object is created properly

Test File: logger_UT.cc
Method: TEST_F(LoggerTests, Constructor)

Automated: yes_X__ no ___

Results: Pass __X___ Fail_____

Preconditions for Test:
A logger object is created, deconstructed, and created again.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	check that a logger object can be created	audit_log	expect no exception	no exceptoin found	
2	Deconstruct the logger object and create an new one	audit_log	expect an exception	no exception found	The file name for the audit file is hard coded, so it should throw an error when you try to open the file and it is present because we do not want to append to a previous election.
3					
4					

Post condition(s) for Test:

A logger object is set up and can be used for futher processing. (this logger is actually not set up properly because it should be accessable by all objects in a system, but it is not.

Project Name: Project 1: Voting System **Team# 3**

Test Stage: Unit X__ System __ **Test Date: 4/2/2020**

Test Case ID#: logger_UT002 **Name(s) of Testers: Bryan Baker**

Test Description:
The test verifies that the logger object will return the correct value of the audit file.

Test File: logger_UT.cc
Method: TEST_F(LoggerTests, GetLogFile)

Automated: yes_X__ no ___

Results: Pass __X___ Fail_____

Preconditions for Test:
A logger object is created

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	check the log file name	audit_log	audit_file.txt	audit_file.txt	
2					
3					
4					

Post condition(s) for Test:

A logger object can return the name of the file it is using for logging. (this logger is actually not set up properly because it should be accessable by all objects in a system, but it is not.

Project Name: Project 1: Voting System **Team# 3**

Test Stage: Unit X__ System __ **Test Date: 4/2/2020**

Test Case ID#: logger_UT003 **Name(s) of Testers: Bryan Baker**

Test Description:
The test verifies that the logger object can log to the file.

Test File: logger_UT.cc
Method: TEST_F(LoggerTests, LogToFile)

Automated: yes___ no _X__

Results: Pass __X___ Fail_____

Preconditions for Test:
A logger object is created

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Log a string to the log file.				
2	Check that the string is added to the log file.				This was a manual test. I opened the log file and verified that the string was present.
3					
4					

Post condition(s) for Test:

A logger object can log string inforamtion to its log file. (this logger is actually not set up properly because it should be accessable by all objects in a system, but it is not.

Project Name: Project 1: Voting System **Team#3**

Test Stage: Unit x__ System __ **Test Date: 4/1/20**

Test Case ID#: Plurality_election_record_UT005 **Name(s) of Testers: Colin Kluegel**

Test Description:Checks that the break ties method works

Test file: plurality_election_record_UT.cc
Method: TEST_F(PluralityElectionRecordTests, BreakTies)
Indicate where are you storing the tests (what file) and the name of the method/functions being used.

Automated: yes__x_ no ___

Results: Pass ___x__ Fail_____

Preconditions for Test: 5 candidate objects and 5 ballot objects are created and put in candidate lists and ballot lists respectively. A new PluralityElectionRecord object is created with these lists

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Set Boolean Test_candidate to be the returned value of static function PluralityElectionRecord::BreakTies()	Bool test_Candidate			
2	Check that test_candidate is either true or false		Test candidate was true or false	Test candidate was true or false	
3					
4					

Post condition(s) for Test:

Same as in the pre conditions

Project Name: The project #, name of your system, and the team#

Test Stage: Indicate whether it is a unit test or a system test.

Test Date: The date the test was performed.

Test Case ID#: A unique ID is required. Decide on a naming convention and use numbering. Example: Ballot_Shuffle_1

Name(s) of Testers: List the names of anyone involved in running this test case.

Test Description: Describe briefly the test objective.

Automated: Indicate if the test is completely automated or being checked manually. (If you have methods running the tests and checking results, select "yes". If you are manually checking results, indicate manual by selecting the "no.")

Results: Indicate if the test passed or failed.

Step #: You will be listing the test steps in order. This number is the step number in the process.

Test Step Description: Details of the test step.

Test Data: What the test data will be for this step. Be clear on what the input data will be. If using a specific file, be clear on the name.

Expected Result: What result are you expecting from the program component or system.

Actual Result: What result were returned based on the test.

Post condition for Test: What will be true after the test has been run? Has the state of the system changed in any way?

Notes: Comments and notes for you and your team members.

Project Name: Project 1: Voting System **Team#3**

Test Stage: Unit _x_ System __ **Test Date: 4/1/20**

Test Case ID#: Plurality_election_record_UT001 **Name(s) of Testers: Colin Kluegel**

Test Description:
The test verifies that the constructor correctly created the nonDistributedBallotList and nonElectedCandidateList

Test file: plurality_election_record_UT.cc
Method: TEST_F(PluralityElectionRecordTests, Constructor)

Automated: yes__x_ no ___

Indicate where are you storing the tests (what file) and the name of the method/functions being used.

Results: Pass ___x__ Fail_____

Preconditions for Test: 5 candidate objects and 5 ballot objects are created and put in candidate lists and ballot lists respectively. A new PluralityElectionRecord object is created with these lists

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create candidate list and set it to the output of election_record->GetNonElectedCandidateList()	Candidate list			
2	Create ballot list and set it to the output of the election_record->GetNonDistributedBallotList()	Ballot list			
3	Check that the retrieved candidate list is equal to the one originally put into the constructor		Lists should be equal	Test passed – Lists are equal	List are the same because no candidates have been moved to the winners or losers list yet
4	Check that the retrieved ballot list is equal to the one originally put into the constructor		Lists should be equal	Test passed – Lists are equal	List are the same because no ballots have been distributed yet

Post condition(s) for Test:

No data was manipulated, post conditions are the same as the preconditions.

Project Name: The project #, name of your system, and the team#

Test Stage: Indicate whether it is a unit test or a system test.

Test Date: The date the test was performed.

Test Case ID#: A unique ID is required. Decide on a naming convention and use numbering. Example: Ballot_Shuffle_1

Name(s) of Testers: List the names of anyone involved in running this test case.

Test Description: Describe briefly the test objective.

Automated: Indicate if the test is completely automated or being checked manually. (If you have methods running the tests and checking results, select "yes". If you are manually checking results, indicate manual by selecting the "no.")

Results: Indicate if the test passed or failed.

Step #: You will be listing the test steps in order. This number is the step number in the process.

Test Step Description: Details of the test step.

Test Data: What the test data will be for this step. Be clear on what the input data will be. If using a specific file, be clear on the name.

Expected Result: What result are you expecting from the program component or system.

Actual Result: What result were returned based on the test.

Post condition for Test: What will be true after the test has been run? Has the state of the system changed in any way?

Notes: Comments and notes for you and your team members.

Project Name: Project 1: Voting System **Team#3**

Test Stage: Unit x __ System __ **Test Date: 4/1/20**

Test Case ID#: Plurality_election_record_UT002 **Name(s) of Testers: Colin Kluegel**

Test Description:Tests that ballots are correctly distributed to all the candidates as expect

Test File: plurality_election_record_UT.cc
Method: TEST_F(PluralityElectionRecordTests, DistributeBallots)
Indicate where are you storing the tests (what file) and the name of the method/functions being used.

Automated: yes_x__ no ___

Results: Pass __x___ Fail_____

Preconditions for Test: 5 candidate objects and 5 ballot objects are created and put in candidate lists and ballot lists respectively. A new PluralityElectionRecord object is created with these lists

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Call election record Distribute ballots method				
2	Create a new ballot_list and set it the return of election_record->GetonDistributedBallotList()				
3	Check that this ballot list is empty	Ballots list	Ballots list is empty	Ballots list is empty – test passed	List is empty because all the ballots have been distributed
4	Create a new candidate list, and set it to the return of election_record->GetNonElectedCandidateList()				
5	Check that size of candidate list size	Candidate list	Size is 5	Size is 5, test passed	List is still full of original candidates because they have not been moved to the winners or losers list
6			Candidate id 1 has 3 ballots Candidate id 2 has 2 ballots All other candidates have 0 ballots	Candidate id 1 has 3 ballots Candidate id 2 has 2 ballots All other candidates have 0 ballots Test passed	

	Iterate through candidate list checking number of ballots each candidate has	Candidate list			
	Iterate through candidate list checking number of ballots each candidate has				

Post condition(s) for Test: NonDistributed ballot list is empty, candidates have been assigned their ballots and are still on the nonElectedCandidateList

Project Name: The project #, name of your system, and the team#

Test Stage: Indicate whether it is a unit test or a system test.

Test Date: The date the test was performed.

Test Case ID#: A unique ID is required. Decide on a naming convention and use numbering. Example: Ballot_Shuffle_1

Name(s) of Testers: List the names of anyone involved in running this test case.

Test Description: Describe briefly the test objective.

Automated: Indicate if the test is completely automated or being checked manually. (If you have methods running the tests and checking results, select "yes". If you are manually checking results, indicate manual by selecting the "no.")

Results: Indicate if the test passed or failed.

Step #: You will be listing the test steps in order. This number is the step number in the process.

Test Step Description: Details of the test step.

Test Data: What the test data will be for this step. Be clear on what the input data will be. If using a specific file, be clear on the name.

Expected Result: What result are you expecting from the program component or system.

Actual Result: What result were returned based on the test.

Post condition for Test: What will be true after the test has been run? Has the state of the system changed in any way?

Notes: Comments and notes for you and your team members.

Project Name: Project 1: Voting System **Team#3**

Test Stage: Unit _x_ System __ **Test Date: 4/1/20**

Test Case ID#: Plurality_election_record_UT009 **Name(s) of Testers: Colin Kluegel**

Test Description:
Checks that when candidates are moved to the losers list all the non-elected candidates are successfully moved to the losers list

Test file: plurality_election_record_UT.cc
Method: TEST_F(PluralityElectionRecordTests, GetLosersList)
Indicate where are you storing the tests (what file) and the name of the method/functions being used.

Automated: yes_x__ no ___

Results: Pass __x___ Fail_____

Preconditions for Test: Preconditions for Test: 5 candidate objects and 5 ballot objects are created and put in candidate lists and ballot lists respectively. A new PluralityElectionRecord object is created with these lists

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Check that losers list is empty		Losers list is empty	Losers list is empty, test passes	
2	Move 2 candidates to winners list				
3	Move the rest of the candidates to the loser list				
4	Create a loser_list of candidates from what is returned by election_record->MoveRemainingCandidatesToLosersList()	Loser_list			
5	Check size of losers list		Size is 3	Size is 3, test passes	
6	Check that the correct candidate is at the front of the losers lsit		Candidate3 is at the front of the losers list	Candidate3 is at the front of the losers list, test passed	
7	Pop first candidate off of loser_list,				
8	Create a candidate losing_candidate that is equal to the first element in loser_list	Losing_candidate			
9	Check that losing_candidate is the correct candidate		Candidate2 is the losing_candidate	Candidate2 is the losing_candidate, test passed	

10	Pop candidate off of loser_list				
11	Set losing_candidate to front of loser list	Losing_candidate			
12	Check that losing_candidate is the correct candidate		Candidate1 is the losing_candidate	Candidate1 is the losing_candidate, test passed	

Post condition(s) for Test:

2 candidates are on the winners list and 3 are on the losers list

Project Name: The project #, name of your system, and the team#

Test Stage: Indicate whether it is a unit test or a system test.

Test Date: The date the test was performed.

Test Case ID#: A unique ID is required. Decide on a naming convention and use numbering. Example: Ballot_Shuffle_1

Name(s) of Testers: List the names of anyone involved in running this test case.

Test Description: Describe briefly the test objective.

Automated: Indicate if the test is completely automated or being checked manually. (If you have methods running the tests and checking results, select “yes”. If you are manually checking results, indicate manual by selecting the “no.”)

Results: Indicate if the test passed or failed.

Step #: You will be listing the test steps in order. This number is the step number in the process.

Test Step Description: Details of the test step.

Test Data: What the test data will be for this step. Be clear on what the input data will be. If using a specific file, be clear on the name.

Expected Result: What result are you expecting from the program component or system.

Actual Result: What result were returned based on the test.

Post condition for Test: What will be true after the test has been run? Has the state of the system changed in any way?

Notes: Comments and notes for you and your team members.

Project Name: Project 1: Voting System **Team#3**

Test Stage: Unit _x_ System __ **Test Date: 4/1/20**

Test Case ID#: Plurality_election_record_UT008 **Name(s) of Testers: Colin Kluegel**

Test Description:
Repeatedly move candidates to winners list, check that the winners list continues to have the correct elements

Test file: plurality_election_record_UT.cc
Method: TEST_F(PluralityElectionRecordTests, GetWinnersList)
Indicate where are you storing the tests (what file) and the name of the method/functions being used.

Automated: yes_x__ no ___

Results: Pass ___x__ Fail_____

Preconditions for Test: Preconditions for Test: 5 candidate objects and 5 ballot objects are created and put in candidate lists and ballot lists respectively. A new PluralityElectionRecord object is created with these lists

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Check that the winners list is empty		Winners list is empty	Winners list is empty, test passed	
2	Move 2 candidates to winners list				
3	Set a winners_list to what is returned by election_record->GetWinnersList	Winners_list			
4	Check size of winners list		Size is 2	Size is 2, test passed	
5	Check that the correct candidate is at the front of the winners list		Candidate5 is at front of winners list	Candidate5 is a front of winners list, test passed	
6	Add 2 more candidates to winners list				
7	Set a winners_list to what is returned by election_record->GetWinnersList	Winners_list			
8	Check size of winners list		Size is 4	Size is 4, test passed	
9	Check which candidate is at front of winners list		Candidate 5 is at front of winners lsit	Candidate5 is at front of winners list, test passed	

10	Pop_candidate off of winners list, set new candidate to the front of the winners_list	Winning_candidate			
11	Verify winning_candidate is the correct candidate		Winning_candidate is candidate4	Winning_candidate is candidate4, test passed	
12	Pop_candidate off of winners list, set new candidate to the front of the winners_list	Winning candidate			
13	Verify winning_candidate is the correct candidate		Winning_candidate is candidate3	Winning_Candidate is candidate3, test passed	
14	Pop_candidate off of winners list, set new candidate to the front of the winners_list	Winning candidate			
15	Verify winning_candidate is the correct candidate		Winning_candidate is candidate2	Winning_Candidate is candidate2, test passed	

Post condition(s) for Test:

4 of the candidates have been moved to the winners list, 1 remains on the non-elected candidate list

Project Name: The project #, name of your system, and the team#

Test Stage: Indicate whether it is a unit test or a system test.

Test Date: The date the test was performed.

Test Case ID#: A unique ID is required. Decide on a naming convention and use numbering. Example: Ballot_Shuffle_1

Name(s) of Testers: List the names of anyone involved in running this test case.

Test Description: Describe briefly the test objective.

Automated: Indicate if the test is completely automated or being checked manually. (If you have methods running the tests and checking results, select "yes". If you are manually checking results, indicate manual by selecting the "no.")

Results: Indicate if the test passed or failed.

Step #: You will be listing the test steps in order. This number is the step number in the process.

Test Step Description: Details of the test step.

Test Data: What the test data will be for this step. Be clear on what the input data will be. If using a specific file, be clear on the name.

Expected Result: What result are you expecting from the program component or system.

Actual Result: What result were returned based on the test.

Post condition for Test: What will be true after the test has been run? Has the state of the system changed in any way?

Notes: Comments and notes for you and your team members.

Project Name: Project 1: Voting System **Team#3**

Test Stage: Unit x __ System __ **Test Date: 4/1/20**

Test Case ID#: Plurality_election_record_UT006 **Name(s) of Testers: Colin Kluegel**

Test Description:
Tests that method MoveFirstNCandidatesFromNonElectedListToWinnersList can move candidates from nonelected list to the winners list

Test file: plurality_election_record_UT.cc
Method: TEST_F(PluralityElectionRecordTests, MoveFirstNCandidatesFromNonElectedListToWinnersList)
Indicate where are you storing the tests (what file) and the name of the method/functions being used.

Automated: yes_x__ no ___

Results: Pass ___x__ Fail_____

Preconditions for Test: Preconditions for Test: 5 candidate objects and 5 ballot objects are created and put in candidate lists and ballot lists respectively. A new PluralityElectionRecord object is created with these lists

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Attempt to move 20 candidates to winners list		Exception thrown	Exception thrown	We expect an exception because there are only 5 candidates
2	Call election_record->MoveFirstNCandidatesFromNonElectedList-ToWinners(3) to move 3 of the candidates to the winners lsit				
3	Create a candidates_list and set it to what is returned by election_record->GetWinnersList(3)	Candidates list			
4	Create a candidate object of the first candidate in the winnersList	Candidate object			
5	Check that we move the correct candidate to the front of the winners list		candidate5 should be at the front of the winners list	Candidate5 is at the front of the winners list, test passed	
6	Pop first candidate off of winners list				
7	Check new candidate at front of winners list	Candidate object	Candidate4 should be at front of winners list	Candidate4 is a front of winners list, Test passed	
8	Pop first candidate off of winners list				

9	Check new candidate at front of winners lsit	Candidate object	Candidate3 should be at front of winners list	Candidate3 is a front of winners list, test passed	

Post condition(s) for Test:

3 candidates have been moved from the Non-elected list to the winners list

Project Name: The project #, name of your system, and the team#

Test Stage: Indicate whether it is a unit test or a system test.

Test Date: The date the test was performed.

Test Case ID#: A unique ID is required. Decide on a naming convention and use numbering. Example: Ballot_Shuffle_1

Name(s) of Testers: List the names of anyone involved in running this test case.

Test Description: Describe briefly the test objective.

Automated: Indicate if the test is completely automated or being checked manually. (If you have methods running the tests and checking results, select "yes". If you are manually checking results, indicate manual by selecting the "no.")

Results: Indicate if the test passed or failed.

Step #: You will be listing the test steps in order. This number is the step number in the process.

Test Step Description: Details of the test step.

Test Data: What the test data will be for this step. Be clear on what the input data will be. If using a specific file, be clear on the name.

Expected Result: What result are you expecting from the program component or system.

Actual Result: What result were returned based on the test.

Post condition for Test: What will be true after the test has been run? Has the state of the system changed in any way?

Notes: Comments and notes for you and your team members.

Project Name: Project 1: Voting System **Team#3**

Test Stage: Unit x__ System __ **Test Date: 4/1/20**

Test Case ID#: Plurality_election_record_UT007 **Name(s) of Testers: Colin Kluegel**

Test Description:After several candidates are move to the winners list we need to check that we can move the remaining candidates to the losers list

Test file: plurality_election_record_UT.cc
Method: TEST_F(PluralityElectionRecordTests, MoveRemainingCandidatesToLosersList)
Indicate where are you storing the tests (what file) and the name of the method/functions being used.

Automated: yes__x_ no ___

Results: Pass ___x__ Fail_____

Preconditions for Test: Preconditions for Test: 5 candidate objects and 5 ballot objects are created and put in candidate lists and ballot lists respectively. A new PluralityElectionRecord object is created with these lists

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Move first 2 candidates on non-elected list to winners list				
2	Call MoveRemainingCandidatesToLosersList to move the rest of the candidates to the losers list				
3	Create losers_lists by setting to what is returned by election_record->GetLosersList()	Losers list			
4	Check size of losers list		Size should be 3	Size is 3, test passed	
5	Create a candidate object and set it to the first candidate in the losers list	Loser candidate			
6	Check that loser candidate is the correct candidate		Loser candidate should be candidate 3	Loser candidate is candidate 3, test passed	
7	Pop first candidate off of losers_list, set next candidate on list to loser candidate				

8	Check that loser candidate is the correct candidate		Loser candidate should be candidate 2	Loser candidate is candidate 3, test passed	
9	Pop first candidate off of losers_list, set next candidate on list to loser candidate				
10	Check that loser candidate is the correct candidate		Loser candidate should be candidate 1	Loser is candidate 1, test passed	

Post condition(s) for Test:

First 2 candidates have been moved from the non-elected list to the winners list, the rest of the candidates are on the losers list

Project Name: The project #, name of your system, and the team#

Test Stage: Indicate whether it is a unit test or a system test.

Test Date: The date the test was performed.

Test Case ID#: A unique ID is required. Decide on a naming convention and use numbering. Example: Ballot_Shuffle_1

Name(s) of Testers: List the names of anyone involved in running this test case.

Test Description: Describe briefly the test objective.

Automated: Indicate if the test is completely automated or being checked manually. (If you have methods running the tests and checking results, select “yes”. If you are manually checking results, indicate manual by selecting the “no.”)

Results: Indicate if the test passed or failed.

Step #: You will be listing the test steps in order. This number is the step number in the process.

Test Step Description: Details of the test step.

Test Data: What the test data will be for this step. Be clear on what the input data will be. If using a specific file, be clear on the name.

Expected Result: What result are you expecting from the program component or system.

Actual Result: What result were returned based on the test.

Post condition for Test: What will be true after the test has been run? Has the state of the system changed in any way?

Notes: Comments and notes for you and your team members.

Project Name: Project 1: Voting System **Team# 3**

Test Stage: Unit X__ System __ **Test Date: 4/2/2020**

Test Case ID#: ballot_UT001 **Name(s) of Testers: Bryan Baker**

Test Description:
The test verifies that ballots are created properly.

Test File: ballot_UT.cc
Method: TEST_F(BallotTests, Constructor)

Automated: yes_X__ no ___

Results: Pass __X___ Fail_____

Preconditions for Test:
Three sets of integer lists were created: 1 for candidate1, 1 for candidate2, and one with a duplicate value. Two variables were defined to use the ballot type.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Check negative ballot ids do not work.	ballot1, candidateList1. Set ballot id to be -1.	Expect an exception.	exception found	
2	Check that candidatelists with duplicate candidates will not work.	ballot1, candidateDup.	Expect an exception.	exception found	
3	Check that normal assignment works.	ballot1, ballot2, candidatelist1, candidatelist2.	Expect no exception.	No exception found for either ballot.	
4					

Post condition(s) for Test:

Two ballot objects will be created and ready to use elsewhere.

Project Name: Project 1: Voting System **Team# 3**

Test Stage: Unit X__ System __ **Test Date: 4/2/2020**

Test Case ID#: ballot_file_tests_UT001 **Name(s) of Testers: Bryan Baker**

Test Description:
The test verifies that a ballot file processor object can be created.

Test File: ballot_file_processor_UT.cc
Method: TEST_F(BallotFileTests, Constructor)

Automated: yes_X__ no ___

Results: Pass __X___ Fail_____

Preconditions for Test:
Create two ballot file processor objects

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Check the normal creation of ballot file processor objects.	pbfp, sbfp	Expect no exception	No exception found.	
2					
3					
4					

Post condition(s) for Test:

Two ballot file processor objects can be created and ready for further use.

Project Name: Project 1: Voting System **Team# 3**

Test Stage: Unit X__ System __ **Test Date: 4/2/2020**

Test Case ID#: ballot_file_tests_UT002 **Name(s) of Testers: Bryan Baker**

Test Description:
The test verifies that ballot files for the plurality election type can be processed. The information gets stored in the votinginfo object.

Test File: ballot_file_processor_UT.cc
Method: TEST_F(BallotFileTests, ProcessPluralityBallots)

Automated: yes_X__ no ___

Results: Pass __X___ Fail_____

Preconditions for Test:
Create a ballot file processor objects and a csv file for the plurality ballots.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Process plurality ballots from csv file	pbfp, csv file			
2	check that the correct number of candidates were added	pbfp, votinginfo	6	6	
3	check that the correct number of ballots were added	pbfp, votinginfo	3	3	
4	Check that the first candidate in the candidate list matches what is expected from the csv file.	pbfp, votinginfo	ID = 0 name = A	ID = 0 name=A	
	Check that the first ballot added to the ballot list matchs what is expected from the csv file.	pbfp, votinginfo	ballot id = 1 candidate_id list = 0	ballot id = 1 candidate_id list = 0	

Post condition(s) for Test:

The ballot file processor can correctly process plurality election ballot files.

Project Name: Project 1: Voting System **Team# 3**

Test Stage: Unit X__ System __ **Test Date: 4/2/2020**

Test Case ID#: ballot_file_tests_UT003 **Name(s) of Testers: Bryan Baker**

Test Description:
The test verifies that ballot files for the STV election type can be processed. The information gets stored in the votinginfo object.

Test File: ballot_file_processor_UT.cc
Method: TEST_F(BallotFileTests, ProcessSTVBallots)

Automated: yes_X__ no ___

Results: Pass __X___ Fail_____

Preconditions for Test:
Create a ballot file processor objects and a csv file for the stv ballots.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Process stv ballots from csv file	pbfp, csv file			
2	check that the correct number of candidates were added	pbfp, votinginfo	6	6	
3	check that the correct number of ballots were added	pbfp, votinginfo	4	4	
4	Check that the first candidate in the candidate list matches what is expected from the csv file.	pbfp, votinginfo	ID = 0 name = A	ID = 0 name=A	
	Check that the first ballot added to the ballot list matchs what is expected from the csv file.	pbfp, votinginfo	ballot id = 1 candidate_id list = 0	ballot id = 1 candidate_id list = 0	

Post condition(s) for Test:

The ballot file processor can correctly process stv election ballot files.

Project Name: Project 1: Voting System **Team# 3**

Test Stage: Unit X__ System __ **Test Date: 4/2/2020**

Test Case ID#: ballot_UT002 **Name(s) of Testers: Bryan Baker**

Test Description:
The test verifies that ballots return the correct ballot id value.

Test File: ballot_UT.cc
Method: TEST_F(BallotTests, GetID)

Automated: yes_X__ no ___

Results: Pass __X___ Fail_____

Preconditions for Test:
Create two ballot objects. Ballot1 with an id of 1 and ballot2 with an id of 2.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Check for correct return of ballot id values for ballot1.	ballot1	1	1	
2	Check for correct return of ballot id values for ballot2.	ballot2	2	2	
3					
4					

Post condition(s) for Test:

Two ballot objects will be known to have the correct ballot values.

Project Name: Project 1: Voting System **Team# 3**

Test Stage: Unit X__ System __ **Test Date: 4/2/2020**

Test Case ID#: ballot_UT003 **Name(s) of Testers: Bryan Baker**

Test Description:
The test verifies that ballots return the correct candidate id lists.

Test File: ballot_UT.cc
Method: TEST_F(BallotTests, GetRankedCandidateList)

Automated: yes_X__ no ___

Results: Pass __X___ Fail_____

Preconditions for Test:
Create two ballot objects. Ballot1 with a candidate list of 5 candidates and ballot2 with a candidate list of 10 candidates.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Check for correct return of candidate ID list values for ballot1.	ballot1, candidateList1	candidateList1	candidateList1	
2	Check for correct return of candidate ID list values for ballot1.	ballot2, candidateList2	candidateList2	candidateList2	
3					
4					

Post condition(s) for Test:

Two ballot objects will be known to have the correct candidate lists.

Project Name: Project 1: Voting System **Team# 3**

Test Stage: Unit X__ System __ **Test Date: 4/2/2020**

Test Case ID#: candidate_UT005 **Name(s) of Testers: Bryan Baker**

Test Description:
The test verifies that candidate objects can have ballots assigned to them.

Test File: candidate_UT.cc
Method: TEST_F(CandidateTests, AddBallot)

Automated: yes_X__ no ___

Results: Pass __X___ Fail_____

Preconditions for Test:
Create two candidate objects candidate1 and candidate2 and two ballot objects ballot1 and ballot2.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Check initial ballot counts candidate1.	candidate1	0	0	
2	Check initial ballot counts candidate2.	candidate2	0	0	
3	Add a ballot to candidate1	candidate1, ballot1			
4	Check ballot counts for candidate1	candidate1	1	1	
5	Check ballot counts for candidate2	candidate2	0	0	checking that candidate2 was not affected.
6	Add a ballot to candidate2	candidate2, ballot2			
7	Check ballot counts for candidate1	candidate1	1	1	make sure that candidate1 did not change.
8	Check ballot counts for candidate2	candidate2	1	1	
9	Add ballot1 to candidate2	candidate2, ballot1			Thinking that this test would be nice to ensure that we can not assign the same ballot to two different candidates.
10	check ballot counts for candidate1	candidate1	1	1	Checking that candidate1 was not affected.
11	Check ballot counts for	candidate2	1	2	Failed this test.

	candidate2				

Post condition(s) for Test:

Two candidate objects can have ballot objects added to them.

Project Name: Project 1: Voting System **Team# 3**

Test Stage: Unit X__ System __ **Test Date: 4/2/2020**

Test Case ID#: candidate_UT001 **Name(s) of Testers: Bryan Baker**

Test Description:
The test verifies that candidate objects can be created properly.

Test File: candidate_UT.cc
Method: TEST_F(CandidateTests, Constructor)

Automated: yes_X__ no ___

Results: Pass __X___ Fail_____

Preconditions for Test:
Create two variables to hold candidate objects, candidate1 and candidate2.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Check that you can not create a candidate with a negative candidate id.	candidate1 with a negative candidate id.	Expect an exception.	Exception found.	
2	Create normal usage of candidate object for candidate1 and candidate2.	candidate1 and candidate2	Expect no exceptions.	No exceptions found.	
3					
4					

Post condition(s) for Test:

Two candidate objects will have been created successfully and will be ready for further processing.

Project Name: Project 1: Voting System **Team# 3**

Test Stage: Unit X__ System __ **Test Date: 4/2/2020**

Test Case ID#: candidate_UT002 **Name(s) of Testers: Bryan Baker**

Test Description:
The test verifies that candidate objects have the correct id number.

Test File: candidate_UT.cc
Method: TEST_F(CandidateTests, GetID)

Automated: yes_X__ no ___

Results: Pass __X___ Fail_____

Preconditions for Test:
Create two candidate objects candidate1 and candidate2.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Check normal usage of getID for candidate1	candidate1	1	1	
2	Check normal usage of getID for candidate2	candidate2	2	2	
3	Assign candidate2 a new candidate object with an id of 43.	candidate2	43	43	
4					

Post condition(s) for Test:

Two candidate objects will be known to have the correct candidate id values.

Project Name: Project 1: Voting System **Team# 3**

Test Stage: Unit X__ System __ **Test Date: 4/2/2020**

Test Case ID#: candidate_UT003 **Name(s) of Testers: Bryan Baker**

Test Description:
The test verifies that candidate objects have the correct candidate name.

Test File: candidate_UT.cc
Method: TEST_F(CandidateTests, GetName)

Automated: yes_X__ no ___

Results: Pass __X___ Fail_____

Preconditions for Test:
Create two candidate objects candidate1 and candidate2.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Check normal usage of getName for candidate1.	candidate1	Allison	Allison	
2	Check normal usage of getName for candidate2.	candidate2	Mark	Mark	
3					
4					

Post condition(s) for Test:

Two candidate objects will be known to have the correct candidate names.

Project Name: Project 1: Voting System **Team# 3**

Test Stage: Unit X__ System __ **Test Date: 4/2/2020**

Test Case ID#: candidate_UT004 **Name(s) of Testers: Bryan Baker**

Test Description:
The test verifies that candidate objects have the correct number of ballots initialized.

Test File: candidate_UT.cc
Method: TEST_F(CandidateTests, GetNumBallots)

Automated: yes_X__ no ___

Results: Pass __X___ Fail_____

Preconditions for Test:
Create two candidate objects candidate1 and candidate2.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Check normal usage of GetNumBallots for candidate1.	candidate1	0	0	
2	Check normal usage of GetNumBallots for candidate2.	candidate2	0	0	
3					
4					

Post condition(s) for Test:

Two candidate objects will be known to have the correct value of number of ballots initialized.

PBI:	Fix Project Code 1
Task Description:	Test help display
Testing Number:	System_tests_S01
Tester:	Bryan Baker
Inputs:	2\n0\n1\ny\n1\n../testing/plurality_ballots.csv\nq
Outputs:	The help screen was displayed.
Passed or Failed:	Passed
Date:	5/1/2020

PBI:	Fix Project Code 1
Task Description:	Execute a plurality election
Testing Number:	System_tests_S02
Tester:	Bryan Baker
Inputs:	0\n1\ny\n1\n../testing/plurality_ballots.csv\nq
Outputs:	----------Election Result---------- * Election Type: Plurality * Number of Ballots: 3 * #Seats: 1 * #Candidates: 6 * Winners are: 1: A (1 ballots; %33.33 Votes) * Losers are: 2: C (1 ballots; %33.33 Votes) 3: F (1 ballots; %33.33 Votes) 4: B (0 ballots; %0.00 Votes) 5: D (0 ballots; %0.00 Votes) 6: E (0 ballots; %0.00 Votes) Location of audit report: \src\AuditFile_2020.04.01.161415.txt ----------End of Result Display----------
Passed or Failed:	Passed
Date:	5/1/2020

PBI:	Fix Project Code 1
Task Description:	Execute an stv election
Testing Number:	System_tests_S03
Tester:	Bryan Baker
Inputs:	1\n1\ny\n1\n../testing/stv_ballots.csv\nq

Outputs:	----------Election Result---------- * Election Type: STV * # Ballots: 4 * # Invalid ballots: 0 * #Seats: 1 * #Candidates: 6 * Winners are: 1: B * Losers are: 1: F 2: E 3: D 4: C 5: A Location of audit report: \src\AuditFile_2020.04.01.161415.txt Location of invalidated ballots report: \src\InvalidBallotFile_2020.04.01.161415.txt ----------End of Result Display----------
Passed or Failed:	passed
Date:	5/1/2020

PBI:	Fix Project Code 1
Task Description:	Try invalid entries
Testing Number:	System_tests_S04
Tester:	Bryan Baker
Inputs:	3\n-1\na\n+\n0\n1\ny\n1\n../testing/plurality_ballots.csv\nq
Outputs:	The system did not hang or go into an infinte loop.
Passed or Failed:	Passed
Date:	5/1/2020

PBI:	Fix Project Code 1
Task Description:	Try invalid number of seats
Testing Number:	System_tests_S05
Tester:	Bryan Baker
Inputs:	0\n-1\na\n1000000\n1\ny\n1\n../testing/plurality_ballots.csv\nq
Outputs:	The system did not hang or go into an infinite loop.
Passed or Failed:	Passed
Date:	5/1/2020

PBI:	Fix Project Code 1
Task Description:	Try invalid answer for is this correct
Testing Number:	System_tests_S06

Tester:	Bryan Baker
Inputs:	0\n1\n2\na\n+\ny\n1\n../testing/plurality_ballots.csv\nq
Outputs:	The system did not hang or go into an infinite loop.
Passed or Failed:	Passed
Date:	5/1/2020

PBI:	Fix Project Code 1
Task Description:	Try no for is this correct
Testing Number:	System_tests_S07
Tester:	Bryan Baker
Inputs:	0\n1\nn\n1\ny\n1\n../testing/plurality_ballots.csv\nq
Outputs:	The system did not hang or go into an infinite loop.
Passed or Failed:	Passed
Date:	5/1/2020

PBI:	Fix Project Code 1
Task Description:	Try invalid ballot file name
Testing Number:	System_tests_S08
Tester:	Bryan Baker
Inputs:	0\n1\ny\n1\n../testing/this_doesnt_exists.csv\nq
Outputs:	The system did not hang or go into an infinite loop. Output: Cannot open ballot file: ../testing/this_doesnt_exists.csv There are no valid ballots. Abort.
Passed or Failed:	Passed
Date:	5/1/2020

PBI:	Fix Project Code 1
Task Description:	Run plurality with seats greater than candidate count
Testing Number:	System_tests_S09
Tester:	Bryan Baker
Inputs:	0\n10\ny\n1\n../testing/plurality_ballots.csv\nq

Outputs:	----------Election Result---------- * Election Type: Plurality * Number of Ballots: 3 * #Seats: 10 * #Candidates: 6 * Winners are: 1: A (1 ballots; %33.33 Votes) 2: C (1 ballots; %33.33 Votes) 3: F (1 ballots; %33.33 Votes) 4: B (0 ballots; %0.00 Votes) 5: D (0 ballots; %0.00 Votes) 6: E (0 ballots; %0.00 Votes) * Losers are: Location of audit report: \src\AuditFile_2020.04.01.161415.txt ----------End of Result Display----------
Passed or Failed:	Passed
Date:	5/1/2020

PBI:	Fix Project Code 1
Task Description:	Run stv election with seats greater than candidate count
Testing Number:	System_tests_S10
Tester:	Bryan Baker
Inputs:	1\n10\ny\n1\n../testing/stv_ballots.csv\nq
Outputs:	----------Election Result---------- * Election Type: STV * # Ballots: 4 * # Invalid ballots: 0 * #Seats: 6 * #Candidates: 6 * Winners are: 1: B 2: C 3: A 4: E 5: D 6: F * Losers are: Location of audit report: \src\AuditFile_2020.04.01.161415.txt Location of invalidated ballots report: \src\InvalidBallotFile_2020.04.01.161415.txt ----------End of Result Display----------
Passed or Failed:	Passed
Date:	5/1/2020

PBI:	Fix Project Code 1

Task Description:	stv election with invalid ballots
Testing Number:	System_tests_S11
Tester:	Bryan Baker
Inputs:	1\n2\ny\n1\n../testing/invalid_ballots.csv\nq
Outputs:	9 Ballots are read in file ../testing/invalid_ballots.csv ----------Election Result---------- * Election Type: STV * # Ballots: 4 * # Invalid ballots: 5 * #Seats: 2 * #Candidates: 6 * Winners are: 1: A 2: C * Losers are: 1: F 2: E 3: D 4: B Location of audit report: \src\AuditFile_2020.04.01.161415.txt Location of invalidated ballots report: \src\InvalidBallotFile_2020.04.01.161415.txt ----------End of Result Display----------
Passed or Failed:	Passed
Date:	5/1/2020

PBI:	Fix Project Code 1
Task Description:	plurality election, many ballots, basic names, 1 seat
Testing Number:	System_tests_S12
Tester:	Bryan Baker
Inputs:	0\n1\ny\n1\n../testing/plurality_10000ballots_10candidates.csv\nq

Outputs:	10000 Ballots are read in file ../testing/plurality_10000ballots_10candidates.csv ----------Election Result---------- * Election Type: Plurality * Number of Ballots: 10000 * #Seats: 1 * #Candidates: 10 * Winners are: 1: G (1068 ballots; %10.68 Votes) * Losers are: 2: J (1056 ballots; %10.56 Votes) 3: E (1004 ballots; %10.04 Votes) 4: A (994 ballots; %9.94 Votes) 5: I (989 ballots; %9.89 Votes) 6: H (989 ballots; %9.89 Votes) 7: F (988 ballots; %9.88 Votes) 8: C (985 ballots; %9.85 Votes) 9: D (980 ballots; %9.80 Votes) 10: B (947 ballots; %9.47 Votes) Location of audit report: \src\AuditFile_2020.04.01.161415.txt ----------End of Result Display----------
Passed or Failed:	Passed
Date:	5/1/2020

PBI:	Fix Project Code 1
Task Description:	plurality election, many ballots, named candidates, 1 seat
Testing Number:	System_tests_S13
Tester:	Bryan Baker
Inputs:	0\n1\ny\n1\n../testing/pluralityLCandidateNames_10000ballots_10candidates.csv\nq

Outputs:	10000 Ballots are read in file ../testing/pluralityLCandidateNames_10000ballots_10candidates.csv ----------Election Result---------- * Election Type: Plurality * Number of Ballots: 10000 * #Seats: 1 * #Candidates: 10 * Winners are: 1: Sophia (1056 ballots; %10.56 Votes) * Losers are: 2: William (1031 ballots; %10.31 Votes) 3: Liam (1022 ballots; %10.22 Votes) 4: Emma (1020 ballots; %10.20 Votes) 5: Noah (995 ballots; %9.95 Votes) 6: Isabella (988 ballots; %9.88 Votes) 7: James (984 ballots; %9.84 Votes) 8: Ava (982 ballots; %9.82 Votes) 9: Olivia (966 ballots; %9.66 Votes) 10: Oliver (956 ballots; %9.56 Votes) Location of audit report: \src\AuditFile_2020.04.01.161418.txt ----------End of Result Display----------
Passed or Failed:	Passed
Date:	5/1/2020

PBI:	Fix Project Code 1
Task Description:	stv election, many ballots, basic names, no invalid ballots, 3 seats
Testing Number:	System_tests_S14
Tester:	Bryan Baker
Inputs:	1\n3\ny\n1\n../testing/stv_10000ballots_10candidates_0pctBadBallots.csv\nq

Outputs:	10000 Ballots are read in file ../testing/stv_10000ballots_10candidates_0pctBadBallots.csv ----------Election Result---------- * Election Type: STV * # Ballots: 10000 * # Invalid ballots: 0 * #Seats: 3 * #Candidates: 10 * Winners are: 1: G 2: F 3: I * Losers are: 1: A 2: C 3: H 4: E 5: B 6: J 7: D Location of audit report: \src\AuditFile_2020.04.01.161421.txt Location of invalidated ballots report: \src\InvalidBallotFile_2020.04.01.161421.txt ----------End of Result Display----------
Passed or Failed:	Passed
Date:	5/1/2020

PBI:	Fix Project Code 1
Task Description:	stv election, many ballots, basic names, with invalid ballots, 3 seats
Testing Number:	System_tests_S15
Tester:	Bryan Baker
Inputs:	1\n3\ny\n1\n../testing/stv_10000ballots_10candidates_20pctBadBallots.csv\nq

Outputs:	10000 Ballots are read in file ../testing/stv_10000ballots_10candidates_20pctBadBallots.csv ----------Election Result---------- * Election Type: STV * # Ballots: 8271 * # Invalid ballots: 1729 * #Seats: 3 * #Candidates: 10 * Winners are: 1: B 2: E 3: J * Losers are: 1: I 2: G 3: C 4: A 5: F 6: H 7: D Location of audit report: \src\AuditFile_2020.04.01.161425.txt Location of invalidated ballots report: \src\InvalidBallotFile_2020.04.01.161425.txt ----------End of Result Display----------
Passed or Failed:	Passed
Date:	5/1/2020

PBI:	Fix Project Code 1
Task Description:	stv election, many ballots, named candidates, no invalid ballots, 3 seats
Testing Number:	System_tests_S16
Tester:	Bryan Baker
Inputs:	1\n3\ny\n1\n../testing/stvLCanNames_10000ballots_10candidates_0pctBadBallots.csv\nq

Outputs:	10000 Ballots are read in file ../testing/stvLCanNames_10000ballots_10candidates_0pctBadBallots.csv ----------Election Result---------- * Election Type: STV * # Ballots: 10000 * # Invalid ballots: 0 * #Seats: 3 * #Candidates: 10 * Winners are: 1: Ava 2: James 3: Oliver * Losers are: 1: Liam 2: Emma 3: Isabella 4: Sophia 5: William 6: Olivia 7: Noah Location of audit report: \src\AuditFile_2020.04.01.161428.txt Location of invalidated ballots report: \src\InvalidBallotFile_2020.04.01.161428.txt ----------End of Result Display----------
Passed or Failed:	Passed
Date:	5/1/2020
PBI:	Fix Project Code 1
Task Description:	Use many ballots, named candidates, with invalid ballots and nominal number of seats
Testing Number:	System_tests_S17
Tester:	Bryan Baker
Inputs:	1\n3\ny\n1\n../testing/stvLCanNames_10000ballots_10candidates_20pctBadBallots.csv\nq

Outputs:	10000 Ballots are read in file ../testing/stvLCanNames_10000ballots_10candidates_20pctBadBallots.csv ----------Election Result---------- * Election Type: STV * # Ballots: 8270 * # Invalid ballots: 1730 * #Seats: 3 * #Candidates: 10 * Winners are: 1: Isabella 2: William 3: Noah * Losers are: 1: Liam 2: Sophia 3: Olivia 4: Oliver 5: James 6: Emma 7: Ava Location of audit report: \src\AuditFile_2020.04.01.161432.txt Location of invalidated ballots report: \src\InvalidBallotFile_2020.04.01.161432.txt ----------End of Result Display----------
Passed or Failed:	Passed
Date:	5/1/2020

PBI:	Fix Project Code 1
Task Description:	plurality multi-file election
Testing Number:	System_tests_S18
Tester:	Bryan Baker
Inputs:	0\n1\ny\n1\n../testing/plurality_110ballots_5candidates.csv\n../testing/plurality_500ballots_5candidates.csv\nq
Outputs:	110 Ballots are read in file ../testing/plurality_110ballots_5candidates.csv 500 Ballots are read in file ../testing/plurality_500ballots_5candidates.csv ----------Election Result---------- * Election Type: Plurality * Number of Ballots: 610 * #Seats: 1 * #Candidates: 5 * Winners are: 1: E (137 ballots; %22.46 Votes) * Losers are: 2: B (136 ballots; %22.30 Votes) 3: C (122 ballots; %20.00 Votes) 4: D (109 ballots; %17.87 Votes) 5: A (106 ballots; %17.38 Votes) Location of audit report: \src\AuditFile_2020.04.01.161435.txt ----------End of Result Display----------

Passed or Failed:	Passed
Date:	5/1/2020

PBI:	Fix Project Code 1
Task Description:	stv multi-file election
Testing Number:	System_tests_S19
Tester:	Bryan Baker
Inputs:	1\n3\ny\n1\n../testing/stv_100ballots_5candidates_0pctBadBallots.csv\n../testing/stv_500ballots_5candidates_10pctBadBallots.csv\nq
Outputs:	100 Ballots are read in file ../testing/stv_100ballots_5candidates_0pctBadBallots.csv 500 Ballots are read in file ../testing/stv_500ballots_5candidates_10pctBadBallots.csv ----------Election Result---------- * Election Type: STV * # Ballots: 565 * # Invalid ballots: 35 * #Seats: 3 * #Candidates: 5 * Winners are: 1: A 2: D 3: C * Losers are: 1: E 2: B Location of audit report: \src\AuditFile_2020.04.01.161435.txt Location of invalidated ballots report: \src\InvalidBallotFile_2020.04.01.161435.txt ----------End of Result Display----------
Passed or Failed:	Passed
Date:	5/1/2020

PBI:	Fix Project Code 1
Task Description:	plurality multi-file election with differing number of candidates
Testing Number:	System_tests_S20
Tester:	Bryan Baker
Inputs:	0\n1\ny\n1\n../testing/plurality_110ballots_5candidates.csv\n../testing/plurality_120ballots_8candidates.csv\nq

Outputs:	110 Ballots are read in file ../testing/plurality_110ballots_5candidates.csv Invalid candidates detected. Skip file ../testing/plurality_120ballots_8candidates.csv ----------Election Result---------- * Election Type: Plurality * Number of Ballots: 110 * #Seats: 1 * #Candidates: 5 * Winners are: 1: D (23 ballots; %20.91 Votes) * Losers are: 2: B (23 ballots; %20.91 Votes) 3: E (22 ballots; %20.00 Votes) 4: C (22 ballots; %20.00 Votes) 5: A (20 ballots; %18.18 Votes) Location of audit report: \src\AuditFile_2020.04.01.161435.txt ----------End of Result Display----------
Passed or Failed:	Passed
Date:	5/1/2020

PBI:	Fix Project Code 1
Task Description:	stv multi-file election with differing number of candidates
Testing Number:	System_tests_S21
Tester:	Bryan Baker
Inputs:	1\n3\ny\n1\n../testing/stv_100ballots_5candidates_0pctBadBallots.csv\n../testing/stv_120ballots_8candidates_20pctBadBallots.csv\nq
Outputs:	100 Ballots are read in file ../testing/stv_100ballots_5candidates_0pctBadBallots.csv Invalid candidates detected. Skip file ../testing/stv_120ballots_8candidates_20pctBadBallots.csv ----------Election Result---------- * Election Type: STV * # Ballots: 100 * # Invalid ballots: 0 * #Seats: 3 * #Candidates: 5 * Winners are: 1: B 2: A 3: D * Losers are: 1: E 2: C Location of audit report: \src\AuditFile_2020.04.01.161435.txt Location of invalidated ballots report: \src\InvalidBallotFile_2020.04.01.161435.txt ----------End of Result Display----------
Passed or Failed:	Passed
Date:	5/1/2020

PBI:	Fix Project Code 1
Task Description:	Test ballot shuffle
Testing Number:	System_tests_S22
Tester:	Bryan Baker
Inputs:	echo "Execute STV election in VotingSystem..." >> system_test_report.txt echo "Use stv_1000ballots_10candidates_10pctBadBallots.csv with nominal number of seats and turnoff ballot shuffle" >> system_test_report.txt printf "1\n3\ny\n1\n../testing/stv_1000ballots_10candidates_10pctBadBallots.csv\nq" | ../src/./VotingSystem -t -m >> system_test_report.txt echo " " echo "The above election output should match the following output:" >> system_test_report.txt echo " " printf "1\n3\ny\n1\n../testing/stv_1000ballots_10candidates_10pctBadBallots.csv\nq" | ../src/./VotingSystem -t -m >> system_test_report.txt echo " " echo "The following election output should be different than above:" >> system_test_report.txt printf "1\n3\ny\n1\n../testing/stv_1000ballots_10candidates_10pctBadBallots.csv\nq" | ../src/./VotingSystem -m >> system_test_report.txt echo "Test Completed." >> system_test_report.txt echo " " >> system_test_report.txt
Outputs:	candidates are different
Passed or Failed:	Passed
Date:	5/1/2020

PBI:	GUI for the directory search
Task Description:	Test plurality election, single ballot file
Testing Number:	System_tests_S23
Tester:	Colin Kluegel
Inputs:	./VotingSystem Enter 0 for Plurality Election Enter 3 for number of seats Select plurality_20ballots_5candidate.csv from GUI
Outputs:	----------Election Result---------- * Election Type: Plurality * Number of Ballots: 20 * #Seats: 3 * #Candidates: 5 * Winners are: 1: C (7 ballots; %35.00 Votes) 2: B (5 ballots; %25.00 Votes) 3: A (4 ballots; %20.00 Votes) * Losers are: 4: D (3 ballots; %15.00 Votes) 5: E (1 ballots; %5.00 Votes) Location of audit report: \src\AuditFile_2020.04.03.140142.txt ----------End of Result Display----------
Passed or Failed:	Passed

Date:	5/3/2020

PBI:	GUI for the directory search
Task Description:	Test plurality election, multiple ballot files
Testing Number:	System_tests_S24
Tester:	Colin Kluegel
Inputs:	./VotingSystem Enter 0 for Plurality Election Enter 3 for number of seats Select plurality_20ballots_5candidate.csv and plurality_110ballots_5candidates.csv from GUI
Outputs:	----------Election Result---------- * Election Type: Plurality * Number of Ballots: 130 * #Seats: 3 * #Candidates: 5 * Winners are: 1: C (29 ballots; %22.31 Votes) 2: B (28 ballots; %21.54 Votes) 3: D (26 ballots; %20.00 Votes) * Losers are: 4: A (24 ballots; %18.46 Votes) 5: E (23 ballots; %17.69 Votes) Location of audit report: \src\AuditFile_2020.04.03.140315.txt ----------End of Result Display----------
Passed or Failed:	Passed
Date:	5/3/2020

PBI:	GUI for the directory search
Task Description:	Test STV election, single ballot file
Testing Number:	System_tests_S25
Tester:	Colin Kluegel
Inputs:	./VotingSystem Enter 1 for STV Election Enter 3 for number of seats Select stv_20ballots_5candidates_0pctBallotBallots.csv from GUI

Outputs:	----------Election Result---------- * Election Type: STV * # Ballots: 20 * # Invalid ballots: 0 * #Seats: 3 * #Candidates: 5 * Winners are: 1: B 2: C 3: A * Losers are: 1: E 2: D Location of audit report: \src\AuditFile_2020.04.03.140456.txt Location of invalidated ballots report: \src\InvalidBallotFile_2020.04.03.140456.txt ----------End of Result Display----------
Passed or Failed:	Passed
Date:	5/3/2020

PBI:	GUI for the directory search
Task Description:	Test STV election, multiple ballot files
Testing Number:	System_tests_S26
Tester:	Colin Kluegel
Inputs:	./VotingSystem Enter 1 for STV Election Enter 3 for number of seats Select stv_20ballots_5candidates_0pctBallotBallots.csv and stv_100ballots_5candidates_0pctBadBallots.csv from GUI

Outputs:	----------Election Result---------- * Election Type: STV * # Ballots: 120 * # Invalid ballots: 0 * #Seats: 3 * #Candidates: 5 * Winners are: 1: B 2: A 3: D * Losers are: 1: E 2: C Location of audit report: \src\AuditFile_2020.04.03.140836.txt Location of invalidated ballots report: \src\InvalidBallotFile_2020.04.03.140836.txt ----------End of Result Display----------
Passed or Failed:	Passed
Date:	5/3/2020

PBI:	GUI for the directory search
Task Description:	Test STV election, with shuffle disabled
Testing Number:	System_tests_S26
Tester:	Colin Kluegel
Inputs:	./VotingSystem -t
Outputs:	AuditFile_2020.04.03.141250.txt Segmentation fault (core dumped)
Passed or Failed:	Failed
Date:	5/3/2020

PBI:	Fix droop and plurality algorithm
Task Description:	Create STVElectionRecord object
Testing Number:	STVElectionRecord_UT_01
Tester:	Hailin Archer
Inputs:	list<STVCandidate*> list<ballot*> int droop
Outputs:	No throw
Passed or Failed:	All tests passed.
Date:	4/22/2020

PBI:	Fix droop and plurality algorithm
Task Description:	Verify GetNonDistributedBallotList function
Testing Number:	STVElectionRecord_UT_02
Tester:	Hailin Archer
Inputs:	Initialize an STVElectionRecord object with empty ballot list
Outputs:	Empty ballot list
Passed or Failed:	All tests passed.
Date:	4/22/2020

PBI:	Fix droop and plurality algorithm
Task Description:	Verify GetWinnersList function
Testing Number:	STVElectionRecord_UT_03
Tester:	Hailin Archer
Inputs:	A predetermined winners list
Outputs:	GetWinnersList function returns the same predetermined winners list
Passed or Failed:	All tests passed.
Date:	4/22/2020

PBI:	Fix droop and plurality algorithm
Task Description:	Verfity GetNonDistributedBallotList with a non empty ballot list
Testing Number:	STVElectionRecord_UT_04

Tester:	Hailin Archer
Inputs:	Initialize an STVElectionRecord object with a non-empty ballot list
Outputs:	GetNonDistributedBallotList returns the same non-empty ballot list
Passed or Failed:	All tests passed.
Date:	4/22/2020

PBI:	Fix droop and plurality algorithm
Task Description:	Verfity SortNonElectedCandidateList function
Testing Number:	STVElectionRecord_UT_05
Tester:	Hailin Archer
Inputs:	A predictable list of candidates and ballots
Outputs:	Results after sorting match the expected results
Passed or Failed:	All tests passed.
Date:	4/22/2020

PBI:	Fix droop and plurality algorithm
Task Description:	Verfity AddCandidateToWinnersList function
Testing Number:	STVElectionRecord_UT_06
Tester:	Hailin Archer
Inputs:	Add a predetermined list of candidates to the winners list by calling the AddCandidateToWinnersList function
Outputs:	Verfity that the candidates are added to the list in sequential order
Passed or Failed:	All tests passed.
Date:	4/22/2020

PBI:	Fix droop and plurality algorithm
Task Description:	Verfity AddCandidateToLosersList function
Testing Number:	STVElectionRecord_UT_07
Tester:	Hailin Archer
Inputs:	Add a predetermined list of candidates to the losers list by calling the AddCandidateToWinnersList function
Outputs:	Verfity that the candidates are added to the list in sequential order
Passed or Failed:	All tests passed.
Date:	4/22/2020

PBI:	Fix droop and plurality algorithm
Task Description:	Verify GetNonElectedCandidateList
Testing Number:	STVElectionRecord_UT_08
Tester:	Hailin Archer
Inputs:	Set a predetermined NonElectedCandidateList in a STVElectionRecord
Outputs:	Verify GetNonElectedCandidateList function returns the same list
Passed or Failed:	All tests passed.
Date:	4/22/2020

PBI:	Fix droop and plurality algorithm
Task Description:	Verify ShuffleBallots function
Testing Number:	STVElectionRecord_UT_09
Tester:	Hailin Archer
Inputs:	Set NonDistributedBallotList in an STVElectionRecord object, call ShuffleBallots function
Outputs:	GetNonDistributedBallotLIst returns a different NonDistributedBallotList (shuffled)
Passed or Failed:	All tests passed.
Date:	4/22/2020

PBI:	Fix droop and plurality algorithm
Task Description:	Verify CheckDroop function
Testing Number:	STVElectionRecord_UT_10
Tester:	Hailin Archer
Inputs:	Call CheckDroop function with several different inputs
Outputs:	CheckDroop returns true when the inputs are greater than droop, false when the inputs are less than droop
Passed or Failed:	All tests passed.
Date:	4/22/2020

PBI:	Fix droop and plurality algorithm
Task Description:	
Testing Number:	STVElectionRecord_UT_11
Tester:	Hailin Archer

Inputs:	
Outputs:	
Passed or Failed:	All tests passed.
Date:	4/22/2020

PBI:	Fix droop and plurality algorithm
Task Description:	Verify RemoveLastCandidateFromNonElectedCandidateList function
Testing Number:	STVElectionRecord_UT_12
Tester:	Hailin Archer
Inputs:	Set NonElectedCandidateList with a predetermined candidate list Call RemoveLastCandidateFromNonElectedCandidateList
Outputs:	NonElectedCandidateList has one less candidate member The candidate returned by the function is the expected candidate
Passed or Failed:	All tests passed.
Date:	4/22/2020

PBI:	Fix droop and plurality algorithm
Task Description:	Verify AddLoserBallotsToNonDistributedBallotList function
Testing Number:	STVElectionRecord_UT_13
Tester:	Hailin Archer
Inputs:	Call AddLoserBallotsToNonDistributedBallotList with a predetermined ballot list input
Outputs:	Verify that the same list is added to NonDistributedBallotList
Passed or Failed:	All tests passed.
Date:	4/22/2020

PBI:	Fix droop and plurality algorithm
Task Description:	Verify PopCandidateOffLosersList function
Testing Number:	STVElectionRecord_UT_14
Tester:	Hailin Archer
Inputs:	Set losersList with a predetermined candidate list Call PopCandidateOffLosersList function
Outputs:	Returned candidate is the expecte candidate
Passed or Failed:	All tests passed.

	Date:	4/22/2020

PBI:	Fix droop and plurality algorithm
Task Description:	Create STVElection object
Testing Number:	STVElectionRecord_UT_01
Tester:	Hailin Archer
Inputs:	VotingInfo object
Outputs:	No throw
Passed or Failed:	All tests passed.
Date:	4/22/2020

PBI:	Fix droop and plurality algorithm
Task Description:	Verify RunElection function
Testing Number:	STVElectionRecord_UT_02
Tester:	Hailin Archer
Inputs:	Call RunElection function
Outputs:	No throw
Passed or Failed:	All tests passed.
Date:	4/22/2020

PBI:	Fix droop and plurality algorithm
Task Description:	Verify DisplayResult function
Testing Number:	STVElectionRecord_UT_03
Tester:	Hailin Archer
Inputs:	Call DisplayResult function
Outputs:	No throw
Passed or Failed:	All tests passed.
Date:	4/22/2020

PBI:	Fix droop and plurality algorithm
Task Description:	Create a Logger object
Testing Number:	Logger_UT_01
Tester:	Hailin Archer
Inputs:	
Outputs:	
Passed or Failed:	All tests passed.
Date:	4/22/2020

PBI:	Fix droop and plurality algorithm
Task Description:	Log message to a log file
Testing Number:	Logger_UT_02
Tester:	Hailin Archer
Inputs:	String message
Outputs:	No throw
Passed or Failed:	All tests passed.
Date:	4/22/2020

PBI:	Fix droop and plurality algorithm
Task Description:	Log data type list<int> to a log file
Testing Number:	Logger_UT_03
Tester:	Hailin Archer
Inputs:	A list of integers
Outputs:	No throw
Passed or Failed:	All tests passed.
Date:	4/22/2020

PBI:	Fix droop and plurality algorithm
Task Description:	Log data type list<Candidate> to a log file
Testing Number:	Logger_UT_04
Tester:	Hailin Archer

	Inputs:	A list of candidate object
	Outputs:	No throw
	Passed or Failed:	All tests passed.
	Date:	4/22/2020
	PBI:	Fix droop and plurality algorithm
	Task Description:	Log data type list<Ballot> to a log file
	Testing Number:	Logger_UT_05
	Tester:	Hailin Archer
	Inputs:	A list of ballot object
	Outputs:	No throw
	Passed or Failed:	All tests passed.
	Date:	4/22/2020

PBI:	Invalidate STV ballots without enough rankings
Task Description:	Create code to identify and store invalid STV ballots
Testing Number:	Ballot_file_tests_UT_04
Tester:	Bryan Baker
Inputs:	invalid_ballots.csv ballot1 ballot4
Outputs:	Tests if a ballot file with invalid ballots can be imported. After importing test the voting info object has the correct number of candidates. Check that there are the correct number of valid ballots. Check that there are the corrrect number of invalid ballots. Check that the correct candidate information has been created. Check that the correct ballot information has been created for valid and invalid ballots. Check that the IsInvalid function works properly for plurality elections, stv elections, and with differing numbers of candidates.
Passed or Failed:	All tests passed.
Date:	4/16/2020

PBI:	Fix droop and plurality algorithm
Task Description	Test voting info constructor
Testing Number:	VotingInfo_UT_01
Tester:	Bryan Baker
Inputs:	NA
Outputs:	Tests if a votinginfo object can be created. Checks if the algorithm is correct. Checks if the number of ballots is zero. Checks if the number of candidates is zero. Checks is the number of seats has been correctly initialized.
Passed or Failed	All tests passed.
Date:	5/4/2020

PBI:	Fix droop and plurality algorithm
Task Description	Test voting info algorithm indicator
Testing Number:	VotingInfo_UT_02
Tester:	Bryan Baker
Inputs:	VotingInfo1 and votinginfo2
Outputs:	Checks that a votinginfo object initialized for plurality has a designation of plurality. Checks that a votinginfo object initialized for stv has a designation of plurality. Checks that a votinginto object with no designation has an algorithm designation that is niether plurality or stv.
Passed or Failed	All tests passed.
Date:	5/4/2020

PBI:	Fix droop and plurality algorithm
Task Description	Test voting info method to check the number of seats in the election.
Testing Number:	VotingInfo_UT_03
Tester:	Bryan Baker
Inputs:	VotingInfo1 and votinginfo2
Outputs:	Checks that a seat count of 3 has been set up for both votinginfo1 and votinginfo2. Checks that an invalid seat count of -1 is established if no seat count is given.
Passed or Failed	All tests passed.

Date:	5/4/2020
PBI:	Fix droop and plurality algorithm
Task Description	Add candidate to candidate list
Testing Number:	VotingInfo_UT_04
Tester:	Bryan Baker
Inputs:	VotingInfo1 and candidate1 and candidate2
Outputs:	Checks that the initial candidate count is zero. Add candidate1. Checks that the candidate count is 1. Add candidate2. Checks that the candidate count is 2. Try to add candidate1 again. Checks that the candidate count is still 2.
Passed or Failed	All tests passed.
Date:	5/4/2020
PBI:	Fix droop and plurality algorithm
Task Description	Add stv candidate to stv candidate list
Testing Number:	VotingInfo_UT_05
Tester:	Bryan Baker
Inputs:	VotingInfo2 and stv_candidate1 and stv_candidate2
Outputs:	Checks that the initial candidate count is zero. Add stv_candidate1. Checks that the candidate count is 1. Add stv_candidate2. Checks that the candidate count is 2. Try to add stv_candidate1 again. Checks that the candidate count is still 2.
Passed or Failed	All tests passed.
Date:	5/4/2020
PBI:	Fix droop and plurality algorithm
Task Description	Add ballot to ballot list
Testing Number:	VotingInfo_UT_06

Tester:	Bryan Baker
Inputs:	VotingInfo1 and ballot1 and ballot2
Outputs:	Checks that the initlal ballot count is zero. Add ballot1. Checks tha the ballot count is 1. Add ballot2. Checks that the ballot count is 2. Try to add ballot1 again. Checks that the ballot count is still 2.
Passed or Failed	All tests passed.
Date:	5/4/2020

PBI:	Invalidate STV ballots without enough rankings
Task Description	Add ballot to invalid list
Testing Number:	VotingInfo_UT_07
Tester:	Bryan Baker
Inputs:	VotingInfo1 and invalid1 and invalid2
Outputs:	Checks that the initlal invalid ballot count is zero. Add invalid1. Checks that the invalid ballot count is 1. Add invalid2. Checks that the invalid ballot count is 2. Try to add invalid1 again. Checks that the invalid ballot count is still 2.
Passed or Failed	All tests passed.
Date:	5/4/2020

PBI:	Fix droop and plurality algorithm
Task Description	Get number of candidates
Testing Number:	VotingInfo_UT_08
Tester:	Bryan Baker
Inputs:	VotingInfo1 and votinginfo2
Outputs:	Check that the intial number of candidates is zero for both votinginfo1 and votinginfo2.
Passed or Failed	All tests passed.
Date:	5/4/2020

PBI:	Fix droop and plurality algorithm
Task Description	Get number of ballots
Testing Number:	VotingInfo_UT_09
Tester:	Bryan Baker
Inputs:	VotingInfo1 and votinginfo2
Outputs:	Check that the intial number of ballots is zero for both votinginfo1 and votinginfo2.
Passed or Failed	All tests passed.
Date:	5/4/2020

PBI:	Fix droop and plurality algorithm
Task Description	Get number of invalid ballots
Testing Number:	VotingInfo_UT_10
Tester:	Bryan Baker
Inputs:	VotingInfo1 and votinginfo2
Outputs:	Check that the intial number of invalid ballots is zero for both votinginfo1 and votinginfo2.
Passed or Failed	All tests passed.
Date:	5/4/2020

PBI:	Fix droop and plurality algorithm
Task Description	Get candidate list
Testing Number:	VotingInfo_UT_11
Tester:	Bryan Baker
Inputs:	VotingInfo1 and candidate1 and candidate2
Outputs:	Check that the initial candidate list is empty. Add candidate 1. Get the candidate list and check that candidate1 was added to the list. Add candidate2. Get the candidate list and check that candidate2 was added to the list. Check that the size of the list is the same as the number of candidates.
Passed or Failed	All tests passed.
Date:	5/4/2020

PBI:	Fix droop and plurality algorithm
Task Description	Get stv candidate list
Testing Number:	VotingInfo_UT_12
Tester:	Bryan Baker
Inputs:	VotingInfo1 and stv_candidate1 and stv_candidate2
Outputs:	Check that the initial candidate list is empty. Add stv_candidate 1. Get the candidate list and check that stv_candidate1 was added to the list. Add stv_candidate2. Get the candidate list and check that stv_candidate2 was added to the list. Check that the size of the list is the same as the number of candidates.
Passed or Failed	All tests passed.
Date:	5/4/2020

PBI:	Fix droop and plurality algorithm
Task Description	Get ballot list
Testing Number:	VotingInfo_UT_13
Tester:	Bryan Baker
Inputs:	VotingInfo1 and ballot1 and ballot2
Outputs:	Check that the initial ballot list is empty. Add ballot1. Get the ballot list and check that ballot1 was added to the list. Add ballot2. Get the ballot list and check that ballot2 was added to the list. Check that the size of the list is the same as the number of ballots.
Passed or Failed	All tests passed.
Date:	5/4/2020

PBI:	Invalidate STV ballots without enough rankings
Task Description	Get invalid list
Testing Number:	VotingInfo_UT_14
Tester:	Bryan Baker
Inputs:	VotingInfo2 and invalid1 and invalid2

Outputs:	Check that the initial invalid ballot list is empty. Add invalid1. Get the invalid ballot list and check that invalid1 was added to the list. Add invalid2. Get the invalid ballot list and check that invalid2 was added to the list. Check that the size of the list is the same as the number of invalid ballots.
Passed or Failed	All tests passed.
Date:	5/4/2020

PBI:	Invalidate STV ballots without enough rankings
Task Description	WriteInvalidBallotsToFile_EvenNumCand
Testing Number:	VotingInfo_UT_15
Tester:	Josh Spitzer-Resnick
Inputs:	votinginfo2, candidate1, candidate2, candidate3, candidate4, invalid1, invalid2, and invalid3
Outputs:	Check that the initial candidate list is empty. Add candidate1, candidate2, candidate3, and candidate4. Check the number of candidates is equal to 4. Divide the number of candidates by 2.0 and check the number of candidates that having less than consistutes an invalid ballot (half) is 2.0. Check that the number of invalid ballots before adding any is 0. Add invalid1, invalid2, and invalid3. Check the number of invalid ballots is equal to 3. Get the list of invalid ballots and check that the size is equal to 3. While the list of invalid ballots is not empty: Get the ranked candidate list of the invalid ballot at the front of the invalid ballot list. Check that the size of the ranked candidate list is less than the number of candidates that having less than consistutes an invalid ballot (half). Pop the invalid ballot at the front of the invalid ballot list.
Passed or Failed	All tests passed.
Date:	5/4/2020

PBI:	Invalidate STV ballots without enough rankings
Task Description	WriteInvalidBallotsToFile_OddNumCand
Testing Number:	VotingInfo_UT_16
Tester:	Josh Spitzer-Resnick

	Inputs:	votinginfo2, candidate1, candidate2, candidate3, invalid1, invalid2, and invalid3	
	Outputs:	Check that the initial candidate list is empty. Add candidate1, candidate2, and candidate3. Check the number of candidates is equal to 3. Divide the number of candidates by 2.0 and check the number of candidates that having less than consistutes an invalid ballot (half) is 1.5. Check that the number of invalid ballots before adding any is 0. Add invalid1, invalid2, and invalid3. Check the number of invalid ballots is equal to 3. Get the list of invalid ballots and check that the size is equal to 3. While the list of invalid ballots is not empty: Get the ranked candidate list of the invalid ballot at the front of the invalid ballot list. Check that the size of the ranked candidate list is less than the number of candidates that having less than consistutes an invalid ballot (half). Pop the invalid ballot at the front of the invalid ballot list.	
	Passed or Failed	All tests passed.	
	Date:	5/4/2020	