

# Programação e Desenvolvimento de Software 2

**Introdução**  
**+Módulo/Biblioteca**



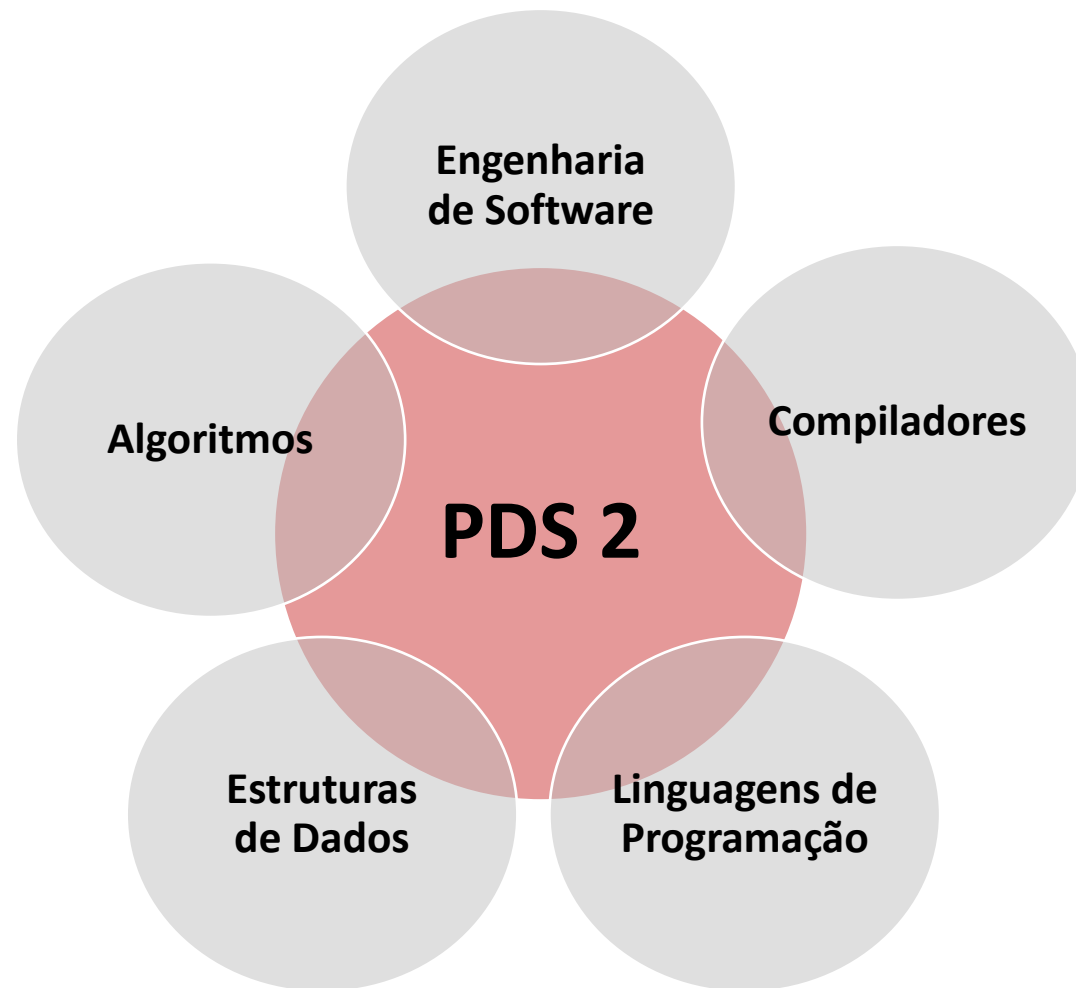
**DCC**

DEPARTAMENTO DE  
CIÊNCIA DA COMPUTAÇÃO

**UFMG**

UNIVERSIDADE FEDERAL  
DE MINAS GERAIS

# Continuação da aula 1...



# Continuação da aula I...

## Evolução do Hardware:

O hardware de computadores evoluiu ao longo das décadas, passando por várias gerações.

Cada geração trouxe melhorias em termos de velocidade de processamento, capacidade de armazenamento, tamanho físico e eficiência energética.

# Continuação da aula I...

## Evolução do Software:

O software também evoluiu para aproveitar as capacidades do hardware.

Linguagens de programação foram desenvolvidas para facilitar a criação de programas complexos.

- Modularidade (1980)

Stroustrup, Bjarne (2009). [\*Programming : principles and practice using C++\*](#). Upper Saddle River, NJ: [s.n.] [OCLC 191927017](#)

# Continuação da aula I...

## Crise de Software (naquele tempo...)

- cerca de um quarto dos projetos de desenvolvimento de grandes sistemas é cancelado antes da conclusão;
- em média, o tempo de desenvolvimento é bem maior do que o estimado;
- três quartos dos grandes sistemas não são usados ou não funcionam como planejado;
- a manutenção e reutilização de software são, em geral, extremamente difíceis e custosas;

<https://www.cse.psu.edu/~gxt29/bug/localCopies/SoftwareCrisis.html>

# Continuação da aula I...

## Crise de Software (naquele tempo...talvez nos dias atuais...)

Antes de tentar resolver um problema é essencial saber exatamente o *que* se quer resolver.



Como o  
cliente  
explicou...



Como o  
analista  
especificou...



Como o  
programador  
desenvolveu...



Como o  
cliente  
realmente  
pretendia!

# Continuação da aula I...

**Qualidade – As técnicas que serão apresentadas durante a disciplina tem impacto sobre características importantes:**

Corretude: *o sistema funciona como previsto?*

Robustez: *o sistema funciona adequadamente em situações não previstas?*

Estensibilidade: *é fácil estender e alterar o sistema?*

Reusabilidade: *é fácil reusar partes de um sistema para construir outro sistema?*

Compatibilidade: *é fácil integrar um sistema com outros sistemas?*

# Continuação da aula I...

## Qualidade – Considere um módulo de conta corrente:

Corretude: *a operação de débito altera o saldo corretamente?*

Robustez: *o sistema retorna uma mensagem adequada quando o cliente não tem saldo?*

Extensibilidade: *o código pode ser estendido para implementar uma conta especial?*

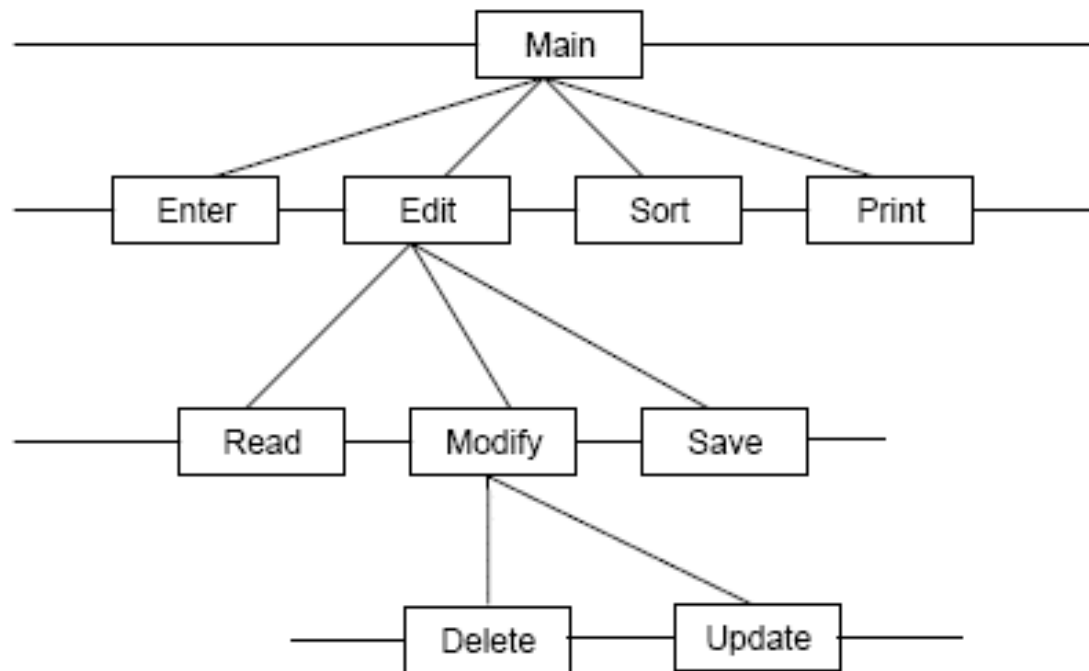
Reusabilidade: *posso utilizar esse código para outra aplicação no sistema financeiro?*

Compatibilidade: *a possibilidade de integrar com outros sistemas?*



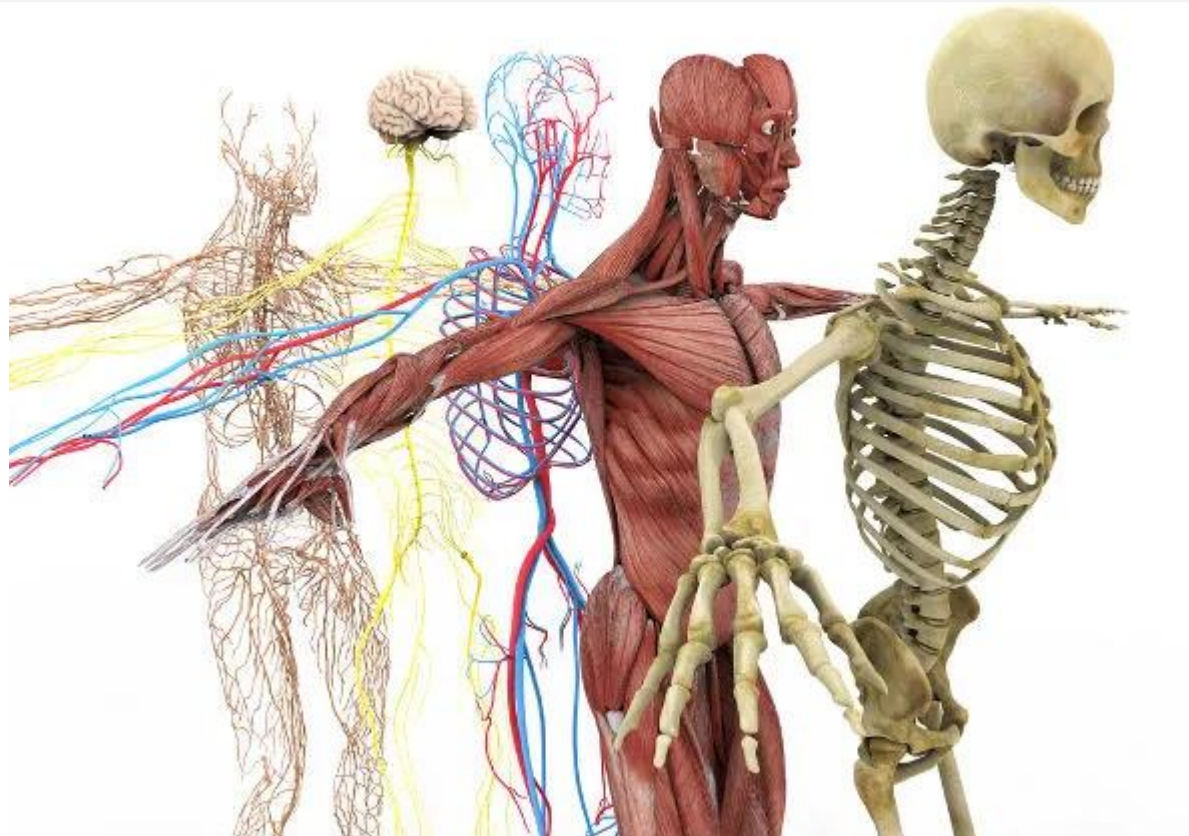
# Módulo/Biblioteca

Programação modular é uma técnica de design de software na qual particionamos o programa em diversos módulos



# Módulo/Biblioteca

Programação modular é uma técnica de design de software na qual particionamos o programa em diversos módulos



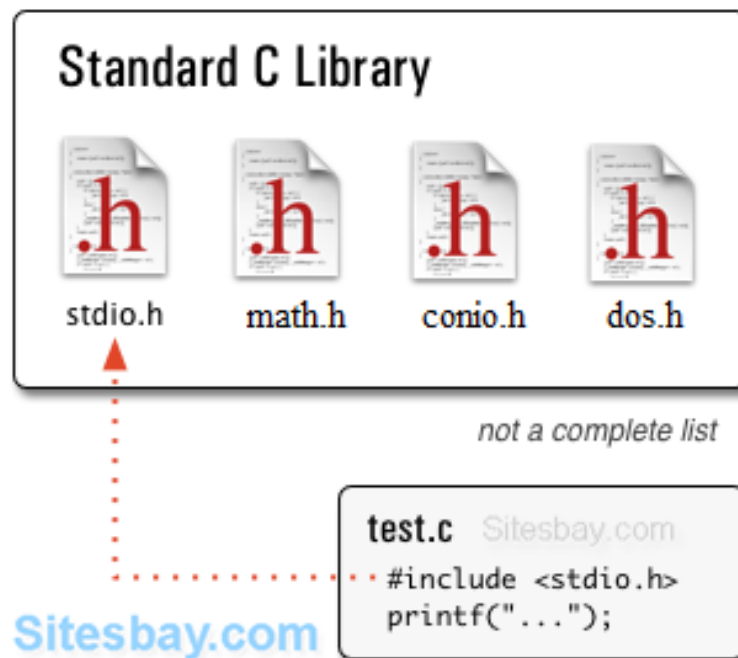
# Módulo/Biblioteca

- **biblioteca** : coleção de funcionalidades relacionadas
- **módulo**: funcionalidade específica

programação modular

# Módulo/Biblioteca

- Podem ser desenvolvidos e compilados independentemente dos demais artefatos que compõe um programa.



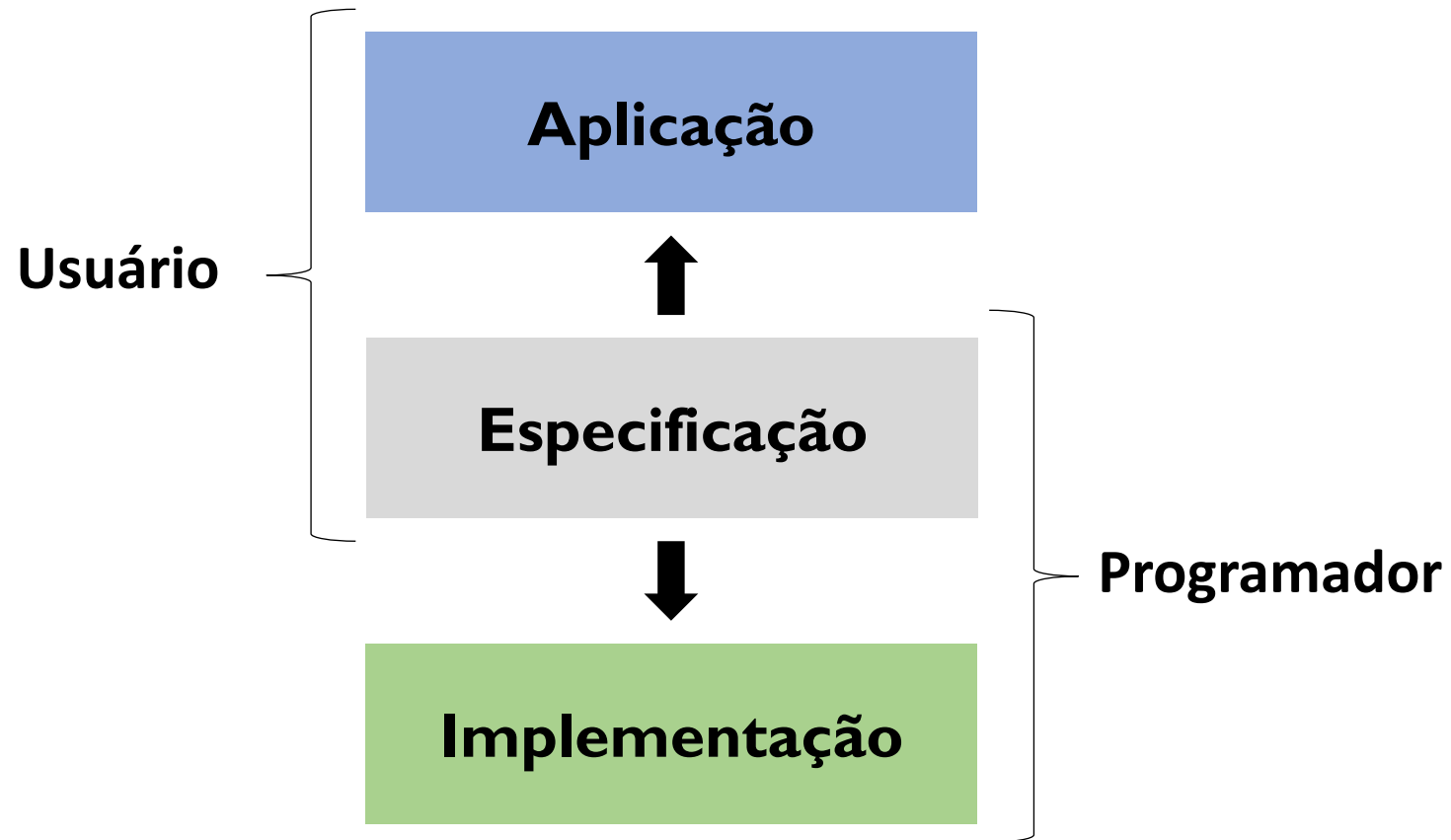
- Vantagens
  - Vencer barreiras de complexidade
  - Distribuir trabalho
  - Reutilizar módulos
  - Tornar gerenciável o processo de desenvolvimento
  - Permitir desenvolvimento incremental

- Desafios
  - Como particionar um programa?
  - Como especificar os módulos?
  - Como assegurar qualidade?
  - Como coordenar e acompanhar o trabalho em equipe?

# Módulo/Biblioteca

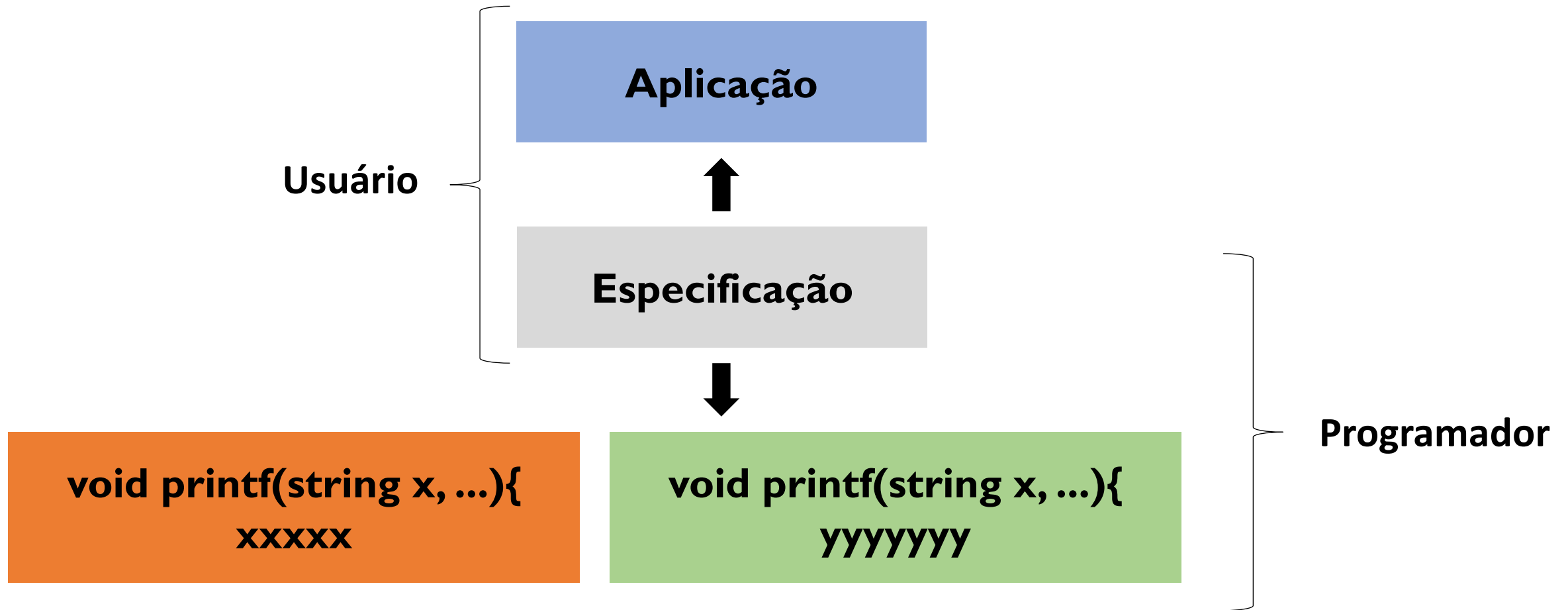
- Incorporar módulos com uso de **interface**
- O objetivo da interface é especificar sua **funcionalidade, não sua implementação.**
  - printf
  - scanf
  - math.h

# Módulo/Biblioteca





# Módulo/Biblioteca



# Módulo/Biblioteca

```
1  #include <stdio.h>
2
3  int main(){
4
5      float altura = 10;
6      float base = 20;
7      printf("Area = %f", altura * base);
8
9
10 }
```

## 1. Funções de Entrada e Saída:

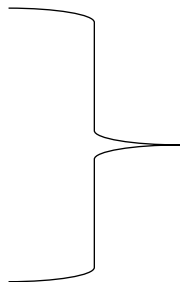
- `printf` : Formata e imprime dados na saída padrão (geralmente a tela).
- `scanf` : Lê dados da entrada padrão (geralmente o teclado) com base em um formato especificado.
- `getchar` e `putchar` : Lê e escreve caracteres individuais.

## 2. Manipulação de Arquivos:

- `FILE` : Estrutura que representa um arquivo aberto.
- `fopen` , `fclose` : Funções para abrir e fechar arquivos.
- `fread` , `fwrite` : Leitura e escrita em arquivos binários.
- `fgets` , `fputs` : Leitura e escrita de strings em arquivos.

# Módulo/Biblioteca

```
1  #include <stdio.h>
2
3  typedef struct
4  {
5      float base;
6      float altura;
7  } Retangulo;
8
9  int main(){
10
11      Retangulo r;
12      r.altura = 10;
13      r.base = 20;
14      printf("Area = %f", r.altura * r.base);
```



Um **struct** permite agrupar diversos tipos de dados relacionados.

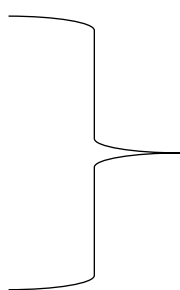
# Módulo/Biblioteca

```
1  #include <stdio.h>
2
3  ✓ typedef struct
4  {
5      float base;
6      float altura;
7  } Retangulo;
8
9  ✓ typedef struct
10 {
11     char nome[50];
12     int idade;
13     float altura;
14 } Pessoa;
```

Um **struct** permite agrupar diversos tipos de dados relacionados.

# Módulo/Biblioteca

```
1  #include <stdio.h>
2
3  typedef struct
4  {
5      float base;
6      float altura;
7  } Retangulo;
8
9  int main(){
10
11      Retangulo r;
12      r.altura = 10;
13      r.base = 20;
14      printf("Area = %f", r.altura * r.base);
```



Neste exemplo permitiu criar um novo tipo de dados denominado Retângulo.

# Módulo/Biblioteca

```
1  #include <stdio.h>
2
3  typedef struct
4  {
5      float base;
6      float altura;
7  } Retangulo;
8
9  int main(){
10     typedef Retangulo figura;
11     figura f;
12     f.altura = 10;
13     f.base = 10;
14     printf("Area = %f", f.altura * f.base);
15
16 }
17
18
```

Também permite renomear um tipo existente.

# Módulo/Biblioteca

```
1  #include <stdio.h>
2
3  typedef struct retangulo
4  {
5      float base;
6      float altura;
7  } Retangulo;
8
9  int main(){
10     struct retangulo r;
11     //Retangulo r;
12     r.altura = 10;
13     r.base = 20;
14     printf("Area = %f", r.altura * r.base);
15
16 }
```



Declarar de forma diferente.

# Módulo/Biblioteca

C: > Users > rclin > Downloads > C retangulo.h > ...

```
1  typedef struct retangulo{
2      float base;
3      float altura;
4  }Retangulo;
5
6  /**
7   * Cria um retângulo com a base e altura especificadas.
8   *
9   * @param base A medida da base do retângulo.
10  * @param altura A medida da altura do retângulo.
11  * @return Um retângulo com a base e altura fornecidas.
12  */
13  Retangulo criar_retangulo(float base, float altura);
14
15  /**
16   * Calcula a área de um retângulo.
17   *
18   * @param retangulo O retângulo para o qual a área será calculada.
19   * @return A área do retângulo.
20   */
21  float calcular_area(Retangulo retangulo);
```



**Especificação**



**Implementação**

**Programador**



# Módulo/Biblioteca

C: > Users > rclin > Downloads > C retangulo.h > ...

```
1  typedef struct retangulo{
2      float base;
3      float altura;
4  }Retangulo;
5
6  /**
7   * Cria um retângulo com a base e altura especificadas.
8   *
9   * @param base A medida da base do retângulo.
10  * @param altura A medida da altura do retângulo.
11  * @return Um retângulo com a base e altura fornecidas.
12  */
13  Retangulo criar_retangulo(float base, float altura);
14
15  /**
16   * Calcula a área de um retângulo.
17   *
18   * @param retangulo O retângulo para o qual a área será calculada.
19   * @return A área do retângulo.
20   */
21  float calcular_area(Retangulo retangulo);
```



```
int main(){
    Retangulo r = criar_retangulo(10,20);
    printf("AREA = %f", calcular_area(r));
}

cal
    calcular_area float calcular_area(Retangulo retangulo)
    clearerr
```

**Especificação**



**Implementação**

**Programador**

# Módulo/Biblioteca

C: > Users > rclin > Downloads > C retangulo.h > ...

```
1  typedef struct retangulo{
2      float base;
3      float altura;
4  }Retangulo;
5
6  /**
7   * Cria um retângulo com a base e altura especificadas.
8   *
9   * @param base A medida da base do retângulo.
10  * @param altura A medida da altura do retângulo.
11  * @return Um retângulo com a base e altura fornecidas.
12  */
13  Retangulo criar_retangulo(float base, float altura);
14
15  /**
16   * Calcula a área de um retângulo.
17   *
18   * @param retangulo O retângulo para o qual a área será calculada.
19   * @return A área do retângulo.
20   */
21  float calcular_area(Retangulo retangulo);
```



## 1. Arquivo de Cabeçalho ( .h ):

- Um arquivo de cabeçalho contém **declarações de funções**, **definições de tipos** e outras informações que são compartilhadas entre vários arquivos-fonte ( .c ).
- Ele não contém a **implementação real** das funções, apenas suas assinaturas.

# Módulo/Biblioteca



C: > Users > rclin > Downloads > C retangulo.c > calcular\_area(Retangulo)

```
1  #include <stdio.h>
2  #include "retangulo.h"
3
4
5  Retangulo criar_retangulo(float base, float altura){
6      Retangulo r;
7      r.altura = altura;
8      r.base = base;
9      return r;
10 }
11
12 float calcular_area(Retangulo retangulo){
13     return retangulo.base * retangulo.altura;
14 }
```

**Especificação**



**Implementação**

**Programador**

# Módulo/Biblioteca



C: > Users > rclin > Downloads > C retangulo.c > calcular\_area(Retangulo)

```
1  #include <stdio.h>
2  #include "retangulo.h"
3
4
5  Retangulo criar_retangulo(float base, float altura){
6      Retangulo r;
7      r.altura = altura;
8      r.base = base;
9      return r;
10 }
11
12 float calcular_area(Retangulo retangulo){
13     return retangulo.base * retangulo.altura;
14 }
```

**Implementação**

# Módulo/Biblioteca

```
C: > Users > rclin > Downloads > C principal.c > main()
1  #include <stdio.h>
2  #include "retangulo.h"
3
4  int main(){
5      Retangulo r = criar_retangulo(10,20);
6      printf("AREA = %f", calcular_area(r));
7  }
```

Usuário

**Aplicação**



**Especificação**

gcc -c retangulo.c



retangulo.o

## 2. Arquivo Objeto ( .o ):

- Um arquivo objeto é gerado a partir de um arquivo-fonte ( .c ) após a compilação.
- Ele contém o **código de máquina** resultante da compilação do código-fonte.
- O arquivo objeto não contém informações sobre funções definidas em outros arquivos.

gcc principal.c retangulo.o -o saida



saida

## 3. Processo de Vinculação:

- A vinculação ocorre após a compilação de todos os arquivos-fonte.
- Durante a vinculação, os arquivos objeto são combinados para formar o **executável final**.
- O linker (vinculador) resolve as referências a funções e variáveis entre os arquivos objeto.
- O linker procura as **definições** das funções declaradas nos arquivos de cabeçalho.

# Módulo/Biblioteca (Fazer Juntos...)

## **Criar o Módulo Ponto:**

Crie um arquivo de cabeçalho chamado ponto.h com as declarações necessárias para o módulo.

Crie um arquivo de implementação chamado ponto.c com as definições das funções relacionadas aos pontos.

## **Definir a Estrutura Ponto:**

No arquivo ponto.h, defina a estrutura Ponto com os campos x e y.

## **Função para Criar um Ponto:**

No arquivo ponto.c, implemente uma função chamada criar\_ponto que recebe as coordenadas x e y como argumentos e retorna um ponto.

## **Função para Calcular a Distância entre Dois Pontos:**

Implemente uma função chamada calcular\_distancia que recebe dois pontos como argumentos e retorna a distância entre eles.

A fórmula para calcular a distância entre dois pontos (x1, y1) e (x2, y2) é:  $distancia = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$ .

## **Exemplo de Uso:**

No arquivo principal (por exemplo, main.c), inclua o arquivo de cabeçalho ponto.h.

Crie pontos, calcule a distância entre eles e exiba o resultado.

# Módulo/Biblioteca (Desafio...)

## **Criação de uma Biblioteca de Matemática:**

Crie uma biblioteca chamada libmatematica.

Essa biblioteca deve conter funções para calcular seno, cosseno e tangente de um ângulo em radianos.

As funções devem ser declaradas em um arquivo de cabeçalho matematica.h e implementadas em um arquivo de implementação matematica.c.

Para facilitar utilize as funções sin, cos e tan definadas em math.h na implementação matematica.c.

## **Funções Necessárias:**

float calcular\_seno(float angulo\_radianos): Calcula o seno do ângulo fornecido.

float calcular\_cosseno(float angulo\_radianos): Calcula o cosseno do ângulo fornecido.

float calcular\_tangente(float angulo\_radianos): Calcula a tangente do ângulo fornecido.

## **Exemplo de Uso:**

No arquivo principal (por exemplo, main.c), inclua o arquivo de cabeçalho matematica.h.

Use as funções da biblioteca para calcular os valores trigonométricos de ângulos específicos.

Lembre-se de compilar todos os arquivos-fonte juntos para criar o executável final. Isso demonstrará como criar e usar uma biblioteca modular em C para cálculos matemáticos. ☐



# Módulo/Biblioteca (Desafio...)

Agora que temos o código da biblioteca de matemática, vamos subi-lo para o GitHub. Siga os passos abaixo:

**Leia atentamente os passos endereçados no link a seguir**

<https://git-scm.com/book/pt-br/v2/GitHub-Configurando-uma-conta>

**Crie um Repositório no GitHub:**

Acesse o GitHub e faça login na sua conta (ou crie uma, se ainda não tiver).

Clique no botão “New” para criar um novo repositório.

Dê um nome ao seu repositório (por exemplo, “libmatematica”).

Escolha a visibilidade (**público** ou privado) e outras configurações desejadas.

Clique em “Create repository”.

**Conecte seu Repositório Local ao GitHub:**

No terminal ou prompt de comando, navegue até o diretório do seu projeto local.

Execute os seguintes comandos: `git init` `git add .` `git commit -m "Initial commit"` `git remote add origin`

`<URL_DO_SEU_REPO> git push -u origin main`

Substitua `<URL_DO_SEU_REPO>` pela URL do repositório que você criou no GitHub.

**Verifique no GitHub:**

Atualize a página do seu repositório no GitHub.

Você verá os arquivos do seu projeto lá!

**Envie no moodle o passo a passo para que o professor possa clonar o seu projeto.**