Chapter – 27
Putting it all together

Requirements

- The program must be long enough to demonstrate modular programming
- Short enough the fit into a chapter
- Complex enough to demonstrate advanced C++ features
- Simple enough for a student to understand
- It must be useful.

The program selected is designed to read C++ files and generate simple statistics.

Specification

Preliminary Specification for a C++ Statistics Gathering Program

Steve Oualline
February 10, 1995

The program stat gathers statistics about C++ source files and prints them. The command line is:

```
stat <files..>
```

Where <files..> is a list of source files. The following shows the output of the program on a short test file.

```
1 (0  {0  #include <iostream>
2 (0  {0  /*****************************************
3 (0  {0   * calc -- a simple 4 function calculator      *
4 (0  {0   *****************************************/
5 (0  {0  int   result;  // the result of the calc.
6 (0  {0  char  oper_char;  // operator the user specified
7 (0  {0  int   value; // value specified after the op.
```


Specification

```
   8 (0  {0  int main()
   9 (0  {1  {
  10 (0  {1      result = 0;          // initialize the result
  11 (0  {1
  12 (0  {1      // loop forever (or until break reached)
  13 (0  {2      while (1) {
..............
  44 (0  {2          }
  45 (0  {1      }
  46 (0  {1      return (0);
  47 (0  {0  }
Total number of lines: 47
Maximum nesting of () : 2
Maximum nesting of {} : 4
Number of blank lines ..................4
Number of comment only lines ..........4
Number of code only lines .............35
Number of lines with code and comments 4
Comment to code ratio 20.5%
```

Code Design

Token Module

Turns input into tokens (a series of “words”)

Example:

```
answer = (123 + 456) / 89;  // Compute something
```

becomes:

`T_ID`	The word "answer"
`T_OPERATOR`	The character "="
`T_L_PAREN`	Left Parenthesis
`T_NUMBER`	The number 123
`T_OPERATOR`	The character "+"
`T_NUMBER`	The number 456
`T_R_PAREN`	The right parenthesis
`T_OPERATOR`	The Divide operator
`T_NUMBER`	The number 89
`T_OPERATOR`	The semicolon
`T_COMMENT`	The // comment
`T_NEW_LINE`	The end of line character

Other Modules

Character type module

Determines the type of a character (letter, digit, etc.)

Statistics class

Consumes tokens and outputs statistics.

Functional Description

`char_type` class.

- Basically a big table indexed by character type.
- Some extra code thrown in for specials like `C_ALPHA_NUMRIC`.

`input_file`

- An `ifstream` with line buffering that copies each line to the output.

`token` class

- Reads characters, outputs tokens.
- There is one trick in the coding, the use of the `TOKEN_LIST` macro.

TOKEN_LIST

```
#define TOKEN_LIST \
   T(T_NUMBER),   /* Simple number (float or int) */ \
   T(T_STRING),   /* String or character constant */ \
   T(T_COMMENT),  /* Comment */                      \
   T(T_NEWLINE),  /* Newline character */            \
   T(T_OPERATOR), /* Arithmetic operator */          \
   T(T_L_PAREN),  /* Character "(" */                \
   T(T_R_PAREN),  /* Character ")" */                \
   T(T_L_CURLY),  /* Character "{" */                \
   T(T_R_CURLY),  /* Character "}" */                \
   T(T_ID),    /* Identifier */                      \
   T(T_EOF)    /* End of File */
```

Functional description (cont.)

`stat` class

```
class stat {
    public:
        virtual void take_token(TOKEN_TYPE token) = 0;
        virtual void line_start(void) {};
        virtual void eof(void) {};
};
```

`line_counter` class

Counts the number of `T_NEW_LINE` tokens.

brace_counter class

```
// Consume tokens,  count the nesting of {}
void brace_counter::take_token(TOKEN_TYPE token) {
    switch (token) {
        case T_L_CURLY:
            ++cur_level;
            if (cur_level > max_level)
                max_level = cur_level;
            break;
        case T_R_CURLY:
            --cur_level;
            break;
        default:
            // Ignore
            break;
    }
}
```

brace_counter class (cont.)

```
// Output start of line statistics
// namely the current line number
void brace_counter::line_start(void) {
   std::cout.setf(ios::left);
   std::cout.width(2);


   std::cout << '{' <<  cur_level << ' ';


   std::cout.unsetf(ios::left);
   std::cout.width();
}


// Output eof statistics
// namely the total number of lines
void brace_counter::eof(void) {
   std::cout << "Maximum nesting of {} : " <<
            max_level << '\n';
}
```

Functional Description

`paren_counter` *class*

Almost the same as brace counter.

`comment_counter` *class*

Keeps track of lines with comments, lines of code, lines with both comment and code and blank lines.

`do_file` procedure

Reads tokens and stuffs them into the statistics classes.

Uses the statistics list for stuffing:

```
static line_counter line_count;         // Counter of lines
static paren_counter paren_count;       // Counter of () levels
static brace_counter brace_count;       // Counter of {} levels
static comment_counter comment_count;  // Counter of cmt info


// A list of the statistics we are collecting
static stat *stat_list[] = {
    &line_count,
    &paren_count,
    &brace_count,
    &comment_count,
    NULL
};
```

Test file

```
/*************************************************
 * This is a mult-line comment                   *
 *      T_COMMENT, T_NEWLINE                     *
 *************************************************/
const int LINE_MAX = 500;       // T_ID, T_OPERATOR, T_NUMBER


// T_L_PAREN, T_R_PAREN
static void do_file( const char *const name)
{
    // T_L_CURLY
    char *name = "Test"         // T_STRING


    // T_R_CURLY
}
// T_EOF
```

The Program

A tour of the source