INF 213 – Estrutura de Dados
Prof. Marcus V. A. Andrade

# Utilitário *Make*

# Utilitário de controle de dependências de arquivo: *make*

- Utilizado para controlar automaticamente a dependência entre os vários arquivos (módulos) que compõem um programa.
- Determina quais módulos do programa devem ser recompilados quando um outro módulo é modificado.
- Não é exclusivo para programas. Pode ser utilizado em quaisquer situações em que alguns arquivos devem ser atualizados a partir de outros, sempre que estes sejam alterados.

# Utilitário de controle de dependências de arquivo: *make*

- Para usar o *make* devemos criar um arquivo chamado *Makefile* (ou *makefile*) descrevendo os relacionamentos entre os arquivos e fornecendo os comandos para atualizar cada arquivo.
- Um arquivo *Makefile* é composto por uma sequência de *regras*, sendo que cada regra tem a seguinte sintaxe:

*objetivos* : *dependências*

<TAB> *comando*

ou

*objetivos* : *dependências* ; *comando*

<TAB> *comando*

# Utilitário de controle de dependências de arquivo: *make*

- Um *objetivo* é normalmente o nome de um arquivo que é gerado por algum programa. Exemplos de objetivos são arquivos executáveis ou objetos. Um objetivo pode ser também o nome de uma ação a ser executada, como, por exemplo, *clean*.

- Uma *dependência* é um arquivo que é usado como entrada para criar o objetivo correspondente. Um objetivo frequentemente depende de diversos arquivos.

# Utilitário de controle de dependências de arquivo: *make*

- Um *comando* é uma ação que o *make* executa. Uma regra pode ter mais de um comando, cada um em uma linha. É fundamental observar que a linha descrevendo um comando deve necessariamente iniciar com um *tab*.
- Uma regra indica ao *make* duas coisas: quando os objetivos estão desatualizados e como atualizá-los, se for o caso.
- O critério de estar desatualizado é dado em termos das dependências que consistem em nomes de arquivos separados por espaços. Um objetivo está desatualizado se ele não existe, ou se ele é mais antigo que qualquer de suas dependências.

# Utilitário de controle de dependências de arquivo: *make*

- **Exemplo:** Considere o programa *TesteFiguras* visto na aula prática que usa as classes *FigBase*, *Circulo* e *Retangulo*

- Uma forma de gerar o executável deste programa é fornecer a seguinte linha de comando (tudo numa mesma linha):

  g++ -o *TesteFiguras.exe FigBase.cpp Retangulo.cpp Circulo.cpp TesteFiguras.cpp*

- Para contornar os inconvenientes deste método de geração de programas executáveis, podemos criar o seguinte arquivo *Makefile:*

# Utilitário de controle de dependências de arquivo: *make*

```
# Makefile para gerar o executavel para o teste das classes
# FigBase, Retangulo, Circulo e Segmento
# Criado por Marcus Vinicius Alvim Andrade em 02/04/11

all: FigBase.o Retangulo.o Circulo.o Segmento.o TestaFiguras.o
   g++ -o TestaFiguras.exe FigBase.o Retangulo.o Circulo.o Segmento.o TestaFiguras.o

FigBase.o: FigBase.h FigBase.cpp
   g++ -c FigBase.cpp

Retangulo.o: Retangulo.cpp FigBase.h Retangulo.h
   g++ -c Retangulo.cpp

Circulo.o: Circulo.cpp FigBase.h Circulo.h
   g++ -c Circulo.cpp

Segmento.o: Segmento.cpp FigBase.h Segmento.h
        g++ -c Segmento.cpp

TestaFiguras.o: TestaFiguras.cpp FigBase.h Retangulo.h Circulo.h Segmento.h
        g++ -c TestaFiguras.cpp
```

# Utilitário de controle de dependências de arquivo: *make*

- Para usar este *Makefile* para criar o arquivo executável *TestaFiguras.exe*, dê o comando

  make

- De uma maneira mais geral, um arquivo *Makefile* pode conter cinco tipos de itens:

1. Uma *regra explícita* diz quando e como refazer um ou mais arquivos, chamados objetivos da regra. Ela lista os outros arquivos de que os objetivos dependem; pode também dar comandos para criar ou atualizar os objetivos.

# Utilitário de controle de dependências de arquivo: *make*

2. Uma *regra implícita* diz quando e como refazer uma classe de arquivos baseado em seus nomes. Ela descreve como um objetivo pode depender de um arquivo com um nome semelhante ao objetivo e dá comandos para criar ou atualizar tal objetivo.

2. Uma *definição de variável* é uma linha que especifica um valor para uma variável consistindo em um texto que pode substituir qualquer ocorrência da variável mais tarde no makefile.

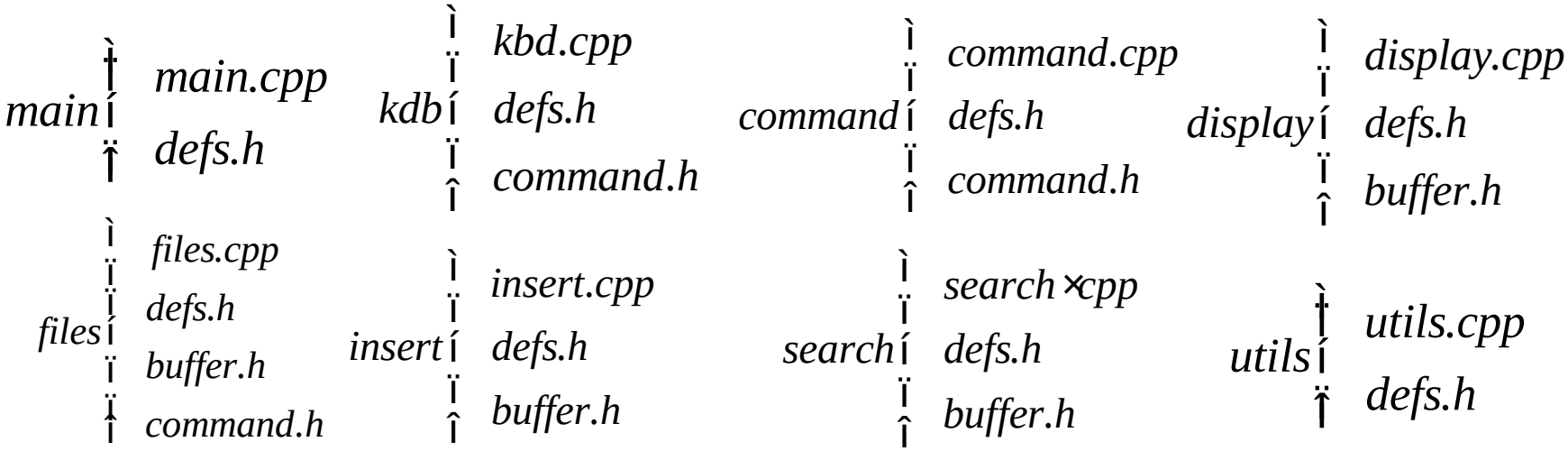
# Utilitário de controle de dependências de arquivo: *make*

4. Uma *diretiva* é um comando para o make executar algo especial durante o processamento do makefile. Pode ser:
   - Ler outro makefile;
   - Decidir (baseado nos valores de variáveis) usar ou ignorar uma parte do makefile;
   - Definir uma variável a partir de uma cadeia contendo múltiplas linhas.

5. O símbolo # em uma linha de um makefile inicia um comentário.

# Utilitário de controle de dependências de arquivo: *make*

- Um exemplo mais elaborado: suponha que queremos produzir um *Makefile* para gerenciar a compilação de um programa denominado edit que é composto por 8 módulos (objetos) denominados: *main*, *kbd*, *command*, *display*, *insert*, *search*, *files* e *utils*, sendo que cada um desses módulos são formados por:



# Utilitário de controle de dependências de arquivo: *make*

```
#
# Makefile para a geração do programa edit
#
edit: main.o kbd.o command.o display.o \
      insert.o search.o files.o utils.o
            g++ -o edit main.o kbd.o command.o display.o \
            insert.o search.o files.o utils.o

main.o: main.cpp defs.h
            g++ -c main.cpp

kbd.o: kbd.cpp defs.h command.h
            g++ -c kbd.cpp

command.o: command.cpp defs.h command.h
            g++ -c command.cpp

display.o: display.cpp defs.h buffer.h
            g++ -c display.cpp
```

# Utilitário de controle de dependências de arquivo: *make*

```
insert.o: insert.cpp defs.h buffer.h
        g++ -c insert.cpp

search.o: search.cpp defs.h buffer.h
        g++ -c search.cpp

files.o: files.cpp defs.h buffer.h command.h
        g++ -c files.cpp

utils.o: utils.cpp defs.h
        g++ -c utils.cpp

clean:
        rm edit main.o kbd.o command.o display.o \
    insert.o search.o files.o utils.o
```

# Utilitário de controle de dependências de arquivo: *make*

- Para simplificar a construção do *Makefile* (evitando repetições) podemos usar *variáveis* que permitem definir uma cadeia uma única vez e substituí-la textualmente em vários locais a partir daí. Por exemplo, o *Makefile* anterior poderia ser reescrito da seguinte forma:

```
#
# Makefile para o programa edit usando variável
#
objects = main.o kbd.o command.o display.o \
          insert.o search.o files.o utils.o

edit: $(objects)
        g++ -o edit $(objects)
```

# Utilitário de controle de dependências de arquivo: *make*

```
main.o: main.cpp defs.h
        g++ -c main.cpp

kbd.o: kbd.cpp defs.h command.h
        g++ -c kbd.cpp

command.o: command.cpp defs.h command.h
        g++ -c command.cpp

display.o: display.cpp defs.h buffer.h
        g++ -c display.cpp

insert.o: insert.cpp defs.h buffer.h
        g++ -c insert.cpp

search.o: search.cpp defs.h buffer.h
        g++ -c search.cpp
```

# Utilitário de controle de dependências de arquivo: *make*

```
files.o: files.cpp defs.h buffer.h command.h
        g++ -c files.cpp

utils.o: utils.cpp defs.h
        g++ -c utils.cpp

clean:
        rm edit $(objects)
```

- Para mais informações sobre o GNU Make, leia as informações disponíveis no *Linux*. Para isso digite:

```
info –-file make.info
```

- Para sair do info, digite q. Para aprender usar o info, digite h.