

## Basics 2 – Inheritance

### Student Information

**Integrity Policy:** All university integrity and class syllabus policies have been followed. I have neither given, nor received, nor have I tolerated others' use of unauthorized aid.

I understand and followed these policies:                      Yes                      No

Name:

Date:

### Submission Details

Final **Changelist** number:

Verified build:                      Yes                      No

Number Tests Passed:

Required Configurations:

Discussion (What did you learn):

## Verify Builds

- Follow the Piazza procedure on submission
  - Verify your submission compiles and works at the changelist number.
- Verify that only MINIMUM files are submitted
  - No – Generated files
    - \*.pdb, \*.suo, \*.sdf, \*.user, \*.obj, \*.exe, \*.log, \*.pdb, \*.db, \*.user
    - Anything that is generated by the compiler should not be included
  - No – Generated directories
    - /Debug, /Release, /Log, /ipch, /.vs
- Typical files project files that are required
  - \*.sln, \*.cpp, \*.h
  - \*.vcxproj, \*.vcxproj.filters, CleanMe.bat

## Standard Rules

### Submit multiple times to Perforce

- Submit your work as you go to perforce several times (at least 5)
  - As soon as you get something working, submit to perforce
  - Have reasonable check-in comments
    - Points will be deducted if minimum is not reached

### Write all programs in cross-platform C++

- Optimize for execution speed and robustness
- Working code doesn't mean full credit

### Submission Report

- Fill out the submission Report
  - No report, no grade

### Code and project needs to compile and run

- Make sure that your program compiles and runs
  - Warning level ALL ...
  - NO Warnings or ERRORS
    - Your code should be squeaky clean.
  - Code needs to work "as-is".
    - No modifications to files or deleting files necessary to compile or run.
  - All your code must compile from perforce with no modifications.
    - Otherwise it's a 0, no exceptions

### Project needs to run to completion

- If it crashes for any reason...
  - It will not be graded and you get a 0

### No Containers

- NO STL allowed {Vector, Lists, Sets, etc...}
  - No automatic containers or arrays
  - You need to do this the old fashion way - **YOU EARNED IT**

### Leave Project Settings

- Do NOT change the project or warning level
  - Any changing of level or suppression of warnings is an integrity issue

### Simple C++

- No modern C++
  - No Lambdas, Autos, templates, etc...
  - No Boost
- NO Streams
  - Used fopen, fread, fwrite...
- No code in MACROS
  - Code needs to be in cpp files to see and debug it easy
- **Exception:**
  - implicit problem needs templates

### Leaking Memory

- If the program leaks memory
  - There is a deduction of 20% of grade
- If a class creates an object using new/malloc
  - It is responsible for its deletion
- Any **MEMORY** dynamically allocated that isn't freed up is **LEAKING**
  - Leaking is **HORRIBLE**, so you lose points

### No Debug code or files disabled

- Make sure the program is returned to the original state
  - If you added debug code, please return to original state
- If you disabled file, you need to re-enable the files
  - All files must be active to get credit.
  - Better to lose points for unit tests than to disable and lose all points

### No Adding files to this project

- This project will work "as-is" do not add files...
- Grading system will overwrite project settings and will ignore any student's added files and will returned program to the original state

### UnitTestFixture file (if provided) needs to be set by user

- Grading will be on the UnitTestFixture settings
  - Please explicitly set which tests you want graded... no regrading if set incorrectly

## Due Dates

- See Piazza for due date and time
- Submit program performance in your student directory assignment supplied.
- Fill out your this **Submission Report** and commit to performance
  - **ONLY** use Adobe Reader to fill out form, all others will be rejected.
  - Fill out the form and discussion for full credit.

## Goals

- Inheritance
  - Public
    - Initialization order
    - Side effects
  - Multiple Inheritance
    - Not that scary or is (getting close to Halloween?)
  - Increasing C++ knowledge and understanding

## Assignments

- General:
  - Look at these classes/structures:
    - `Wrapper.h`
    - `A.h`
    - `C.h`
    - `E.h`
    - `M.h`
  - Follow the instructions provided
    - comment section in the function
  - Run the Unit Tests to verify progress / success
    - 5/5 is the best for this program
- Run Basics2Wrapper.sln to verify correct behavior in wrapper.h

## Validation

*Simple checklist to make sure that everything is submitted correctly*

- Is the project compiling and running without any errors or warnings?
- Does the project run **ALL** the unit tests execute without crashing?
- Is the submission report filled in and submitted to performe?
- Follow the verification process for performe
  - Is all the code there and compiles "as-is"?
  - No extra files
- Is the project leaking memory?

## Hints

Most assignments will have hints in a section like this.

- This is very easy Basic assignment
  - Only thing that might slow you down "Understanding multiple inheritance"
  - Read the book
- I expect this assignment to be completed quickly for most of the students
  - Please make sure you fully understand the commands in this code
    - Look at the memory layout to understand what is really going on.
  - Many little lessons here for those who put in the effort.
- Why do programmers love Halloween and Christmas?
  - OCT 31 == DEC 25
    - (base 8) 31 == (base 10) 25
- Enjoy