Teaching Interactive Computer Science

Group 7

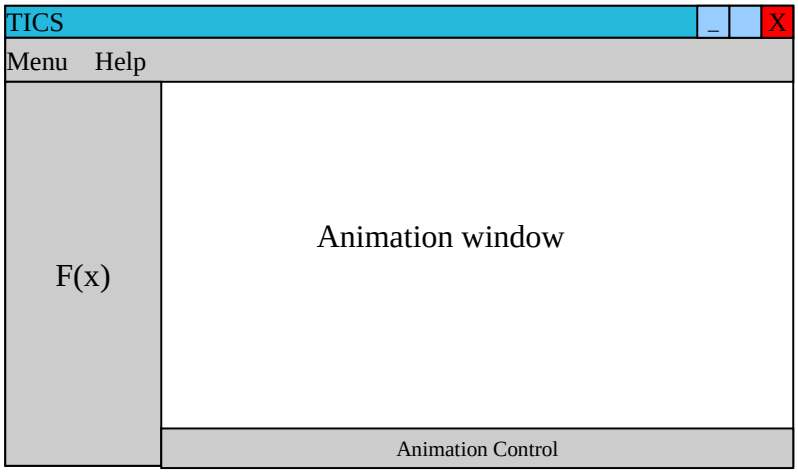
Alexander Kjellén
Björn Delin
Jan-Erik Bredahl
Joakim Israelsson
Erik Skogby

2007-11-27

Functional and non-functional requirement and use Cases

Functional requirements

- Integers are 32-bit signed integers.
- The program shall be shown in a window that looks like this:



- **Menu** – A standard menu list with two options: “Data structures” and “Exit”. Clicking on “Data structures” will bring up a sub list containing the available data structures. Clicking on “Exit” will shut down the program.
- **Help -** A menu list with one option: “About”. Clicking on “About” will show a window with information of the version and the authors.
- **F(x)** – In the window there will be a bar to the left where the functions in the data structures are. Each function will be shown as a button.
- **Animation Control** – This bar will contain the buttons Play, step forward, step backward and finish which will be used when the algorithm animates.
- **Animation window** – This space will contain the describing pictures of the data structures and their algorithms.

Included data structures

- **Array**
  - Indexes will start at zero.
  - The values stored in the array will be integers.
  - The array shall have a maximum size of 32 and a minimum of 0 posts.
  - It will have a counter of type integer with the number of posts in the array.
  - **The data structure Array shall have the following functions which the user can call:**
    1. Add(int arg, int i) – When this function is called, it shall show an animation of how a new post with value arg is inserted on index i in the array.
    2. Insertion_sort() – When this function is called, the function shall show an animation of how the array is sorted by using the insertion sort algorithm. When the algorithm is done, the values in the array shall be ordered, with the lowest value to the left.
    3. Merge_sort() - When this function is called, the function shall show an animation of how the array is sorted by using the mergesort algorithm. When the algorithm is done, the values in the array shall be ordered, with the lowest value to the left.
    4. Quick_sort() - When this function is called, the function shall show an animation of how the array is sorted by using the quicksort algorithm. When the algorithm is done, the values in the array shall be ordered, with the lowest value to the left.
    5. Remove(int i) – Shall show an animation of how the post at index i is removed. removes the post at index i and the posts after the index will move one step to the left of the array list
    6. Set_random_values() – Shall assign new random values to existing values.
  - **Visualization of the array**
    - The array shall be represented as a rectangle made up of the array's posts.
    - The array's posts shall be represented as smaller rectangles which contains numbers representing the value of the post.
    - The posts shall be positioned horizontally next to each other.
    - The array's first post shall be the left most and the last post shall be the right most.

- **Linked list**
  - The list is singly-linked.
  - The linked list contains posts.
  - The linked list's index will start at 0.
  - Each post contains a pointer and an integer value.
  - The list shall handle a maximum of 32 posts and a minimum of 0 posts.
  - It will have a pointer to the first post of the list.
  - It will have a pointer to the last post of the list.
  - It will have an integer counter with the number of post in the list.
  - If the list is empty the first and last pointer will point at null and the counter will be zero.
  - **The data structure Linked list shall have the following functions:**
    1. Add_first(int arg) – When this function is called, it shall show an animation of how a new post with value arg is inserted at the first position in the Linked list.
    2. Insert(int arg, int i) – When this function is called, it shall show an animation of how a new post with value arg is inserted on index i in the Linked List.
    3. Insertion_sort() – When this function is called, the function shall show an animation of how the Linked List is sorted by using the insertionsort algorithm. When the algorithm is done, the linked list's left most value shall be the lowest and the right most shall be the highest.
    4. Set_random_values() – Assigns all posts in the list with new random values. The amount of posts is still the same.
    5. Remove(int i) - Shall show an animation of how the post at index i is removed.
  - **Visualization of the Linked list**
    - Each post in the describing picture of the singly Linked list will be shown as a rectangle with the value of the node in it.
      The linked list will be represented as a rectangle.
    - The Linked list shall have two pointers. The first pointer shall point to the first post in the linked list. The second pointer shall point to the last post.
    - If there is only one post in the linked list, then both of the linked list's pointers shall point to that post.
    - Each post will be represented as a rectangle.
    - Each post will have a pointer to the next post in the list. The only exception is the last post, which shall not have a pointer pointing to any other post.
    - Each value must fit in the post and at the same time be readable.
    - If the linked list is empty, then the pointers of the linked list shall point to a null symbol.

- **Sorted binary tree**
  - The posts stored in the tree will contain integers.
  - The maximum number of posts shall be 31.
  - The minimum number of posts shall be 0.
  - Each post has 2 pointers that point at its children and one pointer to its parent.
  - **The data structure Sorted binary tree shall have the following functions:**
    1. Add(int arg) – When this function is called an animation will start that shows how the add algorithm works.
    2. Remove(int arg) - When this function is called an animation will start that shows how the remove algorithm works.
    3. Search(int arg) - When this function is called an animation will start that shows how a post is found the sorted binary tree.
    4. Depth() - When this function is called an animation will start that shows how to find the depth of the sorted binary tree.
  - **Visualization of the sorted binary tree**
    - Each post of the sorted binary tree will be shown as a rectangle with the value of the node in it.
    - Each post will have pointers to all their children.
    - The pointer will be expressed as an arrow from the parent node to the child node.
    - Each value must fit in the post and at the same time be readable.
    - The post's children will be placed under the post either to the left if it has a value that is lower than the post's value or to the right if it has a value that is higher than the post's value.
    - Under no circumstances may two sub trees collide with each other. If two sub trees grow into each other, the binary tree must be widened to make room for them.
- **All data structures will:**
  - Have a header object containing important information such as start pointers and size.
  - Put up a warning message if invalid arguments, such as negative indexes are supplied.

Non-functional requirements

- **Reliability** – The user shall not be able to call a function on a data structure which causes the system to crash.
- **System** – The computer must be a PC with at least a processor with 800 MHz and 512 MB ram. The program will be used on a 32-bit PC with the operating system “Windows XP”. The user must have SUN Java Run Time Environment 1.5 installed on his computer. Everything you do in the program should have a response-time less than 1 second.
- **Scalability** – Our program will work in the resolution 1024x768 pixels even though it is scalable we do not guarantee everything will fit in other resolutions.
- **Documentation** – All functions and data structures must be well documented so that users can find out exactly what they are looking at and what happens.
- **Space and delivery** – Since the program will be distributed through the internet, the total size of the program should not be larger than 20 Megabyte.
- **Standards** – Data structures should be visualized according to the standards used in books. For example, tree posts should be drawn with the father centered over the children nodes. The same thing for algorithms such as quicksort.

Use cases

Primary actor: Student Greger

Level: Summary

Stakeholder and interests:

1. Student – To get a better understanding of the algorithm without having to read a lot in his algorithm book

2. Teacher – To as fast as possible get the student to understand an algorithm so that he can continue with other parts of his course.

Minimal guarantees: The student sees an animation of the desired algorithm.

Success guarantees: The student understands the algorithm that well that he can explain it to others.

Main Success scenario:

1. Greger starts the program, loads the right data structure and algorithm.
2. He watches the animation, and steps it through a few times.
3. He understands how the algorithm works and exits the program.

Extensions:

1. a). The data structure or algorithm don't exist.
   Not much to do but to read about it in a book or Wikipedia.

2. a). He doesn't understand how the algorithm works.
   Greger has to read about it in a book, or another article that explains how it works before running the animation again.

Start software

Greger, a 20 year old student at KTH, wants to start the program Tics.

1. Greger starts the program Tics by double clicking on its desktop icon.
2. The program main view appears.

The use case ends.

Load a new structure

Greger wants to load a new data structure.

1. Greger presses the dropdown button named menu.
2. From the appearing list he clicks on the load button.
3. From a new visible list of data structures he clicks on the "Linked list".
4. The data structure list disappears.

5. A new list is shown of available functions on the data structure.

The use case ends.

Use a function with parameters

Greger has started the program and loaded the data structure Linked list.

1. Greger clicks on the function named Insert(int arg, int i).
2. Now, Greger gets a new window where he inputs the argument and the position i.
3. Now, he can see on the screen that the argument has been inserted.

The use case ends.

Use a function without parameters

Greger has started the program, loaded the data structure Array, and filled it with some random numbers.

1. Greger clicks on the function quicksort().
2. The buttons on the bottom of the screen becomes available.
3. Greger presses the play button.
4. Now Greger can see an animation of how the list is being sorted by the algorithm.

The use case ends.

Animation control

Greger has started the program, loaded the data structure Array, and filled it with some random numbers.

1. Greger selects the algorithm insertion sort.
2. The buttons on the bottom of the screen becomes available.
3. Greger presses play.
4. Now Greger can see an animation of how the list is being sorted by the algorithm.
5. Greger presses pause and the animation pauses.
6. Greger presses step forward and he sees how the animation makes one step in the algorithm.
7. Greger presses step backward and the animation animates goes one step backward in the algorithm.
8. Greger presses finish and the Array steps directly to the end of the algorithm.

The use case ends.