**Intro to Programming: C++ Coding Standards**

Coding standards, or coding conventions, are sets of rules or guidelines designed to govern the process of code production in a software project. They're usually based on industry best practices or generally accepted conventions. They include naming conventions, style, prohibited features, and more. In order to develop reliable and maintainable applications, you must follow coding standards and best practices.

1. **Use Meaningful, descriptive words to name variables.**
2. Camel Casing for Variables – capitalize the first character of all words, except the first word, and other characters are all lowercase

Example: itemsOrdered

1. The underscore character can be used in a variable name

Example: items\_ordered

**Note:**

* Do not use abbreviations. Do not use single character variable names like l, w, h etc. Use names like length, width, height.

|  |  |
| --- | --- |
| **Good Example** | **Not Good Example** |
| int length | int l |
| string address | string addr |

Exception:

When using i for index in loops

for ( int i = 0; i < 7; i++)

{

….

}

* Do not use variable names that resemble the C++ key words.

**The C++ Key Words**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| bool | break | case | catch | char | const |
| continue | default | do | double | else | false |
| float | for | if | int | long | namespace |
| new | or | private | protected | public | return |
| short | static | switch | throw | try | void |
| while | sizeof | … | | | |

1. **Indentation and spacing**
2. Indentation:

Using Tab not Space for indentation. Default tab space is 4.

Indentation for comments should be at the same level.

|  |  |
| --- | --- |
| **Good Example** | //Display welcome message to student  string name = “John”  cout << “Welcome” << name << “to CMSC 140 class.” << endl; |
| **Not Good Example** | //Display welcome message to student  string name = “John”  cout << “Welcome” << name << “to CMSC 140 class.” <<endl; |

All the lines inside a block should be indented.

Example:

if ( score > 90 )

{

grade = 'A';

cout << "Good Job!\n";

}

1. Spacing:

A single space should be used to separate operators, brackets or tokens.

|  |  |
| --- | --- |
| **Good Example** | **Not Good Example** |
| if ( showResult == true )  {  for ( int i = 0; i < 10; i++ )  {  //  }  } | if(showResult==true)  {  for ( int i= 0; i<10; i++ )  {  //  }  } |
| float distM = 0.0; //distance | float distM=0.0;//distance |

1. **Curly braces {}**

The curly braces should be at the same level as the code outside braces, which means the opening curly brace { must always be aligned with the closing curly brace }.

Example:

if ( score > 90 )

{

grade = 'A';

cout << "Good Job!\n";

}

1. **Comments**

Do not write comments for every line of code but write wherever required.

Comments are very important because they can help readers to better understand the programmer’s codes and logic. Be sure to add comments when it’s necessary.

If a variable is initialized with a specific value, the comment states why that value is chosen.

Good variable names and readable codes require fewer comments.

1. Single line comments starts with // and lasts to the end of the current line

Example:

// sample C++ program

// This program is to display a greeting

#include <iostream>

using namespace std;

int main ( )

{

cout << "Hello, there!";

return 0;

}

1. Multiline comments / Block comments start with /\* and end with \*/

Example:

/\* this is a multi-line

sample C++ program

This program is to display a greeting

\*/

**Notes:**

* Function header comments: A function must come along with a header comment. The function header describes the purpose of the function, pre-conditions (what conditions does the function need to meet with so that it can run without errors) and the post-conditions (results) of the function.

Example:

You can write the function header comments as below:

/\*\*

\* Function: qualityPoints

\* Description: To calculate a student’s quality points for a course

\* Pre: The grade of the course. The grades will be A, B, C, D or E on a 4.0

grading scale

\* Post: The function will return the quality points

\*/

**Or like this:**

/\*\*

\* Function: qualityPoints

\* Parameter:

\* In: char grade – the grade for the course (A, B, C, D, or E)

\* int creditHour – how many credits the course is worth

\* Out: None

\* In/Out: None

\* Returns: The quality points for the course

\* Description: To calculate a student’s quality points for a course

\*/

1. **Constant Variables (Named Constants)**

Constant Variables are declared and initializedonly if you know their values and they will never need to be changed.

Declare them in the beginning of the program in the top.

They are used to represent constant values with descriptive names.

Usually, constant variables are written in all uppercase, and the words are separated by the underscore \_.

Example:

const double TAX\_RATE = 0.0675;

const int NUM\_STATES = 50;

1. **Statements**

A complete statement is completed with a semicolon.

Null statements – the statements are invalid and have no effects.

Example:

if (x > y); // Error!

cout << "x is greater than y\n"; // This will be always executed because the

// conditional expression is invalid

1. **Function**
2. The Function definition – to define or create a function, including return type, function name, parameter list and function body.

The function header declares the function’s return type, name, and parameter list.

Example:

**Return type Function name Parameter list ()**

void displayMessage () //**Function Header**

{

**Function body**

cout << "Hello from the function displayMessage.\n";

}

1. The Function prototype – it is also called function declarations. This must be written ahead of all the function calls. With it, the function definition can be written after all calls to the function.
2. The Function call – a statement to request the function to work

**Note:**

* Do not mess up with the function header, function call and function prototype.

Example:

// This program has three functions: main, first, and second.

#include <iostream>

using namespace std;

**// Function Prototypes**

void first();

void second();

int main()

{

cout << "I am starting in function main.\n";

first(); // **Call function first**

second(); // **Call function second**

cout << "Back in function main again.\n";

return 0;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Definition of function first. \*

// This function displays a message. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void first()

{

**Function definition**

cout << "I am now inside the function first.\n";

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Definition of function second. \*

// This function displays a message. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void second()

{

cout << "I am now inside the function second.\n";

}

1. **Assignment and equality operator**

Assignment operator: =

Equality operator: ==

|  |  |
| --- | --- |
| **Good Example** | **Not Good Example** |
| if (x == 3) // Determine if x is equal to 3  cout << " True!"; | if (x = 3) // Assigns x with 3  cout << "True!"; |

References

Gaddis, T. (2018). *C++ From Control Structures Through Objects* (9th ed.). Pearson.