**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**
     1. ***Mục tiêu của hệ thống***

Mục tiêu của hệ thống là giúp người dùng quản lý các nội dung sao chép và cắt một cách dễ dàng và hiệu quả. Hệ thống sẽ sử dụng cấu trúc dữ liệu ngăn xếp (stack) để lưu trữ các mục clipboard, đảm bảo rằng mỗi mục mới được thêm vào đầu ngăn xếp và các mục sẽ được quản lý theo nguyên tắc LIFO (Last-In, First-Out. Thêm vào đó, hệ thống còn hỗ trợ tính năng lưu trữ lịch sử clipboard vào tệp văn bản (.txt), giúp người dùng có thể lưu lại và dễ dàng truy xuất các mục sao chép trong tương lai. Nhờ đó, người dùng có thể quản lý clipboard một cách linh hoạt và tiết kiệm thời gian hơn trong công việc hàng ngày.

* + 1. ***Các chức năng chính***

Hệ thống sẽ bao gồm một số chức năng chính để đáp ứng các nhu cầu sử dụng của người dùng:

* **Thêm nội dung vào Stack:** Mỗi lần sao chép hoặc cắt nội dung, hệ thống sẽ thêm mục đó vào đầu ngăn xếp, đảm bảo lưu trữ đầy đủ lịch sử clipboard.
* **Hiển thị danh sách các mục đã sao chép/cắt:** Giao diện người dùng sẽ hiển thị các mục trong stack theo thứ tự LIFO, giúp người dùng dễ dàng chọn lại các mục trước đó.
* **Chọn và dán nội dung:** Người dùng có thể chọn lại một mục trong danh sách và dán vào ô nhập liệu hoặc ứng dụng khác.
* **Xóa mục cụ thể khỏi Stack:** Hệ thống cho phép người dùng xóa một mục cụ thể khỏi lịch sử clipboard khi không còn cần thiết.
* **Xóa toàn bộ Stack:** Hệ thống cung cấp chức năng xóa toàn bộ lịch sử clipboard, giúp người dùng làm mới stack khi cần thiết.
* **Lưu lịch sử clipboard vào tệp văn bản**: Hệ thống hỗ trợ lưu toàn bộ lịch sử clipboard vào một tệp tin văn bản với tên tự động dựa trên thời gian lưu.
  + 1. ***Yêu cầu chức năng***

Bảng 1. Yêu cầu chức năng của Clipboard

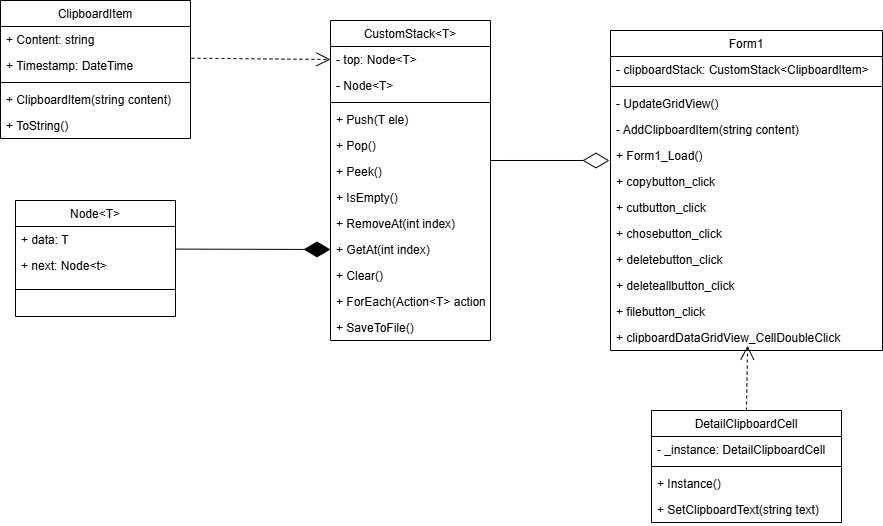
|  |  |
| --- | --- |
| **Mã yêu cầu** | **Mô tả yêu cầu chức năng** |
| RF01 | Hệ thống phải cho phép người dùng nhập nội dung và đẩy vào stack khi thực hiện thao tác sao chép hoặc cắt. |
| RF02 | Hệ thống phải hiển thị tất cả các mục trong stack theo thứ tự LIFO, giúp người dùng dễ dàng truy cập vào mục mới nhất hoặc cũ nhất. |
| RF03 | Hệ thống phải hỗ trợ tính năng double-click để xem chi tiết nội dung của mỗi mục trong stack. |
| RF04 | Hệ thống phải cho phép người dùng xóa một phần tử khỏi stack theo vị trí chọn. |
| RF05 | Hệ thống phải hỗ trợ xóa toàn bộ các mục trong stack. |
| RF06 | Hệ thống phải cung cấp chức năng lưu toàn bộ lịch sử clipboard ra một tệp tin văn bản (.txt). |

* + 1. ***Yêu cầu phi chức năng***

Bảng 2. Yêu cầu phi chức năng của clipboard

|  |  |
| --- | --- |
| **Mã yêu cầu** | **Mô tả yêu cầu chức năng** |
| RF01 | Giao diện hệ thống phải đơn giản, thân thiện và dễ sử dụng, tạo cảm giác dễ chịu cho người dùng khi thao tác. |
| RF02 | Hệ thống phải phản hồi nhanh chóng và không có độ trễ khi thực hiện các thao tác như thêm, xóa hoặc dán nội dung. |
| RF03 | Các tệp lưu trữ lịch sử clipboard phải được đặt tên tự động theo thời gian lưu, giúp người dùng dễ dàng quản lý các tệp lưu trữ. |
| RF04 | Hệ thống phải hiển thị thông báo xác nhận rõ ràng khi thực hiện các thao tác quan trọng như xóa hoặc lưu dữ liệu. |

* 1. **Xác định các lớp và mối quan hệ**
     1. ***Xác định các lớp***
* ClipboardItem: Đại diện cho một mục trong clipboard, chứa nội dung và thời gian sao chép.
* CustomStack<T>: Quản lý cấu trúc Stack theo nguyên tắc LIFO.
* Node<T>: Nút liên kết đơn, được sử dụng bên trong CustomStack<T> để quản lý phần tử trong stack.
* Form1: Giao diện chính của ứng dụng, cho phép người dùng thao tác với clipboard như sao chép, dán, xóa, lưu…
* DetailClipboardCell: Giao diện phụ hiển thị chi tiết nội dung của một mục clipboard được chọn.
  + 1. ***Mối quan hệ giữa các lớp***
* Form1 có thành phần là một CustomStack<ClipboardItem> → quan hệ aggregation.
* CustomStack<T> sử dụng thành phần nội bộ là Node<T> để triển khai stack → quan hệ composition.
* CustomStack<T> tạo và sử dụng ClipboardItem → quan hệ dependency.
* Form1 gọi DetailClipboardCell.Instance để hiển thị nội dung chi tiết → quan hệ dependency.
  1. **Sơ đồ lớp (Class Diagram)**

****

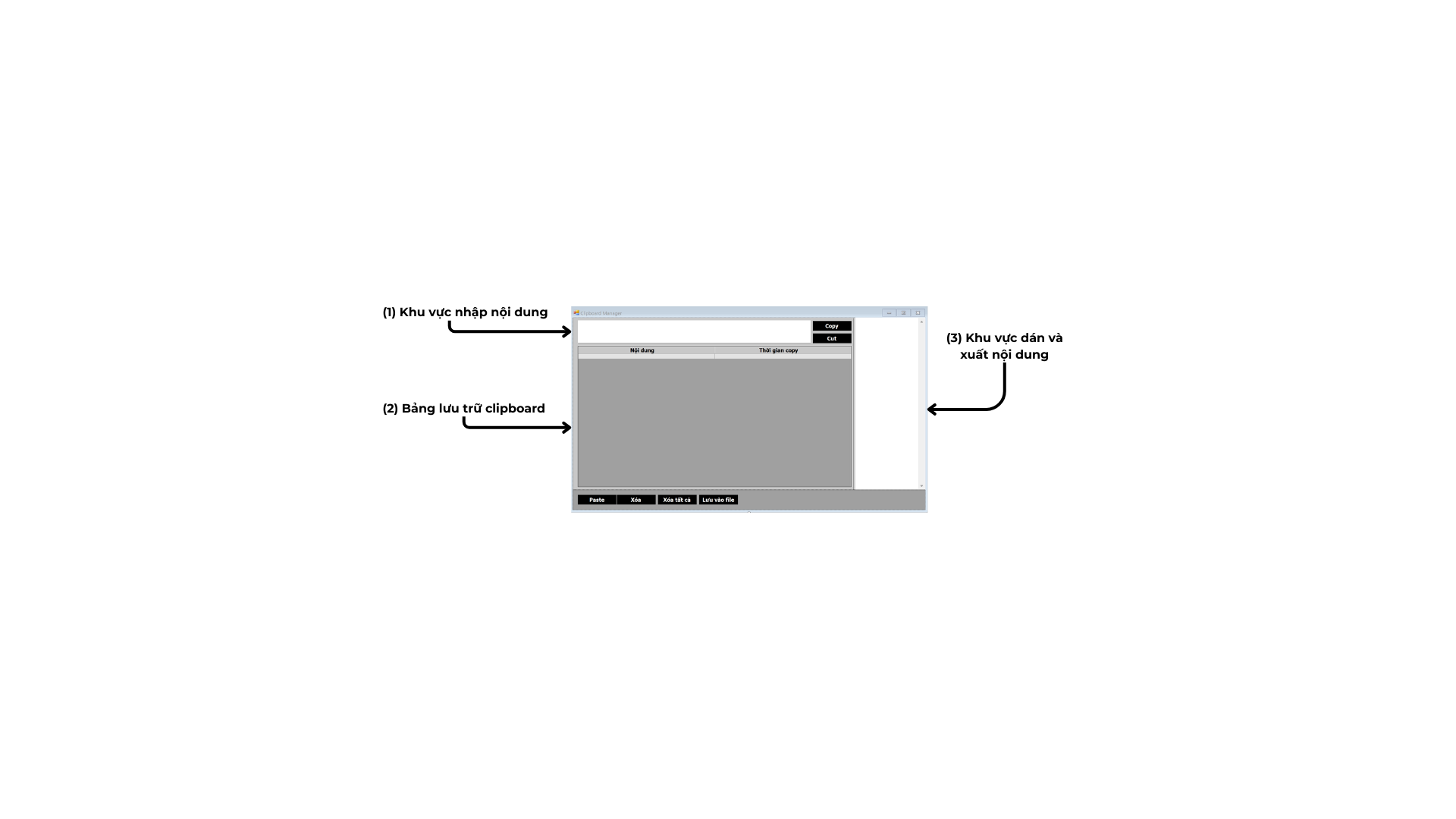
Hình 3. Sơ đồ lớp của hệ thống

**CHƯƠNG 3. THIẾT KẾ GIAO DIỆN**

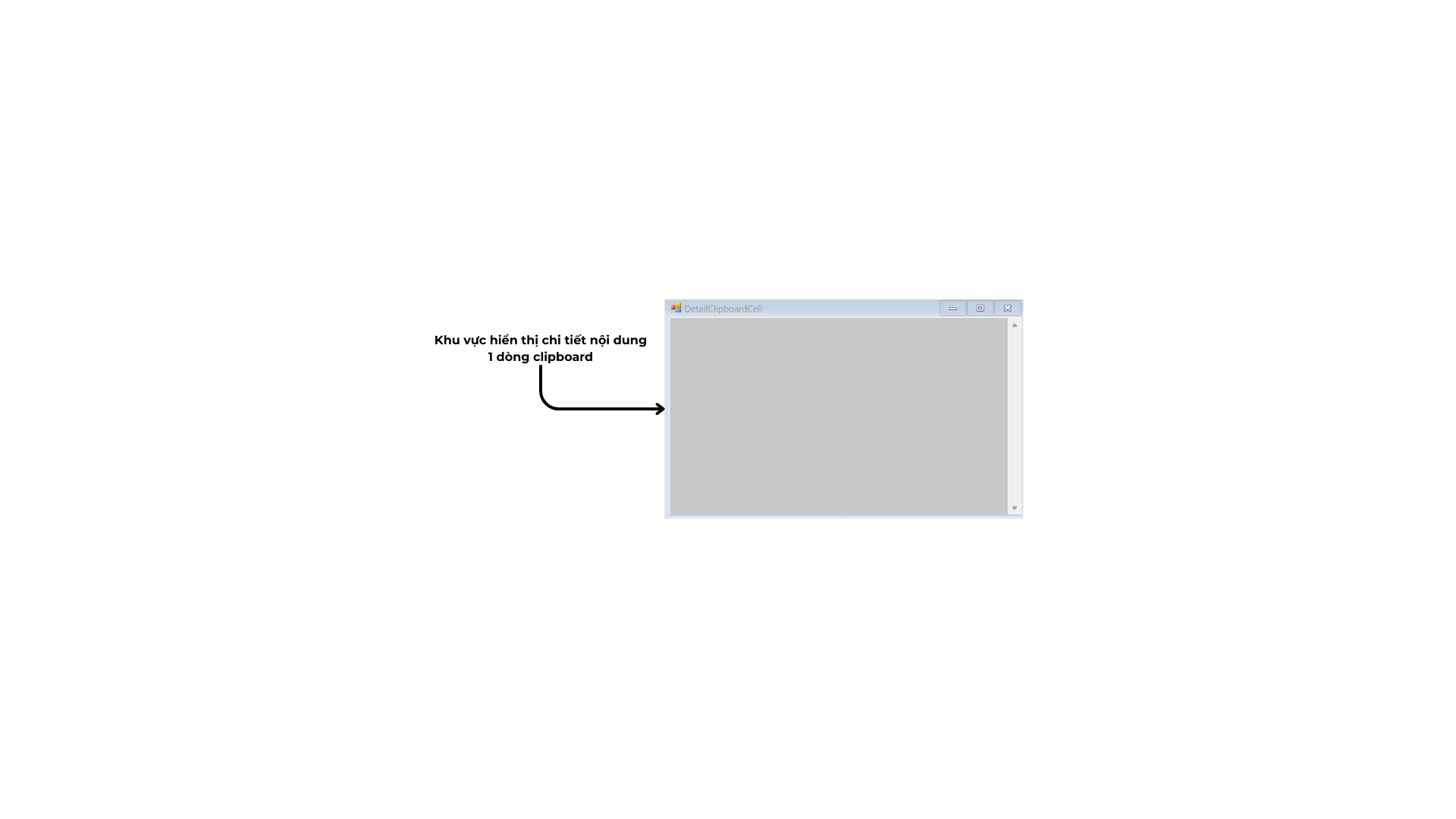
* 1. **Xây dựng giao diện WinForm**

Giao diện người dùng được xây dựng bằng Windows Forms Application trong môi trường C# - Visual Studio. Giao diện được thiết kế đơn giản, trực quan, với bố cục hợp lý, hỗ trợ người dùng dễ dàng thao tác quản lý lịch sử sao chép/dán.

Giao diện chính của hệ thống bao gồm ba khu vực chính:

* (1) Khu vực nhập nội dung (intextBox): Cho phép người dùng nhập liệu, thực hiện các thao tác sao chép hoặc cắt nội dung.
* (2) Bảng lưu trữ clipboard: Dùng để hiển thị lịch sử các nội dung đã được copy/cut, thông qua một DataGridView.
* (3) Khu vực dán và xuất nội dung: Cho phép dán nội dung từ clipboard ra ngoài và thực hiện các chức năng như xóa, lưu.

Hình 4. Giao diện chính của hệ thống

Ngoài ra, ứng dụng còn bao gồm một giao diện phụ được gọi là DetailClipboardCell, dùng để hiển thị chi tiết nội dung clipboard khi người dùng muốn xem toàn bộ đoạn văn bản đã sao chép.

Hình 5. Giao diện phụ DetailClipboardCell

* 1. **Các phương thức hỗ trợ quản lý giao diện và ngăn xếp clipboard**

Để thực hiện các chức năng tương tác trực tiếp với người dùng, ứng dụng Clipboard Manager còn tích hợp một số phương thức nội bộ đóng vai trò hỗ trợ quá trình vận hành hệ thống. Các phương thức này không chỉ đảm bảo dữ liệu trong ngăn xếp clipboard được tổ chức hợp lý mà còn giúp đồng bộ hóa dữ liệu đó với giao diện người dùng thông qua bảng hiển thị (DataGridView).

* + 1. ***UpdateGridView()***

Phương thức này chịu trách nhiệm cập nhật bảng hiển thị clipboardDataGridView mỗi khi có sự thay đổi trong ngăn xếp clipboard. Cụ thể, hàm sẽ xóa toàn bộ dữ liệu đang hiển thị và nạp lại danh sách các mục clipboard hiện hành từ clipboardStack. Đối với mỗi mục, nội dung được rút gọn tối đa 50 ký tự để phù hợp với không gian hiển thị, đồng thời hiển thị thời điểm sao chép nhằm hỗ trợ người dùng theo dõi lịch sử sử dụng.

private void UpdateGridView()

{

clipboardDataGridView.Rows.Clear();

int index = 0;

clipboardStack.ForEach(item =>

{

string preview = item.Content.Length > 50

? item.Content.Substring(0, 50) + "..."

: item.Content;

clipboardDataGridView.Rows.Add(preview, item.Timestamp.ToString("dd/MM/yyyy HH:mm:ss"));

index++;

});

}

* + 1. ***AddClipboardItem(string content)***

Phương thức này thực hiện việc tạo mới một đối tượng ClipboardItem từ chuỗi nội dung đầu vào, sau đó thêm đối tượng này vào ngăn xếp clipboardStack. Ngay sau khi thêm, phương thức gọi UpdateGridView() để đồng bộ giao diện người dùng với dữ liệu nội bộ.

private void AddClipboardItem(string content)

{

ClipboardItem item = new ClipboardItem(content);

clipboardStack.Push(item);

UpdateGridView();

}

* + 1. ***Form1\_Load(object sender, EventArgs e)***

Đây là phương thức khởi tạo được gọi khi biểu mẫu chính của ứng dụng (Form1) được tải lần đầu. Trong phương thức này, UpdateGridView() được gọi để đảm bảo rằng bảng dữ liệu hiển thị trạng thái ban đầu của ngăn xếp clipboard (ngay cả khi ngăn xếp đang rỗng). Điều này giúp giao diện phản ánh đúng dữ liệu hệ thống ngay khi khởi động ứng dụng.

private void Form1\_Load(object sender, EventArgs e)

{

UpdateGridView();

}

* 1. **Chi tiết các chức năng**
     1. ***Sao chép nội dung (Copy)***

Khi người dùng nhập văn bản vào intextBox và nhấn nút Copy, nội dung này sẽ được lưu vào ngăn xếp clipboardStack thông qua hàm AddClipboardItem(), đồng thời cập nhật bảng DataGridView để hiển thị. Nếu nội dung rỗng sẽ không thực hiện thao tác nào.

private void copybutton\_Click(object sender, EventArgs e)

{

string text = intextBox.Text;

if (!string.IsNullOrEmpty(text))

{

AddClipboardItem(text);

}

}

* + 1. ***Cắt nội dung (Cut)***

Chức năng Cut hoạt động tương tự như Copy, nhưng sau khi thêm vào clipboard, nội dung trong intextBox sẽ được xóa.

private void cutbutton\_Click(object sender, EventArgs e)

{

string text = intextBox.Text;

if (!string.IsNullOrEmpty(text))

{

AddClipboardItem(text);

intextBox.Clear();

}

}

* + 1. ***Dán nội dung (Paste)***

Người dùng có thể chọn một dòng trong bảng clipboardDataGridView và nhấn nut “Paste” để dán nội dung tương ứng vào ô văn bản đầu ra outtextBox. Nếu không có dòng nào được chọn, hệ thống sẽ mặc định lấy phần tử đầu tiên trên đỉnh stack (thông qua Peek()). Nội dung được dán kèm dòng mới nếu outtextBox đã có sẵn văn bản.

private void chosebutton\_Click(object sender, EventArgs e)

{

if (clipboardStack.IsEmpty())

{

MessageBox.Show("Clipboard trống!", "Thông báo", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

string selectedText = "";

if (clipboardDataGridView.SelectedRows.Count > 0)

{

int selectedIndex = clipboardDataGridView.SelectedRows[0].Index;

ClipboardItem item = clipboardStack.GetAt(selectedIndex);

selectedText = item.Content;

}

else

{

selectedText = clipboardStack.Peek().Content;

}

if (!string.IsNullOrEmpty(outtextBox.Text))

outtextBox.AppendText(Environment.NewLine);

outtextBox.AppendText(selectedText);

}

* + 1. ***Xóa một mục trong clipboard***

Sau khi chọn một dòng trên bảng, người dùng có thể nhấn nút “Xóa” để loại bỏ phần tử tương ứng ra khỏi stack bằng phương thức RemoveAt(index). Sau thao tác này, bảng sẽ được làm mới lại bằng UpdateGridView() để phản ánh dữ liệu mới.

private void deletebutton\_Click(object sender, EventArgs e)

{

if (clipboardDataGridView.SelectedRows.Count > 0)

{

int selectedIndex = clipboardDataGridView.SelectedRows[0].Index;

clipboardStack.RemoveAt(selectedIndex);

UpdateGridView();

}

}

* + 1. ***Xóa toàn bộ clipboard***

Khi người dùng nhấn nút “Xóa tất cả”, một hộp thoại xác nhận sẽ xuất hiện. Nếu người dùng đồng ý, toàn bộ nội dung của stack sẽ bị xóa bằng Clear() và giao diện sẽ được cập nhật lại.

private void deleteallbutton\_Click(object sender, EventArgs e)

{

DialogResult result = MessageBox.Show

(

"Bạn có chắc chắn muốn xóa tất cả nội dung clipboard?",

"Xác nhận",

MessageBoxButtons.YesNo,

MessageBoxIcon.Warning

);

if (result == DialogResult.Yes)

{

clipboardStack.Clear();

UpdateGridView();

}

}

* + 1. ***Lưu clipboard vào tệp văn bản***

Chức năng lưu được thực hiện qua nút “Lưu vào file”. Toàn bộ phần tử trong stack được ghi xuống một tệp .txt nằm trong thư mục Downloads. File được đặt tên theo định dạng thời gian thực để tránh trùng lặp. Phương thức SaveToFile() chịu trách nhiệm thực hiện chức năng này.

private void filebutton\_Click(object sender, EventArgs e)

{

clipboardStack.SaveToFile();

}

* + 1. ***Hiển thị chi tiết nội dung clipboard***

Người dùng có thể nhấn đúp chuột vào một dòng bất kỳ trong bảng để xem chi tiết toàn bộ nội dung văn bản tại dòng đó. Chức năng này được triển khai trong clipboardDataGridView\_CellDoubleClick, sử dụng phương thức GetAt(index) để lấy phần tử trong stack tương ứng với chỉ số dòng. Cửa sổ DetailClipboardCell sẽ hiển thị nội dung chi tiết đó.

private void clipboardDataGridView\_CellDoubleClick(object sender, DataGridViewCellEventArgs e)

{

if (e.RowIndex >= 0)

{

try

{

ClipboardItem item = clipboardStack.GetAt(e.RowIndex);

string fullContent = item.Content;

DetailClipboardCell.Instance.SetClipboardText(fullContent);

DetailClipboardCell.Instance.ShowDialog();

}

catch (ArgumentOutOfRangeException)

{

MessageBox.Show("Không thể truy cập nội dung clipboard!", "Lỗi", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

Trong giao diện phụ DetailClipboardCell sẽ dùng phương thức SetClipboardText để hiển thị toàn bộ nội dung mà phương thức nhận vào vào ô textBox1

public void SetClipboardText(string text)

{

textBox1.Text = text;

textBox1.SelectionStart = textBox1.Text.Length;

textBox1.SelectionLength = 0;

}

**CHƯƠNG 4. THẢO LUẬN VÀ ĐÁNH GIÁ**

* 1. **Kết quả nhận được**

A screenshot of a computer

AI-generated content may be incorrect.

Hình 6. Giao diện khi nhập dữ liệu

A screenshot of a computer

AI-generated content may be incorrect.

Hình 7. Kết quả khi nhấn nút Copy

A screenshot of a computer

AI-generated content may be incorrect.

Hình 8. Kết quả khi nhấn nút Cut

A screenshot of a computer

AI-generated content may be incorrect.

Hình 9. Kết quả khi nhấn nút Paste cho nội dung dòng 2

A screenshot of a computer

AI-generated content may be incorrect.

Hình 10. Kết quả khi xem chi tiết nội dung clipboard

A screenshot of a computer

AI-generated content may be incorrect.

Hình 11. Kết quả khi nhấn nút Xóa nội dung dòng 2

A screenshot of a computer

AI-generated content may be incorrect.

Hình 12. Kết quả khi nhấn nút Lưu vào file

A screenshot of a computer

AI-generated content may be incorrect.

Hình 13. Tệp clipboard được lưu lại tại mục Downloads

A screenshot of a computer

AI-generated content may be incorrect.

Hình 14. Hộp thoại xuất hiện khi nhấn nút Xóa tất cả

A screenshot of a computer

AI-generated content may be incorrect.

Hình 15. Kết quả khi nhấn nút Xóa tất cả

* 1. **Hạn chế và cải tiến**
     1. ***Thiếu chức năng lưu tự động***
* Hạn chế: Hiện tại, người dùng cần phải thao tác thủ công để lưu nội dung clipboard ra tệp. Nếu ứng dụng bị tắt đột ngột, tất cả dữ liệu chưa được lưu sẽ bị mất. Điều này có thể gây bất tiện khi người dùng quên hoặc không kịp lưu dữ liệu quan trọng.
* Cải tiến: Thêm chức năng lưu tự động mỗi khi có sự thay đổi trong Clipboard, hoặc cho phép người dùng cấu hình chế độ tự động lưu vào tệp sau một khoảng thời gian nhất định.
  + 1. ***Không theo dõi clipboard hệ thống***
* Hạn chế: Ứng dụng không tự động cập nhật các nội dung được người dùng sao chép từ hệ điều hành. Người dùng phải nhập thủ công vào inTextBox để lưu nội dung, điều này làm giảm tính tiện lợi và khả năng tự động hoá.
* Cải tiến: Tích hợp với Clipboard hệ thống để tự động theo dõi và lưu trữ nội dung được sao chép từ bất kỳ phần mềm hoặc cửa sổ nào trên hệ thống mà không cần can thiệp của người dùng.
  + 1. ***Giao diện còn đơn giản***
* Hạn chế: Giao diện người dùng hiện tại khá cơ bản và thiếu các yếu tố hỗ trợ người dùng như biểu tượng minh hoạ trực quan, thao tác qua phím tắt, hoặc chức năng tìm kiếm trong danh sách Clipboard…
* Cải tiến: Nâng cấp giao diện người dùng với các biểu tượng hỗ trợ, thêm phím tắt và tính năng tìm kiếm, giúp người dùng dễ dàng tương tác và tìm kiếm các nội dung trong Clipboard.
  + 1. ***Không giới hạn dung lượng Stack***
* Hạn chế: Hiện tại, không có giới hạn về số lượng mục có thể được lưu trong Stack. Điều này có thể dẫn đến việc bộ nhớ bị chiếm dụng quá mức khi người dùng sao chép quá nhiều nội dung, làm ảnh hưởng đến hiệu suất của ứng dụng.
* Cải tiến: Thêm tính năng giới hạn số lượng mục lưu trữ trong Stack, có thể thiết lập số lượng tối đa phần tử hoặc tự động xoá các mục cũ khi Stack đầy.
  + 1. ***Hạn chế trong việc lưu và quản lý các loại tệp đa phương tiện***
* Hạn chế: Ứng dụng hiện tại chỉ hỗ trợ lưu trữ và quản lý nội dung Clipboard dưới dạng văn bản thuần túy. Điều này có nghĩa là nếu người dùng sao chép các loại dữ liệu khác như hình ảnh, video, âm thanh hay các loại tệp khác vào Clipboard, ứng dụng không thể lưu hoặc hiển thị những dữ liệu này. Việc thiếu hỗ trợ cho các loại tệp đa phương tiện làm hạn chế khả năng sử dụng của ứng dụng trong các trường hợp người dùng sao chép và cần quản lý các tệp không phải văn bản.
* Cải tiến: Cập nhật ứng dụng để hỗ trợ lưu trữ và quản lý nhiều loại tệp khác nhau, bao gồm hình ảnh, video, âm thanh và các tệp dữ liệu khác từ Clipboard. Điều này có thể bao gồm việc kiểm tra và phân loại dữ liệu sao chép, từ đó lưu trữ chúng dưới dạng các đối tượng phù hợp (ví dụ: Image cho hình ảnh, Stream cho âm thanh hoặc video). Giao diện người dùng cần được cập nhật để hỗ trợ hiển thị thumbnail cho hình ảnh, video hoặc âm thanh...
  1. **Hướng phát triển**
     1. ***Hướng phát triển cho CustomStack***
* Quản lý dung lượng Stack: Thêm tính năng giới hạn số lượng mục trong Stack để tránh chiếm dụng quá nhiều bộ nhớ.
* Tính năng undo/redo: Cung cấp khả năng hoàn tác hoặc tái thực hiện các thao tác với các mục trong Stack.
* Hỗ trợ đa dạng dữ liệu: Mở rộng khả năng lưu trữ Stack để hỗ trợ các tệp đa phương tiện như hình ảnh, video và âm thanh.
* Tìm kiếm và lọc dữ liệu: Cung cấp tính năng tìm kiếm và lọc các mục trong Stack theo các tiêu chí như ngày tháng, loại tệp.
  + 1. ***Hướng phát triển cho ứng dụng clipboard***
* Tăng cường khả năng đa nền tảng (Cross-Platform Support): Phát triển ứng dụng để chạy trên nhiều hệ điều hành, giúp người dùng dễ dàng quản lý clipboard trên các thiết bị khác nhau.
* Tính năng đồng bộ dữ liệu: Cung cấp tính năng đồng bộ hóa dữ liệu clipboard giữa các thiết bị, cho phép người dùng truy cập nội dung clipboard từ bất kỳ đâu.
* Bảo mật và quyền riêng tư: Cải thiện bảo mật dữ liệu sao chép với mã hóa và xác thực người dùng, bảo vệ thông tin nhạy cảm trong clipboard.

**PHỤC LỤC**

1. Đoạn mã nguồn chính (github): <https://github.com/tnbchau/Clipboard>
2. Tài liệu hướng dẫn cài đặt và sử dụng
   1. **Phần mềm cần thiết**
   * Visual Studio
   * .NET Framework
   1. **Hướng dẫn cài đặt**

**Bước 1. Mở Project**

1. Mở Visual Studio.
2. Vào File > Open > Project/Solution.
3. Chọn file Clipboard.sln

**Bước 2. Kiểm tra giao diện**

* + Form1.cs: giao diện chính
  + DetailClipboardCell.cs: giao diện chi tiết clipboard.

**Bước 3. Chạy chương trình**

1. Nhấn F5 hoặc chọn Debug > Start Debugging.
2. Giao diện chính của ứng dụng sẽ xuất hiện.
   1. **Hướng dẫn sử dụng**
3. Copy: Nhập nội dung vào ô Input (intextBox) → Nhấn Copy → Dữ liệu được lưu vào Stack và hiển thị trong bảng.
4. Cut: Giống như Copy, nhưng sẽ xóa nội dung khỏi ô Input sau khi lưu vào Stack.
5. Paste: Chọn một dòng trong bảng hoặc không chọn để dán dòng mới nhất. Nội dung sẽ được dán vào ô Output (outtextBox).
6. Xem chi tiết: Nhấp đúp vào một dòng trong bảng để xem toàn bộ nội dung.
7. Xóa: Chọn một dòng trong bảng → Nhấn nút Xóa để xóa dòng đó khỏi Stack.
8. Xóa tất cả: Nhấn nút Xóa tất cả → Xác nhận để xóa toàn bộ dữ liệu clipboard trong Stack.
9. Lưu vào tệp: Nhấn Save to File → Nội dung clipboard sẽ được ghi vào file .txt trong thư mục Downloads, tên file có dạng ClipboardData\_ddMMyyyy\_HHmmss.txt.
   1. **Ghi chú**
   * Ứng dụng không can thiệp trực tiếp vào clipboard hệ thống.
   * Mỗi lần chạy ứng dụng là một clipboard session riêng biệt.
   * File lưu sẽ nằm tại: C:\Users\<Tên người dùng>\Downloads\
10. Phân công công việc

|  |  |  |
| --- | --- | --- |
| **Thành viên** | **Mã số sinh viên** | **Nhiệm vụ** |
| Thái Ngọc Bảo Châu | 31231023230 | - Tổng hợp file word, code  - Thiết kế giao diện  - Viết phương thức RemoveAt, GetAt, Clear của CustomStack  - Viết phương thức UpdateGridView, AddClipboardItem copybutton\_Click, cutbutton\_Click  - Soạn nội dung Chương 3 - Soạn nội dung Tài liệu hướng dẫn cài đặt và sử dụng |
| Phan Nhựt Đăng Khoa | 31241022794 | - Soạn slide  - Viết lớp ClipboardItem  - Viết phương thức clipboardDataGridView\_CellDoubleClick  - Viết lớp DetailClipboardCell : Form  - Soạn nội dung Chương 1 |
| Trần Xuân Ngọc | 31241027849 | -Soạn slide  - Viết phương thức ForEach(Action<T> action), SaveToFile() của CustomStack  - Viết phương thức filebutton\_Click  - Soạn nội dung Chương 4 |
| Mai Đức Phát | 31221021213 | - Soạn slide  - Viết phương thức IsEmpty, Push, Pop, Peek của CustomStack  - Viết phương thức chosebutton\_Click, deletebutton\_Click, deleteallbutton\_Click  - Soạn nội dung Chương 2 |