**Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Biểu tượng máy tính

Mô tả được tạo tự độngĐẠI HỌC UEH**

**TRƯỜNG CÔNG NGHỆ VÀ THIẾT KẾ**

**KHOA KẾ CÔNG NGHỆ THÔNG TIN KINH DOANH**

A blue and orange text on a black background

AI-generated content may be incorrect.

**ĐỒ ÁN MÔN HỌC**

|  |
| --- |
| **Đề tài:**  **Ứng dụng Stack trong quản lý bộ đệm** |

**MÔN HỌC: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Giảng viên** | : | TS. Đặng Ngọc Hoàng Thành | |  | |
| **Mã lớp học phần** | : | 25D1INF50900702 |  | |
| **Sinh viên thực hiện** | : | Thái Ngọc Bảo Châu  Phan Nhựt Đăng Khoa  Trần Xuân Ngọc  Mai Đức Phát |

TP Hồ Chí Minh, ngày 4 tháng 5 năm 2025

MỤC LỤC

**CHƯƠNG 1. TỔNG QUAN VỀ STACK VÀ ỨNG DỤNG TRONG QUẢN LÝ BỘ ĐỆM**

* 1. **Khái niệm liên quan**
     1. ***Stack***

Stack là một cấu trúc dữ liệu đơn giản được sử dụng để lưu trữ dữ liệu. Trong một ngăn xếp, thứ tự của dữ liệu đến là điều quan trọng. Ví dụ như chúng ta có một đống đĩa trong quán ăn tự phục vụ là một ví dụ điển hình về stack. Các đĩa được thêm vào chồng khi chúng được làm sạch và chúng được đặt lên trên cùng. Khi một tấm, được yêu cầu, nó được lấy từ trên cùng của chồng. Tấm đầu tiên được đặt trên chồng là tấm cuối cùng được sử dụng.

Stack là một danh sách có thứ tự trong đó việc chèn và xóa được thực hiện ở một đầu, được gọi là top - đỉnh. Phần tử cuối cùng được chèn là phần tử đầu tiên sẽ bị xóa. Do đó, nó được gọi là Last in First out (LIFO) hoặc First in Last out (FILO) list. Các tên đặc biệt được đặt cho hai thay đổi có thể được thực hiện đối với một ngăn xếp. Khi một phần tử được chèn vào một ngăn xếp, khái niệm được gọi là push và khi một phần tử bị xóa khỏi ngăn xếp, khái niệm được gọi là pop. Việc cố gắng pop một stack trống được gọi là underflow và cố gắng đẩy một phần tử trong một stack đầy được gọi là overflow.

A diagram of a diagram

AI-generated content may be incorrect.

Hình 1. Ví dụ về Stack

* + 1. ***Bộ đệm (Clipboard)***

Clipboard (còn gọi là bộ nhớ đệm) đây chính là một vùng lưu trữ dữ liệu tạm thời trên thiết bị điện tử. Khi các dữ liệu được lưu lại có thể tồn tại ở dạng đoạn văn bản, hình ảnh, tập tin hay file,… được người dùng copy hoặc cut ở một nơi khác.

Sau khi thực hiện các thao tác copy hoặc cut, các dữ liệu này sẽ được thiết bị tự động lưu lại. Tính năng này cho phép lưu trữ và hỗ trợ xem lại dữ liệu của 25 lần sao chép gần nhất. Clipboard có nguyên tắc hoạt động cũng giống với RAM, đó là sau khi tắt máy đi thì dữ liệu cũng tự động bị xóa.

A clipboard with a file transfer

AI-generated content may be incorrect.

Hình 2. Ví dụ về Clipboard

* 1. **Ứng dụng của Stack trong quản lý bộ đệm**

Trong lĩnh vực kỹ thuật phần mềm và hệ điều hành, quản lý bộ đệm (buffer management) là một tác vụ quan trọng nhằm đảm bảo hiệu suất và tính chính xác khi xử lý dữ liệu tạm thời. Một trong những cấu trúc dữ liệu được sử dụng phổ biến để phục vụ cho mục đích này là stack, nhờ vào đặc điểm truy xuất dữ liệu theo nguyên tắc LIFO.

Khi người dùng thực hiện thao tác copy hoặc cut, dữ liệu sẽ được lưu tạm vào một vùng nhớ gọi là clipboard. Mỗi lần thực hiện thao tác mới, dữ liệu mới được lưu đè lên bộ đệm. Tuy nhiên, trong các hệ thống hiện đại, thay vì lưu duy nhất một mục, clipboard có thể ghi nhớ nhiều mục sao chép gần nhất, cho phép người dùng xem lại và dán lại các nội dung đã sao chép trước đó. Tại đây, stack đóng vai trò là cấu trúc dữ liệu cốt lõi để tổ chức danh sách các lần sao chép theo đúng trình tự thời gian.

Ví dụ, trong các trình soạn thảo văn bản hoặc phần mềm thiết kế đồ họa, khi người dùng thực hiện nhiều thao tác sao chép/cắt, các nội dung này sẽ được lưu vào stack clipboard. Mỗi khi người dùng yêu cầu dán (paste), phần tử trên đỉnh stack (mục vừa sao chép gần nhất) sẽ được truy xuất. Người dùng cũng có thể chọn xem và dán lại các phần tử cũ hơn, bằng cách truy cập vào các phần tử thấp hơn trong stack.

* 1. **Cấu trúc và cài đặt Stack**
     1. ***Cấu trúc Stack trong chương trình***

Trong các ngôn ngữ lập trình hiện đại như C#, cấu trúc dữ liệu ngăn xếp (Stack) có thể được triển khai theo nhiều cách khác nhau, trong đó phổ biến nhất là:

* **Cài đặt bằng mảng tuyến tính (array-based stack):** Stack được xây dựng dựa trên một mảng có kích thước cố định hoặc động. Khi thêm hoặc loại bỏ phần tử, chương trình thay đổi chỉ số đỉnh (top index) của mảng. Ưu điểm của cách này là tốc độ truy cập nhanh và đơn giản trong cài đặt. Tuy nhiên, nhược điểm là khó linh hoạt khi số lượng phần tử thay đổi liên tục, hoặc cần mở rộng bộ nhớ động.
* **Cài đặt bằng danh sách liên kết đơn (linked list-based stack):** Mỗi phần tử trong Stack là một nút (node) chứa dữ liệu và một con trỏ liên kết đến nút kế tiếp. Phần tử mới luôn được thêm vào đầu danh sách (đỉnh Stack). Phương pháp này linh hoạt hơn về bộ nhớ vì không cần cấp phát bộ nhớ cố định trước, và phù hợp với ứng dụng có thao tác thêm/xóa liên tục.

Trong chương trình này, nhóm lựa chọn cài đặt Stack bằng danh sách liên kết đơn (singly linked list) thông qua lớp **CustomStack<T>**, thay vì sử dụng sẵn lớp Stack<T> trong thư viện .NET. Việc cài đặt thủ công cấu trúc Stack mang lại nhiều lợi ích trong bối cảnh của đề tài:

* Cho phép toàn quyền kiểm soát và tùy biến hành vi của Stack, chẳng hạn như bổ sung các phương thức như RemoveAt(int index) hoặc GetAt(int index) – những chức năng không có sẵn trong lớp Stack<T>.
* Dễ dàng tích hợp thêm các chức năng lưu trữ, ghi log, hoặc lọc dữ liệu nếu cần thiết trong các phiên bản mở rộng của ứng dụng.

Việc sử dụng danh sách liên kết cũng phù hợp với đặc thù của ứng dụng clipboard: người dùng có thể sao chép hoặc cắt liên tục nhiều đoạn văn bản, và có thể cần truy xuất, xóa hoặc duyệt qua toàn bộ nội dung đã lưu. Với cấu trúc danh sách liên kết, các thao tác thêm, xóa hoặc duyệt Stack có thể thực hiện linh hoạt mà không phải lo lắng về kích thước mảng hay chi phí cấp phát lại bộ nhớ.

* + 1. ***Lớp ClipboardItem – Kiểu dữ liệu lưu trữ bên trong Stack***

Trong chương trình, mỗi nội dung được sao chép vào clipboard không chỉ là một đoạn văn bản đơn thuần, mà còn kèm theo thời điểm mà hành động sao chép diễn ra. Để lưu trữ đồng thời hai thông tin này, cần thiết kế một lớp quy định kiểu dữ liệu lưu vào clipboard – lớp **ClipboardItem.** Lớp này sẽ quy định việcđóng gói nội dung và thời gian sao chép vào một đối tượng duy nhất.

Lớp này bao gồm hai thuộc tính:

* **Content (string):** Lưu trữ nội dung văn bản mà người dùng đã sao chép.
* **Timestamp (DateTime):** Ghi lại thời điểm sao chép, được tự động gán là thời gian hiện tại khi đối tượng được khởi tạo.

Khi khởi tạo ClipboardItem(string content) nhận tham số là chuỗi nội dung cần lưu, sau đó tự động thiết lập Timestamp là thời điểm khởi tạo.

Lớp còn bổ sung phương thức ToString() để định dạng đối tượng ClipboardItem thành một chuỗi để có thể dễ dàng lưu trữ hoặc hiển thị khi lưu danh sách clipboard vào tệp văn bản.

public class ClipboardItem

{

public string Content { get; set; }

public DateTime Timestamp { get; set; }

public ClipboardItem(string content)

{

Content = content;

Timestamp = DateTime.Now;

}

public override string ToString()

{

return $"{Content}|{Timestamp:yyyy-MM-dd HH:mm:ss}";

}

}

* + 1. ***Cài đặt lớp Stack tùy chỉnh (CustomStack<T>)***

Việc cài đặt CustomStack<T> được thực hiện thông qua hai thành phần chính:

* **Lớp Node<T>:** là lớp lồng bên trong dùng để biểu diễn một phần tử trong Stack. Mỗi node gồm hai thuộc tính: data (dữ liệu lưu trữ) và next (trỏ đến phần tử kế tiếp).

private class Node<U>

{

public U data;

public Node<U> next;

}

* **Con trỏ top:** là thuộc tính riêng trong CustomStack<T>, luôn trỏ đến phần tử đầu tiên trong danh sách (tức là phần tử mới nhất được thêm vào Stack). Các thao tác thêm (Push), lấy (Peek), xóa (Pop) đều tương tác với top.

private Node<T> top;

* + 1. ***Các thuật toán trên Stack***

1. *IsEmpty*()

Thuật toán IsEmpty kiểm tra ngăn xếp có rỗng hay không.

* Trả về true nếu top == null, ngược lại trả về false.

public bool IsEmpty()

{

return top == null;

}

1. *Push(T ele)*

Thuật toán Push được sử dụng để thêm một phần tử vào đỉnh của ngăn xếp.

* Tạo một node mới chứa giá trị được truyền vào.
* Gán next của node mới trỏ đến top hiện tại.
* Cập nhật top để trỏ đến node mới.

public void Push(T ele)

{

Node<T> n = new Node<T> { data = ele, next = top };

top = n;

}

1. *Pop()*

Thuật toán Pop dùng để lấy và loại bỏ phần tử đầu tiên (đỉnh) khỏi ngăn xếp.

* Nếu top là null, thông báo Stack rỗng.
* Lưu dữ liệu tại top.
* Cập nhật top sang phần tử kế tiếp.
* Trả về dữ liệu đã lưu.

public T Pop()

{

if (top == null) throw new InvalidOperationException("Stack rỗng!");

T data = top.data;

top = top.next;

return data;

}

1. *Peek()*

Thuật toán Peek trả về phần tử ở đỉnh của stack mà không loại bỏ nó.

* Nếu top rỗng thì thông báo Stack rỗng.
* Trả về phần tử tại top mà không thay đổi Stack

public T Peek()

{

if (top == null) throw new InvalidOperationException("Stack rỗng!");

return top.data;

}

1. *RemoveAt(int index)*

Thuật toán RemoveAt xóa phần tử tại một chỉ số cụ thể trong Stack, tính từ đỉnh (top).

* Nếu index == 0, xóa phần tử đỉnh.
* Duyệt từ đỉnh đến phần tử tại vị trí index (thông qua tăng biến đếm count).
* Gán next của phần tử trước trỏ đến phần tử sau, bỏ qua phần tử cần xóa.

public void RemoveAt(int index)

{

if (index == 0)

{

top = top.next;

return;

}

Node<T> current = top;

Node<T> prev = null;

int count = 0;

while (current != null && count < index)

{

prev = current;

current = current.next;

count++;

}

if (current == null || prev == null)

throw new ArgumentOutOfRangeException(nameof(index), "Vị trí không hợp lệ.");

prev.next = current.next;

}

1. *GetAt(int index)*

Thuật toán GetAt cho phép truy cập và lấy ra phần tử tại một chỉ số cụ thể Stack, tính từ đỉnh top.

* Nếu index < 0, thông báo ngoại lệ
* Duyệt từ đỉnh đến phần tử tại vị trí index (thông qua tăng biến đếm count)
* Nếu count == index, trả về dữ liệu tại nút hiện tại.

public T GetAt(int index)

{

if (index < 0) throw new ArgumentOutOfRangeException(nameof(index));

Node<T> current = top;

int count = 0;

while (current != null)

{

if (count == index)

return current.data;

current = current.next;

count++;

}

throw new ArgumentOutOfRangeException(nameof(index), "Vị trí vượt quá chiều dài của ngăn xếp.");

}

1. Clear()

Thuật toán Clear xóa toàn bộ phần tử trong ngăn xếp.

* Gán top = null.

public void Clear()

{

top = null;

}

1. ForEach(Action<T> action)

Thuật toán ForEach duyệt qua tất cả các phần tử trong Stack và thực thi một hành động với mỗi phần tử.

* Khởi tạo con trỏ current bắt đầu từ đỉnh top của Stack.
* Duyệt qua từng node bằng vòng lặp while (current != null).
* Tại mỗi bước, gọi hàm action(current.data) để thực hiện hành động tùy chỉnh (ví dụ: in ra, ghi vào file, xử lý logic...).
* Chuyển con trỏ sang phần tử kế tiếp bằng current = current.next.

public void ForEach(Action<T> action)

{

Node<T> current = top;

while (current != null)

{

action(current.data);

current = current.next;

}

}

1. SaveToFile()

Thuật toán SaveToFile dùng để lưu toàn bộ nội dung của Stack vào một tệp .txt, đặt tại thư mục Downloads của người dùng.

* Kiểm tra stack có rỗng hay không. Nếu rỗng, hiển thị thông báo và thoát.
* Tạo tên file theo định dạng ClipboardData\_ddMMyyyy\_HHmmss.txt.
* Xác định đường dẫn đến thư mục Downloads của hệ điều hành.
* Dùng StreamWriter để mở file ở chế độ ghi.
* Gọi ForEach để duyệt và ghi từng phần tử vào file theo định dạng ToString().
* Đóng file và hiện thông báo thành công.

**CHƯƠNG 2. PHÂN TÍCH VÀ THIẾT KẾ LỚP**

* 1. **Phân tích và yêu cầu hệ thống**
  2. **Sơ đồ lớp (Class Diagram)**