Home | HW Sets | Schedule | Files | Help | Li

CSCI 200 - Fall 2023
Foundational Programming Concepts & Design

Style Guidelines

Style Guidelines

(Download full source package)

· `main.cpp` · `Functions.h` · `Functions.cpp` · `Template.hpp` · `Class.h` · `Class.cpp` · `Comparable.h` · `AbstractClass.h` · `AbstractClass.cpp` · `ConcreteClass.h` · `ConcreteClass.cpp` · `Makefile` ·

`main.cpp` (Download file)

```
// this is a single line comment

/*
 *  this is a block comment that
 *  can span several lines
 */

// place all header files you've written together, in alphabetical order
// include only the header files you need
#include "AbstractClass.h"
#include "Class.h"
#include "ConcreteClass.h"
#include "Functions.h"
#include "Template.hpp"

// place all C++ standard libraries together, in alphabetical order
// include only the libraries you need
#include <fstream>          // for file streams (ifstream, ofstream)
#include <iomanip>          // for I/O Manipulators (precision, aligning, etc.)
#include <iostream>         // for standard input/output
#include <string>           // for string library
#include <vector>           // for vector library
using namespace std;        // we are using the standard namespace

// place all C standard libraries together, in alphabetical order
```

RMP-H

```
#include <cstdlib>          // for srand(), rand()
#include <ctime>            // for time()

/*
 * any variables defined above main() are in global scope and can be
 *  accessed anywhere.  This is generally BAD.
 * Only declare & define constants in global scope.
 * Constants are named using UPPER_SNAKE_CASE to denote it as an
 * immutable value
 */
const double PI_VALUE = 3.1415926535;

// structs are named using UpperCamelCase to denote it is a datatype
struct ClassRoom {
  string buildingName;
  int roomNumber;
};

// every program must have a main() function
//  - it is the entry point to our program
int main(int argc, char* argv[]) {  // the curly brace begins a new code block
  srand( (unsigned int)time(0) );   // seed the RNG - do this once per program as first st

  // indent the contents of a code block two spaces length

  // main() accepts parameters corresponding to the command line arguments
  // used to call and start the program
  // argc holds how many arguments there are
  cout << "Program run with " << argc << " arguments:" << endl;
  for(int i = 0; i < argc; i++) {
    // argv contains each argument in an array of C-strings
    cout << "\t" << i << ": " << argv[i] << endl;
  }

  // Template for variable declaration
  //        anything inside of [brackets] is optional
  // Version1: [modifiers] dataType identifierName [= initialValue];
  // Version2: [modifiers] dataType identifierName[(initialValue)];

  int ageOfColosseum;               // declare a variable
                                    // use lowerCamelCase to make a descriptive variable r
  ageOfColosseum = 1940;            // define a variable (assign a value)

  int numRomanEmperors = 71;        // declare and define a variable in one line

  const int VATICAN_BUILT = 1626;   // declare and define a constant
                                    // must define a constant when it is declared
                                    // use UPPER_SNAKE_CASE to make a descriptive constant

  // declare (and define) multiple variables of the same type at once
  char firstInitial = 'I', secondInitial('T'), currentEmperorInitial;
```

```
cout << "There have been " << numRomanEmperors << " emperors and the current is "
     << currentEmperorInitial << "." << secondInitial << "."
     << endl;

cout << "The Colosseum was built in 70-80 A.D. and is "
     << ageOfColosseum << " years old." << endl;
cout << endl;

// when  prompting the user to enter a value via cin, preceed the input with a prompt
//    using cout so the user knows what to enter
int currentYear;
cout << "Please enter the current 4 digit year (e.g. 1999): ";
cin >> currentYear;
cout << "St. Peter's Basilica was built in " << VATICAN_BUILT
     << " and is " << (currentYear - VATICAN_BUILT) << " years old." << endl;

if( currentYear >= 2400 ) {
  // indent the contents of a new code block
  cout << "Duck Dodgers of the 24th and a Half Century!" << endl;
} else if( currentYear >= 2000 ) {    // place else if and else on the same line the pri
  cout << "In the year 2000, robots will do 80% of the work. - Conan O'Brien." << endl;
} else {
  // use curly braces to denote a code block for if/else if/else even if the code
  //     block only has one statement
  cout << "Let's party like it's 1999. - Prince" << endl;
}

for( int i = 0; i < 10; i++ ) {
  // indent the contents of a new code block
  // use a code block for a for loop even if it contains only a single statement
  cout << i << endl;
}

char userResponse;
do {
  cout << "Enter a letter (q to quit): ";
  cin >> userResponse;
} while( userResponse != 'q' );   // place the while on the same line as the end of the

// be aware of floating point precision and output
cout << "PI was assigned:    3.1415926535" << endl;
cout << "Our value of PI is: " << PI_VALUE << endl;
cout << "Our value of PI is: " << setprecision(10) << PI_VALUE << endl;
cout << "Our value of PI is: " << setprecision(20) << PI_VALUE << endl;

// for function calls, add spaces within the () and after each argument
cout << "5 + 3 = " << calculator_add( 5, 3 ) << endl;
cout << "5 - 3 = " << calculator_sub( 5, 3 ) << endl;

ClassRoom lectureLab;             // create a variable of our custom struct type
```

```
cout << "The length of the building name is: "
     << lectureLab.buildingName.length()  // access string functions with dot operator
     << endl;

// Template for vector declaration
//        anything inside of [brackets] is optional
// Version1: [const] vector< dataType > identifierName;
// Version2: [const] vector< dataType > identifierName[(initialSize)];
// Version3: [const] vector< dataType > identifierName[(initialSize, initialValue)];
vector<int> numbers;              // create an empty vector of integers
numbers.push_back( 5 );           // add the value 5 to the vector
numbers.at( 0 ) = 6;              // access individual elements using the at() function

Calculator fourFunctionCalc( 4.5, 3.5 );  // object names follow lowerCamelCase
cout << fourFunctionCalc.getLeftHandSide() << " + " << fourFunctionCalc.getRightHandSide
     << fourFunctionCalc.add() << endl;
fourFunctionCalc.setLeftHandSide( 13.0 );
fourFunctionCalc.setRightHandSide( 1.5 );
cout << fourFunctionCalc.getLeftHandSide() << " / " << fourFunctionCalc.getRightHandSide
     << fourFunctionCalc.divide() << endl;

unsigned int arraySize;
cout << "How many elements do you have? ";
cin >> arraySize;

// pointer variables begin with a p and follow lowerCamelCase style
int* pArray = new int[arraySize];      // allocate an array of integers
for( unsigned int i = 0; i < arraySize; i++ ) {
  cout << "Enter array value: ";
  cin >> pArray[i];
}
delete[] pArray;                       // deallocate the array of integers

cout << "3 + 4 = "
     << apply_add_operator( 3, 4 ) << endl;

cout << "\"Hello,\" + \"World\" = "
     << apply_add_operator( "Hello,", "World" ) << endl;

cout << "0.5 + 0.25 = "
     << apply_add_operator( 0.5f, 0.25f ) << endl;

AListInt* const P_list = new ArrayInt();// declare pointer as abstract type and point at
                                        // pointer is constant and will not point at any
try {
  P_list->insert( 0, 5 );                // list contents: 5
  P_list->insert( 1, 7 );                // list contents: 5 7
  cout << "List size is " << P_list->getSize() << endl;  // prints 2
  cout << "List contents are " << P_list->toString() << endl;
  P_list->remove( 0 );                    // list contents: 7
  cout << "List contents are " << P_list->toString() << endl;
  delete P_list;

  return 0;                              // alert the OS our program exited with result 0
}   // the curly brace ends the code block that it corresponds to
    // every open brace needs a matching end brace
```

[Goto Top]

Functions.h (Download file)

```
// place header guards around all your header files
// make the name of the file the name of the value to test for
// use UPPER_SNAKE_CASE for your definition value
#ifndef FUNCTIONS_H         // ask compiler if FUNCTIONS_H has been defined
#define FUNCTIONS_H         // if not, define FUNCTIONS_H

// place ALL function prototypes into the header file
// use lower_snake_case for top level free functions

/**
 * @brief adds two integers together
 * @desc Returns the integer sum of two integer parameters
 * @param x first operand to add
 * @param y second operand to add
 * @return sum of x and y
 */
int calculator_add( int x, int y );

/**
 * @brief subtracts two integers
 * @desc Returns the integer difference of y from x (x - y)
 * @param value to start with
 * @param value to subtract
 * @return difference of x and y
 */
int calculator_sub( int x, int y );

#endif // FUNCTIONS_H       // ends the corresponding #ifndef
```

[Goto Top]

Functions.cpp (Download file)

```
int calculator_add( int x, int y ) {       // function definition
  return x + y;
}

int calculator_sub( int x, int y ) {
  return x - y;
}
```

[Goto Top]

Template.hpp (Download file)

```
// place header guards around all your header files
// make the name of the file the name of the value to test for
// use UPPER_SNAKE_CASE for your definition value
#ifndef TEMPLATE_HPP                 // ask compiler if TEMPLATE_HPP has been defined
#define TEMPLATE_HPP                 // if not, define TEMPLATE_HPP

// when a function or class is templated, its type cannot be known with the corresponding
// usage.  place the corresponding declaration and definition together in a single
// hpp file to denote
//   *.h*  - this is a header file containing declarations
//   *.*pp - this file contains definitions

// place the definitions at the top of the follow to maintain abstraction

/**
 * @brief resolves the binary plus operator for two values of the same type
 * @desc resolves the binary plus operator for two values of the same type as long
 * as the operator is defined for the correspond type
 * @param VAL_ONE left hand side of addition
 * @param VAL_TWO right hand side of addition
 * @return VAL_ONE + VAL_TWO
 */
template<typename T>
T apply_add_operator(const T VAL_ONE, const T VAL_TWO);

//----------------------------------------------------------------------------

// place the corresponding function declarations below to maintain abstraction

template<typename T>
T apply_add_operator(const T VAL_ONE, const T VAL_TWO) {
  return VAL_ONE + VAL_TWO;
}
```

[Goto Top]

`Class.h` (Download file)

```
// place header guards around all your header files
// make the name of the file the name of the value to test for
// use UPPER_SNAKE_CASE for your definition value
#ifndef CLASSNAME_H                 // ask compiler if CLASSNAME_H has been defined
#define CLASSNAME_H                 // if not, define CLASSNAME_H

// place a single Class declaration into its own file

class Calculator {                  // class names follow UpperCamelCase
public:
  /**
   * @brief creates a default calculator
   * @desc Creates a default calculator with both operands set to 1
   */
  Calculator();                     // provide a default constructor
  /**
   * @brief creates a default calculator
   * @desc Creates a default calculator with both operands set to 1
   * @param LHS left hand side of calculation
   * @param RHS right hand side of calculation
   */
  Calculator(const double, const double);      // provide a parameterized constructor

  // create The Big Three if appropriate for memory management
  // Calculator(const Calculator&);             // copy constructor
  // ~Calculator();                             // destructor
  // Calculator& operator=(const Calculator&);  // copy assignment operator

  // create getters and setters (if appropriate) for your private data members
  /**
   * @brief return the left hand side operand
   * @return value of left hand side operand
   */
  double getLeftHandSide() const;   // getters are const functions
  /**
   * @brief set the left hand side operand
   * @param LHS value of left hand side operand
   */
  void setLeftHandSide(const double);

  /**
   * @brief return the right hand side operand
   * @return value of right hand side operand
   */
   * @param RHS value of right hand side operand
   */
  void setRightHandSide(const double);

  // add any other related functions and mark as const if they do not change the callee
  /**
   * @brief return the sum of the operands
   * @return LHS + RHS
   */
  double add() const;
  /**
   * @brief return the difference of the operands
   * @return LHS - RHS
   */
  double subtract() const;
  /**
   * @brief return the product of the operands
   * @return LHS * RHS
   */
  double multiply() const;
  /**
   * @brief return the division of the operands
   * @return LHS / RHS
   */
  double divide() const;

private:
  double _leftHandSide;             // private members begin with _
  double _rightHandSide;
};

#endif // CLASSNAME_H               // ends the corresponding #ifndef
```

[Goto Top]

Class.cpp (Download file)

```
#include "Class.h"          // include the file with the corresponding prototypes

Calculator::Calculator() {
  _leftHandSide = 1.0;
  _rightHandSide = 1.0;
}

Calculator::Calculator(const double LHS, const double RHS) {
  _leftHandSide = LHS;
  _rightHandSide = RHS;
}
```

```
}
void Calculator::setLeftHandSide(const double LHS) {
  _leftHandSide = LHS;
}

double Calculator::getRightHandSide() const {
  return _rightHandSide;
}
void Calculator::setRightHandSide(const double RHS) {
  _rightHandSide = RHS;
}

double Calculator::add() const {
  return _leftHandSide + _rightHandSide;
}

double Calculator::subtract() const {
  return _leftHandSide - _rightHandSide;
}

double Calculator::multiply() const {
  return _leftHandSide * _rightHandSide;
}

double Calculator::divide() const {
  if( _rightHandSide != 0 ) {
    return _leftHandSide / _rightHandSide;
  } else {
    return 0.0;
  }
}
```

[Goto Top]

Comparable.h (Download file)

```
// place header guards around all your header files
// make the name of the file the name of the value to test for
// use UPPER_SNAKE_CASE for your definition value
#ifndef COMPARABLE_H                // ask compiler if COMPARABLE_H has been defined
#define COMPARABLE_H                // if not, define COMPARABLE_H

#include <string>

// interfaces begin with an I
class IComparable {                             // class names follow UpperCamelCase
public:
    virtual ~IComparable() {}                   // interfaces must have a virtual destruc
    *
    * @return std::string holding the object information
    */
    virtual std::string toString() const = 0;

    /**
    * @brief determines if two objects are equivalent
    *
    * @return true if objects are equivalent
    * @return false otherwise
    */
    virtual bool equals(const IComparable&) const = 0;
};

#endif//COMPARABLE_H
```

[Goto Top]

`AbstractClass.h` **(Download file)**

```
// place header guards around all your header files
// make the name of the file the name of the value to test for
// use UPPER_SNAKE_CASE for your definition value
#ifndef ABSTRACT_CLASSNAME_H                 // ask compiler if ABSTRACT_CLASSNAME_H has b
#define ABSTRACT_CLASSNAME_H                 // if not, define ABSTRACT_CLASSNAME_H

#include "Comparable.h"

// place a single Class declaration into its own file

// abstract classes begin with an A
class AListInt : public IComparable {                             // class names follow U
public:
    /**
    * @brief initialize list size to zero
    * @desc initializes list size to zero
    */
    AListInt();

    /**
    * @brief Destroy the AListInt object
    * @desc Abstract classes must have a virtual destructor
    */
    virtual ~AListInt();

    /**
    */
    int getSize() const;

    /**
    * @brief returns the value at a given position
    *
    * @param POS position to retrieve
    * @return int value at POS
    * @throws std::out_of_range if POS is not within [0, size)
    */
    virtual int get(const int POS) const = 0;

    /**
    * @brief add an integer value to the list
    * @desc inserts the corresponding value immediately before the specified position.
    *   list size is increased by one
    * @param POS position to insert before
    * @param VAL value to insert
    */
    virtual void insert(const int POS, const int VAL) = 0;    // create an abstract functi

    /**
    * @brief removes the value at the corresponding position
    * @desc if the position is within range, removes the corresponding element.  list size
    *   is decreased by one
    * @param POS position to remove at
    */
    virtual void remove(const int POS) = 0;    // create a pure virtual function with no c

    /**
    * @brief prints the contents of the list deliminated by spaces
    *
    * @return std::string string representation of list
    */
    std::string toString() const override final;

    /**
    * @brief compares if contents of lists are equivalent by checking their string represe
    *
    * @return true if strings are equal
    * @return false otherwise
    */
    bool equals(const IComparable&) const override final;

protected:
    int mSize;                                // protected data members begin with m
};
```

[Goto Top]

AbstractClass.cpp (Download file)

```
#include "AbstractClass.h"

#include <iostream>
using namespace std;

AListInt::AListInt() {
  cout << "AListInt() called" << endl;
  mSize = 0;
}

AListInt::~AListInt() {
  cout << "~AListInt() called" << endl;
}

int AListInt::getSize() const {
  return mSize;
}

std::string AListInt::toString() const {
  string str = "";
  for(int i = 0; i < mSize; i++) {
    str += to_string( get(i) );
    if(i != mSize - 1) {
      str += " ";
    }
  }
  return str;
}

bool AListInt::equals(const IComparable& OTHER) const {
  return this->toString() == OTHER.toString();
}
```

[Goto Top]

ConcreteClass.h (Download file)

```
// make the name of the file the name of the value to test for
// use UPPER_SNAKE_CASE for your definition value
#ifndef CONCRETE_CLASSNAME_H                 // ask compiler if CONCRETE_CLASSNAME_H has b
#define CONCRETE_CLASSNAME_H                 // if not, define CONCRETE_CLASSNAME_H

// place a single Class declaration into its own file

#include "AbstractClass.h"

class ArrayInt final : public AListInt {      // class names follow UpperCamelCase
// use public inheritance to follow the exposed interface of the parent class
// mark concrete classes as final to close them
public:
    /**
    * @brief Construct a new ArrayInt object
    * @desc initializes a new ArrayInt
    */
    ArrayInt();

    /**
    * @brief Destroy the ArrayInt object
    * @desc deletes the internal array
    */
    ~ArrayInt();

    /**
    * @brief returns the value at a given position
    *
    * @param POS position to retrieve
    * @return int value at POS
    * @throws std::out_of_range if POS is not within [0, size)
    */
    int get(const int POS) const override final;

    /**
    * @brief add an integer value to the list
    * @desc inserts the corresponding value immediately before the specified position.
    *   list size is increased by one
    * @param POS position to insert before
    * @param VAL value to insert
    */
    void insert(const int POS, const int VAL) override final;    // mark concrete functior

    /**
    * @brief removes the value at the corresponding position
    * @desc if the position is within range, removes the corresponding element.  list size
    *   is decreased by one
    * @param POS position to remove at
    */
    void remove(const int POS) override final;  // mark concrete functions as overridden a
private:
    int* _pArray;                             // derived classes can add their own additi
```

[Goto Top]

ConcreteClass.cpp (Download file)

```
#include "ConcreteClass.h"          // include the file with the corresponding prototypes

#include <exception>
#include <iostream>
#include <string>
using namespace std;

ArrayInt::ArrayInt() {
  cout << "ArrayInt() called" << endl;
  _pArray = nullptr;
}

ArrayInt::~ArrayInt() {
  cout << "~ArrayInt() called" << endl;
  delete _pArray;
}

int ArrayInt::get(const int POS) const {
  if(_pArray == nullptr) {
    string msg = "array is uninitialized";
    throw out_of_range(msg);
  }
  if(POS < 0 || POS >= mSize) {
    string msg = to_string(POS);
    msg += " is out of range for array of length ";
    msg += to_string(mSize);
    throw out_of_range(msg);
  }
  return _pArray[POS];
}

void ArrayInt::insert(const int POS, const int VAL) {
  // provide proper implementation
  mSize++;
  if(_pArray != nullptr) _pArray[POS] = VAL;
}

void ArrayInt::remove(const int POS) {
  // provide proper implementation
  mSize--;
```

[Goto Top]

`Makefile` (Download file)

```
# COMMENTS BEGIN WITH A HASH

# THE NAME OF YOUR EXECUTABLE
TARGET = StyleGuideExample
# ALL CPP COMPILABLE IMPLEMENTATION FILES THAT MAKE UP THE PROJECT
SRC_FILES = main.cpp AbstractClass.cpp Class.cpp ConcreteClass.cpp Functions.cpp

# NO EDITS NEEDED BELOW THIS LINE

CXX = g++
CXXFLAGS = -O2
CXXFLAGS_DEBUG = -g
CXXFLAGS_ERRORS = -Werror -Wall -Wextra -Wconversion -Wdouble-promotion -Wunreachable-code
CPPVERSION = -std=c++17

OBJECTS = $(SRC_FILES:.cpp=.o)

ifeq ($(shell echo "Windows"), "Windows")
    TARGET := $(TARGET).exe
    DEL = del
    Q =
else
    DEL = rm -f
    Q = "
endif

all: $(TARGET)

$(TARGET): $(OBJECTS)
    $(CXX) -o $@ $^

.cpp.o:
    $(CXX) $(CXXFLAGS) $(CPPVERSION) $(CXXFLAGS_DEBUG) $(CXXFLAGS_ERRORS) -o $@ -c $<

clean:
    $(DEL) $(TARGET) $(OBJECTS)

depend:
    @sed -i.bak '/^# DEPENDENCIES/,$$d' Makefile
    @$(DEL) sed*
    @echo $(Q)# DEPENDENCIES$(Q) >> Makefile
    @$(CXX) -MM $(SRC_FILES) >> Makefile

.PHONY: all clean depend
```

```
    Functions.h Template.hpp
AbstractClass.o: AbstractClass.cpp AbstractClass.h Comparable.h
Class.o: Class.cpp Class.h
ConcreteClass.o: ConcreteClass.cpp ConcreteClass.h AbstractClass.h \
    Comparable.h
Functions.o: Functions.cpp Functions.h
```

[Goto Top]

[Jump to Top] [Site Map]