Teaching Interactive Computer Science

Group 7

Alexander Kjellén
Björn Delin
Erik Skogby
Jan-Erik Bredahl
Joakim Israelsson

Functional Test Cases

These test corresponds to the Create data structure function in our RD

Create a Data Structure 1

Description: This test case is to assure that creating a data structure works correctly when no other data structure has been created.
Preconditions: No data structure has been created since the program started.
Expected outputs: The specified data structure is created and visible in the drawing area.
Step by step procedure: Choose a data structure from the menu. Restart the program. Repeat this for all data structures.

Create a Data Structure 2

Description: This test case is to assure that creating a data structure works correctly when another data structure has been created and no functions have run.
Preconditions: A data structure has already been created and no functions have run since the program started.
Expected outputs: The specified data structure is created and visible in the drawing area.
Step by step procedure: Choose a data structure from the menu. Repeat this for all data structures.

Create a Data Structure 3

Description: This test case is to assure that creating a data structure works correctly when another data structure has been created and a function has run.
Preconditions: A data structure has already been created and a function has run since the program started.
Expected outputs: When an animation is running the “run function” option is disabled. When the animation is paused the “run function” option is enabled.
Step by step procedure: Choose a data structure from the menu. Repeat this for all data structures.

These test corresponds to the Run a function in our RD

Run a function 1

Description: This test case is to check that running a function works as described in our RD. A function should not have run before on that data structure.
Preconditions: A data structure has already been created and no function has run.
Expected outputs: A function is run as it should.
Step by step procedure: Select a function and specify any eventual parameters.

Run a function 2

Description: This test case is to check that running a function, which takes no parameter, on a data structure works as described in our RD. A function should have run before on that data structure.
Preconditions: A data structure has already been created and a function has run on that data structure.
Expected outputs: A function is run as it should.
Step by step procedure: Select a function and specify any eventual parameters.

These test corresponds to the Add first function in our RD

Add first 1

Description: This test case is created to assure that the Add first function is working as described in our RD when the data structure is empty.
Run on: Linked list and array.
Preconditions: A data structure has been created and the data structure is empty.
Inputs: An integer.
Expected outputs: The parameter value is inserted first.
Step by step procedure: Select the AddFirst function. Select a correct parameter.

Add first 2

Description: This test case is created to assure that the Add first function is working as described in our RD when the data structure is full.
Run on: Linked list and array.
Preconditions: A data structure has been created and the data structure has the maximum amount of elements.
Inputs: An integer.
Expected outputs: An error message notifies the user that the structure is full. Nothing in the structure is changed.
Step by step procedure: Select the AddFirst function.

Add first 3

Description: This test case is created to assure that the Add first function is working as described in our RD when the data structure has some elements but is not full.
Run on: Linked list and array.
Preconditions: A data structure has been created and the data structure has some elements but is not full.
Inputs: An integer.
Expected outputs: The parameter value is inserted first.
Step by step procedure: Select the AddFirst function. Select a correct parameter.

Add first 4

Description: This test case is created to assure that the Add first function is working as described in our RD when bad parameters is inserted.
Run on: Linked list and array.
Preconditions: A data structure has been created.
Inputs: Characters, floats or integers not in our range.
Expected outputs: An error message notifies the user. Nothing in the structure is changed.
Step by step procedure: Select the AddFirst function. Select a incorrect parameter.

These test corresponds to the Insert function in our RD

Insert 1

Description: This test case assures that the Insert function works as specified in our RD when the data structure is empty.
Run on: Linked list, array and binary tree.

Preconditions: A data structure has been created and is empty.
Inputs: An integer.
Expected outputs: The parameter value is inserted at the specified index if singly linked list or array. Insert in sorted binary tree will place the parameter depending on the parameter value.
Step by step procedure: Select the Insert function. Select two correct parameters**.**

Insert 2

Description: This test case assures that the Insert function works as specified in our RD when the data structure is full.
Run on: Linked list, array and binary tree.
Preconditions: A data structure has been created and is full.
Inputs: An integer.
Expected outputs: An error message notifies the user. Nothing in the structure is changed.
Step by step procedure: Select the Insert function.

Insert 3

Description: This test case assures that the Insert function works as specified in our RD when the data structure non-empty and non-full.
Run on: Linked list, array and binary tree.
Preconditions: A data structure has been created and has elements in it but is not full.
Inputs: An integer.
Expected outputs: The parameter value is inserted at the specified index if singly linked list or array. Insert in sorted binary tree will place the parameter depending on the parameter value.
Step by step procedure: Select the Insert function. Select two correct parameters.

Insert 4

Description: This test case assures that the Insert function works as specified in our RD when the bad parameters are used.
Run on: Linked list, array and binary tree.
Preconditions: A data structure has been created.
Inputs: Characters, floats or integers not in our range.
Expected outputs: An error message notifies the user. Nothing in the structure is changed.
Step by step procedure: Select the Insert function. Use an incorrect value parameter. Select the Insert function. Use an incorrect index parameter.

These test corresponds to the Search function in our RD

Search 1

Description: This test case assures that the Search function works as specified in our RD when the data structure is empty.
Run on: Linked list, array and binary tree.
Preconditions: A data structure has been created and is empty.
Inputs: An integer.
Expected outputs: A message will notify the user that there are no elements in the data structure.
Step by step procedure: Select search function**.**

Search 2

Description: This test case assures that the Search function works as specified in our RD when the element searched for does not exist.
Run on: Linked list, array and binary tree.
Preconditions: A data structure has been created.
Inputs: An integer.
Expected outputs: Will go through the search algorithm and when it's done a message will notify the user that there are no element in the data structure that match your input.
Step by step procedure: Select search function. Select a parameter that does not exists in the data structure.

Search 3

Description: This test case assures that the Search function works as specified in our RD when the element searched for does exist.
Run on: Linked list, array and binary tree.
Preconditions: A data structure has been created and has some elements.
Inputs: An integer.
Expected outputs: Will go through the search algorithm and when it finds the element you searched for a message will notify the user that it found an element that match the input parameter.
Step by step procedure: Select search function. Select a parameter that do exist in the data structure.

Search 4

Description: This test case assures that the Search function works as specified in our RD when the parameters are invalid.
Run on: Linked list, array and binary tree.
Preconditions: A data structure has been created.
Inputs: Characters, floats or integers not in our range.
Expected outputs: An error message notifies the user. Nothing in the structure is changed.
Step by step procedure: Select search function. Select an incorrect parameter.

These test corresponds to the Set random values function in our RD

Set random values 1

Description: This test case is to check that the function set random values doesn't accept parameters that aren’t allowed**.**
Run on: Linked list, array and binary tree.
Precondition: A data structure has been loaded.
Inputs: Characters, floats or integers not in our range.
Expected outputs: An error message notifies the user. Nothing in the structure is changed.
Step by step procedure: Select set Random values function. Use incorrect parameter.

Set random values 2

Description: This test case is to check that the function will change the elements correct when you have a correctly input parameter.
Run on: Linked list, array and binary tree.
Precondition: A data structure has been loaded.

Inputs: An integer.
Expected outputs: All elements will be removed and the number of the input random elements will be created.
Step by step procedure: Select set Random values function. Use correct parameter.

These test corresponds to the Remove function in our RD

Remove 1

Description: This test case is to check that the function remove doesn't accept parameters that aren't allowed.
Run on: Linked list, array and binary tree.
Precondition: A data structure has been loaded.
Inputs: Characters, floats or integers not in our range.
Expected outputs: An error message notifies the user. Nothing in the structure is changed.
Step by step procedure: Select the remove function. Use incorrect parameter.

Remove 2

Description: This test case is to check that the function remove still works after you try to remove an element that doesn't exist.
Run on: Linked list, array and binary tree.
Precondition: A data structure has been loaded and has some elements.
Inputs: An integer.
Expected outputs: A message notifies the user. The parameter does not exist.
Step by step procedure: Select the remove function. Use correct parameter.

Remove 3

Description: This test case is to check that the function remove, removes the element after a correctly input parameter.
Run on: Linked list, array and binary tree.
Precondition: A data structure has been loaded, has some elements and has at least one element with the input value.
Inputs: An integer.
Expected outputs: Will go through the remove algorithm and when it finds the element you wanted to remove, it removes the element.
Step by step procedure: Select the remove function. Use correct parameter.

These test corresponds to the Remove by index function in our RD

Remove by index 1

Description: This test case is to check that the function remove by index doesn't accept parameters that aren't allowed.
Run on: Linked list and array.
Precondition: A data structure has been loaded.
Inputs: Characters, floats or integers not in our range.
Expected outputs: An error message notifies the user. Nothing in the structure is changed.
Step by step procedure: Select the remove by index function. Use incorrect parameter.

Remove by index 2

Description: This test case is to check that the function remove, removes the element after a correctly input index parameter.
Run on: Linked list and array.
Precondition: A data structure has been loaded and has at least one element with the input value.
Inputs: An integer.
Expected outputs: The value specified by the index parameter is removed.
Step by step procedure: Select the remove by index function. Use correct parameter**.**

These test corresponds to the Sort function in our RD

Sort 1

Description: This test case assures that each of the sorting algorithms work as specified in the RD when the data structure is empty.
Run on: Array.
Precondition: A data structure has been loaded and is empty.
Expected outputs: Nothing happens.
Step by step procedure: Select Insertion sort function. Select Merge sort function. Select Quick sort function.

Sort 2

Description: This test case assures that each of the sorting algorithms work as specified in the RD when the data structure has only one element.
Run on: Array.
Precondition: A data structure has been loaded and contains one element.
Expected outputs: Nothing happens.
Step by step procedure: Select Insertion sort function. Select Merge sort function. Select Quick sort function.

Sort 3

Description: This test case assures that each of the sorting algorithms work as specified in the RD when the data structure has several randomly generated numbers.
Run on: Array.
Precondition: A data structure has been loaded and has some randomly generated elements.
Expected outputs: After the sorting is run, the structure is sorted.
Step by step procedure: Select Insertion sort function. Fill structure with random values. Select Merge sort function. Fill structure with random values. Select Quick sort function.

Sort 4

Description: This test case assures that each of the sorting algorithms work as specified in the RD when the data structure is sorted.
Run on: Array.
Precondition: A data structure has been loaded and has sorted elements.
Expected outputs: Nothing happens.
Step by step procedure: Select Insertion sort function. Select Merge sort function. Select Quick sort function.

Sort 5

Description: This test case assures that each of the sorting algorithms work as specified in the RD when the data structure is sorted in reverse order.
Run on: Array.
Precondition: A data structure has been loaded and has elements sorted in reversed order.
Expected outputs: After the sorting is run, the structure is sorted.
Step by step procedure: Select Insertion sort function. Fill structure with random values. Select Merge sort function. Fill structure with random values. Select Quick sort function.

Sort 6

Description: This test case assures that each of the sorting algorithms work as specified in the RD when the data structure has a number of identical elements.
Run on: Array.
Precondition: A data structure has been loaded and has elements with the same value.
Expected outputs: Nothing happens.
Step by step procedure: Select Insertion sort function. Select Merge sort function. Select Quick sort function.

These test corresponds to the Animation part of our RD

Press a button

Description: This test case assures that none of the Animation control buttons can be pressed when neither a structure is not loaded nor a function has been run.
Precondition: A function has not been run.
Expected outputs: Nothing happens.
Step by step procedure: Try to press a animation control button.

Play

Description: This test case assures that the Play button only works when a structure is loaded, a function has been run and the animation is in a paused state.
Precondition: A data structure has been loaded, a function has been run and the animation is in paused state.
Expected outputs: The animation is running.
Step by step procedure: Press Play.

Pause

Description: This test case assures that the Pause button only works when a structure is loaded, a function has been run and the animation is in a running state.
Preconditions: A data structure has been loaded, a function has been run and the animation is in running state.
Expected outputs: The animation is paused.
Step by step procedure: Press Pause.

Step forward

Description: This test case assures that the StepForward button only works when a structure is loaded, a function has been run and the animation is in a paused state.
Preconditions: A data structure has been loaded, a function has been run and the animation is in

paused state.
Expected outputs: The animation starts to play and pauses when it finishes a step.
Step by step procedure: Press StepForwards.

Step backwards

Description: This test case assures that the StepBackwards button only works when a structure is loaded, a function has been run and the animation is in a paused state.
Preconditions: A data structure has been loaded, a function has been run and the animation is in paused state.
Expected outputs: The animation starts to play backwards and pauses when it finishes a step.
Step by step procedure: Press StepBackwards**.**

Stop

Description: This test case assures that the Stop button only works when a structure is loaded, a function has been run and the animation is in a paused state.
Preconditions: A data structure has been loaded, a function has been run and the animation is in paused state.
Expected outputs: The animation returns to the beginning immediately.
Step by step procedure: Press StepFirst.

Finish

Description: This test case assures that the Finish button only works when a structure is loaded, a function has been run and the animation is in a paused state.
Preconditions: A data structure has been loaded, a function has been run and the animation is in paused state.
Expected outputs: The animation reaches the end immediately.
Step by step procedure: Press StepLast.