18.2 - Make

- Make é um programa que mantém e atualiza programas e arquivos dependentes
- Os mesmos comandos utilizados em linha de comando podem ser utilizados dentro do arquivo de entrada do make

- Vantagem:
  - Se uma função de um arquivo é modificada, somente este arquivo é recompilado.

- Funcionamento básico:
  - Para cada arquivo é verificado se a data de atualização é posterior a data de compilação.
  - Se for posterior, compila novamente; caso contrário, não.

18.2.1 - Utilização

- Para utilizar o make, deve-se construir um arquivo textual de controle chamado makefile ou Makefile e a partir da linha de comando digitar **make**:

  > make

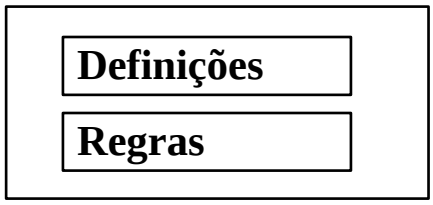
- make procura no diretório corrente o arquivo de nome makefile ou Makefile e executa ações segundo as instruções desse arquivo.

  > make -f Mymakefile

  - Neste caso o arquivo Mymakefile é executado e não o default

18.2.2 - Formato do arquivo de controle

- Um arquivo makefile tem as seguintes partes:
  - macros (definições): são opcionais, isto é, um arquivo makefile pode não conter definições
  - regras: parte obrigatória
  - comentários: também são opcionais



<destino> : dependencias
TAB <comandos>

  - <destino> depende de <dependencias> e é obtido através de <comandos> é o nome da regra, e é chamado de target. O comando make irá construir o primeiro objetivo do arquivo Makefile, a não ser que seja especificado outro na linha de comando.

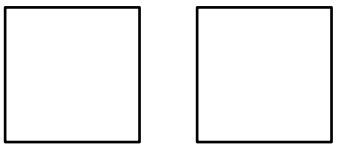
  - makefile contém a descrição textual de como os <destinos> são obtidos em função das dependencias.

  - As várias dependencias são separadas por um ou mais espaços em branco ou por <TAB>s

- Os comandos devem estar separados do inicio da linha por um ou mais <TAB>s (obrigatoriamente)

- Cada linha será executada numa shell independente, linhas consecutivas devem ser unidas pelo caracter de continuação de linha \
- O programa make verifica recursivamente todas as dependências para cada destino, verificando se o mesmo deve ou não ser atualizado.

- Ex.1: Supor os seguintes arquivos:

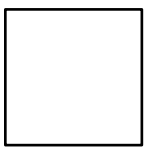


teste.c teste.h

```
teste:   teste.c  teste.h
         gcc teste.c  -o teste
```

makefile

- Ex.2: Supor os seguintes arquivos:



```
#include<arvore.h>
```

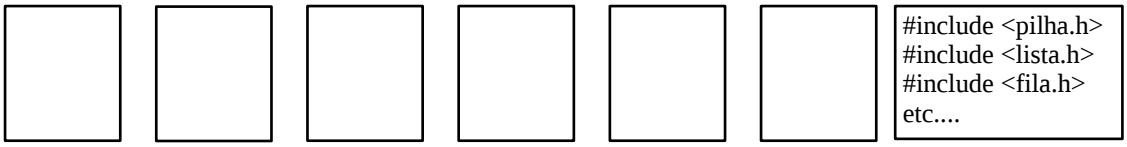
```
#include <math.h>
#include <arvore.h>
etc....
```

arvore.h arvore.c main.c

```
arvore: arvore.o  main.o
	gcc -lm -o arvore arvore.o main.o
arvore.o: arvore.c arvore.h
	gcc -c arvore.c
main.o: main.c arvore.h
	gcc -c main.c
```

makefile

- Ex.3: Supor os seguintes arquivos:



pilha.c pilha.h lista.c lista.h fila.c fila.h main.c

```
lislig: pilha.o fila.o lista.o main.o
	gcc pilha.o fila.o lista.o main.o -o lislig
pilha.o: pilha.c  pilha.h
	gcc -c pilha.c
fila.o: fila.c  fila.h
	gcc -c fila.c
lista.o: lista.c  lista.h
	gcc -c lista.c
main.o: main.c pilha.h lista.h  fila.h
	gcc -c main.c
```

- Comentários: usar #
  - Exemplos:

```
# um comentário no inicio da  linha
XX = 5    # comentário no fim
```

- Continuação de linha: \
  - Exemplos:

```
YY = abc  #  comentário muito comprido \
            continua na outra linha
```

Exemplo:

```
/* arquivo frase.h colocado em um diretório de include */
#include <stdio.h>
int tamanho (char *);
void inverte1 (char *);
void inverte2 (char *);
```

```
/* arquivo inverte1.c  */
void inverte1 (char *frase)
{ int i;
  char * pfrase = frase;
  while (*pfrase != '\0')
     putchar (*pfrase++);
  putchar ('\n');
  while (--pfrase >= frase)
     putchar (*pfrase);
  putchar ('\n');
}
```

```
/* arquivo inverte2.c  */
#include "include/frase.h"
void inverte2 (char *frase)
{ int i;
  for (i=0; i<= (tamanho(frase)-1); i++)
      putchar (*(frase+i));
  putchar ('\n');
  for (i=tamanho(frase)-1; i>=0; i--)
      putchar (frase[i]);
 putchar ('\n');
}

/* arquivo tamanho.c  */
int tamanho (char *str)
{
   int i;
   for ( i=0; str[i]!='\0';  i++);
   return(i+1);
}

/* arquivo main.c  */
#include "include/frase.h"
main ()
{
  char frase[] = "oi, como vai?";
  printf ("%d\n", tamanho(frase));
  inverte1(frase);
  inverte2(frase);
}
```

Makefile 1

```
f rase: main.o tamanho.o inverte1.o inverte2.o
	gcc -ofrase main.o tamanho.o inverte1.o inverte2.o
main.o:  main.c include/frase.h
	gcc -c main.c
tamanho.o:   tamanho.c include/frase.h
	gcc -c tamanho.c
inverte1.o:  inverte1.c include/frase.h
	gcc -c inverte1.c
inverte2.o: inverte2.c include/frase.h
	gcc -c inverte2.c
```

Makefile 2

```
# o all é geralmente utilizado para marcar a primeira regra

all: inicial frase clean
# no caso de inicial e clean o nome da regra (ou target)
# estão associados a uma acao a ser executada

inicial:
	@echo "Iniciando"

frase: main.o tamanho.o inverte1.o inverte2.o
	gcc -ofrase main.o tamanho.o inverte1.o inverte2.o
main.o:  main.c include/frase.h
	gcc -c main.c
tamanho.o:   tamanho.c include/frase.h
	gcc -c tamanho.c
inverte1.o:  inverte1.c include/frase.h
	gcc -c inverte1.c
inverte2.o: inverte2.c include/frase.h
	gcc -c inverte2.c
```

```
#remove arquivos objetos
clean:
	@echo "Finalizando"
	@rm -rf *.o    #@ faz com que o comando não apareça
```

- Macros: Servem para definir caminhos, opções de compilação,

```
DIRETORIO = usr/prof/silvia/ED/arvores
LIBD = lib/obj
LIBFLAGS = -lm -lg
```

Para se utilizar as macros definidas, deve-se colocar o símbolo $ antes.

#Makefile 3

```
# definindo variaveis
CC=@gcc
INC=./include/
OBJ=main.o tamanho.o inverte1.o inverte2.o

all: inicial frase clean

inicial:
	@echo "Iniciando"

frase: $(OBJ)
	$(CC) -ofrase $(OBJ)

main.o: $(INC)frase.h main.c
	$(CC) -c main.c
```

```
tamanho.o:  $(INC)frase.h tamanho.c
            $(CC) -c tamanho.c

inverte1.o: $(INC)frase.h inverte1.c
            $(CC) -c inverte1.c

inverte2.o: $(INC)frase.h inverte2.c
            $(CC) -c inverte2.c

#remove arquivos objetos
clean:
            @echo "Finalizando"
            @rm -rf *.o    #@ para comoando não aparecer
```

- Substituição de sufixos: $(var: string1 = string2)
  - no valor de var, todas as ocorrências de string1, serão substituídas pela string2
  - string1 é um sufixo de var.

    - Exemplo1:

      Sejam as seguintes definições:

      ARQH = pilha.h lista.h   fila.h

      e

      ARQO = $(ARQH:h=o)

      então $(ARQO) equivale a

      pilha.o lista.o fila.o

- Exemplo2:

Sejam as seguintes definições:

XX = a.c b.c c.c

e

YY = $(XX:.c=_x.o)

então $(YY) equivale a

a_x.o b_x.o c_x.o

#Makefile 4

```
# definindo variaveis
CC=@gcc
INC=./include/
SRC=$(wildcard *.c)
# função wildcard: retorna todos os arquivos do diretório
#SRC = todos os arquivos .c do diretorio


all: inicial frase clean
inicial:
	@echo "Iniciando"

#expande SRC e substitui os.c por arquivos de mesmo nome.o
frase: $(SRC:.c=.o)
	$(CC) -ofrase $(SRC:.c=.o)

main.o: $(INC)frase.h main.c
	$(CC) -c main.c

tamanho.o:  $(INC)frase.h tamanho.c
	$(CC) -c tamanho.c
```

```
inverte1.o: $(INC)frase.h inverte1.c
        $(CC) -c inverte1.c

inverte2.o: $(INC)frase.h inverte2.c
        $(CC) -c inverte2.c

#remove arquivos objetos
        clean:
            @echo "Finalizando"
            @rm -rf *.o
```

- Variável dinâmica (interna):

```
# S@  nome da regra  corrente
# S^    dependência anterior
# S<    pega o primeiro valor da lista de dependências anterior.
# $?    lista de dependências mais recentes que a regra.
# $*    nome do arquivo sem sufixo
```

$@

- Toma o valor do destino (regra) corrente.
- Exemplo: Os seguintes arquivos makefile são equivalentes:

```
xxx: yyy.c zzz.c
        gcc -o xxx   yyy.c   zzz.c
```

```
xxx: yyy.c zzz.c
        gcc -o $@  yyy.c   zzz.c
```

- Substituição de padrões: O caractere '%' é usado para casamento de padrões para objetivos

  $(var: op % os = np % ns)
  - no valor de var, todas as ocorrências do prefixo op, serão substituídas por np
  - todas as ocorrências do sufixo os serão substituídas por ns.

  - Exemplo:
    Sejam as seguintes definições:
    XX = xx_a.c xx_b.c xx_c.c
    e
    YY = $(XX:xx_%c = %o)

    então $(YY) equivale à
    a.o b.o c.o

#Makefile 5

```
# definindo variaveis
CC=@gcc
INC=./include/
SRC=$(wildcard *.c)  #SRC = todos os arquivos .c do diretorio

all: inicial frase clean
inicial:
	@echo "Iniciando"

#expande SRC e substitui os.c por arquivos de mesmo nome.o
```

```
#variaves internas
# S@ nome da regra  ==  frase
# S^ dependencia anterior == $(SRC:.c=.o)
# o comando abaixo pode ser então reescrito
# frase: $(SRC:.c=.o)
#       $(CC) -ofrase $(SRC:.c=.o)

frase: $(SRC:.c=.o)
	$(CC) -o $@ $^

main.o: $(INC)frase.h main.c
	$(CC) -c main.c

tamanho.o:  $(INC)frase.h tamanho.c
	$(CC) -c tamanho.c

inverte1.o: $(INC)frase.h inverte1.c
	$(CC) -c inverte1.c

inverte2.o: $(INC)frase.h inverte2.c
	$(CC) -c inverte2.c

clean:
	@echo "Finalizando"
	@rm -rf *.o
```

#Makefile 6

```
# definindo variaveis
CC=gcc
INC=./include/
SRC=$(wildcard *.c)  #SRC = todos os arquivos .c do diretorio

all: inicial frase clean
inicial:
	@echo "Iniciando"
#expande SRC e substitui os.c por arquivos de mesmo nome.o

#variaves internas
# S@ nome da regra  ==  frase
# S^ dependencia anteriror == $(SRC:.c=.o)
# S< pega o primeiro valor da lista de dependencias anterior.
# $?     Lista de dependências mais recentes que a regra.
# $*     Nome do arquivo sem sufixo
# o comando abaixo pode ser então reescrito
# frase: $(SRC:.c=.o)
#	$(CC) -ofrase $(SRC:.c=.o)

frase: $(SRC:.c=.o)
	$(CC) -o $@ $^

# % indica um padrão, fazer para todo .o e .c
%.o: %.c $(INC)frase.h
	$(CC) -c $< -o $@

#remove arquivos objetos
clean:
	@echo "Finalizando"
	@rm -rf *.o
```