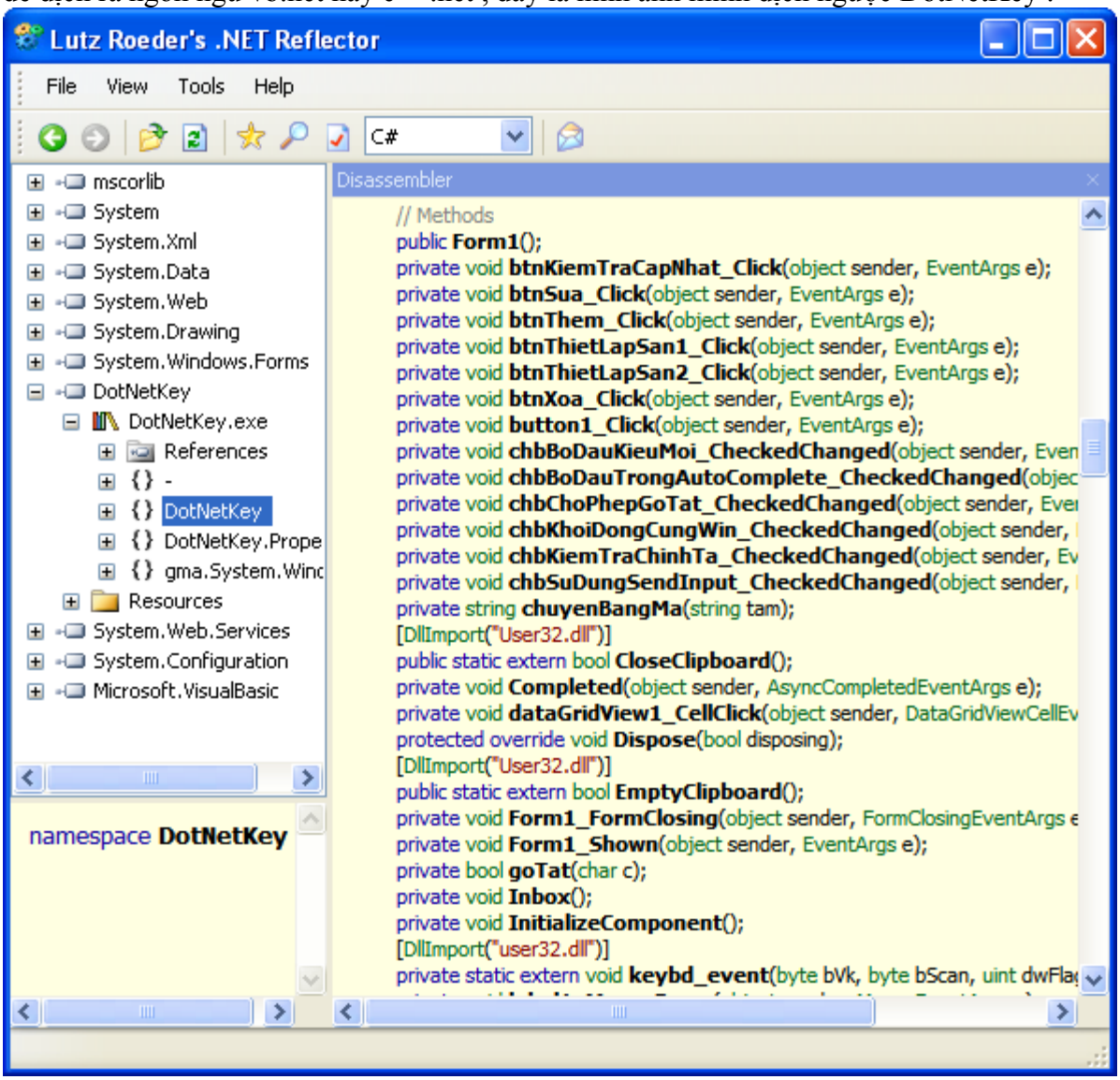
Hướng dẫn căn bản làm bộ gõ Tiếng Việt

(Tạo ebook bằng openoffice xuất ra pdf)

Mở đầu

Bộ gõ tiếng việt là 1 đề tài không quá khó nếu chỉ làm đơn giản , ví dụ nếu chỉ làm bỏ dấu và mũ ngay sau nguyên âm thì phải nói là rất dễ, nhưng bỏ dấu cuối từ, chỉnh vị trí dấu , check chính tả thì vấn đề lớn đó, tùy theo trình độ , và quyết tâm hay mục đích mỗi người mà sẽ phát triển ứng dụng này đến mức tương ứng . Bộ gõ mình làm là DotNetKey , bạn có thể xem chương trình ở đây (mã nguồn có thể dùng reflector để dịch ngược file exe) , lập trình bằng c# 2005 , bạn cũng có thể dùng chương trình này để dịch ra ngôn ngữ vb.net hay c++.net , đây là hình ảnh mình dịch ngược DotNetKey :



http://dotnetkey.googlecode.com/

Đây là hướng dẫn căn bản , mình không hướng dẫn tỉ mỉ đến mức chi tiết thuật toán , vì mỗi người lập trình 1 kiểu riêng , thuật toán riêng.

DotNetKey không công bố mã nguồn trực tiếp vì nhiều lý do ,nhưng chủ yếu là muốn hướng dẫn

những người có đủ trình độ khả năng phát triển mã nguồn , chứ không phải những người chỉ biết build source cho sẵn (thậm chí không biết dùng cả reflector) , chỉnh sửa một tí hoặc vặt bớt nội dung để ra 1 phiên bản khác nhằm phục vụ mục đích mang tính cá nhân , không muốn bỏ công sức nghiên cứu và không mang tính nâng cao , phát triển , đóng góp cho cộng đồng gì cả .

Nhận ký tự gõ vào và xuất tiếng việt

Nhận ký tự gõ

Để nhận ký tự gõ bạn đương nhiên phải nghiên cứu kỹ thuật hook rồi , đối với các ngôn ngữ khác như thế nào không biết nhưng với c# , mình tìm được thư viện hook sẵn trên trang codeproject :

http://www.codeproject.com/KB/cs/globalhook.aspx

Nên mình cũng không có nghiên cứu hook nữa (^_^) , đó chính là phần namespace **gma.System.Windows** trong mã nguồn của mình , cách dùng như sau:

1 . khai báo using gma.System.Windows
2 .Tạo đối tượng , handle event sau đó sử dụng y hệt như các sự kiệu key và mouse trong form là xong như ví dụ sau

```
UserActivityHook actHook;
void MainFormLoad(object sender, System.EventArgs e)
{
actHook= new UserActivityHook(); // crate an instance
// hang on events
actHook.OnMouseActivity+=new MouseEventHandler(MouseMoved); actHook.KeyDown+=new
KeyEventHandler(MyKeyDown); actHook.KeyPress+=new KeyPressEventHandler(MyKeyPress);
actHook.KeyUp+=new KeyEventHandler(MyKeyUp);
}
```

3 . Now, an example of how to process an event:

```
public void MouseMoved(object sender, MouseEventArgs e)
{
labelMousePosition.Text=String.Format("x={0}  y={1}", e.X, e.Y);
if (e.Clicks>0) LogWrite("MouseButton - " + e.Button.ToString());
}
void MyKeyPress(object sender, KeyPressEventArgs e)
{
}
void MyKeyDown(object sender, KeyEventArgs e)
{
}
void MyKeyUp(object sender, KeyEventArgs e)
{
}
```

**Một lưu ý là để build được phải vào property của project ,
chọn tab debug và bỏ chọn "Enable the Visual Studios hosting process"**

Thư viện này chạy rất tốt , nhưng có 1 yếu điểm đó chính là khi đang hook mà bấm nút tắt của cửa sổ thì nó bị đơ mấy giây (^_^) , nên theo mình chúng ta nên chỉnh thuộc tính của form formboderstyle =

none (không có thanh cửa sổ) mà tạo menu giả như DotNetKey của mình .

Hiện nay thư viện này đã có version mới (xem trang web codeproject) có thể add vào toolbar trực tiếp rồi dùng như các control button với các event được tạo bằng cách click trên thanh property. Và không cần bỏ chọn cái tab debug của project nữa

Ngoài ra , mình còn tìm được 1 số đoạn code mouse hook và keyboard hook ở đây nữa :

http://blogs.msdn.com/toub/archive/2006/05/03/589423.aspx
http://blogs.msdn.com/toub/archive/2006/05/03/589468.aspx

Đây là code Keyboard hook:

```
using System;
using System.Diagnostics;
using System.Windows.Forms;
using System.Runtime.InteropServices;

class InterceptKeys
{
private const int WH_KEYBOARD_LL = 13;
private const int WM_KEYDOWN = 0x0100;
private static LowLevelKeyboardProc _proc = HookCallback;
private static IntPtr _hookID = IntPtr.Zero;

public static void Main()
{
_hookID = SetHook(_proc); Application.Run(); UnhookWindowsHookEx(_hookID);
}

private static IntPtr SetHook(LowLevelKeyboardProc proc)
{
using (Process curProcess = Process.GetCurrentProcess())
using (ProcessModule curModule = curProcess.MainModule)
{
return SetWindowsHookEx(WH_KEYBOARD_LL, proc,
GetModuleHandle(curModule.ModuleName), 0);
}
}

private delegate IntPtr LowLevelKeyboardProc(
int nCode, IntPtr wParam, IntPtr lParam);

private static IntPtr HookCallback(
int nCode, IntPtr wParam, IntPtr lParam)
{
if (nCode >= 0 && wParam == (IntPtr)WM_KEYDOWN)
{
int vkCode = Marshal.ReadInt32(lParam); Console.WriteLine((Keys)vkCode);
}
return CallNextHookEx(_hookID, nCode, wParam, lParam);
}

[DllImport("user32.dll", CharSet = CharSet.Auto, SetLastError = true)]
private static extern IntPtr SetWindowsHookEx(int idHook, LowLevelKeyboardProc
lpfn, IntPtr hMod, uint dwThreadId);

[DllImport("user32.dll", CharSet = CharSet.Auto, SetLastError = true)] [return:
MarshalAs(UnmanagedType.Bool)]
private static extern bool UnhookWindowsHookEx(IntPtr hhk);
```

```
[DllImport("user32.dll", CharSet = CharSet.Auto, SetLastError = true)]
private static extern IntPtr CallNextHookEx(IntPtr hhk, int nCode, IntPtr wParam,
IntPtr lParam);

[DllImport("kernel32.dll", CharSet = CharSet.Auto, SetLastError = true)]
private static extern IntPtr GetModuleHandle(string lpModuleName);
}
```

MouseHook

```
class InterceptMouse
{
private static LowLevelMouseProc _proc = HookCallback;
private static IntPtr _hookID = IntPtr.Zero;

public static void Main()
{
_hookID = SetHook(_proc); Application.Run(); UnhookWindowsHookEx(_hookID);
}

private static IntPtr SetHook(LowLevelMouseProc proc)
{
using (Process curProcess = Process.GetCurrentProcess())
using (ProcessModule curModule = curProcess.MainModule)
{
return SetWindowsHookEx(WH_MOUSE_LL, proc, GetModuleHandle(curModule.ModuleName),
0);
}
}

private delegate IntPtr LowLevelMouseProc(int nCode, IntPtr wParam, IntPtr lParam);

private static IntPtr HookCallback(
int nCode, IntPtr wParam, IntPtr lParam)
{
if (nCode >= 0 &&
MouseMessages.WM_LBUTTONDOWN == (MouseMessages)wParam)
{
MSLLHOOKSTRUCT hookStruct =
(MSLLHOOKSTRUCT)Marshal.PtrToStructure(lParam, typeof(MSLLHOOKSTRUCT));
Console.WriteLine(hookStruct.pt.x + ", " + hookStruct.pt.y);
}
return CallNextHookEx(_hookID, nCode, wParam, lParam);
}

private const int WH_MOUSE_LL = 14;

private enum MouseMessages
{
WM_LBUTTONDOWN = 0x0201, WM_LBUTTONUP = 0x0202, WM_MOUSEMOVE = 0x0200,
WM_MOUSEWHEEL = 0x020A, WM_RBUTTONDOWN = 0x0204, WM_RBUTTONUP = 0x0205
}

[StructLayout(LayoutKind.Sequential)]
private struct POINT
{
public int x;
public int y;
}
```

```
[StructLayout(LayoutKind.Sequential)]
private struct MSLLHOOKSTRUCT
{
public POINT pt;
public uint mouseData; public uint flags; public uint time;
public IntPtr dwExtraInfo;
}

[DllImport("user32.dll", CharSet = CharSet.Auto, SetLastError = true)]
private static extern IntPtr SetWindowsHookEx(int idHook,
LowLevelMouseProc lpfn, IntPtr hMod, uint dwThreadId);

[DllImport("user32.dll", CharSet = CharSet.Auto, SetLastError = true)] [return:
MarshalAs(UnmanagedType.Bool)]
private static extern bool UnhookWindowsHookEx(IntPtr hhk);

[DllImport("user32.dll", CharSet = CharSet.Auto, SetLastError = true)]
private static extern IntPtr CallNextHookEx(IntPtr hhk, int nCode,
IntPtr wParam, IntPtr lParam);

[DllImport("kernel32.dll", CharSet = CharSet.Auto, SetLastError = true)]
private static extern IntPtr GetModuleHandle(string lpModuleName);
}
```

Xuất tiếng việt sử dụng clipboard

Xuất tiếng việt, cách thức unikey hay winvnkey làm là senkey , để gõ , nhưng dường như nó không được hữu hiệu lắm trên vista , từ ý tưởng unikey dùng clipboard cho unicode để gõ trong 1 số ứng dụng như pidgin hay vietkey thì dung clipboard trong hầu hết ứng dụng thì mình quyết định sử dụng nó để send tiếng việt.

Giả lập phím và tổ hợp phím

Trong c# có hàm sendkey rất dễ dùng , nhưng đáng tiếc vista 64 nó không hoạt động , vậy phải dùng hàm api keybd_event , trong c# bạn làm như sau , khai báo hàm này như 1 biến toàn cục vậy:

```
[DllImport("user32.dll")]
static extern void keybd_event(byte bVk, byte bScan, uint dwFlags, UIntPtr dwExtraInfo);
```

Cơ chế của việc thay thế tiếng việt ví dụ như từ nhà là khi gõ nha-f , ta giả lập phím back để xóa chữ a, set clipboard chữ à , rồi giả lập phím Shift+Insert (một phím nóng để paste giống như Ctrl+V):

Để send 1 phím back chúng ta dùng lệnh như sau :

```
keybd_event((byte)Keys.Back, 0, 0, UIntPtr.Zero);//press
keybd_event((byte)Keys.Back, 0, 2, UintPtr.Zero);//release
```

Còn đây là shift + insert :

```
keybd_event(0x10, 0, 0, UIntPtr.Zero);
keybd_event(0x2D, 0, 1, UIntPtr.Zero); // không biết vì sao phải thế nhưng thấy có tag 0 và 2 nên chắc có 1 , tích bừa vào thế mà ra :D
keybd_event(0x2D, 0, 2, UIntPtr.Zero);
```

```
keybd_event(0x10, 0, 2, UIntPtr.Zero);
```

còn đây là Ctrl+V , ctrl + V thì phức tạp hơn do nếu người dùng gõ kiểu telex ví dụ AS thì lúc đó người dùng nhấn phím shift , khi đó tổ hợp phím để xuất tiếng việt sẽ là Ctrl+shift+ V (sẽ không ra text đâu), vì vậy phải có if là else như dưới đây , nhưng bù lại nó có khả năng gõ được trên 1 số ứng dụng flash như chat room : http://xat.com hay https://www.meebo.com/rooms/create/ hay trang chat trên web như http://webmessenger.yahoo.com/ … nói chung là các text trong flash

```
if (shift == false)
{
keybd_event(0x11, 0, 0, UIntPtr.Zero);
keybd_event((int)Keys.V, 0, 0, UIntPtr.Zero);
keybd_event((int)Keys.V, 0, 2, UIntPtr.Zero);
keybd_event(0x11, 0, 2, UIntPtr.Zero);
}
else
{
keybd_event(0x10, 0, 2, UintPtr.Zero);// thoát phím shift
keybd_event(0x11, 0, 0, UIntPtr.Zero);
keybd_event((int)Keys.V, 0, 0, UIntPtr.Zero);
keybd_event((int)Keys.V, 0, 2, UIntPtr.Zero);
keybd_event(0x11, 0, 2, UIntPtr.Zero);
keybd_event(0x10, 0, 0, UintPtr.Zero); // nhấn lại phím shift
}
```

ngoài ra , khi người dùng nhấn shift bên phải cần e.handle đi và send lại phím shift trái trong sự kiện keydown vì không hiểu sao khi mình giả lập phím shift phải như trên thì lại không có tác dụng nên phải cho nó là 2 shift trái hết , **nhớ set cả trong sự kiện key up với shift right nữa (vậy thì phím shift**
phải giờ sẽ không còn mà sẽ là 2 phím shift trái trên bàn phím)

```
if (e.KeyData.ToString() == "RShiftKey")
{
e.Handled = true;
keybd_event(0x10, 0, 0, UIntPtr.Zero);
}
```

Còn cách khác không bị đổi phím shift là khi người dùng ấn shift phải ta có thể dùng shift + Insert ở trên , vậy chỉ khi người dùng gõ trên ứng dụng flash tiếng việt mà phải nhấn Ctrl mới không gõ được (rất hiếm khi đụng vào , mà ứng dụng flash cũng không quan trọng mấy , khi gõ Á kiểu telex có thể thay AS thành As). Vậy có 2 cách cho các bạn chọn mỗi cách phải bỏ đi 1 tí , từ dotnekey 3.7 trở đi mình chọn cách 2.

Gán giá trị cho clipboard

nếu ta setClipboard bằng lệnh Clipboard.setText() có sẵn trong thư viện của framework thì cũng được thôi , nó sẽ chạy tốt trong nhiều các trường hợp , nhưng nếu dùng nó với flashget (có bật chế độ monitor clipboard) hay teamviewer ….nói chung là các trình tương tác clipboard thì sẽ gặp lỗi nguyên nhân lỗi thông báo là : Requested Clipboard operation did not succeed. , thành ra chúng ta phải sử dụng 1 số hàm api để tương tác với Clipboard như sau (thực ra dùng có 4 hàm nhưng cứ khai báo cho nó đủ):

```
[DllImport("User32.dll")]
public static extern bool OpenClipboard(IntPtr hWnd); [DllImport("User32.dll")]
public static extern IntPtr SetClipboardData(CF Format, IntPtr hMem);
[DllImport("User32.dll")]
public static extern bool EmptyClipboard(); [DllImport("User32.dll")]
public static extern IntPtr GetClipboardData(CF Format);
[DllImport("User32.dll")]
public static extern bool CloseClipboard();

[DllImport("Kernel32.dll", CharSet = CharSet.Auto)] public static extern bool CloseHandle(IntPtr hObject);
[DllImport("Kernel32.dll", CharSet = CharSet.Auto)] public static extern IntPtr GlobalSize(IntPtr hMem);

public enum CF
{
Text = 1, Bitmap = 2, MetaFilePict = 3, Sylk = 4, Dif = 5, Tiff = 6, OemText = 7, Dib = 8, Palette = 9, Pendata = 10, Riff =
11, Wave = 12, UnicodeText = 13, EnhMetaFile = 14, HDrop = 15, Locale = 16, Dibv5 = 17, OwnerDisplay = 128, DspText
= 129, DspBitmap = 130, DspMetaFilePict = 131, DspEnhMetaFile = 142, PrivateFirst = 512, PrivateLast = 767,
GdiObjFirst = 768, GdiObjLast = 1023
}

private void SetClipboardAPI(string s)
{
OpenClipboard(this.Handle); EmptyClipboard();
SetClipboardData(CF.UnicodeText, Marshal.StringToCoTaskMemUni(s));
CloseClipboard();
}
```

Xuất dữ liệu sử dụng sendInput

Tìm được trên google :
http://forums.microsoft.com/msdn/ShowPost.aspx?PostID=2862633&SiteID=1

Nhưng bạn nhớ là ở window 64 bit thì hàm const cross ở dưới phải là 8 đấy nhé , nên phải tách làm 2 hàm sendInput cho 32 và 64 bit

```
[DllImport("user32.dll", EntryPoint = "SendInput", SetLastError = true)] static extern uint SendInput(uint nInputs,
INPUT[] pInputs, int cbSize); [DllImport("user32.dll", EntryPoint = "SendInput", SetLastError = true)] static extern uint
SendInput(uint nInputs, INPUT64[] pInputs, int cbSize);

const int cross = 4, cross64 = 8; // cái này là 8 nếu dùng cho win 64 bit private enum InputType
{
INPUT_MOUSE = 0, INPUT_KEYBOARD = 1, INPUT_HARDWARE = 2,
}
[Flags()]
private enum MOUSEEVENTF
{
MOVE = 0x0001, // mouse move LEFTDOWN = 0x0002, // left button down LEFTUP = 0x0004, // left button up
RIGHTDOWN = 0x0008, // right button down RIGHTUP = 0x0010, // right button up
MIDDLEDOWN = 0x0020, // middle button down
MIDDLEUP = 0x0040, // middle button up
XDOWN = 0x0080, // x button down
XUP = 0x0100, // x button down
WHEEL = 0x0800, // wheel button rolled
VIRTUALDESK = 0x4000, // map to entire virtual desktop
ABSOLUTE = 0x8000, // absolute move
```

```
}

[Flags()]
private enum KEYEVENTF
{
EXTENDEDKEY = 0x0001, KEYUP = 0x0002, UNICODE = 0x0004, SCANCODE = 0x0008,
}

[StructLayout(LayoutKind.Sequential)]
private struct MOUSEINPUT
{
public int dx;
public int dy;
public int mouseData; public int dwFlags; public int time;
public IntPtr dwExtraInfo;
}

[StructLayout(LayoutKind.Sequential)]
private struct KEYBDINPUT
{
public short wVk; public short wScan; public int dwFlags; public int time;
public IntPtr dwExtraInfo;
}

[StructLayout(LayoutKind.Sequential)]
private struct HARDWAREINPUT
{
public int uMsg;
public short wParamL;
public short wParamH;
}

[StructLayout(LayoutKind.Explicit)]
private struct INPUT
{
[FieldOffset(0)] public int type; [FieldOffset(cross)]
public MOUSEINPUT mi; [FieldOffset(cross)]
public KEYBDINPUT ki; [FieldOffset(cross)]
public HARDWAREINPUT hi;
}
[StructLayout(LayoutKind.Explicit)]
private struct INPUT64
{
[FieldOffset(0)] public int type; [FieldOffset(cross64)] public MOUSEINPUT mi; [FieldOffset(cross64)] public
KEYBDINPUT ki; [FieldOffset(cross64)]
public HARDWAREINPUT hi;
}
private void sendUnicode(char c)
{
if (IntPtr.Size == 4) // nếu là 32 bit
{
INPUT input_down = new INPUT();
input_down.type = (int)InputType.INPUT_KEYBOARD;
input_down.ki.wVk = 0;
input_down.ki.wScan = (short)c;
input_down.ki.dwFlags = (int)KEYEVENTF.UNICODE; INPUT input_up = input_down;
input_up.ki.dwFlags = (int)(KEYEVENTF.KEYUP | KEYEVENTF.UNICODE); INPUT[] input = { input_down, input_up
};
```

```
SendInput(2, input, Marshal.SizeOf(input_down));
}
else
{ // dùng cho win 64 bit
INPUT64 input_down = new INPUT64(); input_down.type = (int)InputType.INPUT_KEYBOARD; input_down.ki.wVk =
0;
input_down.ki.wScan = (short)c;
input_down.ki.dwFlags = (int)KEYEVENTF.UNICODE; INPUT64 input_up = input_down;
input_up.ki.dwFlags = (int)(KEYEVENTF.KEYUP | KEYEVENTF.UNICODE); INPUT64[] input = { input_down,
input_up };
SendInput(2, input, Marshal.SizeOf(input_down));
}
}
private void sendAnsi(Keys k)
{
if (IntPtr.Size == 4)
{
INPUT input_down = new INPUT();
input_down.type = (int)InputType.INPUT_KEYBOARD;
input_down.ki.wVk = (short)k; INPUT input_up = input_down; input_up.ki.dwFlags = (int)KEYEVENTF.KEYUP;
INPUT[] input = { input_down, input_up }; SendInput(2, input, Marshal.SizeOf(input_down));
}
else
{
INPUT64 input_down = new INPUT64(); input_down.type = (int)InputType.INPUT_KEYBOARD; input_down.ki.wVk =
(short)k;
INPUT64 input_up = input_down; input_up.ki.dwFlags = (int)KEYEVENTF.KEYUP; INPUT64[] input = { input_down,
input_up }; SendInput(2, input, Marshal.SizeOf(input_down));
}
}
```

Hủy lệnh gõ 1 phím

Cuối cùng , như trên đã thấy gõ chữ f để thành dấu huyền , nên ta phải hủy lệnh gõ phím f (dấu huyền) nếu không muốn nó xuất hiện ngay đằng sau chữ tiếng việt kiểu như : “nhàf” bằng code như sau :

```
void MyKeyPress(object sender, KeyPressEventArgs e)
{
......
e.Handled = true;
....
}
```

Các nguyên tắc xử lý tiếng việt DotNetKey dùng

Nguyên tắc cơ bản trong xử lý tiếng việt của DotNetKey

Đầu tiên xây dựng 1 class convert mã tiếng việt , như mình tạo class TiengViet , hàm tạo là nhập bảng mã và kiểu gõ vào, rồi có method convert, ta dùng hook ở trên hướng dẫn bắt các ký tự gõ , cứ mỗi lần người dùng gõ , ta lại convert, nếu convert được ra tiếng việt thì xuất tiếng việt như ở trên đã hướng dẫn . Khi kết thúc từ (các ký tự không trong bảng chữ cái và kiểu gõ) thì gán string lưu ký tự gõ ="" để xử lý các ký tự gõ tiếp theo . Bạn cũng có thể làm thêm 1 string khác lưu lại toàn bộ các ký tự gõ trong phạm vi khoảng 100 từ chẳng hạn , để khi xóa từ vẫn có thể bỏ được dấu ví dụ gõ từ Việt nam, xóa hết chữ Nam , đến chữ Việt rồi gõ dấu sắc, vẫn thành chữ Viết được .
Đây là cách tớ lưu trữ dữ liệu 1 từ , các bước xử lý đều dựa vào struct này để xử lý , việc nhận ký tự gõ trong phạm vi 1 từ đều là xử lý để tống vào struct này
Đầu tiên , để xử lý tiếng việt chúng ta phải mô tả được các thành phần của 1 chữ tiếng việt , tớ mô tả bằng 1 struct (hay record trong pascal delphi gì đó) , chơi class cũng được nhưng struct truy cập tốc độ cao hơn.
đây là các thành phần tớ phân ra :
Trích dẫn:
+string Âm đầu và các thành phần đi theo nó :
- kiểu boolean (true / false) là 1 biến xác định âm đầu có phải là chữ đ không
+string Nguyên âm :
- enum dấu (khongdau, sac, hoi, nang , nga , huyen) ;
- vị trí dấu ;
- enum móc mũ (ă , â , ư , không móc) ;
- trùng dấu (khi gõ đúp)
- bool ư gõ ở đầu (để xác định không nhận nhầm là trùng dấu khi gõ tru7o7ng , truwowng)
+string Âm cuối

Định nghĩa kiểu gõ

Giờ tiếp bài về kiểu gõ , tổng hợp 3 kiểu telex , vni , viqr thì chúng ta có thể định nghĩa các kiểu gõ gồm các thành phần sau :
Mã:
```
enum DinhNghiaKieuGo
{
khongDau, sac, huyen, nga, hoi, nang, DauMuChungAOE, AThanh6, EThanh6, OThanh6,
UThanh7OThanh7AThanh8, UOASimple, UThanh7OThanh7, AThanh8, DThanhD
};
```

Mấy cái định nghĩa trên dựa theo kiểu gõ VNI ví dụ AThanh8 (tức là ă đó) , cái nào mọi người không hiểu thì đối chiếu mấy định nghĩa kiểu gõ bên dưới lên enum này .
Đủ dùng rồi đó , tuy không nhiều như unikey nhưng định nghĩa ít , xử lý ít , lỗi ít
Giờ để sử dụng nó ví dụ nạp các kiểu gõ

Mã:

char[] vni ={ '0', '1', '2', '4', '3', '5', '6', ' ', ' ', ' ', ' ',' ', '7', '8', '9' }, telex ={ 'z', 's', 'f', 'x', 'r', 'j', ' ', 'a', 'e', 'o', 'w',' ', ' ', ' ', 'd' }, telexSimple={ 'z', 's', 'f', 'x', 'r', 'j', ' ', 'a', 'e', 'o', ' ','w', ' ', ' ', 'd' }, VIQR ={ '0', '\'', '`', '~', '?', '.', '^', ' ', ' ', ' ', ' ', ' ', '+', '(', 'd' };

Ngoài ra còn 2 phím gõ chung cho các kiểu gõ trên là Ctrl để ngắt tức thời phím ví dụ tiếngviệt (gõ liền có thể gõ tiếng - Ctrl - Việt) cả vietkey và unikey đều có
Và 1 tổ hợp phím để hoàn lại nguyên thể các ký tự vừa gõ để tạo từ Shift +space (hay gì đó thì tùy) , unikey thì dùng shift trái phải nhưng mình thấy space dễ bấm hơn

telex khác telexsimple ở chỗ telex gõ w có thể thành ư luôn
Kiểu gõ telex mở rộng trong Unikey có bổ xung [] {} thành ươ và ƯƠ gì đó nhưng mình thấy thật không cần thiết vì có mấy phân tích sau :
- vì nó xung đột khi gõ phần tử mảng và đánh dấu đoạn code cho coder mà ta gõ wo cũng ra ươ được
- ngoài ra gõ WO có thể ra ƯƠ , Wo , wO rất linh hoạt.
- Kiểu gõ này chỉ thích hợp với gõ mổ cò còn gõ 10 ngón thì dùng [] ta phải dời tay khỏi chỗ 2 phím đánh dấu 2 ngón trỏ , gõ xong lại chuyển lại gây mất thời gian

Nguyên tắc Kiểm tra chính tả Của DotNetKey

Bài viết dưới đây sưu tầm trên mạng , thấy thực ra cũng có phần chưa ổn lắm ví dụ như sơ đồ từ uêp chả biết ở đâu ra , để bắt từ chính xác hơn mình chỉ chia làm 2 phần là âm đầu và vần . Vần mình liệt kê dựa vào cái hình phía dưới , nhưng lược các vần vô lý như uêp ở trên . Ghép với phần âm đầu :

```
string[] amDau = { "r", "d", "gi", "v", "ch", "tr", "s", "x", "l", "n", "qu", "b", "`c", "/k", "`g", "/gh", "h", "kh", "m", "`ng", "/ngh", "nh", "p", "ph", "t", "th" }
```

Trong đó /gh tức là nó chỉ đi với i , e ; “`g” tức là nó không đi với i e .

```
string[] van ={....  "uom", "^", "uom", "*" …. }
```

vẫn chưa liệt kê đủ hết nên chưa viết vào đây (muốn lấy list van mới nhất của tớ thì dịch ngược mã nguồn ra xem , đại thể như sau , 1 vần có 2 string thông tin đi liền nhau "uom", "*" là đi kèm có dấu móc ư , ơ tức là ươm ; có dấu mũ ^ tức là uôm …. (gồm 4 loại dấu móc là : không dấu”-” , móc ư ơ “*”, mũ ô , â, ê “^” ; ă “(“ .

Các bạn dựa vào struct tớ mô tả ở trên để biết cách trích các thành phần của nó so sánh với mấy quy tắc chính tả này

Ngoài ra như bài viết dưới thì những phụ âm cuối : Các chữ kết thúc bằng c, ch, p, t buộc phải có 1 trong 2 dấu thanh sắc hay nặng, không thể không có dấu thanh hay có các dấu thanh hỏi, ngã, huyền. Còn đây là bài viết:

Âm và vần tiếng Việt

Hệ thống âm vị tiếng Việt (sưu tầm trên mạng) Tiếng Việt có:
- 11 nguyên âm đơn (monothong):
a, ă, â, e, ê, i, o, ô, ơ, u, ư,

Chữ y đứng một mình và chữ i đứng một mình là hai lối viết của cùng một nguyên âm, vì thế không tính y.

- 30 nhị trùng âm (diphthong):
ai, ao, au, ay, âu, ây, eo, êu, ia, iê, iu, oa, oă, oe, oi, ôi, ơi, ua, uâ, uê, ui, uô, ươ, uy, ưa, ưi, ươ, ưu, oo, ôô;

Hai âm sau có nhưng rất ít dùng: oo (cái soong), ôô (ốốc dôộc, tức là ốt dột nói giọng Quảng Bình);

- 12 tam trùng âm (triphthong):
iêu, oai, oay, uây, uôi, uyê, uyu, ươi, ươu, uya, oao, oeo;

yêu không khác gì iêu về âm nên không tính

- 15 phụ âm đơn (consonant):
b, c(k,qu), d, đ, g, h, l, m, n, p, r, s, t, v, x;

- 8 phụ âm kép:
ch, gi, kh, nh, ng, ph, th, tr. Tổng-cộng số âm và vần là 76.

- 6 thanh (accent)
không, huyền, hỏi, sắc, ngã, nặng.

Cộng là 82 âm vị (phoneme). Trong đó:

- 17 phụ âm chỉ đứng đầu:
b, d, đ, g, h, k, l, r, s, v, x, qu, gi, kh, ph, th, tr;

- 8 phụ âm có thể đứng đầu và cuối:
c, m, n, p, t, ch, nh, ng;

p thoạt đầu không phải phụ âm đầu nhưng khi du nhập các từ nước ngoài, nó được tính là phụ âm đầu.

- 10 nguyên âm không thể kết thúc chữ (nguyên âm lửng): ă, â, iê, oă, uâ, uô, oo, ôô, ươ, uyê;

- 28 nguyên âm luôn kết-thúc chữ, tức là không có âm gì có thể đứng sau chúng (nguyên âm cuối): ai, ao, au, ay, âu, ây, eo, êu, ia, iu, oi, ôi, ơi, ui, ưa, ưi, ưu, iêu, uôi, uyu, ươi, ươu, oai, oay, uây, uya, oeo, oao;

Một số luật do thói quen

Tôi ủng hộ thói quen, cho dù thói quen không hợp lý, nhưng khi nói tới lý thì tôi cũng không khư khư giữ thói quen. Những luật dưới đây theo tôi là không có cái lý nào cả, chỉ là do thói quen mà ra thôi. Khi xây dựng lý thuyết tôi đề nghị ta không áp dụng các luật này mà chỉ đưa vào sau cùng như một lựa chọn, ai muốn theo thói quen thì cứ theo, ai muốn duy lý thì cứ duy, và hai anh này không có gì phải choảng nhau cả.

- g và ng khi đứng liền trước e, ê, i viết là gh và ngh

- c, k, qu là một âm vị đồng nhất, theo thói quen, người ta viết là:

* c khi đứng trước:
a (ca hát),
ă (cắt đứt),
â (cất giấu), o (co quắp), ô (cô gái),
ơ (cơ khí), u (cu Tèo), ư (cư trú),
ai (cai quản), ao (cao xa), au (cau trầu),
ay (cay đắng), âu (câu cá),
ây (cây cối), oi (coi ngó), ôi (côi cút), ơi (cơi trầu), ua (cua ghẹ), ui (củi lửa), ưa (cưa gỗ),
ưi (khung cửi),
ươ (cương quyết), ưu (cưu mang), ươi (cười cợt), uôi (cuối cùng);

* k khi đứng trước:
e (kẻ thù), ê (hạt kê),
i hay y (kí sự, ký sự), eo (keo kiệt),
êu (kêu gào), ia (kia kìa),
iê (kiên quyết),
iu (kiu kíu - tiếng chó con kêu), iêu (kiêu sa);

* qu khi ứng trước:
oa (qua loa-> qu + oa),
oă (quăn tít -> qu + oăn), oe (que kem -> qu + oe), uy (vu quy -> qu + uy),
uơ (quơ quào -> qu + uơ), uô (tổ quốc -> qu + uốc), uê (quê hương -> qu + uê),
uâ (quân nhân -> qu + uân), oai (quai xách -> qu + oai), oay (quay tròn -> qu + oay), uây (quây quần -> qu + uây),
uyê (chim quyên -> qu + uyên),
uya (giày quya -> qu + uya - phiên âm tiếng Pháp cuir), oeo (chết queo -> qu + oeo)
oao (quơ quào -> qu + oào)

Luật viết lược chữ u hay chữ o khi gặp "qu + u..." hay "qu + o..." là cách giải thích cho các tranh cãi về qua = qu + a hay q + ua (hai cách này không đúng, "qua" cùng vần với "loa" trong "qua loa", thế nên qua = qu + oa.

Có thuyết cho rằng qu là một bán âm, là âm nằm giữa nguyên âm và phụ âm nghĩa là phát âm như nguyên âm nhưng phải đi kèm theo nguyên âm, thí dụ âm |j| trong “young” tiếng Anh.

Qu theo thuyết này là bán âm |kw| là những âm mà người Sài gòn hay nói thành "w": “Buồn quá à, cho em quậy với đi” thành “Buồng wá à, cho em wậy dới đi”. Cũng theo thuyết này, âm “d” của người Sài Gòn dùng khi phát âm “v” của người Bắc, “con vịt vui vẻ” thành “con dịch dui dẻ”, cũng là một bán âm.

- y và i : có thể thay vần y bằng vần i ở mọi nơi (nhưng không thể thay uy bằng ui). lý do-> lí do, ý kiến

-> í kiến

Thuý không thành thúi vì nó có vần uy, tương tự uy quyền không thành ui quiền vì uy có nguyên âm là uy, quyền có nguyên âm là uyê. Thật ra, uyê cũng có thể viết là uiê không xung đột gì cả, vì không có chữ Việt nào có uiê, tóm lại viết uiê hay uyê cũng được, ta nên theo mọi người mà viết uyê. Viết:

Bích Câu đâu nữa bóng chàng Uiên? Sông núi thô sơ bặt tiếng huiền
Có lẽ hồn ta không đẹp nữa
Nét thần thôi hoạ bức thiên duiên.có vẻ không “đẹp”.

Một thói quen khác cũng hoàn toàn vô căn cứ: viết bác sỹ thay vì bác sĩ, tức là thay i bằng y ngay cả khi i không nằm ở đầu chữ. Vài người nói là viết bác sỹ để phân biệt với “một bác hay sĩ diện”, thực ra hai chữ sĩ này là một ngay cả trong chữ Hán.

Có người cho rằng i phát âm khác y, nhưng trong tiếng Việt hai âm này không có gì phân biệt rõ ràng cả.

- yê và iê: yê không khác gì iê: rõ ràng tiên và yên cùng vần nhưng viết khác chỉ vì thói quen thay i bằng y ở đầu chữ thôi, không có cách giải thích nào cho thói quen này.

Tôi cũng thấy ngồi iên lặng nó kì kì, ngồi yên lặng thì đỡ kỳ hơn, nhưng quả thật tôi không thấy iên và yên khác nhau chỗ nào.

Tại sao viết “Tiên Yên” mà không viết “Tiên Iên”, trong khi rõ ràng hai tiếng này cùng một vần?

Thí dụ: “Viết thông cáo rồi niêm iết cho mọi người cùng biết, cần thiết cứ làm riết, cứ xiết chặt miết, rồi cũng iên cái nạn rửa tiền nó xảy ra triền miên”.

Chấp nhận các thói quen này tức là chấp nhận trong tiếng Việt một âm, một vần có thể có nhiều cách viết khác nhau.

Ghép âm cuối

8 âm cuối chia làm 3 nhóm:
- C1: Nhóm ng, c
- C2: Nhóm nh, ch
- C3: Nhóm m,n,p,t

Có 27 nguyên âm luôn kết-thúc chữ, tức là không có âm gì có thể đứng sau chúng (nguyên âm khép): ai, ao, au, ay, âu, ây, eo, êu, ia, iu, oi, ôi, ơi, ui, ưa, ưi, ưu, iêu, uôi, uyu, ươi, oai, oay, uây, uya, oeo, oao.

Có 23 nguyên âm có thể ghép thêm âm cuối, chia làm 3 nhóm
- N1 : â, e, o, ô, u, ư, ơ, ă, oă, oe, uâ, uô, uơ, ươ
- N2: ê, i, uê, uy, ua
- N3: a, oa, iê, uyê

Nhóm N1 chỉ ghép được với C1, C3 mà không ghép được với C2. Thật vậy, các “vần” sau không đọc được: ânh, âch, enh, ech, onh, och, ônh, ôch, unh, uch, ưnh, ưch, ơnh, ơch, ănh, ăch, oănh, oăch, oenh, oech, uânh, uâch, uônh, uôch, ươnh, ươch, ươnh, ươch.

Nhóm N2 chỉ ghép được với C2, C3 mà không ghép được với C1. Thật vậy, các “vần” sau không đọc được: ênh, êch, ing, ic, uênh, uêc, uyng, uyc, uang (phải là oang), uac (phải là oac).

Nhóm N3 có thể ghép được với C1, C2, C3. Tuy nhiên có một số vần hầu như chưa được xài tới trong tiếng Việt, dù chúng có thể đọc được: iênh, iêch, uyênh, uyêc, uyênh, uyêch, uyêm, uyêp.

Ghép âm đầu

Gần như mọi phụ âm đều có thể đứng trước mọi nguyên âm, trừ hai trường hợp nghi ngờ sau:

- Nguyên âm uơ chỉ thấy đi với: qu (quơ quào= qu + uơ), h (huơ tay), th (thuở ấy), kh (khuơ tay)

Các trường hợp khác trông không "hạp nhãn" tí nào: buơ, duơ, đuơ, guơ, luơ, muơ, nuơ, puơ, ruơ, suơ, tuơ, vuơ, xuơ, chuơ, giuơ, nhuơ, nguơ, phuơ, truơ;

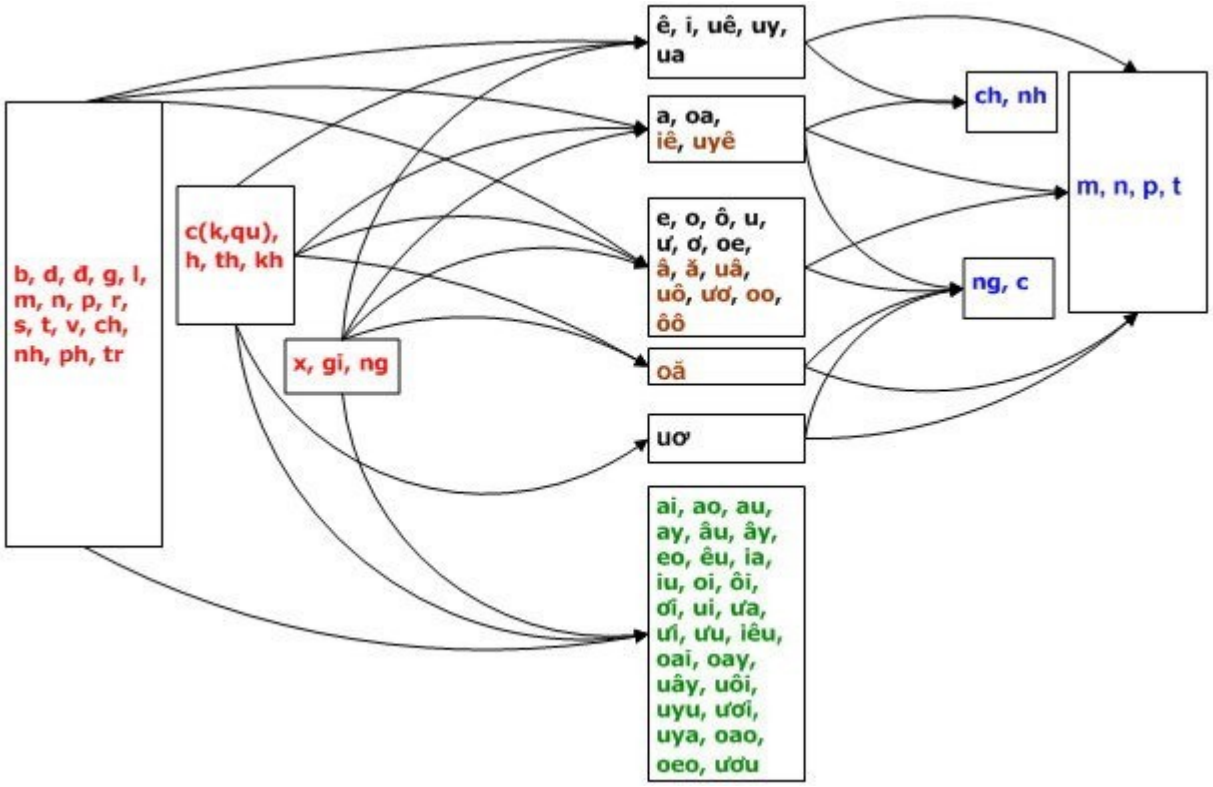
- nguyên âm oă chỉ thấy đi với: qu (loăng quăng = qu+oăng), h (con hoẵng), l (loăng quăng), s (soăn ?), x (xoăn tít), gi (gioăng = miếng đệm), kh (băn khoăn), ng (dài ngoẵng), th (thoăn thoắt)

Các trường hợp khác trông không "hạp nhãn" tí nào: Boă, doă, đoă, goă, moă, noă, poă, roă, toă, voă, choă, nhoă, phoă, troă.

Cũng có những phụ âm đầu và nguyên âm khác chưa bao giờ được ghép với nhau trong tiếng Việt, nhưng thử ghép thì thấy đọc được, không vấn đề gì.

Sơ đồ ghép vần

Những nhận xét trên có thể trình bày thành một sơ đồ ghép vần như sau:



Chú thích: đỏ: phụ âm đầu, xanh: phụ âm cuối, đen: nguyên âm thường (có thể có hay không có phụ âm cuối), nâu: nguyên âm lửng (phải thêm phụ âm cuối), lục: nguyên âm cuối (Không thể ghép thêm phụ âm cuối).

Ghép dấu thanh

Các chữ kết thúc bằng c, ch, p, t buộc phải có 1 trong 2 dấu thanh sắc hay nặng, không thể không có dấu thanh hay có các dấu thanh hỏi, ngã, huyền.

Các chữ khác có thể có bất kỳ dấu thanh nào hay không có dấu thanh. Số từ đơn tối đa của tiếng Việt

Đặt
D1: b, d, đ, g, l, m, n, p, r, s, t, v, ch, nh, ph, tr (16 âm) D2: c(k,qu), h, th, kh (4 âm)

D3: x, gi ,ng (3 âm)
N1: ê, i, uê, uy, ua (5 âm)
N2: a, oa, iê, uyê (4 âm, trong đó 2 âm lửng)
N3 : e, o, ô, u, ơ, oe, â, ă,uâ, uô, ươ (14 âm, trong đó 7 âm lửng) N4: oă (1 âm lửng)
N5: ươ (1 âm)
N6: ai, ao, au, ay, âu, ây, eo, êu, ia, iu, oi, ôi, ơi, ui, ưa, ưi, ưu, iêu, uôi, uyu, ươi, oai, oay, uây, uya, oeo, oao, ươu (28 âm cuối)
C1: nh, ch (2 âm, trong đó ch chỉ chấp nhận dấu sắc và dấu nặng) C2: ng, c (2 âm, trong đó c chỉ chấp nhận dấu sắc và dấu nặng)
C3: m,n,p,t (4 âm, trong đó p, t chỉ chấp nhận dấu sắc và dấu nặng)

Để ý là trong 3 nhóm phụ âm cuối, mỗi nhóm đều có ½ là các phụ âm chỉ chấp nhận 2 dấu thanh sắc, nặng, còn ½ kia là các phụ âm chấp nhận 6 thanh ngang, sắc huyền, hỏi, ngã, nặng, cho nên số chữ có phụ âm cuối khác nhau do dấu thanh bằng số chữ không dấu nhân với (1/2 x 2 + 1/2 x 6) = 4. Với các chữ không phụ âm cuối thì số chữ khác nhau do dấu thanh bằng 6 lần số chữ không dấu. Ta tính số chữ cho mỗi mối ghép giữa các nhóm bằng công thức:
Số pâ đầu trong nhóm x số nâ trong nhóm x số pâ cuối trong nhóm x hệ số thanh

Các chữ có đủ pâ đầu, nâ và pâ cuối: D1-N1-C3: 16x5x4x4=1280
D1-N1-C1: 16x5x2x4=640
D1-N2-C3: 16x4x2x4=512
D1-N2-C1: 16x4x4x4=1024
D1-N2-C2: 16x4x2x4=512
D1-N3-C1: 16x14x4x4=3584
D1-N3-C2: 16x14x2x4=1792
D2-N1-C3: 4x4x4x4=256
D2-N1-C1: 4x4x2x4=128
D2-N2-C3: 4x5x2x4=160
D2-N2-C1: 4x5x4x4=320
D2-N2-C2: 4x5x2x4=160
D2-N3-C1: 4x14x4x4=896
D2-N3-C2: 4x14x2x4=448
D2-N4-C2: 4x1x2x4=32
D2-N4-C1: 4x1x4x4=64
D2-N5-C2: 4x1x2x4=32
D2-N5-C1: 4x1x4x4=64
D3-N1-C3: 3x4x4x4=192
D3-N1-C1: 3x4x2x4=96
D3-N2-C3: 3x5x2x4=120
D3-N2-C1: 3x5x4x4=240
D3-N2-C2: 3x5x2x4=120
D3-N3-C1: 3x14x4x4=672
D3-N3-C2: 3x14x2x4=336
D3-N4-C2: 3x1x2x4=24
D3-N4-C1: 3x1x4x4=48

Các chữ có pâ đầu, nâ cuối: D1-N6: 16x28x6=2688
D2-N6:4x28x6=672
D3-N6:3x28x6=504
D1-N1: 16x5x6=480
D1-N2: 16x2x6=192
D1-N3: 16x7x6=672
D2-N1: 4x5x6=120
D2-N2: 4x2x6=48

D2-N3: 4x7x6=168
D2-N4: 4x1x6=24
D2-N5: 4x1x6=24
D3-N1: 3x5x6=90
D3-N2: 3x2x6=36
D3-N3: 3x7x6=126
D3-N4: 3x1x6=18

Các chữ có nâ đầu, pâ cuối: N1-C3: 5x4x4=80
N1-C1: 5x2x4=40
N2-C1: 4x2x4=32
N2-C3: 4x4x4=64
N2-C2: 4x2x4=32
N3-C3: 14x4x4=224
N3-C1: 14x2x4=112
N4-C2: 1x2x4=8
N4-C3: 1x4x4=16
N5-C2: 1x2x4=8
N5-C3: 1x4x4=16

Các chữ chỉ có nguyên âm

N1: 5x6=30
N2: 2x6=12
N3: 7x6=42
N5: 1x6=6
N6: 28x6=168

Tổng cộng có 20 504 từ đơn viết được, đọc được trong tiếng Việt. Tiếng Việt chỉ mới xài có chừng 1/4 số từ đơn viết và đọc được. Như vậy ghép đại một chữ theo các quy luật trên thì xác xuất nó có nghĩa chỉ chừng 25%. Tuy nhiên bất kỳ chữ nào cũng có thể được xài bất cứ lúc nào, không cần phải có nghĩa, chẳng hạn cho các mục đích sau: phiên âm tiếng nước ngoài: cái béc dầu, cái gioăng; mô tả âm thanh: lợn kêu oen oéc, hắn ta ợ dài: "uợ..."; đặt tên: ở Sài Gòn có ông luật sư đề bảng Luật sư Phan Xuân Giũng, ông này cương quyết phản đối những ai nói ông viết sai. Thiết nghĩ, tên không nhất thiết phải có nghĩa, vì vậy ông luật sư Giũng có lý, vì tên ông là vậy. Nếu bạn cố cãi, xin nhớ là bạn sẽ phải đối đầu với một luật sư đấy nhé!

Quy tắc bỏ dấu

Nguyên tắc bỏ dấu , nếu có 1 nguyên âm thì đó là vị trí bỏ dấu , nếu có 2 nguyên âm thì ta cứ liệt kê ra a-a , a-e , a-i , a-o , a-u …. để điền dấu .

Đối với các từ có 3 âm tiết thì ngoại trừ uyê ra thì chỗ còn lại đều là bỏ dấu ở vị trí thứ 2 . Các âm thế nào thì các bạn xem ở phần kiểm tra chính tả bên trên.

Cứ liệt kê ra mà tìm cách bỏ dấu thôi , nhưng phải chú ý 1 chút , ví dụ :

tòa - toàn , họa - hoạ , vừa – tuần (cùng có 2 nguyên âm là ua) ... có 1 số trường hợp đi sau nguyên âm có phụ âm như n sau chữ tòa , hoặc mũ , móc như hoạ thì vị trí dấu có thể thay đổi

Luu ý (trích nguồn wikipedia):

Trong đời sống, ví dụ như trong các chương trình máy tính giúp nhập tiếng Việt, hiện vẫn tồn tại hai cách đặt dấu thanh trong tiếng Việt. Ví dụ "hòa" là một cách đặt dấu thanh khác cho "hoà", trong đó "hòa" còn gọi là cách đặt dấu thanh "cũ". Bảng sau liệt kê các trường hợp mà hai cách đặt dấu thanh khác nhau:

Cũ	Mới
òa, óa, ỏa, õa, ọa	oà, oá, oả, oã, oạ
òe, óe, ỏe, õe, ọe	oè, oé, oẻ, oẽ, oẹ
ùy, úy, ủy, ũy, ụy	uỳ, uý, uỷ, uỹ, uỵ

Những người ủng hộ cách bỏ dấu kiểu "mới" cho rằng vì oa, oe & uy phiên âm theo IPA là wa, we & wi nên phải bỏ dấu vào vần a, e và i tương đương.

Trong khi đó những người ủng hộ cách bỏ dấu kiểu "cũ" thì cho rằng cách lí luận như trên là thiếu cơ sở vì IPA là để biểu thị cách phát âm chứ không phải biểu thị cách viết do đó không thể dùng để quyết định là cách bỏ dấu kiểu "mới" là đúng hơn. Thêm vào đó, IPA mới chỉ được phát triển vào cuối thế kỉ 19, trong khi chữ Quốc Ngữ đã được phát triển hoàn toàn độc lập và không ngừng thay đổi từ thế kỉ 17. Do đó, theo những người ủng hộ cách bỏ dấu kiểu "cũ" việc dùng IPA để quyết định xem tiếng Việt phải bỏ dấu thế nào là bất hợp lí. Những người này còn cho rằng mặc dù IPA là phương pháp biểu thị cách phát âm phổ dụng nhất nhưng không có nghĩa là cách biểu thị cách phát âm duy nhất cũng như không phải là cách biểu thị cách phát âm chính xác nhất vì vậy không có lí gì lại sử dụng nó làm chuẩn để quyết định cách bỏ dấu tiếng Việt mà không phải là một trong các phương pháp biểu thị cách phát âm khác.

Hơn nữa, theo như những người ủng hộ cách bỏ dấu kiểu "cũ" thì việc dùng IPA để quyết định cách tiếng Việt bỏ dấu là thiếu thống nhất vì lẽ IPA là phương pháp phiên âm quốc tế do đó nếu nó có thể áp dụng trong tiếng Việt để làm chuẩn mực cho vị trí bỏ dấu theo cách lí luận của những người ủng hộ cách bỏ dấu kiểu "mới" thì cũng phải áp dụng được cho toàn bộ các ngôn ngữ khác trên thế giới. Tuy nhiên, cách lí luận đó lại không thể giải thích thỏa đáng việc tại sao chữ C trong tiếng Catalan (phiên âm IPA là /k/ hoặc /s/) lại có thể được bỏ dấu để trở thành Ç (phiên âm IPA là /s/). Những người ủng hộ cách bỏ dấu kiểu "cũ" còn cho rằng lí luận như những người ủng hộ cách bỏ dấu kiểu "mới" là sử dụng một số ngôn ngữ nhất định để làm chuẩn mực cho toàn bộ các ngôn ngữ khác bởi vì đúng là ở một số ngôn ngữ thì không thể bỏ dấu trên chữ w, nhưng ở một số ngôn ngữ (ví dụ như tiếng Wales thì chữ w hoàn toàn có thể được bỏ dấu), do đó, bỏ dấu kiểu "cũ" vẫn dựa chính xác theo cách phát âm.

Trên quan điểm ngôn ngữ là do con người tạo nên và luôn biến đổi theo nhu cầu của con người, những người ủng hộ cách bỏ dấu kiểu "cũ" còn chỉ trích những người ủng hộ cách bỏ dấu kiểu "mới" là đang cố phức tạp hóa tiếng Việt, gây khó khăn không cần thiết nhất là trong giảng dạy học sinh tiểu học cũng như trong việc phát triển thuật toán và xử lí tiếng Việt trên máy vi tính. Họ còn cho rằng, thêm một quy tắc như trên không đem lại gì cho tiếng Việt nói chung và chữ Quốc Ngữ nói riêng do đó là hoàn toàn không cần thiết. Họ lấy dẫn chứng cho quan điểm của mình là việc chữ Quốc Ngữ từ khi được phát triển vào thế kỉ 17 đến nay đã trải qua rất nhiều thay đổi, bổ sung có và loại bỏ cũng có.

TỔNG QUAN VỀ MÃ VÀ FONT TIẾNG VIỆT

Các cách mã hóa tiếng Việt

Văn bản là chuỗi văn, mỗi đoạn văn là gồm chuỗi các từ (word), mỗi từ là chuỗi các ký tự (character). Một cách ngắn gọn: văn bản bất kỳ mà ta muốn xử lý là một chuỗi nhiều ký tự. Cách mã hóa văn bản tự nhiên nhất là mã hóa rời rạc từng ký tự trong văn bản đó. Do số lượng ký tự tiếng Anh (ngôn ngữ phổ biến nhất hiện nay) nhỏ, gồm 26 ký tự chữ từ “a” đến “z” và biến thể chữ hoa, 10 ký số từ 0 đến 9 và một số ký tự đặc biệt) nên người ta chỉ dùng 7 bit trong mỗi byte để mô tả một ký tự. Chuỗi 7 bit có thể mô tả được 128 giá trị khác nhau, mỗi giá trị được gán cho một ký tự. Mã ASCII mà máy tính đã, đang

và sẽ còn dùng được tạo ra theo ý tưởng trên. Tóm lại mã ASCII dùng 1 byte để mô tả một ký tự nhưng chỉ dùng 7 bit trong 1 byte, còn bit thứ 8 chưa dùng.

Đại đa số các tập ký tự của các quốc gia châu Âu thuộc họ La-tinh, gồm chủ yếu các ký tự tiếng Anh, chỉ thêm một ít ký tự có dấu. Để có thể mô tả được nhiều tập ký tự của các nước này, người ta đã nới rộng mã ASCII 7 bit thành ASCII 8 bit để có thể mô tả được 256 ký tự khác nhau: 128 ký tự tiếng Anh đã có cộng thêm 128 ký tự có dấu của một số nước châu Âu. Chuẩn mã hóa 8 bit này là ISO 8859. Do có những khác biệt nhỏ giữa các tập ký tự của các quốc gia châu Âu nên người ta đã tạo ra các biến thể khác nhau từ ISO 8859-1 đến ISO8859-15, trong đó chuẩn mã hóa ISO 8859-1 được sử dụng phổ biến nhất. Khi được cài đặt ở chế độ mặc nhiên, Windows và Linux đều sử dụng chuẩn mã hóa ISO 8859-1.

Tập ký tự tiếng Việt cũng thuộc họ La-tinh nên hầu hết các đơn vị tạo mã tiếng Việt trước đây đều dựa vào cách mà các nước châu Âu đã làm: nới rộng mã ASCII 7 bit thành mã tiếng Việt 8 bit, tuy nhiên việc tạo mã của các đơn vị chỉ có tính cục bộ, tự phát và chưa được tổ chức chuẩn hóa quốc gia và quốc tế thông qua (ngoài bộ mã TCVN 5172). Hiện có trên 40 bảng mã tiếng Việt được tạo ra theo cách trên gồm 2 nhóm chính: dạng mã dựng sẵn và dạng mã tổ hợp.

Dạng mã dựng sẵn (1 byte)

Cố gắng dùng chỉ 1 byte cho bất kỳ ký tự tiếng Việt nào. Phần nới rộng ASCII chỉ có 128 giá trị nên không thể dùng để mô tả đủ số lượng ký tự tiếng Việt có dấu là 134 (chữ thường và chữ hoa). Thường ta phải chọn một trong 3 cách dung hòa sau: hoặc chỉ mô tả chữ thường; hoặc cố gắng mô tả đầy đủ chữ thường và chữ hoa nhưng bỏ 6 ký tự ít dùng nhất; hoặc lấy thêm 6 ký tự ASCII ít được dùng để mô tả cho đủ tập 134 ký tự có dấu. Cách nào cũng có những hạn chế riêng của nó, hơn nữa có một số phần mềm sử dụng một số mã nới rộng này như là mã điều khiển để thực hiện các chức năng riêng của chúng nên nếu ta dùng nó làm ký tự tiếng Việt thì ký tự này sẽ không bao giờ được hiển thị trong các phần mềm này.

Ví dụ: TCVN 5712-VN1, VISCII, BachKhoa I, VietStar... là những mã dựng sẵn với một bảng font (cho cả chữ thường và chữ hoa). TCVN 5712-VN3 (ABC), VietSea, VNU, SC 3.0 là những mã dựng sẵn với hai bảng font (một cho chữ thường và một cho chữ hoa).

Dạng mã tổ hợp (2 byte hay nhiều hơn)

Mỗi ký tự có dấu tiếng Việt được mô tả bởi nhiều thành phần cơ bản ghép lại: mã ký tự cơ bản không dấu cộng thêm các mã ký tự mô tả các dấu. Để đơn giản hóa vấn đề, hầu hết các bảng mã tiếng Việt dạng tổ hợp chỉ dùng 2 byte: 1 byte mô tả mã ký tự cơ bản + 1 byte mô tả các dấu kèm theo (có thể từ 1 tới 2 dấu). Số lượng tổ hợp dấu cho các ký tự tiếng Việt rất nhỏ nên ta có thể chọn lựa thoải mái trong phần mã nới rộng (>128). Lưu ý rằng với cách mã hóa này, số lượng byte mô tả cho từng ký tự sẽ khác nhau: có ký tự chỉ chiếm 1 byte, có ký tự chiếm 2 byte,... Kết quả là việc xử lý văn bản sẽ phức tạp hơn dạng mã dựng sẵn. Hiện phương pháp xử lý tiếng Việt tổng quát là chuyển mã tiếng Việt cần xử lý về dạng mã trung gian (thường là 1 byte), xử lý trên mã trung gian rồi chuyển kết quả về mã ban đầu, như vậy phương pháp tổng quát này sẽ không hiệu quả, nhưng chúng ta phải chấp nhận nó trong bối cảnh có quá nhiều bảng mã tiếng Việt khác nhau.

Ví dụ: VietWare-X, VNI for Windows, TCVN 5712-VN2, BachKhoa II, VS2, 3C25... là những mã tổ hợp.
Lưu ý: Trên Windows 2000 hay Windows 95 tiếng Việt, Microsoft cung cấp bộ mã tiếng Việt với tên là "CodePage 1258", đây là loại mã tiếng Việt 1 byte dạng tổ hợp. Có thể nói Microsoft đã giải quyết vấn đề tiếng Việt rất tốt dựa trên bản mã này: nhập liệu thân thiện, hiển thị, in ấn tốt và quan trọng hơn là tất cả các hoạt động xử lý như sắp xếp, tìm kiếm dữ liệu tiếng Việt đều hoạt động rất tốt. Có thể nói rằng đây là bộ mã tiếng Việt được hỗ trợ hoàn hảo nhất từ trước tới nay, nhưng tiếc rằng nó chưa được chấp nhận.

Mã Unicode tiếng Việt

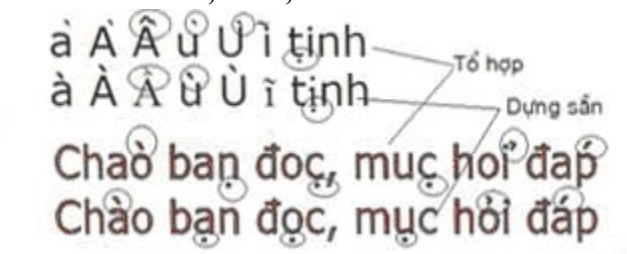
Xu hướng toàn cầu hóa đã và đang diễn ra mạnh mẽ, để một phần mềm được chấp nhận trên phạm vi toàn thế giới, nó phải xử lý được mọi tập ký tự của các quốc gia, nhất là các quốc gia châu Á với số lượng dân đông nhất thế giới. Mặc dù tập ký tự của mỗi quốc gia thường không lớn lắm (ngay cả tập ký tự của Trung Quốc cũng chưa tới 10.000 ký tự) nhưng hội các tập ký tự của các quốc gia (kể cả các tập ký tự của quá khứ mà bây giờ đã hết dùng) là khá lớn. Trong những năm đầu của thập kỷ 90 có 2 tổ chức khác nhau cùng cố gắng định nghĩa bộ mã hợp nhất thế giới này, đó là: Tổ chức chuẩn hóa quốc tế ISO (International Organization of Standardization) với dự án ISO 10646 và Hiệp hội các hãng sản xuất phần mềm đa ngữ với dự án Unicode.

May mắn cho chúng ta là vào năm 1991, các thành viên của cả 2 tổ chức này nhận thấy rằng việc tạo 2 bộ mã khác nhau cho thế giới là không cần thiết nên họ đã hợp tác với nhau để cùng đưa ra bộ mã thống nhất, mặc dù mỗi tổ chức vẫn xuất bản tài liệu và đặt tên riêng cho bộ mã thống nhất này: tổ chức ISO đặt tên bộ mã là ISO 10646 hay UCS (Universal Character Set), còn Hiệp hội các hãng sản xuất phần mềm đa ngữ đặt tên bộ mã là Unicode. Do Unicode là tên bộ mã thống nhất do hiệp hội các hãng sản xuất phần mềm nên ta nghe nói về nó nhiều hơn là ISO 10646.

Unicode đã phát triển qua nhiều version từ 1.0 đến nay là 3.1 và từ 2.0 trở đi thì các ký tự tiếng Việt đã được đưa vào bộ mã. Unicode hiện nay dùng 4 byte để mô tả một ký tự trong không gian mã 231 ký tự (2 tỉ ký tự), con số rất lớn đủ để mô tả mọi ký tự của mọi quốc gia, trong quá khứ lẫn hiện tại cũng như có dự trù cho việc phát triển trong tương lai.

Để dễ quản lý bộ mã, người ta chia nó ra thành nhiều phần (plane - mặt phẳng) khác nhau, mỗi mặt phẳng chứa 65.536 ký tự (dùng 16 bit để mô tả), được đánh số từ 0. Hiện nay người ta mới chỉ tìm ra và thống nhất được khoảng một triệu ký tự, trong số này chỉ có 65.534 ký tự đầu (mã từ 0000 đến fffd) được dùng phổ biến trên thế giới, tập con này được gọi là mặt phẳng đa ngữ cơ bản BMP (Basic Multilingual Plane) và được ký kiệu tắt là BMP0. Với tình hình thực tế trên, hầu hết các hệ điều hành và ứng dụng chỉ cố gắng xử lý được các ký tự Unicode nằm trong BMP0. Để mô tả được 65.534 ký tự khác nhau này, ta chỉ cần 2 byte cho mỗi ký tự là đủ. Unicode cũng cung cấp 2 phương pháp mô tả ký tự khác nhau:

• Mã dựng sẵn (precomposed characters): mỗi ký tự được mô tả bởi một mã Unicode.
• Mã tổ hợp (combining characters): mỗi ký tự được hợp thành từ nhiều thành phần khác nhau như ký tự chính và các dấu, mỗi thành phần có một mã Unicode riêng, như vậy một ký tự tổ hợp được mô tả bởi nhiều mã Unicode.
Ví dụ: Chữ ừ trong các dạng biểu diễn nói trên sẽ là chuỗi ký tự sau:
Chuỗi mã Hex
Dựng sẵn **1EAB**
Tổ hợp **01B0,0300**
Chính tắc **0075,031B,0300**



(*Các dạng biểu diễn chữ ừ*)

Mã dựng sẵn và mã tổ hợp là quá trình mã hoá các ký tự thành các ký tự dựng sẵn hay thành các ký tự tổ hợp. Có thể coi chính tắc là một dạng của mã hoá tổ hợp.

Ưu điểm của mã tổ hợp

• Mã tổ hợp có phần gọn nhẹ và chiếm ít mã hơn trong bảng mã, chỉ cần 20 vị trí cho ký tự thuần Việt (ă, â, ê, ô, ơ, ư, Ă, Â, Ê, Ô, Ơ, Ư, các dấu thanh: huyền, hỏi, ngã, sắc, nặng và dấu tổ hợp nguyên âm: nón, mũ, râu cho dạng chính tắc) trong khi mã dựng sẵn cần đến 134 cho ký tự thuần Việt.
• Mã tổ hợp có phần gần với ngôn ngữ tự nhiên (Việt) hơn trong quá trình ghép chữ, ghép vần.
• Mã tổ hợp sẽ dễ dàng hơn trong việc chuyển đổi chữ hoa/chữ thường, trong một số ứng dụng có thể dùng luôn tính năng Change Case có sẵn để chuyển đổi.
• Mã tổ hợp có vẻ như dễ dàng hơn trong việc sắp xếp tiếng Việt, nhưng thực ra không phải như vậy, lý do là các dấu thanh huyền, sắc ngã, hỏi, nặng - thứ tự trong bảng mã Unicode - lại nằm không đúng theo thứ tự sắp xếp tiếng Việt là huyền, hỏi, ngã, sắc, nặng, do đó vẫn phải thiết kế thuật toán riêng để sắp xếp mà không thể dùng các hàm có sẵn trong tiếng Anh. Khi đã phải dùng thuật toán riêng thì việc sắp xếp cho mã dựng sẵn cũng không khó hơn, không phức tạp nhiều hơn so với việc sắp xếp mã tổ hợp.
• Mã tổ hợp có phần dễ dàng hơn trong việc tìm kiếm tiếng Việt gần đúng, ví dụ những chữ tiếng Việt gần với âm "tha" chẳng hạn, thì các hàm tìm kiếm phổ thông sẽ tìm ra được các chữ thà, thá, thả, thã, thạ... Nhưng nếu tìm những từ gần với âm "than" thì lúc ấy lại phải thiết kế thuật toán riêng, mà khi đã phải dùng thuật toán riêng thì giữa tổ hợp và dựng sẵn thuật toán không khó hơn nhau nhiều.
• Trong thực tế, mã tổ hợp được hỗ trợ tốt hơn trong môi trường Windows 2000, và bộ Microsoft Office 2000, ý tốt hơn ở đây là chuyển đổi chữ hoa/thường, sắp xếp tiếng Việt được thiết kế ngay trong hệ điều hành và một số ứng dụng. Mã tổ hợp có thể hiện thì tốt hơn trong một số control có sẵn của Windows 2000, XP. Với Windows XP, Microsoft đã hỗ trợ luôn cả mã dựng sẵn với tính năng sắp xếp tiếng Việt.

Nhược điểm của mã tổ hợp

• Cài đặt mã tổ hợp khá phức tạp, số lượng môi trường cài đặt bị hạn chế hơn nhiều so với mã dựng sẵn, thông thường chỉ cài đặt được với font vector và bộ font cho phép định nghĩa các ký tự có độ rộng âm, khi đó 2 ký tự có độ rộng âm và dương tổ hợp lại sẽ cho ra ký tự cần hiển thị. Một khó khăn khá lớn nữa là phần lớn các công nghệ font phổ biến ngày nay như TrueType, OpenType, Typel... không cho phép thay đổi động vị trí nét trong hình chưa mà điều này lại rất cần thiết. Ví dụ chữ "à", và "À", thì vị trí của dấu huyền phải nằm ở 2 cao độ khác nhau tùy theo chữ cái cơ sở là chữ thường hay chữ hoa, việc thay đổi động cao độ của dấu than theo ngữ cảnh là chưa thực hiện được bằng kỹ thuật font chữ hiện hành. Để khắc phục vấn đề này, VNI đã phải đề xuất 2 mã riêng cho từng dấu thanh: 2 dấu huyền, một mã cho chữ hoa và một mã cho chữ thường. Trong CP 1258 và Unicode để đảm bảo tính đơn trị (tính một- một) các dấu thanh chỉ có một mã vì thế sẽ rất khó khăn trong hiển thị .
• Phương án thứ 2 mà Microsoft đưa ra để giải quyết vấn đề tăng giảm độ cao dấu thanh là dùng kỹ thuật Hook API thay đổi các hàm Display qua đó ánh xạ (map) chuỗi ký tự tổ hợp về chuỗi ký tự dựng sẵn để hiển thị và in ấn. Cơ chế này chỉ có trong Windows 95 tiếng Việt, Windows 2000, Windows XP mà không có trong Windows 95, Windows 98. Cơ chế này không phải bao giờ cũng thực hiện tốt, ngay cả trên Windows XP.
• Từ việc cài đặt mã tổ hợp phức tạp như vậy dẫn đến một nhược điểm thứ hai khá nghiêm trọng, đó là tính tương thích của mã tổ hợp kém hơn. Có nghĩa là một văn bảng bằng mã tổ hợp ở môi trường này có thể không đọc được trong môi trường khác. Nhất là khi dùng font bitmap để làm font hệ thống thì hầu như không thể cài đặt được mã tổ hợp, cũng như trong DOS, text console và nhiều môi trường Unix, Linux. Mã tổ hợp cài đặt trên các hệ điều hành phổ biến hiện nay là Windows 98 thì chữ rất xấu không thể chấp nhận được. Sẽ là một vấn đề lớn khi dùng mã tổ hợp phải nâng cấp phần cứng máy tính lên Windows 2000, XP (theo khuyến cáo của Microsoft để chạy mã tổ hợp tốt hơn), như vậy sẽ cần kinh phí rất lớn để nâng cấp, đào tạo lại…
• Độ mỹ thuật của mã tổ hợp thường kém hơn nhiều so với mã dựng sẵn, lý do là một ký tự dấu thanh có vị trí và cao độ xác định trong font chữ thường được dùng chung cho nhiều nguyên âm khác nhau và chúng được tổ hợp tự động sau khi nhập đoạn text. Vị trí của dấu thanh có thể hợp và đẹp với nguyên âm này nhưng lại có thể không phù hợp với nguyên âm khác. Ví dụ độ rộng của nguyên âm A thì khác với độ rộng của nguyên âm I (độ rộng rất hẹp) vì thế nếu đẹp cho chữ A thì xấu cho chữ I và ngược lại, để khắc phục tình trạng này, VNI phải định nghĩa riêng các mã cho các chữ ì, í, ỉ, ĩ, ị. Vì vậy chúng ta

thường coi VNI là giải pháp khắc phục tình thế hơn là một bộ mã, vì nó không đảm bảo tính đơn trị, và nhất quán (có nhiều mã cho một dấu - với chữ lại có xử lý khác so với xử lý các nguyên âm khác). Trong khi đó mã dựng sẵn được thiết kế từ trước (dựng sẵn) nên có thể bố trí vị trí dấu thanh nhờ vào vị trí thích hợp nhất cho từng nguyên âm, nên bao giờ cũng có khả năng đẹp hơn nhiều so với mã tổ hợp.
• Xử lý hiệu ứng với đoạn mã tổ hợp có nhiều vấn đề khó khăn hơn so với mã dựng sẵn. Trong nhiều trường hợp, một con chữ tiếng Việt trong lưu trữ và hiển thị với mã dựng sẵn lại không phải là một thể thống nhất (tổ hợp từ những ký tự rời rạc) cho nên khi thực hiện các hiệu ứng với đoạn văn bản như co dãn text, xoay, dồn chữ, canh đều hai bên… thì các dấu và chữ thường bị tách rời nhau, chữ đi một nơi và dấu đi một nơi, ảnh hưởng đến mỹ thuật và độ chính xác. Có thể thấy trên các tít báo dùng font VNI hay xuất hiện các hiện tượng xa rời dấu thanh.
• Xử lý với các ký tự mã tổ hợp phức tạp hơn so với mã dựng sẵn, do mỗi chữ cái trong mã tổ hợp có độ rộng thay đổi, lúc có thể là một ký tự, lúc khác lại được tổ hợp từ nhiều byte khác nhau. Khi tách từ, tách ký tự (theo ngôn ngữ tự nhiên), thường dùng để phân tích cú pháp hay đánh chỉ số phải xây dựng thuật toán riêng khá phức tạp, trong khi mã dựng sẵn có độ rộng cố định nên việc rút ký tự từ đoạn text ra rất đơn giản, không cần xây dựng thuật toán riêng. Ngoài ra việc xử lý ký tự khác như: xóa ký tự, di chuyển cho trỏ di theo đơn vị ký tự thì thực hiện với mã tổ hợp khó khăn và phức tạp hơn: thường phải xóa 2 sẵn cho một chữ, di chuyển 2 lần con trỏ mới đi ra khỏi một chữ, điều này là xa lạ với ngôn ngữ tự nhiên.
• Kích thước các tệp dữ liệu lưu ở dạng tổ hợp thường lớn hơn so với mã dựng sẵn khoảng 25-30% do đó nó chiếm nhiều không gian trong đĩa cứng, bộ nhớ hơn, và trên đường truyền mạng (internet/intranet) ngốn nhiều băng thông hơn.
• Trong cơ sở dữ liệu, thiết kế cấu trúc cơ sở dữ liệu với mã tổ hợp thường phức tạp hơn. Vì mặc dù biết trước số chữ cái max nhưng lại khó đoán nhận chính xác độ dài chuỗi byte tương ứng lớn nhất, nếu thiết kế không khéo sẽ bị tràn bộ nhớ. Và khó khăn trong việc phân tách ký tự, phân tách từ cũng làm khó khăn thêm trong việc xử lý text trong lĩnh vực cơ sở dữ liệu.
• Trong việc đánh chỉ số (index), và tìm kiếm toàn văn (full text search), mà tổ hợp cũng gây nhiều khó khăn hơn (phân tách từ, phân tách ký tự) và các ký tự dấu thanh trong mã tổ hợp thường bị coi là dấu phân cách từ, dẫn đến việc đánh chỉ số bị sai và tìm kiếm toàn văn cũng không đúng. Hiện tượng này thường gặp với đa số các bộ search engine, công cụ tìm kiếm toàn văn trong Oracle và Lotus Notes 5.0 đều bị lỗi đánh chỉ số sai. Tuy rằng Oracle và Lotus Notes đều hỗ trợ Unicode trong phần encoding, nhưng đáng tiếc phần tìm kiếm toàn văn mua lại của hãng thứ 3 INSO và Verity đều thực hiện các phân cách từ sai với mã tổ hợp. Nhưng lỗi này không xảy ra với mã dựng sẵn.
• Tính thực tế của mã tổ hợp kém hơn mã dựng sẵn: đa số ở Việt Nam cùng như ở nước ngoài, Unicode dựng sẵn được dùng rất phổ biến, các website của Việt Nam như Vnexpress và VASC Orient hàng ngày có gần 2 triệu lượt truy nhập chứng tỏ số lượng người dùng mã dựng sẵn rất lớn, trong khi các website dùng mã tổ hợp rất ít.

Dưới đây là một minh hoạ cho sự lệch lạc dấu thanh của mã tổ hợp, dòng trên là soạn bằng mã tổ hợp, dòng dưới được soạn bằng mã dựng sẵn, được dùng cùng một font chữ Verdana, trong hệ điều hành Windows XP, và đoạn text được soạn trong PowerPoint 2000, 2 dòng cuối được soạn bằng WordArt cũng trong Office 2000 và Windows XP. Tất cả các đoạn text trên chưa hề qua một hiệu ứng text nào, mà chữ và dấu thanh đã bị lệch và xa rời nhau trong khi mã dựng sẵn luôn hiển thị đúng và đẹp. Hiện tượng này còn bị phổ biến hơn với tất cả các ứng dụng (kể cả Word XP, Excel XP...) chạy trên hệ điều hành Windows 95, 98.

(*Sự lệch lạc dấu thanh của mã tổ hợp*)

Ưu và nhược điểm của mã dựng sẵn

Những ưu và nhược điểm của mã dựng sẵn cũng đã được phân tích khá kỹ và song song trong quá trình phân tích các ưu và nhược điểm của mã tổ hợp, chỉ xin được tổng kết lại một cách tóm tắt: mã dựng sẵn không bị những nhược điểm của mã tổ hợp, các file sử dụng mã dựng sẵn tốn ít không gian nhớ, cài đặt

đơn giản hơn, chữ hiển thị đẹp hơn, mỹ thuật hơn, xử lý với xâu ký tự dựng sẵn dễ dàng hơn (phân tách từ, phân tách ký tự, xóa và di chuyển con trỏ), chữ và dấu là một khối thống nhất nên khi co dãn text không bị hiện tượng xa rời dấu thanh, và đặc biệt tính tương thích của mã dựng sẵn cao hơn có thể chạy được trên nhiều môi trường khác nhau (từ font vector đến bitmap, từ Windows, Macintosh đến Linux), vì chuyển mang văn bản dữ liệu giữa các môi trường không đồng nhất (Multiplatform) là một điều cũng rất cần thiết.

Tất cả những ưu điểm của mã tổ hợp, thì lại không phải là căn bản và đều có thể giải quyết được tương đối khá dễ dàng, trong khi đó mã tổ hợp có những vấn đề về kỹ thuật khá phức tạp (cài đặt, hiển thị...). Vẫn biết kỹ thuật phải theo chuẩn, nhưng tại thời điểm hiện nay có những giới hạn nhất định về kỹ thuật và công nghệ ví dụ cách đây 30 năm, mã hóa 8 bit là giới hạn chưa giải quyết được do không gian nhớ eo hẹp.. . ngày nay cài đặt mã tổ hợp trong nhiều môi trường chưa thể làm được hoặc phải làm rất khó khăn và đi vòng vo như ánh xạ từ tổ hợp về dựng sẵn.

Các nhược điểm của mã tổ hợp có thể khắc phục được một phần ở thời điểm hiện nay và có thể khắc phục hoàn toàn trong tương lai, nhưng chúng ta nên chọn những phương án đơn giản, dễ cài đặt, đẹp hơn... và nhiều ưu điểm ở trên là mã dựng sẵn bởi vì cái lợi của mã tổ hợp mang lại thì không đáng kể mà để thực hiện hoàn hảo nó thì lại có quá nhiều khó khăn.

Cái có thể nói nhược điểm của mã dựng sẵn hiện nay là chưa được Microsoft chú ý hỗ trợ nhiều. Trong thời gian trước Microsoft có thiên hướng đi theo mà tổ hợp (là phần nâng cấp từ mã 8-bit CP 1258 lên).Những xử lý về tiếng Việt như sắp xếp, chuyển đổi chữ hoa/thường... được tích hợp luôn vào hệ điều hành như thế tốt hơn so với mã dựng sẵn, nhưng điều này chỉ đúng với Windows 2000 (không đúng với Windows 95, 98), ngoài ra trên Windows XP, Microsoft đã bắt đầu hỗ trợ thêm mã dựng sẵn vào trong hệ điều hành: phần sắp xếp tiếng Việt và việc trong tương lai Microsoft sẽ hỗ trợ hoàn toàn mã dựng
sẵn vào hệ điều hành không phải là vấn đề phức tạp về kỹ thuật.

Ngoài ra, trước kia với các mã VNI, ABC-TCVN 5712, chúng ta đâu có được Microsoft hỗ trợ tiếng Việt ở mức hệ điều hành, nhưng vẫn không bị ảnh hưởng máy, CNTT vẫn tiến triển tốt. Thêm nữa, hiện đã có những hỗ trợ tiếng Việt ở mức thấp API (giống như Windows) do các đơn vị ở trong nước thực hiện và đáp ứng đầy đủ các yêu cầu về xử lý tiếng Việt ở tầng thấp.

Tóm lại mã dựng sẵn với nhiều ưu điểm nổi trội có thể đáp ứng được nhu cầu trước mắt cũng như lâu dài và có thể sử dụng trong nhiều lĩnh vực: chế bản văn phòng, web, email, cơ sở dữ liệu và nhiều lĩnh vực khác, chúng ta nên chọn mã dựng sẵn, và tổ hợp chỉ có thể dùng trong một số bài toán đặc thù như phân tích chuỗi ký tự hay để biểu diễn các hình chữ cho dấu thanh.

Unicode và TCVN 6909

Năm 2001, Khoa học Công nghệ và Môi trường ban hành bộ mã chữ quốc ngữ mới với mã số TCVN 6909 nhằm công nhận và hợp pháp hóa bộ phận mã tiếng Việt được định nghĩa trong bộ mã Unicode.

Trước tiên cần phải nêu rõ mối quan hệ giữa Unicode và TCVN 6909. TCVN 6909 là một tập con của Unicode, nhưng đã được chọn lọc kỹ lưỡng để lấy ra đúng những ký tự sẽ dùng trong ngôn ngữ Việt. TCVN 6909 quy định rõ ràng hơn và tính đơn trị một-một cao hơn trong Unicode. Ví dụ trong Unicode có đến 3 chữ 'Đ', 8 dấu nặng '.' nếu không quy định rõ sẽ gây nhầm lẫn, mà thực tế đã xảy ra nhầm lẫn chữ 'Đ' trong bộ gõ Unicode: VPS của Việt kiều Pháp. TCVN 6909 chứa tất cả các ký tự dựng sẵn và cùng có đầy đủ các mã để hiển thị cho dấu thanh, mã cho việc biểu diễn các dấu tổ hợp nguyên âm, như vậy TCVN 6909 là bộ mã rất đầy đủ (trong đó có mã cho tất cả các thành phần tiếng Việt). TCVN 6909 không quy định biểu diễn dựng sẵn hay tổ hợp. Vì vậy nói TCVN 6909 là mã dựng sẵn là không chính xác.

TCVN 6909 kế thừa các ưu điểm của bộ mã Unicode và ISO 10646, cho nên nó hoàn toàn cho phép hội nhập tiếng Việt với các ngôn ngữ khác trong cộng đồng Unicode. Hiện nay đa số các ứng dụng phổ thông đều đa hỗ trợ Unicode: phần mềm văn phòng, cơ sở dữ liệu, web, e-mail... Tuy nhiên, cũng còn một số các phần mềm chuyên ngành chưa hỗ trợ Unicode.

Vấn đề chuyển mã sang Unicode

Vấn đề kỹ thuật của Unicode không chỉ là bộ gõ Unicode (như một số người đã lầm tưởng) mà là một loạt các vấn đề khác, đặc biệt là các công cụ chuyển mã cho văn bản dữ liệu (Microsoft Office,Webpages, Database...). Chuyển mã là vấn đề lớn hơn nhiều, vì khối lượng dữ liệu cần chuyển đổi sang Unicode rất lớn, đa dạng về khuôn thức loại hình, chủng loại. Ngoài ra còn các vấn đề sắp xếp, chuyển đổi chữ hoa, chữ thường, tìm kiếm toàn văn, kiểm tra chính tả tiếng Việt, đặc biệt hỗ trợ ở mức lập trình cho các công cụ phát triển cũng là điều rất cần thiết (hiển thị, lưu trữ…).

Có thể nói chuyển đổi sang Unicode là một việc làm rất cần thiết và phải làm càng sớm càng tốt, vì càng để chậm, kho dữ liệu, các website càng ngày càng phát triển với dữ liệu càng lớn thì quá thì quá trình chuyển mã về sau càng phức tạp, càng tốn kém nhiều công sức và tiền của. Vấn đề Unicode, không chỉ là bộ gõ, mà phải nghĩ đó là một loạt các công cụ đằng sau chuyển mã và hỗ trợ tiếng Việt ở mức hệ thống, ứng dụng trong nhiều môi trường khác nhau.

(sưu tầm)