

MINISTÉRIO DA EDUCAÇÃO UNIVERSIDADE FEDERAL DO PIAUÍ – UFPI
COLÉGIO TÉCNICO DE BOM JESUS CURSO TÉCNICO EM INFORMÁTICA
DISCIPLINA: PROGRAMAÇÃO ESTRUTURADA – 2023.1

Alunos: Analia Beatriz, Briza Augusto e João Victor Verissimo.

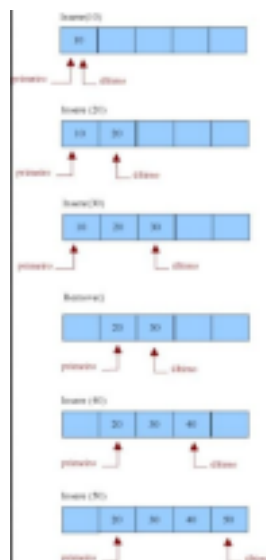
Pesquisa sobre MATRIZ, FILA e LISTA em C.

• Filas em C ;

Este tipo de estrutura de dados é dita ser FIFO (First in, first out), ou seja, o primeiro elemento a entrar na estrutura é o primeiro a sair. O nome fila, por si só, já é auto-explicativo. Imagine uma fila de banco. A primeira pessoa que chegou na fila, é a que vai ser atendida primeiro. Todas as operações em uma fila podem ser imaginadas como as que ocorrem numa fila de pessoas num banco, exceto que o elementos não se movem na fila, conforme o primeiro elemento é retirado. Isto seria muito custoso para o computador. O que se faz na realidade é indicar quem é o primeiro.

.São exemplos de uso de fila em um sistema:

1. Criação da fila (informar a capacidade no caso de implementação sequencial - vetor);
2. Enfileirar (enqueue) - o elemento é o parâmetro nesta operação;
3. Desenfileirar (dequeue);
4. Mostrar a fila (todos os elementos);
5. Verificar se a fila está vazia (isEmpty);
6. Verificar se a fila está cheia (is Full - implementação sequencial - vetor). Exemplo prático:



Exemplo de fila em c:
#include <stdio.h>

```

#define TAMANHO_MAX 5

typedef struct {
    int dados[TAMANHO_MAX];
    int inicio;
    int fim;
} Fila;

void inicializar(Fila* fila) {
    fila->inicio = -1;
    fila->fim = -1;
}

int estaVazia(Fila* fila) {
    return (fila->inicio == -1 && fila->fim == -1);
}

int estaCheia(Fila* fila) {
    return (fila->fim == TAMANHO_MAX - 1);
}

void enfileirar(Fila* fila, int item) {
    if (estaCheia(fila)) {
        printf("A fila está cheia. Não é possível enfileirar.\n");
        return;
    }

    if (estaVazia(fila)) {
        fila->inicio = 0;
    }

    fila->fim++;
    fila->dados[fila->fim] = item;

    printf("Enfileirado: %d\n", item);
}

int desenfileirar(Fila* fila) {
    if (estaVazia(fila)) {
        printf("A fila está vazia. Não é possível desenfileirar.\n");
        return -1; // Ou qualquer outro valor de erro apropriado
    }

    int item = fila->dados[fila->inicio];

    if (fila->inicio == fila->fim) {
        fila->inicio = -1;
        fila->fim = -1;
    } else {
        fila->inicio++;
    }
}

```

```
    }  
  
    printf("Desenfileirado: %d\n", item);  
    return item;  
}
```

```
int main() {  
    Fila fila;  
    inicializar(&fila);  
  
    enfileirar(&fila, 10);  
    enfileirar(&fila, 20);  
    enfileirar(&fila, 30);  
  
    desenfileirar(&fila);  
  
    enfileirar(&fila, 40);  
    enfileirar(&fila, 50);  
  
    desenfileirar(&fila);  
  
    return 0;  
}  
resultado:
```

```
    } Fila;

Enfileirado: 10
Enfileirado: 20
Enfileirado: 30
Desenfileirado: 10
Enfileirado: 40
Enfileirado: 50
Desenfileirado: 20

...Program finished
Press ENTER to exit
```

Aqui estão alguns cuidados a serem considerados ao usar filas:

1. Inicialização adequada: Certifique-se de inicializar corretamente a fila antes de começar a usá-la. Isso geralmente envolve definir os índices de início e fim para um estado válido, como -1, para indicar uma fila vazia.
2. Verificação de fila vazia e cheia: Sempre verifique se a fila está vazia antes de tentar desenfileirar elementos dela. Da mesma forma, verifique se a fila está cheia antes de tentar enfileirar elementos adicionais. Evite operações inválidas em filas vazias ou cheias.
3. Gerenciamento dos índices: Mantenha o controle adequado dos índices de início e fim da fila. Eles devem ser atualizados corretamente sempre que ocorrerem operações de enfileirar ou desenfileirar. Certifique-se de atualizar os índices adequadamente para evitar erros lógicos.
4. Limite de capacidade: Lembre-se de que uma fila tem uma capacidade máxima definida. Certifique-se de que o tamanho da fila não ultrapasse esse limite para evitar erros de estouro de memória ou substituição de dados indesejados.

- Lista em C;

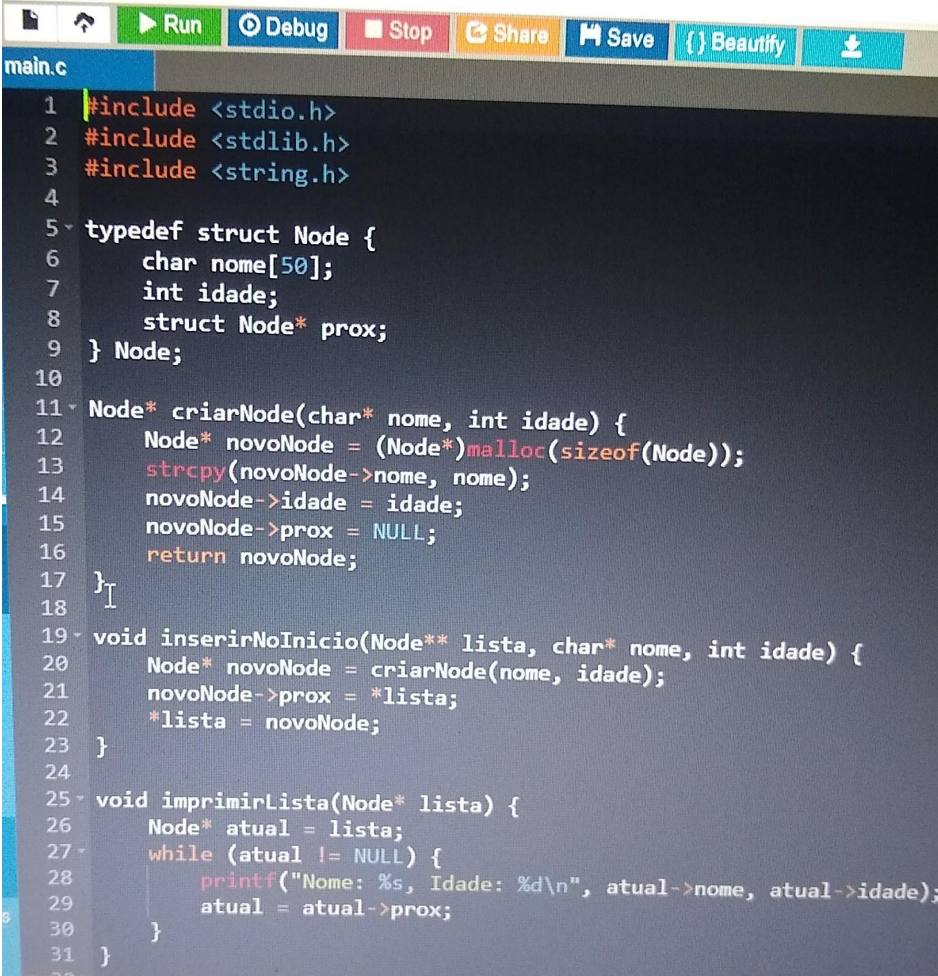
Tradicionalmente, listas em C são implementadas através de estruturas (associadas aos nós) armazenadas na memória dinâmica. A estrutura que implementa um nó de

uma lista ligada deve incluir, além do conteúdo da informação do nó, um ponteiro para o próximo nó. Em C, a alocação de memória é suportada pela rotina malloc. Esta rotina recebe um argumento, que é a dimensão em bytes da área necessária. O valor de retorno é o endereço do início da área que o sistema operacional alocou para este pedido, um ponteiro para o tipo void. Por exemplo, para reservar o espaço necessário para o nó de uma lista, seria necessário ter inicialmente declarado a variável que receberá o ponteiro para um nó.

•Apresentando alguns tipos de lista em C :

1. Listas encadeadas simples - Cada item aponta apenas para o seu sucessor;
2. Listas duplamente encadeadas - Cada item aponta tanto para o seu sucessor quanto para o antecessor;
3. Listas encadeadas circulares -
O último item aponta para o primeiro
4. Listas encadeadas circulares com sentinela - A lista começa e termina em um item sentinela cujos valores são nulos;

•Exemplo de código utilizando lista :



```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 typedef struct Node {
6     char nome[50];
7     int idade;
8     struct Node* prox;
9 } Node;
10
11 Node* criarNode(char* nome, int idade) {
12     Node* novoNode = (Node*)malloc(sizeof(Node));
13     strcpy(novoNode->nome, nome);
14     novoNode->idade = idade;
15     novoNode->prox = NULL;
16     return novoNode;
17 }
18
19 void inserirNoInicio(Node** lista, char* nome, int idade) {
20     Node* novoNode = criarNode(nome, idade);
21     novoNode->prox = *lista;
22     *lista = novoNode;
23 }
24
25 void imprimirLista(Node* lista) {
26     Node* atual = lista;
27     while (atual != NULL) {
28         printf("Nome: %s, Idade: %d\n", atual->nome, atual->idade);
29         atual = atual->prox;
30     }
31 }
```



```

32
33 void liberarLista(Node** lista) {
34     Node* atual = *lista;
35     Node* proximo;
36     while (atual != NULL) {
37         proximo = atual->prox;
38         free(atual);
39         atual = proximo;
40     }
41     *lista = NULL;
42 }
43
44 int main() {
45     Node* lista = NULL;
46
47     inserirNoInicio(&lista, "João", 25);
48     inserirNoInicio(&lista, "Maria", 30);
49     inserirNoInicio(&lista, "Pedro", 40);
50
51     imprimirLista(lista);
52
53     liberarLista(&lista);
54
55     return 0;
56 }

```

Resultado:

```

32
33 void liberarLista(Node** lista) {
34     Node* atual = *lista;
35     Node* proximo;
36     while (atual != NULL) {
37         proximo = atual->prox;
38         free(atual);
39         atual = proximo;

```

Nome: Pedro, Idade: 40
Nome: Maria, Idade: 30
Nome: João, Idade: 25

...Program finished with exit code 0
Press ENTER to exit console.

Alguns dos cuidados que você deve ter ao usar matrizes em C:

1. Tamanho máximo da lista: Ao criar uma lista, você deve definir um tamanho máximo adequado para acomodar o número máximo de elementos que serão armazenados

nela. É importante garantir que o tamanho máximo da lista não seja ultrapassado durante a inserção de elementos.

2. Verificação de limites: Sempre verifique os limites da lista antes de acessar ou modificar elementos. Certifique-se de que o índice do elemento esteja dentro dos limites válidos da lista para evitar acessos a posições inválidas.
3. Alocação de memória: Ao criar uma lista dinamicamente usando a função malloc ou similar, lembre-se de liberar a memória alocada usando free quando a lista não for mais necessária. Isso ajuda a evitar vazamentos de memória.

- Matriz em C;

As matrizes são estruturas de dados fundamentais na programação e matemática. Em C, as matrizes são representadas como arranjos bidimensionais, o que significa que você pode organizar os elementos em linhas e colunas.

A declaração de uma matriz em C segue a seguinte sintaxe:

```
C

tipo nome_da_matriz[linhas][colunas];
```

Aqui, "tipo" representa o tipo de dado dos elementos da matriz (como int, float, char, etc.), "nome_da_matriz" é o nome que você escolhe para sua matriz, "linhas" representa o número de linhas que a matriz terá e "colunas" representa o número de colunas.

- Exemplo de código utilizando matriz :

```
1 #include <stdio.h>
2
3
4 int main()
5 {
6     int i,j;
7     int matriz [4][4];
8     printf("Informe numeros inteiros\n");
9     for(i=0;i<4;i++){
10         for(j=0;j<4;j++){
11             scanf("%d",&matriz[i][j]);
12         }
13     }
14     printf("mostrando:\n");
15     for(i=0;i<4;i++){
16         for(j=0;j<4;j++){
17             printf("%d ",matriz[i][j]);
18         }
19         printf("\n");
20     }
21     return 0;
22 }
```

- Resultado:

```
Informe numeros inteiros
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
mostrando:
10 11 12 13
14 15 16 17
18 19 20 21
22 23 24 25
```

•Alguns dos cuidados que você deve ter ao usar matrizes em C:

1. Índices fora dos limites: Certifique-se de usar índices válidos ao acessar elementos da matriz. Os índices de matriz em C começam em 0 e vão até o tamanho da dimensão correspondente a menos um. Acessar elementos fora desses limites pode causar comportamento indefinido ou falhas no programa.
2. Tamanho da matriz: Verifique se o tamanho da matriz é suficiente para armazenar todos os elementos necessários. Se você tentar armazenar mais elementos do que o tamanho da matriz permite, ocorrerá um estouro de buffer, o que pode corromper a memória e causar falhas no programa.
3. Inicialização: Sempre inicialize a matriz antes de usar seus elementos. Matrizes não inicializadas podem conter valores aleatórios, o que pode levar a resultados imprevisíveis.