

ED62A-COM ESTRUTURAS DE DADOS

Aula 01 - Revisão de fundamentos

Prof. Rafael G. Mantovani

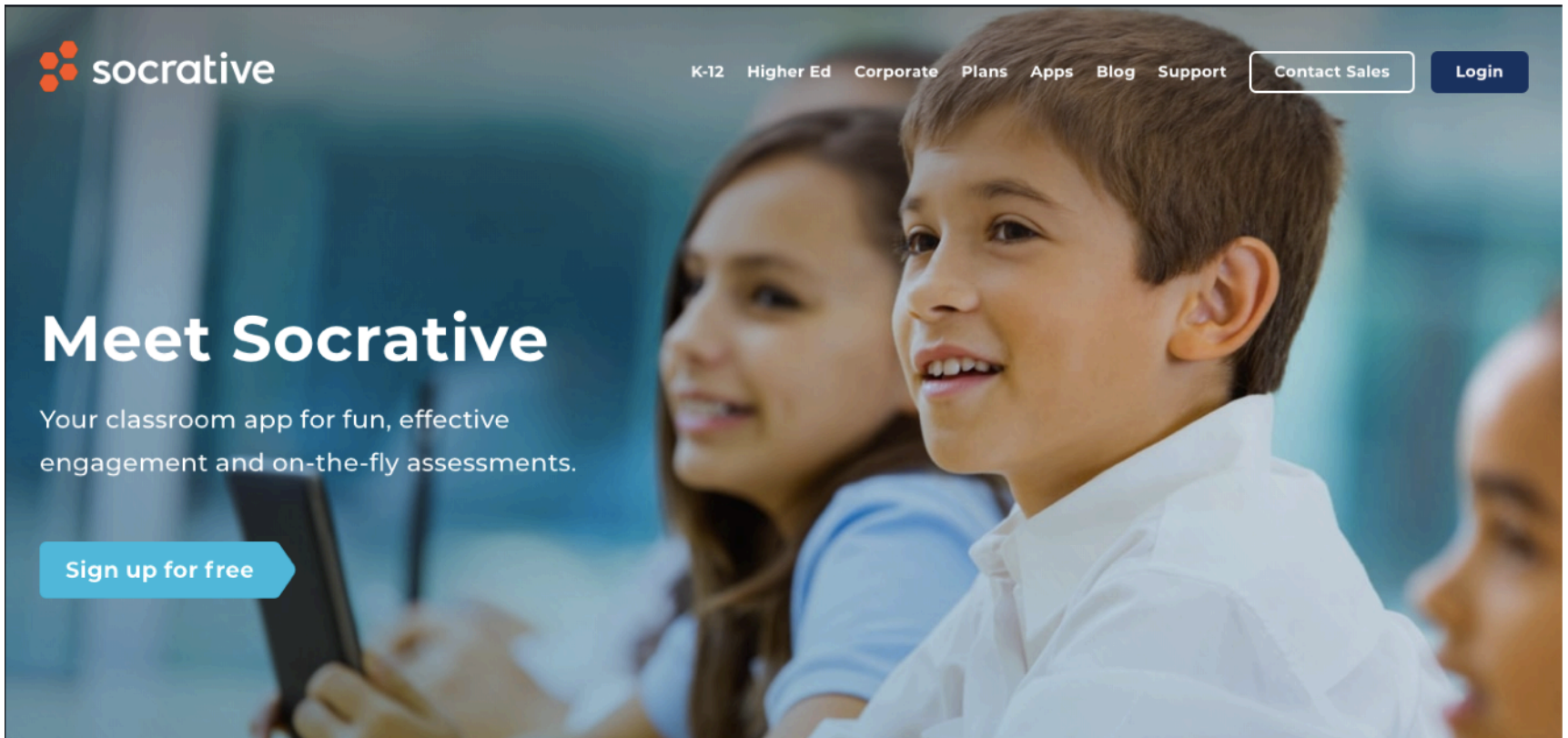
Roteiro

- 1 Quiz**
- 2 Ponteiros**
- 3 Alocação Dinâmica de Memória**
- 4 Recursividade**
- 5 Tipos Abstratos de Dados**
- 6 Arquivos**
- 7 Referências**

Roteiro

- 1 Quiz**
- 2 Ponteiros**
- 3 Alocação Dinâmica de Memória**
- 4 Recursividade**
- 5 Tipos Abstratos de Dados**
- 6 Arquivos**
- 7 Referências**

Quiz

A banner for the Socrative website featuring a blurred background image of three students in a classroom. A girl on the left is holding a tablet, a boy in the center is looking forward with a slight smile, and a girl on the right is partially visible. The Socrative logo is in the top left, and navigation links are in the top right. The main headline 'Meet Socrative' is on the left, followed by a sub-headline and a 'Sign up for free' button.

socrative

K-12 Higher Ed Corporate Plans Apps Blog Support [Contact Sales](#) [Login](#)

Meet Socrative

Your classroom app for fun, effective engagement and on-the-fly assessments.

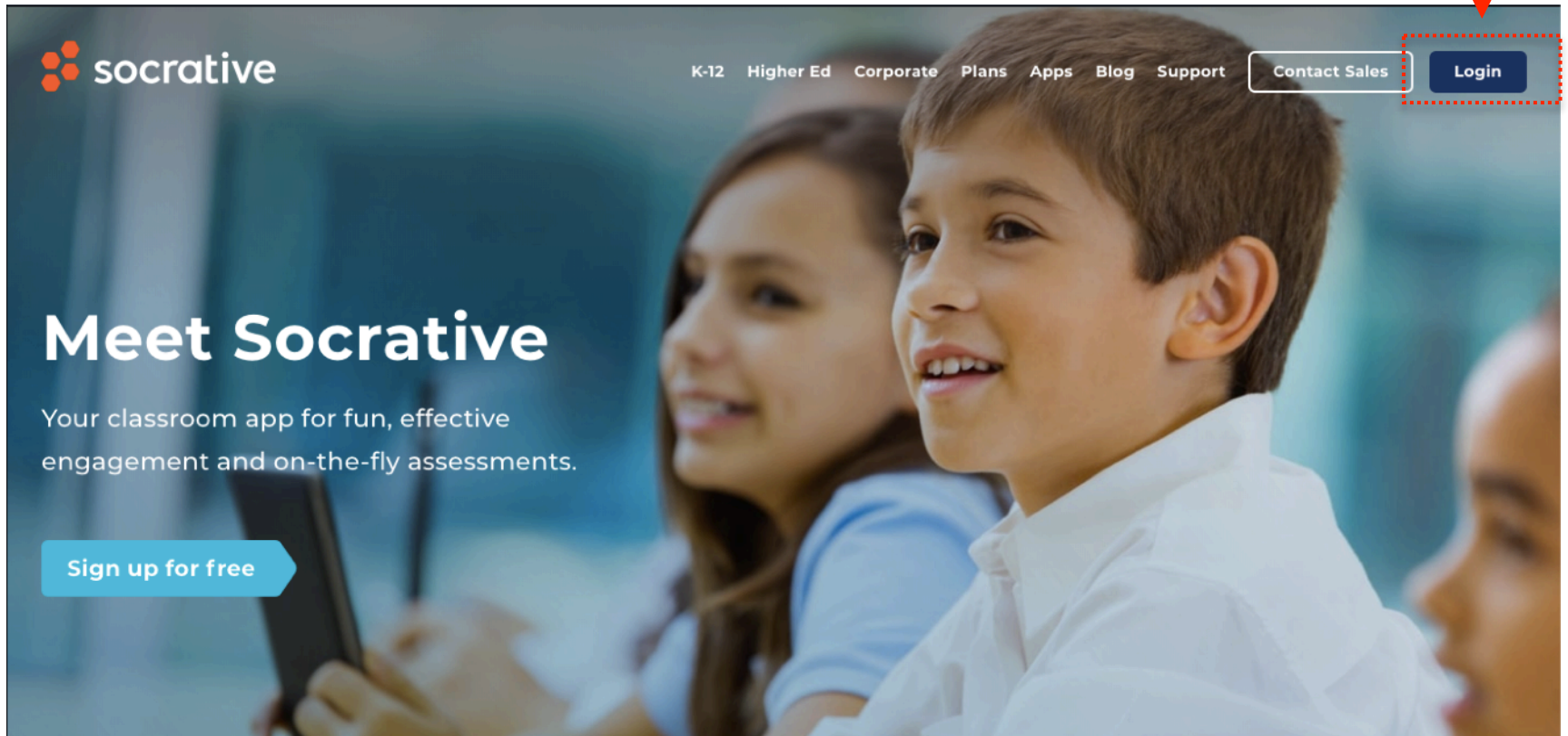
[Sign up for free](#)

Quiz

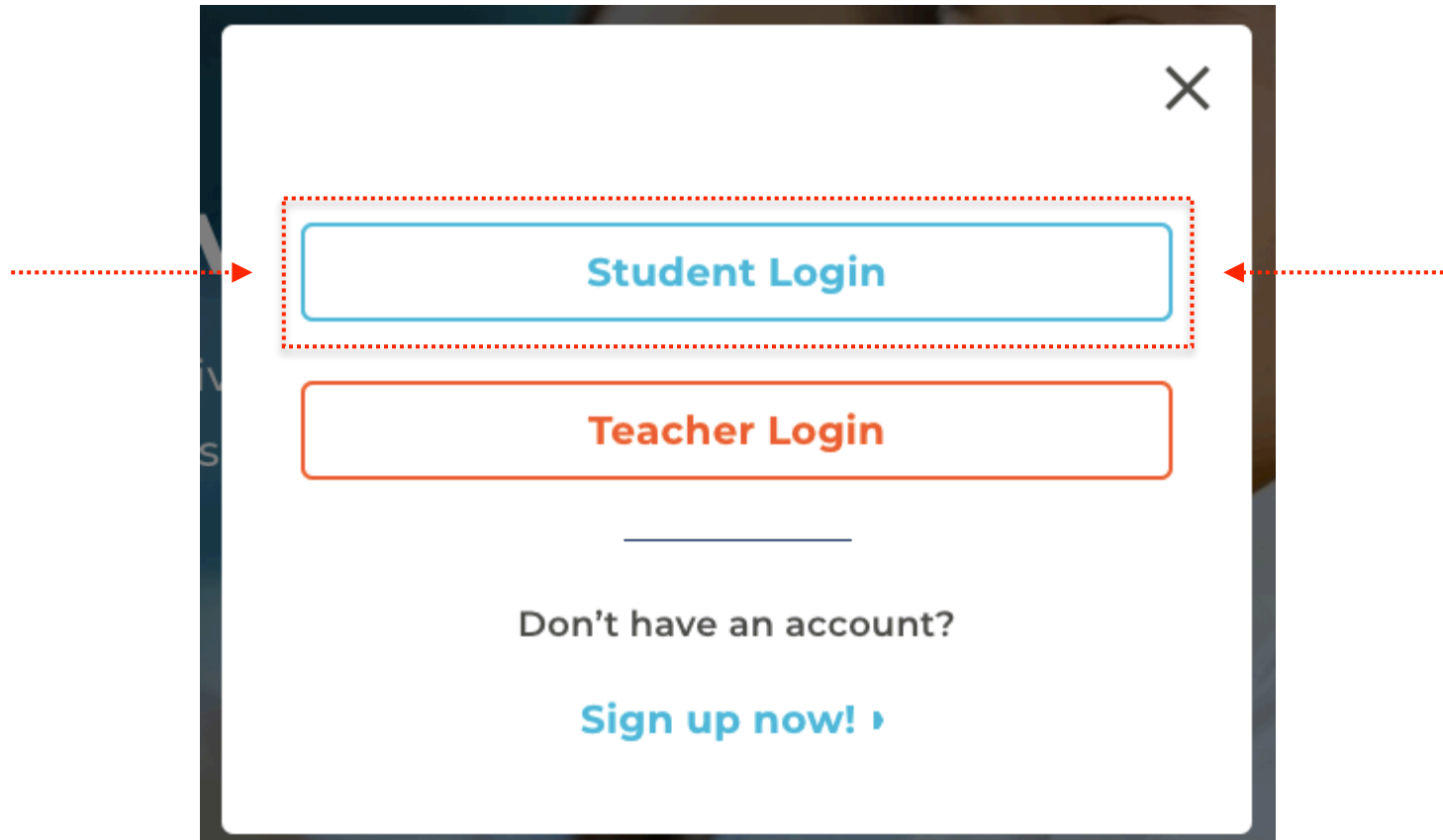


- Instruções:
 1. www.socrative.com
 2. Student login
 3. EDUTFPR2020

Quiz



Quiz



Quiz



Student Login

Room Name

JOIN

 English ▾

Quiz



Student Login

Room Name

EDUTFPR2020

JOIN

 English ▾

Quiz

EDUTFPR2019

Enter your name

Mott, Wilfred

DONE

Quiz

EDUTFPR2019

Deu Certo :))

Enter your name

Mott, Wilfred

DONE

Quiz



10 minutos (máximo)
Tempooooooooo

Roteiro

- 1 Quiz
- 2 Ponteiros
- 3 Alocação Dinâmica de Memória
- 4 Recursividade
- 5 Tipos Abstratos de Dados
- 6 Arquivos
- 7 Referências

Ponteiros

- O que são **ponteiros/apontadores**?

Ponteiros

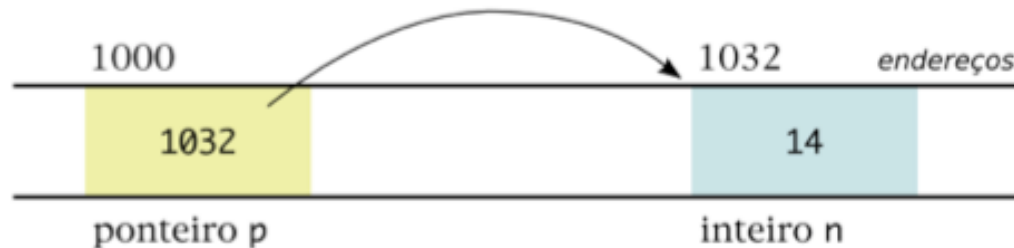
- Considerando uma variável declarada como:

```
int* p;
```

- p*** é um ponteiro para **int**, isto é, uma variável que **armazena o endereço de uma variável** do tipo **int**.
- Supondo que ***p*** armazene o valor 1032, tem-se que:

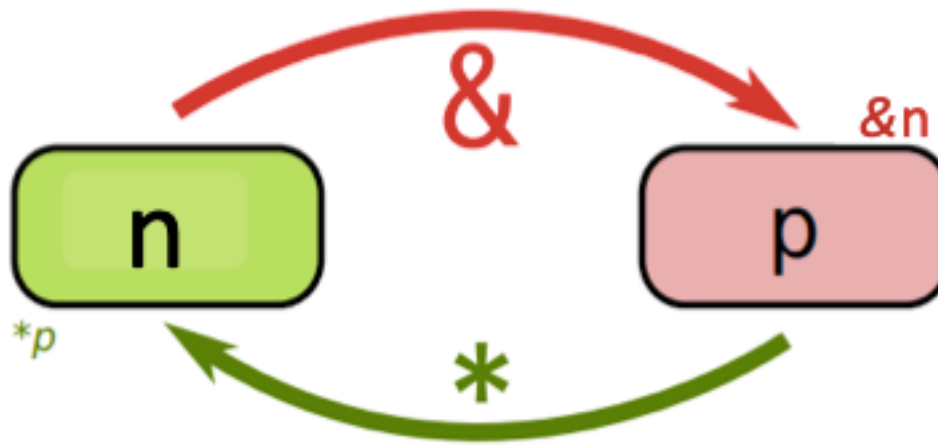
```
p = &n;
```

- Define-se ****p*** como sendo o valor contido na posição de memória apontada por ***p***. Assim, ****p*** vale 14.



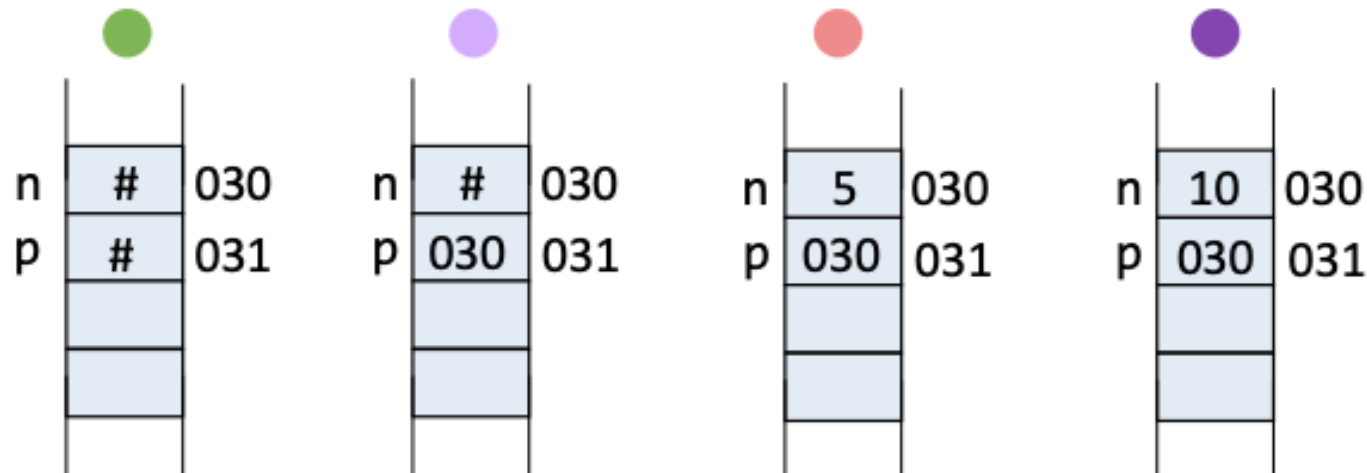
Ponteiros

- Para acessar a variável que é apontada por um ponteiro, usamos o operador `*` (o mesmo asterisco usado na declaração)
 - Se **p** é um ponteiro, podemos acessar a variável para a qual ele aponta com `*p`;
 - Esta expressão pode ser usada tanto para ler o conteúdo da variável quanto para alterá-lo.



Ponteiros

```
1  int main(){
2      ● int n, *p;
3      ○ p = &n; //p aponta para a variável n
4
5      ● *p = 5;
6      printf("n = %d", n); //imprime 5
7      ● n = 10;
8      printf("*p = %d", *p); //imprime 10
9
10     return 0;
11 }
```



Roteiro

- 1 Quiz
- 2 Ponteiros
- 3 Alocação Dinâmica de Memória
- 4 Recursividade
- 5 Tipos Abstratos de Dados
- 6 Arquivos
- 7 Referências

Alocação dinâmica de memória

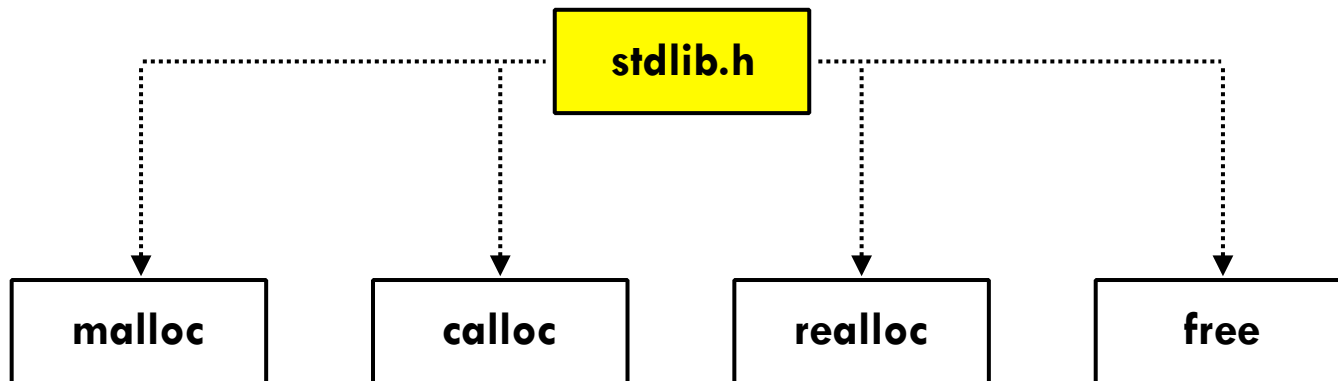
- Quando você declara um vetor em um programa em C, você deve informar quantos elementos devem ser reservados.
 - Se esse número de elementos é conhecido a priori, é trivial
 - Caso contrário, deve-se definir um tamanho máximo para acomodar os dados
- **Desperdício de memória:** caso poucos valores forem armazenados no vetor
- **Falta de Memória:** caso o vetor declarado seja insuficiente

Alocação dinâmica de memória

- Quando você declara um vetor em um programa em C, você deve informar quantos elementos devem ser reservados.
 - Se esse número de elementos é conhecido a priori, é trivial
 - Caso contrário, deve-se definir um tamanho máximo para acomodar os dados
- **Solução: Alocação Dinâmica**
- **Desperdício de Memória:** dados não utilizados, mas ainda assim armazenados no vetor
- **Falta de Memória:** caso o vetor declarado seja insuficiente
-

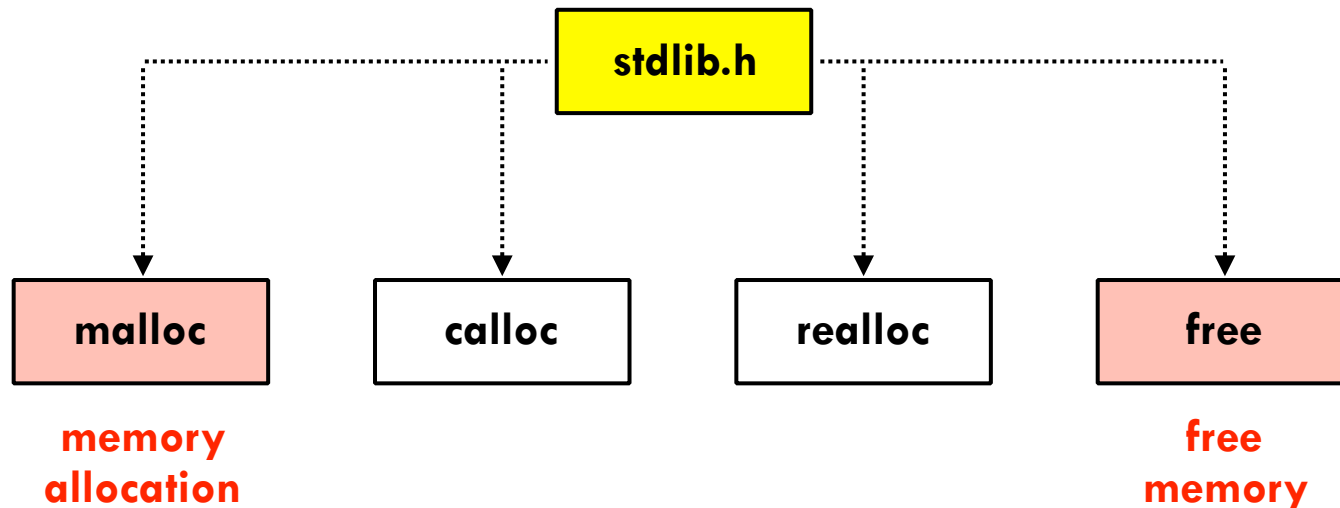
Alocação dinâmica de memória

- Alocação dinâmica é feita em “tempo de **execução**”
 - Durante a execução do programa, mais ou menos memória pode ser utilizada baseada na demanda da aplicação



Alocação dinâmica de memória

- Alocação dinâmica é feita em “tempo de **execução**”
 - Durante a execução do programa, mais ou menos memória pode ser utilizada baseada na demanda da aplicação



Exemplo

- Alocando um vetor de inteiros dinamicamente e calculando a soma de seus elementos

```
1 int main(){
2     int *vet, tam, i, soma=0;
3
4     printf("Informe o tamanho do vetor: ");
5     scanf("%d", &tam);
6
7     vet = (int *)malloc(tam*sizeof(int));
8     for(i=0;i<tam;i++){
9         printf("informe o vet[%d]: ", i);
10        scanf("%d", &vet[i]);
11    }
12    for(i=0;i<tam;i++)
13        soma += vet[i];
14
15    printf("A soma dos elementos do vetor e' %d", soma);
16    free(vet);
17
18    return 0;
19 }
```

Alocando memória para um vetor do tamanho escolhido pelo usuário

Desalocando memória usada pelo vetor

Roteiro

- 1 Quiz
- 2 Ponteiros
- 3 Alocação Dinâmica de Memória
- 4 Recursividade
- 5 Tipos Abstratos de Dados
- 6 Arquivos
- 7 Referências

Recursividade



- O que é uma função **recursiva**?

Recursividade

- função é dita **recursiva** quando dentro do seu código existe uma **chamada para si mesma**.

```
1  int fatorial(int n){
2      int fat;
3      if(n <= 1)
4          return 1;
5      else{
6          fat = n * fatorial(n-1);
7          return fat;
8      }
9  }
10
11 int main(){
12     int n=3, resultado;
13     resultado = fatorial(n);
14     printf("%d", resultado);
15
16     return 0;
17 }
```

Recursividade

- **recursão** é uma técnica que define um problema em termos de um ou mais versões menores deste mesmo problema;
- portanto, pode ser utilizada sempre que for possível expressão a solução de um problema em função do próprio problema.

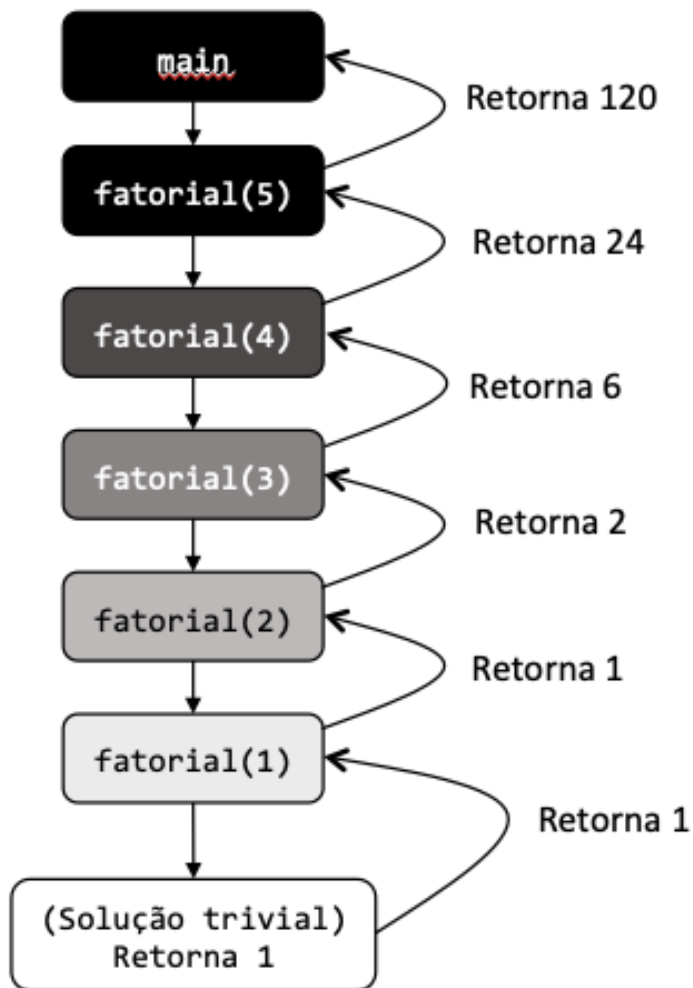


Recursividade

```
1  int fatorial(int n){  
2      if(n == 0)  
3          return 1;  
4      else if(n < 0){  
5          exit(0);  
6      }  
7      return n * fatorial(n-1);  
8  }  
9  
10 int main(){  
11     int n=3, resultado;  
12     resultado = fatorial(n);  
13     printf("%d", resultado);  
14  
15     return 0;  
16 }
```

```
fatorial(5)  
=> return 5 * fatorial(4)  
=> return 4 * fatorial(3)  
    => return 3 * fatorial(2)  
        => return 2 * fatorial(1)  
            => return 1 * fatorial(0)  
                => 0 == 0  
                    <= return 1  
                        <= return 1 * 1 → (1)  
                            <= return 2 * 1 → (2)  
                                <= return 3 * 2 → (6)  
                                    <= return 4 * 6 → (24)  
                                        <= return 5 * 24 → (120)
```

Recursividade



```
fatorial(5)
=> return 5 * fatorial(4)
=> return 4 * fatorial(3)
=> return 3 * fatorial(2)
=> return 2 * fatorial(1)
=> return 1 * fatorial(0)
=> 0 == 0
<= return 1
<= return 1 * 1 → (1)
<= return 2 * 1 → (2)
<= return 3 * 2 → (6)
<= return 4 * 6 → (24)
<= return 5 * 24 → (120)
```

Roteiro

- 1 Quiz
- 2 Ponteiros
- 3 Alocação Dinâmica de Memória
- 4 Recursividade
- 5 Tipos Abstratos de Dados
- 6 Arquivos
- 7 Referências

Struct

- Usamos um tipo de estrutura chamado de **registro** (mais conhecido por seu nome em inglês, ***struct***, uma abreviação de *structure*, ‘estrutura’);
- Esse recurso da linguagem C permite que o usuário “defina” **seus próprios tipos de dados** a partir dos tipos primitivos da linguagem (*int, float, char, etc.*);
- Struct contém um conjunto de variáveis, que têm tipos fixados e são identificadas por nomes (como as variáveis comuns);

Struct

- Uma **struct/registro** é declarada usando a palavra chave **struct** seguida de um bloco (delimitado por chaves) contendo as declarações dos membros, como se fossem declaração de variáveis comuns.

```
struct Produto{  
    char descricao[30];  
    int quantidade;  
    float preco_unitario;  
    float desconto;  
    float preco_total;  
};
```

Definição da struct

Ponto e vírgula
(definição da struct é um comando)

```
struct Produto produto_1, produto_2, produto_3;
```

Declaração de 3
variáveis do tipo
Produto

Uso de structs

- Uma variável estrutura pode ser atribuída a outra do mesmo tipo por meio de uma atribuição simples

```
struct Produto{
    char descricao[30];
    int quantidade;
    float preco_unitario;
    float desconto;
    float preco_total;
};

struct Produto feijao = {"redondo", 1, 20.0, 0, 20.0};
struct Produto feijao_carioca;

feijao_carioca = feijao;
```

Atribuição só pode ser feita com **structs** do mesmo tipo

structs

```
typedef struct{  
    int dia, mes, ano;  
}Data;
```

```
int main()  
{  
    Data atual;  
  
    return 0;  
}
```

São equivalentes



```
struct Data{  
    int dia, mes, ano;  
};
```

```
int main()  
{  
    struct Data atual;  
  
    return 0;  
}
```

structs

- Passagem por valor

```
typedef struct{
    int x, y, z;
}Ponto;

void imprime(int v){
    printf("Valor: %d", v);
}

int main()
{
    Ponto p = {1, 2, 3};
    imprime(p.x);
    ...
}
```

Tem que ser do mesmo tipo!

structs

- Passagem por referência

```
typedef struct{
    int x, y, z;
}Ponto;

void incrementa_imprime(int *v){
    *v = *v + 1;
    printf("Valor: %d", *v);
}

int main()
{
    Ponto p = {1,2,3};

    imprime(&p.y);
    ...
}
```

O operador & precede o nome da estrutura, não o nome da variável membro!

structs

- Passando struct toda como valor

```
1 typedef struct{
2     char nome[30];
3     char matricula[10];
4     float notas[4];
5 }Aluno;
6
7 void imprimeAluno(Aluno a){
8     int i;
9
10    puts(a.nome);
11    puts(a.matricula);
12
13    for(i=0;i<4;i++)
14        printf(" %f ", a.notas[i]);
15 }
```

```
16 int main(){
17     Aluno a;
18     int i;
19
20     scanf("%s", a.nome);
21     scanf("%s", a.matricula);
22
23     for(i=0;i<4;i++)
24         scanf("%f", &a.notas[i]);
25
26     imprimeAluno(a);
27
28     return 0;
29 }
```

structs

- Passando struct toda como referência

```
typedef struct{
    int x, y, z;
}Ponto;

void altera(Ponto *v){
    (*v).x = (*v).x + 1;
    (*v).y = (*v).y + 1;
    (*v).z = (*v).z + 1;
}

int main()
{
    Ponto p = {1, 2, 3};

    altera(&p);
    ...
}
```

Equivalentes



```
typedef struct{
    int x, y, z;
}Ponto;

void altera(Ponto *v){
    v->x = v->x + 1;
    v->y = v->y + 1;
    v->z = v->z + 1;
}

int main()
{
    Ponto p = {1, 2, 3};

    altera(&p);
    ...
}
```

structs

- Retornando structs

```
1 typedef struct{
2     char modelo[20], placa[8];
3     int ano;
4 }Carro;
5
6 Carro iniciaCarro(char *m, char *p, int a){
7     Carro c;
8
9     strcpy(c.modelo, m);
10    strcpy(c.placa, p);
11    c.ano = a;
12
13    return c;
14 }
```

```
int main(){
    Carro novo_carro;

    novo_carro = iniciaCarro("Ferrari", "abc1234", 2018);
    ...
}
```

Roteiro

- 1 Quiz
- 2 Ponteiros
- 3 Alocação Dinâmica de Memória
- 4 Recursividade
- 5 Tipos Abstratos de Dados
- 6 Arquivos
- 7 Referências

Arquivos

- Basicamente, a linguagem C trabalha com dois tipos de arquivos:

Arquivo texto

- Armazena caracteres que podem ser mostrados diretamente na tela ou modificados por um editor de textos;
- Os dados são gravados como caracteres de 8 bits. Ex.: Um número inteiro de 32 bits com 8 dígitos ocupará 64 bits no arquivo.

Arquivo binário

- Armazena uma sequência de bits que está sujeita as convenções do programa que o gerou. Ex.: arquivos compactados;
- Os dados são gravados em binário, ou seja, do mesmo modo que estão na memória. Ex.: um número inteiro de 32 bits com 8 dígitos ocupará 32 bits no arquivo.

A manipulação de arquivos se dá por meio de fluxos (***streams***).

Arquivos

- A biblioteca `stdio.h` dá suporte à utilização de arquivos em C.
 - Renomear e remover;
 - Garantir acesso ao arquivo;
 - Ler e escrever;
 - Alterar o posicionamento dentro do arquivo;
 - Manusear erros;
 - Para mais informações: <http://www.cplusplus.com/reference/cstdio/>
- A linguagem C não possui funções que leiam automaticamente toda a informação de um arquivo:
 - Suas funções limitam-se em **abrir/fechar** e **ler/escrever** caracteres ou bytes;
 - O programador deve instruir o programa na leitura do arquivo de uma maneira específica;

Arquivos

```
FILE *arq;
```

```
arq = fopen(nome_arquivo, modo_de_abertura);
```

└→ A função fopen retorna um ponteiro do tipo FILE

- O modo de abertura determina que tipo de **USO** será feito do arquivo
 - Abrir para leitura;
 - Abrir para escrita;
 - Abrir para leitura e escrita.

Arquivos

- Modos clássicos

Modo	Arquivo	Função
“r”	Texto	Leitura. Arquivo deve existir.
“w”	Texto	Escrita. Criar arquivo se não houver. Apaga o anterior se ele existir.
“a”	Texto	Escrita. Os dados serão adicionados no final do arquivo (<i>append</i>).
“rb”	Binário	Leitura. Arquivo deve existir.
“wb”	Binário	Escrita. Cria arquivo se não houver. Apaga o anterior se ele existir.
“ab”	Binário	Escrita. Os dados serão adicionados no fim do arquivo (<i>append</i>)
“r+”	Texto	Leitura/Escrita. O arquivo deve existir e pode ser modificado.
“w+”	Texto	Leitura/Escrita. Cria arquivo se não houver. Apaga o anterior se ele existir
“a+”	Texto	Leitura/Escrita. Os dados serão adicionados no fim do arquivo (<i>append</i>).
“r+b”	Binário	Leitura/Escrita. O arquivo deve existir e pode ser modificado.
“w+b”	Binário	Leitura/Escrita. Cria arquivo se não houver. Apaga o anterior se ele existir.
“a+b”	Binário	Leitura/Escrita. Os dados serão adicionados no fim do arquivo (<i>append</i>).

Arquivos

- Um arquivo do tipo texto pode ser aberto para escrita utilizando o seguinte conjunto de comandos:
 - A condição `arq == NULL` testa se o arquivo foi aberto com sucesso;
 - No caso de erro a função `fopen` retorna um ponteiro nulo (`NULL`).

```
int main(){  
    FILE *arq;  
  
    arq = fopen("teste.txt", "w");  
    if(arq == NULL)  
        printf("Ocorreu um erro na abertura do arquivo");  
    ...  
}
```

Arquivos

- Um arquivo pode ser fechado pela função `fclose()`;
 - Escreve no arquivo qualquer dado que ainda permanece no *buffer*;
 - Geralmente as informações só são gravadas no disco quando o *buffer* está cheio.
 - O ponteiro do arquivo é passado como parâmetro para `fclose()`;
 - **Esquecer de fechar** o arquivo pode gerar **inúmeros problemas**;

```
int main(){
    FILE *arq;

    arq = fopen("teste.txt", "w");
    if(arq == NULL)
        printf("Ocorreu um erro na abertura do arquivo");
        system("pause");
        exit(1);
    }
    ...
    fclose(arq);

    return 0;
}
```

Arquivos

- A maneira mais fácil de trabalhar com um arquivo é a leitura/escrita de **um único caractere por vez**;
- A função `fputc` (*put character*) pode ser utilizado para esse princípio;

```
1 FILE *arq;  
2 char str[] = "Texto a ser gravado no arquivo";  
3 int i;  
4 arq = fopen("Teste.txt", "w");  
5 if(arq == NULL){  
6     printf("Erro ao abrir o arquivo");  
7     system("pause");  
8     exit(1);  
9 }  
10 for(i=0;i<strlen(str);i++)  
11     fputc(str[i], arq);  
12  
13 fclose(arq);
```

```
fputc(caractere, ponteiro);
```

Equivale à:

```
putc(caractere, ponteiro);
```

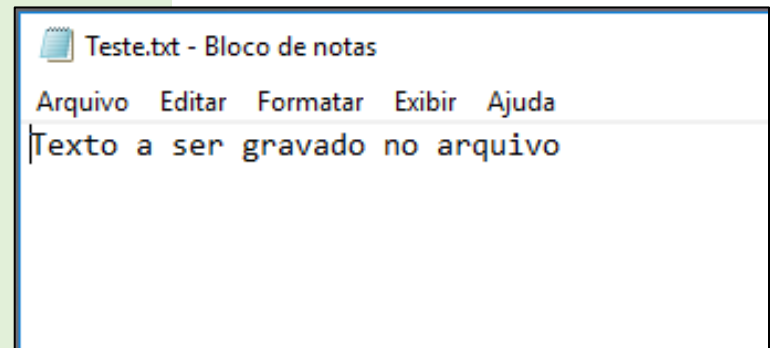
Usada também para impressão:

```
fputc('a', stdout);
```

Arquivos

- Agora usando **while**...

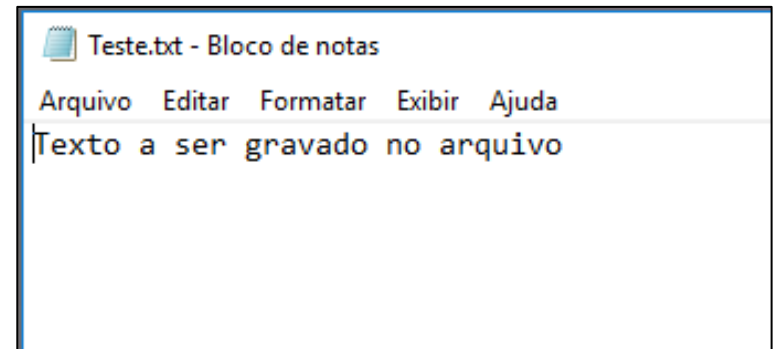
```
1 FILE *arq;
2 char str[] = "Texto a ser gravado no arquivo";
3 int i;
4 arq = fopen("Teste.txt", "w");
5 if(arq == NULL){
6     printf("Erro ao abrir o arquivo");
7     system("pause");
8     exit(1);
9 }
10 i = 0;
11 while(str[i] != '\0'){
12     fputc(str[i], arq);
13     i++;
14 }
15
16 fclose(arq);
```



Arquivos

- Também podemos ler caracteres um a um do arquivo;
- A função usada para isso é a `fgetc` (*get character*);

```
1 FILE *arq;
2 char c;
3 int i;
4 arq = fopen("Teste.txt", "r");
5 if(arq == NULL){
6     printf("Erro ao abrir o arquivo");
7     system("pause");
8     exit(1);
9 }
10
11 c = fgetc(arq);
12 while(c != EOF){
13     printf("%c", c);
14     c = fgetc(arq);
15 }
16
17 fclose(arq);
```



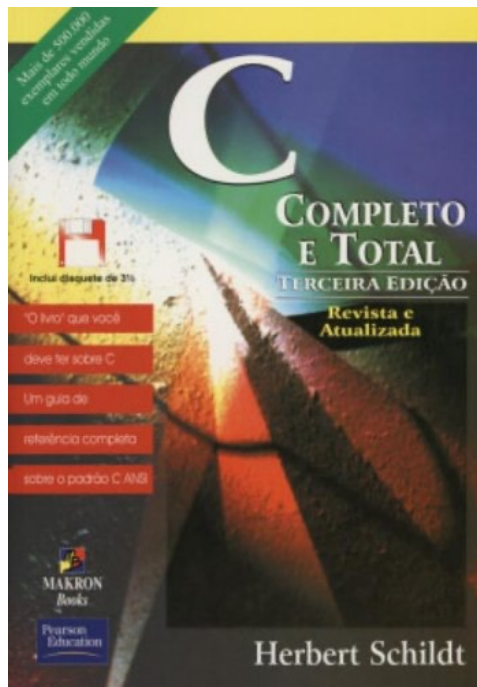
Saída:

Texto a ser gravado no arquivo

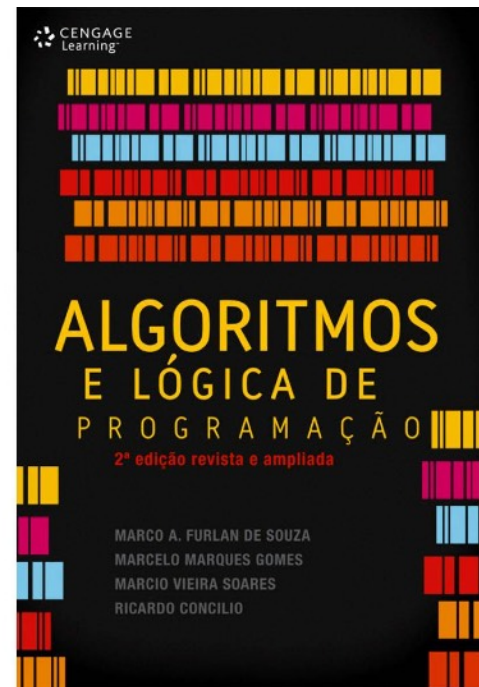
Roteiro

- 1 Quiz
- 2 Ponteiros
- 3 Alocação Dinâmica de Memória
- 4 Recursividade
- 5 Tipos Abstratos de Dados
- 6 Arquivos
- 7 Referências

Referências



[Schildt, 1997]



[de Souza et al, 2011]

Referências



1. Notas de aula profa. Silvana M. A. de Lara. Universidade de São Paulo – São Carlos. ICMC.
2. Notas de aula prof. Luiz Fernando Carvalho. Universidade Tecnológica Federal do Paraná. UTFPR, Apucarana.

□

Perguntas?

Prof. Rafael G. **Mantovani**

rafaelmantovani@utfpr.edu.br