35.- Makefile.-

Makefile Explanation:

This Makefile defines how to build a static library named `libft.a` containing various C functions from source files. Let's break it down line by line:

1. NAME:

- Defines the name of the library to be built: `libft.a`.

2. SRC:

- Lists all the source files (`.c`) that will be compiled into objects for the library.

3. CC, CFLAGS:

- `CC`: Specifies the compiler to use (default: `cc`).
- `CFLAGS`: Defines compiler flags (`-Wall`, `-Wextra`, and `-Werror` for warnings and errors).

4. AR, ARFLAGS:

- `AR`: Specifies the archiver to use (default: `ar`).
- `ARFLAGS`: Defines archiver flags (`rcs` for creating and replacing the library).

5. OBJS:

- Generates a list of object files (`.o`) from the source files by replacing `.c` with `.o`.

6. RM:

- Defines the command for removing files (`rm -f`).

7. all:

- Target rule for building the library.
- Depends on all object files being built.
- Runs the command to create the library using `ar` and the defined flags.

8. $(NAME): $(OBJS):

- Dependency rule for the library target.
- Requires all object files to be built before creating the library.
- Uses the `@` symbol to suppress output from the command below.

9. @ $(AR) $(ARFLAGS) $(NAME) $(OBJS):

- Actual command to build the library.
- Uses the archiver (`ar`) with the specified flags to create the library `libft.a` from the listed object files.

10. clean:

- Target rule for cleaning up object files.
- Runs the `rm` command with the `-f` flag to remove all object files.

11. fclean:

- Depends on the `clean` target.
- Additionally removes the library file using `rm`.

12. re:

- Depends on the `fclean` target (clean everything).
- Rebuilds the library from scratch (clean and then all).

13. .PHONY:

- Declares certain targets (all, clean, fclean, re) as phony, meaning they don't have corresponding files but depend on other rules.

Advantages of Makefiles:

- **Automation:** Automate building and rebuilding projects by specifying dependencies and commands.
- **Efficiency:** Only rebuild parts of the project that have changed, saving time.
- **Customization:** Define different build targets and rules for specific needs.
- **Portability:** Work across different systems with compatible compilers and archivers.

This Makefile provides a basic example of how to manage and build a static library using Makefiles. It demonstrates the advantages of automation and efficiency in the build process.

Here below the code:

```
NAME = libft.a

SRC = ft_isalpha.c ft_isdigit.c ft_isalnum.c ft_isascii.c ft_isprint.c \
      ft_strlen.c ft_memset.c ft_bzero.c ft_memcpy.c ft_memmove.c \
      ft_strlcpy.c ft_strlcat.c ft_toupper.c ft_tolower.c ft_strchr.c\
      ft_strrchr.c ft_strncmp.c ft_memchr.c ft_memcmp.c ft_strnstr.c \
      ft_atoi.c ft_calloc.c ft_strdup.c ft_substr.c ft_strjoin.c ft_strtrim.c \
      ft_split.c ft_itoa.c ft_strmapi.c ft_striteri.c ft_putchar_fd.c \
      ft_putstr_fd.c ft_putendl_fd.c ft_putnbr_fd.c

CC = cc
CFLAGS = -Wall -Wextra -Werror

AR = ar
```

```
ARFLAGS = rcs

OBJS = $(SRC:.c=.o)

RM = rm -f

all: $(NAME)

$(NAME): $(OBJS)
	@ $(AR) $(ARFLAGS) $(NAME) $(OBJS)

clean:
		$(RM) $(OBJS)

fclean: clean
		$(RM) $(NAME)

re:	fclean $(NAME)

.PHONY: all clean fclean re
```