《Makefile 详解及使用方法》

\*Makefile 是一个用于自动化编译程序的脚本文件，它可以帮助我们管理项目的编译过程，特别是在项目包含多个源文件时，使用 Makefile 可以大大提高编译效率。下面我们来详细解读你提供的 Makefile 文件，并说明如何使用它。

### Makefile 文件详细解读

|  |
| --- |
| CC = arm-linux-gnueabihf-gcc  CFLAGS = -Wall -O2 -I./LIB/lcd\_font -I./SYSTEM/show\_bmp\_to\_lcd -std=c99 # 启用C99标准  LDFLAGS = -L./LIB/lcd\_font  LDLIBS = -llcd\_font -lm  EXEC = p  OBJS = USER/p.o SYSTEM/show\_bmp\_to\_lcd/show\_bmp\_to\_lcd.o  all: $(EXEC)  rm -f $(OBJS) # 编译完成后删除.o文件  $(EXEC): $(OBJS)  $(CC) $(LDFLAGS) -o $@ $^ $(LDLIBS)  USER/p.o: USER/p.c LIB/lcd\_font/lcd\_font.h SYSTEM/show\_bmp\_to\_lcd/show\_bmp\_to\_lcd.h  $(CC) $(CFLAGS) -c $< -o $@  SYSTEM/show\_bmp\_to\_lcd/show\_bmp\_to\_lcd.o: SYSTEM/show\_bmp\_to\_lcd/show\_bmp\_to\_lcd.c SYSTEM/show\_bmp\_to\_lcd/show\_bmp\_to\_lcd.h  $(CC) $(CFLAGS) -c $< -o $@  clean:  rm -f $(OBJS) $(EXEC) |

#### 1. 变量定义

* **编译器和编译选项**
  + CC = arm-linux-gnueabihf-gcc：指定使用 arm-linux-gnueabihf-gcc 作为编译器，这通常用于交叉编译，目标平台是 ARM 架构。
  + CFLAGS = -Wall -O2 -I./LIB/lcd\_font -I./SYSTEM/show\_bmp\_to\_lcd -std=c99：编译选项，-Wall 表示显示所有警告信息，-O2 表示使用二级优化，-I 用于指定头文件的搜索路径，-std=c99 表示启用 C99 标准。
  + LDFLAGS = -L./LIB/lcd\_font：链接选项，-L 用于指定库文件的搜索路径。
  + LDLIBS = -llcd\_font -lm：链接的库文件，-llcd\_font 表示链接 lcd\_font 库，-lm 表示链接数学库。
* **目标文件和可执行文件**
  + EXEC = p：指定最终生成的可执行文件名为 p。
  + OBJS = USER/p.o SYSTEM/show\_bmp\_to\_lcd/show\_bmp\_to\_lcd.o：指定编译过程中生成的中间目标文件。

#### 2. 规则定义

* all **规则**

|  |
| --- |
| all: $(EXEC)  rm -f $(OBJS) # 编译完成后删除.o文件 |

* all 是默认的目标，当你在终端中输入 make 命令时，会执行 all 规则。
* $(EXEC) 是 all 规则的依赖项，表示在执行 all 规则之前，需要先生成可执行文件 p。
* rm -f $(OBJS) 是 all 规则的命令，用于在编译完成后删除中间目标文件。
* **可执行文件生成规则**

|  |
| --- |
| $(EXEC): $(OBJS)  $(CC) $(LDFLAGS) -o $@ $^ $(LDLIBS) |

* $(EXEC) 是目标，即最终生成的可执行文件 p。
* $(OBJS) 是依赖项，即生成可执行文件所需的中间目标文件。
* $(CC) $(LDFLAGS) -o $@ $^ $(LDLIBS) 是命令，$@ 表示目标文件，即 p，$^ 表示所有依赖文件，即 USER/p.o 和 SYSTEM/show\_bmp\_to\_lcd/show\_bmp\_to\_lcd.o。该命令的作用是将所有目标文件链接成可执行文件。
* **目标文件生成规则**

|  |
| --- |
| USER/p.o: USER/p.c LIB/lcd\_font/lcd\_font.h SYSTEM/show\_bmp\_to\_lcd/show\_bmp\_to\_lcd.h  $(CC) $(CFLAGS) -c $< -o $@  SYSTEM/show\_bmp\_to\_lcd/show\_bmp\_to\_lcd.o: SYSTEM/show\_bmp\_to\_lcd/show\_bmp\_to\_lcd.c SYSTEM/show\_bmp\_to\_lcd/show\_bmp\_to\_lcd.h  $(CC) $(CFLAGS) -c $< -o $@ |

* 这两个规则分别用于生成 USER/p.o 和 SYSTEM/show\_bmp\_to\_lcd/show\_bmp\_to\_lcd.o 目标文件。
* 每个规则的第一行指定了目标文件和依赖文件，第二行是编译命令，-c 表示只编译不链接，$< 表示第一个依赖文件，即源文件。
* clean **规则**

|  |
| --- |
| clean:  rm -f $(OBJS) $(EXEC) |

* clean 规则用于清理编译过程中生成的中间目标文件和可执行文件。当你在终端中输入 make clean 命令时，会执行该规则。

### 如何使用 Makefile

#### 1. 编译项目

在终端中，进入 Makefile 文件所在的目录，然后输入 make 命令，即可开始编译项目。make 命令会根据 Makefile 文件中的规则，自动编译源文件并链接生成可执行文件。

|  |
| --- |
| cd gua  make |

#### 2. 清理编译结果

如果需要清理编译过程中生成的中间目标文件和可执行文件，可以在终端中输入 make clean 命令。

|  |
| --- |
| make clean |

通过以上步骤，你就可以使用 Makefile 来管理项目的编译过程了。Makefile 会自动检查文件的修改时间，如果某个源文件被修改了，make 命令会只重新编译该文件及其依赖的文件，从而提高编译效率。