Moliendas y Alimentos

Francisco Gonzalez Gonzalez

Coding Standard Template

Purpose	To guide implementation of C++ programs
Program Headers	All programs must have a header at the beginning of the document. In case of having been modified, include the modification
Header Format	/* Program : Name of the program Objective : Description of the program's use Author : Name of the programmer Date : Date it was created Module : To what module does it belong to */ /* Modification Date : Date of modification Author : Name of the programmer Purpose : Purpose of the change Change : Changes made */
Listing Contents	Summary of the contents of the program
Contents Example	/* Start - Entry message Reuse instructions Modification instructions Compilation instructions */

Reuse Instructions	• Describe how the program is used. Provide the declaration format, parameter values and types, and parameter limits. • Provide warnings of illegal values, overflow conditions, or other conditions that could potentially result in improper operation • Indicate whether the function is re-entrant (modifies global variables, writes files, etc.) • Define valid ranges or limits for all parameters • Mention any external dependencies, such as libraries or other modules.
Reuse Instructions Example	/* Reuse instructions int calculateAverage(int *array, int size) Purpose: To calculate and return the average of the values in “array” Limitations: “size” must be greater than 0 and array must not be NULL. Indexing starts at 0. Dynamic of the array is not handled. Only works with static arrays with known size at runtime. Returns the calculated size of the average as a float value Returns -1.0 if invalid parameters are provided */
Identifiers	Use descriptive variable names following a “verbNoun” identification titling. Avoiding abbreviations. Avoid particles

	Arrays and Vectors must be plural. The first letter of a variable and function must be uncapitalized. A variable that contains multiple words must have every first letter capitalized for every word after the first one. Any additional classes must have its first letter capitalized and be plural. Every variable (if possible) must be declared at the beginning of the program. An example of when it is not possible is in recursions and back-tracking
Good Identifier Examples	`int calculateReceipt;` `vector<int> receiptsMonthJanuary(5);` `int tools[5];` `void function(){` `}` `class Tools{` `}`
Bad Identifier Examples	`int calculatetheReceipt;` `int calculateRcpt;` `vector<int> Receipts_of_jan(5);` `int tool[5];` `void Function(){` `}` `class tools{` `}`

	class Tool{ }
Comments	Every comment must explain the purpose of the variable / operation / function it is related to. If the comment is related to a variable or function use it must be to its right. Otherwise it must be above it. Comment every variable declaration. Comment every major operation (for, while, if, etc), but mustn't use description. Comment every function. Comments that are long or have multiple sections / parts must utilize "/* … */" rather than "//"
Good Comment	int calculateReceipt(); //store number values for later processing in receipt average int calculateReceipt; //Variable to save the calculated value of Receipt /* It iterates through every element of the vector receipts and sums their value to obtain the total value of costReceipt*/ for (int i = 0; i < n; i++) { // Content; } // Main operation of the program. Does not need any parameters int main() {

	`  // Content;` `}` `costReceipt = calculateCostReceipt(int n); // Calculates the cost ot the receipt` `/*` `  Comment content 1` `  Comment content 2` `  Comment content 3` `  Comment content 4` `*/`
Bad Comment	`/* Variable to save the calculated value of Receipt*/` `int calculateReceipt;` `/* i goes from 0 to n to get costReceipt*/` `for (int i = 0; i < n; i++) {` `  // Content;` `}` `int main() {` `  // Content;` `} // Main operation of the program. Does not need any parameters` `//Calculates the cost of the receipt` `costReceipt = calculateCostReceipt(int n);` `//Comment content` `//Comment content` `//Comment content` `//Comment content`
Major Sections	`Divide and categorize code processing into sections for further visual understandment`

Examples	/* The following sections processes all number values and reutilizes them for calculating receipt values. (average, min, max) */
Indenting	Use proper indentations inside function parameters, line spacing, and function structuring. Including but not limited to: - Indent each brace level from the preceding level. - Open and closed braces should be on lines by themselves (not together) and aligned If the content within exceeds more than one line of operation, it must include an additional blank line.
Indenting Good Example	CalculateTax() { if (tax == True) { variable++; } else { return 0; } } if (variable > x) break; if (variable > x) break; if (variable > x){ break; }
Indenting Bad	CalculateTax() {

Example	`  if (tax == True) {` `    variable++;` `      }` `  else {` `    return 0;}` `}`
Capitalization	According to data type. (Variable, Array, Function, Class). Comments will follow standard capitalization grammar rules.
Blank Spaces	Every operator (other than ++ and --) must have a space before and after.
Blank Spaces Good Example	`totalSalary++;` `totalSalary = salary + bonus;`
Blank Spaces Bad Example	`totalSalary ++;` `totalSalary=salary+bonus;`
Control Structures	If any additional Control Structure that was not seen in class is utilized, briefly define its purpose, use and must be justified at the beginning of the program. It then must have a comment indicating its use before its utilization.
Control Structures Example	`/*` `Recursive function explanation:` `This program uses a recursive function to determine the total number of students given by a user.` `- Using std::cin to receive a total amount of students` `- Recursion was used to simplify the`

	logic by breaking the problem into smaller, repeatable steps. It provides a clean and efficient approach for handling structured or repetitive tasks programmatically. */ #include <iostream> int countStudents(int n) { if (n == 0) return 0; return 1 + countStudents(n - 1); } int main() { std::cin >> students int total = countStudents(5); std::cout << "There are " << total << " students in the list." << std::endl; return 0; }
Data Structures	If any additional Data Structure that has not been seen throughout the course, briefly define its purpose and use at the beginning of the program and before its utilization.
Data Structures Example	/* HashMap explanation: This program uses a HashMap to store and retrieve students grades using students IDs as keys HashMap allow constant-time access to data - Using std::map to store grades by student names

	- Preferred over arrays or vectors for faster key-based access and automatic key organization */ #include <iostream> #include <map> int main(){ // Declare and initialize the map std::map<std::string, int> grades; grades["Alice"] = 95; grades["Bob"] = 87; std::cout << "Alice's grade: " << grades["Alice"] << std::endl; return 0; }
Functions	When a certain type of operation or procedure is repeated multiple times or with the purpose of making a complicated function easier to follow, it must be created into a function. A function, similarly to a module, must have one unique purpose, but rather than being oriented towards a general view of the system, specifically oriented to the use of the program / module in question.
Functions Good Example	int numberCoins(int counter, int cuantity){

	`` for (int i = 0; i < cuantity; i++){ counter++; } return counter; } int main(){ int counter1, counter2, counter3, counter4; counter1 = numberCoins(0, 10) counter2 = numberCoins(0, 11) counter3 = numberCoins(0, 3) counter4 = numberCoins(0, 15) return 0; } ```
Functions Bad Example	``` int main() { int counter1, counter2, counter3, counter4; for (int i = 0; i < 10; i++){ counter1++; } for (int i = 0; i < 11; i++){ counter2++; } for (int i = 0; i < 3; i++){ counter3++; } for (int i = 0; i < 15; i++){ counter4++; } ```

	return 0; }