# Weekly Homework Report #5

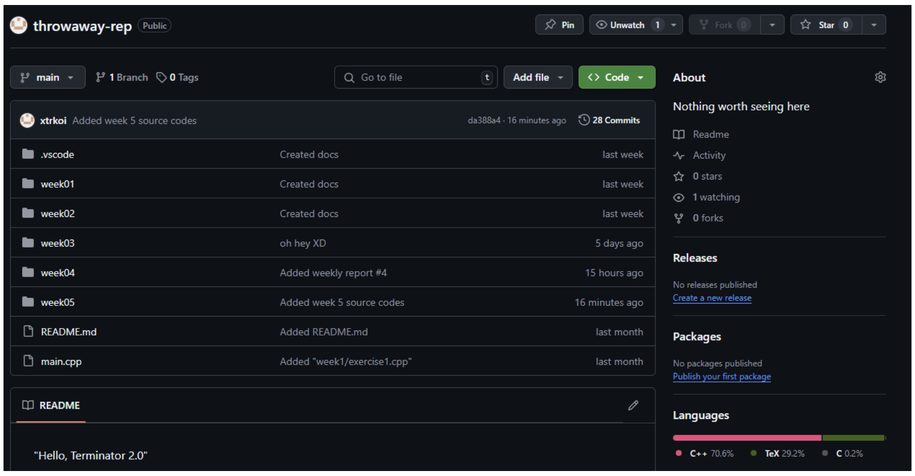
Tran Van Tan Khoi

April 16, 2025

## 1 Introduction

This report will discuss on the stack and queue data structures, notably their features and implementations.

Find all source codes and documentations of this week and previous weeks in this Github repository.



throwaway-rep Public

| | | |
|---|---|---|
| xtrkoi Added week 5 source codes | da388a4 · 16 minutes ago | 28 Commits |
| .vscode | Created docs | last week |
| week01 | Created docs | last week |
| week02 | Created docs | last week |
| week03 | oh hey XD | 5 days ago |
| week04 | Added weekly report #4 | 15 hours ago |
| week05 | Added week 5 source codes | 16 minutes ago |
| README.md | Added README.md | last month |
| main.cpp | Added "week1/exercise1.cpp" | last month |

README

"Hello, Terminator 2.0"

About

Nothing worth seeing here

Readme
Activity
0 stars
1 watching
0 forks

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

C++ 70.6% TeX 29.2% C 0.2%

## 2 Stack

Stack is a data structure that manages data (which will be referred to as objects) linearly, from top to bottom, and dynamically. A stack keeps objects in the order of entering the stack. Typically, the first object entering the stack situates at the bottom of the stack, and the last on the top, which will be referred to as the stack's top. When retrieving data from the stack, the latest data will be retrieved first, then working downward to the oldest data at the bottom. This is usually described as the Last-In-First-Out (LIFO) design pattern.

A stack has three basic functions: push an object to the top, peek at the top object of the stack, and pop (removing) the top object of the stack. A size determining function would also help in many cases. Each object in the stack does not have to follow any rule or have any certain property for the stack to work. However, modify the stack's internal structure can cause problems and is usually forbidden in implementations. If such a need for modifying the stack's internal arises then using a more advanced data structure is recommended.

The C++ STL std :: stack uses std :: deque and specifically, a double ended queue of chunks of memory, as the underlying container. However, the Linked List data structure does support adding, removing, and accessing the front (or back) object in constant time, which is suitable for being used as the underlying container for the stack in this project. Most linked list's functionalities aren't needed and we only need a method to insert a node at the front, removing the node at the front, accessing the front node, keep track of the number of nodes for constant time accesses; and a way to traverse the list for debugging purposes.

## 3 Queue

A queue behaves similarly to a stack, keeping objects of data in a line from left to right dynamically, with the front at one end and the back at the other. The order of the objects is also determined by the order of entering the queue, with the oldest object at front and the lastest at the back. The oldest data will be retrieved first in this case, following the First-In-First-Out (FIFO) design pattern.

A queue also has three basic functions: push (inserting) an object at the back, peek at object at the front, and pop (removing) the front object.

Linked list can also be used in the implementation of a queue.