Material Extra (básico)

Objetivo: Apresentar a criação e uso de bibliotecas.

1) Criando uma biblioteca

Em um arquivo, coloque apenas os protótipos (primeira linha de declaração seguido por um ;) de todas as funções que farão parte da biblioteca (main não entra aqui). Grave este arquivo como `minhabib.h`, por exemplo.

Agora, em outro arquivo, repita as funções, mas desta vez incluindo suas definições (corpo das funções). Salve este arquivo como `minhabib.c` (mesmo nome do `.h`), no mesmo diretório de antes. Inclua, neste arquivo a diretiva `#include "minhabib.h"`. Isso diz ao compilador: inclua as definições contidas no arquivo `minhabib.h` que está no mesmo diretório deste arquivo que está sendo compilado.

Note também que nem `minhabib.h` nem `minhabib.c` possuem uma `main()`.

2) Usando a biblioteca

Em um novo arquivo, digite seu programa, salvando-o como `prog.c`, por exemplo. Inclua a diretiva `#include "minhabib.h"`.

Para compilar:

```
gcc prog.c minhabib.c -o prog
```

Isso gerará o executável `prog`, que usará sua biblioteca. Para executar:

```
./prog
```

3) No dev-C++

No dev-C++ o procedimento é um pouco mais complicado. Em vez de criar um novo arquivo-fonte, você deve criar um novo projeto. Só então você cria um novo arquivo-fonte, adicionando ao projeto.

Neste novo arquivo-fonte digite o `prog.c`. Agora vá no menu *projeto → opções de projeto*. No tab *Parâmetros*, clique em *Adicionar*. Busque o arquivo `minhabib.c` (com *.* a janela mostra a listagem de todos os arquivos) e o adicione.

Pronto! E só compilar. Quando você quiser usar novamente a biblioteca, é só seguir estes passos e pronto, sem precisar copiar o código da biblioteca no seu programa.

4) Makefile

O que é? O `Makefile` é um arquivo para configuração de compilação utilizado pelo programa `make`, cuja idéia é simplificar e agilizar a compilação de programas. Em outras palavras, o `Makefile` é um arquivo de texto responsável por dizer ao programa `make` "o que fazer" e contém o relacionamento entre os arquivos fonte, objeto (pré-executáveis) e executáveis.

Por que usar? Evita a compilação de arquivos desnecessários. Por exemplo, se seu programa utiliza 120 bibliotecas e você altera apenas uma, o `Makefile` descobre (comparando as datas de alteração dos arquivos fontes com as dos arquivos anteriormente compilados) qual arquivo foi alterado e compila apenas a biblioteca necessária.

Como usar? Para utilizar o `Makefile` basta criar o arquivo com o nome `Makefile` no diretório onde se encontram os arquivos fonte para compilação e executar o programa `make` no mesmo diretório. O arquivo pode ser escrito usando qualquer editor de texto puro. A sintaxe será abordada a seguir.

O `Makefile` funciona de acordo com regras, a sintaxe de uma regra é:

regra: dependencias
comandos

Antes de executar os comandos de uma regra, o programa `make` verifica se todas as dependências foram satisfeitas. Uma dependência pode ser outra regra ou então algum arquivo necessário para execução dos comandos. Por exemplo, a regra de compilação de um executável pode ter como dependência as regras que compilam as bibliotecas necessárias e também os arquivos fonte necessários. A regra pode ter qualquer nome. Por exemplo, o nome da regra que compila o arquivo `prog.c` pode ser `prog`.

No caso da dependência ser outra regra, a regra da dependência é avaliada antes da regra dependente. Dependências? Regras? Comandos?

Imagine uma regra como sendo uma receita e as dependências como os ingredientes da receita. Além disso você precisa realizar algumas ações (comandos) para que a receita fique pronta.

Exemplo:

`prog: minhabib.o prog.o`
a regra prog para ser gerada depende de `minhabib.o` e `prog.o`

`prog.o: prog.c minhabib.h`
a regra `prog.o` para ser gerada depende de `prog.c` e `minhabib.h`

`minhabib.o: minhabib.c minhabib.h`
a regra `minhabib.o` para ser gerada depende de `minhabib.c` e `minhabib.h`

Lembrando: `prog.c` é o código fonte do programa principal (contém a `main()`).
`prog` é o código executável.
`minhabib.c` contém as definições das funções.
`minhabib.h` contém os protótipos das funções.
`minhabib.o` e `prog.o` são arquivos pré-executáveis.

Dica: no diretório contendo estes arquivos, digite no terminal o seguinte comando:

```
gcc prog.c minhabib.c -MM
```

e as dependências e regras associadas a `prog.c` e `minhabib.c` serão geradas:

```
prog.o: prog.c minhabib.h
minhabib.o: minhabib.c minhabib.h
```

aí só basta incluir a regra que gera o executável: `prog: minhabib.o prog.o`

Continuando com nosso exemplo. Nosso `Makefile` ficará assim:

```
prog: minhabib.o prog.o
     gcc prog.o minhabib.o -o prog -Wall -lm

minhabib.o: minhabib.c minhabib.h
     gcc minhabib.c -c -Wall -lm

prog.o: prog.c minhabib.h
     gcc prog.c -c -Wall -lm

clean:
     rm prog *.o
```

Passo a passo:

```
prog: minhabib.o prog.o
     gcc prog.o minhabib.o -o prog -Wall -lm
```

A geração da regra `prog` (código executável) depende dos pré-executáveis `minhabib.o` e `prog.o`. O comando `gcc prog.o minhabib.o -o prog -Wall -lm` indica que os pré-executáveis serão compilados e para gerar o executável.

```
minhabib.o: minhabib.c minhabib.h
     gcc minhabib.c -c -Wall -lm
```

A geração da regra `minhabib.o` (pré-executável) depende de `minhabib.c minhabib.h`. O comando `gcc minhabib.c -c -Wall -lm` indica que para gerar o pré executável, `minhabib.c` deve ser compilada (flag `-c` para gerar pré-executáveis).

```
prog.o: prog.c minhabib.h
     gcc prog.c -c -Wall -lm
```

A geração da regra `prog.o` (pré-executável) depende de `prog.c minhabib.h`. O comando `gcc prog.c -c -Wall -lm` indica que para gerar o pré executável, `prog.c` deve ser compilado (flag `-c` para gerar pré-executáveis).

Finalmente, para compilar e executar nosso programa juntamente com as bibliotecas basta digitar, na linha de comando (dentro do diretório onde se encontram todos os arquivos):

```
make
./prog
```

Um caso interessante é quando se chama o `make` duas vezes consecutivas. Se na primeira chamada à compilação ocorreu sem problemas, na segunda chamada o programa não é compilado, ou seja, o `make` percebe que não é necessário compilar o programa novamente.

```
clean:
	rm prog *.o
```

A regra clean não tem depências e quando executada ela apaga todos os arquivos gerados na compilação (`prog prog.o minhabib.o`). Para executá-la basta digitar na linha de comando:

```
make clean
```