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.

Specification

. . . .

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

```
    public:



};
```

`line_counter` class

Counts the number of `T_NEW_LINE` tokens.

brace_counter class

```
                    ++cur_level;


                    break;

                    --cur_level;
                    break;
                default:

                    break;
            }
        }
```

[illegible]

brace_counter class (cont.)

```
        std::cout.setf(ios::left);
        std::cout.width(2);



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






    }
```

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:

```
    &line_count,
    &paren_count,
    &brace_count,
    &comment_count,
    NULL
};
```

Test file

```
/*********************************************************
 *********************************************************/


{


}
```

The Program

A tour of the source