

Introdução a Técnicas de Programação

Aula 15
Extras



Extras

1. Argumentos em linha de comando
2. Automação da compilação através de um Makefile
3. Criação de bibliotecas



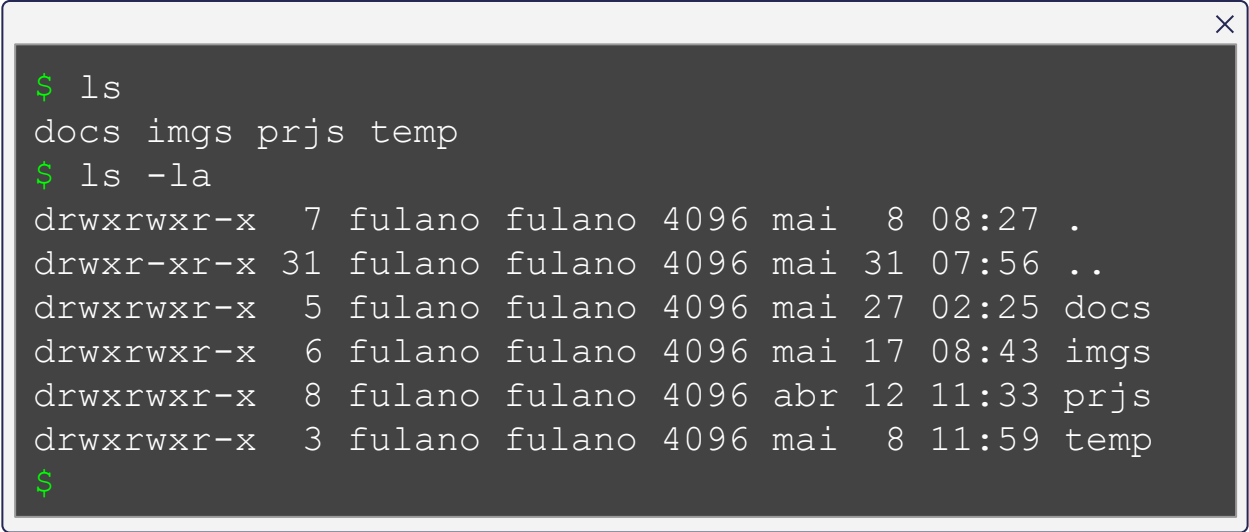
Linha de comando



Argumentos de um programa

Um programa pode se comportar de forma diferente em função de argumentos (parâmetros) que lhe são passados.

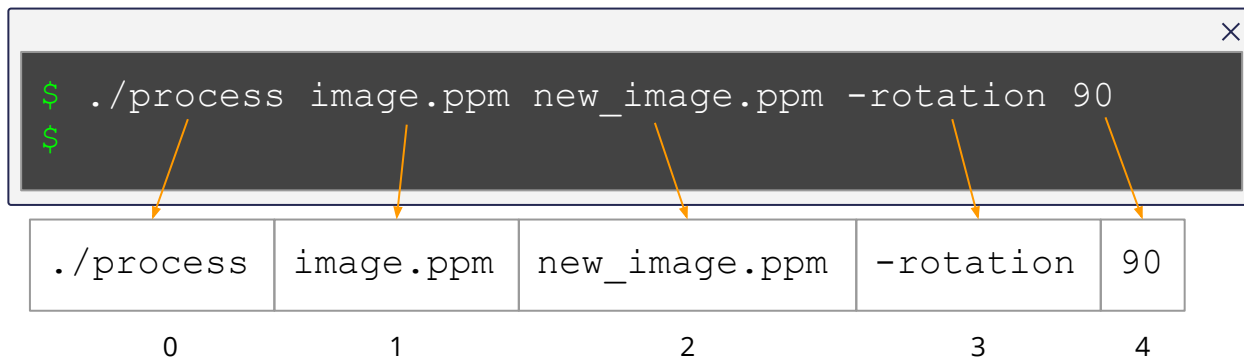
Exemplo:

A terminal window with a dark background and a light border. It shows the execution of two commands: 'ls' and 'ls -la'. The first command lists the contents of the current directory, and the second command lists the contents with detailed permissions, owner, group, size, and modification time. The prompt '\$' is shown in green.

```
$ ls
docs imgs prjs temp
$ ls -la
drwxrwxr-x  7 fulano fulano 4096 mai  8 08:27 .
drwxr-xr-x 31 fulano fulano 4096 mai 31 07:56 ..
drwxrwxr-x  5 fulano fulano 4096 mai 27 02:25 docs
drwxrwxr-x  6 fulano fulano 4096 mai 17 08:43 imgs
drwxrwxr-x  8 fulano fulano 4096 abr 12 11:33 prjs
drwxrwxr-x  3 fulano fulano 4096 mai  8 11:59 temp
$
```

Recebendo argumentos da linha de comando

A função `main()` recebe um array de strings com a linha de comando



É necessário, declarar esse array como parâmetro da função `main()`.


1. **argc** = quantos elementos no array
2. **argv** = array de strings

```
int main(int argc, char *argv[]) {  
    ...  
}
```

Consultando os argumentos

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    for (int i = 0; i < argc; i++) {
        printf("%s\n", argv[i]);
    }
    return 0;
}
```

A terminal window with a dark background and a light border. It shows a command being executed: `./process image.ppm new_image.ppm -rotation 90`. The output of the command is displayed line by line: `./process`, `image.ppm`, `new_image.ppm`, `-rotation`, and `90`. The prompt character is a green dollar sign.

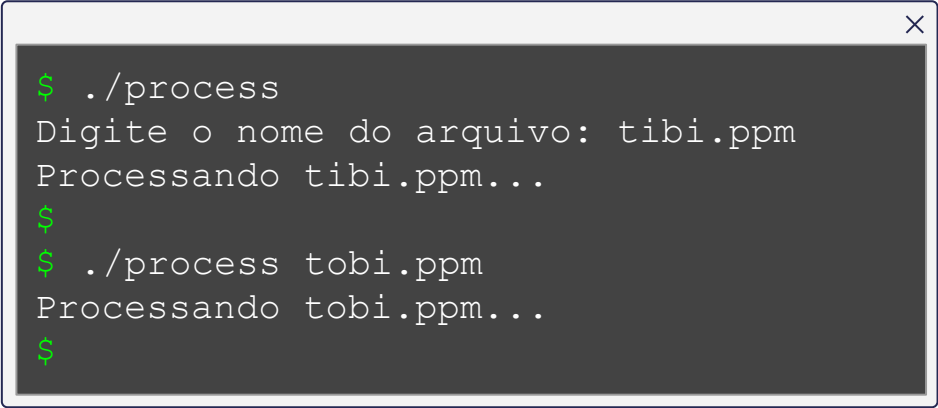
```
$ ./process image.ppm new_image.ppm -rotation
90
./process
image.ppm
new_image.ppm
-rotation
90
$
```

O 1º elemento do array é sempre o nome do programa
Os argumentos começam a partir do 2º elemento (índice 1)

Exemplo de uso dos argumentos

```
#include <stdio.h>
#include <string.h>

void main(int argc, char *argv[]) {
    char filename[50];
    if (argc == 1) { // nenhum argumento
        printf("Digite o nome do arquivo: ");
        scanf("%s", filename);
    }
    else {
        strcpy(filename, argv[1]);
    }
    printf("Processando %s...\n", filename);
}
```

A terminal window with a dark background and a close button in the top right corner. It shows the execution of a program named 'process'. The first run prompts the user to enter a filename, and the second run takes the filename as a command-line argument.

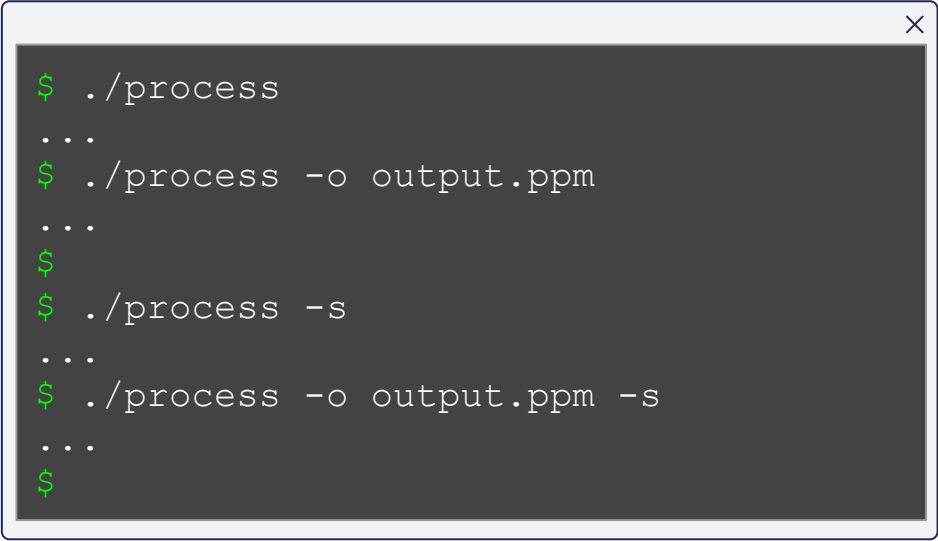
```
$ ./process
Digite o nome do arquivo: tibi.ppm
Processando tibi.ppm...
$
$ ./process tobi.ppm
Processando tobi.ppm...
$
```

O programa pode não solicitar dados se estes são passados em linha de comando.

Exemplo de uso dos argumentos

```
#include <stdio.h>
#include <string.h>

void main(int argc, char *argv[]) {
    for (int i = 1; i < argc; i++) {
        if (strcmp(argv[i], "-o") == 0) {
            i++;
            setOutput(argv[i]);
        }
        else if (strcmp(argv[i], "-s") == 0) {
            setSomething();
        }
        ...
    }
    ...
}
```

A terminal window with a dark background and a close button in the top right corner. It shows several command-line interactions with a program named 'process'. The prompt is '\$'. The commands and their outputs are: './process' followed by '...', './process -o output.ppm' followed by '...', './process -s' followed by '...', and './process -o output.ppm -s' followed by '...'.

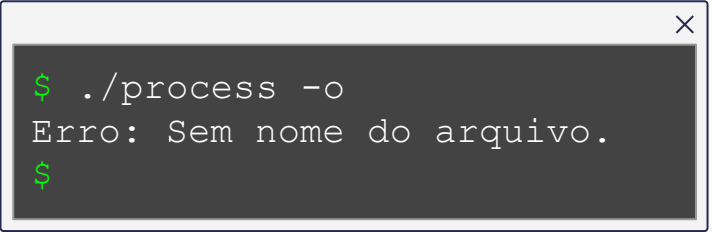
```
$ ./process
...
$ ./process -o output.ppm
...
$
$ ./process -s
...
$ ./process -o output.ppm -s
...
$
```

Pode-se tornar o programa bastante flexível.

Tratando erros

```
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[]) {
    for (int i = 1; i < argc; i++) {
        if (strcmp(argv[i], "-o") == 0) {
            i++;
            if (i == argc) {
                fprintf(stderr, "Erro: Sem nome do arquivo.\n");
                return 1;
            }
            setOutput(argv[i]);
        }
        ...
    }
    return 0;
}
```

A terminal window with a dark background and a close button in the top right corner. It shows a shell prompt '\$' followed by the command './process -o'. The next line shows the output 'Erro: Sem nome do arquivo.' followed by another shell prompt '\$'.

```
$ ./process -o
Erro: Sem nome do arquivo.
$
```

É importante tratar eventuais erros no uso da linha de comando.

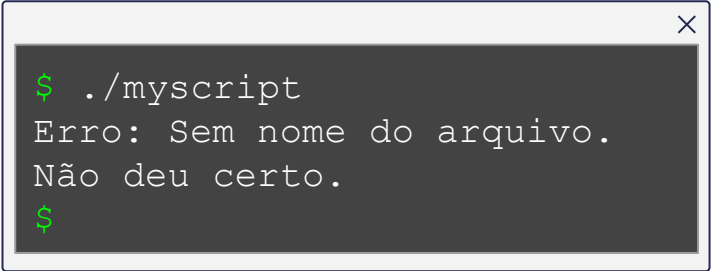
Retorno do programa

```
int main(int argc, char *argv[]) {  
    ...  
    if (...) {  
        ...  
        return 1; // falha  
    }  
    return 0; // sucesso  
}
```

process.c

```
#!/bin/bash  
if ./process -o; then  
    echo Deu certo  
else  
    echo Não deu certo  
fi
```

myscript.sh

A terminal window with a dark background and a light border. It shows a shell prompt '\$' followed by the command './myscript'. The output consists of two lines: 'Erro: Sem nome do arquivo.' and 'Não deu certo.' followed by another shell prompt '\$'.

```
$ ./myscript  
Erro: Sem nome do arquivo.  
Não deu certo.  
$
```

Arquivos de script (shell) podem chamar o programa para automatizar tarefas e depois recuperar o código de retorno do programa (main()) para saber se a tarefa deu certo.

Makefile

Compilação de vários arquivos

Compilar um programa com vários arquivos fontes requer vários parâmetros.



```
$ gcc main.c ui.c file.c net.c manager.c image.c user.c -o prog
```

Se modificarmos apenas um arquivo, precisamos ter que recompilar tudo?

Imagine um projeto com 100 arquivos e você inseriu apenas um comentário

Se gerarmos os arquivos binários para cada arquivo (.o, ver Aula 1, etapas da compilação), quando modificarmos um arquivo, precisamos compilar só ele, mas é necessário juntar os outros .o no executável.




```
$ gcc ui.c -c -o ui.o  
$ gcc main.o ui.o file.o net.o manager.o image.o user.o -o prog
```

Automatização de tarefas

Importante para reduzir tempo (digitação, compilação...)!

O programa **make** foi criado para automatizar tarefas e segue um formato específico (arquivo **Makefile**) para definir **regras de compilação**.

A terminal window with a dark background and a light border. It shows the output of the 'make' command. The prompt is a green '\$'. The output consists of several lines of gcc compilation commands, each compiling a source file from the 'src' directory into an object file in the 'src' directory. The final line links all the object files into an executable 'bin/main' with various linker options. The prompt '\$' is shown again at the bottom.

```
$ make
gcc src/main.c -o src/main.o -c -g -I./inc
gcc src/ui.c -o src/ui.o -c -g -I./inc
gcc src/file.c -o src/file.o -c -g -I./inc
gcc src/net.c -o src/net.o -c -g -I./inc
gcc src/manager.c -o src/manager.o -c -g -I./inc
gcc src/image.c -o src/image.o -c -g -I./inc
gcc src/user.c -o src/user.o -c -g -I./inc
gcc src/main.o src/ui.o src/file.o src/net.o src/manager.o
src/image.o src/user.o -o bin/main -L./lib -lm -lconsole
$
```

Exemplo de makefile simples

O makefile é definido por **regras**, no seguinte formato:

```
regra: dependências  
    comando 1  
    comando 2
```

Uma regra é ativada quando uma de suas dependências está desatualizada. Para atualizar, os comandos são executados.

```
prog: file.o main.o  
    gcc -o prog file.o main.o  
  
file.o: file.c  
    gcc -o file.o -c file.c -W -Wall -ansi -pedantic  
  
main.o: main.c  
    gcc -o main.o -c main.c -W -Wall -ansi -pedantic
```

ATENÇÃO: a indentação dos comandos TEM que ser com TAB!!!

Variáveis

Podemos definir variáveis para substituir em vários locais.

Usa-se o operador = para atribuir um valor

Usa-se \$(var) para consultar ou substituir o valor da variável.

```
CC = gcc
CFLAGS = -W -Wall -ansi -pedantic

prog: file.o main.o
    $(CC) -o prog file.o main.o

file.o: file.c
    $(CC) -o file.o -c file.c $(CFLAGS)

main.o: main.c
    $(CC) -o main.o -c main.c $(CFLAGS)
```

Variáveis e substituições especiais

`$(SOURCES:.c=.o)`

Para todos os nomes em SOURCES que terminarem com .c define um nome .o

`%.o: %.c`

O % substitui um nome qualquer. Assim, a regra indica que qualquer nome terminado com .o tem como dependência o nome terminando com .c.

Por exemplo: main.o depende de main.c

`$@`: substitui o nome da regra

`$<`: substitui o nome da 1ª dependência

...

```
EXEC = prog
```

```
CC = gcc
```

```
CFLAGS = -W -Wall -ansi -pedantic
```

```
SOURCES = main.c file.c ui.c image.c
```

```
OBJS = $(SOURCES:.c=.o)
```

```
%.o: %.c
```

```
$(CC) $< -o $@ -c $(CFLAGS)
```

```
$(EXEC): $(OBJS)
```

```
$(CC) $^ -o $@
```


Makefile usado no início da disciplina

```
code = main
EXEC_NAME = $(code)
EXEC_DIR = ./bin
INC_DIR = ./inc
LIB_DIR = ./lib
SRC_DIR = ./src

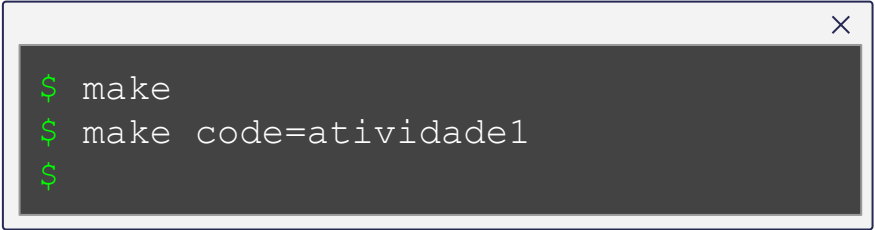
CC = gcc
CFLAGS = -g
LIBS = -lm -lconsole

EXEC = $(EXEC_DIR)/$(EXEC_NAME)
SRC = $(SRC_DIR)/$(code).c
OBJ = $(SRC:.c=.o)
```

```
%.o: %.c
    $(CC) $< -o $@ -c $(CFLAGS) -I$(INC_DIR)

$(EXEC): $(OBJ)
    $(CC) $^ -o $@ -L$(LIB_DIR) $(LIBS)

.PHONY: clean
clean:
    rm -f ./src/*.o ./bin/*
```

A terminal window with a dark background and a title bar containing a close button (X). It shows three lines of text: a green prompt character followed by 'make', a green prompt character followed by 'make code=atividade1', and a green prompt character on the third line.

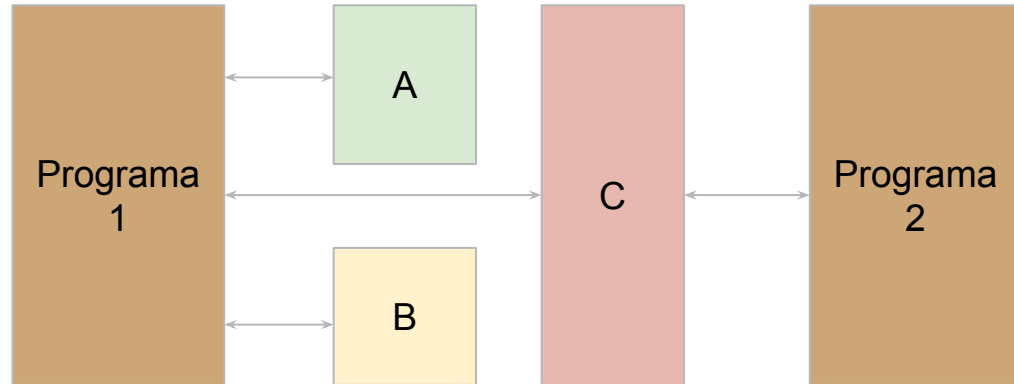
```
$ make
$ make code=atividade1
$
```

Criação de bibliotecas estáticas

Bibliotecas

Coleção de rotinas (“módulos”) que dão apoio a um programa.

Código já compilado (binário), pronto a ser reutilizado em vários programas.



Tipos de bibliotecas

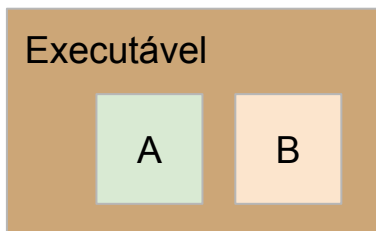
Bibliotecas estáticas

O código binário da biblioteca É incorporado no arquivo executável do programa.

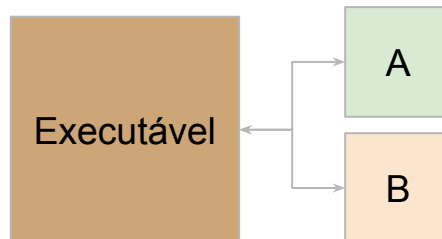
Bibliotecas dinâmicas

O código binário da biblioteca NÃO É incorporado no arquivo executável do programa. Apenas quando o programa é executado, que há uma “ligação dinâmica” com a biblioteca.

Biblioteca estática

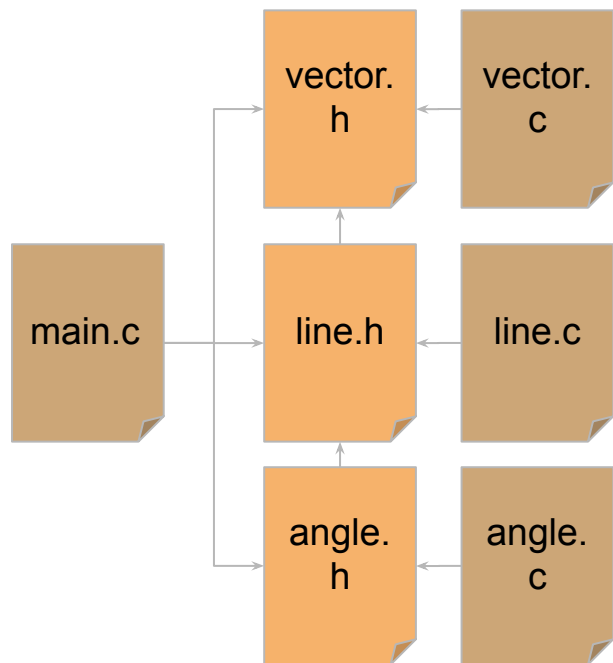


Biblioteca dinâmica



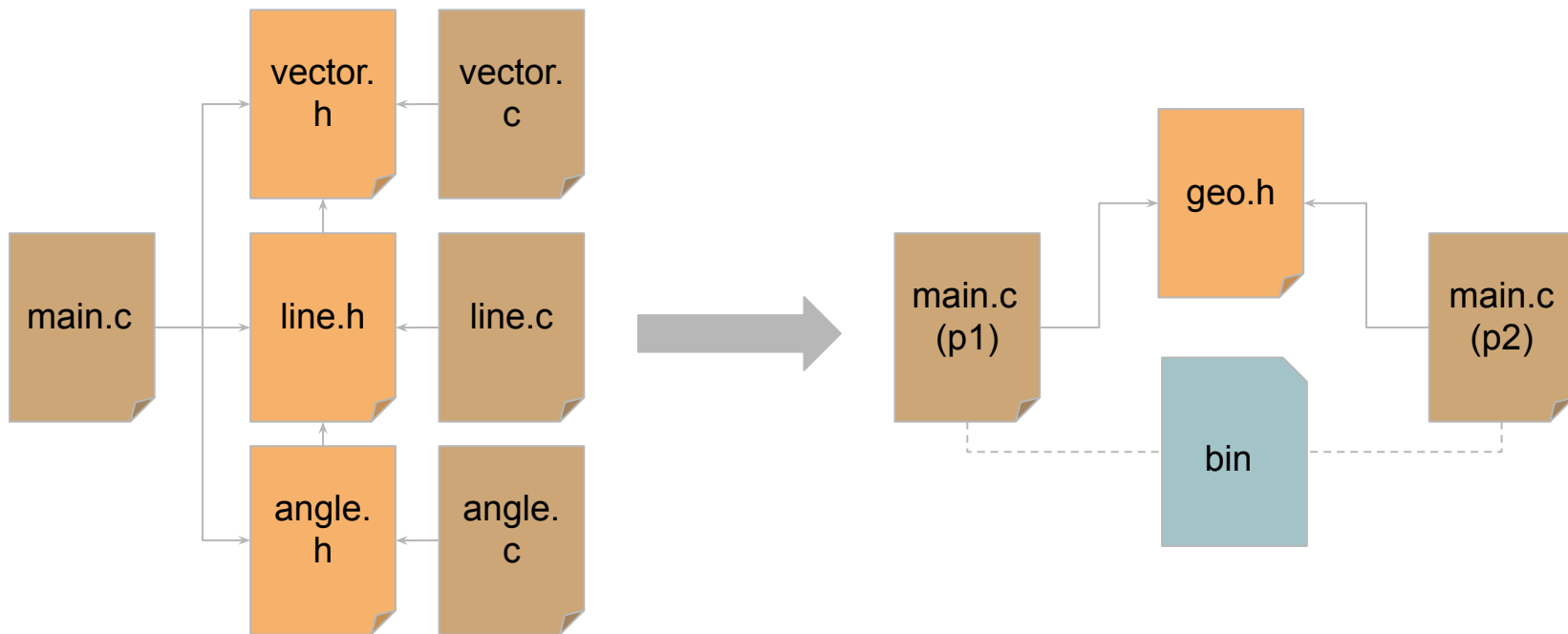
Criando uma biblioteca estática

Exemplo de configuração



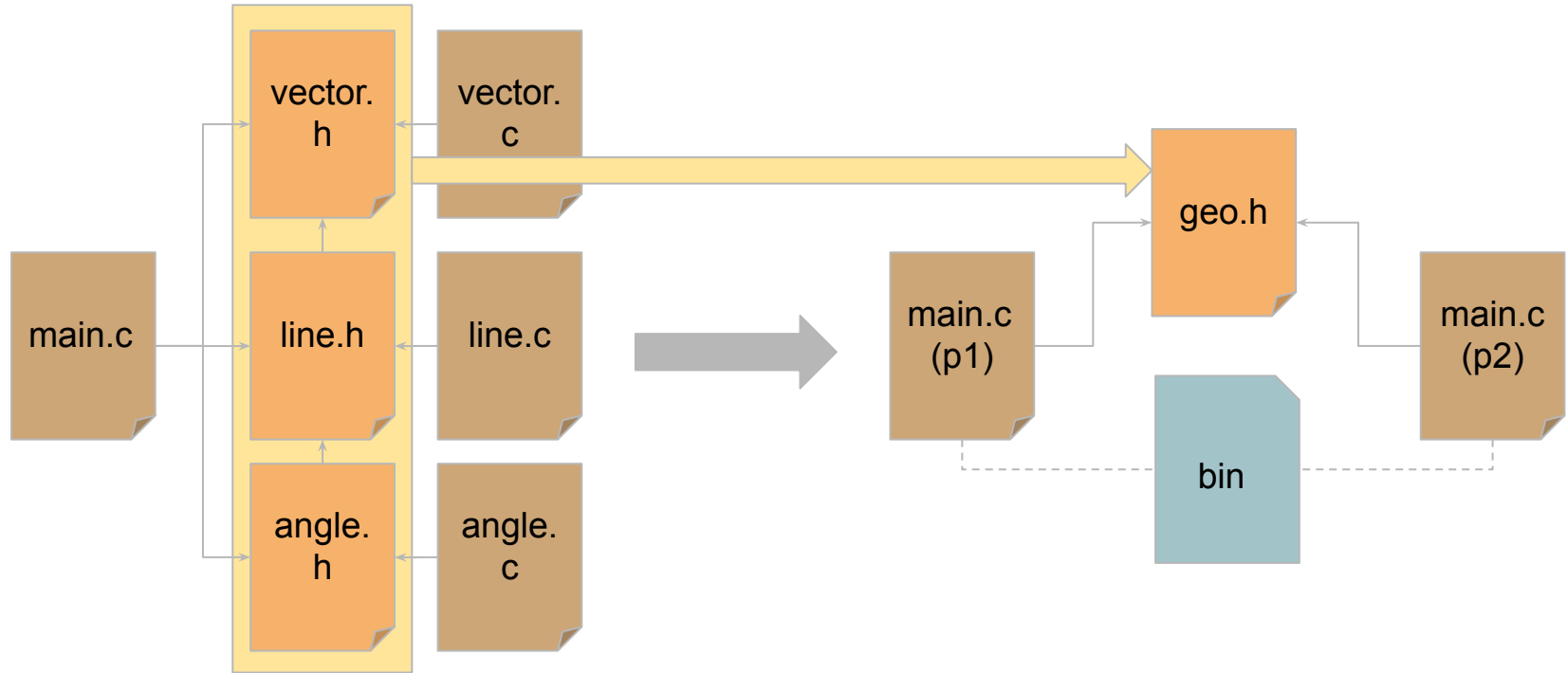
Criando uma biblioteca estática

Exemplo de configuração → reuso do binário por mais de um programa



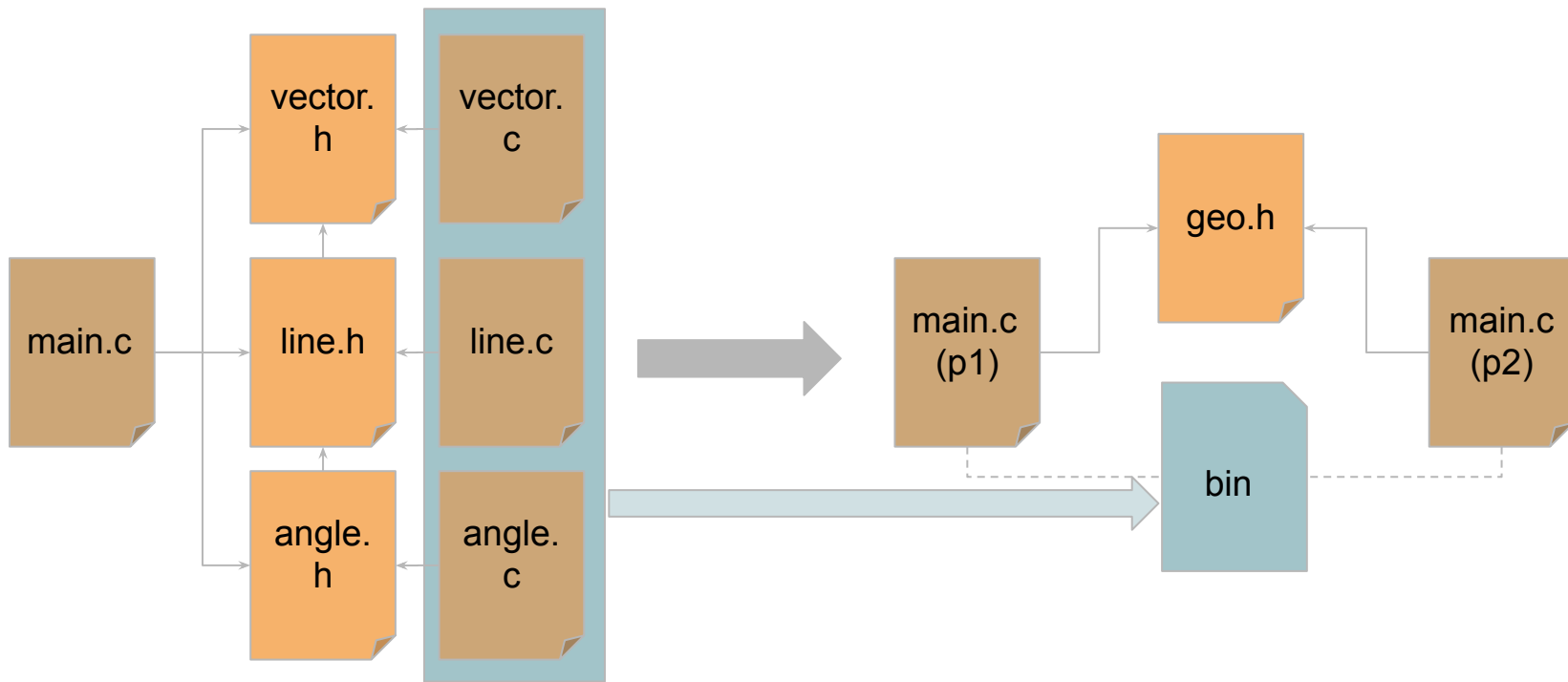
Criando uma biblioteca estática

União dos tipos e assinatura de funções em um único .h



Criando uma biblioteca estática

Compilação dos arquivos .c em um binário (não tem `main()`)



Compilação e uso da biblioteca estática

Primeiro, deve-se gerar os arquivos objeto (.o), binários de cada .c

```
$ gcc vector.c -c -o vector.o  
$ gcc line.c -c -o line.o  
$ gcc angle.c -c -o angle.o
```

Depois, usa-se o programa **ar** para juntar os .o em um arquivo .a

```
$ ar rcs geo.a vector.o line.o angle.o
```

Com o arquivo .a (biblioteca estática), pode-se gerar o executável

```
$ gcc main.o -lgeo -o exec
```