

Programação e Desenvolvimento de Software I

Funções

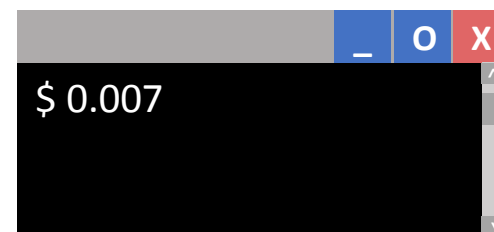
Prof. Héctor Azpúrua
(slides adaptados do Prof. Pedro Olmo)

Repasso da aula anterior

Duvidas

- Posso usar `printf` para imprimir só a parte decimal de um numero?

```
double d = 2.007;
```



- Como fazer?

- Infelizmente não tem um formatador de `printf` que faça isso de forma nativa...
 - Tem que fazer “na mão”!

```
// nao e seguro fazer desta forma!  
double d = 2.55;  
double remainder = d - (int)d;  
printf("%.2f\n", remainder);
```

```
double a = -2.55;  
double b, i;  
// i vai receber a parte inteira  
// b vai receber a parte fraccionaria  
b = modf(a, &i);  
printf("%.2f\n", b);
```

Repasso da aula anterior

Variáveis

- **Cada variável pode possuir uma quantidade diferente de bytes**, uma vez que os tipos de dados são representados de forma diferente
- Portanto, a cada variável está associado um tipo específico de dados
- Logo:

O tipo da variável define quantos bytes de memória serão necessários para **representar os dados** que a variável armazena

Repasso da aula anterior

Tipos de dados

- A Linguagem C dispõe de **quatro tipos básicos de dados**
- Assim, as variáveis poderão assumir os seguintes tipos:

Tipo	Tamanho (bytes)	Valor
char	1	Um caractere ou um inteiro de 0 a 127
int	4	Um número inteiro
float	4	Um número de ponto flutuante (SP)
double	8	Um número de ponto flutuante (DP)

Repasso da aula anterior

Tags do printf

- Na função `printf`, para cada `tag` existente no primeiro parâmetro, deverá haver um novo parâmetro que especifica o valor a ser exibido

```
int a = 9;  
float b = 0.1;  
printf("a=%d, b=%c e c=%f", a, 'm', (a+b));
```

Repasso da aula anterior

Prioridade de execução das operações

- A execução de operações segue uma ordem específica que **sempre é cumprida**:
 - 1) Expressões entre parênteses
 - 2) Multiplicação, divisão e resto da divisão (da esquerda para a direita)
 - 3) Operações de soma e subtração (da esquerda para a direita)



Repasso da aula anterior

Conversão implícita de tipo

- **Atenção!** Observe que os resultados dos seguintes códigos são diferentes!

`d = (float) a / b;`

`d = (float) (a / b);`

- O que tem de diferente?
- No primeiro:
 - Primeiro realiza-se primeiro a conversão explícita de tipo (`a` torna-se `10.0`) e, em seguida, realiza-se a divisão
 - Logo: `d = 3.333333`
- No segundo:
 - Primeiro divide-se `a` por `b` e, em seguida, se faz a conversão explícita de tipo
 - Logo: `d = 3.0`

Exercício

Brutus e Olívia

- Brutus e Olívia foram ao médico, que disse a eles que ambos estão fora do peso ideal. Ambos discordaram veementemente da afirmação do médico. Para provar que estava certo, o médico mostrou o Índice de Massa Corporal (IMC) de ambos, considerando que Brutus tem 1,84m e pesa 112kg e Olívia tem 1,76m e pesa 49kg.
- Implemente um programa para:
 - Mostrar o IMC de Brutus e Olívia
 - Quantos kilos Brutus e Olívia devem perder/ganhar para atingirem um peso saudável segundo a classificação do IMC

Exercício

Brutus e Olívia



IMC	Classificação
< 16	Magreza grave
$16 \text{ a } < 17$	Magreza moderada
$17 \text{ a } < 18.5$	Magreza leve
$18.5 \text{ a } < 25$	Saudável
$25 \text{ a } < 30$	Sobrepeso
$30 \text{ a } < 35$	Obesidade grau I
$35 \text{ a } < 40$	Obesidade grau II (severa)
≥ 40	Obesidade grau III (mórbida)

Table 3.1: Classificação do IMC

Exercício

Brutus e Olívia

```
#include <stdio.h>
```

```
// gcc -o brutus brutus.c && ./brutus
```

```
int main() {  
    float peso_olivia = 45;  
    float altura_olivia = 1.76;  
    float peso_brutus = 122;  
    float altura_brutus = 1.84;  
  
    float peso_ideal_olivia = 18.5 * (altura_olivia * altura_olivia);  
    float peso_ideal_brutus = 25 * (altura_brutus * altura_brutus);  
  
    printf("Olivia deve ganhar %.2f Kg\n", peso_ideal_olivia - peso_olivia);  
    printf("Brutus deve perder %.2f Kg\n", peso_brutus - peso_ideal_brutus);  
    return 0;  
}
```

IMC	Classificação
< 16	Magreza grave
16 a < 17	Magreza moderada
17 a < 18.5	Magreza leve
18.5 a < 25	Saudável
25 a < 30	Sobrepeso
30 a < 35	Obesidade grau I
35 a < 40	Obesidade grau II (severa)
≥ 40	Obesidade grau III (mórbida)

Table 3.1: Classificação do IMC



Exercício

Conta poupança

- Uma conta poupança foi aberta com um depósito de R\$500,00, com rendimentos 1% de juros ao mês. No segundo mês, R\$200,00 reais foram depositados nessa conta poupança. No terceiro mês, R\$50,00 reais foram retirados da conta.
- Quanto haverá nessa conta no quarto mês?

Exercício

Conta poupança

- Pensando no problema...
 - Qual seria o algoritmo a seguir?
- Algoritmo:
 1. Preciso de armazenar o valor dessa conta
 2. O valor será acrescido todo mês de 1%
 3. Vou adicionar 200 reais a esse valor no segundo mês
 4. Vou retirar 50 reais desse valor no terceiro

Conta poupança

Solução I

```
#include <stdio.h>

// gcc -o poupanca1 poupanca1.c && ./poupanca1

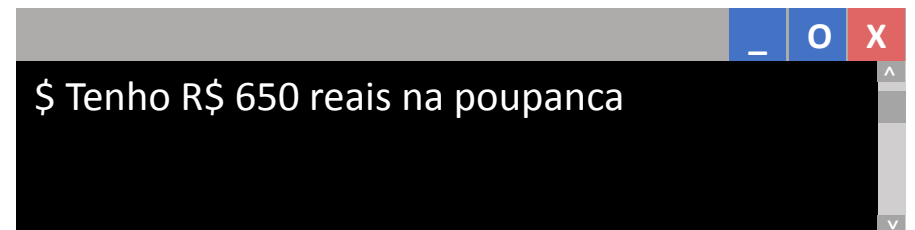
int main() {
    // saldo inicial
    int poupanca = 500;
    // rendimento apos o primeiro mes
    poupanca = poupanca + poupanca * (1 / 100);

    // deposito de 200 reais no segundo mes
    poupanca = poupanca + 200;
    // rendimento apos o segundo mes
    poupanca = poupanca + poupanca * (1 / 100);

    // retirada de 50 reais no terceiro mes
    poupanca = poupanca - 50;
    // rendimento apos o terceiro mes
    poupanca = poupanca + poupanca * (1 / 100);

    printf("Tenho R$ %d na poupanca\n", poupanca);
    return 0;
}
```

- Existe algum problema com esse código?
 - Estamos usando só números inteiros!
 - Divisão de $1/100$ em inteiro da quanto?
 - Zero!



```
$ Tenho R$ 650 reais na poupanca
```

Conta poupança

Solução 2 – Uma melhor forma de fazer o mesmo código

```
#include <stdio.h>
```

```
// gcc -o poupanca1 poupanca1.c && ./poupanca1
```

```
int main() {
```

```
    // saldo inicial
```

```
    float poupanca = 500;
```

```
    // rendimento apos o primeiro mes
```

```
    poupanca = poupanca + poupanca * (1.0 / 100);
```

```
    // deposito de 200 reais no segundo mes
```

```
    poupanca = poupanca + 200;
```

```
    // rendimento apos o segundo mes
```

```
    poupanca = poupanca + poupanca * (1.0 / 100);
```

```
    // retirada de 50 reais no terceiro mes
```

```
    poupanca = poupanca - 50;
```

```
    // rendimento apos o terceiro mes
```

```
    poupanca = poupanca + poupanca * (1.0 / 100);
```

```
    printf("Tenho R$ %.2f na poupanca\n", poupanca);
```

```
    return 0;
```

```
}
```

- Agora o valor parece estar mais correto!
- Pequenos erros podem se acumular com o tempo...
 - Produzindo grandes erros!
- **Tem como melhorar?**
 - **Sim!**



\$ Tenho R\$ 668.67 na poupanca

Conta poupança

Solução 3 – Ainda dá pra melhorar...

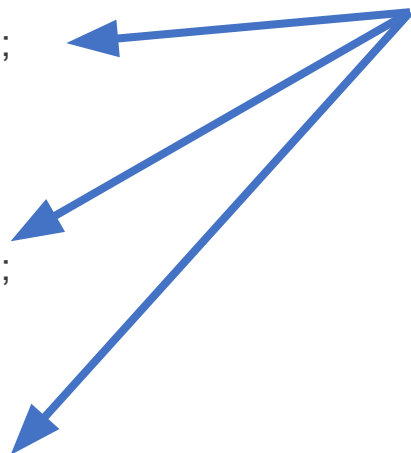
```
#include <stdio.h>
```

```
// gcc -o poupanca1 poupanca1.c && ./poupanca1
```

```
int main() {  
    // saldo inicial  
    float poupanca = 500;  
    // rendimento apos o primeiro mes  
    poupanca = poupanca + poupanca * (1.0 / 100);  
  
    // deposito de 200 reais no segundo mes  
    poupanca = poupanca + 200;  
    // rendimento apos o segundo mes  
    poupanca = poupanca + poupanca * (1.0 / 100);  
  
    // retirada de 50 reais no terceiro mes  
    poupanca = poupanca - 50;  
    // rendimento apos o terceiro mes  
    poupanca = poupanca + poupanca * (1.0 / 100);  
  
    printf("Tenho R$ %.2f na poupanca\n", poupanca);  
    return 0;  
}
```

- Como podemos fazer para mudar o rendimento da poupança?

- De 1% para 2%?



Conta poupança

Solução 3 – Ainda dá pra melhorar...

```
#include <stdio.h>

// gcc -o poupanca3 poupanca3.c && ./poupanca3


int main() {
    float juros = 1.0 / 100;

    // saldo inicial
    float poupanca = 500;
    // rendimento apos o primeiro mes
    poupanca = poupanca + poupanca * juros;

    // deposito de 200 reais no segundo mes
    poupanca = poupanca + 200;
    // rendimento apos o segundo mes
    poupanca = poupanca + poupanca * juros;

    // retirada de 50 reais no terceiro mes
    poupanca = poupanca - 50;
    // rendimento apos o terceiro mes
    poupanca = poupanca + poupanca * juros;

    printf("Tenho R$ %.2f na poupanca\n", poupanca);
    return 0;
}
```



- Como podemos fazer para mudar o rendimento da poupança?

- De 1% para 2%?

- Tem como melhorar ainda mais?

- Sim!

Conta poupança

Solução 4 – Melhorando ainda mais...

■ Um pouco de matemática:

- $\text{poupanca} = \text{poupanca} + \text{poupanca} * \text{juros};$
- $\text{poupanca} = \text{poupanca} * (1 + \text{juros});$
- `//como juros == 0.01, então:`
- $\text{poupanca} = \text{poupanca} * 1.01$

Conta poupança

Solução 4 – Melhorando ainda mais...

```
#include <stdio.h>
```

```
// gcc -o poupanca4 poupanca4.c && ./poupanca4
```

```
int main() {
```

```
    float juros = 1.01;
```

```
    // saldo inicial
```

```
    float poupanca = 500.0;
```

```
    // rendimento apos o primeiro mes
```

```
    poupanca = poupanca * juros;
```

```
    // deposito de 200 reais no segundo mes
```

```
    poupanca = poupanca + 200;
```

```
    // rendimento apos o segundo mes
```

```
    poupanca = poupanca * juros;
```

```
    // retirada de 50 reais no terceiro mes
```

```
    poupanca = poupanca - 50;
```

```
    // rendimento apos o terceiro mes
```

```
    poupanca = poupanca * juros;
```

```
    printf("Tenho R$ %.2f na poupanca\n", poupanca);
```

```
    return 0;
```

■ Simplificando as contas

■ Tem como melhorar ainda mais?

■ Sim! 😊

Conta poupança

Solução 5 – Taxas de manutenção de conta

- E se o banco definir que junto com o rendimento mensal, R\$3,14 reais devem ser retirados da conta como taxa de manutenção?
- **Como fazer?**
 - Podemos criar uma função que centraliza as operações mensais padrões do banco!



Riquinho, dono do banco

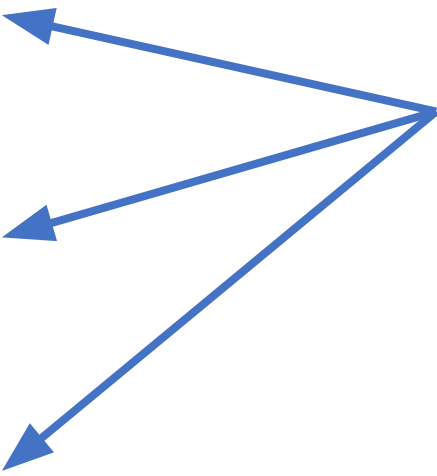
Conta poupança

Solução 5 – Taxas de manutenção de conta

```
int main() {  
    // saldo inicial  
    float poupanca = 500.0;  
    // rendimento apos o primeiro mes  
    poupanca = rende_poupanca(poupanca);  
  
    // deposito de 200 reais no segundo mes  
    poupanca = poupanca + 200;  
    // rendimento apos o segundo mes  
    poupanca = rende_poupanca(poupanca);  
  
    // retirada de 50 reais no terceiro mes  
    poupanca = poupanca - 50;  
    // rendimento apos o terceiro mes  
    poupanca = rende_poupanca(poupanca);  
  
    printf("Tenho R$ %.2f na poupanca\n", poupanca);  
    return 0;  
}
```

- A função `rende_poupanca` é encarregada de realizar todas as operações de rendimento mensais da conta poupança

```
float rende_poupanca(float saldo) {  
    float juros = 1.01;  
    float taxa = 3.14;  
    return (saldo * juros) - taxa;  
}
```



Conta poupança

Solução 5 – Taxas de manutenção de conta

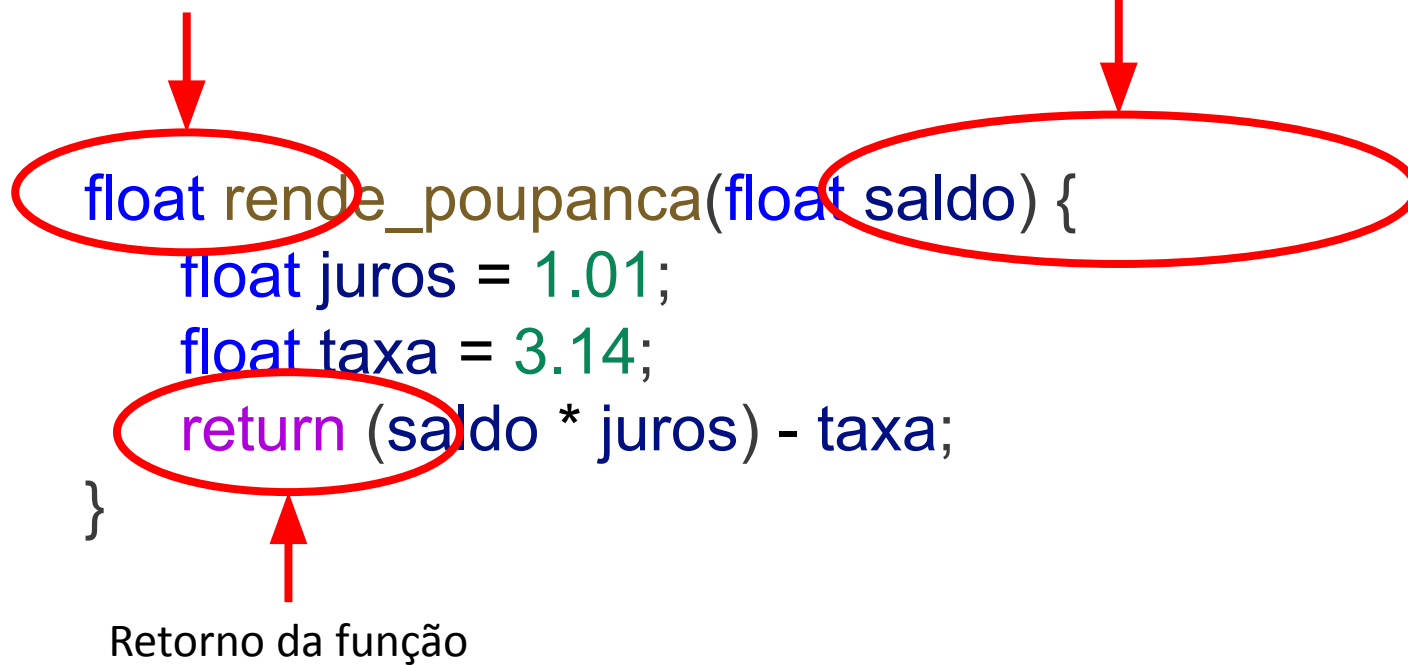
■ Escrevendo uma função em C:

Tipo de retorno da função

Parâmetro da função

```
float rende_poupanca(float saldo) {  
    float juros = 1.01;  
    float taxa = 3.14;  
    return (saldo * juros) - taxa;  
}
```

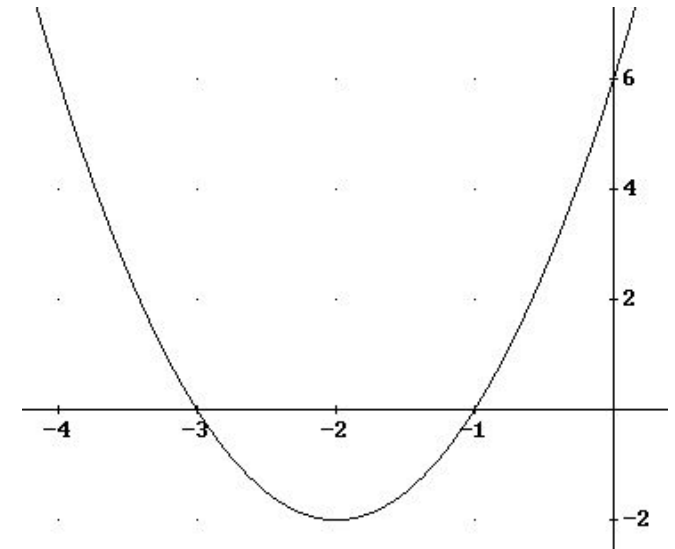
Retorno da função



Funções

Revisão

- Funções definem operações que são usadas frequentemente
- Funções, na matemática, requerem parâmetros de entrada, e definem um valor de saída
- Função quadrática $y = ax^2 + bx + c$
 - Entrada: x
 - Saída: y



Funções na programação

Revisão

- Em linguagens imperativas, TODOS os programas usam funções
- No C, o programa SEMPRE começa executando a função `main`

```
#include <stdio.h>

// gcc -o hola hello_world.c

int main() {
    printf("ola\n");
    return 0;
}
```

Funções na programação

Revisão

- Por que usamos funções?
 - Usamos funções para evitar de escrever várias vezes o mesmo código
 - Código que será executado várias vezes em um programa, mas com valores diferentes
 - Operações comuns a um ou mais programas

Funções na programação

Revisão

- Em C, definimos a função por:

1. Zero ou mais parâmetros de entrada, com os seus tipos
2. Um parâmetro de saída com tipo, sendo que o tipo pode ser “sem saída” (void)
3. Código da função

Importante:

Ao chamarmos a função, devemos passar valores para **TODOS** os parâmetros de entrada, sem exceção (não é o caso no C++...)

Tipo de retorno da função
ou saída da função

Parâmetros de entrada

Código da
função

```
float rende_poupanca(float saldo) {  
    float juros = 1.01;  
    float taxa = 3.14;  
    return (saldo * juros) - taxa;  
}
```


Exemplo

Função logística

- Escrevendo uma função em C:
 - Qual é o nome da função?

Nome da função

```
double logistica(double x) {  
    return 1.0 / (1.0 + exp(-1.0 * x));  
}
```



Exemplo

Função logística

- Escrevendo uma função em C:
 - Quais seriam os parâmetros de entrada?

```
double logistica(double x) {  
    return 1.0 / (1.0 + exp(-1.0 * x));  
}
```

Parâmetros de entrada da
função



Exemplo

Função logística

- Escrevendo uma função em C:
 - Qual seria o tipo de saída?

Tipo de saída da função

```
double logistica(double x) {  
    return 1.0 / (1.0 + exp(-1.0 * x));  
}
```

Exemplo

Função logística

- Escrevendo uma função em C:
 - Qual seria o código da função?

```
double logistica(double x) {  
    return 1.0 / (1.0 + exp(-1.0 * x));  
}
```

Código da função



Exemplo

Função logística

- Escrevendo uma função em C:
 - Quem **recebe** o valor da função logística?

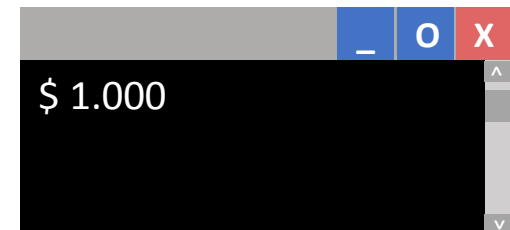
```
#include <math.h>
#include <stdio.h>
```

```
// gcc -o logistica logistica.c && ./logistica
```

```
double logistica(double x) {
    return 1.0 / (1.0 + exp(-1.0 * x));
}
```

```
int main() {
    double entrada = 10;
    double saida = logistica(entrada);
    printf("%.3f\n", saida);
    return 0;
}
```

Variável **saida** recebe o valor da função



Exemplo

Função logística

- Escrevendo uma função em C:
 - Quais são os parâmetros?
 - O que a função retorna?

Retorna um inteiro

```
int sep(int v[], int p, int r) {  
    int w[1000], i = p, j = r, c = v[p], k;  
    for (k = p + 1; k <= r; ++k)  
        if (v[k] <= c)  
            w[i++] = v[k];  
        else  
            w[j--] = v[k];  
    w[i] = c;  
    for (k = p; k <= r; ++k) v[k] = w[k];  
    return i;  
}
```

Parâmetros de entrada

A função `main`

- A função `main` é especial:
 - É a primeira a ser chamada no programa
 - **Todo programa em C/C++ tem uma!**
 - Seu retorno pode indicar se o programa executou corretamente ou não:
 - retorno 0 😊
 - retorno != 0 😞
 - Seus parâmetros, quando existem, são os parâmetros passados para o programa quando foi executado:
 - `int argc, char *argv[]`

Funções sem retorno

- Funções sem retorno (ou **procedimentos**) devem ter o tipo de retorno `void`
- **Exemplo:** função para imprimir mensagem de boas-vindas do programa

```
void saudacao() {  
    printf("Ola usuario! Digite o comando que quer executar:");  
}  
int main() {  
    saudacao();  
    // faz outras coisas...  
    return 0;  
}
```

Funções sem retorno

- Porque não retornar valor?
 - Porque o importante pode ser a ação **colateral** da função, e não o seu valor de saída:
 - Impressão de uma mensagem
 - Ligar/desligar um componente do hardware
 - ...
 - Porque a função sempre executa sem erro:
 - `void exit();`

Funções

Escopo de variáveis

- Variáveis podem ser acessadas **somente dentro do seu escopo**
- No C, o escopo é definido do **momento da declaração até o fim do bloco**
- No C, uma variável declarada dentro de um bloco de laço vive somente uma iteração do laço


Funções

Escopo de variáveis

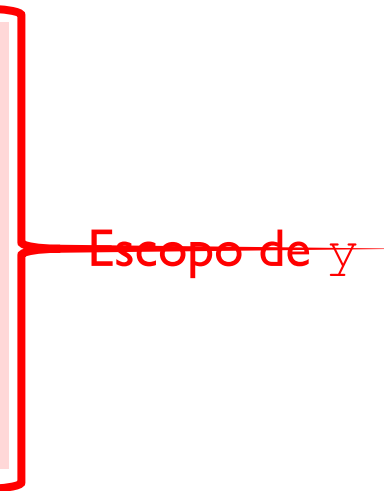
▪ Qual seria o escopo de:

- x?
- y?
- i?

```
int teste(int x) {  
    printf("%d", x);  
    return -1;  
}
```



```
int main() {  
    int y;  
    for (int i = 0; i < 10; i++) {  
        if (i < 5) {  
            int a;  
        } else {  
            int b;  
        }  
    }  
    return 0;  
}
```



Funções

Escopo de variáveis

- Qual seria o escopo de:

- x?
- y?
- i?
- a?

```
int teste(int x) {  
    printf("%d", x);  
    return -1;  
}
```

```
int main() {  
    int y;  
    for (int i = 0; i < 10; i++) {  
        if (i < 5) {  
            int a;  
        } else {  
            int b;  
        }  
    }  
    return 0;  
}
```

Escopo de i

Funções

Escopo de variáveis

▪ Qual seria o escopo de:

- x?
- y?
- i?
- a?
- b?

```
int teste(int x) {  
    printf("%d", x);  
    return -1;  
}
```

```
int main() {  
    int y;  
    for (int i = 0; i < 10; i++) {  
        if (i < 5) {  
            int a;  
        } else {  
            int b;  
        }  
    }  
    return 0;  
}
```

} Escopo de a

Funções

Escopo de variáveis

- Qual seria o escopo de:

- x?
- y?
- i?
- a?
- b?

```
int teste(int x) {  
    printf("%d", x);  
    return -1;  
}
```

```
int main() {  
    int y;  
    for (int i = 0; i < 10; i++) {  
        if (i < 5) {  
            int a;  
        } else {  
            int b;  
        }  
    }  
    return 0;  
}
```

} Escopo de b

Funções

Exemplo: Escopo de variáveis

```
#include <stdio.h>
```

```
float rende_poupanca(float x) {  
    float novo_valor;  
    novo_valor = x * 1.01;  
    return novo_valor;  
}
```

```
int main() {
```

```
    float poupanca;  
    float juros = 1.01;  
    poupanca = 500.0;
```

```
    poupanca = rende_poupanca(poupanca);
```

```
    poupanca = poupanca + 200;  
    poupanca = rende_poupanca(poupanca);
```

```
    poupanca = poupanca - 50;  
    poupanca = rende_poupanca(poupanca);
```

```
    printf("Tenho R$ %.2f na poupanca\n", poupanca);  
    return 0;
```

```
}
```

Endereço	Variável	Conteúdo
4812		
4813		
4814		
4815		
4816		
4817		
4818		
4819		

Funções

Exemplo: Escopo de variáveis

```
#include <stdio.h>
```

```
float rende_poupanca(float x) {
```

```
    float novo_valor;  
    novo_valor = x * 1.01;  
    return novo_valor;  
}
```

```
int main() {
```

```
    float poupanca;  
    float juros = 1.01;  
    poupanca = 500.0;
```

```
    poupanca = rende_poupanca(poupanca);
```

```
    poupanca = poupanca + 200;  
    poupanca = rende_poupanca(poupanca);
```

```
    poupanca = poupanca - 50;  
    poupanca = rende_poupanca(poupanca);
```

```
    printf("Tenho R$ %.2f na poupanca\n", poupanca);  
    return 0;
```

```
}
```

Endereço	Variável	Conteúdo
4812	poupanca	500.0
4813	juros	1.01
4814		
4815		
4816		
4817		
4818		
4819		

Funções

Exemplo: Escopo de variáveis

```
#include <stdio.h>
```

```
float rende_poupanca(float x) {
```

```
    float novo_valor;
```

```
    novo_valor = x * 1.01;
```

```
    return novo_valor;
```

```
}
```

```
int main() {
```

```
    float poupanca;
```

```
    float juros = 1.01;
```

```
    poupanca = 500.0;
```

```
    poupanca = rende_poupanca(poupanca);
```

```
    poupanca = poupanca + 200;
```

```
    poupanca = rende_poupanca(poupanca);
```

```
    poupanca = poupanca - 50;
```

```
    poupanca = rende_poupanca(poupanca);
```

```
    printf("Tenho R$ %.2f na poupanca\n", poupanca);
```

```
    return 0;
```

```
}
```

Endereço	Variável	Conteúdo
4812	poupanca	500.0
4813	juros	1.01
4814	x	500.0
4815		
4816		
4817		
4818		
4819		

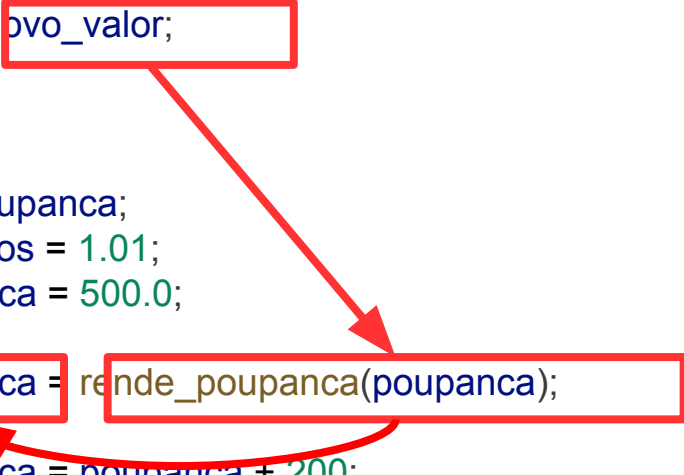
Funções

Exemplo: Escopo de variáveis

```
#include <stdio.h>
```

```
float rende_poupanca(float x) {  
    float novo_valor;  
    novo_valor = x * 1.01;  
    return novo_valor;  
}
```

```
int main() {  
    float poupanca;  
    float juros = 1.01;  
    poupanca = 500.0;  
  
    poupanca = rende_poupanca(poupanca);  
  
    poupanca = poupanca + 200;  
    poupanca = rende_poupanca(poupanca);  
  
    poupanca = poupanca - 50;  
    poupanca = rende_poupanca(poupanca);  
  
    printf("Tenho R$ %.2f na poupanca\n", poupanca);  
    return 0;  
}
```



Endereço	Variável	Conteúdo
4812	poupanca	500.0
4813	juros	1.01
4814	x	500.0
4815	novo_valor	505.0
4816		
4817		
4818		
4819		

Funções

Exemplo: Escopo de variáveis

```
#include <stdio.h>
```

```
float rende_poupanca(float x) {  
    float novo_valor;  
    novo_valor = x * 1.01;  
    return novo_valor;  
}
```

```
int main() {  
    float poupanca;  
    float juros = 1.01;  
    poupanca = 500.0;  
  
    poupanca = rende_poupanca(poupanca);  
  
    poupanca = poupanca + 200;  
    poupanca = rende_poupanca(poupanca);  
  
    poupanca = poupanca - 50;  
    poupanca = rende_poupanca(poupanca);  
  
    printf("Tenho R$ %.2f na poupanca\n", poupanca);  
    return 0;  
}
```

Endereço	Variável	Conteúdo
4812	poupanca	505.0
4813		
4814		
4815		
4816		
4817		
4818		
4819		

Funções

Exemplo: Escopo de variáveis

```
#include <stdio.h>
```

```
float rende_poupanca(float x) {  
    float novo_valor;  
    novo_valor = x * 1.01;  
    return novo_valor;  
}
```

```
int main() {  
    float poupanca;  
    float juros = 1.01;  
    poupanca = 500.0;  
  
    poupanca = rende_poupanca(poupanca);  
    poupanca = poupanca + 200;  
    poupanca = rende_poupanca(poupanca);  
  
    poupanca = poupanca - 50;  
    poupanca = rende_poupanca(poupanca);  
  
    printf("Tenho R$ %.2f na poupanca\n", poupanca);  
    return 0;  
}
```

Endereço	Variável	Conteúdo
4812	poupanca	705.0
4813	juros	1.01
4814	x	500.0
4815	novo_valor	505.0
4816		
4817		
4818		
4819		

Módulos

Introdução

- Um módulo é uma **forma de organizar um programa grande**
- Dividimos o programa em módulos, onde cada um deles possui um conjunto de tarefas bem específico:
 - Módulo de entrada/saída
 - Módulo de gerenciamento de memória
 - Módulo de cálculo, etc...
- Módulos terão uma ou mais funções, que desta forma realizam operações similares
 - Módulo de operações matemáticas
 - Módulo de operações sobre o tempo
 - Módulo para entrada e saída

Módulos

Introdução

- Alguns exemplos de módulos em C:
 - `math.h`
 - operações matemáticas (`log`, `pow`, `sqrt`, ...)
 - `time.h`
 - operações sobre o tempo (`time`, `difftime`, ...)
 - `stdio.h`
 - operações de entrada e saída (`printf`, `scanf`, ...)

Módulos

Bibliotecas

- Módulos muito úteis podem ser empacotados em **bibliotecas**, para que possam ser utilizados em outros programas
- Em C, carregamos módulos e bibliotecas com o comando `#include`

```
#include <stdio.h>
```

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

```
#include "meumodulo.h"
```

- O uso de `<>` ou `" "` depende da localização do módulo/biblioteca
 - No diretório de bibliotecas do sistema: `<>`
 - Em outro lugar (ex, no diretório onde está o programa): `" "`

Módulos

Definindo um módulo

- O módulo consiste em:
 - Arquivo de cabeçalhos de funções e declaração de tipos de dados (extensão .h)
 - Arquivo com o código das funções (extensão .c)

Arquivo simples.h

```
double media(double a, double b);  
  
double dif(double a, double b);
```

Arquivo simples.c

```
#include "simples.h"  
  
double media(double a, double b) {  
    return (a + b) / 2;  
}  
  
double dif(double a, double b) {  
    return a - b;  
}
```

Módulos

Bibliotecas padrão do C

- Muitas funções comuns:
 - `stdio.h` – Entrada e saída
 - `math.h` – Funções matemáticas mais complexas
 - `stdlib.h` – gerenciamento do programa: alocar memória, sair, ...
 - `time.h` – Gerenciar o tempo: imprimir datas, ver a hora/data atual...
- Podemos encontrar a lista de funções em manuais, livros e em sites Web, i.e.:
 - https://en.wikipedia.org/wiki/C_standard_library
 - <https://www.programiz.com/c-programming/library-function>

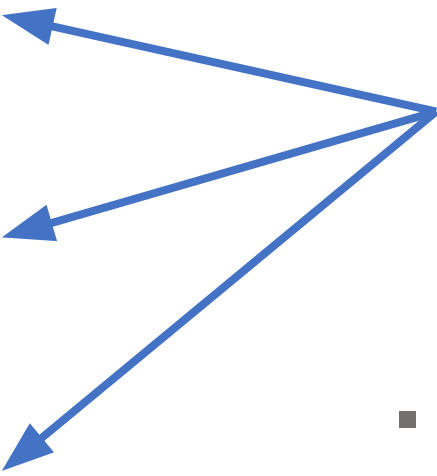
Funções: Conta poupança

Solução 5 – Taxas de manutenção de conta

```
int main() {  
    // saldo inicial  
    float poupanca = 500.0;  
    // rendimento apos o primeiro mes  
    poupanca = rende_poupanca(poupanca);  
  
    // deposito de 200 reais no segundo mes  
    poupanca = poupanca + 200;  
    // rendimento apos o segundo mes  
    poupanca = rende_poupanca(poupanca);  
  
    // retirada de 50 reais no terceiro mes  
    poupanca = poupanca - 50;  
    // rendimento apos o terceiro mes  
    poupanca = rende_poupanca(poupanca);  
  
    printf("Tenho R$ %.2f na poupanca\n", poupanca);  
    return 0;  
}
```

- A função `rende_poupanca` é encarregada de realizar todas as operações de rendimento mensais da conta poupança

```
float rende_poupanca(float saldo) {  
    float juros = 1.01;  
    float taxa = 3.14;  
    return (saldo * juros) - taxa;  
}
```



- Podemos melhorar ainda mais?
 - Sim!

Funções: Conta poupança

Solução 6 – Usando mais funções

```
int main() {  
    // saldo inicial  
    float poupanca = 500.0;  
    // rendimento apos o primeiro mes  
    poupanca = rende_poupanca(poupanca);  
  
    // deposito de 200 reais no segundo mes  
    poupanca = deposito(poupanca, 200);  
    // rendimento apos o segundo mes  
    poupanca = rende_poupanca(poupanca);  
  
    // retirada de 50 reais no terceiro mes  
    poupanca = retirada(poupanca, 50);  
    // rendimento apos o terceiro mes  
    poupanca = rende_poupanca(poupanca);  
  
    printf("Tenho R$ %.2f na poupanca\n", poupanca);  
    return 0;  
}
```

- As funções nós permitem agrupar ações que vão ser repetidas ao longo do código...

```
float rende_poupanca(float saldo) {  
    float juros = 1.01;  
    float taxa = 3.14;  
    return (saldo * juros) - taxa;  
}
```

```
float deposito(float saldo, float valor) {  
    return saldo + valor;  
}
```

```
float retirada(float saldo, float valor) {  
    return saldo - valor;  
}
```



Funções: Conta poupança

Solução 6 – Usando mais funções

- Tanto o “void main” quanto as funções podem estar no mesmo arquivo .c (exemplo: solucao7.c)
- No entanto, pode-se criar um módulo (exemplo: poupanca7_modulo.h) para lidar com as operações padrões do banco

Arquivo poupanca7_modulo.h

```
float rende_poupanca(float saldo);  
float deposito(float saldo, float valor);  
float retirada(float saldo, float valor);
```

Arquivo poupanca7_modulo.c

```
float rende_poupanca(float saldo) {  
    float juros = 1.01;  
    float taxa = 3.14;  
    return (saldo * juros) - taxa;  
}  
  
float deposito(float saldo, float valor) {  
    return saldo + valor;  
}  
  
float retirada(float saldo, float valor) {  
    return saldo - valor;  
}
```

Funções: Conta poupança

Solução 6 – Usando mais funções

- Como compilar esses códigos fonte?

- poupanca7_modulo.h
- poupanca7_modulo.c
- main.c

A terminal window with a black background and white text. The command '\$ gcc -c poupanca7_modulo.c' is entered. The window has a standard Linux-style title bar with a minus sign, a maximize button, and a close button.

```
$ gcc -c poupanca7_modulo.c
```

Gera o arquivo .o



poupanca7_modulo.o

Funções: Conta poupança

Solução 6 – Usando mais funções

```
#include <stdio.h>
```

```
#include "poupanca7_modulo.h"
```

```
int main() {
```

```
    float poupanca = 500.0;
```

```
    // rendimento apos o primeiro mes
```

```
    poupanca = rende_poupanca(poupanca);
```

```
    // deposito de 200 reais no segundo mes
```

```
    poupanca = deposito(poupanca, 200);
```

```
    // rendimento apos o segundo mes
```

```
    poupanca = rende_poupanca(poupanca);
```

```
    // retirada de 50 reais no terceiro mes
```

```
    poupanca = retirada(poupanca, 50);
```

```
    // rendimento apos o terceiro mes
```

```
    poupanca = rende_poupanca(poupanca);
```

```
    printf("Tenho R$ %.2f na poupanca\n", poupanca);
```

```
    return 0;
```

```
}
```

- Para usar as funções que acabamos de criar e compilar, precisamos de incluí-las no nosso código principal!

Funções: Conta poupança

Solução 6 – Usando mais funções

▪ Como compilar esses códigos fonte?

- `poupanca7_modulo.h`
- `poupanca7_modulo.c`
- `main.c`

```
$ gcc -c poupanca7_modulo.c
```

Gera o arquivo .o



`poupanca7_modulo.o`

```
$ gcc poupanca7_main.c poupanca7_modulo.o -o poupanca7_main
```

Gera o programa
`poupanca7_main`



`poupanca7_main`

Processo de compilação

Módulos

■ Passo I:

- Criar dois arquivos com o mesmo nome para o módulo:
 - I arquivo `*.h` contendo só o cabeçalho das funções do módulo
 - `poupanca.h`
 - I arquivo `*.c` contendo a implementação das funções contidas no arquivo `*.h`
 - `poupanca.c`
- Não esqueça de incluir o arquivo `*.h`
 - `#include "poupanca.h"`

Processo de compilação

Módulos

■ Passo 2:

- Compile o módulo criado a partir do arquivo .c criado
 - `gcc -c poupanca.c`
- Se compilar corretamente, você gerou um arquivo .o com o nome do módulo
 - `poupanca.o`

Processo de compilação

Módulos

■ Passo 3:

- crie um arquivo `*.c` para testar o módulo criado
 - `testepoupanca7.c`
- este arquivo deve incluir o `*.h` do módulo criado
 - `#include poupanca.h`

■ Passo 4:

- Compile o arquivo de teste criado fazendo a ligação explícita para o arquivo compilado `.o` do seu módulo
 - `gcc testepoupanca7.c poupanca.o`

■ Passo 5:

- Execute o programa
 - `a.exe` (no windows)
 - `./a` (no linux)

Processo de compilação

Bibliotecas padrão do C – Math.h

- A math.h é especial: precisa de um parâmetro na compilação (somente em sistemas UNIX)
 - `gcc codigo.c -lm`
- `-l"nome"` indica que queremos que o programa “incorpore” código de um módulo externo
- TODA biblioteca precisa do `-l"nome"`
 - EXCETO as funções padrão do C

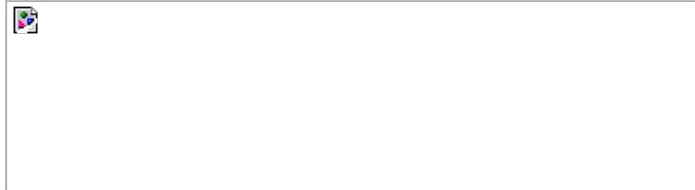
Exercício

Brutus e Olívia

- Brutus e Olívia foram ao médico, que disse a eles que ambos estão fora do peso ideal. Ambos discordaram veementemente da afirmação do médico. Para provar que estava certo, o médico mostrou o Índice de Massa Corporal (IMC) de ambos, considerando que Brutus tem 1,84m e pesa 112kg e Olívia tem 1,76m e pesa 49kg.
- Implemente um programa para:
 - Mostrar o IMC de Brutus e Olívia
 - Quantos kilos Brutus e Olívia devem perder/ganhar para atingirem um peso saudável segundo a classificação do IMC

Exercício

Brutus e Olívia



IMC	Classificação
< 16	Magreza grave
$16 \text{ a } < 17$	Magreza moderada
$17 \text{ a } < 18.5$	Magreza leve
$18.5 \text{ a } < 25$	Saudável
$25 \text{ a } < 30$	Sobrepeso
$30 \text{ a } < 35$	Obesidade grau I
$35 \text{ a } < 40$	Obesidade grau II (severa)
≥ 40	Obesidade grau III (mórbida)

Table 3.1: Classificação do IMC

Exercício

Brutus e Olívia

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

```
#include <stdio.h>
```

```
float calcula_dif_peso(float altura, float peso) {  
    float ic = peso / (altura * altura);  
    float peso_ideal = (altura * altura) * 21;  
    float dif = peso - peso_ideal;  
    printf("IMC=%f, dif=%f\n", ic, dif);  
    return dif;  
}  
  
int main() {  
    float peso_olivia = 45;  
    float altura_olivia = 1.76;  
    float peso_brutus = 122;  
    float altura_brutus = 1.84;  
  
    float dif_olivia = calcula_dif_peso(altura_olivia, peso_olivia);  
    float dif_brutus = calcula_dif_peso(altura_brutus, peso_brutus);  
    float total = fabs(dif_olivia) + fabs(dif_brutus);  
  
    printf("Total de peso que eles devem ganhar o perder: %.2fkg\n", total);  
    return 0;  
}
```

Perguntas?

- E-mail:
 - hector@dcc.ufmg.br
- Material da disciplina:
 - <https://pedroolmo.github.io/teaching/pdsI.html>
- Github:
 - <https://github.com/h3ct0r>



Héctor Azpúrua
h3ct0r