Programming Style Guide

Introduction to Engineering Computing

The purpose of this style guide is to outline best practices that will help make your programs more readable and easier to understand and in turn, make you a better programmer. The programs you submit to be graded should be easy to read and follow your logic. Throughout your readings, your textbook will offer many tidbits of information that will be very useful to you. However, some times the book may say things that we will not follow. When in doubt follow the guidelines in this document; if you do not, you will likely lose easy points on your homework. You are responsible for everything below, but don't worry if some things do not make sense yet, they will eventually.

For this semester, you should adhere to the following in every program you write unless stated otherwise:

1. Do not use `goto`, `break`, or `continue` statements. If you do not know what these statements are, no worries! If and when you do learn what they are, still do not use them; they are bad programming practice. Note: The only exception to this rule is that `break` statements may be used within `switch` statements.
2. Use meaningful names for variables and functions. If the meaningful name is two words, i.e. *account number*, use the variable name `account_number` or `accountNumber`. Avoid single letter variable names (e, p, s, etc.) and uninformative variable names (num1, num2, num3, etc.) at all costs. The only exception is for-loop counter variables.
3. Do not use global variables. Only declare variables within functions. Note: `#define` values are not global variables.
4. Use blank lines where they make the code easier to read.
5. Always indent the block of code within if, else, while, for, and other control statements. For example:

```
if(number == 1)
{
    number = 3 * number;
    cout << "number = " << number << endl;
}
else
{
    number = 2;
    cout << "number = " << number << endl;
}

cout << "Complete." << endl;
```

```
Alternatively,
if(number == 1){
    number = 3 * number;
    cout << "number = " << number << endl;
}else{
    number = 2;
    cout << "number = " << number << endl;
}

cout << "Complete." << endl;
```

Take note of how any time there is an opening curly bracket, '{', the following lines of code are indented farther (either 3 or 4 spaces, stay consistent!) than the line of code before it. Each following line of code should also be indented to line up with the line above it, until the closing curly bracket, '}', is reached. Although indentation is technically not needed to compile and run your program, it is a very good programming practice and is required in this class. Proper indentation allows you and others to read and easily follow the logic of your program.

6. Use explicit brackets at all times to demonstrate the flow of control in your programs. For example, even though option 1, 2 and 3 below will run identically, option 1 will cost you points:

   Option 1 (the incorrect way – although works when ONLY printing ONE line):

```
if(number == 1)
    cout << "number = " << number << endl;

cout << "Complete." << endl;
```

   Option 2 (the correct way):

```
if(number == 1)
{
    cout << "number = " << number << endl;
}

cout << "Complete." << endl;
```

   Option 3 (the correct way):

```
if(number == 1){
    cout << "number = " << number << endl;
}

cout << "Complete." << endl;
```

7. Insert appropriate and informative in-line comments in your program, especially when you do something that may not be entirely obvious to someone not familiar with your program. When questioning whether or not to add a comment, in general, your best bet is to add it. However, be careful, or you will end up with a program with a comment on every line. You are writing a program, not an essay. See the example program at the end of this document for an idea of when to insert comments.

8. Include a comment block at the top of every program. Include your name, the date, the filename, and a description (a few complete sentences) of what the program will do. Example formats:

```
//////////////////////////////////////////////////////////
//
// Programmer: Your name
// Date: Today's date
// Name: The filename
// Description: A brief summary of what the program does.
//
//////////////////////////////////////////////////////////

//------------------------------------------------------
// Programmer: Your name
// Date: Today's date
// Name: The filename
// Description: A brief summary of what the program does.
//------------------------------------------------------

/*******************************************************
 * Programmer: Your name
 * Date: Today's date
 * Name: The filename
 * Description: A brief summary of what the program does.
 *******************************************************/
```

9. Above each function other than main, include a function comment block to describe the function.

```
//////////////////////////////////////////////////////////
//
// Description: << Describe what the function does. >>
// Inputs: << List the name, type, and description of each
//         variable that is being sent to the function. >>
// Output: << List the name, type, and description of any
//         variables that will be returned. >>
//
//////////////////////////////////////////////////////////
```

10. Always include descriptive variable names in a function prototype even though they are not required in C/C++. This will make your code more understandable. For example, suppose a function prints the month, day and year to the screen. The following prototype is ambiguous as to what order to pass the variables day, month, or year to the function since there are six possible combinations.

```
void printDay(int, int, int);
```

However, the following example prototypes provide clear information as to what order to pass variables to the function.

```
void printDay(int month, int day, int year);
void printDay(int day, int month, int year);
void printDay(int year, int month, int day);
```

Example program using good programming practices

```
////////////////////////////////////////////////////////////////
//
// Programmer: John Doe
// Date: January 19, 2015
// Name: futureValue.cpp
// Description: This program calculates a future value for money
//     invested in some account with some interest rate.
//
////////////////////////////////////////////////////////////////

#include<iostream>
using namespace std;

float calculateFutureValue(float monthlyInvestment,
                           float interestRate, int years);

int main()
{
  int choice = 0;                // used to store menu choice
  int years = 0;                 // number of years to invest money
  float monthlyInvestment = 0;   // dollar amount to invest each month
  float interestRate = 0;        // annual interest rate
  float futureValue = 0;         // future value of investment

  choice = 1;           // set to 1 to enter the while loop the first time

  // Loop to repeat future value calculation
  while(choice == 1)
  {
    // Prompt user to enter variables for calculation
    cout << "Enter monthly investment:     ";
    cin >> monthlyInvestment;
    cout << "Enter yearly interest rate:   ";
    cin >> interestRate;
    cout << "Enter number of years:        ");
    cin >>  years;

    // Calculate future value and store in futureValue
    futureValue = calculateFutureValue(monthlyInvestment,
                                       interestRate, years);

    // Print future value of investment
    cout << "\nFuture value: " << futureValue << endl;

    // Ask if user wants to continue
    cout << "\n\nEnter 1 to continue, 0 to quit:  ";
    cin >> choice;
  }

  return 0;   // Return 0 to signal that program terminated properly

}
```

```
//////////////////////////////////////////////////////////////////////
//
// Function: Calculate the future value of investment
// Input:
//   monthlyInvestment - float - how much user deposits each month
//   interestRate - float - interest earned per year
//   years - int - number of years money will gather interest
// Output:
//   futureValue - float - amount in account at end of time period
//
//////////////////////////////////////////////////////////////////////

float calculateFutureValue(float monthlyInvestment,
                           float interestRate, int years)
{
  int i = 0;                       // loop counter
  int months = 0;                  // number of months to invest money
  float monthlyInterestRate = 0;   // monthly interest rate
  float futureValue = 0.0;         // future value

  // Calculate monthly interest rate from the yearly interest rate
  monthlyInterestRate = (interestRate/100)/12;
  months = years * 12;

  // For each month, add investment and accrue interest to balance
  for(i = 0; i < months; i++)
  {
    futureValue = (futureValue + monthlyInvestment)
                  * (1 + monthlyInterestRate);
  }

  return futureValue;
}
```