

## Basics 1 – Overloading

### 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

- C++ Proficiency
  - Real-World Overloading
- Increasing C++ knowledge and understanding

## Assignments

- General:
  - Add methods and operators for overloading.
  - Run the Unit Tests to verify progress / success
    - 15/15 is the best for this program
- **Monkey Class** - Description
  - Background
    - Monkey class will be modified to support overloading correctly.
      - Monkey class - adding the Big Four operators (explicitly - no defaults)
    - The unit tests shake out the program and verify the correct functionality
  - private:
    - Monkey has 2 private variables, x and y
    - Monkey has one char string pointer called status.
  - public:
    - There are several public methods supplied
      - getX(), getY(), getStatus(), printStatus()
  - Methods to Add
    - The Big Four operators to public methods
      - Default constructor
        - initialize
          - x: 878
          - y: 979
        - Dynamically create (use new) a char string, *status*.

- initialize to: "This string was initialized by a default constructor!"
  - Copy constructor
    - deep copies the string
      - What's deep copy?
        - Look it up...
  - Assignment operator
    - deep copies the string
  - Destructor operator
    - deletes the *status* char string
    - use delete keyword
  - Specialize constructor
    - Initialize variables x and y with the passed parameters
    - Dynamically create (use new) a char string, *status*.
      - initialize to: "This string was initialized by a clever user!"
    - Initialize variable x with y being a default parameter = 575
      - Look up default assignment
  - Update the two static variables where appropriate
    - For every new allocation of a string *increment*
      - numStringsCreated
    - For every deletion of a string *increment*
      - numStringsDestroyed
    - See unit tests for verification
      - Reverse engineer the test functions for examples and clarity
  - Modify the Monkey class and run the unit tests
    - Do implementation in Monkey.cpp file, add prototypes in Monkey.h
- *Nyble Class* - Description
  - Background
    - This is class creates an abstract data type, Nyble (4 bits)
      - With overloaded operators
    - You can add numbers to this data type, it will mask if it exceeds the 4 bits of storage.
  - private:
    - Storage of the 4 bit data (actually its 8, but we are treating it as 4 bit)
  - public:
    - Method getData() returns the data
  - Methods to Add
    - The Big Four operators to public methods (explicitly - no defaults)
      - Default constructor
      - Copy constructor
      - Assignment operator
      - Destructor operator
    - Binary operators
      - Nyble + constant

- constant + Nyble
- Nyble + Nyble
- Nyble += Nyble
- Unary operators
  - ~ operator
    - Ones complement
  - +operator
    - returns the value + 3 (for academic purposes)
  - casting operator() to an unsigned int
    - subtracts 3 to the value (for academic purposes)
  - pre-increment ++
    - ++Nyble
  - post-increment ++
    - Nyble++
  - operator <<
    - Use as a rotational shift function within the nyble
    - Each bit rotates to the left by the number specified
    - If a bit fall off the edge it is rotated to the beginning bit.
      - x: 1110b    x<<1    answer x: 1101b
- Modify the Nyble class and run the unit tests
  - Do implementation in Monkey.cpp file, add prototypes in Monkey.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 performce?
- Follow the verification process for performce
  - 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 pretty easy Basic assignment
- I expect this assignment to be completed quickly for most of the students
  - Learning overloading API
- Start will all the files disabled through the \_UnitTestConfiguration.h
  - Enable one test at a time...
    - Slowly fix the linker and compiler bugs

- Once you get that test working... leave it on from that point on
  - This assignment needs to get everything working together
- Overload the big 4 operators
  - Look up the overloading signatures for ones that are new to you

## Challenges

- Create a dynamic C-String with new or malloc
  - Systems one should have introduced C-Strings ( char \*s )
  - Learn how to create, copy, initialize, dynamically create / destroy
  - Key functions to learn:
    - strcpy, strlen, strcmp, memcpy, etc...
  - Do NOT use the **string** class
    - Use C-Strings... faster and no memory slop
    - Know where the null is added in the string
- Overloading operators methods
  - Learn about binary operators
    - *class + class*
    - *class + constant*
    - *constant + class*
  - Defining operators “outside the class” is different than inside the class
- Class basics
  - Scope of methods and variables
    - Implicit conversions and promotion of variables
  - Accessor specifier {public, protected, private}
  - Casting
  - Initializing static class variables
  - Correct usage of *friend*