Bài 11: Stack - Queue

# Stack

Stack (ngăn xếp) là một cấu trúc dữ liệu tuân theo nguyên tắc "Last In, First Out" (LIFO), nghĩa là phần tử cuối cùng được thêm vào stack sẽ là phần tử đầu tiên được lấy ra.

Các thao tác cơ bản trên stack bao gồm:

* “ push” (đẩy) để thêm một phần tử vào đỉnh của stack.
* “ pop” để xóa một phần tử ở đỉnh stack.
* “ top” để lấy giá trị của phần tử ở đỉnh stack.

| #**include** <stdio.h> #**include** <stdlib.h>  **typedef** **struct** **Stack** {  **int**\* items;  **int** size;  **int** top; } Stack;  **void** **initialize**( Stack \*stack, **int** size) {  stack->items = (**int**\*) malloc(**sizeof**(**int**) \* size);  stack->size = size;  stack->top = -1; }  **int** **is\_empty**( Stack stack) {  **return** stack.top == -1; }  **int** **is\_full**( Stack stack) {  **return** stack.top == stack.size - 1; }  **void** **push**( Stack \*stack, **int** value) {  **if** (!is\_full(\*stack)) {  stack->items[++stack->top] = value;  } **else** {  printf("Stack overflow\n");  } }  **int** **pop**( Stack \*stack) {  **if** (!is\_empty(\*stack)) {  **return** stack->items[stack->top--];  } **else** {  printf("Stack underflow\n");  **return** -1;  } }  **int** **top**( Stack stack) {  **if** (!is\_empty(stack)) {  **return** stack.items[stack.top];  } **else** {  printf("Stack is empty\n");  **return** -1;  } }  **int** **main**() {  Stack stack1;  initialize(&stack1, 5);    push(&stack1, 10);  push(&stack1, 20);  push(&stack1, 30);  push(&stack1, 40);  push(&stack1, 50);  push(&stack1, 60);   printf("Top element: %d\n", top(stack1));   printf("Pop element: %d\n", pop(&stack1));  printf("Pop element: %d\n", pop(&stack1));   printf("Top element: %d\n", top(stack1));   **return** 0; } |
| --- |

# Queue

Queue là một cấu trúc dữ liệu tuân theo nguyên tắc "First In, First Out" (FIFO), nghĩa là phần tử đầu tiên được thêm vào hàng đợi sẽ là phần tử đầu tiên được lấy ra.

Các thao tác cơ bản trên hàng đợi bao gồm:

* “enqueue” (thêm phần tử vào cuối hàng đợi)
* “dequeue” (lấy phần tử từ đầu hàng đợi).
* “front” để lấy giá trị của phần tử đứng đầu hàng đợi.

| #**include** <stdio.h> #**include** <stdlib.h>   **typedef** **struct** **Queue** {  **int**\* items;  **int** size;  **int** front, rear; } Queue;  **void** **initialize**(Queue \*queue, **int** size)  {  queue->items = (**int**\*) malloc(**sizeof**(**int**)\* size);  queue->front = -1;  queue->rear = -1;  queue->size = size; }  **int** **is\_empty**(Queue queue) {  **return** queue.front == -1; }  **int** **is\_full**(Queue queue) {  **return** (queue.rear + 1) % queue.size == queue.front; }  **void** **enqueue**(Queue \*queue, **int** value) {  **if** (!is\_full(\*queue)) {  **if** (is\_empty(\*queue)) {  queue->front = queue->rear = 0;  } **else** {  queue->rear = (queue->rear + 1) % queue->size;  }  queue->items[queue->rear] = value;  } **else** {  printf("Queue overflow\n");  } }  **int** **dequeue**(Queue \*queue) {  **if** (!is\_empty(\*queue)) {  **int** dequeued\_value = queue->items[queue->front];  **if** (queue->front == queue->rear) {  queue->front = queue->rear = -1;  } **else** {  queue->front = (queue->front + 1) % queue->size;  }  **return** dequeued\_value;  } **else** {  printf("Queue underflow\n");  **return** -1;  } }  **int** **front**(Queue queue) {  **if** (!is\_empty(queue)) {  **return** queue.items[queue.front];  } **else** {  printf("Queue is empty\n");  **return** -1;  } }  **int** **main**() {  Queue queue;  initialize(&queue, 3);   enqueue(&queue, 10);  enqueue(&queue, 20);  enqueue(&queue, 30);   printf("Front element: %d\n", front(queue));   printf("Dequeue element: %d\n", dequeue(&queue));  printf("Dequeue element: %d\n", dequeue(&queue));   printf("Front element: %d\n", front(queue));   enqueue(&queue, 40);  enqueue(&queue, 50);  printf("Dequeue element: %d\n", dequeue(&queue));  printf("Front element: %d\n", front(queue));   **return** 0; } |
| --- |