Data Structures Recitation 1

Batuhan Can (canb18), Enes Erdoğan (erdogane16)

Outline

Environment setup:

- Installations: Docker, VS Code and extensions
- Template Github Repo
- Makefiles
- Valgrind

Coding Example: Expanding array

Installation: Docker and Visual Studio Code

- **Why do we need Docker?**
  - Docker is a tool to virtualize softwares in operating systems.
  - Allows us to work efficiently in different environments, and meet in a environment with same specifications.
- **Why do we need Visual Studio Code?**
  - It offers lots of cool extensions such as language support, and SSH tools.
- We will refer to a pre-prepared PDF file in order to install Docker and Visual Studio Code to our computers.
- We need to make some arrangements during the installation, **therefore be careful while following the instructions**.

Template Github Repository

- There is a provided GitHub repository, which will be our template for all homeworks.
- In order to download (clone) this repository, we need to enter the following command in our terminal.
  - `git clone https://github.com/ovatman/BLG223E-2024.git`
- Now, we are ready to open our project in Visual Studio Code.
- Visual Studio Code will ask us if we want to open the project in a container… We say yes!
- We need to run the file called `runme.sh` to create our project layout.

Makefile

Why we need Makefiles:

- As the project gets bigger, with many files, arguments and flags, compile and debug commands might get complicated.
- Even though you only change a single line from one file, the whole project is compiled from scratch.

Makefiles offer a straightforward solution for organizing code compilation. They are mostly used to break down the compilation process into separate steps. So, only the required part of the code is compiled and combined with the rest.

A makefile essentially contains a set variables and rules that are executed by the GNU Make build system. A rule generally looks like the following:

```
target: dependencies
    action
```

The target is the file we want to generate using the provided commands given the dependency files are available. Only if dependency files are changed, the target file will be created again. If dependency files are not available it will run the corresponding rule for the dependency.

Valgrind

“Valgrind is a programming tool for memory debugging, **memory leak detection**, and profiling.”

Memory leaks is a very important problem, and you should always check your code to make sure you don’t have them. In this course, **we will check your homeworks with valgrind to see if there is any memory leaks!**

First, compile your code then call:

```
> valgrind --leak-check=full ./bin/main <arg1> <arg2> …
```

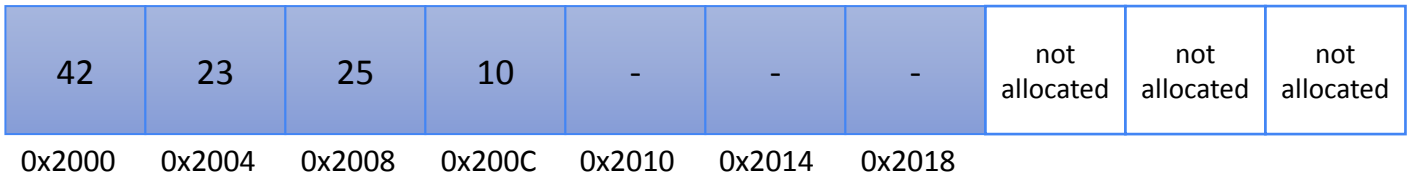
You can check for other useful flags: `--show-leak-kinds=all --track-origins=yes`

What is an expanding array?

- An array that grows when there is not enough space allocated, yet we want to add a new item.
- This array also can shrink when the allocated space is much larger than the used memory.
- A simpler version (for lazy people) : Allocate fixed-size memory, which is the capacity of the array. Then manage the array. **Are there any problems with that version?**

Terms in Expanding Array

- **Capacity** : Maximum # of items an expanding array can store
- **Size** : # of items that is stored currently in the array
- **Expand** : Increasing the capacity of the array
- **Shrink** : Decreasing the capacity of the array
- **Delta** : A constant factor indicates the volume of expansion and shrinkage



Features in Expanding Arrays

- «expand» : increases the capacity of the array
- «shrink» : decreases the capacity of the array
- «append» : adds a new item to the last index
- «insert» : adds a new item to a specified index
- «replace» : alters the value of the item stored in the specified index
- «get» : fetches the item stored in the specified index
- «remove» : deletes the item stored in the specified index
- DON’T FORGET «init» and «destroy»!