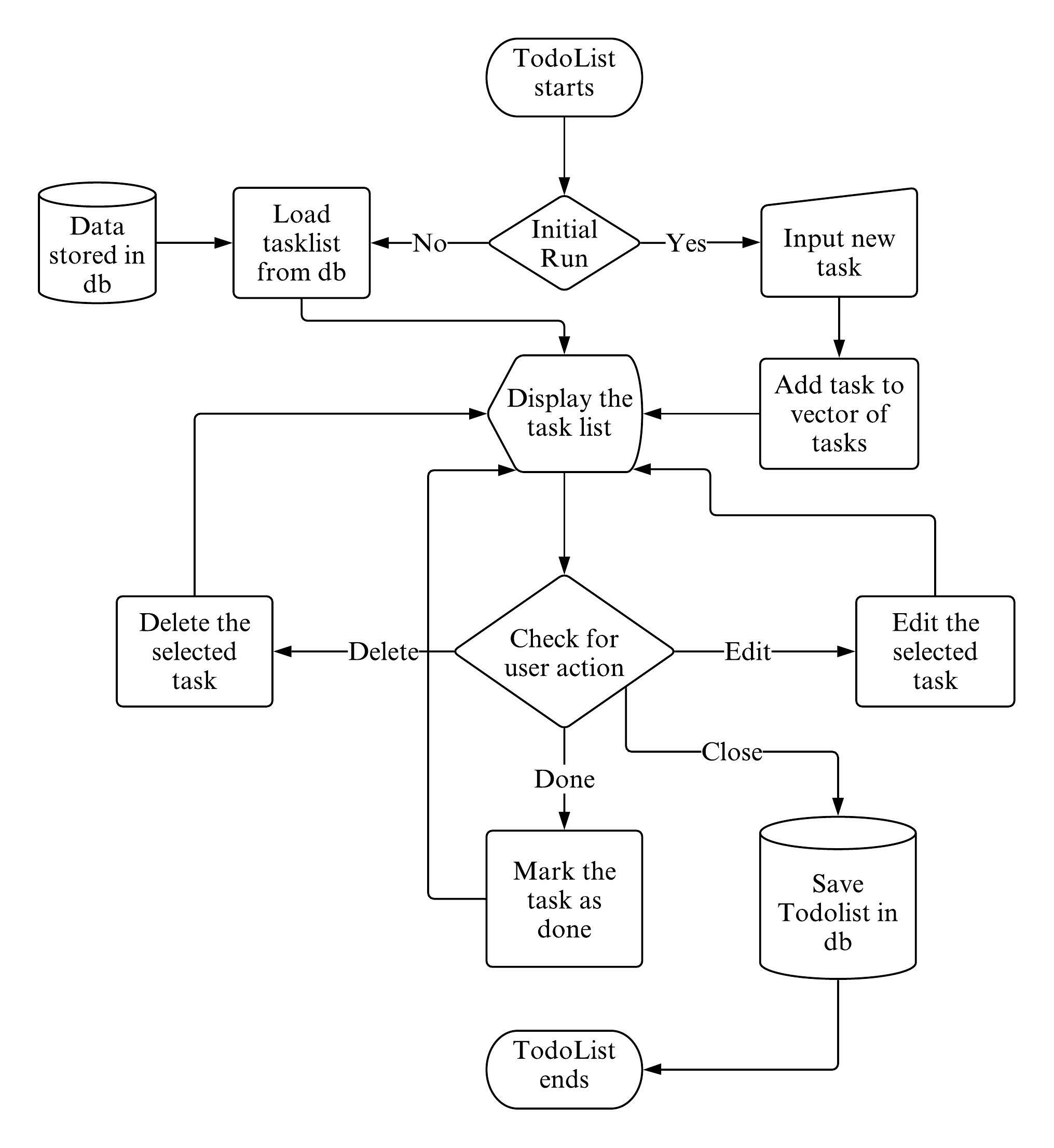
## TodoList:



The proposed flowchart of the program is shown above which shows how the program works. This will be implemented with the help of user defined functions and class. A separate class to take input from the user and store the task is used. And the tasks are stored in a *vector* of the same class. The *TodoList* is printed by iterating the *vector*. There are *buttons* to mark *done*, *edit*, *delete* the task for each task and the user action is checked through a function which points the operations as per the user action and respective action is carried out. Previously added task are loaded from the *database*when the program is started and the added tasks are saved to the *database* when the program is closed. Different parts of our methodology are as follows,

### Loading TodoList from db:

When TodoList is opened, first it is checked whether there are tasks added previously or not in the db and if the task existed, they’re loaded into the program with the help of loader function into the vector of the tasks and display the task in the window along with the Inputfield to add new task. If there were no previous task , we simply display the window without any task and input field to add the task. The loader function interacts with the sqlite api to read the data from the db.

### Adding New Task:

To add the task in our TodoList we have an Inputfield at the top of our task list where user can enter the name of the task and press enter to add the task to the list. For this we’ll use the **sf::Text** object available in sfml to store the task and give output in window, **sf::Event** object to check if the user is entering the text. The input area itself is an object of **Button** *Class* which needs to be created utilizing the previously available classes like **sf::Text**, **sf::Shape** (rectangle, circle), **sf::Event** etc. to get the desired result.

### Editing Tasks:

Every task in the TodoList has its own **EDIT** button to provide user an option to edit the task if they want to modify the task they previously added. The edit button is also an object of **Button** *Class.* When edit button is pressed it’ll load the task into the input area at the top where user can edit the task and press enter to save the changes.

### Deleting Tasks:

If the users need not to perform the task they added previously, every task has its own **DELETE** button to enable user to delete the unwanted task. **DELETE** button is similar to **EDIT** button but instead of editing it deletes the task.

### Marking Completed Tasks:

There is a small **Check button** to mark the completed task for every task in the list. It is also similar to edit and delete button but marks the task as done and draws strikethrough line on the task.

### Showing Due Tasks:

If the task are not completed within the day they’re marked as due and displayed under the Previous Dues List in the TodoList above the tasks that are not overdue.

### Displaying Tasks:

When the task is leaded from the db they’re loaded into the vector of tasks which works as the list of all the added tasks. The tasks are categorized under two groups i.e. overdue or not overdue. The overdue tasks are displayed under the heading **Previous Dues** and rest under the heading **My Todos**.