Ξ HW Sets

Schedule

Files

③ Help ▼

⊘ Lir

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
                           // for vector library
#include <vector>
using namespace std;
                           // we are using the standard namespace
// place all C standard libraries together, in alphabetical order
```

₹ HW Sets

聞 Schedule

Files

③ Help ▼

@ Lir

```
#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;</pre>
  for(int i = 0; i < argc; i++) {
    // argv contains each argument in an array of C-strings
    cout << "\t" << i << ": " << arqv[i] << endl;</pre>
  }
  // 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)
                                    // declare and define a variable in one line
  int numRomanEmperors = 71;
  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;
```

m Schedule

Files

③ Help ▼

Ø Lir

```
cout << "There have been " << numRomanEmperors << " emperors and the current is "</pre>
     << currentEmperorInitial << "." << secondInitial << "."
     << endl;
cout << "The Colosseum was built in 70-80 A.D. and is "</pre>
     << ageOfColosseum << " years old." << endl;
cout << endl;</pre>
// 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): ";</pre>
cin >> currentYear:
cout << "St. Peter's Basilica was built in " << VATICAN BUILT</pre>
     << " and is " << (currentYear - VATICAN BUILT) << " years old." << endl;</pre>
if( currentYear >= 2400 ) {
  // indent the contents of a new code block
  cout << "Duck Dodgers of the 24th and a Half Century!" << endl;</pre>
} 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;</pre>
  // 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;</pre>
}
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;</pre>
char userResponse;
do {
 cout << "Enter a letter (q to quit): ";</pre>
 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:</pre>
                              3.1415926535" << endl;
cout << "Our value of PI is: " << PI_VALUE << endl;</pre>
cout << "Our value of PI is: " << setprecision(10) << PI_VALUE << endl;</pre>
cout << "Our value of PI is: " << setprecision(20) << PI_VALUE << endl;</pre>
// for function calls, add spaces within the () and after each argument
cout << "5 + 3 = " << calculator_add( 5, 3 ) << endl;</pre>
cout << "5 - 3 = " << calculator sub( 5, 3 ) << endl;</pre>
                                   // create a variable of our custom struct type
ClassRoom lectureLab;
```

m Schedule

Files

③ Help ▼

⊘ Lir

```
cout << "The length of the building name is: "</pre>
     << 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</pre>
     << fourFunctionCalc.add() << endl:
fourFunctionCalc.setLeftHandSide( 13.0 );
fourFunctionCalc.setRightHandSide( 1.5 );
cout << fourFunctionCalc.getLeftHandSide() << " / " << fourFunctionCalc.getRightHandSide</pre>
     << fourFunctionCalc.divide() << endl;
unsigned int arraySize;
cout << "How many elements do you have? ";</pre>
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++ ) {</pre>
  cout << "Enter array value: ";</pre>
  cin >> pArray[i];
delete[] pArray;
                                       // deallocate the array of integers
cout << "3 + 4 = "
     << apply_add_operator( 3, 4 ) << endl;</pre>
cout << "\"Hello,\" + \"World\" = "</pre>
     << apply_add_operator( "Hello,", "World" ) << endl;</pre>
cout << "0.5 + 0.25 = "
     << apply_add_operator( 0.5f, 0.25f ) << endl;</pre>
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
                                         // list contents: 5 7
  P list->insert( 1, 7 );
  cout << "List size is " << P_list->getSize() << endl; // prints 2</pre>
  cout << "List contents are " << P_list->toString() << endl;</pre>
  P list->remove( 0 );
                                           // list contents: 7
  cout << "List contents are " << P_list->toString() << endl;</pre>
```

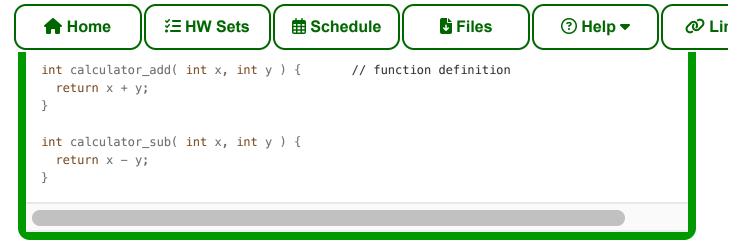


Functions.h (Download file)

```
// 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)



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;
}
```

iii Schedule

Files

? Help ▼

@ Lir

[Goto Top]

Class.h (Download file)

```
// place header quards 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
   st @desc Creates a default calculator with both operands set to 1
                                    // provide a default constructor
  Calculator();
  /**
   * @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
   */
```

Schedule

Files

? Help ▼

@ Lir

```
* @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)

₹ HW Sets

m Schedule

Files

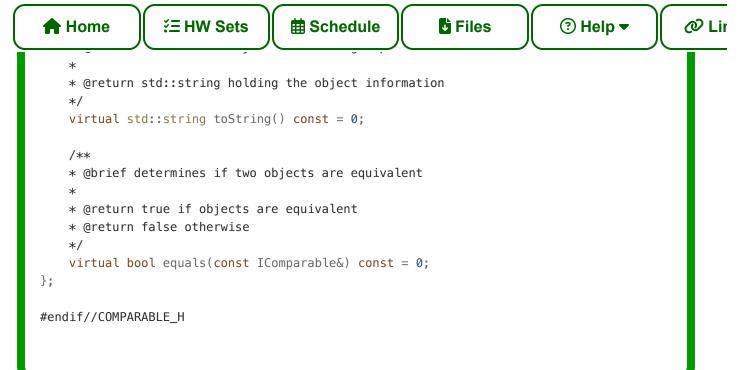
? Help ▼

ல Lir

```
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)



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 t
#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 L
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();
    /**
```

m Schedule

Files

② Help ▼

ல Lir

```
*/
   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
   st @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
   /**
   * @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
```

};



AbstractClass.cpp (Download file)

```
#include "AbstractClass.h"
#include <iostream>
using namespace std;
AListInt::AListInt() {
  cout << "AListInt() called" << endl;</pre>
  mSize = 0;
AListInt::~AListInt() {
  cout << "~AListInt() called" << endl;</pre>
int AListInt::getSize() const {
  return mSize;
}
std::string AListInt::toString() const {
  string str = "";
  for(int i = 0; i < mSize; i++) {</pre>
    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)

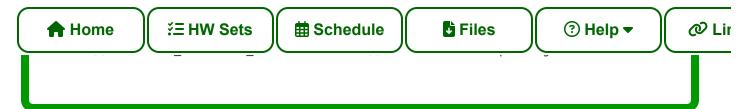
Schedule

Files

③ Help ▼

ல Lir

```
// make the hame of the litte the hame of the varie to rest for
// use UPPER SNAKE CASE for your definition value
#ifndef CONCRETE CLASSNAME H
                                          // ask compiler if CONCRETE_CLASSNAME_H has t
#define CONCRETE CLASSNAME H
                                           // if not, define CONCRETE CLASSNAME H
// place a single Class declaration into its own file
#include "AbstractClass.h"
// 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;
   \ast @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 function
   * @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:
                                             // derived classes can add their own additi
   int* _pArray;
```



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;</pre>
  _pArray = nullptr;
}
ArrayInt::~ArrayInt() {
  cout << "~ArrayInt() called" << endl;</pre>
  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 \mid \mid 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--;
```

iii Schedule

Files

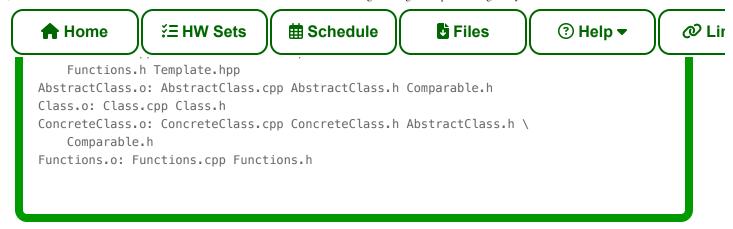
? Help ▼

@ Lir

[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 = q++
CXXFLAGS = -02
CXXFLAGS DEBUG = -q
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
    0 =
else
    DEL = rm - f
    0 = "
endif
all: $(TARGET)
$(TARGET): $(OBJECTS)
    $(CXX) -o $@ $^
.cpp.o:
    $(CXX) $(CXXFLAGS) $(CPPVERSION) $(CXXFLAGS_DEBUG) $(CXXFLAGS_ERRORS) -o $@ -c $<
clean:
    $(DEL) $(TARGET) $(OBJECTS)
    @sed -i.bak '/^# DEPENDENCIES/,$$d' Makefile
    @$(DEL) sed*
    @echo $(Q)# DEPENDENCIES$(Q) >> Makefile
    @$(CXX) -MM $(SRC_FILES) >> Makefile
.PHONY: all clean depend
```



Last Updated: 05/27/23 08:22

Any questions, comments, corrections, or request for use please contact jpaone {at} mines {dot} edu.

Copyright © 2022-2023 Jeffrey R. Paone













[Jump to Top] [Site Map]