

The background of the entire image is a vibrant, futuristic cityscape at night. Numerous tall, dark skyscrapers are illuminated with glowing blue and white lights, creating a sense of depth and scale. In the center, a bright, orange-yellow light source, possibly a setting or rising sun, casts a warm glow across the scene. A prominent feature is a long, straight road or data stream that runs from the foreground towards the horizon, flanked by dark, industrial-looking structures. The sky is filled with streaks of light, suggesting fast-moving objects or data transmission. The overall color palette is dominated by deep blues, oranges, and yellows, giving it a high-tech, cybernetic feel.

ESTRUTURAS DE DADOS

Domine com C

MURIELLY O. NASCIMENTO

ESTRUTURAS DE DADOS

Domine com C

"Mas tudo deve ser feito com decência e ordem."

1 Coríntios 14:40

MURIELLY O. NASCIMENTO

Introdução

Este e-book é seu guia para entender e dominar as principais estruturas de dados usando a linguagem de programação C. Apresentaremos cada estrutura de dados de forma simples e objetiva, acompanhada de exemplos práticos em código.



CAPÍTULO 01

Arrays: A Base de Tudo



Arrays: A Base de Tudo

Arrays são coleções de elementos do mesmo tipo armazenados em posições contíguas na memória. São úteis quando você sabe o número exato de elementos que precisa armazenar.

```
Arrays

#include <stdio.h>

int main() {
    int arr[5] = {1, 2, 3, 4, 5};

    // Acessando elementos do array
    for(int i = 0; i < 5; i++) {
        printf("Elemento %d: %d\n", i, arr[i]);
    }

    return 0;
}
```

CAPÍTULO 03

Listas Ligadas: Flexibilidade Dinâmica



Listas Ligadas: Flexibilidade Dinâmica

Listas ligadas são coleções de nós onde cada nó contém um valor e um ponteiro para o próximo nó na sequência. São úteis quando você precisa de uma coleção de tamanho dinâmico.

```
int main() {
    struct Node* head = NULL;

    push(&head, 1);
    push(&head, 2);
    push(&head, 3);

    printList(head);

    return 0;
}
```

Listas Ligadas

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* next;
};

// Função para adicionar um nó no início
void push(struct Node** head_ref, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

// Função para imprimir a lista
void printList(struct Node* node) {
    while (node != NULL) {
        printf("%d ", node->data);
        node = node->next;
    }
}
```

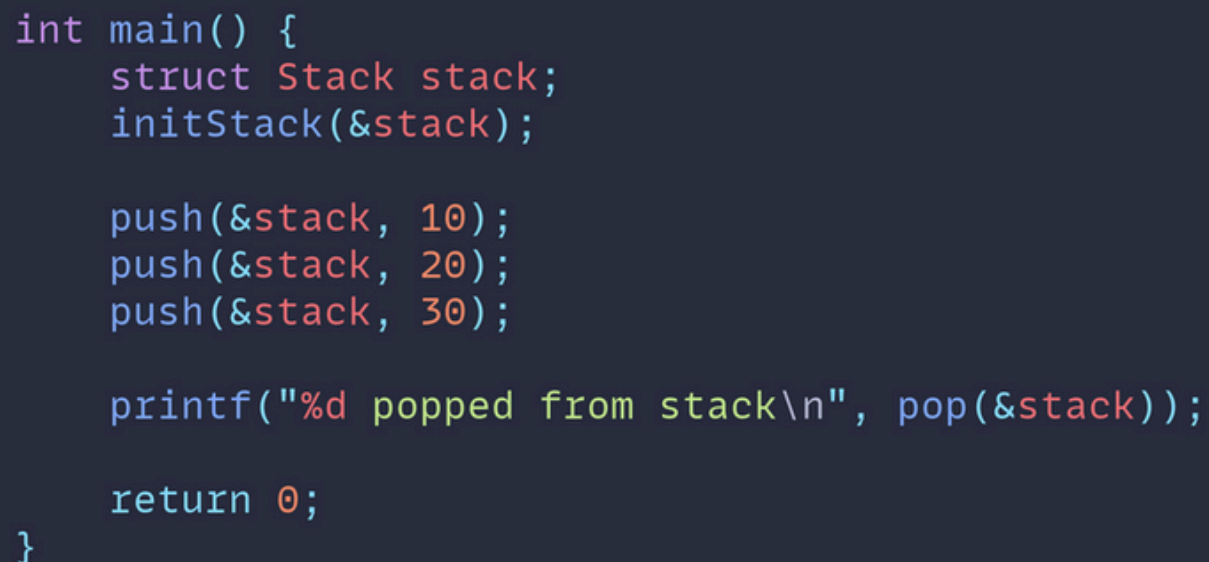

CAPÍTULO 04

Pilhas: LIFO em Ação



Pilhas: LIFO em Ação

Pilhas são coleções de elementos com acesso restrito, seguindo o princípio LIFO (Last In, First Out). São usadas, por exemplo, em algoritmos de reversão de strings.



```
int main() {  
    struct Stack stack;  
    initStack(&stack);  
  
    push(&stack, 10);  
    push(&stack, 20);  
    push(&stack, 30);  
  
    printf("%d popped from stack\n", pop(&stack));  
  
    return 0;  
}
```

```
#include <stdio.h>
#include <stdlib.h>

#define MAX 1000

struct Stack {
    int top;
    int arr[MAX];
};

void initStack(struct Stack* stack) {
    stack->top = -1;
}

int isFull(struct Stack* stack) {
    return stack->top == MAX - 1;
}

int isEmpty(struct Stack* stack) {
    return stack->top == -1;
}

void push(struct Stack* stack, int item) {
    if (isFull(stack)) {
        printf("Stack overflow\n");
        return;
    }
    stack->arr[++stack->top] = item;
}

int pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack underflow\n");
        return -1;
    }
    return stack->arr[stack->top--];
}
```

CAPÍTULO 05

Filas: FIFO em Prática



Filas: FIFO em Prática

Filas são coleções de elementos que seguem o princípio FIFO (First In, First Out). São usadas em sistemas de gerenciamento de tarefas, como filas de impressão.

```
Filas

int main() {
    struct Queue* queue = createQueue(1000);

    enqueue(queue, 10);
    enqueue(queue, 20);
    enqueue(queue, 30);

    printf("%d dequeued from queue\n", dequeue(queue));

    return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>

struct Queue {
    int front, rear, size;
    unsigned capacity;
    int* array;
};

struct Queue* createQueue(unsigned capacity) {
    struct Queue* queue = (struct Queue*) malloc(sizeof(struct Queue));
    queue->capacity = capacity;
    queue->front = queue->size = 0;
    queue->rear = capacity - 1;
    queue->array = (int*) malloc(queue->capacity * sizeof(int));
    return queue;
}

int isFull(struct Queue* queue) {
    return (queue->size == queue->capacity);
}

int isEmpty(struct Queue* queue) {
    return (queue->size == 0);
}

void enqueue(struct Queue* queue, int item) {
    if (isFull(queue))
        return;
    queue->rear = (queue->rear + 1) % queue->capacity;
    queue->array[queue->rear] = item;
    queue->size = queue->size + 1;
    printf("%d enqueued to queue\n", item);
}

int dequeue(struct Queue* queue) {
    if (isEmpty(queue))
        return -1;
    int item = queue->array[queue->front];
    queue->front = (queue->front + 1) % queue->capacity;
    queue->size = queue->size - 1;
    return item;
}
```

CAPÍTULO 06

Árvores Binárias: Hierarquia Eficiente



Árvores Binárias: Hierarquia Eficiente

Árvores binárias são estruturas hierárquicas onde cada nó tem no máximo dois filhos. São úteis para operações rápidas de busca, inserção e exclusão.

```
Árvores

int main() {
    struct Node* root = newNode(1);
    root→left = newNode(2);
    root→right = newNode(3);
    root→left→left = newNode(4);
    root→left→right = newNode(5);

    printf("Inorder traversal: ");
    inorder(root);

    return 0;
}
```


Árvores

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

struct Node* newNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
}

void inorder(struct Node* root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}
```

CAPÍTULO 07

Conclusão



Conclusão

Neste e-book, abordamos as principais estruturas de dados em C com exemplos práticos. Esperamos que você tenha encontrado este guia útil para entender e aplicar essas estruturas em seus projetos. Continue explorando e praticando para dominar completamente essas ferramentas poderosas da programação.



E-book gerado por uma IA e editado por um humano.
2024

