Packet Sniffing and Spoofing

**הנחיות להגשת המטלה:**

את מטלה זו יש להגיש בזוגות כקובץ ZIP עם מספרי ת"ז של הסטודנטים/ות. לתיבת ההגשה במודל. הגשות באיחור יתאפשרו עד 4 ימים כאשר לכל יום איחור ירדו 5 נקודות. שימו לב, יש להגיש גם קובץ pdf המכיל צילומי מסך (בכל מקום שאתם עושים משהו תצלמו מסך ותסבירו איך הגעתם למסקנה) בכל מקום שאתם משדרים מידע צריך לצרף קובץ הקלטה. אין להגיש צילומי מסך של מרוכזים בקובץ אחד עם הסברים, קובץ ללא הסברים או צילומי מסך בלבד לא יבדקו

1. את המטלה יש להגיש עד התאריך המצוין בתיבת ההגשה
2. כל הקבצי המטלה (קוד, פלט תעבורה, הסבר) כולל הסברים שלכם והקלטות Wireshark דחוסים לקובץ ZIP ששמו הוא מס' ת.ז. של המגישים עם קו תחתון ביניהם ID\_ID.
3. הגשה בזוגות זהו הסטנדרט וזו חובה. לאישורים חריגים שלחו מייל למרצה שלכם ותכתבו את אחראי המתרגלים, אלמוג שור.
4. מותר לכם להשתמש בכל החומר שנמצא במודל כולל קוד בתרגולים. חומרים אחרים אין אפשרות. כמובן שאפשר להיעזר באינטרנט להבנה של תהליכים וקוד אבל בשום פנים ואופן לא להעתיק קוד. כל שימוש בכלים כמו copilot or chatgpt יפסלו
5. אין איחורים ללא אישור מיוחד של רכז הקורס (עמית), איחור ללא אישור יגרור אפס אוטומטי.
6. הגשת העבודות תתבצע דרך מערכת ה Moodle של הקורס (לא דרך האימייל(.
7. יש להקפיד על כללי עיצוב הקוד שנלמדו בתואר (נא להקפיד על פלט ברור, הערות קוד במידה ושמות משתנים בעלי משמעות). קוד רץ בלבד יכול לקבל לכל היותר ציון 60, שאר 40 הנקודות זה הסברים שלכם, ידע, קוד קריא וכו'.
8. ניתן להגיש תרגילים למערכת מספר בלתי מוגבל של פעמים כאשר כל הגשה דורסת את הקודמת.
9. העבודה הינה אישית של הזוג ואסור לקבל עזרה מאנשים מחוץ לאוניברסיטה או בתוכה לה. אנשים המתקשים ורוצים עזרה יכולים לפנות לצוות הקורס בשעות הקבלה או להעלות שאלה לאתר הקורס. אם עבדתם עם עוד זוג על בעיות מסוימות כתבו זאת בקובץ ה-pdf
10. אסור להעביר קטעי קוד בין סטודנטים, להעלות פתרונות או חלקי פתרונות לאתרים ברשת האינטרנט, פורומים או בקבוצות תקשורת שונות.
11. סטודנטים שיעתיקו פתרון, יקבלו 0 בכל המטלות בקורס ונעלה דיווח לוועדת המשמעת המוסדית.
12. קבצי הקוד המטלה נדרשים להתקמפל ולרוץ על מערכת הפעלה Ubuntu 22.04LTS

For deep dive and to succeed on this Ex, it is recommending start with learning the materials in Appendix A and the link: <https://www.tcpdump.org/pcap.html>.

The Ex should contain those files:

sniffer.py, spoofer.py, snoffer.py, traceroute.py, pass\_sniffer.py, PDF of explanations, Wireshark pcap files.

Ex body:

Many tools can be used to do sniffing and spoofing, but most of them only provide fixed functionalities. Scapy is different: it can be used not only as a tool, but also as a building block to construct other sniffing and spoofing tools, i.e., we can integrate the Scapy functionalities into our own program. In this set of tasks, we will use Scapy for each task.

\*You don’t have to use Scapy package, feel free to use any other methods there is for succeeding this Ex

**Task A – 15%**

Write your own sniffer for capturing packets. Your sniffer should be able to sniff the following protocols:

* TCP
* UDP
* ICMP
* IGMP
* RAW (other - default)

**Run your Ex 2 from Networks Course**. Use your sniffer to sniff the TCP packets and write them out into a txt file named after your IDs. The format of each packet should be { source\_ip: <input>, dest\_ip: <input>, source\_port: <input>, dest\_port: <input>, timestamp: <input>, total\_length: <input>, cache\_flag: <input>, steps\_flag: <input>, type\_flag: <input>, status\_code: <input>, cache\_control: <input>, data: <input> }

The data output may be unreadable in ASCII form so write the output as hexadecimal.

In the paragraph following your Explanations of this task, please answer the following question:

**Question**

* Why do you need the root privilege to run a sniffer program? Where does the program fail if it is executed without the root privilege?

Submission – sniffer.py and relative pcap file

**Write detailed research in your PDF about Your sniffer abilities and limitations**

**Task B – 15%**

Write a spoofer **for spoofing packets**. Your spoofer should be able to spoof packets by using the following protocols:

* ICMP
* UDP
* TCP (Bonus 3 points to the assignment)

The spoofer should fake the sender’s IP and has a valid response. Your code should be able to spoof other protocols with small changes.

In the paragraph following your Explanations of this task, please answer the questions:

**Question 1.**

Can you set the IP packet length field to an arbitrary value, regardless of how big the actual packet is?

**Question 2**.

Using the raw socket programming, do you have to calculate the checksum for the IP header?

**Write detailed research in your PDF about Your spoofer abilities and limitations**

**Task C – 10%**

The objective of this task is to use Scapy to estimate the distance, in terms of number of routers, between your VM and a selected destination. This is basically what is implemented by the traceroute tool. In this task, we will write our own tool. The idea is quite straightforward: just send an packet (any type) to the destination, with its Time-To-Live (TTL) field set to 1 first. This packet will be dropped by the first router, which will send us an ICMP error message, telling us that the time-to-live has exceeded. That is how we get the IP address of the first router. We then increase our TTL field to 2, send out another packet, and get the IP address of the second router. We will repeat this procedure until our packet finally reach the destination. It should be noted that this experiment only gets an estimated result, because in theory, not all these packets take the same route (but in practice, they may within a short period of time).

\* You should write your tool to perform the entire procedure automatically.

**Task D – 50%**

@Before -read Appendix B for composing the LAN of docker containers.

In this task, you will combine the sniffing and spoofing techniques to implement the following sniff-and-then-spoof program. You need two machines on the same LAN. From machine A, you ping an IP X. This will generate an ICMP echo request packet. If X is alive, the ping program will receive an echo reply, and print out the response. Your sniff-and-then-spoof program runs on the attacker machine, which monitors the LAN through packet sniffing. Whenever it sees an ICMP echo request, regardless of what the target IP address is, your program should immediately send out an echo reply using the packet spoofing technique.

Please follow those steps:

1. Compose the Docker Containers that are in the Zip from the Moodle (The TAs showed you how to do so in Tirgul 09)
   * + Note: If it doesn’t work properly you can create a LAN by composing few Machines on your VirtualBox
2. Run your Ex\_4 codes from Networks Course in this new LAN. Use your sniffer from Task A to sniff the ICMP packets from the seed-attacker. Peel off the needed data for spoofing the packets, and return a packet which made by you.
   1. First run – send a ping from Host A to Host B
   2. Second run – send a ping from Host A to a WAN IP (e.g., google DNS – 8.8.8.8)
   3. Third run – send a ping from Host A to a fake IP.
   4. Fourth run – send a ping from Host A to fake IP in your LAN
3. **Write detailed research in your PDF about Task C**

**Task E - 10%**

Please show how you can use your sniffer program to capture the password when somebody is using telnet on the network that you are monitoring. You may need to modify your sniffer code to print out the data part of a captured TCP packet (telnet uses TCP). It is acceptable if you print out the entire data part, and then manually mark where the password (or part of it) is.

Submission – sniffing\_passwords.py and relative pcap file

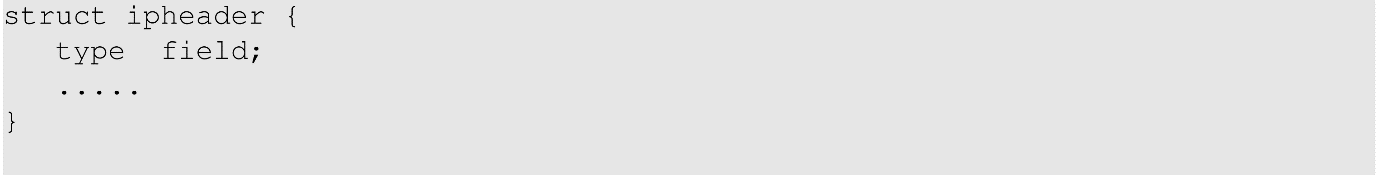
**Write detailed research in your PDF about Your sniffing\_password.py output**

**Note:** The following Appendixes are by [Seed lab](https://seedsecuritylabs.org/Labs_16.04/Networking/Sniffing_Spoofing/)

Appendix A:

**5.1 Filling in Data in Raw Packets**

When you send out a packet using raw sockets, you basically construct the packet inside a buffer, so when you need to send it out, you simply give the operating system the buffer and the size of the packet. Working directly on the buffer is not easy, so a common way is to typecast the buffer (or part of the buffer) into structures, such as IP header structure, so you can refer to the elements of the buffer using the fields of those structures. You can define the IP, ICMP, TCP, UDP and other header structures in your program. The following example show how you can construct an UDP packet:



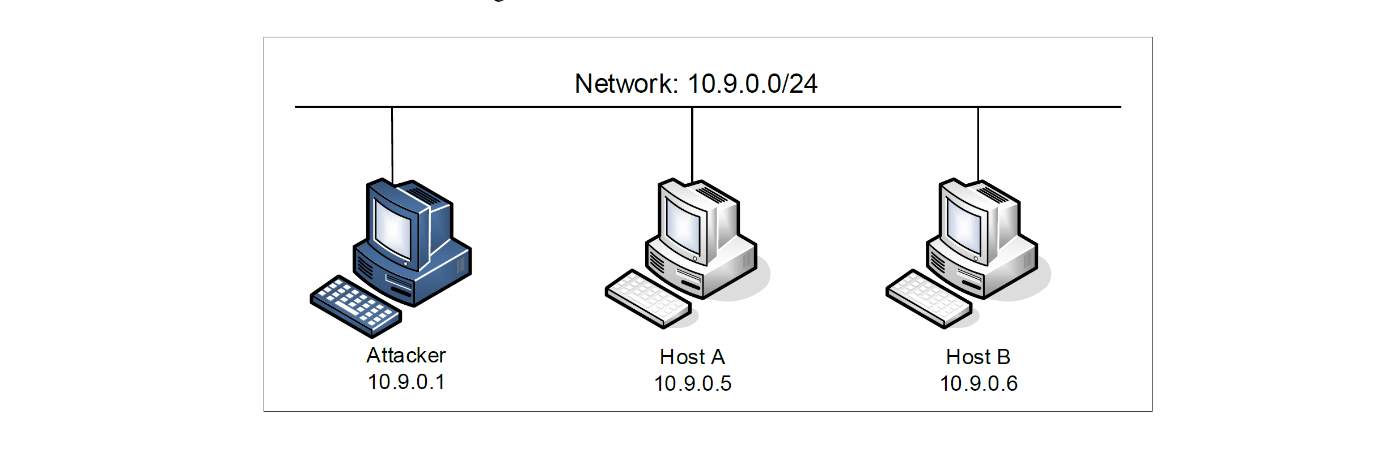
Text, letter

Description automatically generated

Appendix B:

2 Environment Setup using Container

In this Ex, we will use three machines that are connected to the same LAN. We will use three containers. We will do all the attacks on the attacker container, while using the other containers as the user machines. Diagram of the LAN:



**2.1 Container Setup and Commands**

Please download the Labsetup.zip file to your VM from the Moodle, unzip it, enter the Labsetup

folder, and use the docker-compose.yml file to set up the lab environment. Detailed explanation of the content in this file and all the involved Docker file can be found from the user manual.

In the following, we list some of the commonly used commands related to Docker and Compose.

All the containers will be running in the background. To run commands on a container, we often need to get a shell on that container. We first need to use the "docker ps" command to find out the ID of the container, and then use "docker exec" to start a shell on that container.

/\*Note: If a docker command requires a container ID, you do not need to

type the entire ID string. Typing the first few characters will

be sufficient, as long as they are unique among all the containers.

\*/

**2.2 About the Attacker Container**

If you look at the Docker Compose file, you will see that the attacker container is configured differently from the other containers. Here are the differences:

* Shared folder. When we use the attacker container to launch attacks, we need to put the attacking code inside the attacker container. Code editing is more convenient inside the VM than in containers, because we can use our favourite editors. In order for the VM and container to share files, we have created a shared folder between the VM and the container using the Docker volumes. If you look at the Docker Compose file, you will find out that we have added the following entry to some of the containers. It indicates mounting the ./volumes folder on the host machine (i.e., the VM) to the /volumes folder inside the container. We will write our code in the ./volumes folder (on the VM), so they can be used inside the containers.  
  
* Host mode. In this lab, the attacker needs to be able to sniff packets, but running sniffer programs inside a container has problems, because a container is effectively attached to a virtual switch, so it can only see its own traffic, and it is never going to see the packets among other containers. To solve this problem, we use the host mode for the attacker container. This allows the attacker container to see all the traffics. The following entry used on the attacker container:  
    
  When a container is in the host mode, it sees all the host’s network interfaces, and it even has the same IP addresses as the host. Basically, it is put in the same network namespace as the host VM. However, the container is still a separate machine, because its other namespaces are still different from the host.

#include <iostream>

#include <vector>

#include <algorithm>

#include <stdexcept>

#include <iterator>

#include <cmath>

#define DOCTEST\_CONFIG\_IMPLEMENT\_WITH\_MAIN

#include "doctest.h"

*class* *MagicalContainer* {

*public*:

*using* value\_type = *int*;

*void* *add\_element*(*int* element) {

elements.*push\_back*(element);

std::*sort*(elements.*begin*(), elements.*end*());

}

*void* *remove\_element*(*int* element) {

*auto* it = std::*find*(elements.*begin*(), elements.*end*(), element);

*if* (it != elements.*end*()) {

elements.*erase*(it);

}

}

*size\_t* *size*() *const* {

*return* elements.*size*();

}

*const* std::vector<*int*>& *get\_elements*() *const* {

*return* elements;

}

*private*:

std::vector<*int*> elements;

};

*template* <*typename* Predicate>

*class* *CustomIterator* : *public* std::iterator<std::input\_iterator\_tag, *int*> {

*public*:

*CustomIterator*(*const* MagicalContainer& container, Predicate pred)

: *container*(container), *pred*(pred), *index*(0) {

*find\_next\_valid*();

}

*CustomIterator*(*const* CustomIterator& other)

: *container*(other.container), *pred*(other.pred), *index*(other.index) {}

~*CustomIterator*() {}

CustomIterator& *operator*=(*const* CustomIterator& other) {

*if* (*this* != &other) {

container = other.container;

pred = other.pred;

index = other.index;

}

*return* \**this*;

}

*bool* *operator*==(*const* CustomIterator& other) *const* {

*return* &container == &other.container && index == other.index;

}

*bool* *operator*!=(*const* CustomIterator& other) *const* {

*return* !(\**this* == other);

}

*bool* *operator*>(*const* CustomIterator& other) *const* {

*if* (&container != &other.container || *typeid*(pred) != *typeid*(other.pred)) {

*throw* std::*runtime\_error*("Comparing incompatible iterators");

}

*return* index > other.index;

}

*bool* *operator*<(*const* CustomIterator& other) *const* {

*if* (&container != &other.container || *typeid*(pred) != *typeid*(other.pred)) {

*throw* std::*runtime\_error*("Comparing incompatible iterators");

}

*return* index < other.index;

}

*int* *operator*\*() *const* {

*return* container.*get\_elements*()[index];

}

CustomIterator& *operator*++() {

++index;

*find\_next\_valid*();

*return* \**this*;

}

*private*:

*void* *find\_next\_valid*() {

*while* (index < container.*size*() && !*pred*(container.*get\_elements*()[index])) {

++index;

}

}

*const* MagicalContainer& container;

Predicate pred;

*size\_t* index;

};

*struct* *AscendingPredicate* {

*bool* *operator*()(*int*) *const* { *return* true; }

};

*struct* *SideCrossPredicate* {

*bool* *operator*()(*int* element) *const* {

*return* element == start || element == end;

}

*int* start = 1;

*int* end = 14;

};

*struct* *PrimePredicate* {

*bool* *operator*()(*int* element) *const* {

*if* (element < 2) *return* false;

*for* (*int* i = 2; i <= std::*sqrt*(element); ++i) {

*if* (element % i == 0) *return* false;

}

*return* true;

}

};

*using* AscendingIterator = CustomIterator<AscendingPredicate>;

*using* SideCrossIterator = CustomIterator<SideCrossPredicate>;

*using* PrimeIterator = CustomIterator<PrimePredicate>;

*TEST\_CASE*("MagicalContainer and custom iterators") {

MagicalContainer container;

container.*add\_element*(1);

container.*add\_element*(2);

container.*add\_element*(4);

container.*add\_element*(5);

container.*add\_element*(14);

*SUBCASE*("AscendingIterator") {

AscendingIterator *it*(container, AscendingPredicate());

*CHECK*(\*it == 1);

++it;

*CHECK*(\*it == 2);

++it;

*CHECK*(\*it == 4);

++it;

*CHECK*(\*it == 5);

++it;

*CHECK*(\*it == 14);

}

*SUBCASE*("SideCrossIterator") {

SideCrossIterator *it*(container, SideCrossPredicate());

*CHECK*(\*it == 1);

++it;

*CHECK*(\*it == 14);

++it;

*CHECK*(\*it == 2);

++it;

*CHECK*(\*it == 5);

++it;

*CHECK*(\*it == 4);

}

*SUBCASE*("PrimeIterator") {

PrimeIterator *it*(container, PrimePredicate());

*CHECK*(\*it == 2);

++it;

*CHECK*(\*it == 5);

}

*SUBCASE*("Iterator comparisons") {

AscendingIterator *it1*(container, AscendingPredicate());

AscendingIterator *it2*(container, AscendingPredicate());

++it1;

*CHECK*(it1 > it2);

*CHECK*(it2 < it1);

*CHECK*(it1 != it2);

*CHECK*(it2 == it2);

}

}

This implementation uses a template class CustomIterator with a predicate to determine the traversal order. The three custom iterators are then defined as specializations of this template class with their respective predicates. The test cases cover the basic functionality of the iterators and the MagicalContainer class.

GPT-4

Here's a C++ implementation of the MagicalContainer and the three custom iterators. I've also included some doctest test cases to ensure the functionality is working as expected.

cpp

Copy

#include <iostream>

#include <vector>

#include <algorithm>

#include <stdexcept>

#include <iterator>

#include <cmath>

#define DOCTEST\_CONFIG\_IMPLEMENT\_WITH\_MAIN

#include "doctest.h"

*class* *MagicalContainer* {

*public*:

*using* value\_type = *int*;

*void* *add\_element*(*int* element) {

elements.*push\_back*(element);

std::*sort*(elements.*begin*(), elements.*end*());

}

*void* *remove\_element*(*int* element) {

*auto* it = std::*find*(elements.*begin*(), elements.*end*(), element);

*if* (it != elements.*end*()) {

elements.*erase*(it);

}

}

*size\_t* *size*() *const* {

*return* elements.*size*();

}

*const* std::vector<*int*>& *get\_elements*() *const* {

*return* elements;

}

*private*:

std::vector<*int*> elements;

};

*template* <*typename* Predicate>

*class* *CustomIterator* : *public* std::iterator<std::input\_iterator\_tag, *int*> {

*public*:

*CustomIterator*(*const* MagicalContainer& container, Predicate pred)

: *container*(container), *pred*(pred), *index*(0) {

*find\_next\_valid*();

}

*CustomIterator*(*const* CustomIterator& other)

: *container*(other.container), *pred*(other.pred), *index*(other.index) {}

~*CustomIterator*() {}

CustomIterator& *operator*=(*const* CustomIterator& other) {

*if* (*this* != &other) {

container = other.container;

pred = other.pred;

index = other.index;

}

*return* \**this*;

}

*bool* *operator*==(*const* CustomIterator& other) *const* {

*return* &container == &other.container && index == other.index;

}

*bool* *operator*!=(*const* CustomIterator& other) *const* {

*return* !(\**this* == other);

}

*bool* *operator*>(*const* CustomIterator& other) *const* {

*if* (&container != &other.container || *typeid*(pred) != *typeid*(other.pred)) {

*throw* std::*runtime\_error*("Comparing incompatible iterators");

}

*return* index > other.index;

}

*bool* *operator*<(*const* CustomIterator& other) *const* {

*if* (&container != &other.container || *typeid*(pred) != *typeid*(other.pred)) {

*throw* std::*runtime\_error*("Comparing incompatible iterators");

}

*return* index < other.index;

}

*int* *operator*\*() *const* {

*return* container.*get\_elements*()[index];

}

CustomIterator& *operator*++() {

++index;

*find\_next\_valid*();

*return* \**this*;

}

*private*:

*void* *find\_next\_valid*() {

*while* (index < container.*size*() && !*pred*(container.*get\_elements*()[index])) {

++index;

}

}

*const* MagicalContainer& container;

Predicate pred;

*size\_t* index;

};

*struct* *AscendingPredicate* {

*bool* *operator*()(*int*) *const* { *return* true; }

};

*struct* *SideCrossPredicate* {

*bool* *operator*()(*int* element) *const* {

*return* element == start || element == end;

}

*int* start = 1;

*int* end = 14;

};

*struct* *PrimePredicate* {

*bool* *operator*()(*int* element) *const* {

*if* (element < 2) *return* false;

*for* (*int* i = 2; i <= std::*sqrt*(element); ++i) {

*if* (element % i == 0) *return* false;

}

*return* true;

}

};

*using* AscendingIterator = CustomIterator<AscendingPredicate>;

*using* SideCrossIterator = CustomIterator<SideCrossPredicate>;

*using* PrimeIterator = CustomIterator<PrimePredicate>;

*TEST\_CASE*("MagicalContainer and custom iterators") {

MagicalContainer container;

container.*add\_element*(1);

container.*add\_element*(2);

container.*add\_element*(4);

container.*add\_element*(5);

container.*add\_element*(14);

*SUBCASE*("AscendingIterator") {

AscendingIterator *it*(container, AscendingPredicate());

*CHECK*(\*it == 1);

++it;

*CHECK*(\*it == 2);

++it;

*CHECK*(\*it == 4);

++it;

*CHECK*(\*it == 5);

++it;

*CHECK*(\*it == 14);

}

*SUBCASE*("SideCrossIterator") {

SideCrossIterator *it*(container, SideCrossPredicate());

*CHECK*(\*it == 1);

++it;

*CHECK*(\*it == 14);

++it;

*CHECK*(\*it == 2);

++it;

*CHECK*(\*it == 5);

++it;

*CHECK*(\*it == 4);

}

*SUBCASE*("PrimeIterator") {

PrimeIterator *it*(container, PrimePredicate());

*CHECK*(\*it == 2);

++it;

*CHECK*(\*it == 5);

}

*SUBCASE*("Iterator comparisons") {

AscendingIterator *it1*(container, AscendingPredicate());

AscendingIterator *it2*(container, AscendingPredicate());

++it1;

*CHECK*(it1 > it2);

*CHECK*(it2 < it1);

*CHECK*(it1 != it2);

*CHECK*(it2 == it2);

}

}

This implementation uses a template class CustomIterator with a predicate to determine the traversal order. The three custom iterators are then defined as specializations of this template class with their respective predicates. The test cases cover the basic functionality of the iterators and the MagicalContainer class.

GPT-4

To ensure the proper functionality of the MagicalContainer and its iterators, it's essential to write tests for edge cases. Here are some test cases using the doctest framework:

cpp

Copy

#define DOCTEST\_CONFIG\_IMPLEMENT\_WITH\_MAIN

#include "doctest.h"

#include "MagicalContainer.h"

*TEST\_CASE*("Empty container") {

MagicalContainer container;

*CHECK*(container.*size*() == 0);

AscendingIterator asc\_it = container.*begin*(Ascending);

*CHECK*(asc\_it == container.*end*(Ascending));

SideCrossIterator side\_it = container.*begin*(SideCross);

*CHECK*(side\_it == container.*end*(SideCross));

PrimeIterator prime\_it = container.*begin*(Prime);

*CHECK*(prime\_it == container.*end*(Prime));

}

*TEST\_CASE*("Adding and removing elements") {

MagicalContainer container;

container.*add*(1);

container.*add*(2);

container.*add*(3);

container.*add*(4);

container.*add*(5);

*CHECK*(container.*size*() == 5);

container.*remove*(3);

*CHECK*(container.*size*() == 4);

container.*remove*(1);

*CHECK*(container.*size*() == 3);

}

*TEST\_CASE*("Iterators after adding elements") {

MagicalContainer container;

container.*add*(1);

container.*add*(2);

container.*add*(4);

container.*add*(5);

container.*add*(14);

AscendingIterator asc\_it = container.*begin*(Ascending);

*CHECK*(\*asc\_it == 1);

++asc\_it;

*CHECK*(\*asc\_it == 2);

SideCrossIterator side\_it = container.*begin*(SideCross);

*CHECK*(\*side\_it == 1);

++side\_it;

*CHECK*(\*side\_it == 14);

PrimeIterator prime\_it = container.*begin*(Prime);

*CHECK*(\*prime\_it == 2);

++prime\_it;

*CHECK*(\*prime\_it == 5);

}

*TEST\_CASE*("Iterators after removing elements") {

MagicalContainer container;

container.*add*(1);

container.*add*(2);

container.*add*(4);

container.*add*(5);

container.*add*(14);

container.*remove*(1);

container.*remove*(14);

AscendingIterator asc\_it = container.*begin*(Ascending);

*CHECK*(\*asc\_it == 2);

++asc\_it;

*CHECK*(\*asc\_it == 4);

SideCrossIterator side\_it = container.*begin*(SideCross);

*CHECK*(\*side\_it == 2);

++side\_it;

*CHECK*(\*side\_it == 5);

PrimeIterator prime\_it = container.*begin*(Prime);

*CHECK*(\*prime\_it == 2);

++prime\_it;

*CHECK*(\*prime\_it == 5);

}

*TEST\_CASE*("Invalid iterator operations") {

MagicalContainer container;

container.*add*(1);

container.*add*(2);

container.*add*(4);

container.*add*(5);

container.*add*(14);

AscendingIterator asc\_it1 = container.*begin*(Ascending);

AscendingIterator asc\_it2 = container.*begin*(Ascending);

++asc\_it2;

*CHECK\_THROWS\_AS*(asc\_it1 > asc\_it2, std::invalid\_argument);

*CHECK\_THROWS\_AS*(asc\_it1 < asc\_it2, std::invalid\_argument);

SideCrossIterator side\_it = container.*begin*(SideCross);

*CHECK\_THROWS\_AS*(asc\_it1 > side\_it, std::invalid\_argument);

*CHECK\_THROWS\_AS*(asc\_it1 < side\_it, std::invalid\_argument);

PrimeIterator prime\_it = container.*begin*(Prime);

*CHECK\_THROWS\_AS*(asc\_it1 > prime\_it, std::invalid\_argument);

*CHECK\_THROWS\_AS*(asc\_it1 < prime\_it, std::invalid\_argument);

}

These tests cover various edge cases, such as empty containers, adding and removing elements, and invalid iterator operations. Make sure to include the doctest header and link the doctest library to your project to run these tests.

GPT-4

To ensure the proper functionality of the MagicalContainer and its iterators, it's essential to write tests for edge cases. Here are some test cases using the doctest framework:

cpp

Copy

#define DOCTEST\_CONFIG\_IMPLEMENT\_WITH\_MAIN

#include "doctest.h"

#include "MagicalContainer.h"

*TEST\_CASE*("Empty container") {

MagicalContainer container;

*CHECK*(container.*size*() == 0);

AscendingIterator asc\_it = container.*begin*(Ascending);

*CHECK*(asc\_it == container.*end*(Ascending));

SideCrossIterator side\_it = container.*begin*(SideCross);

*CHECK*(side\_it == container.*end*(SideCross));

PrimeIterator prime\_it = container.*begin*(Prime);

*CHECK*(prime\_it == container.*end*(Prime));

}

*TEST\_CASE*("Adding and removing elements") {

MagicalContainer container;

container.*add*(1);

container.*add*(2);

container.*add*(3);

container.*add*(4);

container.*add*(5);

*CHECK*(container.*size*() == 5);

container.*remove*(3);

*CHECK*(container.*size*() == 4);

container.*remove*(1);

*CHECK*(container.*size*() == 3);

}

*TEST\_CASE*("Iterators after adding elements") {

MagicalContainer container;

container.*add*(1);

container.*add*(2);

container.*add*(4);

container.*add*(5);

container.*add*(14);

AscendingIterator asc\_it = container.*begin*(Ascending);

*CHECK*(\*asc\_it == 1);

++asc\_it;

*CHECK*(\*asc\_it == 2);

SideCrossIterator side\_it = container.*begin*(SideCross);

*CHECK*(\*side\_it == 1);

++side\_it;

*CHECK*(\*side\_it == 14);

PrimeIterator prime\_it = container.*begin*(Prime);

*CHECK*(\*prime\_it == 2);

++prime\_it;

*CHECK*(\*prime\_it == 5);

}

*TEST\_CASE*("Iterators after removing elements") {

MagicalContainer container;

container.*add*(1);

container.*add*(2);

container.*add*(4);

container.*add*(5);

container.*add*(14);

container.*remove*(1);

container.*remove*(14);

AscendingIterator asc\_it = container.*begin*(Ascending);

*CHECK*(\*asc\_it == 2);

++asc\_it;

*CHECK*(\*asc\_it == 4);

SideCrossIterator side\_it = container.*begin*(SideCross);

*CHECK*(\*side\_it == 2);

++side\_it;

*CHECK*(\*side\_it == 5);

PrimeIterator prime\_it = container.*begin*(Prime);

*CHECK*(\*prime\_it == 2);

++prime\_it;

*CHECK*(\*prime\_it == 5);

}

*TEST\_CASE*("Invalid iterator operations") {

MagicalContainer container;

container.*add*(1);

container.*add*(2);

container.*add*(4);

container.*add*(5);

container.*add*(14);

AscendingIterator asc\_it1 = container.*begin*(Ascending);

AscendingIterator asc\_it2 = container.*begin*(Ascending);

++asc\_it2;

*CHECK\_THROWS\_AS*(asc\_it1 > asc\_it2, std::invalid\_argument);

*CHECK\_THROWS\_AS*(asc\_it1 < asc\_it2, std::invalid\_argument);

SideCrossIterator side\_it = container.*begin*(SideCross);

*CHECK\_THROWS\_AS*(asc\_it1 > side\_it, std::invalid\_argument);

*CHECK\_THROWS\_AS*(asc\_it1 < side\_it, std::invalid\_argument);

PrimeIterator prime\_it = container.*begin*(Prime);

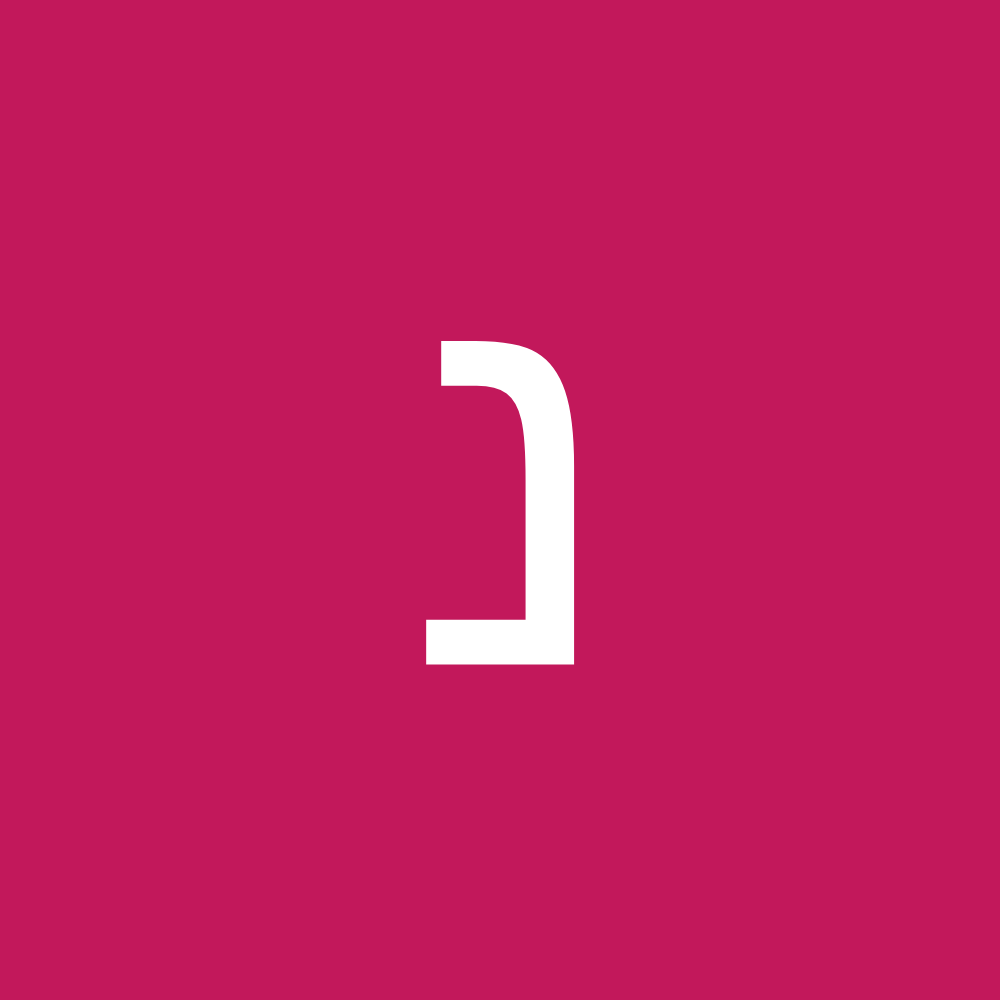
*CHECK\_THROWS\_AS*(asc\_it1 > prime\_it, std::invalid\_argument);

*CHECK\_THROWS\_AS*(asc\_it1 < prime\_it, std::invalid\_argument);

}

These tests cover various edge cases, such as empty containers, adding and removing elements, and invalid iterator operations. Make sure to include the doctest header and link the doctest library to your project to run these tests.

GPT-4



please provide me more test with many edge cases that you can

3 days ago

Here are some additional test cases to cover more edge cases:

cpp

Copy

*TEST\_CASE*("Adding duplicate elements") {

MagicalContainer container;

container.*add\_element*(1);

container.*add\_element*(1);

container.*add\_element*(2);

container.*add\_element*(2);

*CHECK*(container.*size*() == 2);

AscendingIterator asc\_it = container.*begin*(Ascending);

*CHECK*(\*asc\_it == 1);

++asc\_it;

*CHECK*(\*asc\_it == 2);

}

*TEST\_CASE*("Removing non-existent elements") {

MagicalContainer container;

container.*add\_element*(1);

container.*add\_element*(2);

container.*add\_element*(3);

container.*remove\_element*(4);

container.*remove\_element*(5);

*CHECK*(container.*size*() == 3);

}

*TEST\_CASE*("Large container") {

MagicalContainer container;

*for* (*int* i = 1; i <= 1000; ++i) {

container.*add\_element*(i);

}

*CHECK*(container.*size*() == 1000);

AscendingIterator asc\_it = container.*begin*(Ascending);

*for* (*int* i = 1; i <= 1000; ++i) {

*CHECK*(\*asc\_it == i);

++asc\_it;

}

PrimeIterator prime\_it = container.*begin*(Prime);

*int* prime\_count = 0;

*while* (prime\_it != container.*end*(Prime)) {

++prime\_count;

++prime\_it;

}

*CHECK*(prime\_count == 168); *// There are 168 prime numbers between 1 and 1000*

}

*TEST\_CASE*("Iterators at the end of the container") {

MagicalContainer container;

container.*add\_element*(1);

container.*add\_element*(2);

container.*add\_element*(4);

container.*add\_element*(5);

container.*add\_element*(14);

AscendingIterator asc\_it = container.*begin*(Ascending);

*for* (*int* i = 0; i < 5; ++i) {

++asc\_it;

}

*CHECK*(asc\_it == container.*end*(Ascending));

SideCrossIterator side\_it = container.*begin*(SideCross);

*for* (*int* i = 0; i < 2; ++i) {

++side\_it;

}

*CHECK*(side\_it == container.*end*(SideCross));

PrimeIterator prime\_it = container.*begin*(Prime);

*for* (*int* i = 0; i < 2; ++i) {

++prime\_it;

}

*CHECK*(prime\_it == container.*end*(Prime));

}

*TEST\_CASE*("Iterating beyond the end of the container") {

MagicalContainer container;

container.*add\_element*(1);

container.*add\_element*(2);

container.*add\_element*(4);

container.*add\_element*(5);

container.*add\_element*(14);

AscendingIterator asc\_it = container.*begin*(Ascending);

*for* (*int* i = 0; i < 6; ++i) {

++asc\_it;

}

*CHECK\_THROWS\_AS*(\*asc\_it, std::out\_of\_range);

}

These test cases cover scenarios such as adding duplicate elements, removing non-existent elements, handling large containers, iterators at the end of the container, and iterating beyond the end of the container. Make sure to include these tests in your test suite to ensure the proper functionality of the MagicalContainer and its iterators.

GPT-4

Here are some additional test cases to cover more edge cases:

cpp

Copy

*TEST\_CASE*("Adding duplicate elements") {

MagicalContainer container;

container.*add\_element*(1);

container.*add\_element*(1);

container.*add\_element*(2);

container.*add\_element*(2);

*CHECK*(container.*size*() == 2);

AscendingIterator asc\_it = container.*begin*(Ascending);

*CHECK*(\*asc\_it == 1);

++asc\_it;

*CHECK*(\*asc\_it == 2);

}

*TEST\_CASE*("Removing non-existent elements") {

MagicalContainer container;

container.*add\_element*(1);

container.*add\_element*(2);

container.*add\_element*(3);

container.*remove\_element*(4);

container.*remove\_element*(5);

*CHECK*(container.*size*() == 3);

}

*TEST\_CASE*("Large container") {

MagicalContainer container;

*for* (*int* i = 1; i <= 1000; ++i) {

container.*add\_element*(i);

}

*CHECK*(container.*size*() == 1000);

AscendingIterator asc\_it = container.*begin*(Ascending);

*for* (*int* i = 1; i <= 1000; ++i) {

*CHECK*(\*asc\_it == i);

++asc\_it;

}

PrimeIterator prime\_it = container.*begin*(Prime);

*int* prime\_count = 0;

*while* (prime\_it != container.*end*(Prime)) {

++prime\_count;

++prime\_it;

}

*CHECK*(prime\_count == 168); *// There are 168 prime numbers between 1 and 1000*

}

*TEST\_CASE*("Iterators at the end of the container") {

MagicalContainer container;

container.*add\_element*(1);

container.*add\_element*(2);

container.*add\_element*(4);

container.*add\_element*(5);

container.*add\_element*(14);

AscendingIterator asc\_it = container.*begin*(Ascending);

*for* (*int* i = 0; i < 5; ++i) {

++asc\_it;

}

*CHECK*(asc\_it == container.*end*(Ascending));

SideCrossIterator side\_it = container.*begin*(SideCross);

*for* (*int* i = 0; i < 2; ++i) {

++side\_it;

}

*CHECK*(side\_it == container.*end*(SideCross));

PrimeIterator prime\_it = container.*begin*(Prime);

*for* (*int* i = 0; i < 2; ++i) {

++prime\_it;

}

*CHECK*(prime\_it == container.*end*(Prime));

}

*TEST\_CASE*("Iterating beyond the end of the container") {

MagicalContainer container;

container.*add\_element*(1);

container.*add\_element*(2);

container.*add\_element*(4);

container.*add\_element*(5);

container.*add\_element*(14);

AscendingIterator asc\_it = container.*begin*(Ascending);

*for* (*int* i = 0; i < 6; ++i) {

++asc\_it;

}

*CHECK\_THROWS\_AS*(\*asc\_it, std::out\_of\_range);

}

These test cases cover scenarios such as adding duplicate elements, removing non-existent elements, handling large containers, iterators at the end of the container, and iterating beyond the end of the container. Make sure to include these tests in your test suite to ensure the proper functionality of the MagicalContainer and its iterators.