Machine Learning based on logic synthesis methods for a humanoid robot to imitate human behaviors

Final Report

for

Advanced Logic Syntheses ECE 572
Portland State University, Fall 2003
Stefan Gebauer

1. Abstract

There were some approaches by several students who experimented with humanoid robots [1]. Basically, they used common algorithms, which are known for many years. They checked only if some conditions are true and created reactions that are every time the same in a particular situation. The behavior of the robot couldn't be improved. Since a couple of years the Portland State University, especially Prof. Perkowski spends a lot of effort to develop smart algorithms for Multi Value Decomposition. We want to show in our project that the algorithms can also be adapted to machine learning. The final goal is to demonstrate relational decomposition as a new general purpose machine learning method. Finally, we hope to contribute a new way of thinking to the classical learning methods such as Neuronal Networks and Reinforcement Learning.

2. The information flow chart in our project

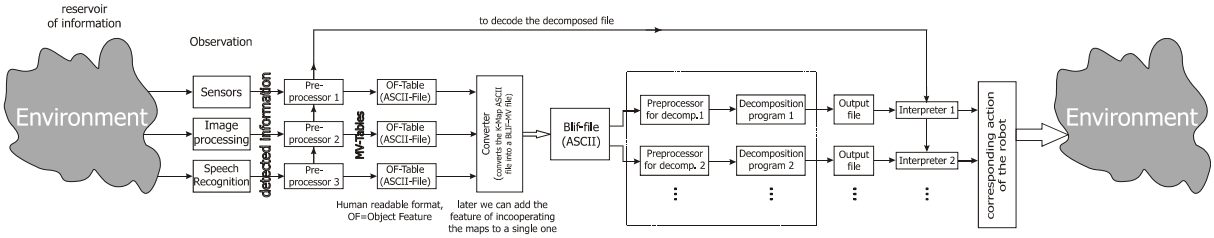

Fig. 1: the information flow of the project

We have some sensors and a camera to observe the environment/room where the robot is located. A special kind of gathering will be done by speech recognition. We're going to have short conversations between a person and the robot. While the person is speaking we try to recognize single words ("Word Spotting, which works like a bandpass filter. It only gathers information that are valid.) that are loaded from a key-word list. The robot is only sensitive to this information. The detected information is propagated to its corresponding preprocessor (compare *Fig. 1*) that stores the data (objects and features) in an external table. It may be easily readable. Therefore we decided to create an ASCII file. Please refer to chapter 5.2 for more information. We have then 3 different tables, for each observer (Camera [image processing], Sensors, Speech Recognizer) one. The file is being converted into a BLIF-MV[1] file by an external converter (Aminul). The BLIF-MV file is either readable by the decomposition program like MVSIS[2] or we have to write a new program which converts the BLIF-MV-file into the format of the decomposition program. For the case where we use MVSIS the preprocessing program and the decomposition program are one, like I give to understand it with pointed rectangle in Fig. 1. After the decomposition we get an output file which has to be interpreted by an interpreter. For the case we use MVSIS, the output file is a BLIF-MV file again. We loose information while the detected information is preprocessed for the first time. We can solve this either by storing the necessary data in an external file and use it, or we store the information internally in the program. If we decide to store the data externally we have to design a new file format which contains something like a decode table. While the interpreter is running to interpret the data it has to decode the information in order to know what was which variable and which feature was associated to which object.

[1] BLIF-MV Berkeley Logic Interchange Format
[2] MVSIS Multi Value Decomposition Program

The interpreted information is transferred to the thread or program that provides control over the mechanical/TTS part of the robot.

3. A machine learning example

In order to utilize all the concepts of machine learning for a humanoid robot, let's use an example for demonstrating of machine learning methods. In this example, we are to obtain an age of a person without the help of image processing.

Following properties can be asked by robot to gather information to achieve our main goal.

- Pitch of the voice
- The height of a person
- Color of the hair

We will also use four different ages of people for this example. They are in the following lists:

- Joan is a kid
- Mike is a teenager
- Peter is a middle age
- Frank is an old person

Let's use following scale for the properties or features of each persons.
a = pitch
b = height of a person
c = color of the hair

Pitch	Very High	High	Middle	Low
Values	3	2	1	0

Height	Very Tall	Tall	Middle	Short
Values	3	2	1	0

Color	Grey	Black	Brown	Blonde
Values	3	2	1	0

We can create a table that will describe all the features from above.

Name (examples)	**Age (output)**	Pitch	Height	Hair Color
Joan	Kid (0)	a(3)	b(0)	c(0)
Mike	Teenager (1)	a(2)	b(1)	c(1)
Peter	Mid-age (2)	a(1)	b(2)	c(2)
Frank	Old (3)	a(0)	b(3)	c(3)

Table 1: learning examples

Based on the above table, I can fill the cells in the K-Map with the multi-valued output.

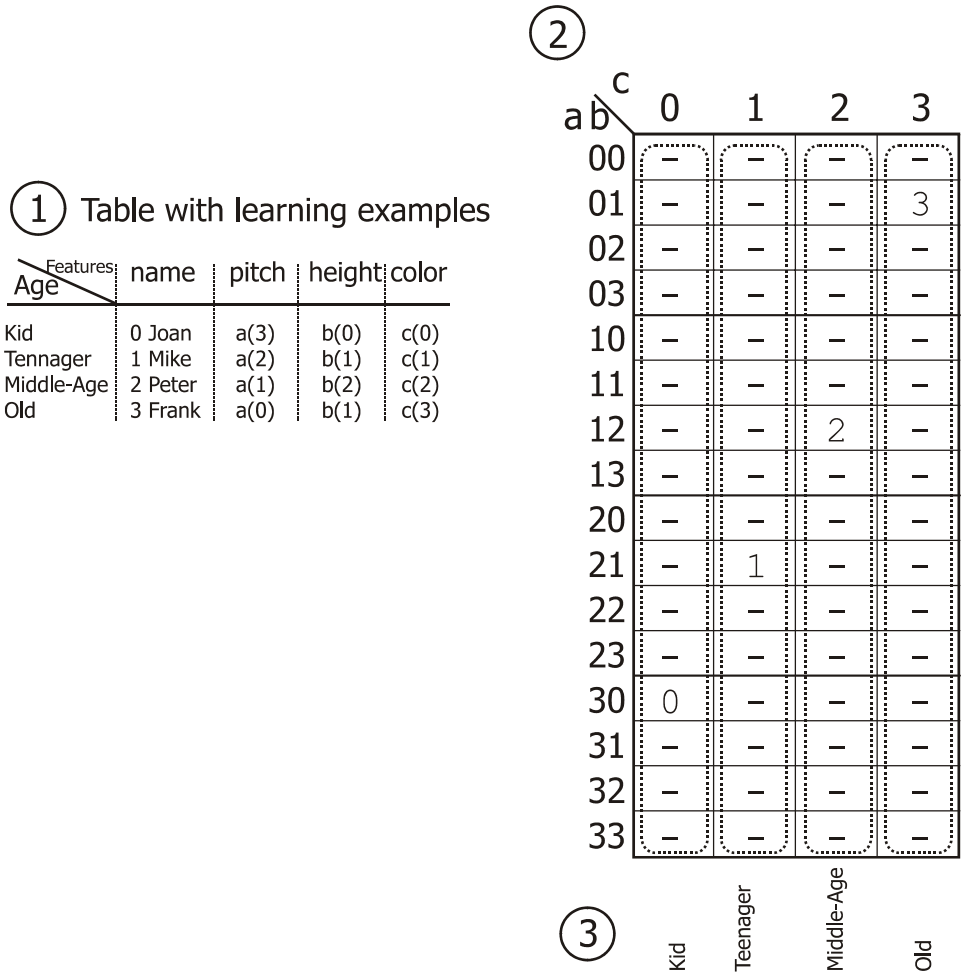

Fig. 2: K-Map with the examples and the decomposition

By observing the decomposition process in *Fig. 2*, one can not really distinguish a person's age by knowing the color of the hair. I created these examples intentional in order to show the learning process even if the system is incomplete, it will be able to improve it's prediction if it gets more examples. From the point of view of a human, it is obvious that the decomposition isn't good because the color of hair isn't the feature for getting the age. In this case, our system will provide wrong outputs at this time. Only by chance it will get the right age. What can we do? How could the system be improved? In order to perform better decomposition the systems needs more examples. The robot could ask if the age he found is right or wrong. If the answer is wrong, the system takes the answers of the person and treats them as a new example. As we will provide more knowledge (examples), our system will update its information. Based on the updated information, robot will provide a better output.

Now, let's make little change to our previous information. From our above table, we know that teenager has high pitch, tall height, and black hair color. Now if we update this information with some thing more, we will obtain really interesting output. Let's assume that person's name is Jake, who is a teenager with high pitch, tall height, and blonde hair color.

Age	Name	Pitch	Height	Hair Color
Teenager (1)	Jake	2	1	0

Let's draw a new karnaught map based on our updated information.

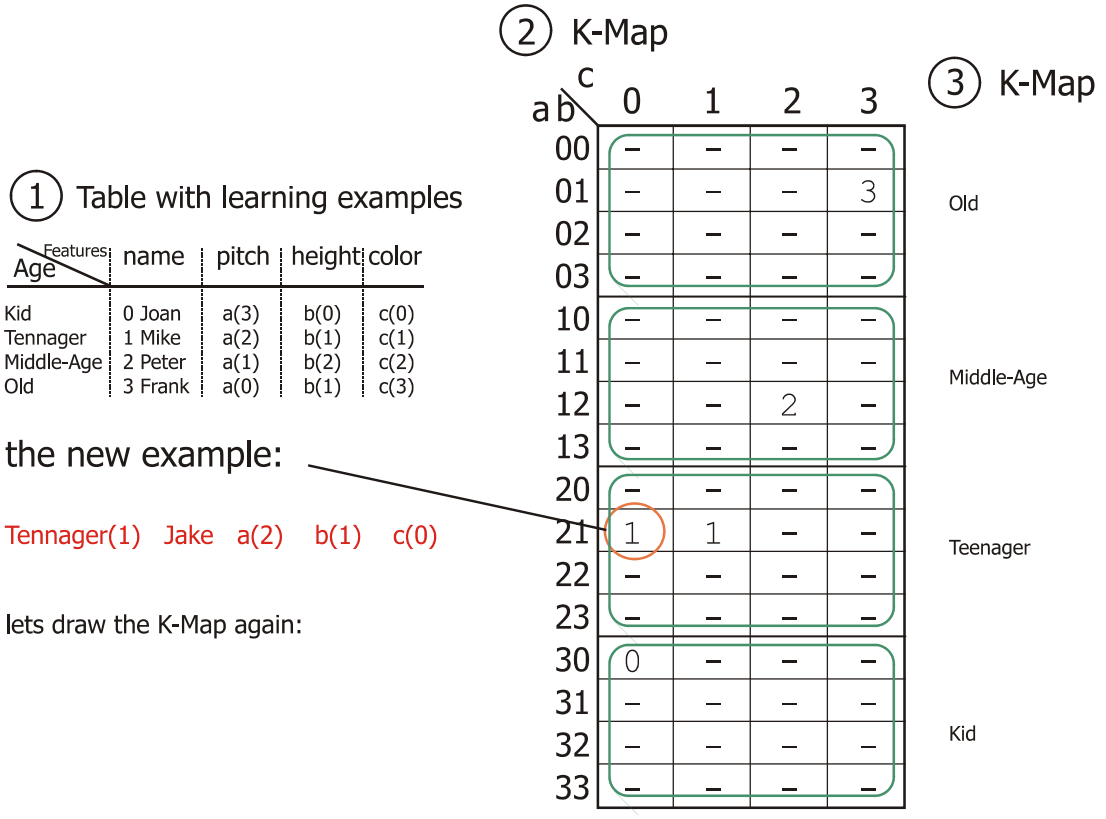

Fig. 3: modified K-Map

From the above K-Map, now our system will distinguish a person's age by knowing the pitch. It seems that is a better decomposition and it's amazing that we got this good result after 5 examples.

4. Decomposition with MVSIS [3]

4.1 A short introduction to BLIF-MV [2]

I'll give only a brief overview about the BLIF-MV file format. It is of interest to understand what MVSIS does. In the following there are only those commands discussed, those are valid for our project.

BLIF (Berkeley Logic Interchange Format) is used to describe circuits in text format. It is used by several logic minimization programs as input and output file format. Later on, it was enhanced to MV variables. BLIF-MV is a language designed for describing hierarchical sequential systems with non-determinism [1]. It is possible to describe systems hierarchical due to the ability to describe systems sequential. You can implement non-deterministic behaviors. This is done by allowing non-deterministic gates in descriptions. Non-deterministic gates generate an output arbitrarily form the set of pre-specified outputs. Another extension of the BLIF-MV format is the support of multi-valued variables.

The # sign is used for a command

```
.model <model-name>
.inputs <input-list>
.outputs <output-list>
.mv <variable list> <number of values> (in enumerative case)
<command>
...
<command>
.end
```

- model-name specifies the name of the model, it can be every name.

- .input command is used to specify the inputs where the input-list specifies the input variables, they are separated by a white-space, same syntax is used for .output
- .mv defines variables as muli-valued variables, they are separated by comma, the last number presents the number of values
- maybe important for our project is the capability to use symbolic variables like small, tall and middle. Then the notation is a little bit different: `.mv <variable list> <number of values> <value-list>`
- Example `.mv a, b, c 3 small middle tall`
- <command> is replaced by table, .latch, .reset or .subckt. It is the definition of the model.
- `.default output` means all output combinations which doesn't occur in the model description have as default the number you specified for output.
- with `.table var1 var2 -> out1 out2` begins the description of the table. Every combination that isn't mentioned is assumed as don't care output.
- Example: `.table a b -> y1 y2`
  `0 1 1 1`
  means for a=0 and b=1 y1 and y2 are, for the other inputs you have as output DC's you can easily describe the table in a better way if you add more lines like that
- there are shortcuts to make it easier to specify the table:

`.mv a b 5`	(six value variables)
`.table a ->b`	(a input, b output)
`1 -`	(if a=1, output is DC)
`{2-4} 1`	(for a=2, and a=3, as well as a=4, output is 1)
`(0,5) 4`	(for a=0 and a=5 output is 4)

  you can do the same with the output
- `.end` is the last line of the BLIF-MV file

4.2 The input file

I created a BLIF-MV file form the example in Chapter 3 on page 1.

```
# this is the machine learning example. We gonna get the age of a person
# we teach the system first by several examples
# the following file instantiate some examples
#
.model age
#.inputs <input-list>
# a=pitch, b=height, c=color of hairs
.inputs a b c
#.outputs <output-list>
.outputs age
#.mv <variable list> <number of values>
# four valued variables
.mv a, b, c, age 4
# .table defines the table which might be decomposed
# we obtain the values from the K-Map
.table a b c -> age
# dont cares are represented by an -
0 0 0 -
0 0 1 -
0 0 2 -
0 0 3 -
0 1 0 -
0 1 1 -
0 1 2 -
0 1 3 3
0 2 0 -
0 2 1 -
0 2 2 -
0 2 3 -
0 3 0 -
0 3 1 -
0 3 2 -
0 3 3 -
#
1 0 0 -
1 0 1 -
1 0 2 -
1 0 3 -
1 1 0 -
1 1 1 -
1 1 2 -
1 1 3 -
1 2 0 -
1 2 1 -
1 2 2 2
1 2 3 -
1 3 0 -
1 3 1 -
1 3 2 -
```

```
1 3 3 -
#
2 0 0 -
2 0 1 -
2 0 2 -
2 0 3 -
2 1 0 -
2 1 1 1
2 1 2 -
2 1 3 -
2 2 0 -
2 2 1 -
2 2 2 -
2 2 3 -
2 3 0 -
2 3 1 -
2 3 2 -
2 3 3 -
#
3 0 0 0
3 0 1 -
3 0 2 -
3 0 3 -
3 1 0 -
3 1 1 -
3 1 2 -
3 1 3 -
3 2 0 -
3 2 1 -
3 2 2 -
3 2 3 -
3 3 0 -
3 3 1 -
3 3 2 -
3 3 3 -
.end
```

4.3 Manipulation

First of all you have to load the file in MVSIS with `read_blif_mv` *`filename.mv`*. The command `help` prints the available commands. With `decomp` you are going to decompose the table. It creates intermediate nodes which we don't need. We removed them with the command `eliminate`. Still, the table is not minimal. We did this one time with the `fullsimp` operation and a second time with `mfs` operation.

4.4 Output file

The file can be saved by typing `write_blif_mv` *`filename.mv`*.

`mfs simplification`	`fullsimp simplification`

`.model age`
`.spec example.mv`
`.inputs a b c`
`.outputs age`
`.mv a 4`
`.mv b 4`
`.mv c 4`
`.mv age 4`
`.table b c age`
`.default 0`
`- 1  1`
`2 2  2`
`- 3  3`
`.end`
	`.model age`
	`.spec example.mv`
	`.inputs a b c`
	`.outputs age`
	`.mv a 4`
	`.mv b 4`
	`.mv c 4`
	`.mv age 4`
	`.table a b c age`
	`.default 1`
	`(0,2,3) (0,2,3) -  0`
	`- (0,2,3) (0,1,3)  0`
	`(1,2,3) (0,1,3) (0,2,3)  0`
	`(0,1,3) (0,1,3) (0,1,2)  0`
	`- (1,2,3) (0,2)  2`
	`(0,1,2) (0,2,3) -  2`
	`(1,2,3) - (2,3)  2`
	`(0,1,3) - (1,2)  2`
	`(0,1,2) - (0,3)  3`
	`- (1,2,3) (0,3)  3`
	`(0,1,3) (0,1,3) (1,2,3)  3`
	`(0,2,3) - (2,3)  3`
	`- (0,2,3) (1,3)  3`
	`.end`

The verify command says the decomposition is good, in booth cases. Unfortunately I don't have time to show it in the report. Hopefully Myron or Aminul will redraw the K-Maps.

5. What is done so far

5.1 Description of the TTS program

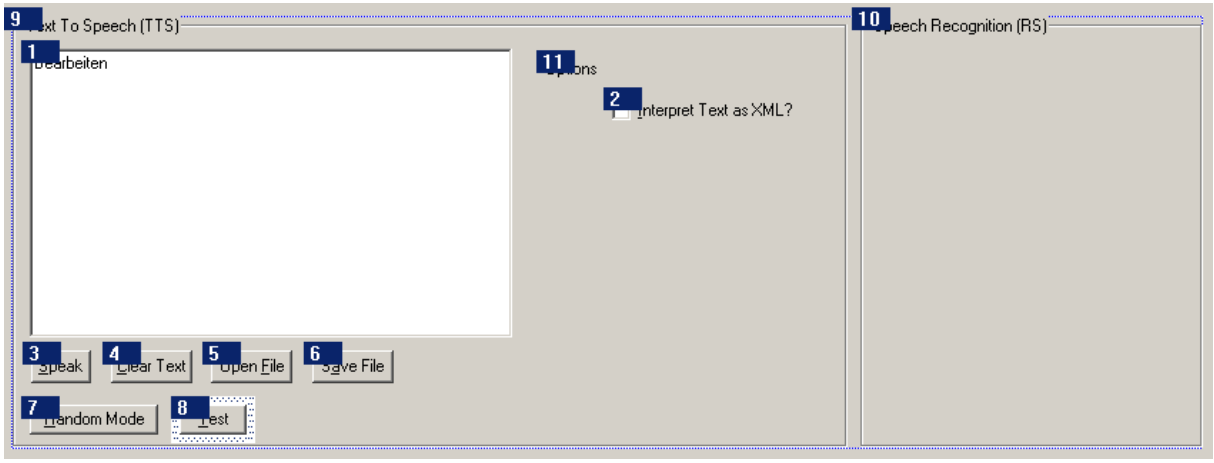

Fig. 4: The TTS program window

The First Item is a text box. You can write down text there. If you click on 3 “Speak” the text will be spoken. Button 4 “Clear Text” clears the text box. Button 6 “Save File” saves the text in the text box. You should use the extension *.sst (speech synthesis text) because this is the default extension in the file open dialog which appears by clicking 5 “Open File”. You can open any plain text file or a SAPI XML file. I provide with my program an xml example file. If you open a text file, the check box 2 will be checked automatically. If you write xml code and you want to interpret it as xml you have to make sure the box is checked before you click the “Speak” button. The Button 7 “Random Mode” is used to generate random sentence. Be sure you have a file called “random.txt” in the applications directory. The file can be edited; content can be added and removed arbitrarily. Last but not least we have the “Test” button. If you don’t hear a voice speaking, probably the MS SAPI isn’t installed on your computer.

If you want to compile my source code, please refer to the readme files and documentation that comes along in the archive. The main part of my source code is listed in 8. Appendix. Note, that every button has it’s own function. This makes it easy to analyze the code.

5.2 Description of the OF Table (Object Feature Table), important for Aminul

the file has the following structure:

```
.variables
.output
.table
object-name variable1=value variable2=value | output1=value
output2=value;
.EOT
```

There are two versions of declaring the variables. If you use `.variables` command the variables are binary variables. If you use `.mvariables` the variables are multi valued. The last argument assigns the number of values to the variables.

Examples:

```
.variables a, b, c, d        (binary variables)
----
.variables a                 (same result as before)
.variables b
.variables c
.variables d
---
.mvariables a, b, c 5        (muli valued variables, possible values are 0-5)
```

```
---
.mvariables a 5        (same result as before)
.mvariables b 5
.mvariables c 5
```

The output variables are defined in the same way. We have again two versions: `.output` and `.moutput`. The notation is the same.

`.table` declares the very beginning of the table. It doesn't have arguments

A row in the table has the following structure:

```
object-name variable1=value variable2=value | output1=value output2=value;
```

I derive an example from chapter 3:

```
.mvariables a, b, c 3
.moutputs d 3
# this is a comment
Joan a=3 b=0 c=0 | d=0;
Mike a=2 b=1 c=1| d=1;
Peter a=1 b=2 c=2 | d=2;
Frank a=0 b=1 c=3 | d=3;
.EOT
```

`.EOT` declares the end of the table. A comment can be initiated by the hash sign (#).

6. Conclusion

We have now the TTS tool. The architecture of our project is developed so far. The next steps are the recognition part, the BLIF converter as well as the BLIF interpreter. The image group deals with recognizing features in a captured picture and how to capture the pictures, at all. We did the first step, but still, there are a lot of things to do.

7. Literature

[1] Uland Wong and Marek Perkowski, "A new Approach to Robot's Imitations of Behaviors by Decomposition of Multiple-Valued Relations", paper, Portland 2002

[2] Yuji Kukimoto, BLIV-MV, http://www-cad.eecs.berkeley.edu/Respep/Research/vis/doc/blifmv/blifmv/blifmv.html, May 1996

[3] Donald Chai, Jie-Hong Jiang, Yunjian Jiang, Yinghaua Li, Alan Mishchenko, Robert Brayton, "MVSIS 2.0 User's Manual, Berkeley 2002

8. Appendix

Note: German comments are created by the Visual C++ 6.0. You don't have to deal with them because they set up some basic things for a window program. Refer only to the English comments!

```
// TabSpeech.h : header-file

#if !defined(AFX_TABSPEECH_H__40089743_4A80_4F3A_928E_BED857741120__INCLUDED_)
#define AFX_TABSPEECH_H__40089743_4A80_4F3A_928E_BED857741120__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
//
/////////////////////////////////////////////////////////////////////////////
// Dialogfeld CTabSpeech

class CTabSpeech : public CDialog
{
// Konstruktion
public:
```

```
        CTabSpeech(CWnd* pParent = NULL);   // Standardkonstruktor

    // Dialogfelddaten
        //{{AFX_DATA(CTabSpeech)
        enum { IDD = IDD_SPEECH };
        CString     m_szTTSText;
        BOOL        m_bxml;
        //}}AFX_DATA
    // Überschreibungen
        // Vom Klassen-Assistenten generierte virtuelle Funktionsüberschreibungen
        //{{AFX_VIRTUAL(CTabSpeech)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV-Unterstützung
        //}}AFX_VIRTUAL
    // Own implementations
        private:

                bool        Speak(CString *strSpeakThis);
                CString ReadLine(long &i, char *FileBuffer);
    protected:
        // Generierte Nachrichtenzuordnungsfunktionen
        //{{AFX_MSG(CTabSpeech)
        afx_msg void OnTEST();
        afx_msg void OnTTS();
        afx_msg void OnClear();
        afx_msg void OnOpen();
        afx_msg void OnSave();
        afx_msg void OnXml();
        afx_msg void OnRandom();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
    };
    //{{AFX_INSERT_LOCATION}}
    // Microsoft Visual C++ fügt unmittelbar vor der vorhergehenden Zeile zusätzliche Deklarationen ein.
    #endif // AFX_TABSPEECH_H__40089743_4A80_4F3A_928E_BED857741120__INCLUDED_
```

```
// TabSpeech.cpp: Implementationfile
#include "stdafx.h"
#include "rocout.h"
#include "TabSpeech.h"
#include <time.h>                       // I use the library to generate random numbers

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
/////////////////////////////////////////////////////////////////////////////
// Dialogfeld CTabSpeech

CTabSpeech::CTabSpeech(CWnd* pParent /*=NULL*/)
    : CDialog(CTabSpeech::IDD, pParent)
{
    //{{AFX_DATA_INIT(CTabSpeech)
    m_szTTSText = _T("Enter some words you wish spoken here.");
    m_bxml = FALSE;                                     //initialize checkbox
    //}}AFX_DATA_INIT
}

void CTabSpeech::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CTabSpeech)
    DDX_Text(pDX, IDC_TEXT, m_szTTSText);
    DDX_Check(pDX, IDC_XML, m_bxml);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CTabSpeech, CDialog)
    //{{AFX_MSG_MAP(CTabSpeech)
    ON_BN_CLICKED(IDC_TEST, OnTEST)
    ON_BN_CLICKED(IDC_TTS, OnTTS)
    ON_BN_CLICKED(IDC_CLEAR, OnClear)
```

```
	ON_BN_CLICKED(IDC_OPEN, OnOpen)
	ON_BN_CLICKED(IDC_SAVE, OnSave)
	ON_BN_CLICKED(IDC_XML, OnXml)
	ON_BN_CLICKED(IDC_RANDOM, OnRandom)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()
////////////////////////////////////////////////////////////////////////////
/// Own functions, these may become moduls later

bool CTabSpeech::Speak(CString *strSpeakThis)
{
	/*a little string tutorial
		Strings: unicode characters are represented by w_char_t data type, you've to prefix the "L" character:
		example: w_char_t wchar = L'5'			//2 bytes 0x0035
		example: w_char_t* wszWord = L"HI"; //3 bytes, 3 wide characters, encoded in memory as 48 00 (=H) 49 00 (=I) 00 00 (EoS)
		strcpy(), sprintf(), atol() can only be used for single byte strings (sbcs = single byte character set)
		However, there are other functions like wcscpy(), swprintf(), _wtol() for use only with unicode strings.
		There are another type: MBCS, i'll skip it coz it is commonly used for chinese and japanese characters.
		However instead of using the strxxx() and _mbsxxx() functions, there are some macros like _tcsrchr (re-
		places strrchr() or _mbsrchr() or wcsrchr()).
		some additional example:

		CString s1 = "char string";				// construct from a constant char string
		CString s2 = L"wide char string";  // construct from a constant unicode char string

		For full documentation look under topic "Generic-Text Routine Mappings" in the MSDN-Library.
		*/
		//First Step is to convert the CString into a WCHAR ...

		WCHAR   *szWSpeakText = NULL;				//pointer to unicode characters
		//LPWSTR szWSpeakText = NULL;				//another way to do this
		int			nTextLength;
		ISpVoice *pVoice = NULL;				//setting the pointer to SAPI voice
		HRESULT hr = S_OK;					//handle SAPI

		// Find the length of the string in the editControl

		nTextLength = strSpeakThis->GetLength();		//get the lenght of the string from the edit control
		szWSpeakText = new WCHAR[ nTextLength+1 ];		//dont forget the termination of the string

		if (szWSpeakText == NULL)				//didn't get memory?
		{
		//Message box with OK_Button and ! sign, title: ERROR, MSG: "Error initializing speech objects"
		//MessageBox("Insufficient Memory, Memory could not be allocated!\nOccured while allocating memory for szWSpeakText!","ERROR", MB_OK | MB_ICONEXCLAMATION);
			AfxMessageBox("Insufficient Memory, Memory could not be allocated!\nOccured while allocating memory for
szWSpeakText!", MB_OK | MB_ICONEXCLAMATION);
			return(false);
		}
		else
		{
			//now I copy the CString object in a WCHAR object (SAPI expects WCHAR)
			mbstowcs(szWSpeakText, *strSpeakThis, nTextLength);

			/*
			another piece of code that would probably do the same. I use the line above coz it replaces all the
			lines over here.
			if you have trouble, use this code.

			LPSTR lpTempStr = NULL;				//a temporary string
			lpTempStr = m_szTTSText.GetBuffer(nTextLength);		//point to original string
			int nLpStrLen = MultiByteToWideChar(CP_ACP, 0,lpTempStr, -1,NULL, NULL);	//get the string length if
string would be a MultiByte String
			szWSpeakText = new WCHAR[ nLpStrLen + 2 ];	//allocate memory for the new unicode string, dont forget the
termination!
			MultiByteToWideChar(CP_ACP, 0, lpTempStr, -1, szWSpeakText, nLpStrLen);  //copy String to WCHAR
			delete[] lpTempStr;				//release memory
			lpTempStr = NULL;
			*/

			hr = CoCreateInstance(CLSID_SpVoice, NULL, CLSCTX_ALL, IID_ISpVoice, (void **)&pVoice);
			if( SUCCEEDED( hr ) )
			{

			// do we have to interpret XML?
			//this doesnt work I dont know why at the moment hr = pVoice->Speak( szWSpeakText, SPF_ASYNC | (m_bxml ?
			SPF_IS_XML : SPF_IS_NOT_XML), NULL);
			hr = pVoice->Speak( szWSpeakText, (m_bxml ? SPF_IS_XML : SPF_IS_NOT_XML), NULL);
			//hr = pVoice->Speak(szWSpeakText, SPF_IS_NOT_XML, NULL);		//old version without XML
```

```
                        pVoice->Release();          //Release memory
                        pVoice = NULL;
                        delete[] szWSpeakText;      //Release memory
                        szWSpeakText = NULL;
                        }
                        //if problems with SAPI initialization then error message

                        if( FAILED( hr ) )
                        {
                                //Message box with OK_Button and ! sign, title: ERROR, MSG: "Error initializing speech objects"
                                MessageBox("Error initializing speech objects","ERROR", MB_OK | MB_ICONEXCLAMATION);
                return(false);
                        }




                }

        return (true);
}


// function takes a String, traverse the string and is looking for a newline (LF CR) character
// then it isolates the line and retuns it in a CString Object.
// FileBuffer = Content of a Text File
// i all bytes in the file

CString CTabSpeech::ReadLine(long &i, char *FileBuffer)
{
   CString strLine;

        if (FileBuffer[i] != NULL)                                              // is string empty by chance?
        {
                i++;                    // skip the first sign in the file & skip CR 0x0A

                //check, if we reached LF \r or CR \n, if yes. The first byte, that we reach is probably LF 0x0D

                while ((FileBuffer[i] != '\n') && (FileBuffer[i] != NULL) && (FileBuffer[i] != '\r'))
      {
        strLine += FileBuffer[i];
                        i++;
            }// End of  For loop
        }   i = i+1;                                                                    // skip LF 0x0D
        FileBuffer = NULL;                                                              // delete pointer
        return (strLine);
}




////////////////////////////////////////////////////////////////////////////
// Routins for messages in CTabSpeech

void CTabSpeech::OnTEST()
{
        // TODO: insert here code to handle control message
        //this is routine for clicked test button
        //disable the Test button
        GetDlgItem(IDC_TEST)->EnableWindow(false);

        //setting the pointer to SAPI voice
        ISpVoice *pVoice = NULL;
        HRESULT hr = S_OK;

        hr = CoCreateInstance(CLSID_SpVoice, NULL, CLSCTX_ALL, IID_ISpVoice, (void **)&pVoice);

        if( SUCCEEDED( hr ) )
   {

                hr = pVoice->Speak(L"<pitch middle = '-10'/><rate speed = '-2'/>Hey master, I am a robot and I'm going to serve you",
SPF_IS_XML, NULL);

                pVoice->Release();          //Release memory
      pVoice = NULL;
   }
```

```
	//if problems with SAPI initialization then error message

	if( FAILED( hr ) )
  {
    //Message box with OK_Button and ! sign, title: ERROR, MSG: "Error initializing speech objects"
			MessageBox("Error initializing speech objects","ERROR", MB_OK | MB_ICONEXCLAMATION);

  }

	// and enable button again
	GetDlgItem(IDC_TEST)->EnableWindow(true);

}


void CTabSpeech::OnTTS()
{
	// TODO: if you click on button "Speech", the following code is executed

	//disable Speak button
	GetDlgItem(IDC_TTS)->EnableWindow(FALSE);

	/*
	another posiblity

	CWnd* pm_hSpeechPage = NULL;				//pointer to store a handle the speech sheet window
	pm_hSpeechPage = GetDlgItem(IDC_TTS);			//convert a control ID to a CWnd pointer, here to the sheet-window

	::EnableWindow( ::GetDlgItem( pm_hSpeechPage, IDC_TTS ), false );	//disable button
	or

	pm_hSpeechPage->EnableWindow(false);
	*/


	UpdateData(true);				//text -> variable

	// check if the string is empty because it's not worth to perform the whole procedure if no speech
	if (m_szTTSText.IsEmpty())
	{
		//Warning
		MessageBox("You should add some words before you click speak","Hint", MB_OK |  MB_ICONINFORMATION);
	}
	else
	{
		if (!Speak(&m_szTTSText))
		{
			//Something was wrong with the output
			MessageBox("Maybe the input file or the text does not fit the specifications", "Error(s) in Synthesis module", MB_OK
| MB_ICONINFORMATION);
		}
	}

	// and enable button again
	GetDlgItem(IDC_TTS)->EnableWindow(TRUE);

	/*
	the other posiblity

	::EnableWindow( ::GetDlgItem( pm_hSpeechPage, IDC_TTS ), true );	//enable Button
	pm_hSpeechPage = NULL;
	*/
}

void CTabSpeech::OnClear()
{
	// TODO: Code for clear Text in control edit box
	UpdateData (true);
	if (m_szTTSText.IsEmpty()) {}
	else
	{
		m_szTTSText.Empty();
		UpdateData (false);
	}


}
```

```
void CTabSpeech::OnOpen()
{
	// TODO: this routine opens a file reads the content and stores it in the editctrl window

/*

CFileDialog( BOOL bOpenFileDialog, LPCTSTR  lpszDefExt = NULL, LPCTSTR lpszFileName  = NULL,
DWORD dwFlags = OFN_HIDEREADONLY  | OFN_OVERWRITEPROMPT, LPCTSTR lpszFilter  = NULL, CWnd* pParentWnd = NULL  );

bOpenFileDialog : Set to TRUE to construct a File Open dialog box or FALSE to construct a File Save As dialog box.

lpszDefExt : The default filename extension. If the user does not include an extension in the Filename edit box,
the extension specified by lpszDefExt is automatically appended to the filename. If this parameter is NULL, no file extension is appended.

lpszFileName : The initial filename that appears in the filename edit box. If NULL, no filename initially appears.

dwFlags: A combination of one or more flags that allow you to customize the dialog box. For a description of these flags,
see the OPENFILENAME structure in the Win32 SDK documentation. If you modify the m_ofn.Flags structure member,
use a bitwise-OR operator in your changes to keep the default behavior intact.

lpszFilter :A series of string pairs that specify filters you can apply to the file. If you specify file filters,
only selected files will appear in the Files list box. See the Remarks section for more information on how to work with file filters.

pParentWnd :A pointer to the file dialog-box object's parent or owner window.
*/


	CString		sPathname;		//stores the path to the file
	char		szBuffer[_MAX_PATH];	// maximum path length
	int		nByteCount;		//count of bytes read from the file
	const int	max=64000;		//max file size in byte

	static char szFilter[] = _T("Speech Synthesis Text (*.sst)|*.sst|")		//spezial file filters, that are useful for us
		_T("SAPI XML files (*.xml)|*.xml|")
		_T("All Files (*.*)|*.*||");

	//Call the Dialog
	CFileDialog m_dlgFile(TRUE, NULL, NULL ,OFN_HIDEREADONLY | OFN_FILEMUSTEXIST | OFN_PATHMUSTEXIST,szFilter,
this  );  // TRUE = open a file,FALSE = save file

	VERIFY(::GetModuleFileName(AfxGetInstanceHandle(), szBuffer, _MAX_PATH));
							// get the path of the executed program
	CString sPath = (CString)szBuffer;
	                                                                                                        // converting ...
	sPath = sPath.Left(sPath.ReverseFind('\\'));
	m_dlgFile.m_ofn.lpstrInitialDir = _T(sPath);
	                                                                        // set the application path as the default path

	if( m_dlgFile.DoModal() == IDOK )
	// is button OK clicked
	{
		sPathname = m_dlgFile.GetPathName();                                                    // gets the path of the file
		CString czFileExtension = m_dlgFile.GetFileExt();                                       // get the file extension
		czFileExtension.MakeLower();
	// make low letters

		if ( czFileExtension =="xml")
		{
			m_bxml=true;
		}

		TCHAR czLoadString[max];
		// new String
		CFile FileToBeOpen;
			// new CFile Object
	//	CFileException fileException;

		//Error Handling, important if you work with files!
	//	if ( !FileToBeOpen.Open( sPathname, CFile::modeRead, &fileException) )
	//	{
	//	TRACE( "Can't open file %s, error = %u\n", sPathname, fileException.m_cause );
	//	}


		FileToBeOpen.Open( sPathname, CFile::modeRead );                                // open the file in read only mode
		DWORD lFileLength = FileToBeOpen.GetLength();                                           // get the file length
```

```
		if (lFileLength > max)
		// compare it with the maximum file length
		{
			AfxMessageBox("File is too Big!");
			return;
		}

		FileToBeOpen.Seek( 0, CFile::begin );                                                                    // goto position 0
		nByteCount = FileToBeOpen.Read( czLoadString, sizeof( czLoadString) );    //Read the file and count read bytes
		//Integer to String itoa(value, string, base);
		//char temp[4];
		//itoa(nAByteCount, temp, 10);
		//MessageBox(temp);
		FileToBeOpen.Close();
			// close file
		czLoadString[nByteCount] = NULL;
	// shorten the readed String to it's actual lenght
		m_szTTSText = czLoadString;
		// copy it
		UpdateData( FALSE );
			// and update the control
	}

}

void CTabSpeech::OnSave()
{
	// TODO: saves the data in the edit ctrl window

	UpdateData( true );                                                    // update all variables

	CString		sPathname;                                         //stores the path to the file
	char szBuffer[_MAX_PATH];                                          // maximum path length
	//int                    nByteCount;                                 //count of bytes written into the file
	const int    max=64000;                                            //max file size in byte

	static char szFilter[] = _T("Speech Synthesis Text (*.SST)|*.sst|")        //spezial file filters, that are useful for us
		_T("SAPI XML files (*.xml)|*.xml|")
		_T("All Files (*.*)|*.*||");

	//Call the Dialog
	CFileDialog m_dlgFile(FALSE, NULL, NULL ,OFN_HIDEREADONLY | OFN_FILEMUSTEXIST | OFN_PATHMUSTEXIST,szFilter,
this  );  // TRUE = open a file,FALSE = save file

	VERIFY(::GetModuleFileName(AfxGetInstanceHandle(), szBuffer, _MAX_PATH));    // get the path of the executed program
	CString sPath = (CString)szBuffer;                                           // converting ...
	sPath = sPath.Left(sPath.ReverseFind('\\'));
	m_dlgFile.m_ofn.lpstrInitialDir = _T(sPath);
                                                                                 // set the application path as the default path

	if( m_dlgFile.DoModal() == IDOK )
	// is button OK clicked
	{
		sPathname = m_dlgFile.GetPathName();                                 // gets the path of the file
		const long lLength = m_szTTSText.GetLength()+1;                      // lenght of the string that may be stored
		if (lLength > 4096)
		{
			AfxMessageBox("That amount of data can't be stored!\nReduse it and try again.");
			return;
		}
		TCHAR czSaveString[4096];
	// new String
		czSaveString[lLength]=NULL;
	// cut String to its real length
		_tcscpy(czSaveString, m_szTTSText);                                                                      //
copies variable in a new String

		CFile FileToBeStore( sPathname, CFile::modeCreate | CFile::modeWrite);

	//		CFileException fileException;
	//		if ( !FileToBeStore.Open( sPathname, CFile::modeCreate|CFile::modeWrite|CFile::typeText|CFile::shareDenyNone,
&fileException ) )
	//		{
	//			TRACE( "Can't create file %s, error = %u\n", sPathname, fileException.m_cause );
	//		}
```

```
		FileToBeStore.Write( czSaveString, sizeof(czSaveString) );	//Read the file and count read bytes
		//Integer to String itoa(value, string, base);
		//char temp[4];
		//itoa(nAByteCount, temp, 10);
		//MessageBox(temp);
		FileToBeStore.Close();
			// close file
	}

	UpdateData(false);

}

void CTabSpeech::OnXml()
{
	// TODO: Code for checkbox element XML

	if (m_bxml)
	{
		m_bxml = false;
	}
	else
	{
/*		//we have to test, if the text has XML-format and if it is SAPI conform
		//im looking only for an incomplete tag because it is allowed to specify the
		//xml language version in the first tag
		int nXml = m_szTTSText.Find("<xml");
		if (nXml == -1)
		{
			//Noting was found, perhaps we should look for upper letters
			nXml = m_szTTSText.Find("<XML");
		}
		int nSapi = m_szTTSText.Find("<sapi>");
		if (nSapi == -1)
		{
			//Noting was found, perhaps we should look for upper letters
			nSapi = m_szTTSText.Find("<SAPI>");
		}

		if (nSapi && nXml)
		m_bxml = true;
		else
		//Warning
		MessageBox("This is not a plain XML SAPI file!\nYou can't check the box unless it is SAPI XML file!","Attention!", MB_OK |
MB_ICONINFORMATION);
*/
		m_bxml = true;
	}

//	UpdateData(false);

}

void CTabSpeech::OnRandom()
{
	// generates random sentence and speaks these.
	// data and source code is separated please look to random.txt for further information

	GetDlgItem(IDC_RANDOM)->EnableWindow(false);

	srand(time(NULL));	// intialize time for generating random
numbers

	CString	strTemp;
	bool	bRead=false;
	char	cCountSubjects=0;
	char	cCountObjects=0;
	char	cCountTense=0;
	char	cCountPredicate=0;
	char	szBuffer[_MAX_PATH];		// maximum path length
	char	*pBuff;	// file buffer

	long	lFileSize;			// filelength in bytes
	long	FilePos=0;


	struct	lelem
```

```
	{
		CString str;
		lelem *prev, *next;

	};

	lelem *anchorSub = NULL;                                        // create a new anchor pointer for Subject
	lelem *listptrSub = NULL;
	lelem *anchorObj = NULL;                                        // create a new anchor pointer for Object
	lelem *listptrObj = NULL;
	lelem *anchorPre = NULL;                                        // create a new anchor pointer for Predicate
	lelem *listptrPre = NULL;
	lelem *anchorTen = NULL;                                        // create a new anchor pointer for Tense
	lelem *listptrTen = NULL;


	// at first read the file with the pieces of sentence

	VERIFY(::GetModuleFileName(AfxGetInstanceHandle(), szBuffer, _MAX_PATH));
                                                        // get the path of the executed program
	CString sPath = (CString)szBuffer;
                                                                                                                        // converting ...
	sPath = sPath.Left(sPath.ReverseFind('\\'));
	sPath +="\\random.txt";

	CFile InFile;
	InFile.Open(sPath, CFile::modeRead);

	// Note the following 2 lines could be replaced with:
 // lFileSize = file.GetLength();
	lFileSize = InFile.Seek(0, CFile::end);
 InFile.Seek(0, CFile::begin);

	pBuff = new char[lFileSize+1];

	if (pBuff == NULL)
  {
    AfxMessageBox("Memory Allocation problem");
    return;
  }

	InFile.Read(pBuff, lFileSize);
	pBuff[lFileSize] = 0;

	/* my old version

	CArchive ar(&InFile,CArchive::load);
	ar>>strTemp;                                                                                    // read the first line

*/
	strTemp = ReadLine(FilePos, pBuff);                                             // read the first line

	while (strTemp != "EOF")
	{

		if ((strTemp.Left(1) != ";") && (strTemp != ""))          // not empty or a comment (;)
		{
			if (strTemp.Left(1) == "!")
			{
				//found a section
				strTemp.Delete(0,1);                                                    // Remove the first character

				if (strTemp == "Subject")
				{
					//found the Subject section in the file
					strTemp = ReadLine(FilePos, pBuff);

					while ((strTemp.Left(1) == ";") || (strTemp == ""))       // skip all lines without content
						strTemp = ReadLine(FilePos, pBuff);

					//its time to set up our struture to store the data ...

					anchorSub = new lelem;
				// create a new anchor

					if (!anchorSub)
					// didnt get memory, that's a pitty!
```

```
                                        {
                                                //immediately a message out
                                                MessageBox("Random Demo: Out of Memory!","ERROR", MB_OK |
MB_ICONEXCLAMATION);
                                                return;                         // exit
                                        }

                                listptrSub = anchorSub;
                                        // auxiliary pointer to the actual list element
                                listptrSub->str = strTemp;                      // first piece of sentence
                                listptrSub->prev = NULL;                        // the first element hasn't a previous
element
                                cCountSubjects++;

                                strTemp = ReadLine(FilePos, pBuff);
                                                                        // we need new data!

                                while ((strTemp.Left(1) != ";") && (strTemp != ""))     // if we find piece of sentence then ...
                                {
                                                                        // put'em to the list!
                                        listptrSub->next = new lelem;
                        // create new element

                                         if (!listptrSub->next)                 // didn't get memory?
                                         {
                                                MessageBox("Random Demo: Out of Memory!","ERROR", MB_OK |
MB_ICONEXCLAMATION);   //sorry!
                                                while (listptrSub->prev)
                                                 {
                                                        listptrSub = listptrSub->prev;
                                                        delete listptrSub->next;                // Zombie-Killer
                                                 }
                                                delete anchorSub;                       // remove everything!
                                                return;                                 // exit
                                                }

                                                listptrSub->next->prev = listptrSub;            // Set the pointers for the
new element
                                                listptrSub = listptrSub->next;                  // our the traverse pointer has
to move as well
                                                listptrSub->str = strTemp;
                                         // store next line
                                                cCountSubjects++;
                                                strTemp = ReadLine(FilePos, pBuff);
                                }
                                  listptrSub->next = NULL;                      // we've to terminate the last element
                                  listptrSub = anchorSub;
                                  bRead = true;

                        } // this is the end of the if for subject

                        //*************************************
                        //We have to do the same for predicate
                        //*************************************

                        if (strTemp == "Predicate")
                        {
                                //found the Predicate section in the file
                                strTemp = ReadLine(FilePos, pBuff);

                                while ((strTemp.Left(1) == ";") || (strTemp == ""))     // skip all lines without content
                                        strTemp = ReadLine(FilePos, pBuff);

                                //its time to set up our struture to store the data ...

                                anchorPre = new lelem;
                        // create a new anchor

                                if (!anchorPre)
                                // didnt get memory, that's a pitty!
                                        {
                                                //immediately a message out
                                                MessageBox("Random Demo: Out of Memory!","ERROR", MB_OK |
MB_ICONEXCLAMATION);
                                                return;                         // exit
                                        }
```

```
                                        listptrPre = anchorPre;
                                // auxiliary pointer to the actual list element
                                        listptrPre->str = strTemp;                      // first piece of sentence
                                        listptrPre->prev = NULL;                        // the first element hasn't a previous element
                                        cCountPredicate++;

                                        strTemp = ReadLine(FilePos, pBuff);
                                                                                        // we need new data!

                                        while ((strTemp.Left(1) != ";") && (strTemp != ""))     // if we find piece of sentence then ...
                                        {
                                                                                        // put'em to the list!
                                                listptrPre->next = new lelem;       // create new element

                                                 if (!listptrPre->next)                         // didn't get memory?
                                                 {
                                                        MessageBox("Random Demo: Out of Memory!","ERROR", MB_OK |
MB_ICONEXCLAMATION);   //sorry!
                                                        while (listptrPre->prev)
                                                         {
                                                                listptrPre = listptrPre->prev;
                                                                delete listptrPre->next;                        // Zombie-Killer
                                                         }
                                                        delete anchorPre;                       // remove everything!
                                                        return;                                 // exit
                                                       }

                                                 listptrPre->next->prev = listptrPre;                   // Set the pointers for the new
element
                                                 listptrPre = listptrPre->next;                         // our the traverse pointer has to
move as well
                                                 listptrPre->str = strTemp;
                                 // store next line
                                                 cCountPredicate++;
                                                 strTemp = ReadLine(FilePos, pBuff);
                                        }
                                          listptrPre->next = NULL;                      // we've to terminate the last element
                                          listptrPre = anchorPre;
                                          bRead = true;

                                } // this is the end of the if for predicate

                                //************************************
                                //We have to do the same for Object
                                //************************************

                                if (strTemp == "Object")
                                {
                                        //found the Object section in the file
                                        strTemp = ReadLine(FilePos, pBuff);

                                        while ((strTemp.Left(1) == ";") || (strTemp == ""))                             // skip all lines
without content
                                                strTemp = ReadLine(FilePos, pBuff);

                                        //its time to set up our struture to store the data ...

                                        anchorObj = new lelem;
                                // create a new anchor

                                        if (!anchorObj)
                                        // didnt get memory, that's a pitty!
                                                {
                                                        //immediately a message out
                                                        MessageBox("Random Demo: Out of Memory!","ERROR", MB_OK |
MB_ICONEXCLAMATION);
                                                        return;                                 // exit
                                                }

                                        listptrObj = anchorObj;
// auxiliary pointer to the actual list element
                                        listptrObj->str = strTemp;                      // first piece of sentence
                                        listptrObj->prev = NULL;                        // the first element hasn't a previous
element
                                        cCountObjects++;
```

```
                    strTemp = ReadLine(FilePos, pBuff);
                                                        // we need new data!

                    while ((strTemp.Left(1) != ";") && (strTemp != ""))                  // if
we find piece of sentence then ...
                    {
                                                        // put'em to the list!

                            listptrObj->next = new lelem;
                // create new element

                             if (!listptrObj->next)                 // didn't get memory?
                             {
                                    MessageBox("Random Demo: Out of Memory!","ERROR", MB_OK |
MB_ICONEXCLAMATION);   //sorry!

                                    while (listptrObj->prev)
                                     {
                                            listptrObj = listptrObj->prev;
                                            delete listptrObj->next;                // Zombie-Killer
                                     }
                                    delete anchorObj;                       // remove everything!
                                    return;                                 // exit
                                    }

                             listptrObj->next->prev = listptrObj;           // Set the pointers for the new
element
                             listptrObj = listptrObj->next;                 // our the traverse pointer has to
move as well
                             listptrObj->str = strTemp;
                 // store next line
                             cCountObjects++;
                             strTemp = ReadLine(FilePos, pBuff);
                    }
                      listptrObj->next = NULL;                      // we've to terminate the last element
                      listptrObj = anchorObj;
                      bRead = true;
            } // this is the end of the if for Object
            if (strTemp == "Tense")
            {

                    //***************************************
                    //finally, we've only the pieces for tense
                    //***************************************

                    strTemp = ReadLine(FilePos, pBuff);

                    while ((strTemp.Left(1) == ";") || (strTemp == ""))                 // skip all lines
without content
                            strTemp = ReadLine(FilePos, pBuff);

                    //its time to set up our struture to store the data ...

                    anchorTen = new lelem;
            // create a new anchor

                    if (!anchorTen)
                    // didnt get memory, that's a pitty!
                            {
                                    //immediately a message out
                                    MessageBox("Random Demo: Out of Memory!","ERROR", MB_OK |
MB_ICONEXCLAMATION);

                                    return;                         // exit
                            }

                    listptrTen = anchorTen;                                     // auxiliary
pointer to the actual list element
                    listptrTen->str = strTemp;                  // first piece of sentence
                    listptrTen->prev = NULL;                    // the first element hasn't a previous
element
                    cCountTense++;

                    strTemp = ReadLine(FilePos, pBuff);
            // we need new data!

                    while ((strTemp.Left(1) != ";") && (strTemp != ""))                 // if
we find piece of sentence then ...
```

```
                                        {
                                                                        // put'em to the list!

                                                listptrTen->next = new lelem;
                                // create new element

                                                if (!listptrTen->next)                  // didn't get memory?
                                                 {
                                                        MessageBox("Random Demo: Out of Memory!","ERROR", MB_OK |
MB_ICONEXCLAMATION);   //sorry!
                                                        while (listptrTen->prev)
                                                         {
                                                                listptrTen = listptrTen->prev;
                                                                delete listptrTen->next;                // Zombie-Killer
                                                         }
                                                        delete anchorTen;                       // remove everything!
                                                        return;                                 // exit
                                                       }

                                                listptrTen->next->prev = listptrTen;            // Set the pointers for the new
element
                                                listptrTen = listptrTen->next;                  // our the traverse pointer has to
move as well
                                                listptrTen->str = strTemp;
                                // store next line
                                                cCountTense++;
                                                strTemp = ReadLine(FilePos, pBuff);
                        // read new line
                                        }
                                          listptrTen->next = NULL;                      // we've to terminate the last element
                                          listptrTen = anchorTen;
                                          bRead = true;

                                        }//end if tense

                        }//end looking for a new section

                }//end not a comment or empty line
                if (bRead == false)
                {
                        strTemp = ReadLine(FilePos, pBuff);
                                 // read a new line
                }
                bRead = false;
        }//End first While

InFile.Close();
delete []pBuff;
pBuff = NULL;

// Okay, the file is loaded and we can start the game ...
// at first we need 4 random numbers, to select one sentence of each section;


// we start with the the generation. We use the rand() function modulo the maximum # of sentence. Thus, we get numbers between 0 and nCount... -1

int nRandSub    =       rand()%(cCountSubjects+1);      // get a random number for Subject
int nRandPred   =       rand()%(cCountPredicate+1);     // get a random number for Predicate
int nRandObj    =       rand()%(cCountObjects+1);       // get a random nunber for Object
int nRandTens   =       rand()%(cCountTense+1);                 // get a random number for Tense

// lets form the sentence:

CString cstrFinal;

//Build the string ...

if (nRandSub <= cCountSubjects)                                         // works the random gernerator properly?
{
        for (char i=1; i<nRandSub; i++)
        {
                listptrSub = listptrSub->next;                          // traverse the list to the right point
        }

        cstrFinal = listptrSub->str;
        cstrFinal += " ";
```

```
	if (nRandPred <= cCountPredicate)
	{

		for (char j=1; j<nRandPred; j++)
		{
			listptrPre = listptrPre->next;
		}

		cstrFinal += listptrPre->str;
		cstrFinal += " ";

		if (nRandObj <= cCountObjects)
		{
			for (char k=1; k<nRandObj; k++)
			{
				listptrObj = listptrObj->next;
			}

			cstrFinal += *listptrObj->str;
			cstrFinal += " ";

			if (nRandTens <=cCountTense)
			{
				for (char l=1; l<nRandTens; l++)
				{
					listptrTen = listptrTen->next;
				}

				cstrFinal += listptrTen->str;
				cstrFinal += ".";

			}
			else
			{
				AfxMessageBox("An Error occured while processing the random module!\nExcat position: nRandTens",
MB_ICONEXCLAMATION | MB_OK);
				return;
			}

		}
		else
		{
			AfxMessageBox("An Error occured while processing the random module!\nExcat position: nRandObj",
MB_ICONEXCLAMATION | MB_OK);
			return;
		}

	}
	else
	{
		AfxMessageBox("An Error occured while processing the random module!\nExcat position: nRandPred",
MB_ICONEXCLAMATION | MB_OK);
		return;
	}

}
else
{
	AfxMessageBox("An Error occured while processing the random module!\nExcat position: nRandSub", MB_ICONEXCLAMATION |
MB_OK);
	return;
}


// output


if (!Speak(&cstrFinal))
		{
			//Something was wrong with the output
			MessageBox("Maybe it helps to try again", "Error(s) in Random Synthesis module", MB_OK |
MB_ICONINFORMATION);
		}


// we have to release the memory after all!
```

```
listptrSub = anchorSub;

while (listptrSub->next)
{
	listptrSub = listptrSub->next;
	delete listptrSub->prev;
}

delete listptrSub;
anchorSub = NULL;
listptrSub = NULL;


listptrPre = anchorPre;

while (listptrPre->next)
{
	listptrPre = listptrPre->next;
	delete listptrPre->prev;
}

delete listptrPre;
anchorPre = NULL;
listptrPre = NULL;

listptrObj = anchorObj;

while (listptrObj->next)
{
	listptrObj = listptrObj->next;
	delete listptrObj->prev;
}

delete listptrObj;
anchorObj = NULL;
listptrObj = NULL;

listptrTen = anchorTen;

while (listptrTen->next)
{
	listptrTen = listptrTen->next;
	delete listptrTen->prev;
}

delete listptrTen;
anchorTen = NULL;
listptrTen = NULL;


//enable button
	GetDlgItem(IDC_RANDOM)->EnableWindow(true);

}
```