gthc.c low_level.rs

```
GREEN TEAM HACKER CLUB {
    [Programação Low Level]

        < Listas, Pilhas e Filas >

}
```

UFABC GitHub Low Level

Tabela de ‘Conteúdo’ {

}

gthc.c

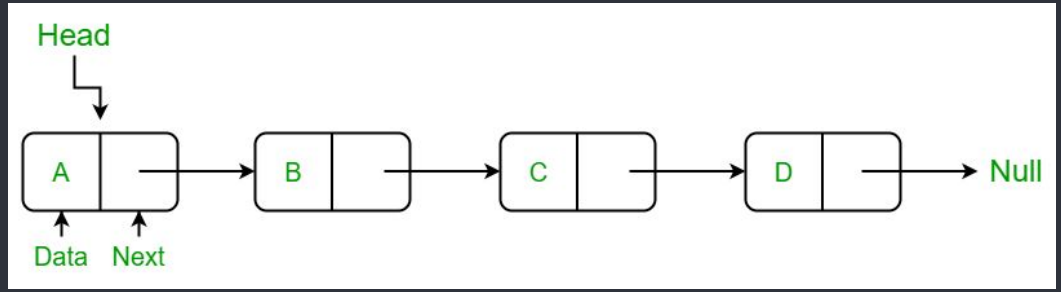
low_level.rs

```
01 {
    [Linked Lists]
}
```


Definição < /1 > {

Em ciência da computação, uma lista encadeada é uma coleção linear de elementos de dados cuja ordem não é dada por sua colocação física na memória. Em vez disso, cada elemento aponta para o próximo. É uma estrutura de dados que consiste em uma coleção de nós que juntos representam uma sequência. Em sua forma mais básica, cada nó contém dados e uma referência (em outras palavras, um link) para o próximo nó na sequência.

}



Tempo < /2 > {

Operation	Time Complexity: Worst Case	Average Case
Insert at beginning or end	O(1)	O(1)
Delete at beginning or end	O(1)	O(1)
Search	O(n)	O(n)
Access	O(n)	O(n)

Não é possível realizar acesso aleatório

}

Operações < /1 > {

A implementação da estrutura de dados da lista pode fornecer algumas das seguintes operações:

- criar
- testar para vazio
- adicionar item ao início ou fim
- acessar o primeiro ou último item
- acessar um item por índice

}

Implementação< /1 > {

```
// Definition of a Node in a singly linked list
struct Node {

    // Data part of the node
    int data;

    // Pointer to the next node in the list
    Node* next;

    // Constructor to initialize the node with data
    Node(int data)
    {
        this->data = data;
        this->next = nullptr;
    }
};
```

}

Implementação< /1 > {

```c
// Definition of a Node in a singly linked list
struct Node {
    int data;
    struct Node* next;
};

// Function to create a new Node
struct Node* newNode(int data) {
    struct Node* temp =
      (struct Node*)malloc(sizeof(struct Node));
    temp->data = data;
    temp->next = NULL;
    return temp;
}
```

}

Implementação< /1 > {

```c
// Function to search for a value in the Linked List
bool searchLinkedList(struct Node* head, int target)
{
    // Traverse the Linked List
    while (head != NULL) {

        // Check if the current node's
        // data matches the target value
        if (head->data == target) {
            return true; // Value found
        }

        // Move to the next node
        head = head->next;
    }

    return false; // Value not found
}
```

}

Implementação< /1 > {

```c
// Function to insert a new node at the beginning of the linked list
struct Node* insertAtBeginning(struct Node* head, int value)
{
    // Create a new node with the given value
    struct Node* new_node = newNode(value);

    // Set the next pointer of the new node to the current head
    new_node->next = head;

    // Move the head to point to the new node
    head = new_node;

    // Return the new head of the linked list
    return head;
}
```

}

Implementação< /1 > {

```
// Function to insert a node at the end of the linked list
struct Node* insertAtEnd(struct Node* head, int value)
{
    // Create a new node with the given value
    struct Node* new_node = newNode(value);
    // If the list is empty, make the new node the head
    if (head == NULL)
        return new_node;
    // Traverse the list until the last node is reached
    struct Node* curr = head;
    while (curr->next != NULL) {
        curr = curr->next;
    }
    // Link the new node to the current last node
    curr->next = new_node;

    return head;
}
```

}

Implementação< /1 > {

```
// Function to insert a node at a specified position
struct Node* insertPos(struct Node* head, int pos, int data) {
    if (pos < 1) {
        printf("Invalid position!\n");
        return head;
    }

    // Special case for inserting at the head
    if (pos == 1) {
        struct Node* temp = getNode(data);
        temp->next = head;
        return temp;
    }

    // Traverse the list to find the node
    // before the insertion point
    struct Node* prev = head;
    int count = 1;
    while (count < pos - 1 && prev != NULL) {
        prev = prev->next;
        count++;
    }

    // If position is greater than the
number of nodes
    if (prev == NULL) {
        printf("Invalid position!\n");
        return head;
    }

    // Insert the new node at the
specified position
    struct Node* temp = getNode(data);
    temp->next = prev->next;
    prev->next = temp;

    return head;
}
```

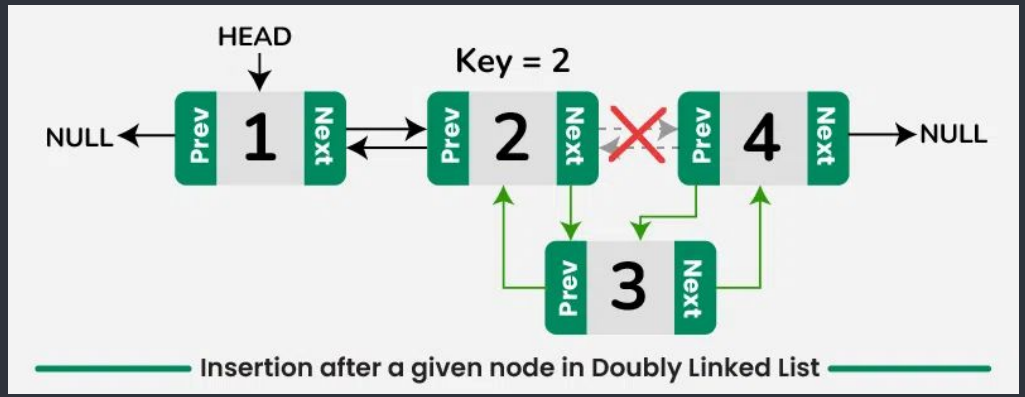
}

```
02 {
    [Double Linked Lists]
}
```


Definição < /1 > {

Em uma 'lista duplamente ligada, cada nó contém, além do link do próximo nó, um segundo campo de link apontando para o nó 'anterior' na sequência. Os dois links podem ser chamados de 'para frente(s') e 'para trás'. ', ou 'próximo' e 'anterior'('anterior').

}



Tempo < /2 > {

Operation	Time Complexity: Worst Case	Average Case
Insert at beginning or end	O(1)	O(1)
Delete at beginning or end	O(1)	O(1)
Search	O(n)	O(n)
Access	O(n)	O(n)

}

```
1  Implementação< /1 > {
2
3
4      Link da Implementação
5
6
7
8
9
10 }
11
12
13
14
```


gthc.c

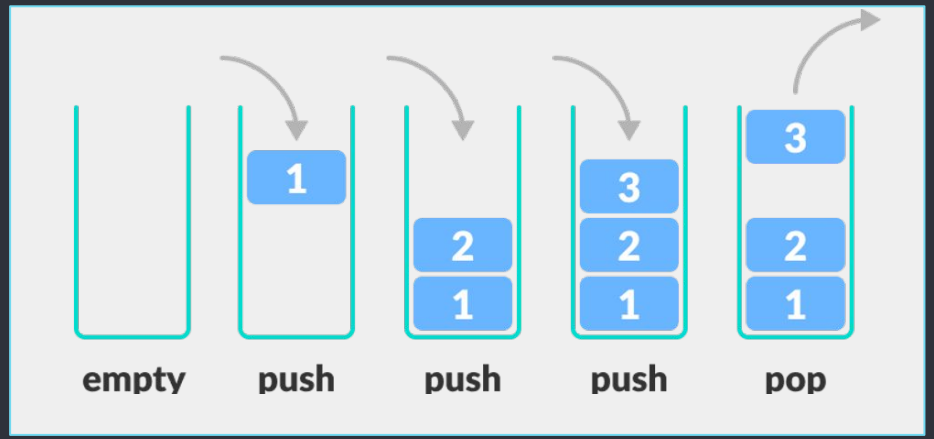
low_level.rs

```
03 {
    [Stack]
}
```


Definição < /1 > {

Uma pilha é uma estrutura de dados linear que segue uma ordem específica em que as operações são executadas. A ordem pode ser LIFO (Last In First Out) ou FILO (First In Last Out).

}



Tempo < /2 > {

Operation	Time Complexity: Worst Case	Average Case
Push	O(1)	O(1)
Pop	O(1)	O(1)

}

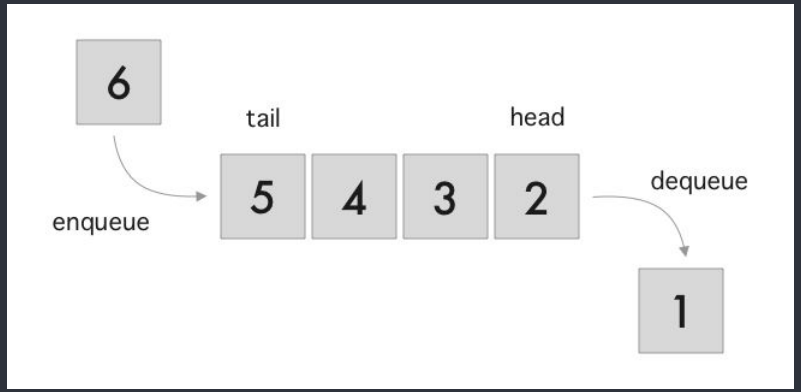
```
1  Implementação< /1 > {
2
3
4      Link da Implementação
5
6
7
8
9
10 }
11
12
13
14
```


```
04 {
    [Queue]
}
```


Definição < /1 > {

Uma Estrutura de Dados de Fila é um conceito fundamental em ciência da computação utilizado para armazenar e gerenciar dados em uma ordem específica. Segue o princípio “Primeiro a entrar, primeiro a sair” (FIFO), onde o primeiro elemento adicionado à fila é o primeiro. para ser removido.

}



Tempo < /2 > {

Operation	*Time Complexity:* *Worst Case*	*Average Case*
Enqueue	*O(1)*	*O(1)*
Dequeue	*O(1)*	*O(1)*

}

```
1  Implementação< /1 > {
2
3
4      Link da Implementação
5
6
7
8
9
10 }
11
12
13
14
```


Visualização {

<Data Structure Visualizations - https://www.cs.usfca.edu/~galles/visualization/Algorithms.html >

}

Referências {

<Data Structures Using C 2nd edition - https://github.com/GauravWalia19/Free-Algorithms-Books/blob/main/Library/src/C/Data-Structures-Using-C-2nd-edition.pdf >

}

Próximo Encontro {

< Árvores>

}

```
Obrigado; {

    'Dúvidas?'

        luccas.h.cortes@hotmail.com
        https://github.com/Cortesz/

                CREDITS: This presentation template was
                created by Slidesgo, including icons by
                Flaticon, and infographics & images by Freepik

                < https://github.com/greenteamhc >

}
```