ĐẠI HỌC UEH
TRƯỜNG CÔNG NGHỆ VÀ THIẾT KẾ
KHOA CÔNG NGHỆ THÔNG TIN KINH DOANH

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Đồ án môn học

ỨNG DỤNG STACK TRONG QUẢN LÝ BỘ ĐỆM

Chuyên ngành: Kỹ thuật phần mềm - Khóa: 48

Advisor: PhD. Đặng Ngọc Hoàng Thành
Group 10: Đặng Đại Lợi
Lê Hoàng Khang
Châu Thuận Đức
Từ Gia Bảo
Nguyễn Nam Thái

Ho Chi Minh City, December 2023

Contents

1 NGĂN XẾP STACK VÀ CLIPBOARD

1.1 Các khái niệm liên quan

1.1.1 Stack

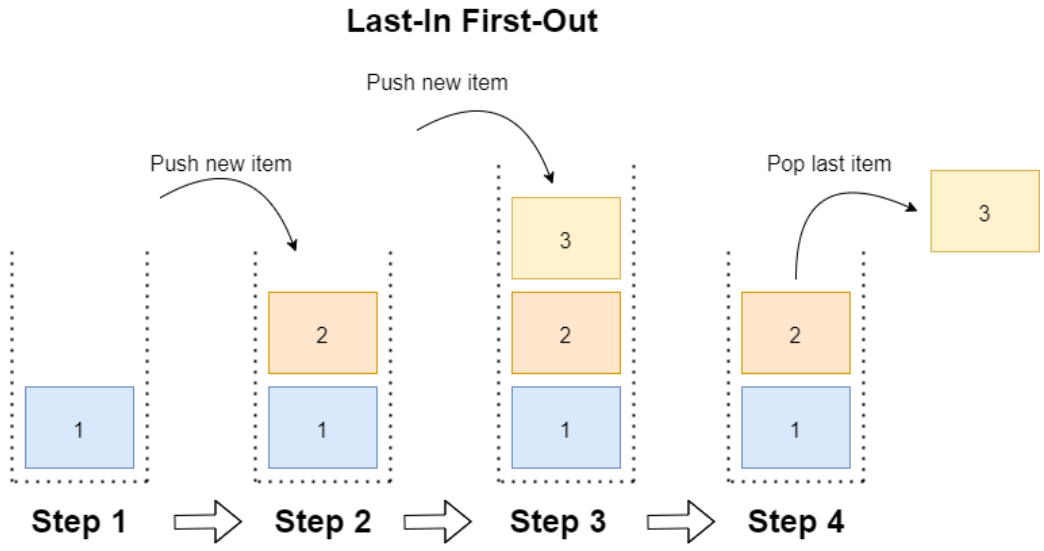
Ngăn xếp, còn được gọi là bộ xếp chồng, là một cấu trúc dữ liệu trừu tượng tuân theo nguyên tắc "vào sau ra trước" (Last In First Out - LIFO). Theo nguyên tắc này, phần tử cuối cùng được thêm vào ngăn xếp sẽ trở thành phần tử đầu tiên được lấy ra khi cần. Mô hình này có thể được mô tả một cách đơn giản như việc xếp sách lên nhau, với sách mới được đặt lên trên cùng và sách nào được đặt sau cùng sẽ được lấy ra trước.

Ngăn xếp không chỉ là một khái niệm trừu tượng, mà còn là một công cụ quan trọng trong lập trình và các ứng dụng thực tế. Ví dụ, trong quản lý lời gọi hàm của chương trình máy tính, ngăn xếp giúp theo dõi thứ tự của các hàm được gọi và đảm bảo rằng hàm mới nhất được thực thi trước. Trong trình duyệt web, ngăn xếp có thể được sử dụng để lưu trữ lịch sử của các trang web đã được duyệt, cho phép người dùng quay lại các trang trước đó một cách dễ dàng.

Ngăn xếp không chỉ được sử dụng trong lập trình máy tính, mà còn xuất hiện trong nhiều lĩnh vực khác nhau. Ví dụ, trong quản lý lời gọi hàm của một chương trình, ngăn xếp giúp theo dõi thứ tự thực hiện các hàm và trở về từ chúng. Trong trình duyệt web, ngăn xếp có thể được sử dụng để theo dõi lịch sử duyệt web, cho phép người dùng quay lại các trang đã xem trước đó.

Ngoài ra, ngăn xếp còn là một phần quan trọng của nhiều thuật toán, như thuật toán duyệt đồ thị theo chiều sâu (DFS), nơi nó giúp theo dõi các đỉnh của đồ thị theo cách có thứ tự và có tổ chức. Điều này làm cho ngăn xếp trở thành một công cụ linh hoạt và quan trọng trong lĩnh vực khoa học máy tính và lập trình.

Tóm lại, ngăn xếp không chỉ là một khái niệm lý thuyết mà còn là một công cụ quan trọng hỗ trợ trong nhiều lĩnh vực khác nhau của khoa học máy tính và lập trình.



Hình 1: Stack With Steps

Trong C#, để sử dụng Stack ta dùng lớp Stack trong thư viện Collections

```
using System.Collections;

class Example
{
  static void Main()
```

```
  {
    Stack s = new Stack();
  }
}
```

1.1.2 Bộ đệm Clipboard

Clipboard, hay còn được gọi là bộ nhớ đệm, đại diện cho một phần quan trọng của hệ thống lưu trữ tạm thời trên thiết bị điện tử. Trên Clipboard, dữ liệu có thể được lưu giữ ở nhiều định dạng khác nhau như đoạn văn bản, hình ảnh, tập tin hay file, và chúng thường được người dùng sao chép (Copy) hoặc cắt (Cut) từ một nguồn và sau đó được dán (Paste) ở một vị trí khác.

Nguyên tắc hoạt động của Clipboard là tạm thời giữ lại dữ liệu sau khi người dùng thực hiện các thao tác Copy hoặc Cut. Trong quá trình này, dữ liệu được lưu trữ tại vị trí tạm thời, chờ đợi để được sử dụng lại. Điều này giúp người dùng dễ dàng di chuyển thông tin từ một ứng dụng hoặc vị trí sang một nơi khác mà không cần phải nhập lại dữ liệu từ đầu.

Clipboard chủ yếu đóng vai trò quan trọng trong các hoạt động như sao chép và dán trong các ứng dụng văn bản, biên tập hình ảnh, và nhiều ứng dụng khác. Nó là một công cụ tiện ích giúp tăng cường tính tương tác và hiệu suất của người dùng trên các thiết bị điện tử.

1.2 Cấu trúc và cài đặt Stack

Trong việc cài đặt ngăn xếp, có nhiều phương pháp khác nhau và một trong những cách phổ biến là sử dụng mảng. Dưới đây là một phần nhỏ về cài đặt ngăn xếp bằng mảng:

1.2.1 Cài đặt bằng mảng

Ưu điểm:

- Dễ thực hiện: Việc sử dụng mảng làm cấu trúc dữ liệu cho ngăn xếp thường dễ hiểu và triển khai.
- Bộ nhớ được lưu: Do không liên quan đến con trỏ, cài đặt này không yêu cầu quản lý con trỏ, giảm rủi ro lỗi.

Nhược điểm:

- Không năng động: Mặc dù việc thêm và lấy phần tử từ ngăn xếp là nhanh chóng với mảng, nhưng kích thước của mảng phải được định trước và không thể thay đổi linh hoạt trong quá trình chạy.
- Không phát triển và thu nhỏ: Kích thước của ngăn xếp sẽ không thay đổi theo nhu cầu, điều này có thể dẫn đến lãng phí bộ nhớ nếu không tính toán kích thước mảng đúng cách.

Cài đặt bằng mảng thường được ưa chuộng trong các trường hợp đơn giản và nơi mà kích thước của ngăn xếp có thể được dự đoán trước. Tuy nhiên, với yêu cầu linh hoạt cao, cũng như khi cần phát triển và thu nhỏ động, các cách tiếp cận sử dụng danh sách liên kết có thể được xem xét.

```
using System;
class Stack
{
    private const int MAX_SIZE = 100;
    private int top;
    private int[] arr;
```

```
7
8       public Stack()
9       {
10          top = -1;
11          arr = new int[MAX_SIZE];
12      }
13
14      public void Push(int value)
15      {
16          if (top < MAX_SIZE - 1)
17          {
18              arr[++top] = value;
19              Console.WriteLine($"Đã thêm {value} vào ngăn xếp.");
20          }
21          else
22          {
23              Console.WriteLine("Ngăn xếp đầy. Không thể thêm phần tử mới.");
24          }
25      }
26
27      public void Pop()
28      {
29          if (top >= 0)
30          {
31              Console.WriteLine($"Đã lấy ra phần tử {arr[top--]} khỏi ngăn xếp.");
32          }
33          else
34          {
35              Console.WriteLine("Ngăn xếp rỗng. Không thể lấy ra phần tử.");
36          }
37      }
38
39      public void Display()
40      {
41          if (top >= 0)
42          {
43              Console.Write("Nội dung của ngăn xếp: ");
44              for (int i = 0; i <= top; i++)
45              {
46                  Console.Write(arr[i] + " ");
47              }
48              Console.WriteLine();
49          }
50          else
51          {
52              Console.WriteLine("Ngăn xếp rỗng.");
53          }
54      }
55  }
56
57  class Program
58  {
59      static void Main()
```

```
    {
        Stack myStack = new Stack();

        myStack.Push(10);
        myStack.Push(20);
        myStack.Push(30);

        myStack.Display();

        myStack.Pop();
        myStack.Display();
    }
}
```

1.2.2 Cài đặt bằng danh sách liên kết

Khi cài đặt ngăn xếp bằng danh sách liên kết, chúng ta sử dụng một cấu trúc dữ liệu linh hoạt, cho phép thêm hoặc xóa phần tử một cách linh hoạt. Dưới đây là một số đặc điểm của cài đặt này:

Ưu điểm:

- Phát triển và thu nhỏ theo nhu cầu: Danh sách liên kết cho phép kích thước của ngăn xếp phát triển và thu nhỏ linh hoạt tùy thuộc vào nhu cầu trong quá trình chạy ứng dụng.
- Linht hoạt: Các thao tác thêm và xóa phần tử có độ phức tạp thấp và có thể thực hiện một cách linh hoạt mà không làm thay đổi cấu trúc ngăn xếp.

Nhược điểm:

- Yêu cầu thêm bộ nhớ: Sự tham gia của con trỏ trong danh sách liên kết có thể tăng yêu cầu về bộ nhớ so với cài đặt bằng mảng.
- Hiệu suất thấp hơn trong một số trường hợp: Trong một số tình huống, danh sách liên kết có thể có hiệu suất thấp hơn so với mảng do overhead của con trỏ và không gian bộ nhớ phụ trợ.

```
using System;

class Node
{
    public int Data;
    public Node Next;

    public Node(int data)
    {
        Data = data;
        Next = null;
    }
}

class LinkedStack
{
    private Node top;

```

```
    public LinkedStack()
    {
        top = null;
    }

    public void Push(int value)
    {
        Node newNode = new Node(value);
        newNode.Next = top;
        top = newNode;
        Console.WriteLine($"Đã thêm {value} vào ngăn xếp.");
    }

    public void Pop()
    {
        if (top != null)
        {
            Console.WriteLine($"Đã lấy ra phần tử {top.Data} khỏi ngăn xếp.");
            top = top.Next;
        }
        else
        {
            Console.WriteLine("Ngăn xếp rỗng. Không thể lấy ra phần tử.");
        }
    }

    public void Display()
    {
        Node current = top;

        if (current != null)
        {
            Console.Write("Nội dung của ngăn xếp: ");
            while (current != null)
            {
                Console.Write(current.Data + " ");
                current = current.Next;
            }
            Console.WriteLine();
        }
        else
        {
            Console.WriteLine("Ngăn xếp rỗng.");
        }
    }
}

class Program
{
    static void Main()
    {
        LinkedStack myStack = new LinkedStack();
```

```
71
72          myStack.Push(10);
73          myStack.Push(20);
74          myStack.Push(30);
75
76          myStack.Display();
77
78          myStack.Pop();
79          myStack.Display();
80      }
81  }
```

1.3 Các Thuật Toán Trên Stack

Các thuật toán trên ngăn xếp thường có độ phức tạp thời gian là O(1), điều này làm cho ngăn xếp trở thành một cấu trúc dữ liệu hiệu quả cho việc quản lý dữ liệu theo nguyên tắc "vào sau ra trước". Các hàm cơ bản thường được thực hiện trên ngăn xếp bao gồm push(), pop(), isEmpty(), và peek(). Dưới đây là mô tả chi tiết về độ phức tạp và chức năng của từng hàm:

Hoạt động	Mô tả
Pop	Xóa bỏ phần tử đầu tiên ở đỉnh của ngăn xếp, số phần tử của ngăn xếp giảm đi 1.
Peek	Lấy giá trị của phần tử đầu tiên ở đỉnh của ngăn xếp, số phần tử của ngăn xếp không thay đổi.
Push	Thêm một phần tử vào đỉnh của ngăn xếp, số phần tử của ngăn xếp tăng lên 1.
IsEmpty	Kiểm tra ngăn xếp trống hay không. Ngăn xếp trống là ngăn xếp không có phần tử nào.
Size	Lấy số lượng phần tử stack đang có.

Bảng 1: Mô tả các hoạt động của ngăn xếp

1.3.1 Thuật Toán Push

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

- Tạo một node mới với giá trị được nhập vào.
- Đặt node tiếp theo của node mới là top.
- Cuối cùng, tăng top để trỏ đến node mới.

```
1  public void Push(object ele)
2  {
3      Node n = new Node();
4      n.data = ele;
5      n.next = top;
6      top = n;
7  }
```

1.3.2 Thuật Toán Pop

Thuật toán pop được sử dụng để lấy một phần tử ra khỏi đỉnh của ngăn xếp.

- Chỉ thực hiện Pop khi ngăn xếp không trống.
- Kiểm tra xem nút đỉnh trỏ đến có rỗng không.
- Gán đỉnh của danh sách vào nút tiếp theo (node kế tiếp).

```
public Node Pop()
{
    if (top == null)
    {
        return null;
    }
    Node d = top;
    top = top.next;
    return d;
}
```

1.3.3 Thuật Toán IsEmpty

Thuật toán isEmpty được sử dụng để kiểm tra xem ngăn xếp có rỗng hay không.

```
public bool IsEmpty()
{
    return top == null;
}
```

2 PHÂN TÍCH VÀ THIẾT KẾ LỚP

2.1 Phân Tích Bài Toán

Xây dựng bộ quản lý clipboard với các chức năng: copy và paste (dựa trên Stack).
Tạo một lớp generic ClipboardRB<T>, trong lớp ClipboardRB sẽ gọi ra lớp Stack<T> để khởi tạo nên một ngăn xếp bao gồm các hàm Add, Paste, ClearStack, ClearInput.

```
public void Add(string input, string selected)
```

Là phương thức public void (để gọi biến từ bên ngoài) với biến đầu vào là phần văn bản nhập vào và phần văn bản được chọn bằng con trỏ chuột khi nhập vào.

- Khi không có văn bản được chọn, thì sẽ đẩy vào Stack phần văn bản được nhập thô từ bàn phím.
- Và ngược lại, các văn bản được được bằng con trỏ chuột sẽ được đẩy vào Stack

```
public void Add(string input, string selected)
{
    if (selected == "")
    {
        ms.Push(input);
    }
    else
    {
        ms.Push(selected);
    }
}
```

```
public string Paste(string result)
```

Là phương thức public string gán giá trị từ Stack ra ngoài với biến đầu vào là chuỗi result và cũng trả lại giá trị result đó.

- Điều kiện để phương thức gán hoạt động là trong Stack luôn chứa giá trị được đẩy vào.
- Khởi tạo một chuỗi chứa giá trị của Stack đưa ra và gán vào chuỗi result trong vòng lặp chứa điều kiện trên.
- Khởi tạo một chuỗi chứa giá trị của Stack đưa ra và gán vào chuỗi result trong vòng lặp chứa điều kiện trên

```
public string Paste(string result)
{
    while (ms.Count != 0)
    {
        string str = ms.Pop().ToString();
        result += str;
    }
```

```
    return result;
}
```

```
public bool ClearStack()
```

Là phương thức dùng để xóa các giá trị trong Stack khi gọi từ bên ngoài và trả về giá trị bool (true or false)

```
public bool ClearStack()
{
    ms.Clear();
    return true;
}
```

```
public string ClearInput(string input)
```

Là phương thức dùng để xóa các chuỗi trong ô văn bản (textbox)

```
public string ClearInput(string input)
{
    input = string.Empty;
    return input;
}
```

```
ClipboardRB<object> cb = new ClipboardRB<object>();
// CODE GIAO DIỆN
```

Khi chạy form sẽ focus vào ô nhập

```
private void fCOPY_Load(object sender, EventArgs e)
{
    textBox1.Focus();
}
private void fCOPY_Shown(object sender, EventArgs e)
{
    textBox1.Focus();
}
```

```
COPY
```

Khi nhấn vào nút COPY thì sẽ thêm vào stack giá trị trong Textbox1 (một là văn bản thô, hai là văn bản được select).

```
private void button1_Click(object sender, EventArgs e)
{
```

```
    cb.ClearStack();
    cb.Add(textBox1.Text, textBox1.SelectedText);
}
```

Khi nhấn vào nút PASTE sẽ Pop giá trị trong Textbox thứ nhất (một là văn bản thô, hai là văn bản được select) và gán chúng vào Textbox thứ hai tuần tự với số lần nhấn nút, sau đó sẽ lại thêm vào Stack giá trị trong ô Textbox đầu tiên. Nếu là văn bản thô thì sẽ giữ nguyên phần văn bản đó và nếu là văn bản được select thì cũng sẽ giữ nguyên văn bản được select trong Textbox thứ nhất.

```
private void button2_Click(object sender, EventArgs e)
{
    textBox2.Text = cb.Paste(textBox2.Text);
    cb.Add(textBox1.Text, textBox1.SelectedText);
    textBox1.Focus();
}
```

Nút RESET sẽ xóa các nội dung trong ô Textbox 1 và 2 và cũng đồng thời xóa những giá trị trong Stack.

```
private void button3_Click(object sender, EventArgs e)
{
    textBox1.Text = cb.ClearInput(textBox1.Text);
    textBox2.Text = cb.ClearInput(textBox2.Text);
    cb.ClearStack();
}
```

```
CUT
```

Nút CUT có chức năng thêm vào stack giá trị trong Textbox1 (một là văn bản thô, hai là văn bản được select).

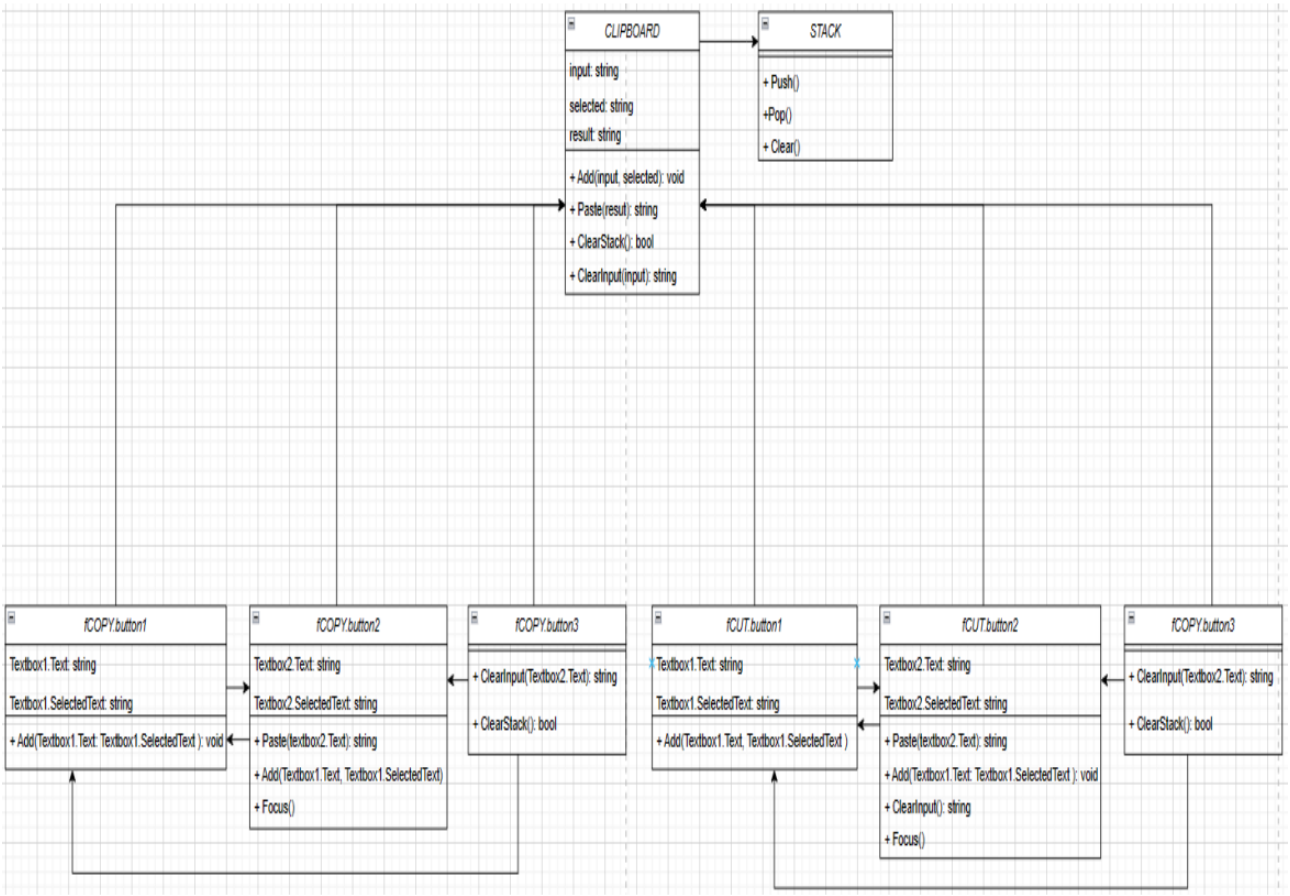
```
private void button1_Click(object sender, EventArgs e)
{
    cb.Add(textBox1.Text, textBox1.SelectedText);
}
```

Nút PASTE sẽ Pop giá trị trong Stack (một là văn bản thô, hai là văn bản được select) và gán vào Textbox thứ hai. Nếu là một văn bản thô thì sẽ xóa toàn bộ nội dung trong Textbox1 và ngược lại nếu là văn bản được select thì sẽ chỉ xóa phần văn bản được select và giữ lại phần văn bản thô.

```
private void button2_Click(object sender, EventArgs e)
{
    textBox2.Text = cb.Paste(textBox2.Text);
    if (textBox1.SelectedText == cb.ClearInput(textBox1.SelectedText))
    {
        textBox1.Text = cb.ClearInput(textBox1.Text);
    }
    textBox1.SelectedText = cb.ClearInput(textBox1.SelectedText);
    textBox1.Focus();
}
```

Nút RESET sẽ xóa các nội dung trong ô Textbox 1 và 2 và cũng đồng thời xóa những giá trị trong Stack.

```
private void button3_Click(object sender, EventArgs e)
{
    textBox1.Text = cb.ClearInput(textBox1.Text);
    textBox2.Text = cb.ClearInput(textBox2.Text);
    cb.ClearStack();
}
```



Hình 2: Sơ đồ lớp

2.2 Cài Đặt Lớp

```
using System.Collections.Generic;

public class ClipboardRB<T>
{
    private Stack<object> ms;
    public ClipboardRB()
    {
        ms = new Stack<object>();
    }
    public void Add(string input, string selected)
```

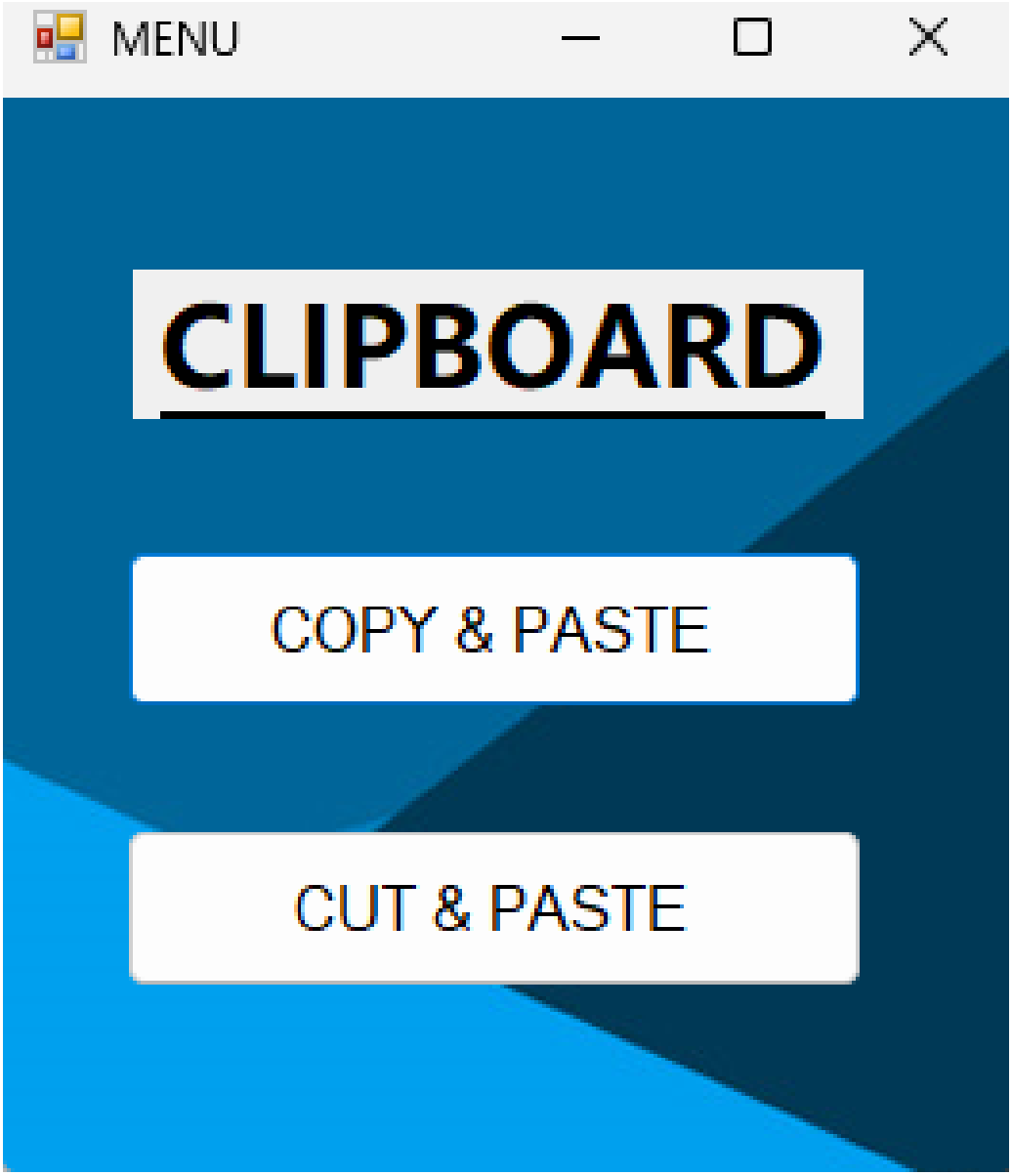
```
    {
        if (selected == string.Empty)
        {
            ms.Push(input);
        }
        else
        {
            ms.Push(selected);
        }
    }
    public string Paste(string result)
    {
        while (ms.Count != 0)
        {
            string str = ms.Pop().ToString();
            result += str;
        }
        return result;
    }
    public bool ClearStack()
    {
        ms.Clear();
        return true;
    }

public string ClearInput(string input)
    {
        input = string.Empty;
        return input;
    }
}
```

3 THIẾT KẾ GIAO DIỆN

3.1 Giao Diện Menu Chính

Menu chọn các chức năng thao tác ứng dụng stack trong quản lí bộ đệm

3.1.1 Menu chọn chức năng

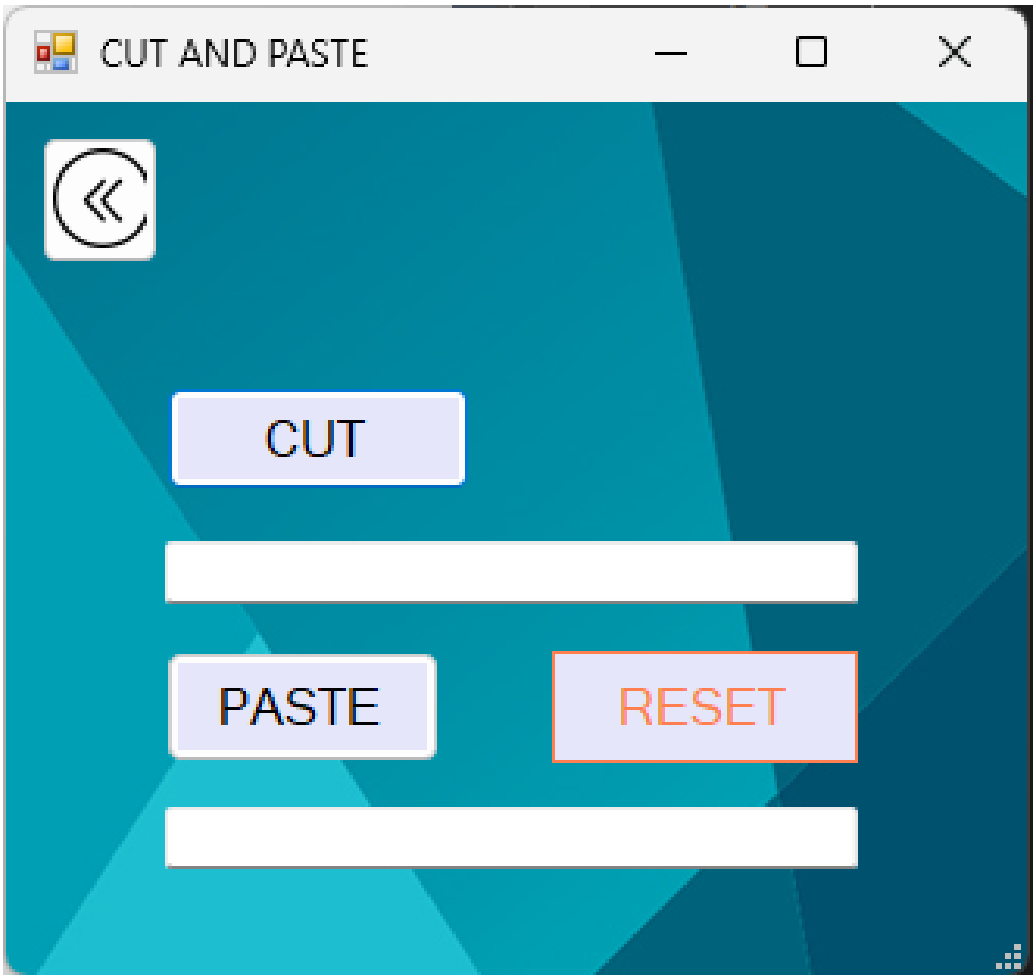


Hình 3: Menu clipboard

3.1.1.1 Clipboard Copy and Paste

Hình 4: Clipboard Copy and Paste

3.1.2 Clipboard Cut and Paste



Hình 5: Clipboard Cut and

3.2 Chi tiết các chức năng

Ứng dụng stack trong quản lí bộ đệm được nghiên cứu và xây dựng thành 2 chức năng chính:

- **Chức năng 1:** Clipboard Copy and Paste
- **Chức năng 2:** Clipboard Cut and Paste

3.2.1 Mục đích chính của các chức năng

Chức năng Clipboard Copy and Paste :

- Nút "Sao chép" (button1) sao chép văn bản từ textBox1 và lưu vào Clipboard.
- Nút "Dán" (button2) dán văn bản từ Clipboard vào textBox2.
- Nút "Xóa" (button3) xóa nội dung trong cả hai ô nhập liệu và trong Clipboard.
- Nút "Đóng" (button4) đóng ứng dụng.

Thực hiện các thao tác Copy và Paste sử dụng ngăn xếp.

Chức năng sao chép và dán văn bản giữa hai ô nhập liệu (textBox1 và textBox2) bằng cách sử dụng một cấu trúc dữ liệu ngăn xếp (ClipboardRB).

Khi nhấn vào nút "Copy" (button1), văn bản từ textBox1 được sao chép và lưu vào ngăn xếp (ClearStack sau đó Add).

Khi nhấn vào nút "Paste" (button2), văn bản từ ngăn xếp được dán vào textBox2, đồng thời văn bản từ textBox1 cũng được thêm vào ngăn xếp.

Nút "Clear" (button3) xóa nội dung của cả hai ô nhập liệu và xóa dữ liệu trong ngăn xếp.

Nút "Close" (button4) đóng ứng dụng.

```
namespace fCOPY
{
    public partial class fCOPY : Form
    {
        public fCOPY()
        {
            InitializeComponent();
        }

        CLIPBOARDRB.ClipboardRB<object> cb = new CLIPBOARDRB.ClipboardRB<object>();

        private void fCOPY_Load(object sender, EventArgs e)
        {
            textBox1.Focus();
        }
        private void fCOPY_Shown(object sender, EventArgs e)
        {
            textBox1.Focus();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            cb.ClearStack();
            cb.Add(textBox1.Text, textBox1.SelectedText);
        }
        private void button2_Click(object sender, EventArgs e)
```

```
        {
            textBox2.Text = cb.Paste(textBox2.Text);
            cb.Add(textBox1.Text, textBox1.SelectedText);
            textBox1.Focus();
        }

        private void button3_Click(object sender, EventArgs e)
        {
            textBox1.Clear();
            textBox2.Clear();
            cb.ClearStack();
        }

        private void textBox1_TextChanged(object sender, EventArgs e)
        {

        }

        private void textBox2_TextChanged(object sender, EventArgs e)
        {

        }

        private void button4_Click(object sender, EventArgs e)
        {
            this.Close();
        }
    }
}
```

Chức năng Clipboard Cut and Paste :

- Nút "Sao chép" (button1) thực hiện việc lưu trữ văn bản từ textBox1 vào Clipboard.
- Nút "Dán" (button2) dán văn bản từ Clipboard vào textBox2. Nếu văn bản đã được chọn trong textBox1, nó sẽ được xóa khỏi textBox1 sau khi được dán vào textBox2.
- Nút "Xóa" (button3) xóa nội dung của cả hai ô nhập liệu và xóa dữ liệu trong Clipboard.
- Nút "Đóng" (button4) đóng ứng dụng.

Thực hiện các thao tác Cut và Paste sử dụng ngăn xếp.

Thao tác cắt và dán văn bản giữa hai ô nhập liệu (textBox1 và textBox2) bằng cách sử dụng một cấu trúc dữ liệu ngăn xếp (ClipboardRB).

Khi người dùng nhấn "Cut" (button1), phần văn bản được chọn từ textBox1 sẽ được cắt và lưu vào ngăn xếp.

Khi người dùng nhấn "Paste" (button2), văn bản từ ngăn xếp sẽ được dán vào textBox2. Nếu phần văn bản được chọn trong textBox1 giống với nội dung từ ngăn xếp, nó sẽ được xóa khỏi textBox1 trước khi dán vào textBox2.

Nút "Clear" (button3) sẽ xóa nội dung của cả hai ô nhập liệu và xóa toàn bộ dữ liệu trong ngăn xếp.

Nút "Close" (button4) sẽ đóng ứng dụng.

```
namespace fCUT
{
    public partial class fCUT : Form
    {
        public fCUT()
        {
            InitializeComponent();
        }

        CLIPBOARDRB.ClipboardRB<object> cb = new CLIPBOARDRB.ClipboardRB<object>();

        private void button1_Click(object sender, EventArgs e)
        {
            cb.Add(textBox1.Text, textBox1.SelectedText);
        }

        private void button2_Click(object sender, EventArgs e)
        {
            textBox2.Text = cb.Paste(textBox2.Text);
            if (textBox1.SelectedText == cb.ClearInput(textBox1.SelectedText))
            {
                textBox1.Text = cb.ClearInput(textBox1.Text);
            }
            textBox1.SelectedText = cb.ClearInput(textBox1.SelectedText);
            textBox1.Focus();
        }

        private void button3_Click(object sender, EventArgs e)
        {
            textBox1.Text = cb.ClearInput(textBox1.Text);
            textBox2.Text = cb.ClearInput(textBox2.Text);
            cb.ClearStack();
        }


        private void button4_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        private void textBox1_TextChanged(object sender, EventArgs e)
        {

        }

        private void textBox2_TextChanged(object sender, EventArgs e)
        {

        }

        private void fCUT_Load(object sender, EventArgs e)
        {
```

```
53
54          }
55      }
56  }
```

Menue chính

```
1   using fCOPY;
2   using System;
3   using System.Collections.Generic;
4   using System.ComponentModel;
5   using System.Data;
6   using System.Drawing;
7   using System.Linq;
8   using System.Text;
9   using System.Threading.Tasks;
10  using System.Windows.Forms;
11  namespace DISPLAYMENU
12  {
13      public partial class DISPLAYMENU : Form
14      {
15          public DISPLAYMENU()
16
17          {
18              InitializeComponent();
19          }
20          private void button1_Click(object sender, EventArgs e)
21          {
22              fCOPY.fCOPY fcopy = new fCOPY.fCOPY();
23              DISPLAYMENU dISPLAYMENU = this;
24              dISPLAYMENU.Hide();
25              fcopy.ShowDialog();
26              this.Show();
27          }
28          private void button2_Click(object sender, EventArgs e)
29          {
30              fCUT.fCUT fcut = new fCUT.fCUT();
31              DISPLAYMENU dISPLAYMENU = this;
32              dISPLAYMENU.Hide();
33              fcut.ShowDialog();
34              this.Show();
35          }
36          private void DISPLAYMENU_Load(object sender, EventArgs e)
37          {
38          }
39          private void label1_Click(object sender, EventArgs e)
40          {
41          }
42
43      }
44  }
```

4 THẢO LUẬN & ĐÁNH GIÁ

4.1 Các kết quả nhận được

4.1.1 Chức năng 1 : Clipboard Copy and Paste

Giao diện khi nhập dữ liệu

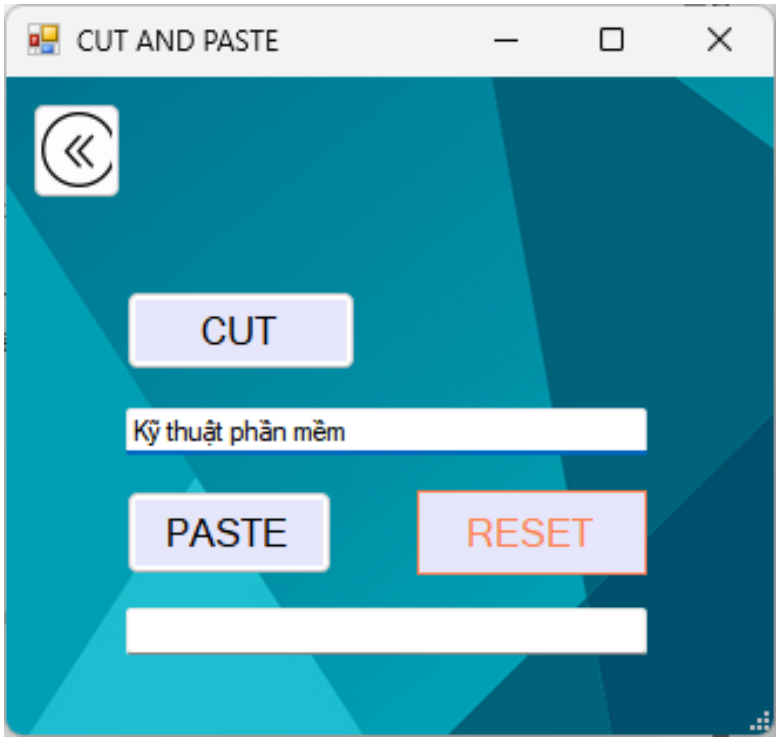
Hình 6: Giao diện khi nhập dữ liệu

Giao diện sau khi nhấn nút Copy và Paste

Hình 7: Giao diện sau khi nhấn nút Copy và Paste

4.1.2 Chức năng 2 : Clipboard Cut and Paste

Giao diện khi nhập dữ liệu



Hình 8: Giao diện khi nhập dữ liệu

Giao diện sau khi nhấn nút Cut và Paste

Hình 9: Giao diện sau khi nhấn nút Cut và Paste

4.2 Một số vấn đề còn tồn đọng

4.2.1 Khả năng tràn bộ đệm ở Stack (stack buffer overflow hay stack buffer overrun)

- Khi vùng cấu trúc dữ liệu không bao gồm phần chương trình viết địa chỉ bộ nhớ lên vùng gọi stack của chương trình.
- Dữ liệu liền kề trên ngăn xếp có thể bị hư hỏng, gây vỡ chương trình và chạy sai.
- Tạo kẻ hở cho kẻ xấu khai thác, nắm quyền kiểm soát dữ liệu, khiến mật độ bảo mật không còn được đảm bảo.

4.2.2 Sự phức tạp khi cần bổ sung ngăn xếp

- Thêm nhiều hàm push, pop và những hàm khác có thể khiến chức năng bị rối, phức tạp.

4.3 Một số ý tưởng và hướng phát triển tiềm năng cho ứng dụng

4.3.1 Ý tưởng

4.3.1.1 Quản Lý Bộ Đệm Tiên Tiến

Ứng dụng sẽ tập trung vào việc cải thiện quản lý bộ đệm thông qua việc tích hợp các công nghệ tiên tiến và chiến lược hiệu suất. Dưới đây là một số tính năng và ý tưởng chủ đạo:

4.3.1.2 Học Máy và Dự Đoán

Sử dụng học máy để phân tích mô hình sử dụng bộ đệm theo thời gian. Dự đoán nhu cầu bộ đệm dựa trên mô hình lịch sử và dữ liệu hoạt động, từ đó tối ưu hóa kích thước bộ đệm và nguồn lực.

4.3.1.3 Cơ Sở Dữ Liệu Phân Tán

Xây dựng cơ sở dữ liệu bộ đệm phân tán để hỗ trợ môi trường phân tán và tăng cường khả năng mở rộng của hệ thống. Tối ưu hóa quá trình đồng bộ hóa giữa các bộ đệm trên các server.

4.3.1.4 Tích Hợp Đồng Bộ Hóa Real-time

Sử dụng cơ chế đồng bộ hóa thời gian thực để đảm bảo sự nhất quán giữa các bộ đệm trong môi trường đa server. Điều này giúp tránh được các vấn đề liên quan đến sự chệch lệch dữ liệu giữa các bộ đệm.

Kiểm Soát Tải Đồng Đều Phát triển cơ chế kiểm soát tải để phân phối công việc đồng đều giữa các bộ đệm. Tối ưu hóa hiệu suất của ứng dụng bằng cách tránh tình trạng quá tải hoặc thiếu tải.

4.3.1.5 Hệ Thống Cảnh Báo và Giám Sát

Tích hợp hệ thống cảnh báo để thông báo về bất kỳ vấn đề nào liên quan đến bộ đệm. Cung cấp bảng điều khiển giám sát để theo dõi hiệu suất và sự hoạt động của bộ đệm.

4.3.2 Kế hoạch phát triển

Giai Đoạn 1 - Tích Hợp Học Máy:
Thu thập dữ liệu lịch sử hoạt động bộ đệm và xây dựng mô hình học máy sử dụng các thuật toán phù hợp. Tích hợp dự đoán nhu cầu bộ đệm vào quá trình quyết định về kích thước và chiến lược thay thế.

Giai Đoạn 2 - Phát Triển Cơ Sở Dữ Liệu Phân Tán:
Thực hiện cơ sở dữ liệu bộ đệm phân tán sử dụng công nghệ hiện đại như NoSQL hoặc NewSQL. Xây dựng cơ chế đồng bộ hóa giữa các bộ đệm để đảm bảo tính nhất quán.

Giai Đoạn 3 - Tối Ưu Hóa Đồng Đều Tải:
Phát triển cơ chế kiểm soát tải để đảm bảo sự phân phối công việc đồng đều giữa các bộ đệm. Tích hợp cơ chế tự động điều chỉnh để thích ứng với biến động trong tải công việc.

Giai Đoạn 4 - Cảnh Báo và Giám Sát:
Xây dựng hệ thống cảnh báo để tự động thông báo về các vấn đề liên quan đến bộ đệm. Phát triển bảng điều khiển giám sát đa chiều để theo dõi hiệu suất và sự hoạt động của bộ đệm.

4.3.3 Kết Luận

Phát triển theo hướng này sẽ mang lại một ứng dụng quản lý bộ đệm hiện đại, linh hoạt và có khả năng tối ưu hóa tự động, đáp ứng linh hoạt với nhu cầu người dùng và biến động trong môi trường hệ thống.

Phụ Lục

GitHub

- LINK: https://github.com/dangdailoi/DSA/tree/master
- URL: https://github.com/dangdailoi/DSA.git

Hướng Dẫn Cài Đặt và Chạy Ứng Dụng

1. Git clone URL
2. Đảm bảo máy tính đã cài đặt Visual Studio và ngôn ngữ lập trình C#.
3. Mở file DISPLAYMENU.sln bằng Visual Studio.
4. Nhấn phím F5 để chạy chương trình.
5. Sử dụng ứng dụng theo hướng dẫn trong tài liệu.

Phân công công việc

Thành viên nhóm	Nhiệm vụ
Đặng Đại Lơi	Tổng hợp file, chỉnh sửa Latex, tổng hợp code, thiết kế giao diện, nội dung chương 1
Lê Hoàng Khang	Viết hàm fCopy, làm Power Point, nội dung chương 3
Châu Thuận Đức	Viết hàm fCut, nội dung chương 4
Từ Gia Bảo	Thiết kế sơ đồ lớp, nội dung chương 2
Nguyễn Nam Thái	Viết hàm Display Menu, nội dung chương 2

References

[1] Paul J. Deitel and Harvey M. Deitel. *C 2010 for Programmers*. Prentice Hall, 2011.

[2] Paul J. Deitel and Harvey M. Deitel. *Data Structures and Algorithms with Object-Oriented Design Patterns in C*. 2004.

[3] Đại học Kinh tế Thành Phố Hồ Chí Minh. *Tài liệu học tập Code theo buổi Học phần Cấu trúc dữ liệu và giải thuật*. Internal document. 2023.

[4] Michael McMillan. *Data Structures and Algorithms Using C*. Cambridge University Press, Mar. 2007.

[5] Trọng Nhân. *Thiết kế giao diện cho WinForms – Series Giải pháp WinForms (2)*. May 2019. URL: https://tuhocict.com/giai-phap-winforms-2-ap-dung-mo-hinh-kien-truc-ui/.