Universidade de São Paulo
Brasil

Aula 18
Automação de Compilação

MAC0216 - Técnicas de Programação I

Professores: Alfredo, Daniel, Fabio e Kelly

Departamento de Ciência da Computação
Instituto de Matemática e Estatística

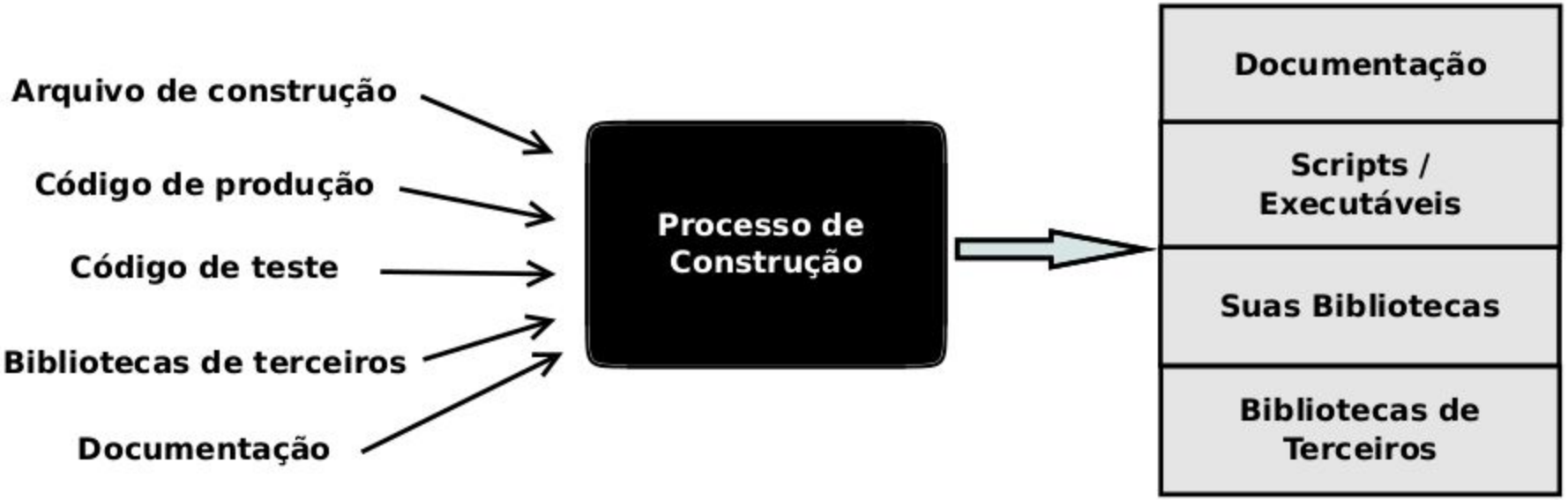
Escrita de software × Construção de software

- **Escrita de software:**
  - É um processo que envolve arte, ciência e engenharia

- **Construção (*built*) de software:**
  - É como fazer linguiça – o código fonte é moído em pedaços, para ser consumido pelo computador
    - Geralmente, não estamos interessados nos detalhes envolvidos nesse processo
  - É um processo repetitivo, **passível de automação**

Automação de construção

Também conhecida como “automação da compilação”



Automação de construção

- O **arquivo de construção (*build file*)** contém o passo-a-passo de tudo que tem que ser executado para a construção de um programa
- Ele lista os itens envolvidos nesses passos
  - Códigos fontes, arquivos de configuração, bibliotecas, etc.
- Ele pode ser criado manualmente pelo desenvolvedor ou pode ser gerado de forma automática por algumas ferramentas

Ferramentas para a automação de compilação

- **GNU Make** – bastante usada para código em C, C++, Latex
  - Pode ser usada com qualquer linguagem cujo compilador possa ser executado com um comando shell
- **Apache Ant** – usada principalmente para código Java (mas também funciona para C, C++, ...)
- **Apache Maven** – usada em projetos em Java
  - É mais do que um gerenciador de compilação; ela auxilia no gerenciamento do projeto
- **MSBuild (Microsoft)** – funciona conjuntamente com o Visual Studio (Visual Basic, Visual C++, C#, ...)
- ...

GNU Make

GNU Make

- Determina automaticamente quais pedaços de um programa precisam ser (re)compilados e dispara os comandos que os (re)compilam
- Pode ser usado com qualquer linguagem de programação cujo compilador possa ser executado com um comando shell

GNU Make

- Pode ser usado para descrever qualquer tarefa em que alguns arquivos precisam ser atualizados automaticamente a partir de outros sempre que houver alterações nesses outros
  - Não se limita à construção de programas!

Arquivo Makefile

- Descreve os relacionamentos entre os arquivos e indica comandos para atualizar cada arquivo
- O Make usa as informações contidas no Makefile e os horários da última modificação dos arquivos para decidir quando um arquivo precisa ser atualizado
- O próprio Makefile indica como a atualização deve ser feita

Exemplo: Makefile de um programa em C

- O arquivo executável é construído a partir de arquivos objetos que, por sua vez, são gerados a partir de código-fonte
- O Makefile pode indicar como os fontes são compilados e ligados para gerar o executável
- Uma vez que um Makefile correto exista, basta executar o comando make para que todas as (re)compilações necessárias para a criação/atualização do programa sejam executadas

Introdução a Makefiles

Um Makefile é composto por regras do tipo:

```
alvo ... : pré-requisitos ...
    receita ... ...
```

- **Alvo**: geralmente é o nome de um arquivo a ser gerado
  - Mas pode ser também o nome de uma ação a ser executada
- **Pré-requisitos**: arquivos usados como entrada na criação do alvo
- **Receita**: ação que o make executará
  - Pode possuir um ou mais comandos (na mesma linha ou um em cada linha)
  - Cada linha da receita precisa ser iniciada por um **tab**

```
all : testarand testamoeda testadado
# Arquivos objeto
testarand.o : testarand.c
	gcc -Wall -L. -c testarand.c
testamoeda.o : testamoeda.c
	gcc -Wall -L. -c testamoeda.c
testadado.o : testadado.c aleatorio.h
	gcc -Wall -L. -c testadado.c
aleatorio.o : aleatorio.c aleatorio.h
	gcc -Wall -L. -c aleatorio.c
# Bibliotecas
libaleatorio.a : aleatorio.o
	ar rcv libaleatorio.a aleatorio.o
# Executaveis
testarand : testarand.o
	gcc -Wall -L. -o testarand testarand.o
testamoeda : testamoeda.o
	gcc -Wall -L. -o testamoeda testamoeda.o
testadado : testadado.o aleatorio.h libaleatorio.a
	gcc -Wall -L. -o testadado testadado.o -laleatorio
```

Por padrão, o Make começa com o primeiro alvo do makefile. Ele é a meta padrão (*default goal*).

Uma regra do makefile só é processada de forma automática pelo Make se seu alvo é um pré-requisito direto ou indireto da meta

Exemplo de arquivo Makefile

Funcionamento do Make

- Quando o alvo é um arquivo, ele precisa ser atualizado se um dos seus pré-requisitos mudam

- Um pré-requisito que é automaticamente gerado (ou seja, que é o alvo de uma regra também) deve ser atualizado antes de ser usado

```
PROGRAMAS = testarand testamoeda testadado
CC = gcc                    # compilador C usado
CFLAGS = -Wall -L.          # parametros para o compilador
all : $(PROGRAMAS)
# Arquivos objeto
testarand.o : testarand.c
	$(CC) $(CFLAGS) -c testarand.c
testamoeda.o : testamoeda.c
	$(CC) $(CFLAGS) -c testamoeda.c
testadado.o : testadado.c aleatorio.h
	$(CC) $(CFLAGS) -c testadado.c
aleatorio.o : aleatorio.c aleatorio.h
	$(CC) $(CFLAGS) -c aleatorio.c
# Bibliotecas
libaleatorio.a : aleatorio.o
	ar rcv libaleatorio.a aleatorio.o
# Executaveis
testarand : testarand.o
	$(CC) $(CFLAGS) -o testarand testarand.o
testamoeda : testamoeda.o
	$(CC) $(CFLAGS) -o testamoeda testamoeda.o
testadado : testadado.o aleatorio.h libaleatorio.a
	$(CC) $(CFLAGS) -o testadado testadado.o -laleatorio
```

Definição de variáveis

Variáveis nos ajudam a evitar replicações no makefile. Elas ajudam a manter a consistência e diminuem a possibilidade de erros na manutenção.

Exemplo de arquivo Makefile

Regras implícitas

- Quando o alvo é um arquivo .o, o Make é capaz de deduzir que ele deve usar o arquivo .c correspondente para gerá-lo usando um comando 'gcc -c'
  - Podemos omitir as receitas para a geração de arquivos objeto
- Exemplo de regra que pode ser omitida no makefile

```
testarand.o : testarand.c
    gcc -c testarand.c
```

- Exemplo de regra válida no makefile

```
testadado.o : aleatorio.h
```

Note que o arquivo .c usado para a geração do .o é deduzido

```
PROGRAMAS = testarand testamoeda testadado
CC = gcc                    # compilador C usado
CFLAGS = -Wall -L.          # parametros para o compilador
all : $(PROGRAMAS)
clean :
	rm -f *.o *.a $(PROGRAMAS)
# Arquivos objeto
testadado.o : aleatorio.h
aleatorio.o : aleatorio.h
# Bibliotecas
libaleatorio.a : aleatorio.o
	ar rcv libaleatorio.a aleatorio.o
# Executaveis
testarand : testarand.o
	$(CC) $(CFLAGS) -o testarand testarand.o
testamoeda : testamoeda.o
	$(CC) $(CFLAGS) -o testamoeda testamoeda.o
testadado : testadado.o aleatorio.h libaleatorio.a
	$(CC) $(CFLAGS) -o testadado testadado.o -laleatorio
```

Regra falsa
(não tem pré-requisitos)

Regras implícitas
(omitem a receita)

Exemplo de arquivo makefile

Regras implícitas

- Quando os objetos de um makefile são criados por regras implícitas, é possível usar um outro estilo de definição
- Nesse estilo de makefile, as entradas são agrupadas de acordo com seus pré-requisitos (e não por seus alvos)
- Exemplo:

```
main.o : defs.h
kbd.o : defs.h command.h
command.o : defs.h command.h
display.o : defs.h buffer.h
insert.o : defs.h buffer.h
search.o : defs.h buffer.h
files.o : defs.h buffer.h command.h
utils.o : defs.h
```

```
main.o kbd.o command.o display.o insert.o \
search.o files.o utils.o : defs.h
kbd.o command.o files.o : command.h
display.o insert.o search.o files.o :
buffer.h
```

```
PROGRAMAS = testarand testamoeda testadado
CC = gcc                    # compilador C usado
CFLAGS = -Wall -L.          # parametros para o compilador
all : $(PROGRAMAS)
clean :
      rm -f *.o *.a $(PROGRAMAS)
# Arquivos objeto
testadado.o aleatorio.o: aleatorio.h
# Bibliotecas
libaleatorio.a : aleatorio.o
      ar rcv libaleatorio.a aleatorio.o
# Executaveis
testarand : testarand.o
      $(CC) $(CFLAGS) -o testarand testarand.o
testamoeda : testamoeda.o
      $(CC) $(CFLAGS) -o testamoeda testamoeda.o
testadado : testadado.o aleatorio.h libaleatorio.a
      $(CC) $(CFLAGS) -o testadado testadado.o -laleatorio
```

Regras implícitas agrupadas por pré-requisito

Exemplo de arquivo makefile

Regras falsas

- As regras falsas possibilitam a execução de comandos não relacionados à compilação
- Exemplo de remoção dos objetos, bibliotecas e progs

```
clean :
    rm -f *.o *.a $(PROGRAMAS)
```

- Como clean não é um pré-requisito para o alvo `all`, sua regra nunca será executada com a chamada de make sem parâmetros
  - Ela só será executada quando for invocada explicitamente, com o comando `make clean`

Regras falsas

- No exemplo da regra clean, se um arquivo com o nome clean fosse criado no diretório do makefile, a receita da regra nunca mais seria executada
  - Como regra falsa não tem pré-requisitos, o arquivo alvo criado “externamente” estaria sempre atualizado
- Solução: declarar explicitamente que o alvo da regra é falso usando o alvo especial .PHONY

```
.PHONY : clean
clean :
    -rm -f *.o *.a $(PROGRAMAS)
```

O caractere ‘-’ força que o make continue executando mesmo que a execução do comando retorne erros

Variáveis automáticas

- **$<** o nome do primeiro pré-requisito da regra
- **$@** o nome do alvo
- **$?** a lista de nomes de todos os pré-requisitos mais novos que o alvo
- **$^** a lista de nomes de todos os pré-requisitos
- $* string que “casou” com o % no alvo
- Exemplo de uso:

```
testarand : testarand.o
    gcc -o $@ $^
```

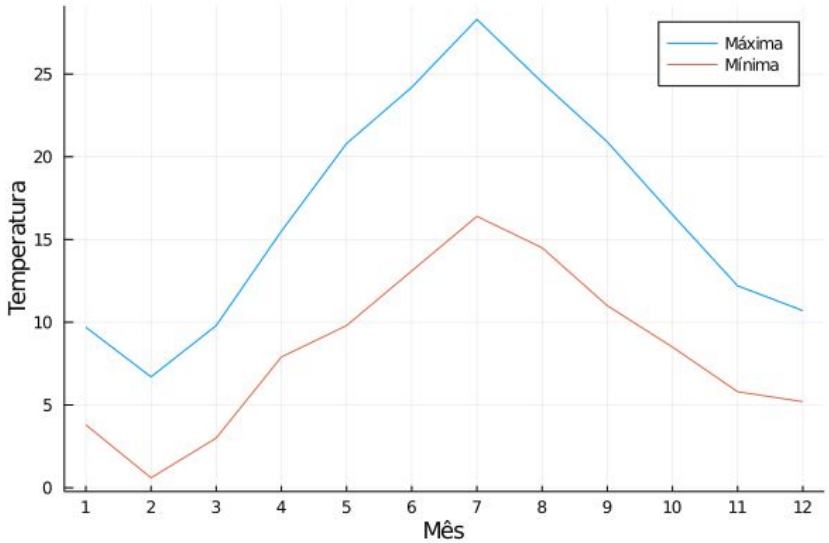
Outro exemplo usando o GNU Make

Relatório sobre Variação de Temperatura ao longo do Ano

- Relatório (documento) escrito em LaTeX contendo um gráfico gerado por um programa escrito na linguagem Julia a partir de dados de temperatura armazenados num arquivo CSV

Arquivo CSV

```
Year,Month,Tmax,Tmin,Rain,Sun
2018,1,9.7,3.8,58,46.5
2018,2,6.7,0.6,29,92.0
2018,3,9.8,3,81.2,70.3
2018,4,15.5,7.9,65.2,113.4
2018,5,20.8,9.8,58.4,248.3
2018,6,24.2,13.1,0.4,234.5
2018,7,28.3,16.4,14.8,272.5
2018,8,24.5,14.5,48.2,182.1
...
```



Arquivo makefile

```
relatorio.pdf : relatorio.tex grafico_temperaturas.png
   pdflatex relatorio.tex

grafico_%.png : %.csv gera_grafico.jl
   julia gera_grafico.jl -i $*.csv -o $@
```

Neste exemplo,
$* = “Temperaturas”
$@ = “grafico_temperaturas.png”
porque o alvo “grafico_%.png” casa com o pré-requisito “grafico_temperaturas.png”

Arquivo relatorio.tex

```
\documentclass{article}
\usepackage{graphicx}
\title{Variação da Temperatura ao Longo do Ano}

\begin{document}
\maketitle

O gráfico da Figura \ref{grafico_temperaturas} mostra as temperaturas máximas e
mínimas para cada mês do ano analisado.

\begin{figure}[h]
\includegraphics[scale=0.6]{grafico_temperaturas.png}
\caption{Temperaturas máximas e mínimas para cada mês do ano.}
\label{grafico_temperaturas}
\end{figure}

\end{document}
```

Código fonte em LaTeX do documento do relatório

Inclusão da imagem do gráfico no documento

Arquivo gera_grafico.jl

Programa em Julia para a criação do gráfico

```
using CSV, DataFrames, Plots, ArgParse
# define os parâmetros de linha de comando do programa
s = ArgParseSettings()
@add_arg_table! s  begin
   "--entrada", "-i"          # nome do arquivo (.csv) de entrada
       required =  true
   "--saida", "-o"            # nome do arquivo (.png) de saída
       required =  true
end
# faz o parsing dos argumentos do programa
parsed_args = parse_args(s)
# lê os dados do arquivo CSV de entrada
dados = DataFrame!(CSV.File(parsed_args[ "entrada"]))
# gera o gráfico
plot(dados.Month,[dados.Tmax, dados.Tmin], xticks =  0:1:13,
   xlabel = "Mês", ylabel = "Temperatura", labels = ["Máxima" "Mínima"])
# grava o gráfico num arquivo
savefig(parsed_args[ "saida"])
```

Geração automática de dependências

Existem ferramentas que geram automaticamente regras de dependências entre arquivos e até mesmo makefiles completos

Exemplos:

- gcc -M e gcc -MM
  https://gcc.gnu.org/onlinedocs/gcc/Preprocessor-Options.html

- GNU Automake http://www.gnu.org/software/automake/

Referências

- GNU Make Manual
  https://www.gnu.org/software/make/manual/
- Curso *The Missing Semester of Your CS Education* - MIT
  Assunto: Metaprogramming
  https://missing.csail.mit.edu/2020/metaprogramming/
- Notas das aulas de MAC0211 feitas pelo Prof. Kon
  http://www.ime.usp.br/~kon/MAC211