Lecture 20.1

Topics

1. Coding Conventions and Styles

____

1. Coding Conventions and Styles

There are several coding conventions and styles that are used.

This lecture will elaborate on what to be used in our class, that means writing code in class, homework, and exam.

The following code is from `cppCodingConventionStyle20230912.cpp` file, which is an update from a previous post on Canvas.

```
/**
 * Program Name: cppCodingConventionStyle20230912.cpp
 * Discussion:   Coding Style/Requirements
 *                 - Simple reminders
 *                 - Referencing other style guides is
 *                   very welcome
 * Written By:   Author
 * Date:         2022/08/17, 2023/01/25, 2023/01/28, 2023/09/12
 */

// Making sure the above "Program Comment Block" to have the
// exact indentation, alignment, spacing, and format

// Choosing one of the two styles for curly braces
//   1. On-the-same-line
//   2. On-the-next-line

// The coding demo will use the on-the-same-line style
// Include/Header File(s)
#include <iostream>
using namespace std;

// Function Prototypes

void readAndReadAndRead(void);
void demoStyleA(void);
void demoStyle(void);
void checkBeforeSubmittingWork(void);
void requireProperStructure(void);
void observeCodeCosting(void);
void avoidThese(void);

// Application Driver
int main() {

    observeCodeCosting();

    requireProperStructure();

    readAndReadAndRead();

    checkBeforeSubmittingWork();

    avoidThese();
    return 0;
}


// Function Definitions

void readAndReadAndRead() {
    cout << "\n\n+++++++++++++++++++++++++++++++"
        "\n\nPLEASE READ THE CODE SHOWN IN THE FUNCTIONS BELOW"
        "\n\n           demoStyle() & demoStyleA()\n"
        "\n                     and others"
        << endl;
}

void demoStyleA(void) { // DON'T -- no "void" in the argument
                        //          list of a function definition
    cout << "\n!!!???%%% :-(  :-(()" << endl;
}

void demoStyle() { // DO
    // The following considerations and implementations are,
    // at least, for C and C++. Some of these can be used in
    // other programming languages as you see fit!

    // ++++++++++++++++++++
    // The body of a code block is indented.

    // A code block may be the body of
    //   - function definition,
    //   - if, switch, for, while, or do-while block
    //     (or statement)
    // In general, a code block can be formed by just
    // coding the logic between a pair of curly braces.

    // Blank line is used to separate logic --
    //     1 blank line please!

    // Consistently, a line indentation must be of 4 spaces.

    int meaningfulName; // Variable declaration

                        // In my classes, please declare all
                        // variables at the top of function
                        // before any other logical/operational
                        // statements.

                        // Values to be used for variables must
                        // be reasonable, meaningful, and
                        // logically proper!

                        // Anything else besides reasonable,
                        // meaningful, and logically proper are
                        // considered as ill-advised tricks and
                        // magic (numbers).

                        // Tricks and magic numbers are not to be
                        // used in any of my classes!

    int negatingValue;

    double anotherMeaningFulName = 1234.5678; // why 1234.5678?

                                              // what/How should
```

```
                                            // 1234.5678 be
                                            // interpreted and
                                            // understood for?

                                            // 1234.5678 MAGIC!

// space before and after binary operators
cout << "\nEnter an int: ";
cin >> meaningfulName;

if (meaningfulName != 0) { // DON'T
    cout << "\nDo something!\n" << endl;
}

if (meaningfulName) { // DO
    cout << "\nDo something!\n" << endl;
}

if (meaningfulName == 0) { // DON'T
    cout << "\nDo something!\n" << endl;
} else {
    cout << "\nDo some other thing!\n" << endl;
}

if (meaningfulName) { // DO
    cout << "\nDo some other thing!\n" << endl;
} else {
    cout << "\nDo something!\n" << endl;
}

   // one space between "if" and the opening '('
   // one space between the closing ')' and the opening '{'
   // no space after '('
   // no space before ')'

// BAD
if(meaningfulName){
    cout << "\nDo something!\n" << endl;
}

// GOOD and DO
if (meaningfulName) {
    cout << "\nDo something!\n" << endl;
}

// No space between the operand and unary operators

// BAD
meaningfulName ++;
-- meaningfulName;
negatingValue = - meaningfulName;

// GOOD
meaningfulName++;
--meaningfulName;
negatingValue = -meaningfulName;

negatingValue += -meaningfulName;

// NEVER DO
meaningfulName = -1 * meaningfulName;

// ALWAYS DO
```

```
meaningfulName = -meaningfulName;

// one space between "while" and the opening '('
while (meaningfulName) {
    cout << "\nDo something!\n" << endl;

    meaningfulName = 0;
}

// one space between "for" and the opening '('
// one space after ';' within the for-header

// BAD
for (int i=0;i<1;i++) {
    cout<<"\nDo something!\n"<<endl;
}

// GOOD
for (int i = 0; i < 1; i++) {
    cout << "\nDo something!\n" << endl;
}

// one space between "do" and the opening '{'
do {
    cout << "\nDo something!\n" << endl;
} while (meaningfulName);

// Again, NO NO
negatingValue = -1 * meaningfulName; // NO NO

negatingValue = -meaningfulName; // ABSOLUTE YES YES
meaningfulName = -meaningfulName;

// YES to the below
cout << ((meaningfulName < 0) ? -meaningfulName
                              : meaningfulName)
    << endl;

// NOT so great!
if (meaningfulName < 0) {
    cout << -meaningfulName << endl;
} else {
    cout << meaningfulName << endl;
}

// Be aware
cout << "\nAdditional structural hints for code expressions"
    "\nand statements may be added to the above list or "
    "\ndiscussed accordingly :-))"
    << endl;

// For my class homework submissions
cout << "\nPlease strip off all of the comments regarding"
    "\nyour code logic or implementation!"
    "\n\nI can certainly manage to read and understand your"
    "\ncode at this level of your code work."
    "\n\nStudents can use only syntax and structures"
    "\nintroduced in class discussions to implement"
    "\nclass work. Please double-check with the class and"
    "\ninstructor if you are NOT clear on the issues and"
    "\napplications!"
    "\n\nIf I do not understand any part of your code, "
    "\nI will check with you individually for clarification."
```

```
        << endl;
}


void checkBeforeSubmittingWork() {

    cout << "\n++++++++++++++++++++"
        "\n\nJust before submitting any of your work, please"
        "\ncheck your filenames to make sure that they are as"
        "\ncorrectly required and"
        "\n\n  - Code order and placement of "
        "\n    \"Function Prototypes\", \"Application Driver\","
        "\n    and \"Function Definitions\" must be correctly"
        "\n    applied as shown in class discussions and"
        "\n    code demonstrations!"
        "\n\n  - Please strip off all of the comments regarding"
        "\n    your code logic or implementation!"
        "\n\n  - I can certainly manage to read and understand"
        "\n    your code at this level of your code work."
        "\n\n  - If I do not understand any of your code, I will"
        "\n    check with you individually for clarification."
        "\n\n  - Students are only allowed to used syntax,"
        "\n    structures, and hints introduced in class"
        "\n    discussions to implement class work."
        "\n\n  - Please double-check with the class and"
        "\n    instructor if you are NOT clear on the issues and"
        "\n    how-to applications!"
        "\n\nAbsolutely (perhaps, no not perhaps!!!???), when"
        "\nwriting code, please keep ALL LINES appropriately and"
        "\nproperly long enough!\n"
        << endl;
}

void requireProperStructure() {

    //Conditional Structures
    cout << "\nConditional Structures"
        "\n++++++++++++++++++++"
        "\nConditional Structures and Indentation Levels --"
        "\n\n  One can just use "
        "\n  if (testExpression1) {"
        "\n      //TODO's"
        "\n  }"
        "\n\n  Or, if logically possible, one can use "
        "\n  if (testExpresion2) {"
        "\n      //TODO's"
        "\n  } else {"
        "\n      //TODO's"
        "\n  }"
        "\n\n  Or, if logically possible, one can use"
        "\n  if (testExpression3) {"
        "\n      //TODO's"
        "\n  } else if (anotherTestExpression) {"
        "\n      //TODO's"
        "\n  } else {"
        "\n      //TODO's"
        "\n  }"
        "\n\n  Note that the above extended if-else if-else must"
        "\n  be as proper with an ending of an \"else\"; please"
        "\n  keep this as one of many \"REQUIREMENTS\" when"
        "\n  writing code in all of my programming classes!"
        "\n\nEven if your code/program runs and produces the"
        "\nrequired outcomes, but the required coding styles and"
        "\nformats are not proper and not met as demonstrated,"
        "\nthen you will still receive very low score for"
        "\nall of your code, programs and work!"
        << endl;

    // Dynamic Memory Management
    cout << "\nDYNAMIC MEMORY MANAGEMENT"
        "\n+++++   +++++  +++++"
        "\n  All dynamic memory must be understood through the"
        "\n  two (2) operators \"new\" and \"delete\"."
        "\n\n  An application of \"new\" must then be"
        "\n  accompanied by an application of \"delete\" at"
        "\n  logically appropriate location in the code"
        "\n  fragment!"
        "\n\n  ABSOLUTELY, NO MEMORY LOSS in the implementation"
        "\n  at any point of code execution!"
        "\n\nSmart pointer objects (or just smart pointers) are"
        "\nto be used when reminded/noted/required!"
        << endl;
}


void observeCodeCosting() {

    cout << "\n++++++++++++++++++++"
        "\nSOME GENERAL UNDERSTANDING OF CODE IMPLEMENTATION --"
        "\n\nLet the cost of performing an integer '=' operation"
        "\nbe 1 unit of effort (1 UoF)."
        "\n\nAnd, let the cost of performing an integer"
        "\n'+' operation be 4 units of effort (4 UoF's)."
        "\n\nThe following general observations and understanding"
        "\nwould be helpful in deciding or selecting code"
        "\nexpressions and structures while writing the program."
        "\n\n  The unary negation operation does not cost more"
        "\n  than the '+' operation."
        "\n\n  The '-' operation does not cost less than"
        "\n  the '+' operation."
        "\n\n  The '*' operation does not cost less than"
        "\n  the '+' operation."
        "\n\n  The '/' operation does not cost less than"
        "\n  the '*' operation."
        "\n\n  The '%' operation does not cost less than"
        "\n  the '*' operation"
        "\n\n  The combined assignment operations will cost less"
        "\n  than the two individual operations combined."
        "\n\n  In an expression, the less is mostly the better!"
        "\n    - Less operations"
        "\n    - Less operands"
        "\n    - Less operations and operands in combination"
        "\n\n  The setup of if-statement will cost more than any"
        "\n  direct operator usage (if one is possible)."
        "\n\n  If usage is possible, a switch-statement is not"
        "\n  costing more than an if-else if-else statement."
        "\n\n  Just the setup of function call alone will cost"
        "\n  much more than any of the above"
        "\n  code-implementation-costing considerations."
        "\n\n  However, functions are the \"MUST USE\" in writing"
        "\n  code. Then, one must design the solution-algorithm"
        "\n  very carefully and effectively before going to"
        "\n  implementing this solution-algorithm efficiently!"
        "\n\nAnd in general, the floating-point operations will"
        "\ncost NO less than the corresponding integral"
        "\noperations."
        "\n\nAbsolutely (perhaps, no not perhaps!!!???), when"
        "\nwriting code, please keep ALL LINES appropriately and"
        "\nproperly long enough!\n"
        << endl;
}

void avoidThese() {

    cout << "\nDo NOT use TAB or '\\t'."
        "\n\nInstead, use space as many of them as you may need!\n"
        << endl;
}


/** Program_Output

++++++++++++++++++++
SOME GENERAL UNDERSTANDING OF CODE IMPLEMENTATION --

Let the cost of performing an integer '=' operation
be 1 unit of effort (1 UoF).

And, let the cost of performing an integer
'+' operation be 4 units of effort (4 UoF's).

The following general observations and understanding
would be helpful in deciding or selecting code
expressions and structures while writing the program.

  The unary negation operation does not cost more
  than the '+' operation.

  The '-' operation does not cost less than
  the '+' operation.

  The '*' operation does not cost less than
  the '+' operation.

  The '/' operation does not cost less than
  the '*' operation.

  The '%' operation does not cost less than
  the '*' operation

  The combined assignment operations will cost less
  than the two individual operations combined.

  In an expression, the less is mostly the better!
    - Less operations
    - Less operands
    - Less operations and operands in combination

  The setup of if-statement will cost more than any
  direct operator usage (if one is possible).

  If usage is possible, a switch-statement is not
  costing more than an if-else if-else statement.

  Just the setup of function call alone will cost
  much more than any of the above
  code-implementation-costing considerations.

  However, functions are the "MUST USE" in writing
  code. Then, one must design the solution-algorithm
  very carefully and effectively before going to
  implementing this solution-algorithm efficiently!

And in general, the floating-point operations will
cost NO less than the corresponding integral
operations.

Absolutely (perhaps, no not perhaps!!!???), when
writing code, please keep ALL LINES appropriately and
properly long enough!


Conditional Structures
++++++++++++++++++++
Conditional Structures and Indentation Levels --

  One can just use
  if (testExpression1) {
      //TODO's
  }

  Or, if logically possible, one can use
  if (testExpresion2) {
      //TODO's
  } else {
      //TODO's
  }

  Or, if logically possible, one can use
  if (testExpression3) {
      //TODO's
  } else if (anotherTestExpression) {
      //TODO's
  } else {
      //TODO's
  }

  Note that the above extended if-else if-else must
  be as proper with an ending of an "else"; please
  keep this as one of many "REQUIREMENTS" when
  writing code in all of my programming classes!

Even if your code/program runs and produces the
required outcomes, but the required coding styles and
formats are not proper and not met as demonstrated,
then you will still receive very low score for
all of your code, programs and work!

DYNAMIC MEMORY MANAGEMENT
+++++   +++++  +++++
  All dynamic memory must be understood through the
  two (2) operators "new" and "delete".

  An application of "new" must then be
  accompanied by an application of "delete" at
  logically appropriate location in the code
  fragment!

  ABSOLUTELY, NO MEMORY LOSS in the implementation
  at any point of code execution!

Smart pointer objects (or just smart pointers) are
```

```
to be used when reminded/noted/required!


++++++++++++++++++++++++++++++

PLEASE READ THE CODE SHOWN IN THE FUNCTIONS BELOW

           demoStyle() & demoStyleA()

                    and others

++++++++++++++++++++

Just before submitting any of your work, please
check your filenames to make sure that they are as
correctly required and

  - Code order and placement of
    "Function Prototypes", "Application Driver",
    and "Function Definitions" must be correctly
    applied as shown in class discussions and
    code demonstrations!

  - Please strip off all of the comments regarding
    your code logic or implementation!

  - I can certainly manage to read and understand
    your code at this level of your code work.

  - If I do not understand any of your code, I will
    check with you individually for clarification.

  - Students are only allowed to used syntax,
    structures, and hints introduced in class
    discussions to implement class work.

  - Please double-check with the class and
    instructor if you are NOT clear on the issues and
    how-to applications!

Absolutely (perhaps, no not perhaps!!!???), when
writing code, please keep ALL LINES appropriately and
properly long enough!


Do NOT use TAB or '\t'.

Instead, use space as many of them as you may n

*/
```