

IMD0030

LINGUAGEM DE PROGRAMAÇÃO I

Makefile, CMake
(material baseado nas notas de aula do Prof. Silvio Sampaio)

Introdução

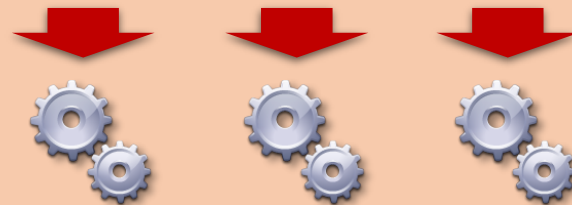
- Compilar X Ligar

Compilar: `g++ -c main.cpp func.cpp util.cpp`

Arquivos fonte (.cpp/.h)



Compilador



Arquivo objeto (.o)

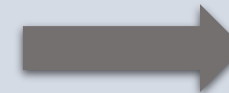


Ligar (linker): `g++ -o prog main.o func.o util.o`

Arquivos objeto (.o)



Runtime Support



Biblioteca
(.lib, .a, .so)

Uma biblioteca (library) é uma coleção de arquivos objeto pré-compilados que permitem a reutilização de código entre diferentes projetos.

Arquivo Executável



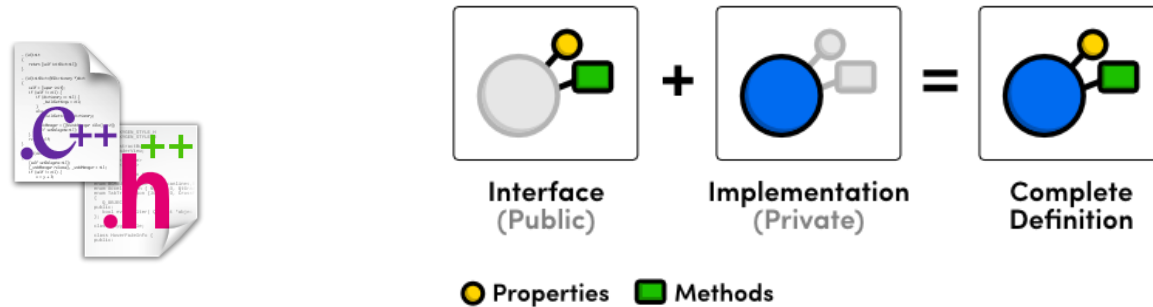
Compilar: `g++ -c main.cpp func.cpp util.cpp`

Ligar: `g++ -o prog main.o func.o util.o`

Compilar e ligar: `g++ -o prog main.cpp func.cpp util.cpp`

Modularização externa

- Por convenção da linguagem C++, a organização do código de um programa pode ser feita da seguinte maneira:
 - **Arquivos de cabeçalho (.h)** contêm declarações de estruturas, tipos, variáveis globais, protótipos de funções, constantes, etc. e não podem conter a função principal do programa (`main`)
 - **Arquivos de corpo (.cpp)** implementam ou fazem chamadas ao que é definido nos arquivos de cabeçalho



Ferramentas de auxílio à compilação

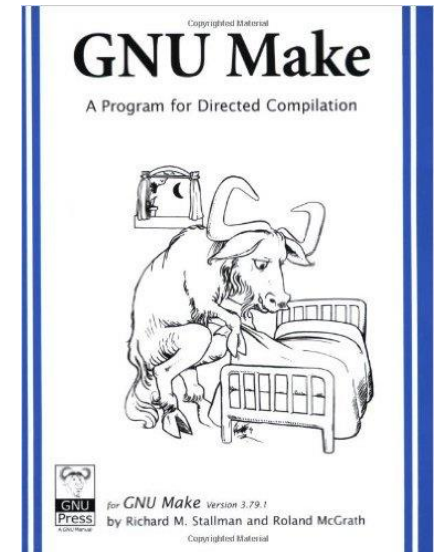
- Makefile
- CMake

Sugestão: comandos básicos da linha de comando

- <https://www.dataquest.io/blog/why-learn-the-command-line>
 - https://en.wikipedia.org/wiki/Unix_filesystem#Conventional_directory_layout
 - https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Understanding_client-side_tools/Command_line
-

Makefile

- A compilação de projetos acaba por se tornar uma tarefa difícil
 - Muitos arquivos que devem ser compilados
 - Repetição dos comandos de compilação para cada arquivo
 - A alteração em um único arquivo implica na recompilação de quais partes do projeto? Por garantia sempre compilamos tudo novamente?
- Esta dificuldade se agrava de acordo com a complexidade do projeto, que pode envolver centenas de arquivos



Makefile

- A título de exemplo, imagine um projeto com os seguintes arquivos: **ccountln.h**, **ccountln.cpp**, **fileops.h**, **fileops.cpp**, **process.h**, **process.cpp**, **parser.h**, **parser.cpp** e **main.cpp**
 - Para compilarmos manualmente este projeto, podemos seguir duas abordagens:
 - Compilar cada arquivo individualmente e ligar os arquivos objeto para criar o executável
 - `g++ -O0 -g -Wall -ansi -pedantic -c ccountln.cpp`
 - `g++ -O0 -g -Wall -ansi -pedantic -c parser.cpp`
 - `g++ -O0 -g -Wall -ansi -pedantic -c fileops.cpp`
 - `g++ -O0 -g -Wall -ansi -pedantic -c process.cpp`
 - `g++ -O0 -g -Wall -ansi -pedantic -c main.cpp`
 - `g++ ccountln.o parser.o fileops.o process.o main.o -o processos`
 - Compilar e ligar na mesma linha de comando
 - `g++ -O0 -g -Wall -ansi -pedantic -o processos ccountln.cpp parser.cpp fileops.cpp process.cpp main.cpp`
-

Makefile

- **Make** (responsável por processar o Makefile) é um utilitário GNU que determina quais partes de um projeto necessitam ser compilados ou recompilados, permitindo configurar os comandos para compilar e ligar o executável de forma automatizada (para todos os arquivos indicados)
 - Nos salva do tédio de repetir as linhas de comando ou de comandos gigantes do **g++**, além de economizar tempo
-

Makefile

- Um arquivo Makefile consiste de uma série de regras (*rules*), na seguinte forma:


alvo : pre-requisitos ...

comando1

comando2

comando3

...



```
imd0030: main.cpp prime.h
    g++ -o imd0030 main.cpp
```

<TAB>

- A regra explica como e quando gerar (ou regerar) o arquivo alvo
 - A simples chamada ao comando *make* executa, por padrão, a primeira regra
 - Para executar uma regra específica, é preciso chamar o comando “**make <alvo>**”
 - **Make exige um caracter <TAB> antes de cada comando (causa erros!)**
-


Makefile

- **“alvo”**: Usualmente o nome de um executável/binário ou arquivo objeto (.o) que deve ser gerado pelo compilador, mas pode indicar também uma ação a ser realizada (Ex: “clean”)
 - **“pré-requisitos”**: Uma lista dos arquivos necessários para criar o alvo
 - Se um destes arquivos tiver sido alterado, então o utilitário *make* irá reconstruir o alvo
 - Também chamado de “dependências”
 - **“comando”**: Uma ação a ser realizada
 - Usualmente, uma compilação ou ligação (usando o g++, por exemplo)
 - Pode ser algum comando do S.O. ou programa externo
 - No *make*, os comandos são executados e geram saídas como se estivessem a ser rodados a partir da linha de comando
-

Exemplo de um makefile

imd0030: main.cpp prime.h

g++ -o imd0030 main.cpp



apenas uma
regra

imd0030: main.o processos.o

g++ -o imd0030 main.o processos.o



série de regras

main.o: main.cpp util.h

g++ -c main.cpp

processos.o: processos.cpp prime.h

g++ -c processos.cpp

Exemplo de um makefile

#Descrição do programa

#Autor

Variável

PROG = imd0030

Executável

CC = g++

Compilador

CPPFLAGS = -O0 -g -Wall -pedantic -I/usr/imd0030/include

Diretivas de
compilação

OBJS = main.o processos.o database.o util.o

Arquivos objeto definidos como pré-requisitos

\$(PROG) : \$(OBJS)

Alvo padrão

\$(CC) -o \$(PROG) \$(OBJS)

Regra de
construção do
executável

(Cont.)

main.o :

\$(CC) \$(CPPFLAGS) -c main.cpp

Regra de
construção dos
arquivos objeto

processos.o : processos.h

\$(CC) \$(CPPFLAGS) -c processos.cpp

database.o : database.h

\$(CC) \$(CPPFLAGS) -c database.cpp

util.o : util.h

\$(CC) \$(CPPFLAGS) -c util.cpp

clean:

Alvo "clean"

rm -f \$(PROG) \$(OBJS)

Regra de
limpeza dos
arquivos

Makefile

- Após a criação do arquivo *Makefile*, basta digitar o comando **make** no diretório contendo o *Makefile* para dar início ao processo automatizado de compilação e ligação
 - O arquivo *Makefile* pode vir a ser grande e complexo (de acordo com o seu projeto), mas, uma vez pronto e funcional, basta o uso do comando **make** e pronto!
-

Makefile - desafios

- Um desafio do Makefile é que, em projetos grandes, o arquivo pode tornar-se extenso e de difícil compreensão
 - problemas com manutenção e erros
- Makefiles requerem sistemas operacionais baseados em Unix (Linux, macOS, etc.)
 - problemas com portabilidade

CMake



- CMake: conjunto de ferramentas para construção, teste e empacotamento de programas
 - *open-source, cross-platform*
- A partir de um arquivo de configuração “CMakeLists.txt”, gera um makefile compatível com a plataforma de compilação/execução
- CMake é invocado através do comando `cmake`

<https://cmake.org>

Exemplo de arquivo CMakeLists.txt

```
#versão mínima requerida para o CMake
cmake_minimum_required(VERSION 3.5)

#configurações tais como flags para o compilador
set(CMAKE_CXX_FLAGS "-Wall -pedantic")
#especificação de versão da linguagem (
set(CMAKE_CXX_STANDARD 11)

#nome do projeto
project(TEMPERATURA VERSION 1.0 LANGUAGES CXX )

# Diretório contendo os arquivos de cabeçalho
include_directories(include)

# Cria um executável com nome "programa", a partir dos arquivos main.cpp
prime.cpp tools.cpp
add_executable( programa main.cpp prime.cpp tools.cpp)
```

Exercício de Aprendizagem

1. Escreva um programa que calcula a média aritmética de 3 números inteiros fornecidos pelo usuário.
2. Modifique o programa (1) para que a entrada seja fornecida pela entrada padrão
3. Escreva um programa que calcule a média de um conjunto de números. O programa deve receber pela entrada padrão o número de elementos do conjunto e os elementos do conjunto. A saída deve ser impressa na tela.

OBS: Crie um makefile para todos. Cada parte do programa deve ser modularizada. Main deve conter apenas chamadas aos módulos (internos e/ou externos).

Exercício de Sala

4. Escreva um programa que calcule a variância populacional de um conjunto de números. O programa deve receber pela entrada padrão o número de elementos do conjunto e, após execução, solicitar os números ao usuário. Os elementos digitados e a variância devem ser impressas na tela.

4.1 – A entrada e escrita dos dados devem ser feitas, cada uma, por uma função. Elas devem estar em um arquivo `.h/.cpp`

4.2 – A variância deve ser calculada em outra função em outro arquivo `.h/.cpp`

4.3 – Faça um makefile
