

Data Structure Visualization

Nguyen Hoang Phuc - 22125076

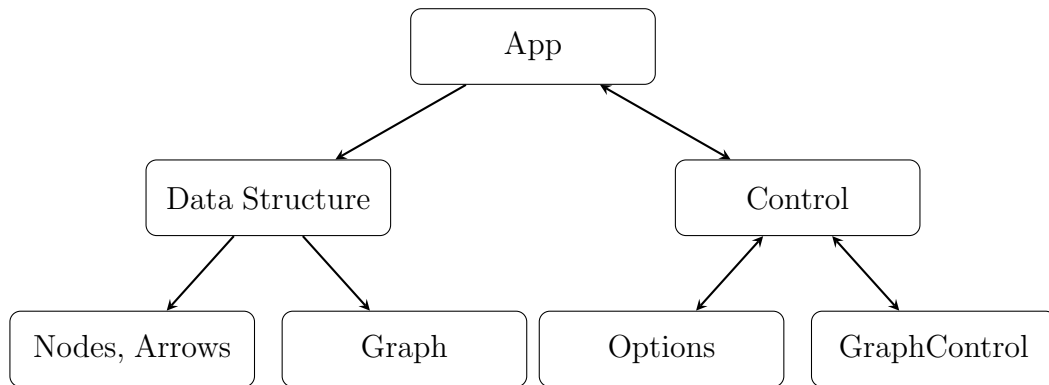
Saturday, 29/04/2023

Contents

1	Program structures	3
1.1	Overview	3
1.2	Description	3
1.3	Work flow	3
1.4	Details	3
1.4.1	Class App	3
1.4.2	Class Control	4
1.4.3	Struct Node	4
1.4.4	Struct Arrow	7
1.4.5	Class Graph	8
1.4.6	Other class/struct	11
2	Functional DS	12
3	User's manual	13
3.1	Main menu	13
3.2	Singly Linked List	13
3.2.1	Graph Control	14
3.2.2	Operations	14
4	Commit list	20
5	Link of git	20
6	Link of demo video	20

1 Program structures

1.1 Overview



1.2 Description

- Graph:
 - Graph manages to do steps in order.
 - Each step has a time interval for drawing and contains draw functions of nodes and arrows.
 - The order of draw functions is arranged so that in a frame, chosen nodes and arrows should be drawn on screen properly.
- Control: it contains Options and GraphControl
 - Options: contains several buttons such as Create, Insert, Search, Update, Delete, ...
 - GraphControl: contains several buttons such as Play, Pause, NextStep, PrevStep, GoToBegin, GoToEnd.

1.3 Work flow

When user click a button, the **Options** or **GraphControl** receives that and sends a command to **Control**. In every frame, **Control** asks **App** to execute the commands which are waiting. Those commands can be Create, Insert, Play, Pause, ... If there is a command is waiting, **App** will execute that by calling according functions in **Data Structure** and remove that command.

1.4 Details

1.4.1 Class App

1. `App()`;
 - Inits window configuration.
 - Loads font, background image.
2. `~App()`;
 - Deletes dynamically allocated memory of **Data Structure**.
3. `void processInput()`;
 - Transfers events from window to **Control** so that **Control** knows whether a button is clicked.
 - Handles when user want to exit the application.
4. `void update()`;

- Calls update functions of **Data Structure**.
5. `void draw();`
 - Calls draw function of **Graph** and draw background.
 6. `void run();`
 - Handles while loop of `processInput(); update(); draw();`, which is frame over frame.
 7. `void SLL_Update(); void DLL_Update();...`
 - Receives commands from **Control**.
 - Calls consistent functions of **Data Structure** such as **Create, Insert, Search, ...**

1.4.2 Class Control

1. `Control();`
 - Loads option's button background image.
 - Loads suboption's button background image.
 - Loads **Play, Pause, ...** button background image.
2. `void handleEvent(sf::Event& event, sf::RenderWindow* window);`
 - Receives events from **App**.
 - Generates a command if a button is clicked.
3. `void update(float dt);`
 - Calls update function of **InputBox** so that the cursor in the input box appears or disappears with respect of time `dt`.
4. `bool getCommand(Command& command);`
 - If there is at least a command in the `commandQueue`, that command will be assigned to `command` and return true.
 - On the other hand, return false.
5. `void loadSubOption();`
 - Loads corresponding suboptions if a option's button is clicked.
6. `void draw(sf::RenderTarget& target, sf::RenderStates states) const;`
 - Calls draw function of option's buttons
 - Calls draw function of suboption's buttons.
 - Calls draw function of **Play, NextStep, PrevStep, ...** buttons.

1.4.3 Struct Node

1. `Node(int val, sf::Vector2f pos);`
 - Init node's value and position on the window.
 - **Parameter:**
 - `val`: the initial value.
 - `pos`: the initial position.
2. `void drawCircle(sf::RenderWindow* window, sf::CircleShape* circle, sf::Color inColor, sf::Color outColor, sf::Text* num, sf::Color numColor);`

- Draw node on the window with given parameters.
 - **Parameter**
 - **window**: the pointer of current window.
 - **circle**: the shape of circle (is generated in **Graph**).
 - **inColor**: the inside color.
 - **outColor**: the outline color.
 - **num**: the text is num represented in string.
 - **numColor**: the text's color of value of that node.
3. `void drawCircleGrow(sf::RenderWindow* window, sf::CircleShape* circle, sf::Color inColor, sf::Color outColor, sf::Text* num, sf::Color numColor, float percent);`
- Draw node on the window with given parameters.
 - As the time goes by (**percent** increases gradually), the size of the circle will grow by multiplying **percent** with the radius.
 - **Parameter**:
 - **window**: the pointer of current window.
 - **circle**: the shape of circle (is generated in **Graph**).
 - **inColor**: the inside color.
 - **outColor**: the outline color.
 - **num**: the text is num represented in string.
 - **numColor**: the text's color of value of that node.
 - **percent**: the percentage of the size of the circle ($0 \leq \text{percent} \leq 1$).
4. `void drawCircleshrink(sf::RenderWindow* window, sf::CircleShape* circle, sf::Color inColor, sf::Color outColor, sf::Text* num, sf::Color numColor, float percent);`
- Draw node on the window with given parameters.
 - As the time goes by (**percent** decreases gradually), the size of the circle will shrink by multiplying **percent** with the radius.
 - **Parameter**:
 - **window**: the pointer of current window.
 - **circle**: the shape of circle (is generated in **Graph**).
 - **inColor**: the inside color.
 - **outColor**: the outline color.
 - **num**: the text is num represented in string.
 - **numColor**: the text's color of value of that node.
 - **percent**: the percentage of the size of the circle ($0 \leq \text{percent} \leq 1$).
5. `void drawCircleFadeIn(sf::RenderWindow* window, sf::CircleShape* circle, sf::Color inColor, sf::Color outColor, sf::Text* num, sf::Color numColor, float percent);`
- Draw node on the window with given parameters.
 - As the time goes by (**percent** increases gradually), the opacity of the circle will increase by setting the $\text{alpha} = \text{percent} \cdot 255$ of the RGBA color.
 - **Parameter**:
 - **window**: the pointer of current window.
 - **circle**: the shape of circle (is generated in **Graph**).
 - **inColor**: the inside color.
 - **outColor**: the outline color.

- `num`: the text is `num` represented in string.
 - `numColor`: the text's color of value of that node.
 - `percent`: the percentage of the opacity of the circle ($0 \leq \text{percent} \leq 1$).
6. `void drawCircleFadeOut(sf::RenderWindow* window, sf::CircleShape* circle, sf::Color inColor, sf::Color outColor, sf::Text* num, sf::Color numColor, float percent);`
- Draw node on the window with given parameters.
 - As the time goes by (`percent` decreases gradually), the opacity of the circle will decrease by setting the `alpha=percent*255` of the RGBA color.
 - **Parameter:**
 - `window`: the pointer of current window.
 - `circle`: the shape of circle (is generated in **Graph**).
 - `inColor`: the inside color.
 - `outColor`: the outline color.
 - `num`: the text is `num` represented in string.
 - `numColor`: the text's color of value of that node.
 - `percent`: the percentage of the opacity of the circle ($0 \leq \text{percent} \leq 1$).
7. `void drawCircleMove(sf::RenderWindow* window, sf::Vector2f src, sf::Vector2f dest, sf::CircleShape* circle, sf::Color inColor, sf::Color outColor, sf::Text* num, sf::Color numColor, float percent);`
- Draw node on the window with given parameters.
 - As the time goes by (`percent` decreases gradually), the size of the circle will go from `src` to `dest` by multiplying `percent` with the length of the path.
 - **Parameter:**
 - `window`: the pointer of current window.
 - `circle`: the shape of circle (is generated in **Graph**).
 - `inColor`: the inside color.
 - `outColor`: the outline color.
 - `num`: the text is `num` represented in string.
 - `numColor`: the text's color of value of that node.
 - `percent`: the percentage of the size of the circle ($0 \leq \text{percent} \leq 1$).
8. The draw functions of square figure are similar to the draw functions of circle figure (1-7).
9. `void drawSubscript(sf::RenderWindow* window, sf::Text* text, std::string str, sf::Color textColor, SubscriptDir dir);`
- Draw subscript besides node on the window with given parameters.
 - **Parameter**
 - `window`: the pointer of current window.
 - `text`: the pointer of text figure.
 - `str`: the subscript's string.
 - `textColor`: the text's color.
 - `dir`: position of the subscript with respect to the position of the chosen node (UP/ DOWN/ LEFT/ RIGHT).

1.4.4 Struct Arrow

1. `Arrow(sf::Vector2f src, sf::Vector2f dest, bool flag);`
 - Set the source node and destination node of the arrow.
 - **Parameter:**
 - `src`: the source node.
 - `dest`: the destination node.
 - `flag`: the flag to determine it's for Singly Linked List or Doubly Linked List.
2. `void update(ArrowFigure* arrowFig);`
 - Update the arrow's position, length, rotation.
 - **Parameter:**
 - `arrowFig`: the pointer of arrow figure.
3. `void draw(sf::RenderWindow* window, ArrowFigure* arrowFig, sf::Color color);`
 - Draw the arrow on the window with given parameters.
 - **Parameter:**
 - `window`: the pointer of current window.
 - `arrowFig`: the pointer of arrow figure.
 - `color`: the color of the arrow.
4. `void drawGrow(sf::RenderWindow* window, ArrowFigure* arrowFig, sf::Color color, float percent);`
 - Draw the arrow on the window with given parameters.
 - As the time goes by (`percent` increases gradually), the size of the arrow will grow by multiplying `percent` with the length of the arrow.
 - **Parameter:**
 - `window`: the pointer of current window.
 - `arrowFig`: the pointer of arrow figure.
 - `color`: the color of the arrow.
 - `percent`: the percentage of the size of the arrow ($0 \leq \text{percent} \leq 1$).
5. `void drawShrink(sf::RenderWindow* window, ArrowFigure* arrowFig, sf::Color color, float percent);`
 - Draw the arrow on the window with given parameters.
 - As the time goes by (`percent` decreases gradually), the size of the arrow will shrink by multiplying `percent` with the length of the arrow.
 - **Parameter:**
 - `window`: the pointer of current window.
 - `arrowFig`: the pointer of arrow figure.
 - `color`: the color of the arrow.
 - `percent`: the percentage of the size of the arrow ($0 \leq \text{percent} \leq 1$).
6. `void drawFadeIn(sf::RenderWindow* window, ArrowFigure* arrowFig, sf::Color color, float percent);`
 - Draw the arrow on the window with given parameters.
 - As the time goes by (`percent` increases gradually), the opacity of the arrow will increase by setting `alpha=percent*255` in RGBA color.
 - **Parameter:**

- **window**: the pointer of current window.
 - **arrowFig**: the pointer of arrow figure.
 - **color**: the color of the arrow.
 - **percent**: the percentage of the opacity of the arrow ($0 \leq \text{percent} \leq 1$).
7. `void drawFadeOut(sf::RenderWindow* window, ArrowFigure* arrowFig, sf::Color color, float percent);`
- Draw the arrow on the window with given parameters.
 - As the time goes by (**percent** decreases gradually), the opacity of the arrow will decrease by setting `alpha=percent*255` in `RGBA` color.
 - **Parameter**:
 - **window**: the pointer of current window.
 - **arrowFig**: the pointer of arrow figure.
 - **color**: the color of the arrow.
 - **percent**: the percentage of the opacity of the arrow ($0 \leq \text{percent} \leq 1$).

1.4.5 Class Graph

1. `Graph();`
 - Get window pointer and font from **App**.
2. `void clear();`
 - Clear all steps.
 - Set `curFrame=curStep=0`.
3. `void Graph::finishAllSteps();`
 - Finish all steps and clear all previous draw functions.
4. `void addStep(int frames);`
 - Add new step in `drawFunc`
 - **Parameter**:
 - **frames**: Number of frames of added step.
5. `void goToBegin();`
 - Go to the first step.
6. `void goToEnd();`
 - Go to the last step.
7. `bool isDoneAllSteps();`
 - Check if the last step is done or not.
8. `void nextStep();`
 - Finish current step and execute next step.
9. `void prevStep();`
 - Finish current step and execute previous step.
10. `void setVisualDir(VisualDir d);`
 - Set visual direction (`FORWARD/ BACKWARD`).
 - **Parameter**

- d: visual direction.
11. `void setVisualType(VisualType t);`
 - Set visual type (AUTO, STEP_BY_STEP).
 - **Parameter**
 - t: visual type.
 12. `void draw(Node* node, NodeType type, sf::Color inColor, sf::Color outColor, sf::Color numColor);`
 - Push back draw function of node in drawFunc.
 - **Parameter**
 - node: the pointer of chosen node.
 - type: shape of the node (SQUARE/ CIRCLE).
 - inColor: the inside color.
 - outColor: the outline color.
 - numColor: the text's color of value of that node.
 13. `void drawGrow(Node* node, NodeType type, sf::Color inColor, sf::Color outColor, sf::Color numColor);`
 - Push back drawGrow function of node in drawFunc.
 - **Parameter**
 - node: the pointer of chosen node.
 - type: shape of the node (SQUARE/ CIRCLE).
 - inColor: the inside color.
 - outColor: the outline color.
 - numColor: the text's color of value of that node.
 14. `void drawShrink(Node* node, NodeType type, sf::Color inColor, sf::Color outColor, sf::Color numColor);`
 - Push back drawShrink function of node in drawFunc.
 - **Parameter**
 - node: the pointer of chosen node.
 - type: shape of the node (SQUARE/ CIRCLE).
 - inColor: the inside color.
 - outColor: the outline color.
 - numColor: the text's color of value of that node.
 15. `void drawFadeIn(Node* node, NodeType type, sf::Color inColor, sf::Color outColor, sf::Color numColor);`
 - Push back drawFadeIn function of node in drawFunc.
 - **Parameter**
 - node: the pointer of chosen node.
 - type: shape of the node (SQUARE/ CIRCLE).
 - inColor: the inside color.
 - outColor: the outline color.
 - numColor: the text's color of value of that node.
 16. `void drawFadeOut(Node* node, NodeType type, sf::Color inColor, sf::Color outColor, sf::Color numColor);`
 - Push back drawFadeOut function of node in drawFunc.

- **Parameter**
 - `node`: the pointer of chosen node.
 - `type`: shape of the node (SQUARE/ CIRCLE).
 - `inColor`: the inside color.
 - `outColor`: the outline color.
 - `numColor`: the text's color of value of that node.
17. `void drawSubscript(Node* node, std::string str, sf::Color textColor, SubscriptDir dir=DOWN);`
- Push back `drawSubscript` function of `node` in `drawFunc`.
 - **Parameter**
 - `node`: the pointer of chosen node.
 - `str`: the subscript's string.
 - `textColor`: the text's color.
 - `dir`: position of the subscript with respect to the position of the chosen node (UP/ DOWN/ LEFT/ RIGHT).
18. `void draw(Arrow* arrow, sf::Color color);`
- Push back `draw` function of `arrow` in `drawFunc`.
 - **Parameter**
 - `arrow`: the pointer of the chosen arrow.
 - `color`: the color of the arrow.
19. `void drawGrow(Arrow* arrow, sf::Color color);`
- Push back `drawGrow` function of `arrow` in `drawFunc`.
 - **Parameter**
 - `arrow`: the pointer of the chosen arrow.
 - `color`: the color of the arrow.
20. `void drawShrink(Arrow* arrow, sf::Color color);`
- Push back `drawShrink` function of `arrow` in `drawFunc`.
 - **Parameter**
 - `arrow`: the pointer of the chosen arrow.
 - `color`: the color of the arrow.
21. `void drawFadeIn(Arrow* arrow, sf::Color color);`
- Push back `drawFadeIn` function of `arrow` in `drawFunc`.
 - **Parameter**
 - `arrow`: the pointer of the chosen arrow.
 - `color`: the color of the arrow.
22. `void drawFadeOut(Arrow* arrow, sf::Color color);`
- Push back `drawFadeOut` function of `arrow` in `drawFunc`.
 - **Parameter**
 - `arrow`: the pointer of the chosen arrow.
 - `color`: the color of the arrow.
23. `void draw(CodeBox* codeBox, int pos);`
- Push back `draw` function of `codeBox` in `drawFunc`.

- **Parameter**

- `codeBox`: the pointer of the chosen code box.
- `pos`: the line number to be highlighted.

24. `void draw();`

- Calls all draw functions of current step that was pushed to `drawFunc`.

1.4.6 Other class/struct

There are more classes or structs in the application that are not mentioned above. They are:

1. **Struct Figure**: generate the figure of circle, square, and arrow.
2. **Struct List**: manage properties and methods of the list of elements (node/arrow).
3. **Struct ListElement**: manage properties and methods of a element (node/arrow).
4. **Class ArrowFigure**: manage properties and methods of the arrow figure.
5. **Class Button**: generate, manage properties and methods of a button.
6. **Class CodeBox**: generate, manage properties and methods of a code box.
7. **Class Graph_Control**: manage methods of the graph control.
8. **Class InputBox**: generate, manage properties and methods of a input box.
9. **Class Menu**: generate, manage properties and methods of a menu.
10. **Class TextBox**: generate, manage properties and methods of a text box.
11. **Class SLL_Control**, **Class DLL_Control**, ...: manage the options, suboptions, and commands.

The reason why I do not write detail about them is that they are not important as the classes/structs mentioned above. They are just used to support the application.

2 Functional DS

There are several abbreviation of the name of the data structures in the table:

1. **SLL**: Singly Linked List.
2. **DLL**: Doubly Linked List.
3. **CLL**: Circular Linked List.
4. **SArr**: Static Array.
5. **DArr**: Dynamic Array.

There are notes in the table such as:

- ✓: the function was developed.
- \oplus : the function has been developing.
- \times : the function has not been done.

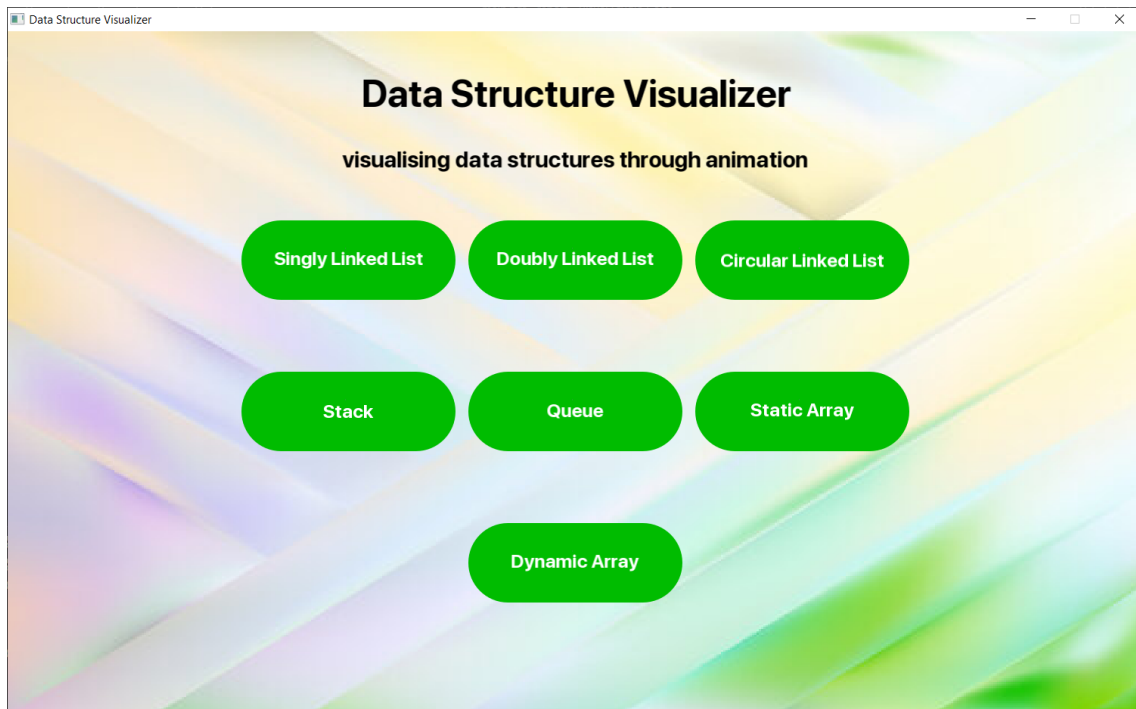
Data Structures		Stack	Queue
Functions			
Create	Empty Manual Random Random fixed size Load from file	✓	✓
Push		✓	✓
Pop		✓	✓
Top/Front		✓	✓

Data Structures		SLL	DLL	CLL	SArr	DArr
Functions						
Create	Empty Manual Random Random fixed size Load from file	✓	✓	\times	✓	✓
Insert	At the first After the last In the middle	✓	✓	\times	✓	✓
Delete	The first The last In the middle	✓	✓	\times	✓	✓
Update		✓	✓	\times	✓	✓
Search		✓	✓	\times	✓	✓

3 User's manual

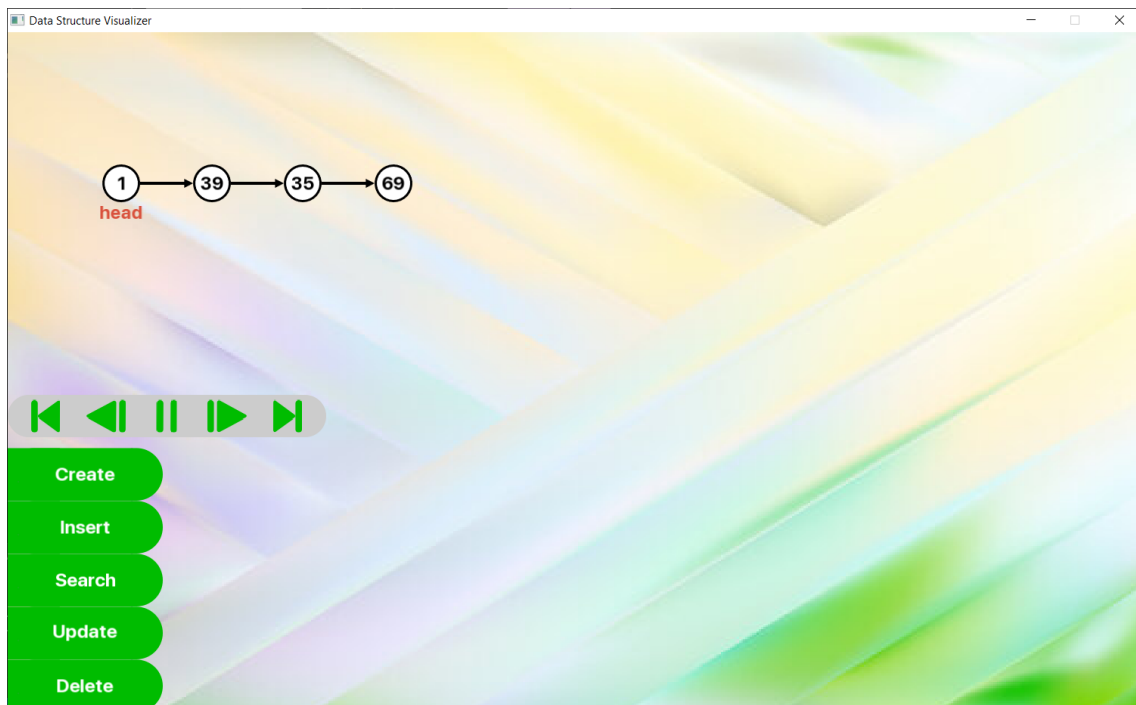
3.1 Main menu

In order to start the application, you need to run the file `DSV.exe` in the folder `Release`. This is the dash board of the application:



3.2 Singly Linked List

If you choose `Singly Linked List` by clicking the button `Singly Linked List`, you will see the following window:



3.2.1 Graph Control



From left to right we have buttons which are:

1. Go to the begining of the animation.
2. Go to previous step.
3. Play/Pause the animation.
4. Go to next step.
5. Go to the end of the animation.

3.2.2 Operations

1. Create:

- (a) **Empty**: create an empty list.
- (b) **Manual**: create a list by inputing elements.
- (c) **Random**: create a list by random elements.
- (d) **Random fixed size**: create a list by random elements with a fixed size.
- (e) **Load from file**: create a list by loading from a file.



Figure 1: (a)



Figure 2: (b)



Figure 3: (c)



Figure 4: (d)

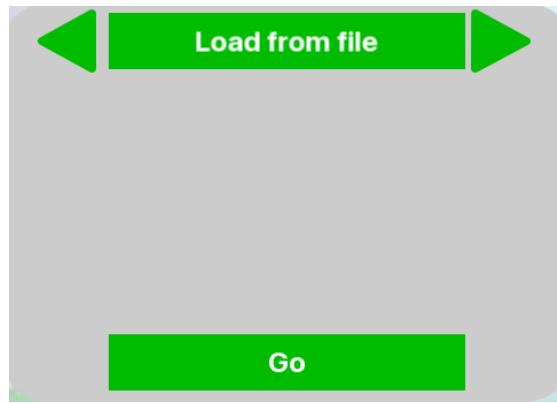


Figure 5: (d)

You can switch between options by clicking the triangle buttons.
If you want to perform the operation, click the button **Go**.

2. Insert:

- (a) **At the first:** insert v at the first position.
- (b) **After the last:** insert v after the last position.
- (c) **In the middle:** insert v at the position i .

Constraints:

- v must be an integer and $v \in [0, 99]$.
- i must be an integer and $i \in [0, \text{size} - 1]$.

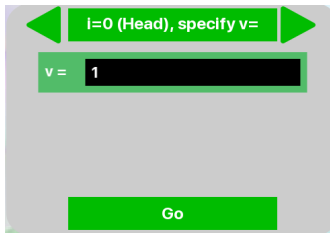


Figure 6: (a)

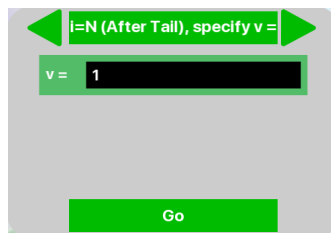


Figure 7: (b)

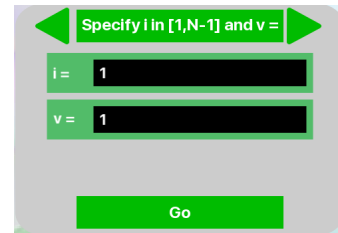
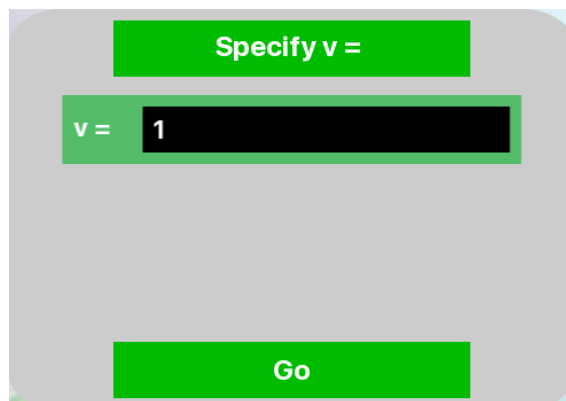


Figure 8: (c)

3. Search: search v in the list.

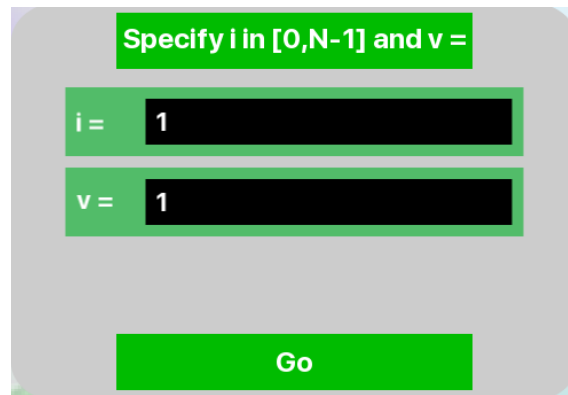
Constraints: v must be an integer and $v \in [0, 99]$.



4. **Update:** update the value of the node at the position i to v .

Constraints:

- v must be an integer and $v \in [0, 99]$.
- i must be an integer and $i \in [0, \text{size} - 1]$.

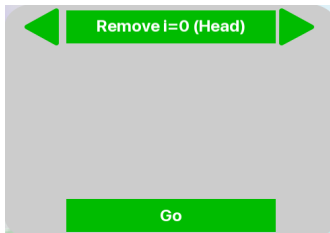


A user interface for the 'Update' operation. It features a green header bar with the text 'Specify i in [0,N-1] and v ='. Below this, there are two input fields: the first is labeled 'i =' and contains the value '1'; the second is labeled 'v =' and also contains the value '1'. At the bottom of the interface is a green button labeled 'Go'.

5. **Delete:**

- (a) **The first:** delete the first node.
- (b) **The last:** delete the last node.
- (c) **In the middle:** delete the node at the position i .

Constraints: i must be an integer and $i \in [0, \text{size} - 1]$.



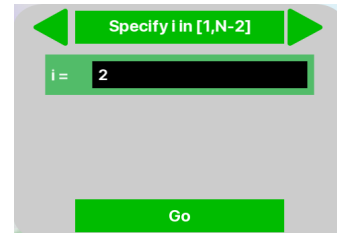
A user interface for deleting the first node. It has a green header bar with the text 'Remove i=0 (Head)' flanked by left and right arrow icons. At the bottom is a green button labeled 'Go'.

Figure 9: (a)



A user interface for deleting the last node. It has a green header bar with the text 'Remove i=N-1 (Tail)' flanked by left and right arrow icons. At the bottom is a green button labeled 'Go'.

Figure 10: (b)



A user interface for deleting a node in the middle. It has a green header bar with the text 'Specify i in [1,N-2]' flanked by left and right arrow icons. Below the header is an input field labeled 'i =' containing the value '2'. At the bottom is a green button labeled 'Go'.

Figure 11: (c)

You can see some examples of animation below:

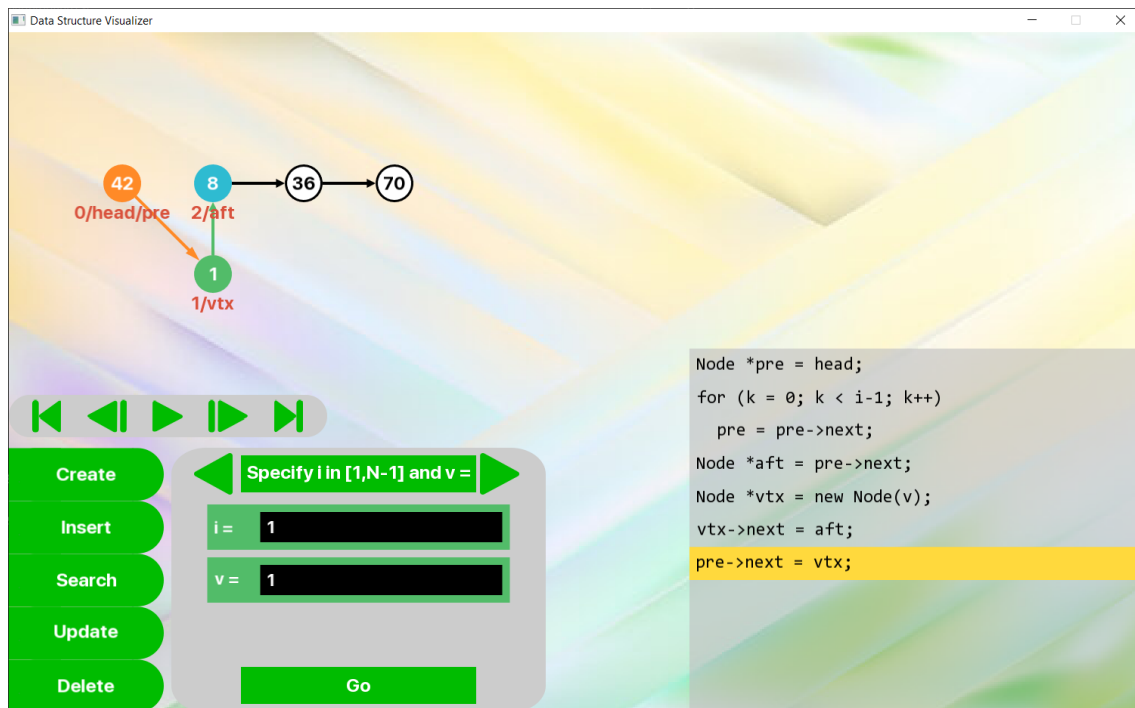


Figure 12: Inserting a node has value 1 at position 2

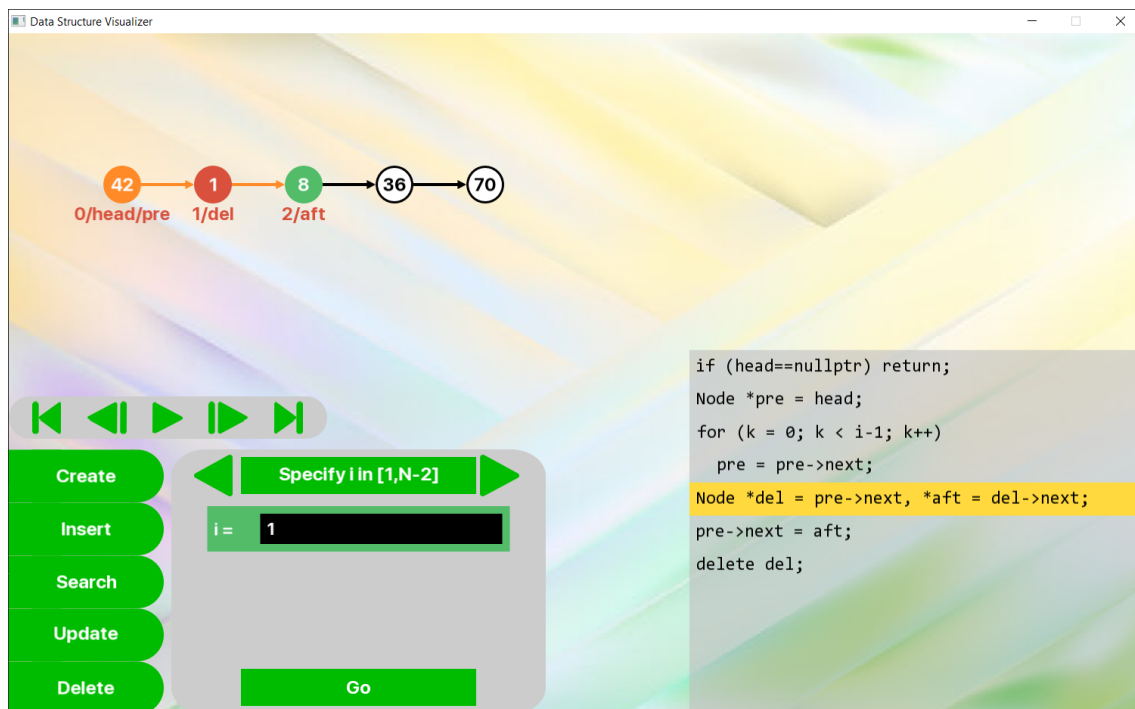


Figure 13: Deleting a node at position 2

Press **Esc** to go back to the main menu.

Press **Alt-F4** or click the **X** button on the top right corner to exit the program.

Other data structures have similar appearance and operations.

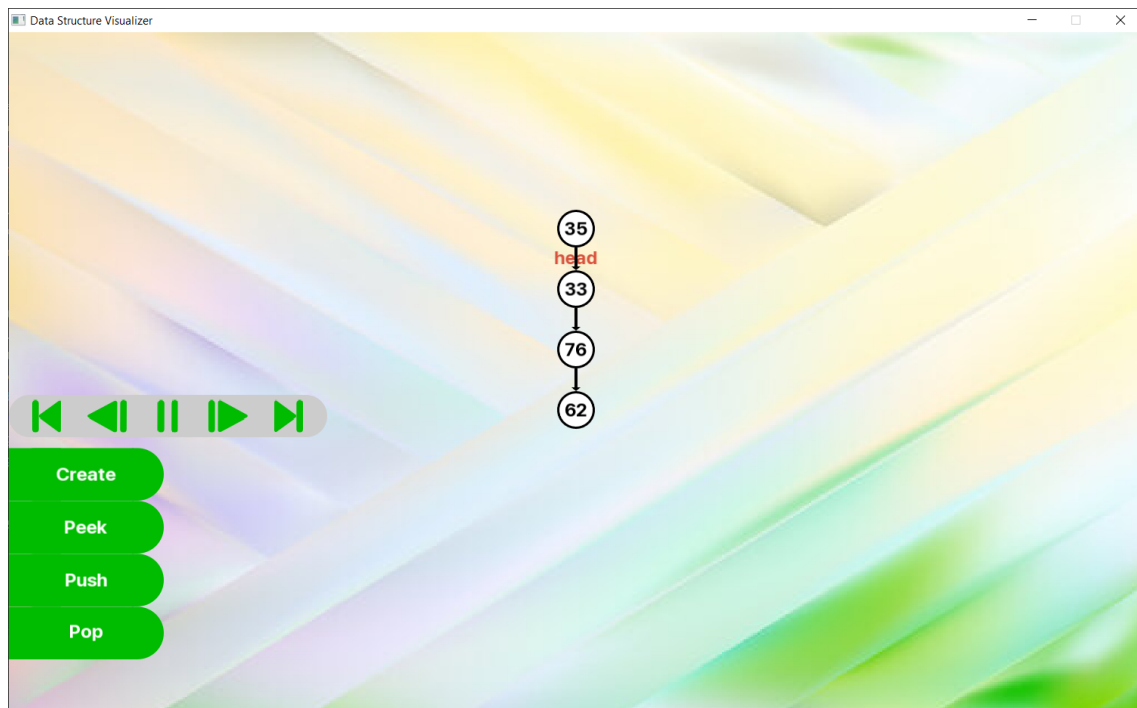


Figure 14: Stack

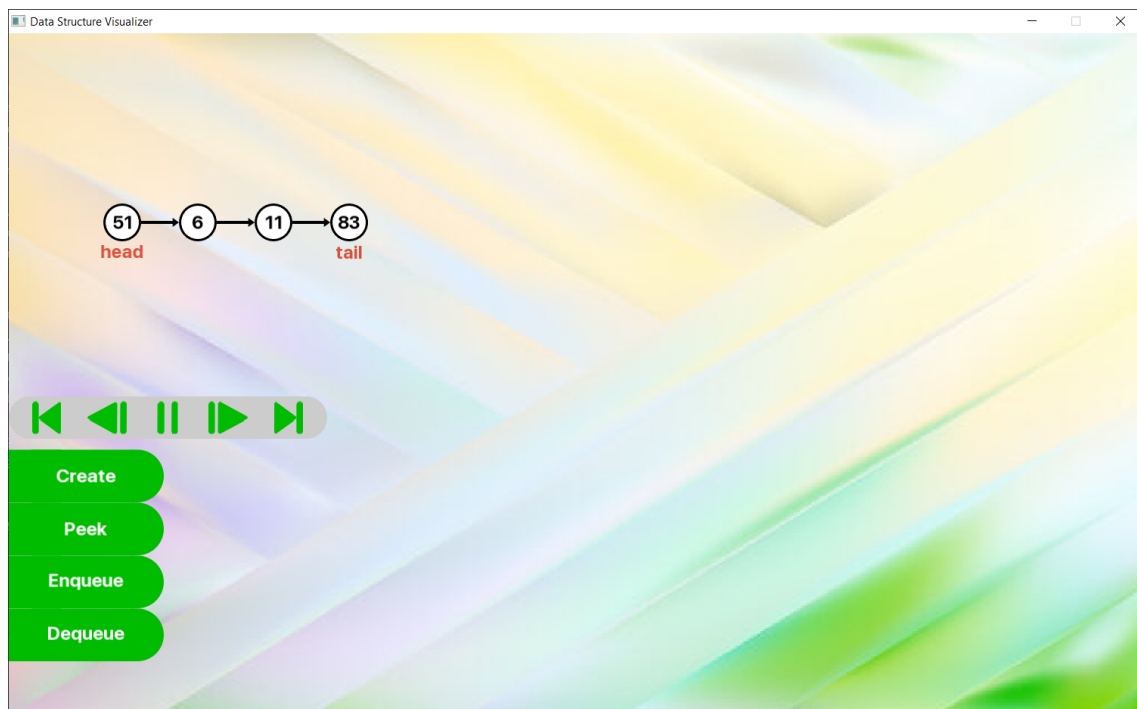


Figure 15: Queue

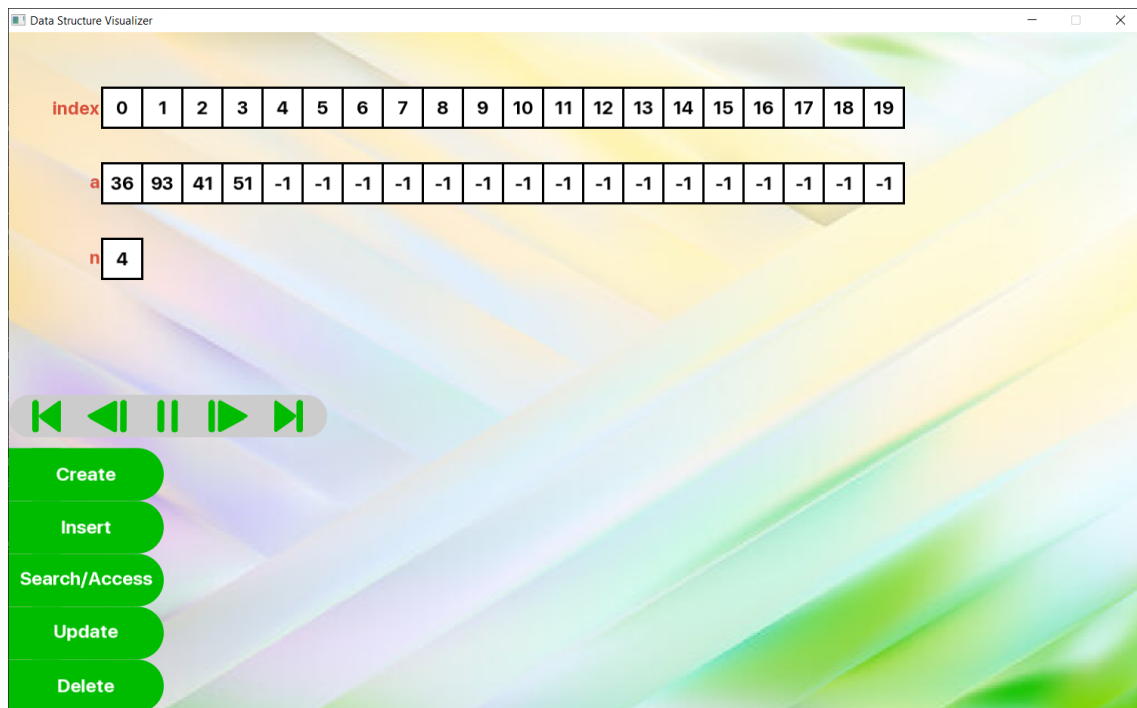


Figure 16: Static Array

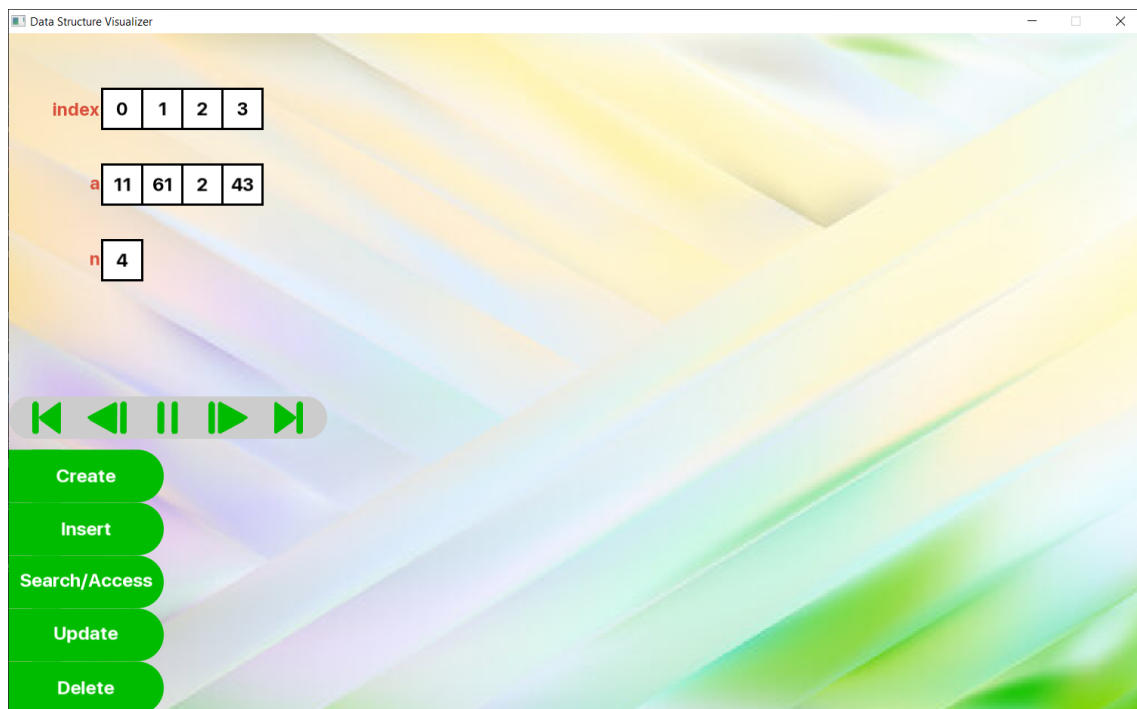


Figure 17: Dynamic Array

4 Commit list

<https://docs.google.com/document/d/1No2OZ4w78wAb2kXCmfLugrSHtgGDN4zQazmpcmGRgjM/edit?usp=sharing>

5 Link of git

https://github.com/slo248/CS162_DS_Visualizer.git

6 Link of demo video

<https://youtu.be/07sIxsPMg9Q>