# BÁO CÁO ĐỒ ÁN: LẬP TRÌNH TRỰC QUAN

# ĐỀ TÀI: MÔ PHỎNG CHƯƠNG TRÌNH NOTEPAD++

## LỜI NÓI ĐẦU

……………………………………………………………………………………………………………....

## LỜI CẢM ƠN

……………………………………………………………………………………………………………....

Mục lục

[BÁO CÁO ĐỒ ÁN: LẬP TRÌNH TRỰC QUAN 1](#_Toc485221234)

[ĐỀ TÀI: MÔ PHỎNG CHƯƠNG TRÌNH NOTEPAD++ 1](#_Toc485221235)

[LỜI NÓI ĐẦU 1](#_Toc485221236)

[LỜI CẢM ƠN 1](#_Toc485221237)

[I. TỔNG QUAN 3](#_Toc485221238)

[1. Lý do chọn đề tài 3](#_Toc485221239)

[2. Mục đích nghiên cứu đề tài 3](#_Toc485221240)

[3. Phạm vi nghiên cứu đề tài 3](#_Toc485221241)

[4. Công nghệ sử dụng 3](#_Toc485221242)

[II. GIAO DIỆN CHƯƠNG TRÌNH VÀ CHỨC NĂNG CHÍNH 4](#_Toc485221243)

[1. Màn hình giao diện chính 5](#_Toc485221244)

[2. Thanh công cụ (Menu Bar) 6](#_Toc485221245)

[2.1. File 6](#_Toc485221246)

[2.2. Edit 6](#_Toc485221247)

[2.3. Search 6](#_Toc485221248)

[2.4. View 6](#_Toc485221249)

[2.5. Language 7](#_Toc485221250)

[2.6. Settings 7](#_Toc485221251)

[3. Một số chức năng nổi bật 7](#_Toc485221252)

[III. HIỆN THỰC HÓA MỘT SỐ CHỨC NĂNG NỔI BẬT 7](#_Toc485221253)

[1. Text Area 7](#_Toc485221254)

[1.1. Syntax HighLighting 8](#_Toc485221255)

[1.2. AutoComplete 11](#_Toc485221256)

[1.3. AutoIndenting và BraceMatching 14](#_Toc485221257)

[2. Number Margin 16](#_Toc485221258)

[3. Bookmark Margin 18](#_Toc485221259)

[4. Document Map 21](#_Toc485221260)

[5. Tìm kiếm và thay thế 23](#_Toc485221261)

[6. Tab Bar 26](#_Toc485221262)

[7. Compile C# và Visual Basic 29](#_Toc485221263)

[IV. KHÁC 30](#_Toc485221264)

[1. Thuận lợi và khó khăn 30](#_Toc485221265)

[2. Phân công công việc 30](#_Toc485221266)

[3. Tài liệu tham khảo 30](#_Toc485221267)

## I. TỔNG QUAN

### 1. Lý do chọn đề tài

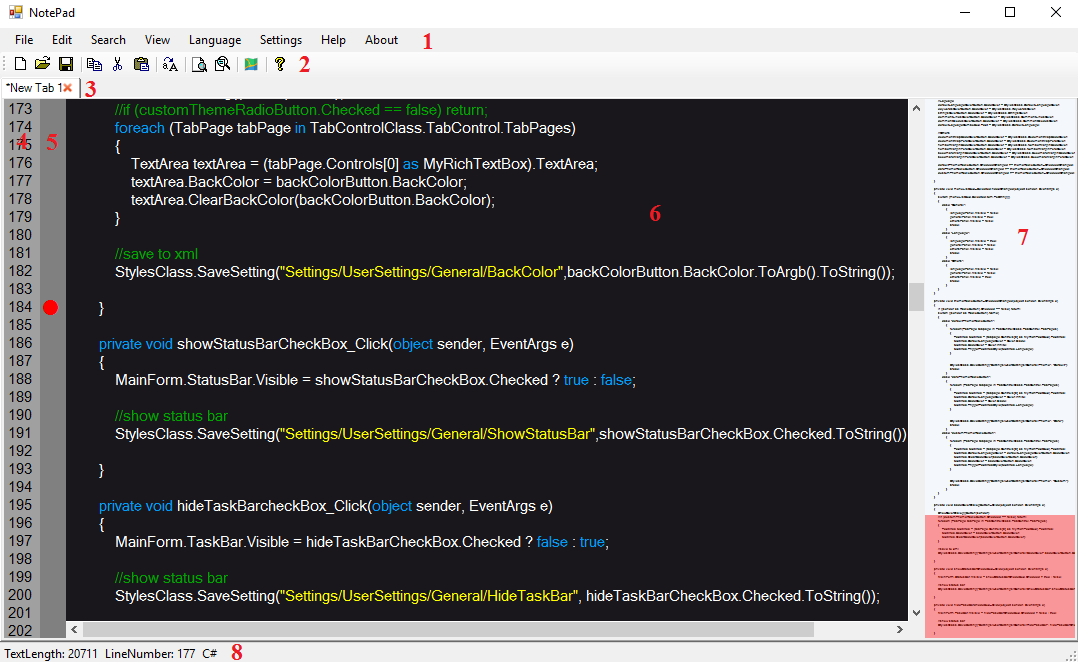
### 2. Mục đích nghiên cứu đề tài

### 3. Phạm vi nghiên cứu đề tài

### 4. Công nghệ sử dụng

## II. GIAO DIỆN CHƯƠNG TRÌNH VÀ CHỨC NĂNG CHÍNH

### 1. Màn hình giao diện chính



|  |  |  |
| --- | --- | --- |
| **STT** | **Tên gọi** | **Chức năng** |
| 1 | Thanh công cụ (Menu Bar) | Cung cấp các thao tác đối với văn bản |
| 2 | Thanh công cụ truy cập nhanh (ToolBar) | Cung cấp các thao tác nhanh đối với văn bản |
| 3 | Thanh quản lý trang (Tab Bar) | Quản lý các trang đã được mở |
| 4 | Thanh đánh số dòng  (Number Margin) | Hiển thị chỉ số dòng |
| 5 | Thanh đánh dấu  (Bookmark Margin) | Đánh dấu một dòng bất kì trong trang văn bản |
| 6 | Trang soạn thảo (Text Area) | Soạn thảo văn bản |
| 7 | Document Map | Xem sơ lược toàn bộ trang văn bản |
| 8 | Thanh trạng thái (Status bar) | Hiển thị một số trạng thái của trang văn bản (Độ dài, số dòng, ngôn ngữ) |

### 2. Thanh công cụ (Menu Bar)

#### 2.1. File

|  |  |
| --- | --- |
|  | **New:** Tạo một trang văn bản mới.  **Open:** Mở một trang văn bản theo đường dẫn người dùng mong muốn.  **Save:** Lưu một trang văn bản xuống máy tính theo đường dẫn người dùng mong muốn (Lưu ngầm định nếu văn bản đã có đường dẫn trước đó)  **Save As:** Lưu một trang văn bản xuống máy tính theo đường dẫn người dùng mong muốn (Không ngầm định)  **Page Setup:** Cho phépthiết lập cài đặt trang trước khi in.  **Preview:** Xem tổng quan các trang trước khi in.  **Print:** Thiết lập và in trang.  **Close:** Đóng trang hiện tại.  **Close All:** Tắt toàn bộ trang hiện đang mở. |

#### 2.2. Edit

|  |  |
| --- | --- |
|  | **Undo:** Quay trở lại hành động trước đó đã thực hiện đối với trang văn bản.  **Redo:** Quay trở lại hành động Undo.  **Copy:** Sao chép đoạn văn bản được chọn.  **Cut:** Cắt đoạn văn bản được chọn.  **Paste:** Dán đoạn văn bản đã Copy hoặc Cut.  **Select All:** Bôi đen toàn bộ nội dung văn bản.  **Clear:** Xóa toàn bộ nội dung văn bản.  **Uppercase:** Thay thế đoạn văn bản được chọn với toàn bộ chữ in hoa.  **Lowercase:** Thay thế đoạn văn bản được chọn với toàn bộ chữ in thường. |

#### 2.3. Search

|  |  |
| --- | --- |
|  | **Find:** Tìm kiếm một đoạn văn bản.  **Find And Replace:** Tìm kiếm và thay thế một đoạn văn bản. |

#### 2.4. View

|  |  |
| --- | --- |
|  | **Zoom In:** Phóng to trang văn bản.  **Zoom Out:** Thu nhỏ trang văn bản.  **Zoom 100%:** Đưa trang văn bản về kích cỡ mặc định.  **Collapse All:**  **Expand All:**  **Document Map:** Hiển thị document map. |

#### 2.5. Language

|  |  |
| --- | --- |
|  | **NormalText:** Soạn thảo văn bản thông thường.  **C:** Soạn thảo văn bản với ngôn ngữ C.  **C++:** Soạn thảo văn bản với ngôn ngữ C++.  **C#:** Soạn thảo văn bản với ngôn ngữ C#.  **VB:** Soạn thảo văn bản với ngôn ngữ VB. |

#### 2.6. Settings

|  |  |
| --- | --- |
|  | **Preferences:** Cho phép một số tùy chỉnh đối với giao diện, ngôn ngữ và văn bản. |

### 3. Một số chức năng nổi bật

* Syntax highlighting
* AutoComplete (Keyword completion)
* AutoIndenting và BraceMatching
* Undo và Redo
* Hỗ trợ đa ngôn ngữ
* Document map
* Multi-Document (Tab interface)
* Tìm kiếm và thay thế
* Compile C# và Visual Basic

## III. HIỆN THỰC HÓA MỘT SỐ CHỨC NĂNG NỔI BẬT

### 1. Text Area

Để thuận tiện cho việc xử lý text (undo, redo, syntax highlight, auto completion, auto indentation,…), Text Area được tạo lập thành một class riêng kế thừa từ class RichTextBox có sẵn.

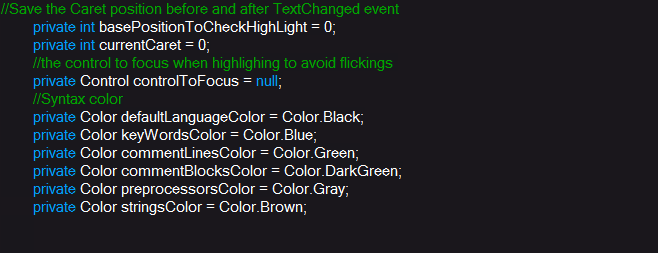
**Cấu trúc:** public class TextArea : RichTextBox

Việc kế thừa từ class RichTextBox cho phép ta chỉnh sửa, tô màu các đoạn text rời rạc thông qua các thuộc tính (property): SelectionStart, SelectionLength, SelectionColor, SelectedText. Đây cũng chính là các thuộc tính được sử dụng nhiều nhất trong hiện thực hóa chương trình và cách sử dụng các thuộc tính này sẽ được nói đến ngay sau đây.

#### 1.1. Syntax HighLighting

##### 1.1.1. Giới thiệu

Syntax HighLighting là một trong những chức năng quan trọng nhất trong việc coding cho lập trình viên. Chức năng này hiển thị những đoạn text bằng các màu sắc khác nhau tùy thuộc vào ngôn ngữ mà lập trình viên sử dụng, cụ thể hơn là những từ khóa (keywords) mà ngôn ngữ đó cung cấp. Syntax Highlighting giúp lập trình viên dễ dàng hơn trong việc viết cũng như phân biệt các cấu trúc, cú pháp của một ngôn ngữ nhất định. Tuy nhiên cần lưu ý rằng Syntax Highlighing không hề ảnh hưởng đến ý nghĩa của đoạn text.



*Hình :Ví dụ về syntax highlighing*

##### 1.1.2. Hiện thực hóa

* Việc tô màu các đoạn text được thực hiện bằng cách kiểm tra dòng mà người dùng hiện đang nhập liệu. Dòng này liên tục được trả về màu mặc định sau đó tìm kiếm các đoạn text cần được tô màu và tô màu những đoạn text đó. Tuy nhiên, việc chỉ kiểm tra trên dòng người dùng hiện đang nhập liệu có thể gây ra thiếu sót nếu người dùng sao chép và dán một lượng lớn text vào Text Area. Vì vậy, vị trí bắt đầu để kiểm tra tô màu nên là vị trí đầu tiên của dòng chứa dấu sót(caret) trước đó và vị trí kết thúc nên là vị trí cuối cùng của dòng chứa dấu sót (caret) hiện tại.

//Field is used to save the previous caret

private int previousCaret= 0;

/// <summary>

/// Save the previous caret by Onpreviewkeydown

/// </summary>

/// <param name="e"></param>

protected override void OnPreviewKeyDown(PreviewKeyDownEventArgs e)

{

base.OnPreviewKeyDown(e);

previousCaret= this.SelectionStart;

}

* Việc trả về màu mặc định được thực hiện bởi hàm ClearStyle

/// <summary>

/// Clear the style of current line or current selected text

/// </summary>

/// <param name="color"></param>

private void ClearStyle(Color foreColor)

{

//currentCaret is the current caret position

if (previousCaret > currentCaret)

{

previousCaret = currentCaret - 1;

}

//Set SelectionStart

//Selection start should be the first character of the line that it contains

//the basePositionToCheckHighLight

SelectionStart = GetFirstCharIndexOfLine(previousCaret);

//Set selectionLength

//Selection end should be the last chareacter of current line that it contains

//the current caret

SelectionLength = GetLastCharIndexOfLine(currentCaret) - SelectionStart;

//Set color

SelectionColor = foreColor;

//Set font for sure

SelectionFont = this.Font;

//Set backColor for sure

SelectionBackColor = this.BackColor;

}

* Vì việc tô màu các đoạn text làm cho vị trí caret thay đổi, cho nên vị trí caret hiện tại được lưu bởi biến currentCaret.

//Field is used to save the current caret position

private int currentCaret = 0;

/// <summary>

/// Save the current caret by OnTextChanged event

/// </summary>

/// <param name="e"></param>

protected override void OnTextChanged(EventArgs e)

{

base.OnTextChanged(e);

currentCarret = this.SelectionStart;

}

* Hàm GetFirstCharIndexOfLine và GetLastCharIndexOfLine được thực hiện như sau:

/// <summary>

/// Get first char index of the real line from a specified char index

/// </summary>

private int GetFirstCharIndexOfLine(int anyCharIndexInThisLine)

{

int firstCharIndex = anyCharIndexInThisLine;

if (firstCharIndex < 0 || firstCharIndex > TextLength) return 0;

while (firstCharIndex > 0 && Text[firstCharIndex - 1] != '\n')

{

firstCharIndex--;

}

return firstCharIndex;

}

/// <summary>

/// Get last char index of the real line from a specified char index

/// </summary>

private int GetLastCharIndexOfLine(int anyCharIndexInThisLine)

{

int lastCharIndex = anyCharIndexInThisLine;

if (lastCharIndex <0 || lastCharIndex > Text.Length) return TextLength;

while lastCharIndex < Text.Length && Text[lastCharIndex] != '\n')

{

lastCharIndex++;

}

return lastCharIndex;

}

* Hàm SetStyle được dùng để tìm kiếm các từ khóa cần được tô màu mà tô màu chúng (Việc tìm kiếm các từ khóa được hỗ trợ từ thư viện Regex có sẵn)

/// <summary>

/// Set the style for current line or current selected text

/// </summary>

/// <param name="pattern"></param>

/// <param name="color"></param>

private void SetStyle(string pattern, Color foreColor)

{

//save selectionStart and selection length for futher check comment or string

int start = SelectionStart;

int length = SelectionLength;

//the text to check pattern

string textToCheck = Text.Substring(SelectionStart, SelectionLength);

//the offset of this text related to the TextArea

int offset = SelectionStart;

//Get Match Collection

MatchCollection patternMatches = Regex.Matches(textToCheck, pattern);

//loop through the MathCollection

foreach (Match m in patternMatches)

{

//set selection start

SelectionStart = m.Index + offset;

//set selection length

SelectionLength = m.Length;

//Set color

SelectionColor = foreColor;

}

SelectionStart = start;

SelectionLength = length;

}

* Cuối cùng, sau khi kết hợp các hàm hợp lý, hàm AutoSyntaxHighLight có nội dung như sau: (Hàm AutoSyntaxHighLight được đặt vào event OnTextChanged)

/// <summary>

/// Auto Syntax highlight

/// </summary>

/// <param name="selectedLanguage"></param>

private void AutoSyntaxHighLight(string selectedLanguage)

{

// saving the original caret position

int originalLength = SelectionLength;

currentCaret = SelectionStart;

//Syntax highlighting

switch (selectedLanguage)

{

case "C#":

{

ClearStyle(defaultLanguageColor);

SetStyle(@"\b(abstract|as|base|bool|break)\b", keyWordsColor);

break;

}

}

// restoring the original caret position

SelectionStart = currentCaret;

SelectionLength = originalLength;

SelectionColor = defaultLanguageColor;

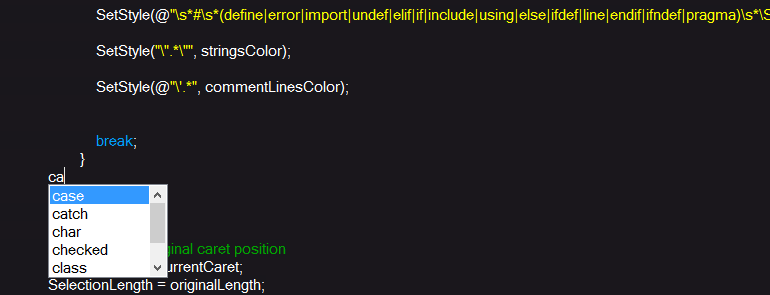
previousCaret = currentCaret;

}

#### 1.2. AutoComplete

##### 1.2.1. Giới thiệu

Auto Completion giúp lập trình viên có thể hoàn thành các từ khóa (keywords) có sẵn dễ dàng hơn mà không cần phải nhập hoặc nhớ cả từ khóa. Chức năng này đưa ra các dự đoán về phần còn lại của từ khóa mà lập trình viên đang nhập và người dùng có thể lựa chọn các từ khóa này thông qua các phím mũi tên và phím tab hoặc enter để chấp nhận hoàn thành từ khóa. Auto Completion giúp lập trình viên tăng tốc độ viết code lên rất nhiều.



*Hình :Ví dụ về Auto completion*

##### 1.2.2. Hiện thực hóa

* Class ListBox có sẵn cho phép ta thêm các từ khóa(keywords) vào thể hiện(instance) của lớp này khá dễ dàng. Để đơn giản và mang lại hiệu năng cao, ta có thể khởi tạo một chuỗi chứa tất cả các từ khóa cần thiết ngăn cách bởi dấu khoảng trắng, cắt chuỗi và thêm vào thể hiện đã khởi tạo.

//autocomplete listbox

private ListBox autoCompleteListBox = new ListBox();;

/// <summary>

/// add keywords to list box

/// </summary>

/// <param name="keywordsString"></param>

private void AddKeyWordsToListBox(string keywordsString)

{

if (autoCompleteListBox == null) return;

//clear for sure

autoCompleteListBox.Items.Clear();

//split keywords

string[] keywords = keywordsString.Split(' ');

//add items to listbox

autoCompleteListBox.Items.AddRange(keywords);

}

* Tính toán và hiển thị ListBox được thực hiện bởi hàm AutoShowCompleteListBox

private void AutoShowCompleteListBox(string selectedLanguage)

{

string currentWord = GetCurrentWord();

if (currentWord != "")

{

//if listbox hasn't shown

if (autoCompleteListBox.Visible == false)

{

//calculate position to show autocompletelistbox

int wordStartPosition = GetWordStartPosition(SelectionStart);

Point rawPostionToShowListBox =

GetPositionFromCharIndex(wordStartPosition);

autoCompleteListBox.Location =

new Point(rawPostionToShowListBox.X,

rawPostionToShowListBox.Y + (int)(this.FontHeight \*

this.ZoomFactor));

}

//set selected index

autoCompleteListBox.SelectedIndex =

autoCompleteListBox.FindString(currentWord);

//show autocompletelistbox

if (autoCompleteListBox.SelectedIndex >= 0)

{

autoCompleteListBox.Visible = true;

}

else

{

autoCompleteListBox.Visible = false;

}

}

else

{

autoCompleteListBox.Visible = false;

}

}

* Hàm GetCurrentWord có thể được viết như sau:

/// <summary>

/// Get current word

/// </summary>

/// <returns></returns>

private string GetCurrentWord()

{

int wordStartPosition = GetWordStartPosition(SelectionStart);

int wordEndPosition = GetWordEndPosition(SelectionStart);

return this.Text.Substring(wordStartPosition, wordEndPosition –

wordStartPosition);

}

/// <summary>

/// Get word start position by a specified position

/// </summary>

/// <param name="charIndex"></param>

/// <returns></returns>

private int GetWordStartPosition(int charIndex)

{

while ((charIndex > 0) && Text[charIndex - 1] != ' ' && Text[charIndex - 1] !=

'\t' && Text[charIndex - 1] != '\n')

{

charIndex--;

}

return charIndex;

}

/// <summary>

/// Get word end position by a specified position

/// </summary>

/// <param name="charIndex"></param>

/// <returns></returns>

private int GetWordEndPosition(int charIndex)

{

while ((charIndex < TextLength) && Text[charIndex] != ' ' && Text[charIndex] !=

'\t' && Text[charIndex] != '\n')

{

charIndex++;

}

return charIndex;

}

* Để điều khiển hoạt động của ListBox này, ta có thể sử dụng ProcessCmdKey. Hàm này cho phép ta điều khiển hoạt động của ListBox mà vẫn không để mất “focus” ở Text area.

/// <summary>

/// Override ProcessCmdKey to have full control of autocompletelistbox

/// And catch keys down to control Undo Action

/// </summary>

/// <param name="m"></param>

/// <param name="keyData"></param>

/// <returns></returns>

protected override bool ProcessCmdKey(ref Message m, Keys keyData)

{

//Control ListBox//

//If an auto complete list box has already shown

if (autoCompleteListBox.Visible/\*isShownAutoCompleteListBox\*/)

{

switch (keyData)

{

//move down to the next auto complete words

case Keys.Down:

{

if (autoCompleteListBox.SelectedIndex !=

autoCompleteListBox.Items.Count - 1)

{

autoCompleteListBox.SelectedIndex += 1;

}

return true;

}

//move up to the next auto complete words

case Keys.Up:

{

if (autoCompleteListBox.SelectedIndex != 0)

{

autoCompleteListBox.SelectedIndex -= 1;

}

return true;

}

//insert selected auto complete word

case Keys.Enter:

case Keys.Tab:

{

//Get the appropriate text to replace by autocomplete word

this.SelectionStart=GetWordStartPosition(SelectionStart);

this.SelectionLength = GetCurrentWord().Length;

this.SelectedText =

autoCompleteListBox.SelectedItem.ToString() + " ";

autoCompleteListBox.Visible = false;

return true;

}

}

}

return base.ProcessCmdKey(ref m, keyData);

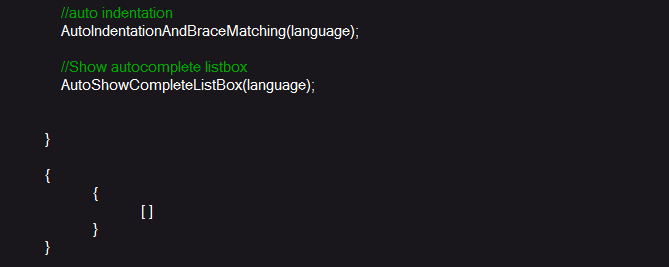
}

#### 1.3. AutoIndenting và BraceMatching

##### 1.3.1. Giới thiệu

AutoIndenting (Tự động cách dòng) giúp lập trình viên dễ dàng hơn trong việc đọc hiểu một lượng code lớn vì các cấp độ của code được phân chia rõ ràng (như if else, switch case,...).

BraceMatching có chức năng tương tự như AutoComplete giúp lập trình viên hoàn thành dấu ngoặc còn thiếu như: {, [, {,…



*Hình :Ví dụ về AutoIndenting và BraceMatching*

##### 1.3.2. Hiện thực hóa

* Để thực hiện BraceMatching, ta chỉ cần kiểm tra xem ký tự trước đó có phải là một trong các kí tự {, [, (,.. hay không. Nếu có, ta thực hiện thêm vào vị trí caret hiện tại ký tự đóng ngoặc phù hợp.

switch(this.Text[this.SelectionStart - 1])

{

case '{':

{

this.SelectedText = " }";

break;

}

case '(':

{

this.SelectedText = " )";

break;

}

case '[':

{

this.SelectedText = " ]";

break;

}

case '<':

{

this.SelectedText = " >";

break;

}

}

this.SelectionStart = this.SelectionStart - 1;

* AutoIndenting được thực hiện bằng cách kiểm tra lượng lùi dòng (indent) của dòng trước đó và thêm lượng lùi dòng (indent) vào dòng hiện tại sao cho phù hợp.

int previousLineNumber = this.GetLineFromCharIndex(this.SelectionStart) - 1;

//get previous line

string previousLine = this.Lines[previousLineNumber];

//get the amount of indent of previous line

//read more about regex here: https://autohotkey.com/docs/misc/RegEx-QuickRef.htm

Match indent = Regex.Match(previousLine, @"^[ \t]\*");

//add indent

this.SelectedText = indent.Value;

//if selection start is '}', do some works

if (Regex.IsMatch(this.Lines[this.GetLineFromCharIndex(this.SelectionStart)],@"}\s\*"))

{

this.SelectedText = "\t\n" + indent.Value;

this.SelectionStart = this.GetFirstCharIndexOfCurrentLine() - 1;

}

### 2. Number Margin

* Để tiện lợi cho việc vẽ chỉ số dòng, ta có thể kế thừa class NumberMargin từ class Panel hoặc Control có sẵn.

**Cấu trúc:** public class NumberMargin : Control

* Ý tưởng để vẽ chỉ số dòng là tìm vị trí của dòng đầu tiên và dòng cuối cùng mà vùng Text area đang hiển thị và vẽ các chỉ số này với vị trí song song thích hợp. Và việc tìm vị trí dòng đầu tiên và dòng cuối cùng này còn giúp việc vẽ các chỉ số nhanh hơn vì ta không cần phải vẽ lại toàn bộ chỉ số dòng.

protected override void OnPaint(PaintEventArgs e)

{

base.OnPaint(e);

if (textArea != null)

{

//get the zoom factor of the text Area

//this is needed when we zoom in or zoom out the text area

float zoom = textArea.ZoomFactor;

//Font to draw number

//Since we just get the familly name of numberFont, we have to compute //the actually size of the font to draw numbers

Font fontToDrawNumber = new Font(numberFont.FontFamily.Name,

textArea.Font.Size \* zoom);

//set the width of this control

this.Width =

TextRenderer.MeasureText((textArea.Lines.Length+1).ToString(),

fontToDrawNumber).Width + widthPadding;

//we just need to draw the line number of the lines visibly in thescreen

//Get the first and last char index currently visible in the screen

firstVisibleCharIndex =

textArea.GetCharIndexFromPosition(new Point(0, 0));

lastVisibleCharIndex =

textArea.GetCharIndexFromPosition(new Point(textArea.Width, textArea.Height));

//Get the first and last line currently visible in the screen

firstVisibleLine = this.GetLineFromCharIndex(firstVisibleCharIndex);

lastVisibleLine = this.GetLineFromCharIndex(lastVisibleCharIndex);

//save the position of the current line

//note that this is the location within the textArea

Point rawPosition;

//save to position to draw number

//this is the actually position to draw number within this control

Point positionToDrawNumber;

//loop through all the lines currently visible in the screen

for (int lineIndex = firstVisibleLine; lineIndex <= lastVisibleLine;

lineIndex++)

{

//get the location of current line

//note that this is the location within the textArea

rawPosition = textArea.GetPositionFromCharIndex(

textArea.GetFirstCharIndexFromLine(lineIndex));

//get the position to draw current line number

positionToDrawNumber = new Point(widthPadding / 2,

rawPosition.Y + yOffset);

//draw current line number

e.Graphics.DrawString((lineIndex + 1).ToString(),

fontToDrawNumber, numberBrush, positionToDrawNumber);

}

//Draw one more number//

//get the location of current line

rawPosition = textArea.GetPositionFromCharIndex(

textArea.GetFirstCharIndexFromLine(lastVisibleLine));

//get the position to draw current line number

positionToDrawNumber = new Point(widthPadding / 2,

rawPosition.Y + yOffset + fontToDrawNumber.Height);

//draw line number

e.Graphics.DrawString((lastVisibleLine + 2).ToString(),

fontToDrawNumber, numberBrush, positionToDrawNumber);

//Dispose

fontToDrawNumber.Dispose();

}

}

* Ta cần vẽ lại (hay kích hoạt sự kiện OnPaint) control này mỗi khi Text Area thay đổi . Vì vậy, Text Area cần được thêm vào các sự kiện sau:

protected override void OnTextChanged(EventArgs e)

{

base.OnTextChanged(e);

if (numberMargin != null)

{

numberMargin.Refresh();

}

}

protected override void OnResize(EventArgs e)

{

base.OnResize(e);

if (numberMargin != null)

{

numberMargin.Refresh();

}

}

protected override void OnContentsResized(ContentsResizedEventArgs e)

{

base.OnContentsResized(e);

if (numberMargin != null)

{

numberMargin.Refresh();

}

}

protected override void OnVScroll(EventArgs e)

{

base.OnVScroll(e);

if (numberMargin != null)

{

numberMargin.Refresh();

}

}

### 3. Bookmark Margin

* Ý tưởng của bookmark margin hoàn toàn giống với Number margin.

**Cấu trúc:** public class BookmarkMargin: Control

* Để lưu vị trí của các dòng đã được đánh dấu, ta đơn giản sử dụng một danh sách kiểu int.

private List<int> bookMarkedLines = new List<int>();

* Điều khiển việc thêm hoặc xóa đánh dấu chỉ số dòng khi người dùng click vào một vị trí bất kì trong bookmark margin được thực hiện bởi hàm OnClick.

protected override void OnClick(EventArgs e)

{

base.OnClick(e);

if (textArea == null || bookMarkedLines == null) return;

//get the line number nearest the MousePosition

int charIndex =

textArea.GetCharIndexFromPosition( textArea.PointToClient(MousePosition));

int line = textArea.GetLineFromCharIndex(charIndex);

if (bookMarkedLines.Contains(line) == false)

{

bookMarkedLines.Add(line);

this.Refresh();

}

else

{

bookMarkedLines.Remove(line);

this.Refresh();

}

}

* OnPaint được sử dụng để vẽ kí hiệu đánh dấu dòng

//the font of BookMark

private Font bookMarkFont = new Font("Webdings", 10);

//the brush of number

private Brush bookMarkBrush = Brushes.Red;

override void OnPaint(PaintEventArgs e)

{

base.OnPaint(e);

if (textArea == null || bookMarkedLines == null) return;

//the zoom factor of parent TextArea

float zoom = textArea.ZoomFactor;

//font to Draw Bookmark

Font fontToDrawBookMark = new Font(bookMarkFont.FontFamily.Name,

textArea.Font.Size \* zoom);

//Compute the width of this control

this.Width = TextRenderer.MeasureText("n", fontToDrawBookMark).Width;

if (bookMarkedLines.Count > 0)

{

//a temp list to easily store bookmarked lines

List<int> cloneBookMarkedLines = new List<int>(bookMarkedLines);

//Get the first and last char index currently visible in the screen

firstVisibleCharIndex =

textArea.GetCharIndexFromPosition(new Point(0, 0));

lastVisibleCharIndex =

textArea.GetCharIndexFromPosition(new Point(textArea.Width, textArea.Height));

//Get the first and last line currently visible in the screen

firstVisibleLine = this.GetLineFromCharIndex(firstVisibleCharIndex);

lastVisibleLine = this.GetLineFromCharIndex(lastVisibleCharIndex);

for (int lineIndex = firstVisibleLine; lineIndex <= lastVisibleLine;

lineIndex++)

{

//if there is this line in tempList

if (cloneBookMarkedLines.Contains(lineIndex))

{

//Draw bookmark symbol

Point rawPosition = textArea.GetPositionFromCharIndex(

textArea.GetFirstCharIndexFromLine(lineIndex));

e.Graphics.DrawString("n", fontToDrawBookMark,

bookMarkBrush, new Point((int)widthPadding / 2,

rawPosition.Y + yOffset - 1));

//Remove this line from tempList to be able to check the //next bookmarked line

cloneBookMarkedLines.Remove(lineIndex);

}

//if cloneBookMarhedLines has nothing left

if (cloneBookMarkedLines.Count == 0) break;

}

//clear for sure

cloneBookMarkedLines.Clear();

}

//dispose

fontToDrawBookMark.Dispose();

}

* Và cũng tương tự với Number margin, ta cần vẽ lại (hay kích hoạt sự kiện OnPaint) control mỗi khi Text Area thay đổi . Tuy nhiên, event TextChanged có một chút phức tạp hơn khi ta phải tính toán lại vị trí của hợp lý trước khi thực hiện việc đánh dấu chỉ số dòng.

protected override void OnTextChanged(EventArgs e)

{

base.OnTextChanged(e);

if (bookmarkMargin != null)

{

bookmarkMargin. CalculateOnTextAreaTextChanged();

}

}

* Hàm CalculateOnTextAreaTextChanged được viết như sau:

private int previousLinesLength = 0;

public void CalculateOnTextAreaTextChanged()

{

if (textArea == null || bookMarkedLines == null) return;

//Calculate the bookmark position

if (previousLinesLength != textArea.Lines.Length)

{

//get the line number nearest the SelectionStart

int selectedLine =

textArea.GetLineFromCharIndex(textArea.SelectionStart);

//get the start line position to calculate from

int startLineToCalculate =

selectedLine - (textArea.Lines.Length – previousLinesLength);

//get lines are needed to recalculate bookmark position

List<int> needToReCaculateLines =

bookMarkedLines.FindAll(i => i >=startLineToCalculate);

//remove all the line >= startLineToCalculate

//in list of bookmarked lines

bookMarkedLines.RemoveAll(i => i >= startLineToCalculate);

//recalculate bookmark position

for (int i = 0; i < needToReCaculateLines.Count; i++)

{

//if startLineToCalculate has a bookmark

if (needToReCaculateLines[i] == startLineToCalculate)

{

//if we cut this line at somewhere in the middle,

//we shouldn't recalculate the bookmark of this line

if (textArea.Lines.Length - previousLinesLength > 0 &&

textArea.SelectionStart >= 2 &&

textArea.Text[textArea.SelectionStart - 2] != '\n')

{

continue;

}

}

//calculate the new bookmark position

needToReCaculateLines[i] +=

textArea.Lines.Length – previousLinesLength;

}

//Add to bookmarked list again

bookMarkedLines.AddRange(needToReCaculateLines);

//reset previous lines length

previousLinesLength = textArea.Lines.Length;

this.Refresh();

}

}

### 4. Document Map

* Document map được kế thừa từ class Control hoặc Panel có sẵn.

**Cấu trúc:** public class DocumentMap : Control

* Ý tưởng: ..

protected override void OnPaint(PaintEventArgs e)

{

base.OnPaint(e);

if (textArea == null) return;

//Get the font used to draw text

Font fontToDrawText = new Font(textArea.Font.FontFamily.Name, sizeOfText,

FontStyle.Regular);

//Calculate lineOffset between this control and textArea

int linesOffset = textArea.Height/fontToDrawText.Height –

(int)(textArea.Height/(textArea.Font.Height \* textArea.ZoomFactor));

//Get the first and last char index currently visible in the screen

firstVisibleCharIndex = textArea.GetCharIndexFromPosition(new Point(0, 0));

lastVisibleCharIndex =

textArea.GetCharIndexFromPosition(new Point(textArea.Width, textArea.Height));

//Get the first and last line currently visible in the screen

firstVisibleLine = this.GetLineFromCharIndex(firstVisibleCharIndex);

lastVisibleLine = this.GetLineFromCharIndex(lastVisibleCharIndex);

//Calculate base char index to draw text from //

//if firstVisibleLine < linesOffset, just get the first character

//of the text of textArea

if (firstVisibleLine - linesOffset < 0 || linesOffset<0)

linesOffset = firstVisibleLine;

int baseCharIndex =

textArea.GetFirstCharIndexFromLine(firstVisibleLine - linesOffset);

//DRAW TEXT//

string TextToDraw = textArea.Text.Substring(baseCharIndex,

textArea.TextLength - baseCharIndex);

//Draw the text of the parent text area into this control

TextRenderer.DrawText(e.Graphics, TextToDraw, fontToDrawText,

new Point(0, 0), this.ForeColor);

//DRAW RECTANGLE//

//calculate y Offset of the rectangle

int yOffset =

TextRenderer.MeasureText(e.Graphics, textArea.Text.Substring(baseCharIndex,

firstVisibleCharIndex - baseCharIndex), fontToDrawText).Height;

//calculate the height of the rectangle

int rectangleHeight = TextRenderer.MeasureText(e.Graphics,

textArea.Text.Substring(firstVisibleCharIndex,

lastVisibleCharIndex - firstVisibleCharIndex),fontToDrawText).Height;

//Draw rectangle

e.Graphics.FillRectangle(rectangleBrush, 0f, yOffset, this.Width, rectangleHeight);

//Dispose for sure

fontToDrawText.Dispose();

}

* Tương tự number margin, ta cần vẽ lại (hay kích hoạt sự kiện OnPaint) control này mỗi khi Text Area thay đổi .

protected override void OnTextChanged(EventArgs e)

{

base.OnTextChanged(e);

if (documentMap != null)

{

documentMap.Refresh();

}

}

protected override void OnResize(EventArgs e)

{

base.OnResize(e);

if (documentMap!= null)

{

documentMap.Refresh();

}

}

protected override void OnContentsResized(ContentsResizedEventArgs e)

{

base.OnContentsResized(e);

if (documentMap!= null)

{

documentMap.Refresh();

}

}

protected override void OnVScroll(EventArgs e)

{

base.OnVScroll(e);

if (documentMap!= null)

{

documentMap.Refresh();

}

}

### 5. Tìm kiếm và thay thế

* Tìm kiếm tất cả:

//List of texts found out

List<int> textsFound = new List<int>();

int indexOfSearchText = -1;

Color allFoundTextBackColor = Color.Aquamarine;

Color defaultBackColor = Color.White;

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

{

////Remove highlighted text of previous search

textArea.Select(0, this.TextLength);

textArea.SelectionBackColor = defaultBackColor;

textArea.SelectionLength = 0;

// Clear for sure

textsFound.Clear();

// Highlighted backcolor of all found text

textsFound = textArea.FindAndColorAll(searchTermTextBox.Text,

allFoundTextBackColor);

indexOfSearchText = -1;

}

* Hàm FindAndColorAll:

public List<int> FindAndColorAll(string textToSearch, Color backColor)

{

//list to hold all the positions

List<int> listOfPositions = new List<int>();

int position = -1;

int searchStart = 0;

int searchEnd = this.TextLength;

// A valid starting index should be specified.

// if index= -1, the end of search

while (searchStart < this.Text.Length)

{

// A valid ending index

// Find the position of search string in RichTextBox

position = this.Find(textToSearch, searchStart, searchEnd,

RichTextBoxFinds.None);

// Determine whether the text was found in textArea

if (position != -1)

{

listOfPositions.Add(position);

this.Select(position, textToSearch.Length);

this.SelectionBackColor = backColor;

searchStart = position + textToSearch.Length;

}

else

{

break;

}

}

return listOfPositions;

}

* Tìm vị trí tiếp theo:

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

{

if(previousText != textArea.Text)

{

previousText = textArea.Text;

//get this again because we might have changed the text in text area and

it made some of the found text position changed

textsFound.Clear();

textsFound = textArea.FindAndColorAll(searchTermTextBox.Text,

allFoundTextBackColor);

}

if (textsFound.Count != 0)

{

if (searchTermTextBox.Text.Length == 0)

return;

//Next position

indexOfSearchText++;

if (indexOfSearchText == textsFound.Count)

{

indexOfSearchText = 0;

}

//highlight backcolor for select text

textArea.Select(textsFound[indexOfSearchText],

searchTermTextBox.Text.Length);

}

}

* Hàm tìm vị trí trước đó hoàn toàn tương tự với hàm tìm vị trí kế tiếp:

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

{

if(previousText != textArea.Text)

{

previousText = textArea.Text;

//get this again because we might have changed the text in text area and

it made some of the found text position changed

textsFound.Clear();

textsFound = textArea.FindAndColorAll(searchTermTextBox.Text,

allFoundTextBackColor);

}

if (textsFound.Count != 0)

{

if (searchTermTextBox.Text.Length == 0)

return;

//previous position

indexOfSearchText--;

if (indexOfSearchText <= -1)

{

indexOfSearchText = textsFound.Count - 1;

}

//highlight backcolor for select text

textArea.Select(textsFound[indexOfSearchText],

searchTermTextBox.Text.Length);

}

}

* Hàm thay thế:

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

{

textArea.Focus();

if (indexOfSearchText == -1 ||

searchTermTextBox.Text == replacementTextBox.Text) ||

textArea.SelectionLength == 0)

{

return;

}

//get this again because we might have changed the text in text area and it

made some of the found text position changed

textsFound.Clear();

textsFound = textArea.FindAndColorAll(searchTermTextBox.Text,

allFoundTextBackColor);

textArea.Select(textsFound[indexOfSearchText], searchTermTextBox.Text.Length);

textArea.SelectedText = replacementTextBox.Text;

//just select for nothing much

textArea.Select(textsFound[indexOfSearchText], replacementTextBox.Text.Length);

}

* Hàm thay thế tất cả:

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

{

if (searchTermTextBox.Text.Equals(replacementTextBox.Text))

{

return;

}

//get this again because we might have changed the text in text area and it

made some of the found text position changed

textsFound.Clear();

textsFound = textArea.FindAndColorAll(searchTermTextBox.Text,

allFoundTextBackColor);

int index = 0;

int offset = 0;

while (index < TextsFound.Count)

{

textArea.Select(TextsFound[index], searchTermTextBox.Text.Length);

textArea.SelectedText = replacementTextBox.Text;

index++;

if (index < TextsFound.Count)

{

offset +=

searchTermTextBox.Text.Length – replacementTextBox.Text.Length;

textsFound[index] = TextsFound[index] - offset;

}

}

//string textToReplace = textArea.Text.Replace(searchTermTextBox.Text, //replacementTextBox.Text);

//textArea.Select(0, textArea.TextLength);

//textArea.ReplaceSelectedText(textToReplace);

}

### 6. Tab Bar

* Vẽ dấu “x” trên mỗi tab page

/// <summary>

/// Draw a close image in the right side of a tab

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private static void tabControl\_DrawItem(object sender, DrawItemEventArgs e)

{

//Get the rectangle of the tab being draw

Rectangle tabRect = tabControl.GetTabRect(e.Index);

if (e.Index == tabControl.SelectedIndex)

{

//Get the rectangle of the selected tab

tabRect = tabControl.GetTabRect(tabControl.SelectedIndex);

//Set background Color of this tab

e.Graphics.FillRectangle(Brushes.White, tabRect);

}

//Inflate it (actually this code deflate this rectangle by -2 both x and y )

tabRect.Inflate(-2, -2);

//"x" image

Rectangle imageRect = new Rectangle(tabRect.Right - CloseImage.Width,

tabRect.Top, CloseImage.Width, CloseImage.Height);

//draw the string text

e.Graphics.DrawString(tabControl.TabPages[e.Index].Text,

tabControl.Font, Brushes.Black, tabRect);

//draw the "x" image

e.Graphics.DrawImage(CloseImage, imageRect.Location);

}

* Kéo thả các tab page

/// <summary>

/// Allow DoDragDrop on Mouse Move

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private static void tabControl\_MouseMove(object sender, MouseEventArgs e)

{

//if user click left mouse

if ((e.Button != MouseButtons.Left)) return;

// start drag and drop

tabControl.DoDragDrop(tabControl.SelectedTab, DragDropEffects.All);

}

/// <summary>

/// switch tag when drag around

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private static void tabControl\_DragOver(object sender, DragEventArgs e)

{

// if there is no tab being draged

if (e.Data.GetData(typeof(TabPage)) == null) return;

//get the tab page being drag

TabPage dragTab = (TabPage)e.Data.GetData(typeof(TabPage));

//get the index of the tab being draged

int dragTab\_index = tabControl.TabPages.IndexOf(dragTab);

// get the index of tab page having the mouse on it

int hoverTab\_index = getHoverTabIndex();

//if not, return

if (hoverTab\_index < 0) { e.Effect = DragDropEffects.None; return; }

//get the tab having the mouse on it

TabPage hoverTab = tabControl.TabPages[hoverTab\_index];

//set the effect when mouse move around

//(just to make it look like we are draging something)

//this is not very important

e.Effect = DragDropEffects.Move;

// if drag tab and hover tab are the same, just return

if (dragTab == hoverTab) return;

// swap dragTab & hoverTab - avoids toggeling

Rectangle dragTabRect = tabControl.GetTabRect(dragTab\_index);

Rectangle hoverTabRect = tabControl.GetTabRect(hoverTab\_index);

//some calculations

if (dragTabRect.Width < hoverTabRect.Width)

{

Point tcLocation = tabControl.PointToScreen(tabControl.Location);

if (dragTab\_index < hoverTab\_index)

{

if ((e.X - tcLocation.X) >

((hoverTabRect.X + hoverTabRect.Width) - dragTabRect.Width))

swapTabPages(dragTab, hoverTab);

}

else

{

if ((e.X - tcLocation.X) < (hoverTabRect.X + dragTabRect.Width))

swapTabPages(dragTab, hoverTab);

}

}

else

{

swapTabPages(dragTab, hoverTab);

}

//set the selected tab by draged tab

//this is very important

tabControl.SelectedIndex = tabControl.TabPages.IndexOf(dragTab);

}

/// <summary>

/// get the index of tab having the mouse on it

/// </summary>

/// <param name="tc"></param>

/// <returns></returns>

private static int getHoverTabIndex()

{

for (int i = 0; i < tabControl.TabPages.Count; i++)

{

if(tabControl.GetTabRect(i).Contains(

tabControl.PointToClient(Cursor.Position)))

return i;

}

return -1;

}

/// <summary>

/// swap tab pages

/// </summary>

/// <param name="tc"></param>

/// <param name="src"></param>

/// <param name="dst"></param>

private static void swapTabPages(TabPage src, TabPage dst)

{

//swap

int index\_src = tabControl.TabPages.IndexOf(src);

int index\_dst = tabControl.TabPages.IndexOf(dst);

tabControl.TabPages[index\_dst] = src;

tabControl.TabPages[index\_src] = dst;

}

* Tắt tab khi nhấn dấu “x”

/// <summary>

/// when user click "x" image

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private static void tabControl\_MouseDown(object sender, MouseEventArgs e)

{

//this for loop check if user click mouse at the position of an "x" image

//loop through all tab page to demonstrate which tab tage is being clicked

foreach (TabPage tabPage in tabControl.TabPages)

{

//get the shape of the tab

Rectangle tabRect =

tabControl.GetTabRect(tabControl.TabPages.IndexOf(tabPage));

//inflate

tabRect.Inflate(-2, -2);

//get the shape of the "x" image

Rectangle imageRect = new Rectangle(tabRect.Right - CloseImage.Width,

tabRect.Top, CloseImage.Width, CloseImage.Height);

//if mouse click in the shape of "x" image

if (imageRect.Contains(e.Location))

{

//remove this tab

//if this is the last tab page remaining, don't close it

if (tabControl.TabPages.Count > 1)

{

//remove tab page

tabControl.TabPages.Remove(tabPage);

tabControl.SelectedTab = 0;

}

break;

}

}

}

### 7. Compile C# và Visual Basic

* CodeDomProvider có sẵn có thể được sử dụng để compile C# và Visual Basic.
* Ở code dưới đây, language là ngôn ngữ được sử dụng để compile (C# hoặc Visual Basic). mainClassTextBox chứa tên class chính.

if (CodeDomProvider.IsDefinedLanguage(language) == false)

{

MessageBox.Show("This laguage might haven't been supported","Something went

wrong");

return;

}

CodeDomProvider compiler = CodeDomProvider.CreateProvider(language);

CompilerParameters parameters = new CompilerParameters();

parameters.GenerateExecutable = true;

parameters.OutputAssembly = "Application.exe";

parameters.MainClass = mainClassTextBox.Text;

//Include debug info

parameters.IncludeDebugInformation = true;

// Add available assemblies - this should be enough for the simplest

// applications.

foreach (Assembly asm in AppDomain.CurrentDomain.GetAssemblies())

{

parameters.ReferencedAssemblies.Add(asm.Location);

}

String code = textArea.Text;

CompilerResults results = compiler.CompileAssemblyFromSource(parameters, code);

if (results.Errors.Count > 0)

{

string errors = "Compilation failed:\n";

foreach (CompilerError err in results.Errors)

{

errors += err.ToString() + "\n";

}

MessageBox.Show(this, errors, "There were compilation errors");

}

else

{

// try to execute application

try

{

if (!System.IO.File.Exists("Application.exe"))

{

MessageBox.Show(String.Format("Can't find {0}",

"Application.exe"), "Can't execute.");

return;

}

ProcessStartInfo pInfo = new ProcessStartInfo("Application.exe");

Process.Start(pInfo);

}

catch (Exception ex)

{

MessageBox.Show(String.Format("Error while executing {0}", "Application.exe") + ex.ToString(), "Can't execute.");

}

}

### Code folding

* Code folding sử dụng Regex để tìm kiếm vị trí mở đầu và kết thúc của đoạn code. Regex được dùng cho ngôn ngữ C, C++, C# là ((\{) | (\})) – tìm đoạn text có ký tự là “{“ hoặc “}”
* Thuật toán cơ bản:
  + Tạo 1 stack lưu vị trí của ký tự “{“
  + Dùng regex scan đoạn text từ trên xuống dưới, tương ứng ứng mỗi match:
    - Nếu là “{“, them vị trí của “{“ của stack.
    - Nếu là “}”, pop stack được vị trí của “{“, đoạn code nằm giữa vị trí “{“ và “}” là đoạn code có thể gập. Thêm vào danh sách

## IV. KHÁC

### 1. Thuận lợi và khó khăn

### 2. Phân công công việc

### 3. Tài liệu tham khảo

<https://github.com/jacobslusser/ScintillaNET>

<https://www.codeproject.com/articles/161871/fast-colored-textbox-for-syntax-highlighting>

<https://www.codeproject.com/Articles/1931/C-Compiler>

<https://www.codeproject.com/Articles/1106140/VisualStudio-Like-TabControl>

(jacobslusser)

jacobslusser. (n.d.). Retrieved from GitHub: https://github.com/jacobslusser/ScintillaNET

Và một vài nguồn tài liệu khác.