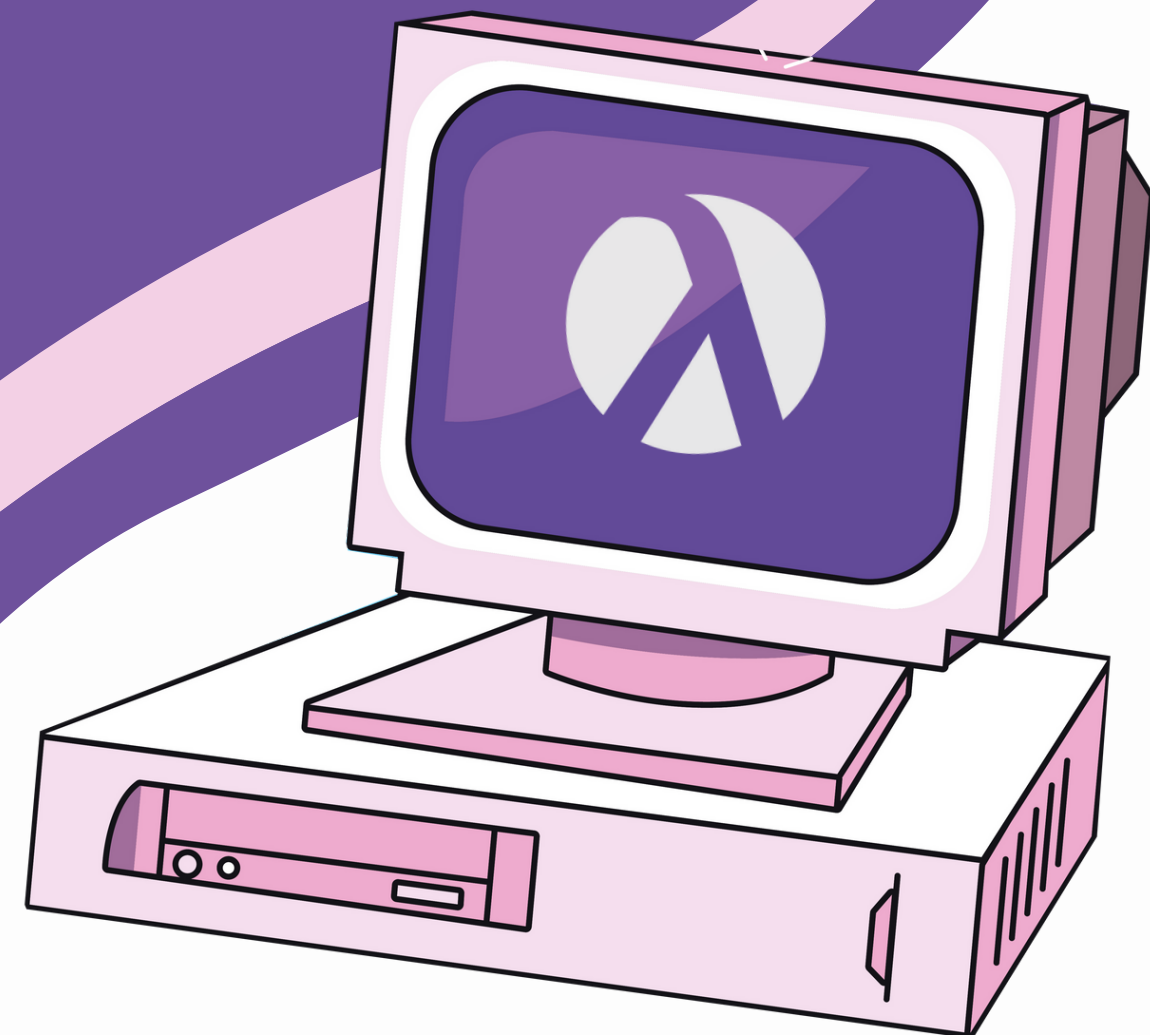


LÓGICA DE PROGRAMAÇÃO E C

DIA 4



Funções



Calma lá, "função"?



- Um bloco de instruções fora da "main"
- É executado ao ser chamado na "main" ou em outras funções
- A "main" é a função principal, que todo programa deve ter

Calma lá, "função"?



- Também chamada de procedimento ou subrotina
- Assim como na Matemática, podem receber valores e retornar valores

$$f(x) = 2x$$

```
int dobro(int x) {  
    |     return 2 * x;  
}
```

Tá, e por que eu usaria?



- **Acima de tudo: simplificação**
- **Menos código pra manter na cabeça, mais claro**
- **Menos repetição, mais conciso**
- **O segredo da programação é manter a complexidade baixa. Simples é melhor!**

Tá, e por que eu usaria?



Você pensa desse jeito?

```
rotina() {  
  desligarDespertador();  
  levantarPe("esquerdo");  
  levantarPe("direito");  
  pular();  
  andarAte("banheiro");  
  pegar("escova");  
  abrir("torneira");  
  escovar("dentes");  
  fechar("torneira");  
  guardar("escova");  
  andarAte("cozinha");  
  abrir("armario");  
  pegar("prato, copo");  
  fechar("armario");  
}
```

Ou desse?

```
rotina() {  
  acordar();  
  escovarDentes();  
  tomar("cafe");  
  ir("trabalho");  
  tomar("almoco");  
  voltar();  
  tomar("janta");  
  dormir();  
}
```



Tá, e por que eu usaria?

Isso é mais claro?

```
int main() {  
    // ler dados do usuário  
    printf("Insira um numero: ");  
    double x = lerNumero();  
    printf("Insira uma operacao (+, -, /, *): ");  
    char op = lerOperacao();  
    printf("Insira outro numero: ");  
    double y = lerNumero();  
  
    // calcular resultado  
    double resultado = calcular(x, op, y);  
  
    // mostrar resultado  
    printf("%lf %c %lf = %lf\n", x, op, y, resultado);  
}
```



Tá, e por que eu usaria?

Ou isso?

```
main() {
// ler dados do usuário
printf("Insira um numero: ");
double x = 0;
while (1) {
    int r = scanf(" %lf", &x);
    flush_in();
    if (r == 1) {
        break;
    } else {
        puts("Erro de leitura.");
    }
}
printf("Insira uma operacao (+, -, /, *): ");
char op = 0;
int r;
while (1) {
    r = scanf(" %c", &op);
    flush_in();
    if (r == 1) {
        switch (op) {
            case '+':
            case '-':
            case '*':
```

```
            case '*':
            case '/':
                goto post;
            default:
                puts("Operacao invalida.");
        }
    } else {
        puts("Erro de leitura.");
    }
}
post:
printf("Insira outro numero: ");
double y = 0;
while (1) {
    int r = scanf(" %lf", &y);
    flush_in();
    if (r == 1) {
        break;
    } else {
        puts("Erro de leitura.");
    }
}
```

```
    } else {
        puts("Erro de leitura.");
    }
}
// calcular resultado
double resultado = 0.0;
switch (op) {
    case '+': resultado = x + y; break;
    case '-': resultado = x - y; break;
    case '/': resultado = x / y; break;
    case '*': resultado = x * y; break;
}
// mostrar resultado
printf("%lf %c %lf = %lf\n", x, op, y, re
```



Tá, e por que eu usaria?



```
int main() {  
    // ler dados do usuário  
    printf("Insira um numero: ");  
    double x = lerNumero();  
    printf("Insira uma operacao (+, -, /, *): ");  
    char op = lerOperacao();  
    printf("Insira outro numero: ");  
    double y = lerNumero();  
  
    // calcular resultado  
    double resultado = calcular(x, op, y);  
  
    // mostrar resultado  
    printf("%lf %c %lf = %lf\n", x, op, y, resultado);  
}
```

- **Permite que você pense sobre o seu programa em um nível mais alto**

Tá, e por que eu usaria?



```
double lerNumero() {  
    while (1) {  
        double l = 0;  
        int r = scanf(" %lf", &l);  
        flush_in();  
        if (r == 1) {  
            return l;  
        } else {  
            puts("Erro de leitura.");  
        }  
    }  
}
```

- **Evita repetição: consertar um erro em um lugar é consertar em todos**

Tá, e por que eu usaria?



```
char lerOperacao() {
    while (1) {
        char ch = 0;
        int r = scanf(" %c", &ch);
        flush_in();
        if (r == 1) {
            switch (ch) {
                case '+':
                case '-':
                case '*':
                case '/':
                    return ch;
                default:
                    puts("Operacao invalida.");
            }
        } else {
            puts("Erro de leitura.");
        }
    }
}
```

- **Oculto complexidade:**
apresenta operações complicadas de uma forma mais simples

Tá, e por que eu usaria?




```
double resultado(double x, char op, double y) {  
    switch (op) {  
        case '+': return x + y;  
        case '-': return x - y;  
        case '/': return x / y;  
        case '*': return x * y;  
    }  
    return -1.0;  
}
```

- **Generaliza: uma mesma função pode fazer vários papéis!**

E como fica em C?

```
tipo Nome(parâmetros) {  
    instruções;  
    retorno;  
}
```






```
tipo Nome(parâmetros) {  
    instruções;  
    retorno;  
}
```

tipo:

- **Tipo do valor retornado**
- **Mesmos tipos das variáveis**
 - int, float, char, bool
- **Sem retorno: void**

retorno:

- **Pode ou não existir**
- **Retornado um valor do tipo já especificado**



```
tipo Nome(parâmetros) {  
    instruções;  
    retorno;  
}
```

nome:

- Mesmas restrições de nomenclatura de variáveis
- Não pode se chamar "main"

parâmetros:

- Podem ou não existir
- Valores recebidos de outras funções
- Como variáveis:
(tipo nome, tipo nome, ...)

EXERCÍCIO

O que as seguintes funções recebem e retornam?



1.

```
bool par(int a) {  
    |     return a % 2 == 0;  
}
```

2.

```
float div(float a) {  
    |     return a / 2.5;  
}
```

3.

```
float comprimento(float x, float y) {  
    |     return sqrtf(x*x + y*y);  
}
```


EXERCÍCIO

O que as seguintes funções recebem e retornam?

```
1. bool par(int a) {  
    |     return a % 2 == 0;  
    |  
    }  
}
```

R: A função `par` recebe um inteiro e retorna um booleano indicando se o número é par.



EXERCÍCIO

O que as seguintes funções recebem e retornam?

```
2. float div(float a) {  
    |     return a / 2.5;  
    |  
    }  
}
```

R: A função `div` recebe um `float` e também retorna um `float`, resultado da divisão por 2,5.



EXERCÍCIO

O que as seguintes funções recebem e retornam?

```
3. float comprimento(float x, float y) {  
    |     return sqrtf(x*x + y*y);  
    |  
    }  
}
```

R: A função `comprimento` recebe um par de floats `x`, `y` e retorna a raiz quadrada da soma dos quadrados de `x` e `y` como floats (eita!)
(ou a hipotenusa de um triângulo de catetos `x`, `y`)
(obs: requer a biblioteca `<math.h>`)



Enviando valores

"Passagem de parâmetros"

Por valor

- Valor da variável no local de onde a função foi chamada **NÃO** muda

Por referência

- Se o valor for alterado, o valor no local de onde a função foi chamada também **SERÁ** alterado



Passagem por valor:



```
3  int soma(int a, int b) 10  int main() {
4  {                      11      int a = 3;
5      int c = a + b;      12      int b = 2;
6                          13      int resultado;
7      return c;           14
8  }                      15      resultado = soma(a, b);
                          16  }
```

Qual o valor de *a*, *b* e *resultado* em main após a linha 15?

Passagem por valor:



```
3  int soma(int a, int b) 10  int main() {
4  {                      11      int a = 3;
5      int c = a + b;      12      int b = 2;
6                          13      int resultado;
7      return c;           14
8  }                      15      resultado = soma(a, b);
                          16  }
```

Qual o valor de *a*, *b* e *resultado*
em main após a linha 15?

a = 3 *b* = 2
resultado = 5

Passagem por valor:



```
3  int soma(int a, int b)
4  {
5      a = 8;
6      int c = a + b;
7
8      return c;
9  }
```

```
10 int main() {
11     int a = 3;
12     int b = 2;
13     int resultado;
14
15     resultado = soma(a, b);
16 }
```

Qual o valor de *a*, *b* e *resultado* em main após a linha 15?

Passagem por valor:



```
3  int soma(int a, int b)
4  {
5      a = 8;
6      int c = a + b;
7
8      return c;
9  }
```

```
10 int main() {
11     int a = 3;
12     int b = 2;
13     int resultado;
14
15     resultado = soma(a, b);
16 }
```

Qual o valor de *a*, *b* e *resultado*
em main após a linha 15?

a = 3 *b* = 2
resultado = 10

Passagem por referência:



```
3  int soma(int *a, int *b)
4  {
5      int c = *a + *b;
6
7      return c;
8  }

10 int main() {
11     int a = 3;
12     int b = 2;
13     int resultado;
14
15     resultado = soma(&a, &b);
16 }
```

Qual o valor de *a*, *b* e *resultado* em main após a linha 15?

Passagem por referência:



```
3  int soma(int *a, int *b)
4  {
5      int c = *a + *b;
6
7      return c;
8  }

10 int main() {
11     int a = 3;
12     int b = 2;
13     int resultado;
14
15     resultado = soma(&a, &b);
16 }
```

Qual o valor de *a*, *b* e *resultado*
em main após a linha 15?

a = 3 *b* = 2
resultado = 5

Passagem por referência:



```
3  int soma(int *a, int *b)
4  {
5      *a = 8;
6      int c = *a + *b;
7
8      return c;
9  }

10 int main() {
11     int a = 3;
12     int b = 2;
13     int resultado;
14
15     resultado = soma(&a, &b);
16 }
```

Qual o valor de *a*, *b* e *resultado* em main após a linha 15?

Passagem por referência:



```
3  int soma(int *a, int *b)
4  {
5      *a = 8;
6      int c = *a + *b;
7
8      return c;
9  }

10 int main() {
11     int a = 3;
12     int b = 2;
13     int resultado;
14
15     resultado = soma(&a, &b);
16 }
```

Qual o valor de *a*, *b* e *resultado*
em main após a linha 15?

a = 8 *b* = 2
resultado = 10

Observação:

```
3  struct n {
4  |    int membro;
5  };
6  typedef struct n n;
7
8  void quadruplica(n *estrut) {
9  |    (*estrut).membro = (*estrut).membro * 2;
10 |    // igual a
11 |    estrut->membro = estrut->membro * 2;
12 }
13
14 int main() {
15 |    n hi;
16 |    hi.membro = 2;
17 |    quadruplica(&hi);
18 |    // hi.membro == 8
19 }
```

- Para acessar o membro de um struct passado por referência, utiliza-se **`(*estrut).membro`** ou **`estrut->membro`** (são formas equivalentes).



Retornando valores



- Não retornando:
 - Função tipo **void**
 - Sem "return"

```
4 void par_ou_impar(int n) {  
5     if (n % 2 == 0) {  
6         printf("%d: Par", n);  
7     } else {  
8         printf("%d: Impar", n);  
9     }  
10 }  
11  
12 int main() {  
13     int n = 3;  
14     par_ou_impar(n);  
15 }
```

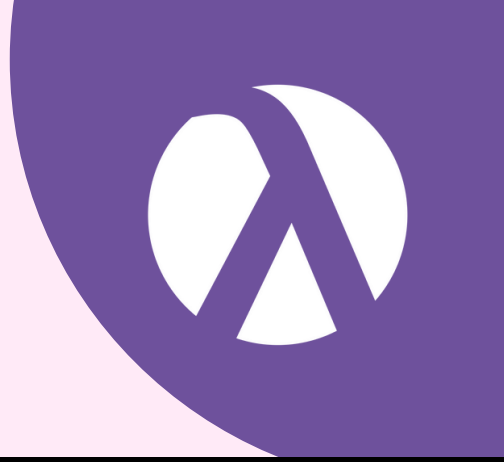
Retornando valores



- Retornando:
 - Função e variável que recebe devem ser do mesmo tipo

```
4  int ao_quadrado(int n) {
5      int nn;
6      nn = n * n;
7      return nn;
8  }
9
10 int main() {
11     int n = 2;
12     int quadrado = ao_quadrado(n);
13 }
14
```

Retornando valores



- Por referência:
 - Os parâmetros são modificados pela função
 - Útil quando for necessário retornar mais de um valor ou modificar structs
 - Exemplo: scanf!

```
1  #include <stdio.h>
2  #include <math.h>
3
4  void raizes(
5      float a, float b, float c,
6      float *x1, float *x2
7  ) {
8      float delta = b*b - 4.0*a*c;
9
10     if (delta >= 0.0) {
11         float raiz_delta = sqrtf(delta);
12
13         *x1 = (-b - raiz_delta)/2.0;
14         *x2 = (-b + raiz_delta)/2.0;
15     }
16 }
17
18 int main() {
19     float x, y;
20     raizes(1, -4, 4, &x, &y);
21     printf("Raizes de x² - 4x + 4: %f e %f\n", x, y);
22 }
23
```


Observação:



- **Vetores, strings, matrizes e similares são sempre passados por referência, isto é, nunca são copiados**
- **O tamanho dos vetores e matrizes é perdido quando passado e deve ser fornecido como parâmetro extra**

Observação:

- Exemplo: função que dobra todos os elementos de um vetor

```
3 void dobraVetor(int tam, int vetor[]) {
4     for (int i = 0; i < tam; i++) {
5         vetor[i] = vetor[i] * 2;
6     }
7 }
8
9 int main() {
10     int lista[] = {1, 2, 3, 4};
11     dobraVetor(4, lista);
12     // lista = {2, 4, 6, 8}
13 }
14
```

EXERCÍCIO

Crie uma função que receba o seguinte struct passado por referência e dobre o seu valor interno.

```
typedef struct S {  
    int interno;  
} S;  
  
int main(void) {  
    S str;  
    str.interno = 2;  
    dobraStruct(&str); // <- - !!!!  
    printf("%d", str.interno == 4);  
}
```

EXERCÍCIO

Crie uma função que receba o seguinte struct passado por referência e dobre o seu valor interno.

R:

```
void dobraStruct(S* str) {
    str->interno *= 2;
}

// ou...

void dobraStruct(S* str) {
    (*str).interno *= 2;
}
```

```
typedef struct S {
    int interno;
} S;

int main(void) {
    S str;
    str.interno = 2;
    dobraStruct(&str);
    printf("%d", str.interno == 4);
}
```



EXERCÍCIO

Crie uma função que receba dois inteiros e retorne o maior deles.

EXERCÍCIO

Crie uma função que receba dois inteiros e retorne o maior deles.

R:

```
1  #include <stdio.h>
2
3  int maior(int a, int b)
4  {
5      if (a > b)
6      |   return a;
7      else if (b > a)
8      |   return b;
9      else
10     {
11         printf("Os valores sao iguais.");
12         return a;
13     }
14 }
```

```
16  int main() {
17      |   int a, b, maiorValor;
18      |   maiorValor = maior(a,b);
19      }
```



Mais exemplos!



- Nós já utilizamos várias funções até agora!
 - `scanf`, `printf`, `sqrt`, `puts`,
`getchar`...
- Funções das bibliotecas `stdio.h` e `math.h`

Ordem de declaração

- O compilador lê o código de cima para baixo, linha por linha, apenas uma vez;
- Funções devem ser declaradas antes de serem utilizadas, pois antes da declaração, o compilador não as conhece.
- Tranquilo, certo?



Ordem de declaração

Como eu faço esse código compilar?

```
1  #include <stdio.h>
2
3  int main() {
4      |   haha();
5      |
6
7      void haha() {
8          |   if (2 + 2 == 5) {
9              |       printf("Ta safe !\n");
10             |   } else {
11                 |       printf("Eita !\n");
12                 |   }
13             |
14         }
```

Ordem de declaração

Como eu faço esse código compilar?

```
1  #include <stdio.h>
2
3  void haha() {
4      if (2 + 2 == 5) {
5          printf("Ta safe !\n");
6      } else {
7          printf("Eita !\n");
8      }
9  }
10
11 int main() {
12     haha();
13 }
14
```



Ordem de declaração

Como eu faço esse código compilar?

```
1  #include <stdio.h>
2
3  void b() {
4      |    a();
5  }
6
7  void a() {}
8
9  int main() {
10     |    b();
11 }
12
```

Ordem de declaração

Como eu faço esse código compilar?

```
1  #include <stdio.h>
2
3  void a() {}
4
5  void b() {
6      |    a();
7  }
8
9  int main() {
10     |    b();
11 }
12
```



Ordem de declaração

E agora? Como eu faço para compilar?

```
1  #include <stdio.h>
2
3  int foo(int x) {
4      |   if (x == 0) printf("Foo!");
5      |   else bar(x-1);
6      |
7  }
8
9  int bar(int y) {
10     |   if (y == 0) printf("Bar!");
11     |   else foo(y-1);
12     |
13 }
14
15 int main() {
16     |   foo(9); // Bar!
17 }
```

Ordem de declaração

E agora? Como eu faço para compilar?

```
1  #include <stdio.h>
2
3  int foo(int x) {
4      if (x == 0) printf("Foo!");
5      else bar(x-1);
6  }
7
8  int bar(int y) {
9      if (y == 0) printf("Bar!");
10     else foo(y-1);
11 }
12
13 int main() {
14     foo(9); // Bar!
15 }
16
```

- Note: é *impossível* reordenar o código de modo a fazer o programa compilar sem avisos.
- Se foo vem antes de bar: erro
- Se bar vem antes de foo: erro
- Mas esse é um algoritmo válido. E agora?

Ordem de declaração



```
1  #include <stdio.h>
2
3  int foo(int x);
4  int bar(int y);
5
6  int foo(int x) {
7      if (x == 0) printf("Foo!");
8      else bar(x-1);
9  }
10
11 int bar(int y) {
12     if (y == 0) printf("Bar!");
13     else foo(y-1);
14 }
15
16 int main() {
17     foo(0); // Bar!
```

- Escreva o nome e o tipo da função e de seus parâmetros no topo do arquivo.
- Essa linha é uma "declaração" da função.
- Funções podem ter N declarações mas apenas 1 definição (código).
- Uma definição contém uma declaração.

Ordem de declaração



```
/* Write formatted output to STREAM.

   This function is a possible cancellation point and therefore
   marked with __THROW.  */
extern int fprintf (FILE *__restrict __stream,
                  | const char *__restrict __format, ...);
/* Write formatted output to stdout.

   This function is a possible cancellation point and therefore
   marked with __THROW.  */
extern int printf (const char *__restrict __format, ...);
/* Write formatted output to S.  */
extern int sprintf (char *__restrict __s,
                  | const char *__restrict __format, ...) __THROWNL;
/* Write formatted output to S from argument list ARG.

   This function is a possible cancellation point and therefore
   marked with __THROW.  */
extern int vfprintf (FILE *__restrict __s, const char *__res
                  | __gnuc_va_list __arg);
/* Write formatted output to stdout from argument list ARG.
```

- Uma curiosidade: bibliotecas são geralmente coleções de declarações de funções!
- As definições dessas funções residem nos arquivos .dll e .so do seu sistema.

Um ponto sobre variáveis



- O lugar onde as variáveis são declaradas define onde elas podem ser acessadas.
- A parte do código onde uma variável pode ser acessada (é *vísivel*) é denominada de *escopo da variável*.
- O escopo de uma variável começa na sua declaração e termina ao fim do bloco ou arquivo.

Um ponto sobre variáveis



- Variáveis declaradas dentro de funções ou blocos tem **escopo local** e só podem ser acessadas dentro do bloco onde foram declaradas.
- Variáveis declaradas fora de funções ou blocos, livres, tem **escopo global** e podem ser acessadas em todo o programa.

Um ponto sobre variáveis

```
int main() {  
    int x = 1;  
    // cálculos...  
    int x = 2; // erro!  
}
```



```
int main() {  
    if (2 + 2 == 4) {  
        int x = 1;  
    }  
  
    if (2 + 3 == 5) {  
        int x = 2;  
    }  
}
```



- Não é possível ter duas variáveis com o mesmo nome no mesmo escopo...
- ...mas variáveis com o mesmo nome podem ser declaradas em escopos diferentes.



Um ponto sobre variáveis



```
// x ainda não existe
```

```
int x = 2;
```

```
// x = 2
```

```
{
```

```
    // x = 2
```

```
    int x = 3;
```

```
    // x = 3
```

```
    {
```

```
        x = 4;
```

```
    }
```

```
    // x = 4
```

```
}
```

```
// x = 2
```

- Um bloco interno a outro bloco pode ocultar variáveis declarando novas variáveis com um nome já utilizado.
- Caso existam várias variáveis com um mesmo nome, a que for visível e mais interna será priorizada.

EXERCÍCIO



- **Escreva uma função que receba um vetor de inteiros e encontre o maior elemento desse vetor.**
- **Escreva uma função que receba um vetor de inteiros e um elemento e conte quantas vezes esse elemento ocorre no vetor.**
- **Use as duas funções para contar o número de vezes que o maior elemento de um vetor ocorre neste.**

EXERCÍCIO

- Escreva uma função que receba um vetor de inteiros e encontre o maior elemento desse vetor.

R:

```
1  int maiorDe(int n, int vetor[]) {  
2      int maior = vetor[0];  
3      for (int i = 0; i < n; i++) {  
4          if (vetor[i] > maior) {  
5              maior = vetor[i];  
6          }  
7      }  
8      return maior;  
9  }
```



EXERCÍCIO

- Escreva uma função que receba um vetor de inteiros e um elemento e conte quantas vezes esse elemento ocorre no vetor.

R:

```
1  int contaEm(int n, int vetor[], int elem) {  
2      int conta = 0;  
3      for (int i = 0; i < n; i++) {  
4          if (vetor[i] == elem) {  
5              conta++;  
6          }  
7      }  
8      return conta;  
9  }
```

EXERCÍCIO

- Use as duas funções para contar o número de vezes que o maior elemento de um vetor ocorre neste.

R:

```
1  int main(void) {  
2      int vetor[8] = {1, 10, 3, 6, 3, 0, -1, 10};  
3      int maior = maiorDe(8, vetor);  
4      int n = contaEm(8, vetor, maior);  
5      printf("Maior: %d (%d vezes)", maior, n);  
6  }
```


EXERCÍCIO



- A sequência de Fibonacci é uma sequência numérica cujos termos são iguais à soma dos dois termos anteriores, e cujos dois primeiros termos são 1 e 1.

1, 1, 2, 3, 5, 8, 13, 21, 34 ...

2 = 1+1, 3=1+2, 5=2+3, ...

- Escreva uma função que retorne o n-ésimo termo da sequência de Fibonacci.

EXERCÍCIO

- Escreva uma função que retorne o n-ésimo termo da sequência de Fibonacci.

R:

```
1  int fib(int n) {
2      int a = 1;
3      int b = 1;
4
5      for (int i = 3; i <= n; i++) {
6          int s = a + b;
7          a = b;
8          b = s;
9      }
10
11     return b;
12 }
```

Cadeias de caracteres



O que são cadeias de caracteres?



- Cadeias de caracteres, ou strings, são sequências de caracteres que determinam texto.
- No código, são obtidas colocando-se texto entre aspas duplas.

Conte-me mais.



```
"Legal!";  
"Eitcha nois!";  
"Alerta! Quebra de linha a seguir: \n";
```

- **Strings podem conter caracteres especiais que representam quebras de linha ('\\n'), tabulações ('\\t'), o caractere nulo ('\\0'), etc.**

Conte-me mais.



```
printf("Eita," " nos!");  
// Eita, nos!
```

- Se dois strings delimitados por aspas estiverem justapostos, eles serão unidos.

Conte-me mais.

```
char str[] = "Hello!";  
  
char str[7] = "Hello!";  
  
char str[] =  
    {'H', 'e', 'l', 'l', 'o', '!', '\0'};  
  
char str[7];  
str[0] = 'H';  
str[1] = 'e';  
str[2] = 'l';  
str[3] = 'l';  
str[4] = 'o';  
str[5] = '!';  
str[6] = '\0';
```

- Em C, strings são representados por vetores do tipo char, cujo fim é indicado pelo char nulo, `\0`.
- As definições ao lado são equivalentes.



Conte-me mais.

```
char str[] = "Hello!";  
  
char str[7] = "Hello!";  
  
char str[] =  
    {'H', 'e', 'l', 'l', 'o', '!', '\0'};  
  
char str[7];  
str[0] = 'H';  
str[1] = 'e';  
str[2] = 'l';  
str[3] = 'l';  
str[4] = 'o';  
str[5] = '!';  
str[6] = '\0';
```

- Quando um string entre aspas é lido pelo compilador, é adicionado um `\0` implicitamente ao fim do texto.
- É sempre reservado um espaço a mais no vetor para o char nulo, `'\0'`.



Observação



- O char nulo ' $\backslash 0$ ' **não é** o char zero '0'!
 - O valor ASCII do número 0 é 48.
 - O valor ASCII do char nulo é 0.
 - Cuidado para não confundir os dois.
 - Strings podem conter quantos números zero você quiser.

Observação

- Em C, um vetor de chars que não termina com um char `'\0'` **não é considerado um string e não deve ser passado** para as funções da biblioteca `string.h`, pois elas irão corromper a memória.
 - Isso ocorre pois a biblioteca lê a memória até encontrar um char `'\0'`, o fim de uma string. Caso esse `'\0'` não exista, elas continuam lendo além da memória do string (e isso causa problemas!)



Biblioteca `string.h`



- **Contém funções para manipular cadeias de caracteres**
- **Existem quatro funções principais: `strlen()`, `strcpy()`, `strcmp()` e `strcat()`**

strlen(string)



- Retorna o tamanho de uma string (a posição do caractere '\0')

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main() {
5      char string[4] = "UEM";
6      int tamanho = strlen(string);
7      printf("%d", tamanho);
8  }
9
```

strcpy(to, from)

- Copia uma string que está em uma determinada variável para outra

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main() {
5      char string[4] = "UEM";
6      char string2[4];
7      strcpy(string2, string);
8      printf("%s", string2);
9      // UEM
10 }
11
```

strcat(l, r)

- Concatena duas strings

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main() {
5      char string[19] = "PET";
6      char string2[] = " Informatica";
7      strcat(string, string2);
8      printf("%s", string); // PET Informatica
9  }
10
```

strcmp(a, b)

- Compara duas strings e determina a ordem (alfabética) das duas

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main() {
5      char string[] = "a";
6      char string1[] = "b";
7      int resposta;
8      resposta = strcmp(string, string1);
9      printf("%d", resposta); // -1
10     // pois a < b
11 }
12
```

se a precede b -> -1
se a sucede b -> 1
se a igual a b -> 0



strcmp(a, b)

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main() {
5      char a[5] = "baba";
6      char b[5] = "abab";
7      int res = strcmp(a, b);
8      if (res < 0) {
9          // a < b, a vem antes de b
10     } else if (res == 0) {
11         // a == b, a eh igual a b
12     } else if (res > 0) {
13         // a > b, a vem depois de b
14     }
15 }
16
```

- **Dica: para não confundir, compare o resultado com o e observe o operador.**



EXERCÍCIO

- Sabendo que strings são vetores de char terminadas pelo char nulo ('\0'), reimplemente a função `strlen` da biblioteca `<string.h>`, que recebe uma string e retorna o comprimento desta string.

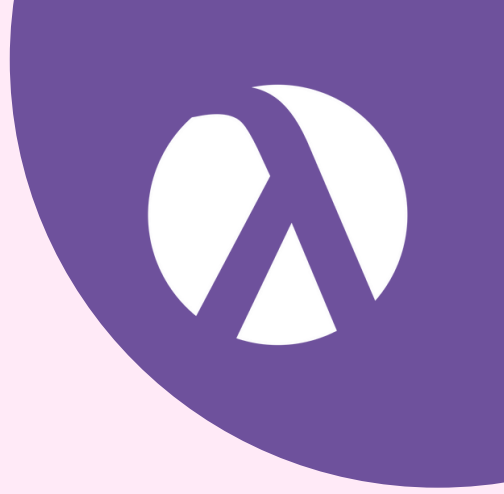


EXERCÍCIO

- Reimplemente a função `strlen` da biblioteca `<string.h>`, que recebe uma `string` e retorna o comprimento desta `string`.

R:

```
1  int strlen(char string[]) {  
2      int i = 0;  
3      while (string[i] != '\0') {  
4          i++;  
5      }  
6      return i;  
7  }  
8
```



EXERCÍCIO

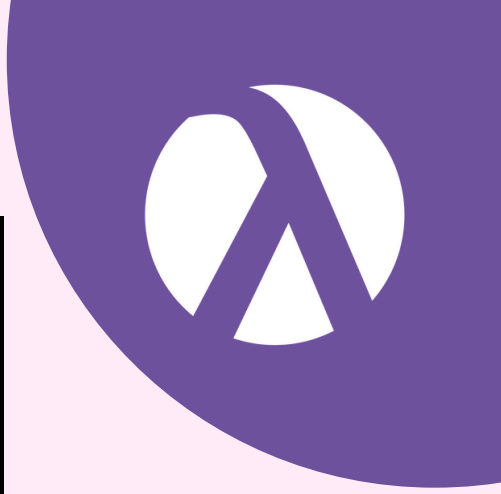


- **Faça um programa que verifique se uma palavra é considerada um palíndromo.**
- **Exemplo: a palavra arara é palíndromo, pois a sua inversão é igual ao original.**
- **arara = arara**
- **ovo = ovo**
- **reviver = reviver**
- **ônibus != subino**
- **pessoa != aossep**
- **live != evil**

EXERCÍCIO

R:

```
1  #include <string.h>
2  #include <stdio.h>
3
4  int main(){
5      char palavra[30], invertePalavra[30];
6      int tam = 0;
7
8      printf("Digite uma palavra: ");
9      scanf("%s", palavra);
10
11     tam = strlen(palavra);
12     for (int i = 0; i < tam; i++) {
13         invertePalavra[i] = palavra[tam - i - 1];
14     }
15     invertePalavra[tam] = '\0';
16
17     if (strcmp(invertePalavra, palavra) == 0) {
18         printf("É palindromo\n");
19     } else {
20         printf("Nao é palindromo\n");
21     }
22 }
```



PROJETINHO



Projeto

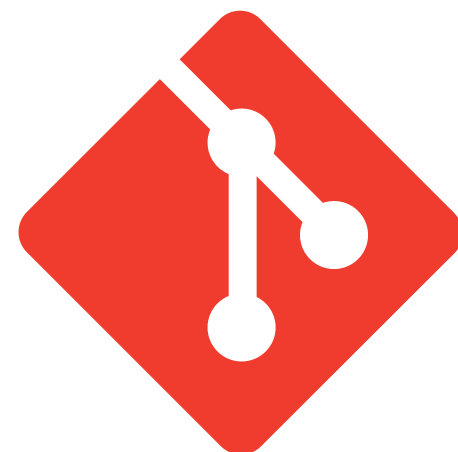
- Crie uma função cadastro que inicialize os dados dos produtos;
- Crie uma função menu que recebe como parâmetros o nome do usuário (cliente, gerente) e o nome da operação e pede para o usuário selecionar um produto ou sair da operação, retornando a escolha;
- Crie duas funções, cliente e gerente, que usam a função menu para executar as operações do cliente e do gerente;
- Use essas funções na função main.



WORKSHOP GIT E GITHUB



DIAS 03 E 10 DE
SETEMBRO



DAS 14 ÀS 16 HORAS!



Inscrições
abertas!



Vagas
limitadas!



Bloco C 56
(DIN)



MUITO OBRIGADO!

 @petinfouem

 pet@din.uem.br

 petinformaticauem

 discord.gg/5JaS4p4mWJ

