**Coding Standard**

Language: **C++**

Author: **Dan Turk**

Date: **1997 Mar 24**

Adapted from:

<table>
<thead>
<tr>
<th>Purpose</th>
<th>To guide the development of C++ programs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Program Headers</strong></td>
<td>• Begin all programs with a descriptive header.</td>
</tr>
<tr>
<td><strong>Program Header Format</strong></td>
<td>/**********************************************************************************</td>
</tr>
<tr>
<td></td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>* Program Name: <em>the program name</em></td>
</tr>
<tr>
<td></td>
<td>* Version: <em>version number</em></td>
</tr>
<tr>
<td></td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>* Author: <em>the author’s name</em></td>
</tr>
<tr>
<td></td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>* Date written: <em>date</em></td>
</tr>
<tr>
<td></td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>* Description: <em>sentence / paragraph description of what the program does</em></td>
</tr>
<tr>
<td></td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>* Platform(s) tested on: <em>hardware / OS</em></td>
</tr>
<tr>
<td></td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>* Modification history:</td>
</tr>
<tr>
<td></td>
<td>* list of dates, authors, and changes made</td>
</tr>
<tr>
<td></td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>**********************************************************************************/</td>
</tr>
</tbody>
</table>
Program Header Example

```c
/*********************************************************
*
* Program Name: date.cpp
* Version: 1.0
* 
* Author: Dan Turk
* 
* Date written: 1997 Mar 10
* 
* Description: date.cpp performs data calculations such
*   as determining the how many days old a person is,
*   given the current date and their birthday, determining
*   the day a person was born given the current date,
*   day of week, and their birthday, determining if a
*   given year is a leap year or not, determining how
*   many days there are in a given month, etc.
* 
* Platform(s) tested on: Pentium / Windows 95
* 
* Modification history:
*   1997 Mar 21     Dan Turk     Added “days old” calculations
*   1997 Mar 15     Dan Turk     Corrected leap year determination
*
*********************************************************/
```

Function Headers

• Begin each function with a descriptive header.

<table>
<thead>
<tr>
<th>Function Header Format</th>
</tr>
</thead>
</table>
| /*
* Function Name: the function name
* Version: version number
* 
* Author: the author's name
* 
* Date written: date
* 
* Description: sentence / paragraph description of what the
*   function does
* 
* Parameter descriptions:
*   name and description of each parameter
* 
* Modification history:
*   list of dates, authors, and changes made
*
```c
(userInfo)
```
### Function Header Example

```c
/*********************************************************
* Function Name: days_in_month()
* Version: 1.0
* 
* Author: Dan Turk
* 
* Date written: 1990 Feb 13
* 
* Description: days_in_month() determines the number of days in month m.
* It returns the days in the month if m is a valid month number (1-12),
* or -1 if m is invalid.
* 
* Parameter descriptions:
* INPUT:
* m month (1-12) for which to determine number of days
* OUTPUT:
* none
* RETURN:
* 28, 29, 30, or 31 depending on the valid month number
* -1 if m is not valid
* 
* Modification history:
* 
**********************************************************/
```

### White Space
- Write programs with sufficient spacing so that they do not appear crowded.
- Separate every program construct with at least one space.

### Blank Lines
- Use blank lines to separate logical blocks of code and to improve readability.
- Put at least one blank line between the end of one function and the beginning of the next.

### Indentation
- Indent every level of logic from the previous one.
- Indent a minimum of 2 and a maximum of 8 spaces for each additional level.
- Start all lines at the same logical level at the same indentation level.

### Line Spacing
- Single-space all lines, except when double-spacing (inserting blank lines) will clarify sections of code, such as setting off logical blocks of code from one another.

### Begin-End block delimiters
- Put begin-block braces on the same line as the beginning statement.
- Put end-block braces on a separate line, indented to the same level as all code within the block.
### Examples of effective use of White Space, Blank Lines, Indentation, and Line Spacing

- **Good Example:**
  ```c
  void main (void) {
      int i, n;
      for (i=0; i<n; i++) {
          cout << i;
          cout << " Hello, world!\n";
      } // for
  } // main()
  ```

- **Bad Example:**
  ```c
  void main(void){
  int i,n;
  for(i=0;i<n;i++)
  {cout<i;
  cout<"Hello, world!\n";
  }
  }
  ```

### Grouping
- Group logical types of code together (Ex: #includes for header files, #defines, prototypes)

### Prototypes
- Declare all prototypes at the beginning of the program before main()

### Includes
- Include all header files at the beginning of the program before main()

### Defines
- Define all constants & macros at the beginning of the program before main()

### Naming Conventions
- Use meaningful names for all variables, constants, and functions.
- Use lower-case names for variables, and upper-case for constants.
- Separate portions of long names with underscores.

  - **Examples of good naming conventions:**
    ```c
    int total_cost, color;
    #define TAX_RATE 0.06
    void calculate_taxes (float gross_income, int exemptions);
    ```

  - **Examples of bad naming conventions:**
    ```c
    int tc, c;
    #define R 0.06
    void calc (float gi, int ex);
    ```
Comments

• Document the code as necessary so the reader can easily understand it.
• Make sure comments say more than what the code already says.
• Do not comment every line of code.
• Comment the beginning of logical blocks of code.
• Clarify end-blocks by commenting them.

Good Examples:

// read until EOF and count number of input items
total = 0;
while (cin << i) {
    n++;
    total += i;
} // while
avg = total / n;

// print results
cout << “total=” << total << “\n”;
cout << “n=” << n << “\n”;
cout << “avg=” << avg << “\n”;

Bad Examples:

total = 0; // set total to zero
while (cin << i) // read i
{
    n++; // add 1 to n
    total += i; // add i to total
} // while
avg = total / n; // calculate average
cout << “total=” << total << “\n”; // print total
cout << “n=” << n << “\n”; // print n
cout << “avg=” << avg << “\n”; // print average