

# Estrutura de Dados

# Linguagem C

## Laços em C:

É muito comum, em programas computacionais, termos procedimentos iterativos, ou seja, procedimentos que devem ser executados em vários passos. Vamos analisar como exemplo abaixo o cálculo do valor do fatorial de um número inteiro não negativo.

$$b! = b \times (b - 1) \times (b - 2) \dots 3 \times 2 \times 1, \text{ onde } 0! = 1$$

# Laços em C

Mas antes disso vamos ver que a linguagem C nos ajuda nesta interação por isso podemos fazer essas interações através de laços de repetição.

Laço de repetição: while

Exemplo:

```
1  int i = 1;  
2  while (i <= 10) {  
3      printf("%d", i++);  
4  }
```

Ou

```
1  int i = 10;  
2  while (i <= 10) {  
3      printf("%d", i--);  
4  }  
5  |
```

Vamos ver a solução do exemplo do fatorial com o laço While:

C:\Users\JOÃO\Desktop\Sem Título1.cpp - Dev-C++ 5.11

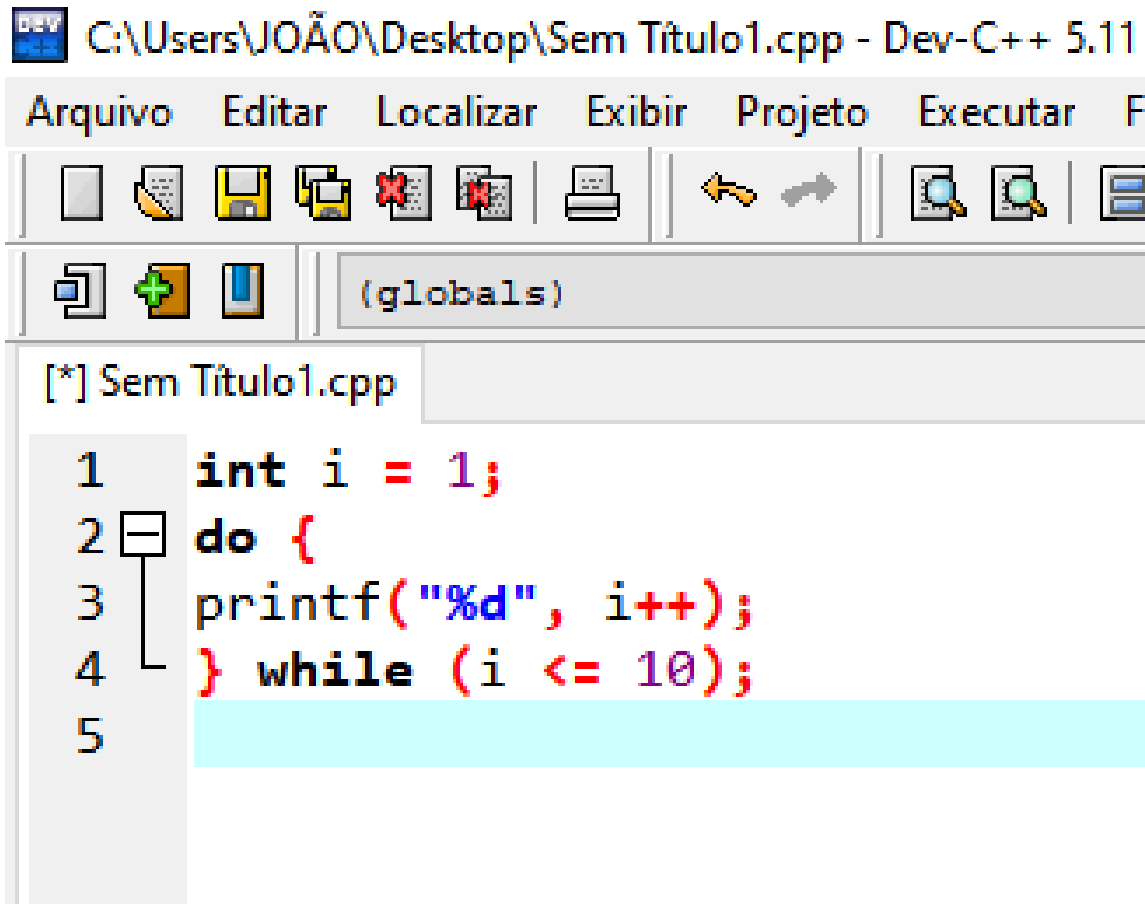
Arquivo Editar Localizar Exibir Projeto Executar Ferramentas AStyle Janela Ajuda

(globals)

[\*] Sem Título1.cpp

```
1  #include <stdio.h>
2  int main (void)
3  {
4      int i;
5      int n;
6      int fatorial = 1;
7      printf("Digite um numero inteiro nao negativo:");
8      scanf("%d", &n);
9      // calcula fatorial
10     i = 1;
11     while (i <= n)
12     {
13         fatorial *= i;
14         i++;
15     }
16     printf("O Fatorial = %d \n", fatorial);
17     return 0;
18 }
```

## Laço de repetição: Do



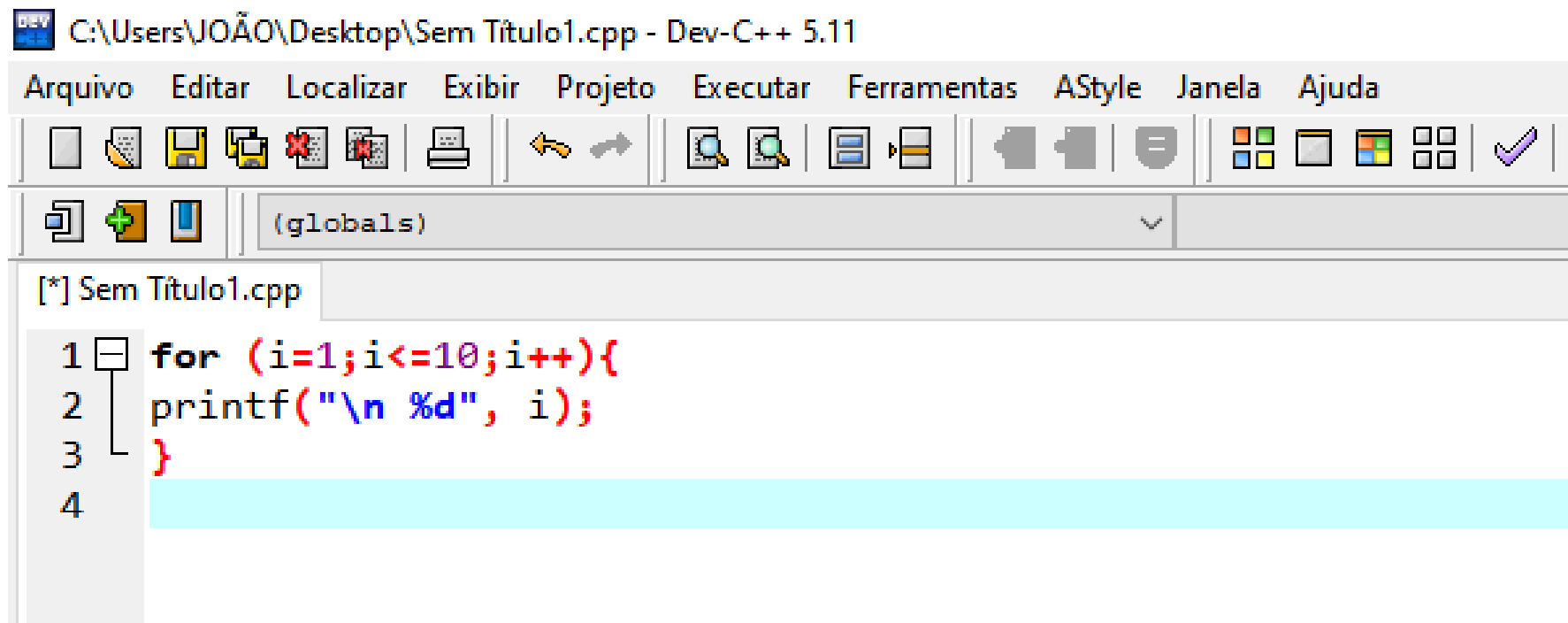
The screenshot shows the Dev-C++ 5.11 IDE interface. The title bar indicates the file path: C:\Users\JOÃO\Desktop\Sem Título1.cpp - Dev-C++ 5.11. The menu bar includes Arquivo, Editar, Localizar, Exibir, Projeto, Executar, and F. The toolbar contains icons for file operations (new, open, save, close, print) and editing (undo, redo, find, replace). The project explorer on the left shows a folder named (globals). The main editor window displays the code for Sem Título1.cpp:

```
1  int i = 1;
2  do {
3      printf("%d", i++);
4  } while (i <= 10);
5
```

The code is a C++ program that demonstrates a do-while loop. It initializes a variable `i` to 1 and then enters a loop that prints the value of `i` and increments it until it reaches 10. The loop body is highlighted in light blue.

# Laço de repetição: For

## Exemplo:



The screenshot shows the Dev-C++ 5.11 IDE interface. The title bar reads "C:\Users\JOÃO\Desktop\Sem Título1.cpp - Dev-C++ 5.11". The menu bar includes "Arquivo", "Editar", "Localizar", "Exibir", "Projeto", "Executar", "Ferramentas", "AStyle", "Janela", and "Ajuda". The toolbar contains various icons for file operations, editing, and execution. The "globals" pane is empty. The main editor window, titled "[\*] Sem Título1.cpp", contains the following C++ code:

```
1 for (i=1;i<=10;i++){  
2     printf("\n %d", i);  
3 }  
4
```

The code is a simple for loop that prints the numbers 1 through 10, each on a new line. The line numbers 1 through 4 are visible in the left margin. The code is color-coded: "for" is red, "i=1" is blue, "i<=10" is purple, "i++" is red, "printf" is blue, and the format string and variable "i" are blue.

## Como ficaria o algoritmo do Fatorial já mostrado utilizando o For?

DEV C++ C:\Users\JOÃO\Desktop\Sem Título1.cpp - Dev-C++ 5.11

Arquivo Editar Localizar Exibir Projeto Executar Ferramentas AStyle Janela Ajuda

(globals)

[\*] Sem Título1.cpp

```
1  #include <stdio.h>
2  int main (void)
3  {
4      int i;
5      int n;
6      int fatorial = 1;
7      printf("Digite um número inteiro nao negativo:");
8      scanf("%d", &n);
9      // calculo do fatorial
10     for (i = 1; i <= n; i++)
11     {
12         fatorial *= i;
13     }
14     printf("O Fatorial = %d \n", fatorial);
15     return 0;
16 }
```

**Como exemplo podemos incrementar o algoritmo acima já analisado criando uma função em C e mostrando seu fatorial:**

**Vamos ver o que acontece na memória:**



```
C:\Users\JOÃO\Desktop\Sem Título1.cpp - Dev-C++ 5.11
Arquivo  Editar  Localizar  Exibir  Projeto  Executar  Ferramentas  AStyle  Janela  Ajuda
[Icons]
(globalis)
Sem Título1.cpp
1  #include <stdio.h>
2  int fat (int n);
3  int main (void)
4  {
5  int n = 0;
6  int r;
7  printf("Digite um numero inteiro nao negativo");
8  scanf("%d",&n);
9  r = fat ( n );
10 printf("Fatorial de %d = %d \n", n, r);
11 return 0;
12 }
13 //criando uma função de para calcular o fatorial
14 int fat (int n)
15 {
16 int fatorial = 1.0;
17 while (n != 0)
18 {
19 fatorial *= n;
20 n--;
21 }
22 return fatorial;
23 }
```

# Vamos falar de vetores em C ?

O vetor é uma estrutura de dados linear que necessita de somente um índice para que seus elementos sejam endereçados. É utilizado para armazenar uma lista de valores do mesmo tipo, ou seja, o tipo vetor permite armazenar mais de um valor em uma mesma variável.

## Exemplo de vetores:

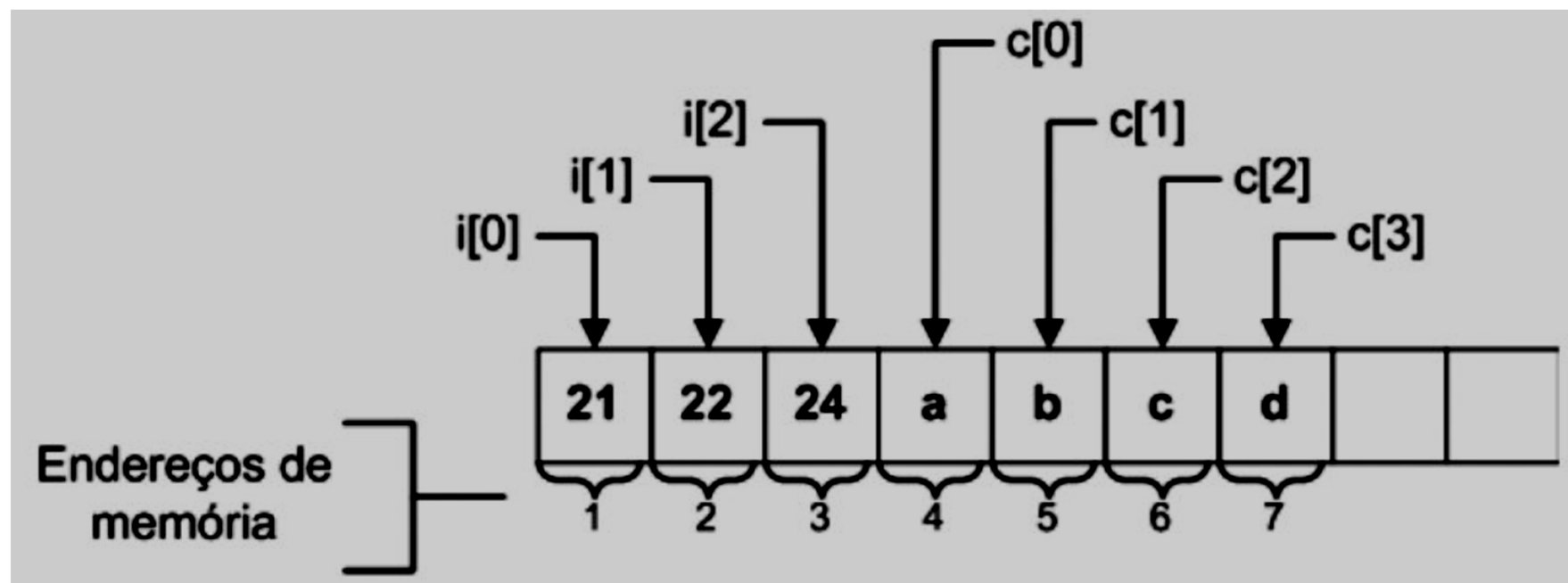
```
int vetorX[10]; //Vetor de inteiros
```

obs: vetorX[0] é o primeiro elemento e  
vetorX[9] o último.

exemplo:

```
int x[5];  
x[0]=12;  
x[1]=13;  
x[2]=34;  
x[3]=72;  
x[4]=01;
```

Um exemplo o que esta acontecendo na memoria do computador:



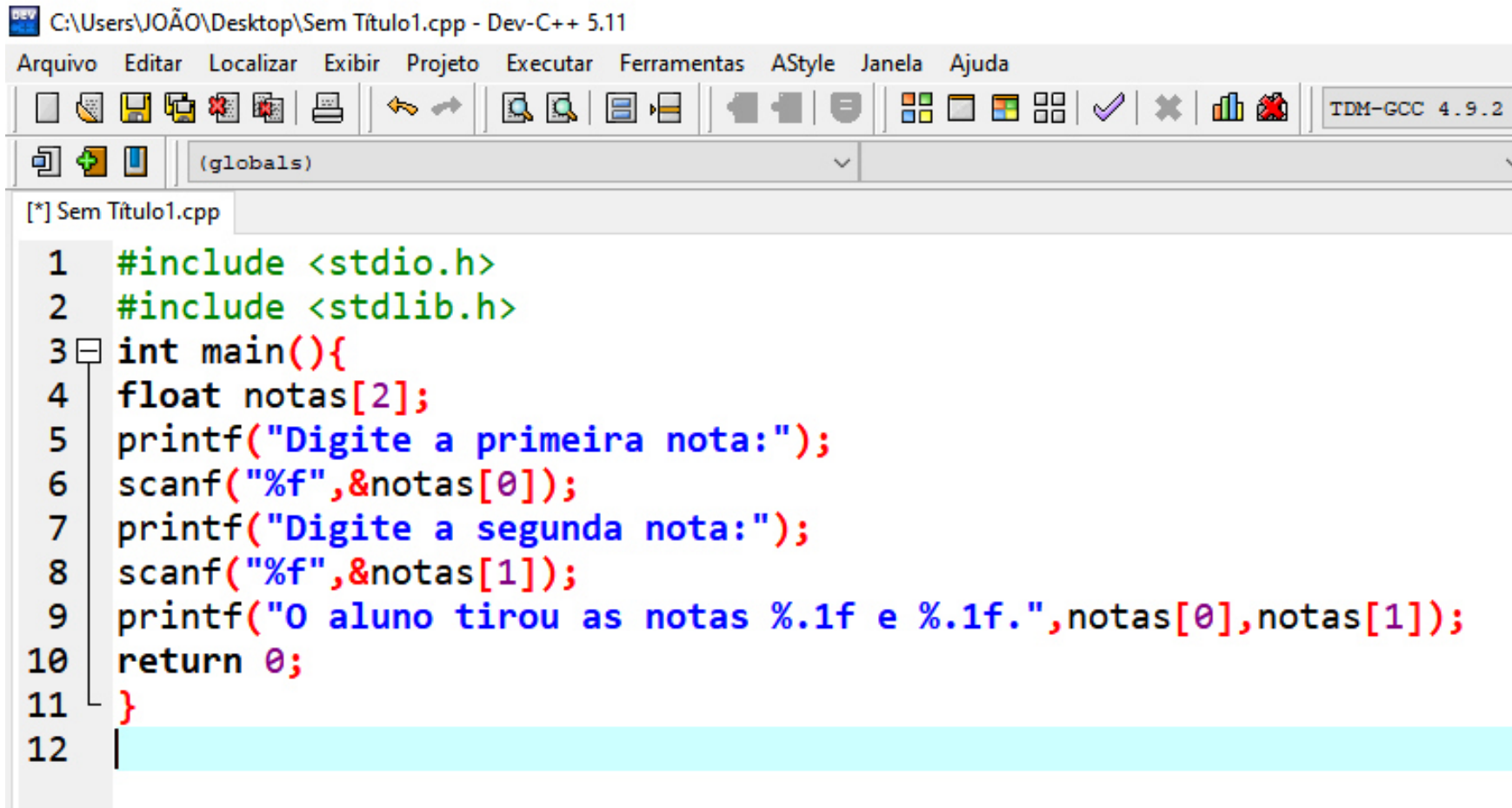
# Algumas observações:

## características de um vetor:

- Alocação estática (deve-se conhecer as dimensões da estrutura no momento da declaração em C)
- Estrutura homogênea
- Alocação sequencial (bytes contíguos)
- Inserção/Exclusão
  - Realocação dos elementos;
  - Posição de memória não liberada;

Vamos ver como funciona um vetor simples:

## Exemplo pratico:



The screenshot shows the Dev-C++ 5.11 IDE interface. The title bar indicates the file path: C:\Users\JOÃO\Desktop\Sem Título1.cpp - Dev-C++ 5.11. The menu bar includes: Arquivo, Editar, Localizar, Exibir, Projeto, Executar, Ferramentas, AStyle, Janela, Ajuda. The toolbar contains various icons for file operations, editing, and execution. The status bar at the bottom shows the compiler: TDM-GCC 4.9.2. The main editor window displays the following C++ code:

```
[*] Sem Título1.cpp
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(){
4      float notas[2];
5      printf("Digite a primeira nota:");
6      scanf("%f",&notas[0]);
7      printf("Digite a segunda nota:");
8      scanf("%f",&notas[1]);
9      printf("O aluno tirou as notas %.1f e %.1f.",notas[0],notas[1]);
10     return 0;
11 }
12
```

## Exercício 01:

Faça um algoritmo que receba valores inteiros de em um vetor de tamanho 3x2 e preencha um vetor inteiro de tamanho 6. Imprima o vetor preenchido.



DEV C++ C:\Users\JOÃO\Desktop\Sem Título1.cpp - Dev-C++ 5.11

Arquivo Editar Localizar Exibir Projeto Executar Ferramentas

(globals)

[\*] Sem Título1.cpp

```
1  #include<stdio.h>
2  #include<math.h>
3  int main(){
4  int vetor[6], i;
5  for (i = 0; i < 6; i++){
6  scanf("%d",&vetor[i]);
7  printf("%d \n",vetor[i]);
8  }
9  return 0;
10 }
11
```

## Exercício 02:

Faça um cadastro de várias matrículas de alunos da faculdade e armazene-os em um vetor até o mesmo ser preenchido por 18 matrículas. Esses números são distintos, ou seja, o vetor não armazenará valores repetidos.

C:\Users\JOÃO\Desktop\Sem Título1.cpp - Dev-C++ 5.11



[\*] Sem Título1.cpp

```
1  #include<math.h>
2  #include<stdio.h>
3  #include<string.h>
4  int main(){
5      int vetor[18], numero, cont, posicao = 0 ;
6      while (posicao < 18){
7          scanf("%d",&numero);
8          if (posicao == 0){
9              vetor[posicao] = numero;
10             printf("%d\n",vetor[posicao]);
11             posicao++;
12         }
13         else{
14             for(cont = 0; (cont < posicao)&&(vetor[cont] != numero); cont++);
15             if (cont >= posicao){
16                 vetor[posicao] = numero;
17                 printf("%d\n",vetor[posicao]);
18                 posicao++;
19             }
20         }
21     }
22     return 0;
23 }
```

## **Exercício 03:**

Criar 4 vetores, o primeiro com a nota da primeira prova, o segundo com a nota da segunda prova e o terceiro, no quarto vetor com a média das 3 primeiras notas, e imprima o resultado “APROVADO” para aqueles que obtiverem uma média igual ou acima de 7, e “REPROVADO” para quem obtiverem uma média abaixo de 7.

OBS.: Saia do laço quando a primeira nota for igual a -7.

C:\Users\JOÃO\Desktop\Sem Título1.cpp - Dev-C++ 5.11

```
Arquivo  Editar  Localizar  Exibir  Projeto  Executar  Ferramentas  AStyle  Janela  Ajuda

[Sem Título1.cpp] [Sem Título2]

1  #include<stdio.h>
2  int main(void){
3      int contador = 0;
4      float vetor[4], nota = 0 ;
5      scanf("%f",&nota);
6      while ( nota != -7){
7          vetor[contador] = nota;
8          contador++;
9          if (contador == 3){
10             vetor[contador] = (vetor[0] + vetor[1] + vetor[2]) / 3;
11             printf("%.2f\n",vetor[contador]);
12             if(vetor[contador] >= 6){
13                 printf("APROVADO\n");
14             }
15             else{
16                 printf("REPROVADO\n");
17             }
18             contador = 0;
19         }
20         scanf("%f",&nota);
21     }
22 }
```

# **Vamos falar sobre matrizes em C ?**

Também chamado de vetores multidimensionais, esses tipos de vetores são divididos em linhas e colunas de dados. Como por exemplo nesta declaração:

```
float mat[2][2]; //Tabela com 2 linhas e 2 colunas
```

**Matriz M**

**i/j**

	0	1
0		
1		

Também chamado de vetores multidimensionais, esses tipos de vetores são divididos em linhas e colunas de dados.

```
int x[2][5]; //Tabela com 2 linhas e 5 colunas
```

Uma melhor visualização seria :

<Variável(x) >	Coluna 0	Coluna 1	Coluna 2	Coluna 3	Coluna 4
<b>Linha 0</b>	2	54	60	23	44
<b>Linha 1</b>	4	63	7	11	55



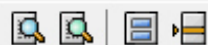
Um exemplo de utilização dessa matriz seria para um programa para uma loja de departamentos com 2 filiais, cada uma vendendo 3 itens, poderia incluir um vetor declarado como abaixo:

```
int vendas [2][3];
```

Cada elemento vendas [ i ] [ j ] representa a quantidade do item j vendida na filial i, declarando como demonstrado abaixo:

C:\Users\JOÃO\Desktop\Sem Título1.cpp - Dev-C++ 5.11

Arquivo Editar Localizar Exibir Projeto Executar Ferramentas AStyle Janela Ajuda



TDM-GCC 4.9.2 64-bit Re

(globals)

Sem Título1.cpp [\*] Sem Título2

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(){
4  int vendas [2][3];
5  int i,j;
6  for(i=0;i<=1;i++){
7  printf("Filial 0-: %d \n",i+1);
8  for(j=0;j<=2;j++){
9  printf("Digite o item 0-: %d:",j+1);
10 scanf("%d",vendas[i,j]);
11 }
12 system("cls");
13 }
14 printf("Itens cadastrados com sucesso!!!");
15 return 0;
16 }
```

# Exercício

Construa um sistema de controle de estoque de uma pequena padaria. Seu programa deve oferecer um menu para (a) cadastrar um novo produto; (b) aumentar o estoque de um produto cadastrado (quando há compras); (c) diminuir o estoque de um produto cadastrado (quando há vendas); e (d) imprimir os produtos cadastrados e suas características. De cada produto deseja-se cadastrar: nome (tamanho máximo 50 caracteres) e preço (número real) além, da quantidade em estoque. Assuma que a padaria venderá no máximo, 100 produtos diferentes.

**OBS:** Para facilitar a localização de um produto (para as funções (b) e (c)), use a posição dos mesmos no vetor que representa o estoque como código do produto, informando-a ao fim do cadastro. Então, peça ao usuário para digitar o código do produto que deseja aumentar/diminuir o estoque.

# Tipos de dados abstratos

As propriedades lógicas de um tipo de dado é o tipo de dado abstrato, ou TDA.

Fundamentalmente, um tipo de dado significa um conjunto de valores e uma sequência de operações sobre estes valores.

Este conjunto e estas operações formam uma construção matemática que pode ser implementada usando determinada estrutura de dados do hardware ou do software.

A expressão "tipo de dado abstrato" refere-se ao conceito matemático básico que define o tipo de dado.

Como podemos definir novos tipos de dados mas antes disso temos que pensar em  
A estrutura da informação propriamente dita.  
Na linguagem de programação C representamos estrutura da seguinte forma:

```
struct coordenada {  
int x;  
int y;  
}
```

## Tipo de dados abstrato

A palavra reservada para isso é a **struct**.

```
struct coordenada {  
    int x;  
    int y;  
}  
  
struct coordenada coord;  
coord.x = 3;  
coord.y = 5;
```

Com base no exemplo acima represente novas estrutura de informações, e imprima na tela do console esse dados, lembrando que tem que ser 15 tipos novos.



Como visto nos slide anterior vamos agora criar novas estruturas abstratas de dados, para isto utilizamos a palavra reservada *TYPEDEF*:

```
typedef struct coordenada {  
    int x;  
    int y;  
} Coordenada;  
Coordenada c;  
c.x = 10;
```

# Ponteiros

Para começar temos que analisar três propriedades que um programa deve manter quando armazena dados:

- onde a informação é armazenada;
- que valor é mantido lá;
- que tipo de informação é armazenada.

A definição de uma variável simples obedece a estes três pontos. A declaração prove o tipo e um nome simbólico para o valor. Também faz com que o programa aloque memória para o valor e mantenha o local internamente.

## Ponteiros operador de endereço: &

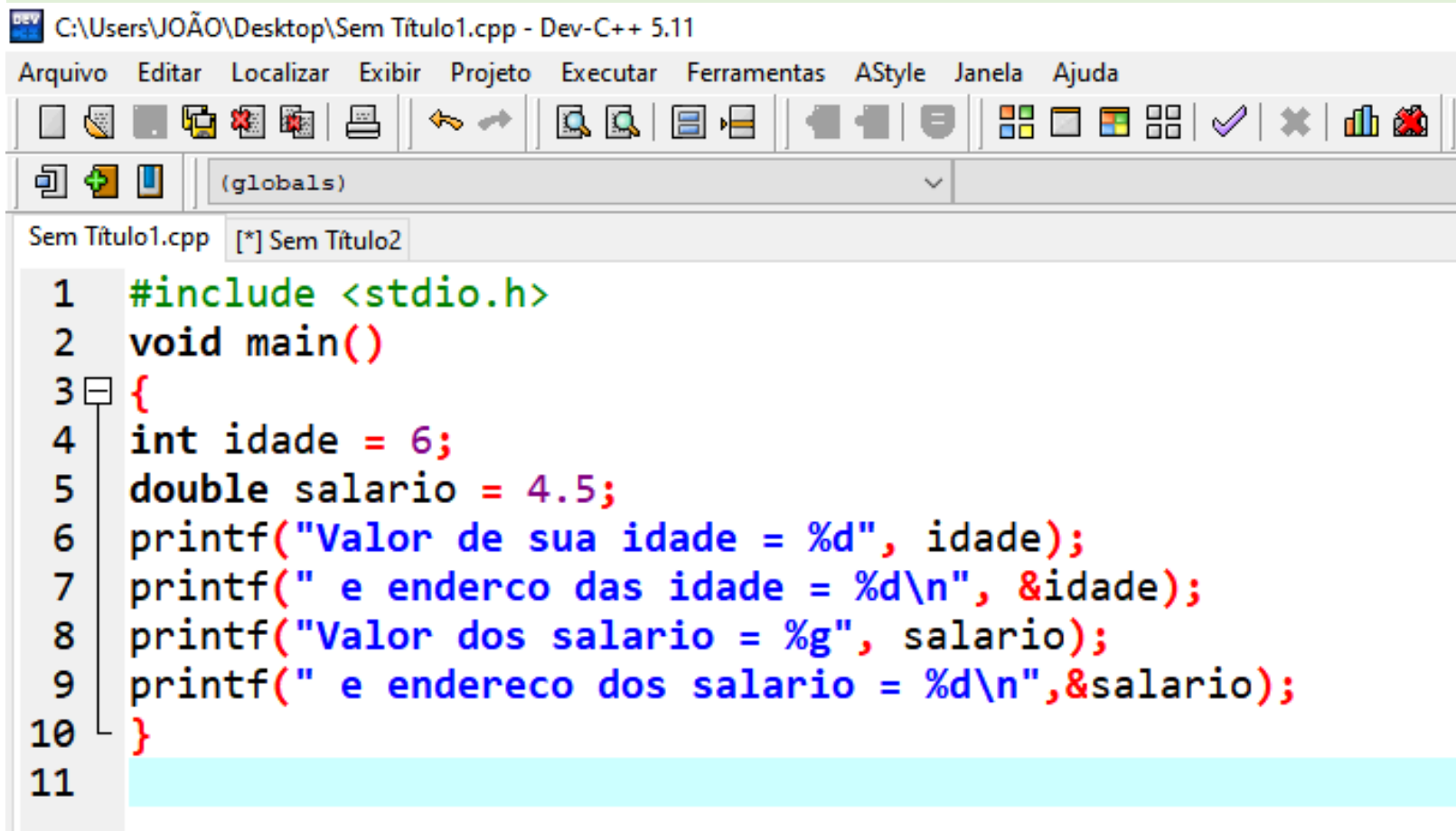
Segunda estratégia baseada em ponteiros, que são variáveis que armazenam endereços ao invés dos próprios valores.

Mas antes de discutir ponteiros, vejamos como achar endereços explicitamente para variáveis comuns.

Aplique o operador de endereço, &, a uma variável para pegar sua posição; por exemplo, se *notas* é uma variável, *&notas* é seu endereço.

## Ponteiro operador de endereço: &

Exemplo:



The screenshot shows the Dev-C++ 5.11 IDE interface. The title bar indicates the file path: C:\Users\JOÃO\Desktop\Sem Título1.cpp - Dev-C++ 5.11. The menu bar includes: Arquivo, Editar, Localizar, Exibir, Projeto, Executar, Ferramentas, AStyle, Janela, Ajuda. The toolbar contains various icons for file operations, editing, and execution. The variable declaration pane on the left shows '(globals)'. The main editor window displays the following C++ code:

```
1  #include <stdio.h>
2  void main()
3  {
4  int idade = 6;
5  double salario = 4.5;
6  printf("Valor de sua idade = %d", idade);
7  printf(" e enderco das idade = %d\n", &idade);
8  printf("Valor dos salario = %g", salario);
9  printf(" e endereco dos salario = %d\n",&salario);
10 }
11
```

# Alocação de Memória

A linguagem C permite duas formas de reservar memória para qualquer objeto (por exemplo, inteiros, estruturas, arrays, etc.):

- ***Alocação estática de memória.*** Até agora, a reserva ou alocação de memória era feita em tempo de compilação através de declaração de variáveis.
- ***Alocação dinâmica de memória.*** A alocação dinâmica de memória é utilizada para reservar ou libertar memória durante a execução do programa.

# Alocação estática

A alocação estática ocorre com variáveis globais (alocadas fora de funções) ou quando variáveis locais (internas a uma função) são alocadas usando o modificador "static". Uma variável alocada estaticamente mantém seu valor durante toda a vida do programa, exceto quando explicitamente modificada.

C:\Users\JOÃO\Desktop\exemplo01.cpp - Dev-C++ 5.11

Arquivo Editar Localizar Exibir Projeto Executar Ferramentas AStyle Janela Ajuda

TDM-GCC 4.9.2 64-bit Release

(globals)

exemplo01.cpp

```
1  #include <stdio.h>
2
3  int a = 0 ; // variável global, aloc. estática
4
5  void incrementa(void)
6  {
7      int b = 0 ; // variável local, aloc. automática
8      static int c = 0 ; // variável local, aloc. estática
9
10     printf ("a: %d, b: %d, c: %d\n", a, b, c) ;
11     a++ ;
12     b++ ;
13     c++ ;
14 }
15
16 int main(void)
17 {
18     int i ;
19
20     for (i = 0; i < 5; i++)
21         incrementa() ;
22
23     return 0 ;
24 }
```



C:\Users\JOJO\Desktop\exemplo01.exe

```
a: 0, b: 0, c: 0  
a: 1, b: 0, c: 1  
a: 2, b: 0, c: 2  
a: 3, b: 0, c: 3  
a: 4, b: 0, c: 4
```

```
-----  
Process exited after 0.1683 seconds with return value 0  
Pressione qualquer tecla para continuar. . . █
```

As variáveis com alocação estática ("a" e "c") são alocadas e inicializadas uma única vez, portanto seus valores se preservam entre chamadas consecutivas da função "incrementa". Por outro lado, a variável com alocação automática "b" é alocada e descartada a cada execução da função, portanto seu valor não é preservado.

# Alocação Dinâmica em C

# Alocação Estática

Na alocação estática de memória, os tipos de dados tem tamanho predefinido. Neste caso, o compilador vai alocar de forma automática o espaço de memória necessário. Sendo assim, dizemos que a alocação estática é feita em tempo de compilação. Este tipo de alocação tende a desperdiçar recursos, já que nem sempre é possível determinar previamente qual é o espaço necessário para armazenar as informações. Quando não se conhece o espaço total necessário, a tendência é o programador exagerar pois é melhor superdimensionar do que faltar espaço.

# Alocação Dinâmica

Na alocação dinâmica podemos alocar espaços durante a execução de um programa, ou seja, a alocação dinâmica é feita em tempo de execução. Isto é bem interessante do ponto de vista do programador, pois permite que o espaço em memória seja alocado apenas quando necessário. Além disso, a alocação dinâmica permite aumentar ou até diminuir a quantidade de memória alocada.

# Sizeof

A função sizeof determina o número de bytes para um determinado tipo de dados.

É interessante notar que o número de bytes reservados pode variar de acordo com o compilador utilizado.

Exemplo:

```
x = sizeof(int);
```

# Malloc

A função malloc aloca um espaço de memória e retorna um ponteiro do tipo void para o início do espaço de memória alocado.

# **free**

A função free libera o espaço de memória alocado.

## **Exemplo: Vetor Dinâmico**

Quando um programador define tipo e o número de elementos de um vetor ele está utilizando alocação estática.

Uma alternativa interessante é declarar um vetor como ponteiro, a fim de utilizar alocação dinâmica. Para tanto devemos usar a função malloc. Porém, esta função necessita saber a quantidade de bytes que devem ser reservados. Para fazer esse cálculo usamos o comando sizeof.

**Vejamos como implementar esses detalhes no exemplo prático abaixo:**



DEV C++ C:\Users\JOÃO\Desktop\exemplo01.cpp - Dev-C++ 5.11

Arquivo Editar Localizar Exibir Projeto Executar Ferramentas AStyle Janela Ajuda



(globals)

exemplo01.cpp

```
1  #include <stdio.h>
2  #include <stdlib.h> //necessário para usar as funções malloc() e free()
3  #include <conio.h>
4  int main(void)
5  {
6      float *v; //definindo o ponteiro v
7      int i, num_componentes;
8
9      printf("Informe o numero de componentes do vetor\n");
10     scanf("%d", &num_componentes);
11
```

C:\Users\JOÃO\Desktop\exemplo01.cpp - Dev-C++ 5.11

Arquivo Editar Localizar Exibir Projeto Executar Ferramentas AStyle Janela Ajuda

TDM-GCC 4.9.2 64-bit Release

(globals)

exemplo01.cpp

```
11
12  /* ----- Alocando dinamicamente o espaço necessário -----
13
14  1 - Calcular o número de bytes necessários
15  primeiramente multiplicamos o número de componentes do vetor pela
16  quantidade de bytes que é dada pelo comando sizeof,
17  portanto temos:
18  num_componentes * sizeof(float)
19
20  2 - Reservar a quantidade de memória
21  usamos malloc para reservar essa quantidade de memória,
22  então temos:
23  malloc(num_componentes * sizeof(float))
24
25  3 - Converter o ponteiro para o tipo de dados desejado
26  como a função malloc retorna um ponteiro do tipo void,
27  precisamos converter esse ponteiro para o tipo da nossa variável, no caso float,
28  por isso usamos o comando de conversão explícita:
29  (float *)
30
31  4 - juntando tudo e atribuindo em v temos o comando abaixo: */
32
```

C:\Users\JOÃO\Desktop\exemplo01.cpp - Dev-C++ 5.11

Arquivo Editar Localizar Exibir Projeto Executar Ferramentas AStyle Janela Ajuda

TDM-GCC 4.9.2 64-bit Release

(globals)

[\*] exemplo01.cpp

```
32
33     v = (float *) malloc(num_componentes * sizeof(float));
34
35     //Armazenando os dados em um vetor
36     for (i = 0; i < num_componentes; i++)
37     {
38         printf("\nDigite o valor para a posicao %d do vetor: ", i+1);
39         scanf("%f",&v[i]);
40     }
41
42     // ----- Percorrendo o vetor e imprimindo os valores -----
43     printf("\n***** Valores do vetor dinamico *****\n\n");
44
45     for (i = 0; i < num_componentes; i++)
46     {
47         printf("%.2f\n",v[i]);
48     }
49
50     //liberando o espaço de memória alocado
51     free(v);
52
53     getch();
54     return 0;
55 }
```

C:\Users\jojo\Desktop\exemplo01.exe

Informe o numero de componentes do vetor

7

Digite o valor para a posicao 1 do vetor: 5

Digite o valor para a posicao 2 do vetor: 4

Digite o valor para a posicao 3 do vetor: 9

Digite o valor para a posicao 4 do vetor: 10

Digite o valor para a posicao 5 do vetor: 6

Digite o valor para a posicao 6 do vetor: 1

Digite o valor para a posicao 7 do vetor: 0

\*\*\*\*\* Valores do vetor dinamico \*\*\*\*\*

5.00

4.00

9.00

10.00

6.00

1.00

0.00

\_

1- Na alocação estática, o espaço de memória é definido durante o processo de compilação, já na alocação dinâmica o espaço de memória é reservado durante a execução do programa.

2- Na alocação estática não é possível alterar o tamanho do espaço de memória que foi definido durante a compilação, já na alocação dinâmica este espaço pode ser alterado dinamicamente durante a execução.

3- alocação estática tem a vantagem de manter os dados organizados na memória, dispostos um ao lado do outro de forma linear e sequencial. Isto facilita a sua localização e manipulação, em contrapartida, precisamos estabelecer previamente a quantidade máxima necessária de memória para armazenar uma determinada estrutura de dados.

Exemplo:

```
int vetor[5] = {13, 30, 35, 55, 70};
```



A quantidade de memória que se deve alocar estaticamente é definida durante a compilação, portanto corre-se o risco de sub ou superestimar a quantidade de memória alocada.

Isso não acontece na alocação dinâmica, pois como a alocação é feita durante a execução, sabe-se exatamente a quantidade necessária. Isso permite otimizar o uso da memória.

A alocação dinâmica é feita por meio de funções de alocação e liberação de memória e é de responsabilidade do programador usar essas funções de forma coerente, pois o seu uso incorreto pode causar efeitos colaterais indesejados no programa como vazamento de memória.



# Exercícios

- 1) Escreva um programa em linguagem C que solicita ao usuário a quantidade de alunos de uma turma e aloca um vetor de notas (números reais). Depois de ler as notas, imprime a média aritmética. Obs: não deve ocorrer desperdício de memória; e após ser utilizada a memória deve ser devolvida.
- 2) Desenvolva um programa que calcule a soma de duas matrizes  $M \times N$  de números reais (double). A implementação deste programa deve considerar as dimensões fornecida pelo usuário (Dica: represente a matriz através de variáveis do tipo double \*\*, usando alocação dinâmica de memória).

# Lista Linear

**Um exemplo disto seria um consultório médico: as pessoas na sala de espera estão sentadas em qualquer lugar, porém sabe-se quem é o próximo a ser atendido, e o seguinte, e assim por diante. Assim, é importante ressaltar que uma lista linear permite representar um conjunto de dados afins (de um mesmo tipo) de forma a preservar a relação de ordem entre seus elementos. Cada elemento da lista é chamado de nó.**

## **Definição:**

Conjunto de  $N$  nós, onde  $N \geq 0$ ,  $X_1, X_2, \dots, X_n$ , organizados de forma a refletir a posição relativa dos mesmos. Se  $N \geq 0$ , então  $X_1$  é o primeiro nó.

Para  $1 < k < n$ , o nó  $X_k$  é precedido pelo nó  $X_{k-1}$  e seguido pelo nó  $X_{k+1}$  e  $X_n$  é o último nó. Quando  $N = 0$ , diz-se que a lista está vazia.

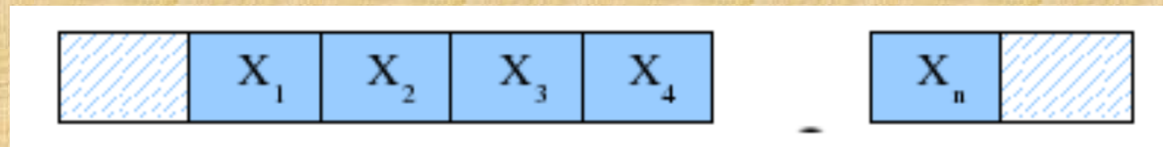
## **Exemplos de listas lineares:**

- 1- Pessoas na fila de um banco;**
- 2- Letras em uma palavra;**
- 3- Relação de notas dos alunos de uma turma;**
- 4- Itens em estoque em uma empresa;**
- 5- Dias da semana;**
- 6 -Vagões de um trem;**
- 7- Pilha de pratos;**
- 8- Cartas de baralho.**

# Alocação de uma lista

Quanto a forma de alocar memória para armazenamento de seus elementos, uma lista pode ser:

## Sequencial ou Contígua



Numa lista linear contígua, os nós além de estarem em uma sequência lógica, estão também fisicamente em sequência. A maneira mais simples de acomodar uma lista linear em um computador é através da utilização de um vetor.

## **Lista Linear Encadeada**

Os elementos não estão necessariamente armazenados sequencialmente na memória, porém a ordem lógica entre os elementos que compõem a lista deve ser mantida.

# Operações com Listas

As operações comumente realizadas com listas são:

- 1- Criação de uma lista
- 2- Remoção de uma lista
- 3- Inserção de um elemento da lista
- 4- Remoção de um elemento da lista
- 5- Acesso de um elemento da lista



As operações comumente realizadas com listas são:

- 6- Alteração de um elemento da lista
- 7- Combinação de duas ou mais listas
- 8- Classificação da lista
- 9- Cópia da lista
- 10- Localizar nodo através de info.

Listas encadeadas são representadas em C utilizando-se estruturas (struct).

A estrutura de cada célula de uma lista ligada pode ser definida da seguinte maneira:

```
struct Node{  
    int num;  
    struct Node *prox;  
};
```

Uma outra maneira de representar, utilizando typedef, seria:

```
typedef struct Node node;  
int tam;
```

Uma célula  $c$  e um ponteiro  $p$  para uma célula podem ser declarados assim:

```
node c;  
node *p;
```

Se  $c$  é uma node, então  $c.num$  é o conteúdo da célula e  $c.prox$  é o endereço da próxima célula.

Se  $p$  é o endereço de uma célula, então  $p \rightarrow num$  é o conteúdo da célula e  $p \rightarrow prox$  é o endereço da próxima célula.

Uma célula  $c$  e um ponteiro  $p$  para uma célula podem ser declarados assim:

```
node c;  
node *p;
```

Se  $p$  é o endereço da última célula da lista, então  $p \rightarrow \text{prox}$  vale NULL.

O endereço de uma lista encadeada é o endereço de sua primeira node.

Se  $p$  é o endereço de uma lista, pode-se dizer simplesmente " $p$  é uma lista".

Todas as operações serão apresentadas considerando-se que a lista possui um nó inicial, cujo valor não se tem interesse, denominado "cabeça" da lista. Esse nó tem apenas a função de apontar para o primeiro elemento inserido na lista.

```
int main(void)
{
node *LISTA = (node *)
malloc(sizeof(node));
if(!LISTA){
printf("Sem memoria disponivel!\n");
exit(1);
}else{
inicia(LISTA);
free(LISTA);
return 0;
}
}
```

```
void inicia(node *LISTA)
{
    LISTA->prox = NULL;
    tam=0;
}
```



Implemente um programa em C que utiliza a estrutura apresentada para implementar uma lista. O programa deve mostrar ao usuário duas opções.

Se o usuário escolher 1, a lista deve ser impressa; se escolher 2, ele deve entrar com o valor do conteúdo do novo elemento da lista.

