Побудова сканера для формул електронної таблиці

Сканер – це є програма, яка текст однієї формули ділить на окремі лексеми і кодує лексеми певним способом, наприклад, цілими числами. Отже, замість тексту формули будемо мати масив цілих чисел, що дозволить більш ефективно будувати програми перетворення і обчислення формул. Наприклад, якщо маємо формулу

```
= B5+B6-SUM(C4:C10)/2
```

то поділ на лексеми мав би виглядати так:

=	B5	+	B6	-	SUM	(	C4	:	C10	)	/	2

Щоб програмувати сканер, потрібно укласти повний список всіх допустимих лексем і визначити таблицю кодування лексем. В основі таблиці кодування має бути деякий принцип чи загальний підхід. Наприклад, для кожної лексеми можна надати три специфікатори: перший визначає тип лексеми, а один або два наступні уточнюють зміст лексеми. Отже, для однієї лексеми можна визначити такі структури:

```
struct ETindex // індекс таблиці
{
  int row,col; // індекси рядка і стовпця електронної таблиці
};

struct Ttoken  // визначення лексеми
{
  int token;    // тип лексеми
  // уточнення змісту лексеми
  union // спільна ділянка пам'яті (накладання)
  {
    ETindex index; // індекси рядка і стовпця
    double cellsvalue; // або значення числа в комірці
    char * lit;     // текстовий рядок (літерал)
  };
};
```

Величина *token* є головною і визначає тип лексеми. Величини *row* і *col*, які входять в структуру ETindex, є допоміжними і визначають уточнюючі параметри лексеми, а саме – номер рядка і номер стовпця комірки електронної таблиці, на яку є посилання в даній лексемі. Величина *cellsvalue* визначає безпосередньо число, якщо воно є значенням лексеми. З метою економії пам'яті використовуємо накладання величин (union) *index* і *cellsvalue*, враховуючи ту обставину, що лексема не може бути одночасно адресою комірки і числом. Величина *lit* є вказівником на текстовий рядок, що є значенням лексеми, у випадку, якщо лексема є літералом.

Раніше були визначені такі граматичні правила для чисел і формул (з уточненнями):

Числа:

```
<чс> ::= < число >  |  + < число >  |  - < число >
<число> ::= <ціле> | <фіксована крапка> | <експоненціальний формат>
<ціле>  ::=   цифра  |  <ціле> цифра
<фіксована крапка>  ::=   <ціле> . <ціле>
<експоненціальний формат>  ::=   <мантиса> <порядок>
<мантиса> ::=  <ціле>  |  <фіксована крапка>
<порядок> ::=  E <знак> <ціле>  |  E <ціле>
<знак> ::=   +  |  -
```

Формули:

```
<формула> ::=   = <вираз>
<вираз> ::=  <доданок> | <вираз> + <доданок>   <вираз> - <доданок>
<доданок>::= <множник> | <доданок> * <множник> | <доданок> / <множник>
<множник> ::= <число> | <комірка> | <функція> | ( <вираз> )
<комірка> ::=  <букви> <цифри>
<букви> ::= буква | буква буква
<цифри> ::= цифра | цифра цифра
<функція> ::=  SUM ( <ряд> )  |  MAX ( <ряд> )  |  MIN ( <ряд> )  |
```

```
                ABS ( <комірка> )  |  INT ( <комірка> )
  <ряд> ::=  <комірка> : <комірка>
```

Список лексем, які є допустимими для визначених правил записування формул, можна кодувати так:

лексема	token	row	col	лексема	token	row	col
		cellsvalue / lit				cellsvalue / lit	
=	101			SUM	201		
+	102			MAX	202		
-	103			MIN	203		
*	104			ABS	204		
/	105			INT	205		
(	106			комірка	1	рядок	стовпець
)	107			число	2	значення числа	
:	108			'текст'	301	вказівник на рядок	
				"текст"	302	вказівник на рядок	

Отже, деякі загальні правила. Знаки (однолітерні розділювачі) кодуємо числами, починаючи від 101. Імена функцій кодуємо числами, починаючи від 201.

Всі комірки кодуємо числом 1, а *row* і *col* визначають, відповідно, номер рядка і номер стовпця електроної таблиці для даної комірки. Наприклад, комірка D15 буде зображена як (1,15,4). Щодо нумерації комірок потрібно прийняти певні допущення. Наприклад, технічні характеристики і обмеження Microsoft Excel визначають максимальний розмір листа таблиці як 65536 рядків і 256 стовпців. Для наших цілей можемо прийняти обмеження до 99 рядків і до 256 стовпців. Це дозволить використати не більше двох букв і не більше двох цифр для адресування комірки. Отже, номери стовпців можуть бути в межах від A до IV (букви "I" "V"), а номери рядків – в межах від 1 до 99. Адресу комірки поза такими межами будемо вважати помилкою.

Всі числа кодуємо як 2, а *cellsvalue* визначає безпосередньо власне значення константи. Тобто значення константи зберігається в структурі лексеми як тип double.

Літерали (текстові рядки) кодуємо як 301 або 302, залежно від того, якими лапками (одинарними чи подвійними) обмежений рядок, а поле *lit* в цьому випадку визначає вказівник на рядок. Пам'ять для рядків надає сканер при розпізнаванні тексту рядка.

Решту полів *row* і *col*, які в таблиці не заповнені, не використовуємо, їм можна надати значення нуль.

Якщо поле *token* має значення нуль, значить це є нульова (фіктивна) лексема, таку лексему можна використовувати в ситуаціях, коли виникають помилки.

Формати лексем і алгоритми розпізнавання. Сканер фактично повинний розпізнавати лише такі групи лексем: однолітерні розділювачі (знаки); ідентифікатори (імена функцій і комірок); числа (різних форматів); літерали в одинарних лапках; літерали в подвійних лапках. Все решту вважається помилкою запису формули. Головне правило для розпізнавання лексем є таке: групу, до якої належить поточна лексема, можна визначити за першою літерою лексеми:

```
if (цифра) { група чисел }
else if (буква) { група ідентифікаторів }
else if (допустимий знак) { група однолітерних розділювачів }
else if (одинарний апостроф) { літерал в одинарних лапках }
else if (подвійний апостроф) { літерал в подвійних апострофах }
else { помилка: недопустима літера }
```

Формати всіх чисел можна визначити такою формулою:

```
<число> ::= цифра { цифра } [ . цифра { цифра } ]
            [ E [ + | - ] цифра { цифра } ]
```

Тут і надалі в подібних формулах використовуємо такі позначення. Вертикальна риска означає "або", тобто вибір одного з декількох варіантів. Квадратні дужки [і] означають, що заключена в них частина формули може бути або відсутня, або присутня один раз. Фігурні дужки { і } означають, що заключена в них частина формули може бути або відсутня, або присутня багато разів.

Зауважимо, що ця формула є зведеною для записаних вище граматичних правил визначення чисел.

Формат всіх ідентифікаторів визначаємо формулою:

```
<ідентифікатор> ::=  буква { буква | цифра }
```

Такий формат об'єднує записані вище правила визначення понять "комірка" і "ім'я функції", а заодно є розширенням до будь-якого іншого імені функції чи комірки. За цією формулою визначаємо як імена функцій, а також імена комірок. Відрізнити функцію від комірки потрібно шляхом додаткових перевірок.

Формат однолітерних розділювачів:

```
<однолітерний розділювач> ::=   = | + | - | * | / | ( | ) | :
```

Список всіх допустимих однолітерних розділювачів визначаємо за поданими вище граматичними правилами для формули.

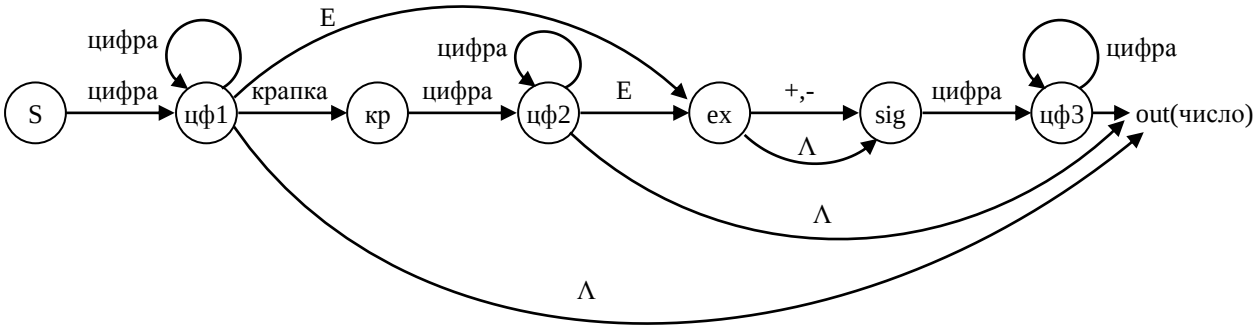
Формат літерала в одинарних лапках:

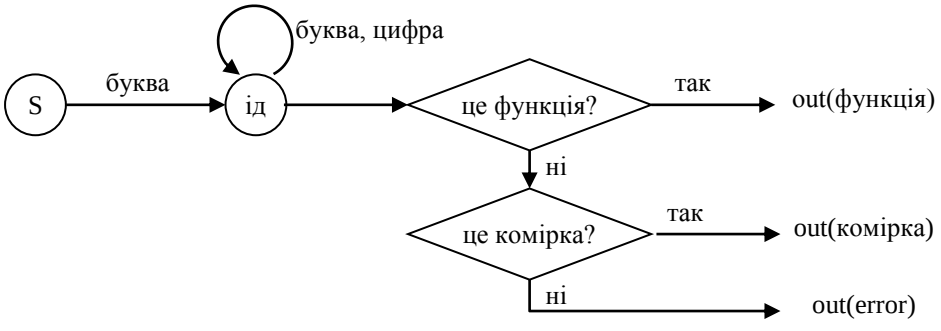
```
< літерал в одинарних лапках > ::=   ' { літера крім ' } '
```

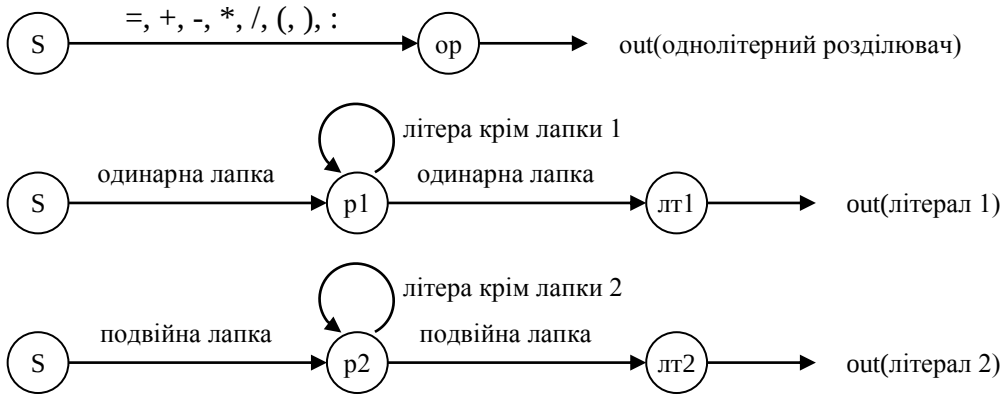
Формат літерала в подвійних лапках:

```
< літерал в подвійних лапках > ::=   " { літера крім " } "
```

Оскільки для розпізнавання лексем потрібно аналізувати по черзі кожну окрему літеру формули, то алгоритми розпізнавання лексем доцільно подати у формі діаграм станів детермінованого скінченного автомата. Нижче подано відповідні діаграми станів. Позначення станів є умовними. Знаком "Λ" позначають пусту літеру, іншими словами, це є перехід без розпізнавання будь-якої літери за умови, що інші переходи неможливі. Out означає вихід з відповідною розпізнаною лексемою.



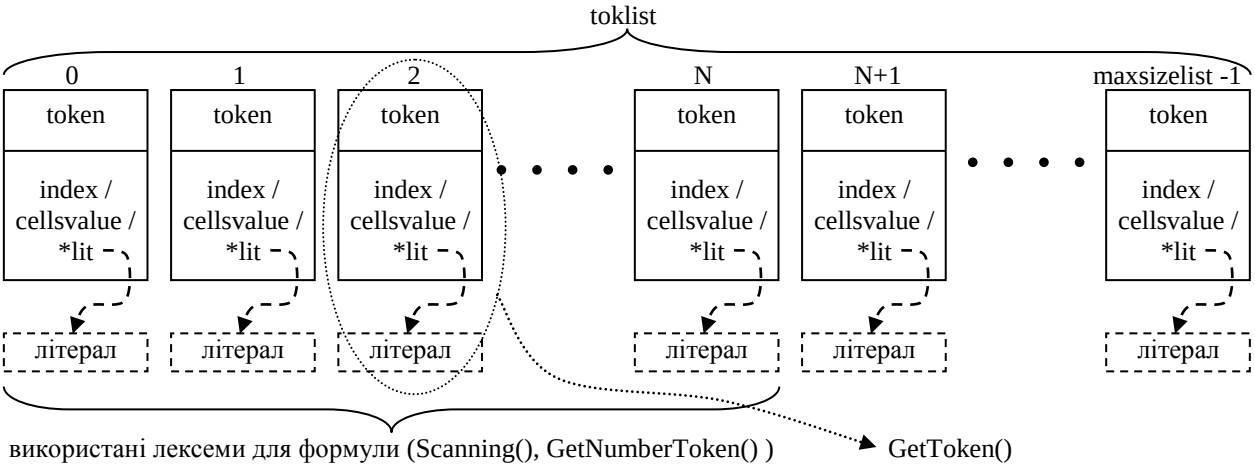




Далі надруковано текст файла, у якому визначений сканер, і файл з головною функцією main() для тестування сканера і перегляду результатів його роботи. Для сканера побудовано клас TScanner. Щоб сканер зміг виконати сканування формули, тобто, кодування лексем, текст формули необхідно спочатку копіювати у внутрішню пам'ять сканера (CopyForm), бо при цьому автоматично реалізується приведення регістра літер до однакового (великі літери). Основний метод класу для сканування – Scanning. Сканер може виявляти помилки, які стосуються окремих лексем, і у випадку помилки повідомляє код помилки і лексему з помилкою. У випадку успішного сканування можна прочитати по черзі всі закодовані лексеми (метод GetToken).

Файл з головною функцією має допоміжні функції для тестування сканера і друкування діагностики сканування.

Сканер зберігає список лексем однієї формули:



Сканер необхідно викликати по черзі для кожної комірки електронної таблиці, в якій записана формула. У випадку успішного сканування формули всі використані лексеми потрібно копіювати в іншу (зовнішню) структуру, прив'язану, наприклад, до відповідної комірки електронної таблиці. Якщо лексема є літералом, то залежно від контексту використання сканера, потрібно проаналізувати, чи додатково копіювати сам літерал, бо пам'ять для літерала надавав сканер.

```
#ifndef DataStructuresForCalcET001
#define DataStructuresForCalcET001
// структури даних для обчислення формул - файл DataStruct.h
//-------------------------------------------------------------
// структури для сканера
struct ETindex // індекс таблиці
{
  int row,col; // індекси рядка і стовпця електронної таблиці
}; // struct ETindex
```

```
struct Ttoken  // визначення лексеми
{
  int token;    // тип лексеми
   // уточнення змісту лексеми
   union // спільна ділянка пам'яті (накладання)
   {
     ETindex index; // індекси рядка і стовпця
      double cellsvalue; // або значення числа в комірці
      char * lit;     // або текстовий рядок (літерал)
   };
}; // struct Ttoken

const Ttoken NulTok = {0,0,0}; // фіктивна лексема

enum { maxsizelist=100 }; // максимально допустимий розмір списку лексем

enum   // список імен і кодів лексем
{ s_equal=101, s_plus=102, s_minus=103, s_mult=104, s_div=105, openbracket=106,
   closebracket=107, colon=108, fsum=201, fmax=202, fmin=203, fmyabs=204,
   fint=205, cell_et=1, number_et=2, lit1=301, lit2=302, end_list=0 };
//-------------------------------------------------------------
#endif
```

```
#ifndef MyScanerForDemoVer1
#define MyScanerForDemoVer1
/*  -------------   Ч.В.В.  Системне програмування  --------------
    Демонстраційний варіант побудови сканера формул електронних таблиць
    ------------------------------------------------------------------- */

// визначення класу сканера для сканування формули - файл CrtTk.h

#include "DataStruct.h"  // файл структур даних
#include <string>  // використаємо стандартний клас string
#include <iostream>
using namespace std;

class TScanner  // клас сканера
{
  protected:
     Ttoken toklist[maxsizelist];  // список лексем однієї формули
    int N;       // номер останньої лексеми в списку (0<=N<100)
    string formula;    // текст заданої формули
     int fln; // довжина формули в літерах
    //basic_string<char> formula;  // можна використати і такий клас
    int ff;        // номер поточної літера формули для аналізу
     char letter;  // поточна літера формули
     bool isinit;   // чи було визначено текст формули
     bool isscanning;  // чи був побудований список лексем
     string tx;  // текст однієї лексеми
     bool error; // ознака наявності помилки при скануванні
     int codeerror;  // номер помилки за класифікацією сканера
     /* визначено помилки з номерами: 1001-1004,1111 */

     // допоміжні функції для сканування
     void Init() // знайти першу значущу літеру, починаючи з поточної позиції
     {
       while(ff<fln && (formula[ff]==' ' || formula[ff]=='\t' ||
             formula[ff]=='\n')) ff++;
       if(ff<fln) letter=formula[ff]; else letter='\0';
       tx = ""; // початковий текст лексеми
       if(letter == '\0') return; // кінець тексту формули
       if(N < maxsizelist-1) N++;  // номер наступної лексеми
       else  // формула завелика (список лексем завеликий)
```

```
      { codeerror=1111; error=true; tx="Formula too long"; }
    } // void Init()

    void AddLetterToToken() // дописати чергову літеру до лексеми
    {
     tx += letter;
    } // void AddLetterToToken()

    void GetNextChar() // перейти до наступної літери формули
    {
      if(ff<fln-1) { ff++; letter=formula[ff]; } else { ff=fln; letter='\0'; }
    } // void GetNextChar()

    int FunctionNameCode(string xfn) // кодування імені функції xfn числом
    {
     if(xfn=="SUM") return fsum;
      else if(xfn=="MAX") return fmax;
      else if(xfn=="MIN") return fmin;
      else if(xfn=="ABS") return fmyabs;
      else if(xfn=="INT") return fint;
      else  return 0; // такої функції немає в списку
    } // int FunctionNameCode(string xfn)

    int GetCodeTok1L(char t) // кодування однолітерних розділювачів
    {
      switch(t)
      {
      case '=': return s_equal;
      case '+': return s_plus;
      case '-': return s_minus;
      case '*': return s_mult;
      case '/': return s_div;
      case '(': return openbracket;
      case ')': return closebracket;
      case ':': return colon;
      default:   return 0; // це не розділювач
      }
    } // int GetCodeTok1L(char t)

    int ConvertA1toR1C1(const string & s)
    { // перетворити буквенний номер s стовпця на числовий
     int num=0;
      string::size_type k;
      for(k=0; k<s.length(); k++) num = num*26 + (s[k]-'A'+1);
      return num;
    } // int ConvertA1toR1C1(const string & s)

  public:
    TScanner() { N= -1; ff=0; fln=0; isinit=false; isscanning=false; }
                // конструктор
    ~TScanner() { cout << "Scanner destroyed\n"; }   // деструктор

    void CopyForm(char * f)
    { // копіювати формулу f в formula
      formula = f;
       fln = formula.length(); // запам'ятати кількість літер формули
                               // без нульової
      // замінити всі малі латинські літери великими (для формул однакові)
      int i=0;
      while(i<fln) // поки не дійшли до кінця формули
      {
        if(formula[i] != '"' && formula[i] != '\'') // це не літерал
          {  formula[i]=toupper(formula[i]); i++; }
       else  // літерали пропускаємо без зміни
```

```
          if(formula[i] == '"') // літерал починається з літери "
            { do { i++; }
              while(i<fln && formula[i] != '"'); // кінець літерала або формули
             if(i<fln) i++;
            }
          else  // formula[i] == '\''
          { do { i++; }
            while(i<fln && formula[i] != '\'');
             if(i<fln) i++;
          }
        } // while(i<fln)
         isinit=true;  // текст формули визначений
         isscanning=false;  // але ще не сканований
         N= -1;  // списку лексем ще немає
         error=false; // ознака наявності помилки - немає
      } // void CopyForm(char * f)

      string ReadForm()  // прочитати приведену формулу (для контролю)
      {
       if(isinit) return formula;
        else  return "Formula not defined";
      } // string ReadForm()

      int Scanning(string & badtoken);  // сканувати формулу
                                        //  і побудувати список лексем

      int GetNumberToken() { return N+1; } // прочитати кількість лексем

      const Ttoken& GetToken(int k) // прочитати лексему номер k
      {
       if(isscanning && k>=0 && k<=N) return toklist[k];
        else return NulTok; // фіктивно
      } // const Ttoken& GetToken(int k)
};  // class TScanner
// ----------------------------------------------------------------------------
// визначення методів класу: основна функція для сканування
int TScanner::Scanning(string & badtoken) // сканування формули
{   // badtoken може означати лексему, де була помилка
  if(!isinit) return 0;  // якщо текст формули не був визначений
   ff = 0; // номер найпершої літери формули
   letter=formula[0]; // найперша літера формули
   N= -1;  // порожній список лексем
   error=false; // ознака наявності помилки

   // цикл сканування і кодування лексем;
   // аналіз виконуємо за поточною літерою letter
   while (letter != '\0' && !error)
   {
     Init(); // знайти першу значущу літеру від поточної позиції
      if ( error || letter == '\0') break; // помилка функції Init()
                                           // або кінець формули
      // за першою літерою розпізнаємо групу, до якої належить лексема
      if(letter>='0' && letter<='9') // це є число
      {
        // цифри перед крапкою
        while(isdigit(letter)) { AddLetterToToken(); GetNextChar(); }
        if(letter == '.') // може бути крапка
        {
          AddLetterToToken(); GetNextChar(); // сама крапка
          // цифри після крапки
          if( ! isdigit(letter)) // після крапки немає цифри
          {
            codeerror=1002; error=true; break; // після крапки немає цифри
          }
```

```
      else // прочитати цифри після крапки
        while(isdigit(letter)) { AddLetterToToken(); GetNextChar(); }
    } // if(letter == '.')
    // далі за текстом може бути записаний порядок числа
    if(letter=='E') // порядок присутній
    {
      AddLetterToToken(); GetNextChar(); // дописати букву E
      if(letter=='+' || letter=='-') // може бути знак порядку
      { AddLetterToToken(); GetNextChar(); } // дописати знак
      // далі мають бути цифри порядку
      if( ! isdigit(letter)) // після букви E немає цифри
      {
        codeerror=1003; error=true; break; // після букви E немає цифри
      }
      // дописати цифри порядку
      while(isdigit(letter)) { AddLetterToToken(); GetNextChar(); }
    } // порядок присутній

    // перетворити число з текстової форми в double і записати лексему
    double x = atof(tx.c_str());
    toklist[N].token=number_et;  toklist[N].cellsvalue=x;
  } // це є число
  else
    if(letter>='A' && letter<='Z') // це комірка або функція
    {
      // спочатку прочитати всі букви
      while(letter>='A' && letter<='Z') {AddLetterToToken(); GetNextChar();}
      // далі можуть бути цифри і букви в довільному порядку
      while(isdigit(letter) || letter>='A' && letter<='Z')
        { AddLetterToToken(); GetNextChar(); }
      // перевіримо, чи це є іменем функції
      int fnc = FunctionNameCode(tx);
      if(fnc) { toklist[N].token=fnc; } // знайшли ім'я функції
      else // перевіримо, чи це правильне ім'я комірки
      {
        // розділити лексему на букви і цифри
        int i=0;
        string tempcol="";
        for(; i<tx.length() && tx[i]>='A'&&tx[i]<='Z'; i++) tempcol+= tx[i];
        string temprow="";
        for(; i<tx.length() && isdigit(tx[i]); i++)  temprow+= tx[i];
        int numrow=atoi(temprow.c_str());
        int numcol=ConvertA1toR1C1(tempcol);
        if(numrow>=1 && numrow<=99 && numcol>=1 && numcol<=256
                     && i>=tx.length())
        { // правильна лексема
          toklist[N].token=cell_et;
          toklist[N].index.col=numcol; toklist[N].index.row=numrow;
        }
        else // помилка
        {
          codeerror=1004; error=true; break;
                    // неправильне ім'я комірки або функції
        }
      } // перевірка правильності імені комірки
    } // це комірка або функція
    else
      if(GetCodeTok1L(letter)>0) // це однолітерний розділювач
      {
        toklist[N].token=GetCodeTok1L(letter);
        GetNextChar(); // до наступної літери формули
      } // це однолітерний розділювач
      else
        if(letter=='\'') // це літерал з одинарними апострофами
```

```
            { // читати до кінця літерала
              GetNextChar(); // наступна літера після апострофа
            while(letter != '\'' && letter != '\0')
                 { AddLetterToToken(); GetNextChar(); }
              toklist[N].token=lit1;
              toklist[N].lit= new char[tx.length()+1]; // надати пам'ять

              strcpy(toklist[N].lit,tx.c_str()); // копіювати літерал
              GetNextChar(); // перейти до наступної літери за апострофом
            } // це літерал з одинарними апострофами
            else
              if(letter=='\"') // літерал з подвійними апострофами
              { // читати до кінця літерала
                GetNextChar(); // наступна літера після апострофа
              while(letter != '\"' && letter != '\0')
                   { AddLetterToToken(); GetNextChar(); }
                toklist[N].token=lit2;
                toklist[N].lit= new char[tx.length()+1]; // надати пам'ять

                strcpy(toklist[N].lit,tx.c_str()); // копіювати літерал
                GetNextChar(); // перейти до наступної літери за апострофом
              } // літерал з подвійними апострофами
              else // помилка - невизначена лексема
              {
                AddLetterToToken(); // запам'ятати недопустиму літеру в лексемі
                codeerror=1001; // невизначена лексема (недопустима літера)
                error=true;
              } // помилка - невизначена лексема
  } // while (letter != '\0' && !error)

  isscanning=true;  // список лексем побудований
  if(!error) return 0; // помилок не було
  else
  { // у випадку помилки копіюється лексема з помилкою
    //  і повертається номер помилки
     badtoken = tx; // лексема з виявленою помилкою
    return codeerror; // номер знайденої помилки
  }
} // int TScanner::Scanning(string & badtoken)

#endif
```

```
/*  -------------   Ч.В.В.  Системне програмування  --------------
                      Приклад застосування сканера
    ------------------------------------------------------------------ */

#include "CrtTk.h"

using namespace std;

// прототипи допоміжних функцій для тестування
string ConvertR1C1toA1(const Ttoken & tok);
void Print1Token(const Ttoken & tok);
void PrintTokenList();
void DiagnosticScanning(int err, string & tok);

TScanner Scanner; // екземпляр сканера

// головна функція
int main()
{
  string errtok; // на випадок помилок у лексемах
  char ex1[] = "=b23 +  A3 - Sum(C2:c15) / 2.6e1"; // приклад формули
  cout << "Formula orig:  " << ex1 << endl; // для контролю
```

```
   Scanner.CopyForm(ex1);  // спочатку формулу копіювати
   cout << "Formula toupper:  " << Scanner.ReadForm() << endl;
            // перевірка приведення
   int retcode = Scanner.Scanning(errtok); // сканувати формулу
   DiagnosticScanning(retcode,errtok); // надрукувати діагностику

   if(!retcode)  PrintTokenList();  // надрукувати для контролю список лексем

   // затримати вікно консолі
  cin.ignore(80,'\n');
  return 0;
} // main()

string ConvertR1C1toA1(const Ttoken & tok) // для тестування
{ // обернене перетворення номера стовпця комірки (col) у формат A1
   string conv;  string c1,c2;
   if(tok.index.col<=26) conv = (char)('A'-1+tok.index.col);
   else // будуть дві букви
   {
     c1 = (char)(tok.index.col / 26 - 1 + 'A');
      c2 = (char)(tok.index.col % 26 - 1 + 'A');
      conv = c1+c2;
   }
  return conv;
} // string ConvertR1C1toA1(const Ttoken & tok)

void Print1Token(const Ttoken & tok)
{ // декодувати і друкувати одну лексему tok
   switch(tok.token) // дешифруємо
      {
      case cell_et:      cout << ConvertR1C1toA1(tok) << tok.index.row; break;
      case number_et:    cout << tok.cellsvalue; break;
      case s_equal:      cout << '='; break;
      case s_plus:       cout << '+'; break;
      case s_minus:      cout << '-'; break;
      case s_mult:       cout << '*'; break;
      case s_div:        cout << '/'; break;
      case openbracket:  cout << '('; break;
      case closebracket: cout << ')'; break;
      case colon:        cout << ':'; break;
      case fsum:         cout << "SUM"; break;
      case fmax:         cout << "MAX"; break;
      case fmin:         cout << "MIN"; break;
      case fmyabs:       cout << "ABS"; break;
      case fint:         cout << "INT"; break;
      case lit1:         cout << '\'' << tok.lit << '\''; break;
      case lit2:         cout << '\"' << tok.lit << '\"'; break;
      } // switch(tok.token)
   cout << ' '; // лексеми розділяємо пропуском
} // void Print1Token(const Ttoken & tok)

void PrintTokenList()
{ // декодування лексем і друкування дешифрованого списку лексем
   cout << "toklist: count tokens " << Scanner.GetNumberToken() << endl;
   int k;
   Ttoken leks;
   for(k=0; k<Scanner.GetNumberToken(); k++)
   {
      leks=Scanner.GetToken(k); // читаємо чергову лексему
      Print1Token(leks); // і друкуємо
   } // for
   cout << endl;
} // void PrintTokenList()
```

```
void DiagnosticScanning(int err, string & tok) // діагностика сканування
{ // err - код помилки, tok - лексема з помилкою
   if(!err) cout << "Scanning succeeded\n"; // без помилок
   else // друкувати повідомлення про помилку
   {
     cout << "Error S" << err << ": ";
     switch(err)
     {
     case 1001: cout << "letter \'" << tok << "\' not allow"; break;
     case 1002:; case 1003: cout << "wrong number \'" << tok << "\'"; break;
     case 1004: cout << "wrong name \'" << tok << "\' of function or cell";
break;
     case 1111: cout << "Formula too long"; break;
     default:  cout << "indefinite error"; break;
     } // switch(err)
     cout << endl;
   }
} // void DiagnosticScanning(int err, int pos)
```