STACK và QUEUE

MỤC TIÊU

Hoàn tất phần thực hành này, sinh viên có thể:

- Hiểu được cách thức sử dụng stack và queue trên cơ sở sử dụng danh sách liên kết để cài đặt.
- Hiểu và vận dụng các cấu trúc stack và queue trong những bài toán đơn giản.

Thời gian thực hành: 120 phút đến 360 phút.

Lưu ý: yêu cầu vận dụng thành thạo danh sách liên kết ở Lab02.

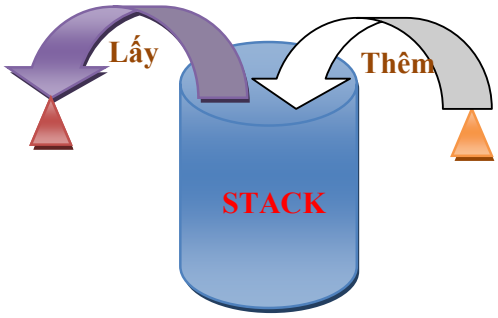
TÓM TẮT

- Stack (ngăn xếp) và queue (hang đợi) là những cấu trúc dữ liệu dùng để lưu trữ các phần tử của tập hợp theo những nguyên tắc đặc biệt khi thêm phần tử cũng như lấy phần tử ra khỏi cấu trúc.
- Stack (last in, first out – LIFO): phần tử vào stack sau cùng, là phần tử được lấy ra khỏi stack trước nhất.
- Queue (first in, first out – FIFO): phần tử vào queue trước nhất, là phần tử được lấy ra khỏi queue trước nhất.

☺Lab03 là phần vận dụng danh sách liên kết đã thực hành ở Lab02 để cài đặt Stack và Queue.

(Lưu ý: chúng ta cũng có thể dùng mảng để cài đặt stack và queue, nhưng mảng đặc trưng cho cơ chế tĩnh, do vậy danh sách liên kết – cơ chế động - là cấu trúc tốt hơn mảng khi hiện thực Stack và Queue).

Ví dụ: minh họa Stack



+ Phần tử mới được thêm vào đỉnh của ngăn xếp.

+ Thao tác lấy phần tử ra khỏi ngăn xếp, nếu ngăn xếp khác rỗng thì phần tử ở đầu ngăn xếp được lấy ra, ngược lại, ngăn xếp rỗng thì thao tác lấy phần tử thất bại.

Ví dụ: minh họa Queue

+ Phần tử được thêm vào ở đầu queue. Do vậy, phần tử vào đầu tiên sẽ ở đáy của queue. Do vậy, khi lấy phần tử ra, nếu queue khác rỗng thì phần tử ở đáy queue được lấy ra, ngược lại, queue bị rỗng thì thao tác lấy phần tử ra khỏi queue thất bại.

NỘI DUNG THỰC HÀNH

Cơ bản

Yêu cầu: cài đặt stack và queue bằng danh sách liên kết.

Do đặc trưng của stack và queue, chúng ta cần xây dựng 2 thao tác chính là thêm 1 phần tử vào stack hoặc queue, và lấy 1 phần tử ra khỏi stack hoặc queue.

Dựa vào nguyên tắc thêm và lấy phần tử ra khỏi stack/queue, ta cần xây dựng các hàm sau:

- Đối với Stack
  - Thêm phần tử: thêm phần tử vào đầu danh sách liên kết.
  - Lấy phần tử: lấy phần tử ở đầu danh sách ra khỏi danh sách liên kết.

(Lưu ý: ta cũng có thể thêm phần tử vào cuối danh sách liên kết, do vậy thao tác lấy phần tử, ta thực hiện lấy phần tử ở cuối danh sách liên kết).

- Đối với Queue
  - Thêm phần tử: thêm vào đầu danh sách liên kết.
  - Lấy phần tử: lấy phần tử ở cuối danh sách liên kết.

(Lưu ý: ta cũng có thể thực hiện việc thêm phần tử vào cuối danh sách liên kết và lấy ra ở đầu danh sách liên kết).

Sử dụng bài tập Lab02

Chương trình mẫu

```
#include <stdio.h>

struct NODE{
	int Key;
	NODE *pNext;
};

NODE* CreateNode(int Data)
{
	NODE* pNode;
	pNode = new NODE;
	if (pNode == NULL)
		return NULL;
	pNode->Key = Data;
	pNode->pNext = NULL;
	return pNode;
}

bool AddHead(NODE* &pHead, int Data)
{
	NODE *pNode;
	pNode = CreateNode(Data);
	if (pNode == NULL)
		return false;
	if (pHead == NULL)
		pHead = pNode;
	else  {
		pNode->pNext = pHead;
		pHead = pNode;
```

```
	}
	return true;
}

NODE* RemoveHead(NODE* &pHead)
{
	if(pHead == NULL)
		return NULL;
	NODE* pResult = pHead;
	pHead = pHead->pNext;
	return pResult;
}

NODE* RemoveTail(NODE* &pHead)
{
	NODE *pNode;
	if(pHead == NULL)                          //<1>
	{
		return NULL;
	}
	else if(pHead->pNext == NULL)              //<2>
	{
		pNode = pHead;
		pHead = NULL;
		return pNode;
	}

	pNode = pHead;
	while(pNode->pNext->pNext != NULL)         //<3>
	{
		pNode = pNode->pNext;
	}

	NODE* pResult = pNode->pNext;
	pNode->pNext = NULL;
	return pResult;
}

//-------STACK :
//----PUSH tương ứng AddHead
//----POP tương ứng RemoveHead

bool PushStack(NODE* &pStack, int Data)
{
	return AddHead(pStack, Data);
}

NODE* PopStack(NODE* &pStack)
{
	return RemoveHead(pStack);
}

//--------QUEUE :
//----ENQUEUE tương ứng AddHead
//----DEQUEUE tương ứng RemoveTail
bool EnQueue(NODE* &pQueue, int Data)
{
	return AddHead(pQueue, Data);
}

NODE* DeQueue(NODE* &pQueue)
{
	return RemoveTail(pQueue);
}
```

```
void main()
{
	NODE* pStack = NULL;
	NODE* pQueue = NULL;

	int n = 10;
	while(n!=0)
	{
		PushStack(pStack, n);
		EnQueue(pQueue, n);
		n--;
	}

	NODE* pNode = DeQueue(pQueue);
	if(pNode != NULL)                    //<4>
		printf("\nGia tri phan tu (Queue) : %d\n", pNode->Key);
	else
		printf("\nNULL\n");

	NODE* pNode2 = PopStack(pStack);
	if(pNode2 != NULL)
		printf("\nGia tri phan tu (Stack) : %d\n", pNode2->Key);
	else
		printf("\nNULL\n");
}
```

1. Biên dịch đoạn chương trình trên.
2. Thay giá trị n=10 thành n=1 trong main, đọc giá trị Queue và Stack in ra màn hình.
3. Giải thích <4> khi nào giá trị pNode khác NULL, khi nào pNode bằng NULL, ý nghĩa của mỗi trường hợp trên.
4. Giải thích hàm RemoveTail ở các điểm <1>, <2>, <3> cho biết ý nghĩa của chúng.
5. Sử dụng các hàm PushStack, PopStack, EnQueue, DeQueue để cài đặt.
   a. Về Stack: Trong hàm main, thực hiện việc thêm vào 3 giá trị do người dùng nhập vào (thực hiện 3 lệnh thêm phần tử vào stack), sau đó thực hiện 4 lần lệnh lấy giá trị phần tử ra khỏi stack, nếu có, in giá trị phần tử ra màn hình, nếu không có (stack rỗng), in ra màn hình "STACK RONG, KHONG LAY DUOC PHAN TU".
   b. Về Queue: Trong hàm main, thực hiện việc thêm vào 3 giá trị do người dùng nhập vào (thực hiện 3 lần lệnh thêm phần tử vào queue), sau đó thực hiện 4 lần lệnh lấy giá trị phần tử ra khỏi queue, nếu có, in giá trị phần tử ra màn hình, nếu không có (queue rỗng), in ra màn hình "QUEUE RONG, KHONG LAY DUOC PHAN TU".

Áp dụng – Nâng cao

1. Cài đặt hàm AddTail để có phiên bản cài đặt Stack (thêm phần tử vào cuối danh sách và lấy phần tử ở cuối danh sách liên kết) cũng như áp dụng 1 phiên bản khác khi cài đặt Queue (thêm phần tử vào cuối danh sách liên kết và lấy phần tử ở đầu danh sách liên kết).
2. Nhận xét cách cài đặt trên ở phần 1 (áp dụng – nâng cao) so với chương trình mẫu đối với trường hợp stack cũng như queue.
3. Sử dụng cấu trúc Stack để chuyển giá trị từ cơ số 10 sang cơ số 2.
   Gợi ý : thực hiện việc chia liên tiếp giá trị trong cơ số 10 cho 2, lấy phần dư đưa vào stack, cho đến khi giá trị đem đi chia là 0. In giá trị trong stack ra (đó chính là kết quả khi chuyển số từ hệ cơ số 10 sang hệ cơ số 2).

BÀI TẬP THÊM

Tìm đường trong mê cung (thực hiện loang theo chiều rộng <sử dụng queue> hoặc loang theo chiều sâu <sử dụng stack>).

Bài toán: cho ma trận mxn, mỗi phần tử là số 0 hoặc 1.

Giá trị 1 : có thể đi tới và giá trị 0 : không thể đi tới được.

Câu hỏi:

Từ ô ban đầu có tọa độ (x1, y1) có thể đi tới ô (x2, y2) không?

Biết rằng từ 1 ô (x,y) chỉ có thể đi qua ô có chung cạnh với ô đang đứng và mang giá trị là 1, ngược lại không có đường đi.