

第四章 语法分析

Syntax Analysis

Part I

主要内容

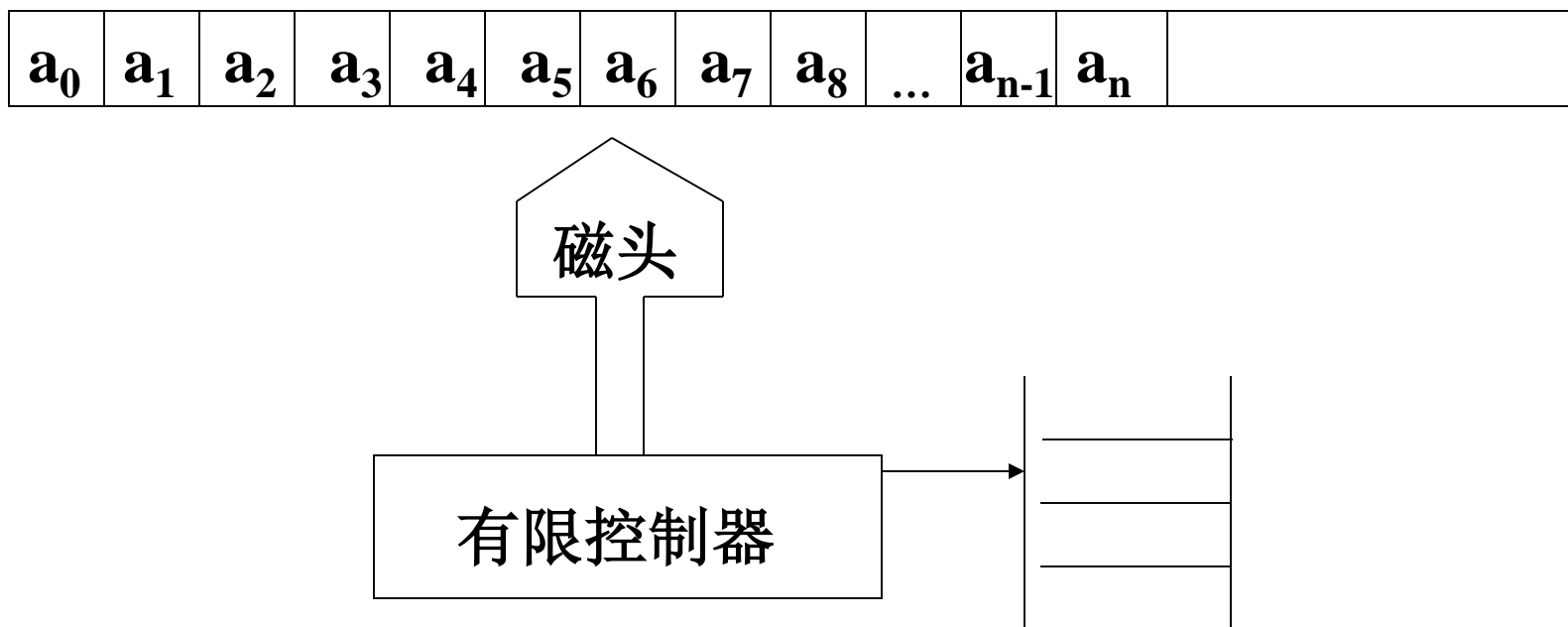
- PDA
- 语法分析概述
- 递归子程序法
- 常用终结符号集计算
- LL(1)分析方法
- LR分析方法

1、下推自动机 (PDA)

- Push Down Automata
- $M=(S, \Sigma, \Gamma, \delta, K, x_0, F)$
- S : 状态集合
- Σ : 输入字母表
- Γ : 下推字母表
- $\delta: S \times (\Sigma \cup \{\}) \times \Gamma \rightarrow S \times \Gamma^*$
- K : 初态集合
- x_0 : 下推栈中的初始符号
- F : 终态集合

- $\delta (S_i, a, x_k) = (S_j, \beta)$
- ϵ -转换：输入符号全部读完，但PDA的状态仍然可以进行转换

识别程序的数学模型下推自动机



例:

- $S = \{S_0\}$
- $\Sigma = \{ (,) \}$
- $\Gamma = \{ A, (\}$
- $\delta: S \times (\Sigma \cup \{ \epsilon \}) \times \Gamma \rightarrow S \times \Gamma^*$
- $K = \{S_0\}$
- $x_0 = A$
- $F = \{S_0\}$

$$\delta (S_0, (, A) = \{ (S_0, (A) \}$$

$$\delta (S_0, (, () = \{ (S_0, (() \}$$

$$\delta (S_0,), () = \{ (S_0, \epsilon) \}$$

$$\delta (S_0, \epsilon, A) = \{ (S_0, \epsilon) \}$$

对于输入串: $((()())$

如何进行识别?

例: PDA $P = (\{A, B, C\}, \{a, b, c\}, \{h, i\}, f, A, i, \{\})$

$f(A, a, i) = (B, h)$ $f(B, a, h) = (B, hh)$

$f(C, b, h) = (C, \varepsilon)$ $f(A, c, i) = (A, \varepsilon)$

$f(B, c, h) = (C, h)$

接受输入串aacbb的过程

$(A, aacbb, i)$ 读a, pop i, push h, goto B

$(B, acbb, h)$ 读a, pop h, push hh, goto B

(B, cbb, hh) 读c, pop h, push h, goto C

(C, bb, hh) 读b, pop h, push ε , goto C

(C, b, h) 读b, pop h, push ε , goto C

$(C, \varepsilon, \varepsilon)$

PDA与语法分析

- PDA: NPDA, DPDA
- 2型文法-程序设计语言-PDA

终止和接受的条件：

- 1.到达输入串结尾时，处在F中的一个状态(终态)
- 或
- 2.某个动作序列导致栈空时

2、语法分析概述

- 语法分析任务
- 语法分析分类

2、语法分析概述

- 语法分析任务：根据语法规则逐一分析词法分析得到的属性字（单词序列），检查语法错误，若无错，则给出正确的语法结构；若有错，则报错。

2、语法分析概述

- 语法分析分类
 - 自顶向下top-down parsing
 - 自底向上bottom-up parsing

句型的分析算法分类

分析算法可分为：

自上而下分析法：

从文法的开始符号出发，反复使用文法的产生式，寻找与输入符号串匹配的推导。

自下而上分析法：

从输入符号串开始，逐步进行归约，直至归约到文法的开始符号。

自顶向下

- **$G[Z]$:**

- **$Z \rightarrow aBd$**

- **$B \rightarrow d \mid c$**

- **$B \rightarrow bB$**

给定符号串 **abcd**, 如何 **$Z^* \Rightarrow abcd$** ?

PDA模拟

设下推栈#S，状态控制器中状态只有一个，整个分析过程是在语法分析程序控制下进行的：

- 1、若栈顶X为 V_n ，则查询语法分析表，找出一个以X为左部的产生式（语法规则），将X弹出栈，而把产生式右部的符号串以从右向左的次序进栈（推导）
- 2、若栈顶X为 V_t ，且读头所指向的输入符号也是X，则匹配。此时，X出栈，读头右移
- 3、**ERROR**：若栈顶X为 V_t ，且读头所指向的输入符号不是X，则说明前面推导时选错了规则，应退到上次规则之前（回溯）
- 4、重新选规则进行推导
- 5、若栈内只有栈底符号#，而读头也指到了输入的最后符号#，则分析成功

- 自顶向下的关键问题是：
每次该选择哪条规则

自底向上

- **G[S]**
 - **S**->**aAcBe**
 - **A**->**Ab | b**
 - **B**->**d**
- **abbcde**

自底向上的栈式分析过程

- 从输入串依次读入输入符号，直到一个简单短语出现在分析栈的栈顶，然后将栈顶的简单短语归约成相应的 V_n ,重复上述过程，直到栈中只剩下开始符号，而输入串全部被处理完。

- 自底向上主要问题是**何时应该归约**，以及**选择哪条规则进行归约**。

句型分析的有关问题

1) 在自上而下的分析方法中如何选择使用哪个产生式进行推导？

假定要被代换的最左非终结符号是B，且有n条规则： $B \rightarrow A_1 | A_2 | \dots | A_n$ ，那么如何确定用哪个右部去替代B？

2) 在自下而上的分析方法中如何识别可归约的串？

在分析程序工作的每一步，都是从当前串中选择一个子串，将它归约到某个非终结符号，该子串称为“可归约串”

刻画“可归约串”

文法 $G[S]$

句型的短语

$S \Rightarrow^* \alpha A \delta$ 且 $A \Rightarrow^+ \beta$ ，则称 β 是句型 $\alpha \beta \delta$ 相对于非终结符 A 的短语

句型的直接短语

若有 $A \Rightarrow \beta$ ，则称 β 是句型 $\alpha \beta \delta$ 相对于非终结符 A 的直接短语

句型的句柄

一个句型的最左直接短语称为该句型的句柄

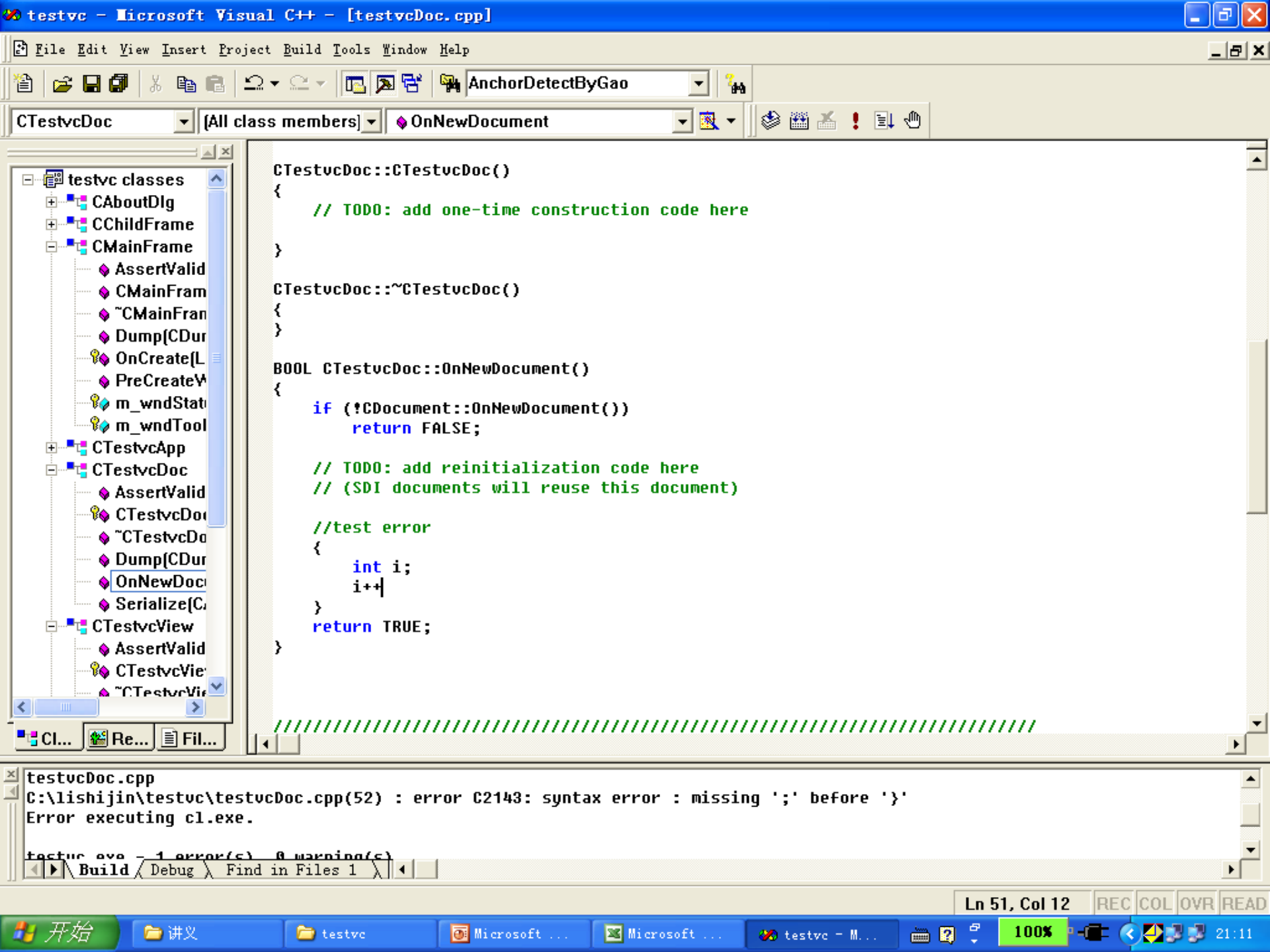
2、语法分析概述

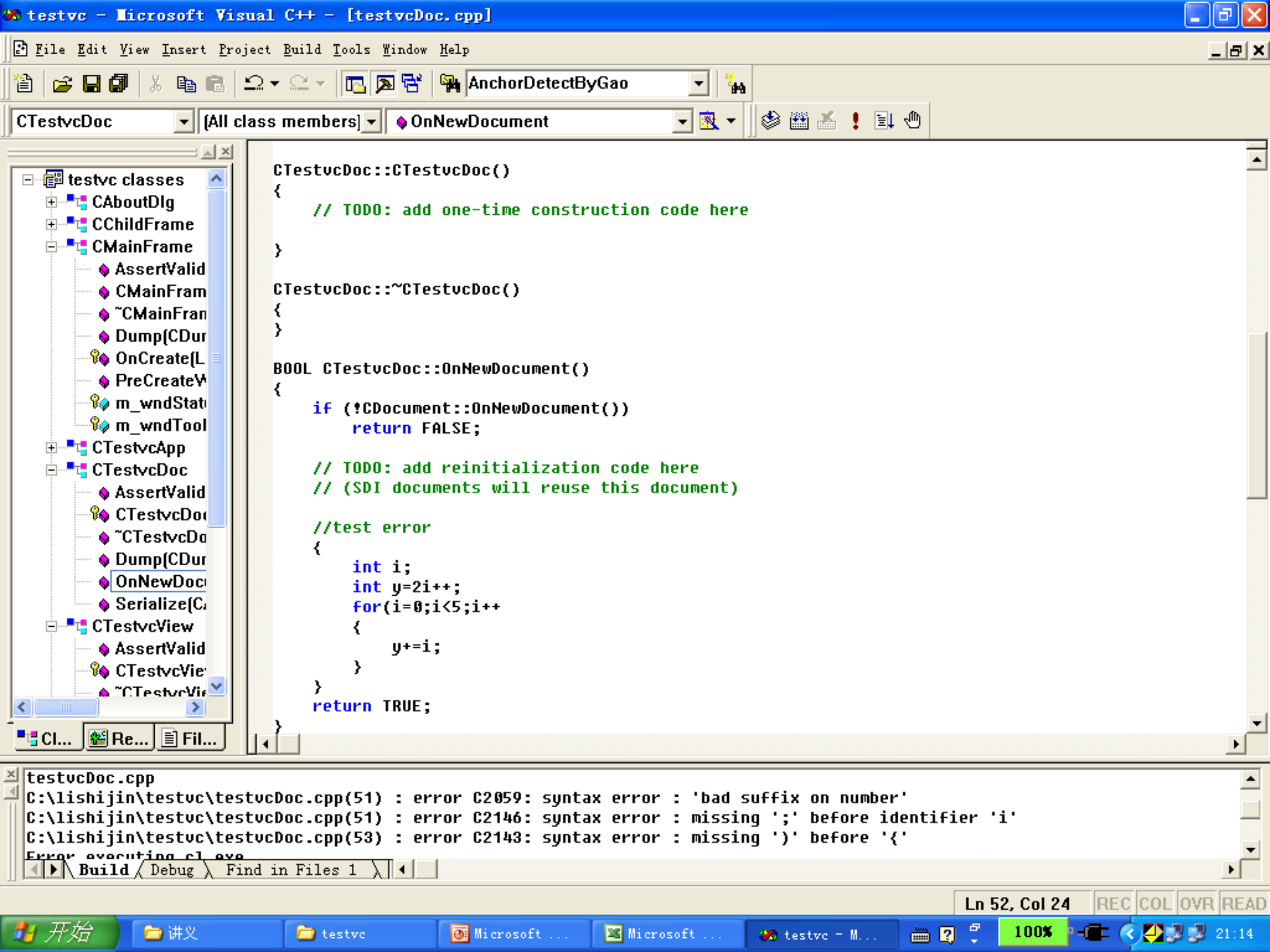
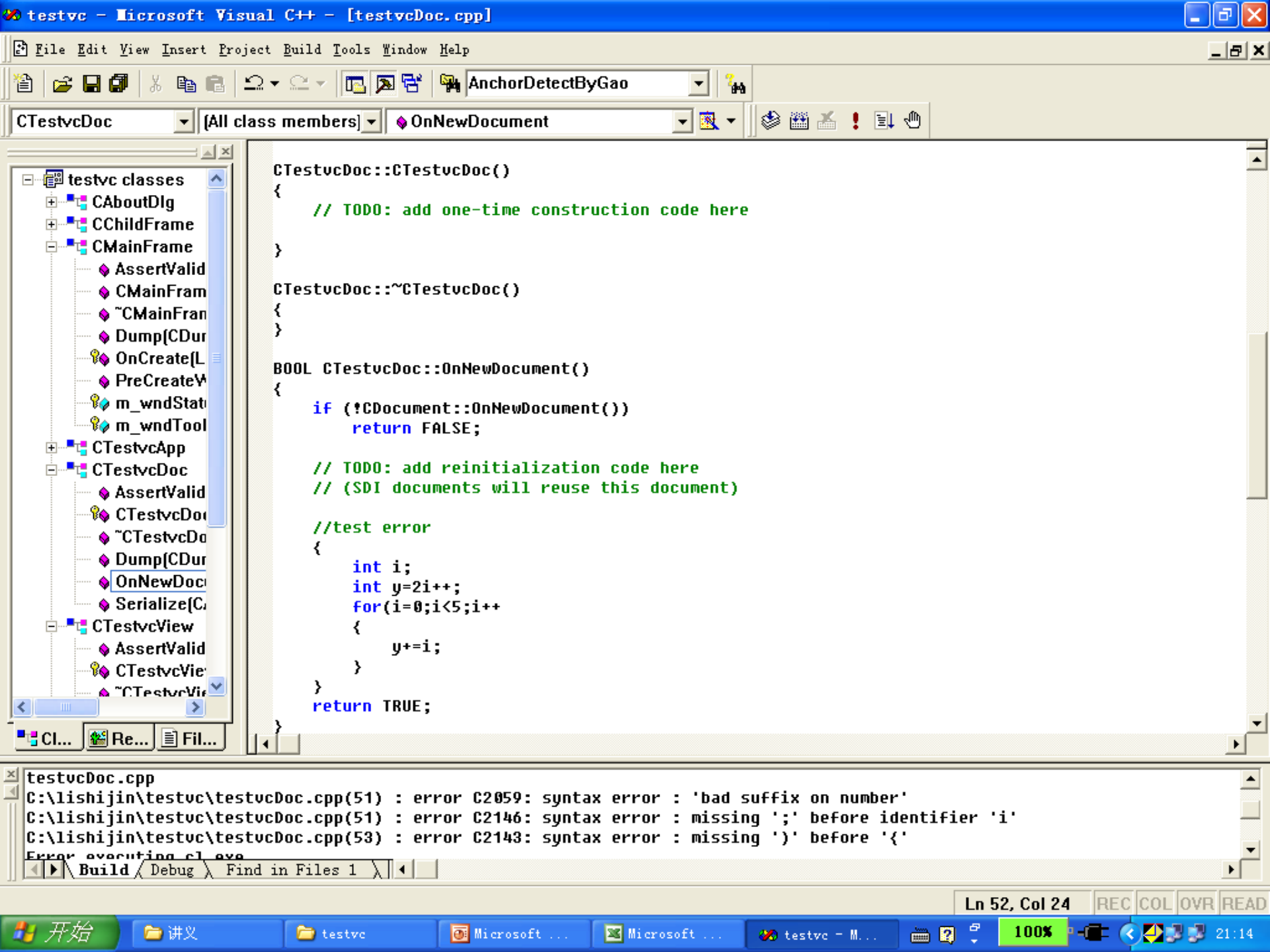
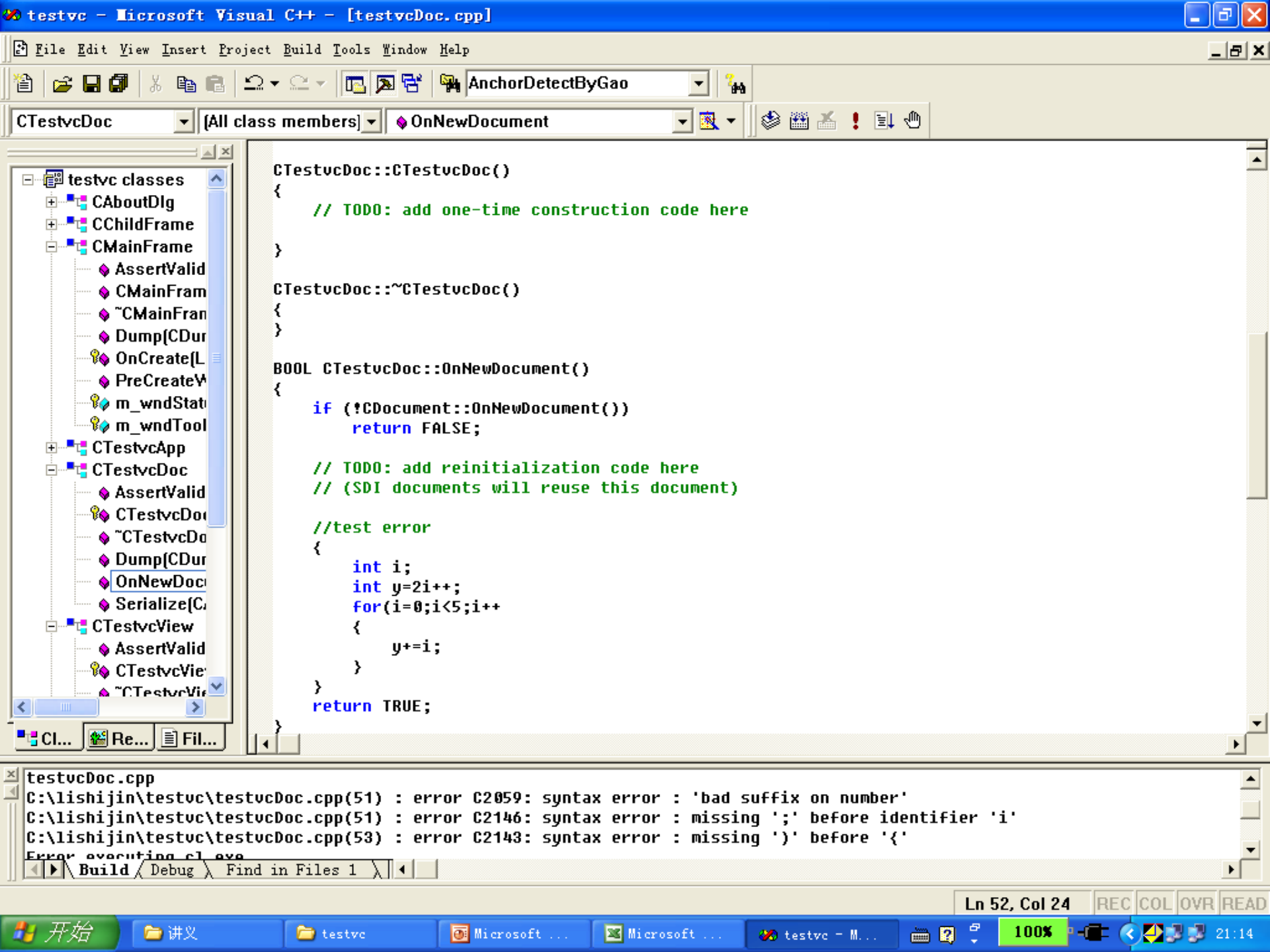
- 错误处理

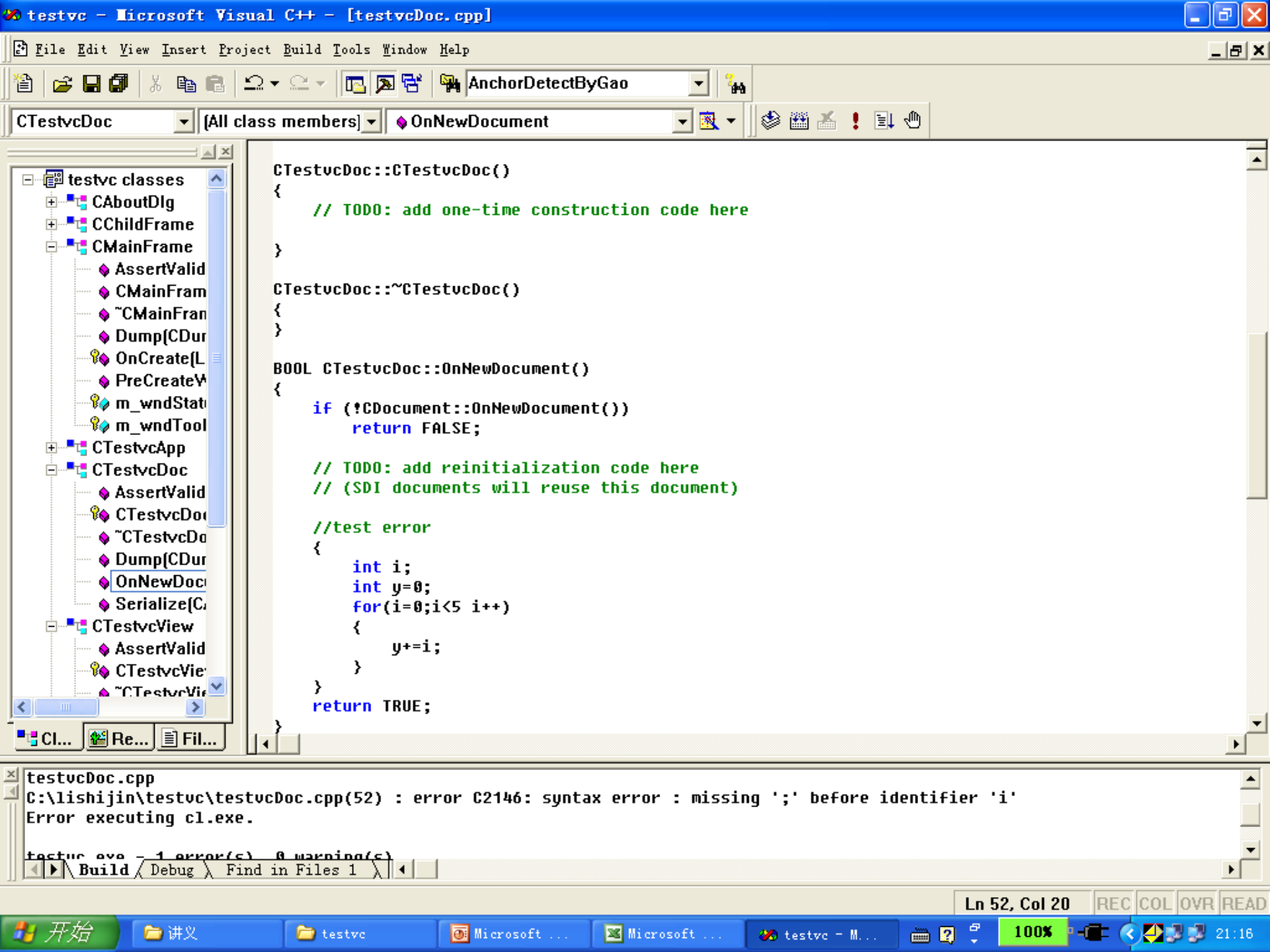


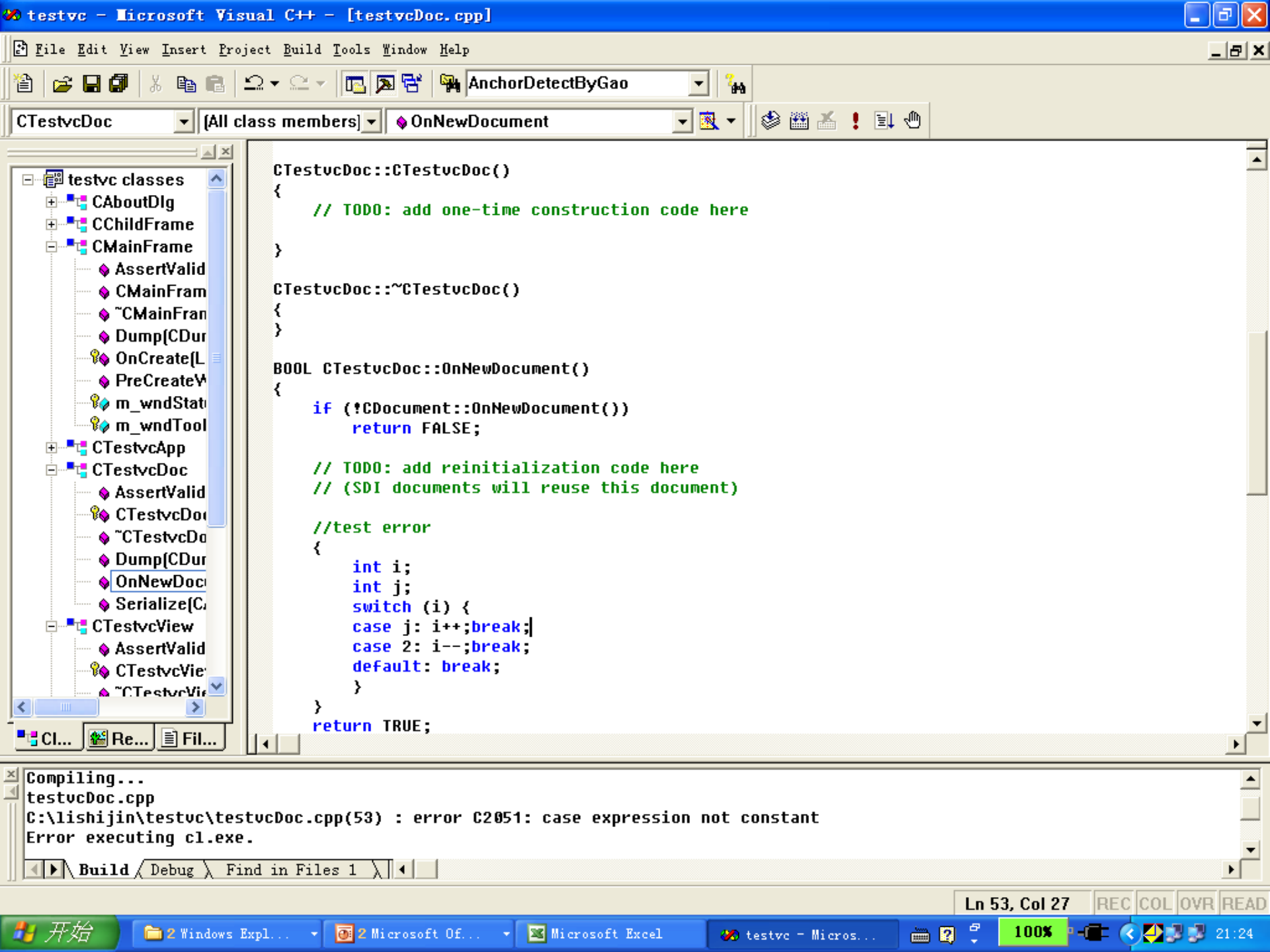
错误的种类

- 开始单词错
- 后继单词错
- 标识符和常量错
- 括号类配对错
- 分隔符错









3、递归子程序法

- 基本思想：对源程序的每个语法成分编制一个处理子程序，从处理这个语法成分的子程序开始，在分析过程中调用一系列过程或函数，对源程序进行语法语义分析，直到整个源程序处理完毕。

- 子程序
 - 简单子程序
 - 嵌套子程序
 - 递归子程序



子程序执行
机制是什么？

- 子程序执行机制分析：进入子程序时要保存现场；退出子程序时要恢复现场。

递归子程序语法分析方法

- 语法规则预处理：
 - 消除左递归
 - 提取公因子

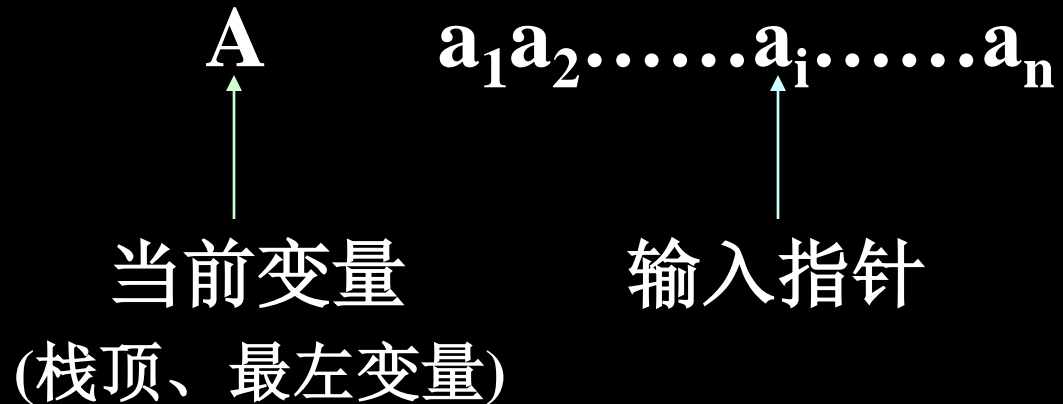
消除左递归

???

- 无法根据左递归文法进行自顶向下的分析

- 直接左递归

– $A \Rightarrow A\alpha$



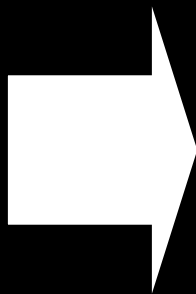
- 间接左递归

– $A \Rightarrow^+ A\alpha$

- 左递归的消除方法

– 将 $A \rightarrow A\alpha | \beta$ 替换为 $A \rightarrow \beta A'$ 和 $A' \rightarrow \alpha A' | \varepsilon$

例：表达式文法直接左递归的消除

$$E \rightarrow E + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow (E) \mid \text{id}$$

$$E \rightarrow T E'$$
$$E' \rightarrow + T E' \mid \varepsilon$$
$$T \rightarrow F T'$$
$$T' \rightarrow * F T' \mid \varepsilon$$
$$F \rightarrow (E) \mid \text{id}$$

例：间接左递归的消除

$S \rightarrow Ac|c$ $A \rightarrow Bb|b$ $B \rightarrow Sa|a$

将B的定义代入A产生式得： $A \rightarrow Sab|ab|b$

将A的定义代入S产生式得： $S \rightarrow Sabc|abc|bc|c$

消除直接左递归： $S \rightarrow abcS'|bcS'|cS'$

$S' \rightarrow abcS'|\epsilon$

删除“多余的”产生式： $A \rightarrow Sab|ab|b$ 和 $B \rightarrow Sa|a$

结果： $S \rightarrow abcS'|bcS'|cS'$

$S' \rightarrow abcS'|\epsilon$

消除左递归的一般方法

- 用产生式组

- $A \rightarrow \beta_1 B | \beta_2 B | \dots | \beta_m B$

- $B \rightarrow \alpha_1 B | \alpha_2 B | \dots | \alpha_n B | \varepsilon$

- 替换产生式组

- $A \rightarrow A\alpha_1 | A\alpha_2 | \dots | A\alpha_n | \beta_1 | \beta_2 | \dots | \beta_m$

- 其中：B为新变量，相当于A'

提取左因子

- 例：if语句的原始文法
 - $S \rightarrow \text{if } E \text{ then } S$
 - | $\text{if } E \text{ then } S \text{ else } S$
 - | other
- 存在左因子 $\text{if } E \text{ then } S$
- 影响分析:遇到 if 时难以判断用哪一个产生式进行匹配（推导）

左因子提取方法

将形如

$$A \rightarrow \alpha\beta_1 | \alpha\beta_2 | \dots | \alpha\beta_n | \gamma_1 | \gamma_2 | \dots | \gamma_m$$

的规则改写为

$$A \rightarrow \alpha A' | \gamma_1 | \gamma_2 | \dots | \gamma_m \text{ 和 } A' \rightarrow \beta_1 | \beta_2 | \dots | \beta_n$$

• 结果:

– $S \rightarrow \text{if } E \text{ then } SS' | \text{other}$

– $S' \rightarrow \varepsilon | \text{else } S$

例：简单算术表达式的分析器

- E的子程序

procedure E;

begin

T;

T 的过程调用

while lookahead='+' do

begin

当前符号等于+时

match('+');

处理终结符+

T

T 的过程调用

end

end;

lookahead: 当前符号

$E \rightarrow T E'$

$E' \rightarrow + T E' | \epsilon$

T的子程序

procedure T;

begin

F;

while lookahead='*' then

begin

match('*');

F

end

end;

$T \rightarrow F T'$

$T' \rightarrow * F T' \mid \varepsilon$

F 的过程调用

当前符号等于 * 时

处理终结符 *

F 的递归调用

F的子程序 ($F \rightarrow (E)id$)

procedure F;

begin

if lookahead='(' then

begin

当前符号等于 (

match('(');

处理终结符 (

E;

E 的递归调用

match(')');

处理终结符)

end

else if lookahead=id then

match(id)

处理终结符id

else error

出错处理

end

主程序

begin

lookhead:=nexttoken;

调词法分析程序

E

E 的过程调用

end

procedure match(t:token);

begin

if lookhead=t then

lookhead:=nexttoken

else error

出错处理程序

end;

递归下降子程序

program \rightarrow function_list

function_list \rightarrow function function_list $\mid \epsilon$

function \rightarrow FUNC identifier (parameter_list)
statement

```
void ParseFunction()  
{  
    MatchToken(T_FUNC);  
    ParseIdentifier();  
    MatchToken(T_LPAREN);  
    ParseParameterList();  
    MatchToken(T_RPAREN);  
    ParseStatement();  
}
```

优缺点分析

- 优点：
 - 1) 直观、简单、可读性好
 - 2) 便于扩充
- 缺点：
 - 1) 递归算法的实现效率低
 - 2) 处理能力相对有限
 - 3) 通用性差，难以自动生成