Guião da Ficha Trabalho 2

Tarefa 1 - Compilação Separada.

O encapsulamento de código e a separação de preocupações são formas para se lidar com o aumento da complexidade de um programa. Isto favorece a divisão do trabalho entre diferentes pessoas de uma mesma equipa e também o reúso de código por meio de bibliotecas. Considere como exemplo o programa:

principal.c

```
#include <stdio.h>
#include “indicadores.h”

int main() {
    float peso, altura, indice;
    printf("Indique o seu peso (em Kg): ");
    scanf("%f",&peso);
    printf("Indique a sua altura (em m): ");
    scanf("%f",&altura);
    indice = imc(peso, altura);
    printf("O valor do seu IMC é: %f\n", indice);
    return 0;
}
```

indicadores.h

```
float imc(float p, float a);
```

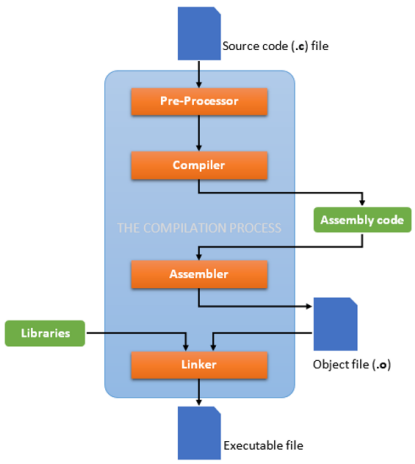
indicadores.c

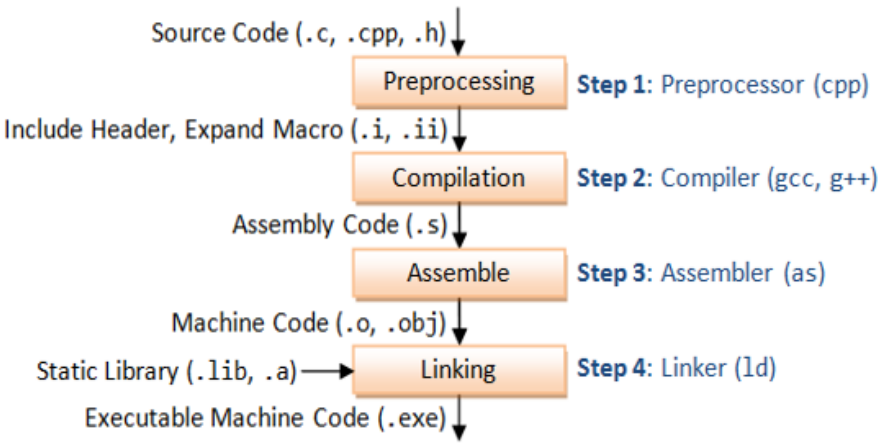
```
float imc(float p, float a) {
    float res;
    res = p / (a*a);
    return res;
}
```

a) Crie este programa no CLion e descreva quais alterações ocorrem no CMake.
b) Utilizando um editor de textos simples e o GCC no *shell* de comando, escreva compile manualmente este mesmo programa, isto deve resultar em um script próximo à:

```
gcc -c principal.c
gcc -c indicadores.c
gcc -o imc principal.o indicadores.o
```

Ainda, tendo como referência as figuras abaixo, relacione o script utilizado na geração do executável com as etapas de compilação no C. Comente isto de maneira sucinta durante a avaliação da tarefa junto com uma explicação dos diversos comando utilizados no script (experimente `gcc --h`).

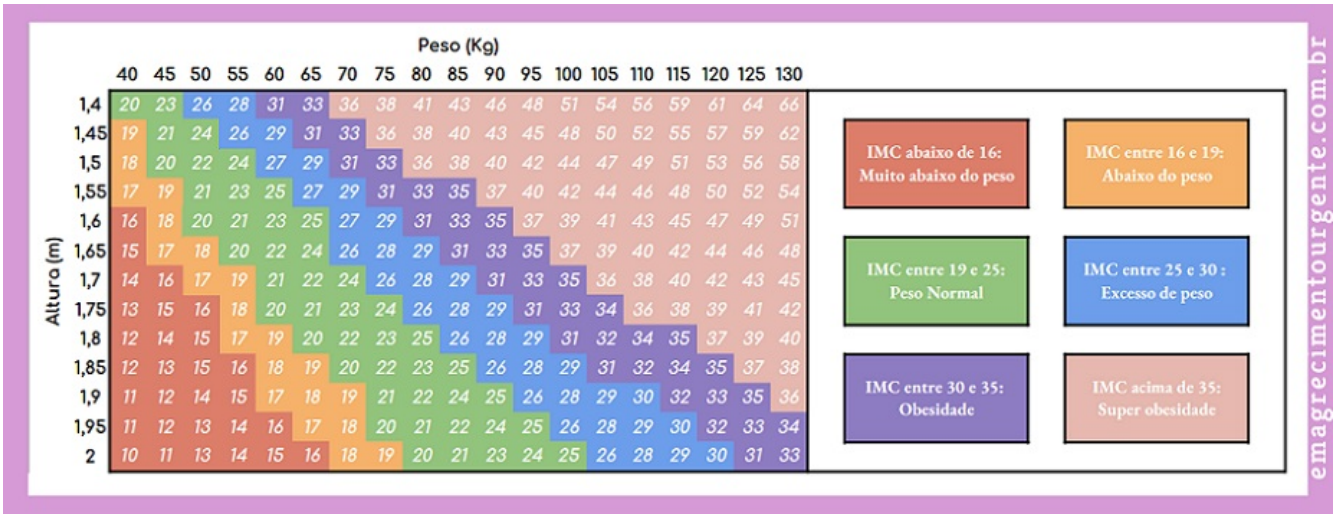




Tenha em atenção que o âmbito das variáveis declaradas numa função pois são criadas quando a função é invocada e são destruídas quando a função termina.

Tarefa 2 – How to Solve It.

Utilizando o processo de resolução de problemas anexo, reescreva o programa do IMC. Para tanto considere a tabela do IMC como referência. Um exemplo de questionamento a ser feito é se esta mesma tabela pode ser utilizada para crianças, adultos e idosos. Outra, é se existe alguma restrição nos valores de entrada, por exemplo, faz sentido calcular o IMC para algo com 1 cm de altura e 5 toneladas de peso? Em um caso como este, ainda seria válido falar em IMC?



O `assert` é uma estrutura conveniente para a criação dos testes de unidade e pode ser utilizado também para garantir que as condições de estado estejam satisfeitas durante o *runtime*. O código a seguir é um exemplo de como utilizar o `assert`:

```
#include <assert.h>
#include <stdbool.h>

assert(condição == true && “mensagem”);
```

No programa do exercício anterior, a regra de negócio está separada mas o fluxo de controlo e os elementos de fronteira (interface) estão misturados. Crie mais um módulo para encapsular a preocupação de fronteira, separando-a do fluxo de controlo.

Tarefa 3 - Ensino Assistido por Computador.

Este exercício foi retirado do livro Deital, P. Deitel, H. (2010). "C How to Program". Pearson Education, 6a Ed., p.193-194. Atenção pois trata-se de um único exercício organizado em itens visando facilitar sua realização durante a etapa inicial de aprendizagem da programação.

a) A utilização de computadores na educação é chamada Computer-assisted Instruction (CAI). Escreva um programa que poderia ajudar estudantes de escola básica a aprender multiplicação. Utilize a função rand para produzir dois inteiros positivos com um dígito. O programa deve apresentar questões como: "Quanto é 6 vezes 7?" e aguardar o aluno inserir a resposta. Em seguida, o programa verifica a resposta, se estiver correcta apresenta a mensagem "Muito bem!" e apresenta uma nova questão ou, se estiver errada, a mensagem "Não. Tente novamente." e apresenta novamente a mesma questão. Uma função separada deve ser utilizada para gerar cada nova questão.

b) Um problema com os CAI é a fadiga. Isto pode ser reduzido variando as respostas do computador para manter a atenção do estudante. Modifique o exercício anterior para que vários comentários possam ser exibidos para cada pergunta. Por exemplo, se correcto: "Óptimo!", "Belo trabalho!", "Continue assim!"; se incorrecto: "Errado, tente novamente.", "Não desista!", "Não, tente mais uma vez.". Use um gerador de números aleatórios para escolher a resposta.

Tarefas opcionais (não avaliadas)

c) Crie níveis de dificuldades no programa anterior. No primeiro nível apresente multiplicações com um dígito, no segundo com dois e no terceiro com três.

d) Modifique o programa anterior para para que o utilizador possa escolher o tipo de questão aritmética a ser trabalhado (soma, subtracção, multiplicação e divisão).

HOW TO SOLVE IT

Processo de *problem solving cf.* [Polya, 1957] adaptado para programação. Atenção, mesmo processo pode ser aplicado recursivamente para problemas essenciais e acidentais. A programação é um *wicked problem [Rittel & Webber, 1973]*, portanto não reutilize *throwaway code* na solução final. Este processo é adequado para programas pequenos (até 10k loc).

Primeiro. Entenda o problema. P {Q} S	Qual o **resultado** esperado (pós-condição de estado)? O **resultado** esperado é verificável? Quais os **dados** de entrada (pré-condição e invariantes de estado)? Como os **dados** de entrada se ligam aos **resultados** (**fluxo**)? É possível obter o **resultado** esperado com base nos **dados** de entrada? Os **dados** de entrada são suficientes, não contraditórios e não redundantes? Faça um diagrama e introduza uma notação (separe as várias partes do **fluxo**). Escreva o problema relacionando o resultado esperado com os dados de entrada, de forma não ambígua. Caso exista mais de um problema escreva cada um de maneira separada e coesa.
Segundo. Ache a conexão entre os **dados** e o **resultado**. Talvez seja necessário considerar problemas auxiliares caso uma conexão imediata não possa ser encontrada. Deve obter um plano da solução. Programação Exploratória (*throwaway code*)	Já viste o problema antes? E um problema parecido? Conhece um problema correlato? Conhece alguma API que seja útil? Olhe para o **resultado**! Tente pensar em um problema familiar em que seja esperado um resultado similar. **Havendo um problema parecido,** e solucionado anteriormente, você pode utilizá-lo? Pode usar seu resultado? Pode usar o seu método? A introdução de um elemento auxiliar torna isto possível? Você poderia reescrever o problema? Poderia reescrevê-lo um pouco diferente? Volte às definições. **Não conseguindo resolver o problema proposto**, experimente resolver um problema parecido. Consegue imaginar um problema parecido mas mais acessível? Talvez, um problema mais geral (ou talvez mais específico)? Consegue resolver uma parte do problema? Mantenha uma parte do fluxo e descarte o resto, quão longe se fica do resultado? É possível derivar algo útil a partir dos dados? Consegue pensar em outros dados ou fluxos que seja apropriados para se chegar ao resultado? Consegue mudar o resultado ou os dados para algo análogo de forma com que fiquem mais próximos? Todos os conceitos necessários para solução do problema foram levados em consideração?
Terceiro. Execute o plano.	Inicie pela estrutura e organize a abstracção de maneira top-down. Cada função deve ter apenas uma responsabilidade e o nome deve reflectir essa responsabilidade de maneira precisa. A sequencia deve ser lida como um texto estruturado para cada nível de abstracção. Garanta que os estados das variáveis de entrada e saída estejam dentro do definido (pré-condição e invariantes de estado). Separe as preocupações de fronteira, controle e regras de negócio em módulos distintos.
Quarto. Examine a solução obtida.	Verifique a solução. Cada etapa é evidentemente correcto? Pode provar? O código em cada função está no nível de abstracção correcto? As preocupações estão modularizadas correctamente? O estado das variáveis está sendo verificado? A verificação está correcta? Resolva os *warnings* do compilador. Escreva testes automáticos para as condições comuns e de contorno (lembre-se de testar se o programa interrompe quando as condições não são esperadas). Faz sentido utilizar algum dos módulos para compor uma biblioteca pessoal? Corrija aquilo que for necessário.