



ft\_irc

## Internet Relay Chat

*Summary: This project is about creating your own IRC server.  
You will use an actual IRC client to connect to your server and test it.  
The Internet is governed by solid standard protocols that allow connected computers to  
interact with each other.  
It's always beneficial to understand these protocols.*

*Version: 10.0*

# Contents

<b>I</b>	<b>Introduction</b>	<b>2</b>
<b>II</b>	<b>General rules</b>	<b>3</b>
<b>III</b>	<b>AI Instructions</b>	<b>4</b>
<b>IV</b>	<b>Mandatory Part</b>	<b>6</b>
IV.1	Requirements . . . . .	8
IV.2	For MacOS only . . . . .	9
IV.3	Test example . . . . .	9
<b>V</b>	<b>Readme Requirements</b>	<b>10</b>
<b>VI</b>	<b>Bonus part</b>	<b>11</b>
<b>VII</b>	<b>Submission and peer-evaluation</b>	<b>12</b>

# Chapter I

## Introduction

**Internet Relay Chat** or IRC is a text-based communication protocol on the Internet. It offers real-time messaging that can be either public or private. Users can exchange direct messages and join group channels.

IRC clients connect to IRC servers in order to join channels. IRC servers are connected together to form a network.

# Chapter II

## General rules

- Your program should not crash in any circumstances (even when it runs out of memory), and should not quit unexpectedly.  
If it happens, your project will be considered non-functional and your grade will be 0.
- You have to turn in a **Makefile** which will compile your source files. It must not perform unnecessary relinking.
- Your **Makefile** must at least contain the rules:  
`$(NAME)`, `all`, `clean`, `fclean` and `re`.
- Compile your code with `c++` using the flags `-Wall -Wextra -Werror`.
- Your code must comply with the **C++ 98 standard**. Then, it should still compile if you add the flag `-std=c++98`.
- Try to always code using C++ features when available (for example, choose `<cstring>` over `<string.h>`). You are allowed to use C functions, but always prefer their C++ versions if possible.
- Any external library and Boost libraries are forbidden.

# Chapter III

## AI Instructions

### ● Context

During your learning journey, AI can assist with many different tasks. Take the time to explore the various capabilities of AI tools and how they can support your work. However, always approach them with caution and critically assess the results. Whether it's code, documentation, ideas, or technical explanations, you can never be completely sure that your question was well-formed or that the generated content is accurate. Your peers are a valuable resource to help you avoid mistakes and blind spots.

### ● Main message

- 👉 Use AI to reduce repetitive or tedious tasks.
- 👉 Develop prompting skills — both coding and non-coding — that will benefit your future career.
- 👉 Learn how AI systems work to better anticipate and avoid common risks, biases, and ethical issues.
- 👉 Continue building both technical and power skills by working with your peers.
- 👉 Only use AI-generated content that you fully understand and can take responsibility for.

### ● Learner rules:

- You should take the time to explore AI tools and understand how they work, so you can use them ethically and reduce potential biases.
- You should reflect on your problem before prompting — this helps you write clearer, more detailed, and more relevant prompts using accurate vocabulary.
- You should develop the habit of systematically checking, reviewing, questioning, and testing anything generated by AI.
- You should always seek peer review — don't rely solely on your own validation.

## ● Phase outcomes:

- Develop both general-purpose and domain-specific prompting skills.
- Boost your productivity with effective use of AI tools.
- Continue strengthening computational thinking, problem-solving, adaptability, and collaboration.

## ● Comments and examples:

- You'll regularly encounter situations — exams, evaluations, and more — where you must demonstrate real understanding. Be prepared, keep building both your technical and interpersonal skills.
- Explaining your reasoning and debating with peers often reveals gaps in your understanding. Make peer learning a priority.
- AI tools often lack your specific context and tend to provide generic responses. Your peers, who share your environment, can offer more relevant and accurate insights.
- Where AI tends to generate the most likely answer, your peers can provide alternative perspectives and valuable nuance. Rely on them as a quality checkpoint.

### ✓ Good practice:

I ask AI: “How do I test a sorting function?” It gives me a few ideas. I try them out and review the results with a peer. We refine the approach together.

### ✗ Bad practice:

I ask AI to write a whole function, copy-paste it into my project. During peer-evaluation, I can't explain what it does or why. I lose credibility — and I fail my project.

### ✓ Good practice:

I use AI to help design a parser. Then I walk through the logic with a peer. We catch two bugs and rewrite it together — better, cleaner, and fully understood.

### ✗ Bad practice:

I let Copilot generate my code for a key part of my project. It compiles, but I can't explain how it handles pipes. During the evaluation, I fail to justify and I fail my project.

# Chapter IV

## Mandatory Part

Program Name	ircserv
Files to Submit	Makefile, *.{h, cpp}, *.hpp, *.ipp, an optional configuration file
Makefile	NAME, all, clean, fclean, re
Arguments	port: The listening port password: The connection password
External Function	Everything in C++ 98. socket, close, setsockopt, getsockname, getprotobyname, gethostbyname, getaddrinfo, freeaddrinfo, bind, connect, listen, accept, htons, htonl, ntohs, ntohl, inet_addr, inet_ntoa, inet_ntop, send, recv, signal, sigaction, sigemptyset, sigfillset, sigaddset, sigdelset, sigismember, lseek, fstat, fcntl, poll (or equivalent)
Libft authorized	n/a
Description	An IRC server in C++ 98

You are required to develop an IRC server using the C++ 98 standard.

You **must not** develop an IRC client.

You **must not** implement server-to-server communication.

Your executable will be run as follows:

```
./ircserv <port> <password>
```

- **port:** The port number on which your IRC server will be listening for incoming IRC connections.
- **password:** The connection password. It will be needed by any IRC client that tries to connect to your server.



Even though `poll()` is mentioned in the subject and the evaluation scale, you may use any equivalent such as `select()`, `kqueue()`, or `epoll()`.



## IV.1 Requirements

- The server must be capable of handling multiple clients simultaneously without hanging.
- Forking is prohibited. All I/O operations must be **non-blocking**.
- Only **1** `poll()` (or equivalent) can be used for handling all these operations (read, write, but also listen, and so forth).



Because you have to use non-blocking file descriptors, it is possible to use `read/recv` or `write/send` functions with no `poll()` (or equivalent), and your server wouldn't be blocking. However, it would consume more system resources. Therefore, if you attempt to `read/recv` or `write/send` in any file descriptor without using `poll()` (or equivalent), your grade will be 0.

- Several IRC clients exist. You have to choose one of them as a **reference**. Your reference client will be used during the evaluation process.
- Your reference client must be able to connect to your server without encountering any error.
- Communication between client and server has to be done via TCP/IP (v4 or v6).
- Using your reference client with your server must be similar to using it with any official IRC server. However, you only have to implement the following features:
  - You must be able to authenticate, set a nickname, a username, join a channel, send and receive private messages using your reference client.
  - All the messages sent from one client to a channel have to be forwarded to every other client that joined the channel.
  - You must have **operators** and regular users.
  - Then, you have to implement the commands that are specific to **channel operators**:
    - \* KICK - Eject a client from the channel
    - \* INVITE - Invite a client to a channel
    - \* TOPIC - Change or view the channel topic
    - \* MODE - Change the channel's mode:
      - i: Set/remove Invite-only channel
      - t: Set/remove the restrictions of the TOPIC command to channel operators
      - k: Set/remove the channel key (password)

- o: Give/take channel operator privilege
- l: Set/remove the user limit to channel
- Of course, you are expected to write a clean code.

## IV.2 For MacOS only



Since MacOS does not implement `write()` in the same way as other Unix OSes, you are permitted to use `fcntl()`.

You must use file descriptors in non-blocking mode in order to get a behavior similar to the one of other Unix OSes.



However, you are allowed to use `fcntl()` only as follows:

```
fcntl(fd, F_SETFL, O_NONBLOCK);
```

Any other flag is forbidden.

## IV.3 Test example

Verify every possible error and issue, such as receiving partial data, low bandwidth, etc.

To ensure that your server correctly processes all data sent to it, the following simple test using `nc` can be performed:

```
\$> nc -C 127.0.0.1 6667
com~Dman~Dd
\$>
```

Use `ctrl+D` to send the command in several parts: `'com'`, then `'man'`, then `'d\n'`.

In order to process a command, you have to first aggregate the received packets in order to rebuild it.

# Chapter V

## Readme Requirements

A README.md file must be provided at the root of your Git repository. Its purpose is to allow anyone unfamiliar with the project (peers, staff, recruiters, etc.) to quickly understand what the project is about, how to run it, and where to find more information on the topic.

The README.md must include at least:

- The very first line must be italicized and read: *This project has been created as part of the 42 curriculum by <login1>[, <login2>[, <login3>[...]]]*.
- A “**Description**” section that clearly presents the project, including its goal and a brief overview.
- An “**Instructions**” section containing any relevant information about compilation, installation, and/or execution.
- A “**Resources**” section listing classic references related to the topic (documentation, articles, tutorials, etc.), as well as a description of how AI was used — specifying for which tasks and which parts of the project.

➡ **Additional sections may be required depending on the project** (e.g., usage examples, feature list, technical choices, etc.).

Any required additions will be explicitly listed below.



Your README must be written in English.

# Chapter VI

## Bonus part

Here are additional features you may add to your IRC server to make it resemble an actual IRC server more closely:

- Handle file transfer.
- A bot.



The bonus part will only be assessed if the mandatory part is PERFECT. Perfect means the mandatory part has been integrally done and works without malfunctioning. If you have not passed ALL the mandatory requirements, your bonus part will not be evaluated at all.

# Chapter VII

## Submission and peer-evaluation

Submit your assignment to your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Do not hesitate to double-check the names of your files to ensure they are correct.

You are encouraged to create test programs for your project even though they **will not be submitted or graded**. Those tests could be especially useful to test your server during defense, but also your peer's if you have to evaluate another `ft_irc` one day. Indeed, you are free to use whatever tests you need during the evaluation process.



Your reference client will be used during the evaluation process.

During the evaluation, a brief **modification of the project** may occasionally be requested. This could involve a minor behavior change, a few lines of code to write or rewrite, or an easy-to-add feature.

While this step may **not be applicable to every project**, you must be prepared for it if it is mentioned in the evaluation guidelines.

This step is meant to verify your actual understanding of a specific part of the project. The modification can be performed in any development environment you choose (e.g., your usual setup), and it should be feasible within a few minutes — unless a specific timeframe is defined as part of the evaluation.

You can, for example, be asked to make a small update to a function or script, modify a display, or adjust a data structure to store new information, etc.

The details (scope, target, etc.) will be specified in the **evaluation guidelines** and may vary from one evaluation to another for the same project.