Basics2 – Creating Threads

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: Joey Domino

Date: 9/25/2022

Submission Details

Final ***Changelist*** number: there's one. This number

Verified build: ( ) Yes ( ) No

Number Tests Passed: 11

Required Configurations: 2: { Debug/Release | x86 }

Discussion (What did you learn):

I learned how to create a thread 11 different ways and how to use lambda functions with and without parameters or capture clauses.

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
    - 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, *.suo,
  - *.vcxproj, *.vcxproj.filters, *.vcxproj.user
  - *.cpp, *.h
  - 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
    - Seriously, I'm checking

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

Leave Project Settings

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

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
- Disable your debug printing otherwise you will lose points

Due Dates

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

Goals

- Learn
  - How to spawn a thread
    - Functions, Functors, Lambdas, Function Pointers, Member functions
    - Manys to create callable object

Assignments

1. ***Spawn several threads…***
    a. Instructions are in each Unit Test
    b. You have a lot of latitude in the way and how to prove that you created every scenario.
        i. Please take the time and verify that you are passing the data correctly
        ii. I'm verifying that you can created and launch threads in the different ways
    c. Often data is copied and reference is actually on the copy not the source.
        i. Print some addresses to help you show that you have correct addresses
    d. You will need to look up syntax and dig into the formats
        i. Sadly no one location had good references
    e. PLEASE – honor the spirit of the this assignment and try to do the threads correctly not by gaming the unit test system
2. ***Details***
    - You will spawn many different threads using different techniques
    - All threads will be immediately join() after creation
    - You will sometimes pass data to the thread for launch

```
// ---------------------------------------------------------------
// A) Spawn thread A
//    With a function
//                  With no parameters in the function
//
// 1) Create a function that takes no parameters
//    Modify A.h and A.cpp
// 2) Add to the bottome A function
//    A_result.SetTestData();
// 3) Spawn the thread here
//    See AZUL_INSERT_CODE_HERE
//---------------------------------------------------------------
```

    - Simple thread... Same as example in class

```
// ---------------------------------------------------------------
// B) Spawn thread tB
//    With a function
//        With at three parameters
//           * value - x
//           * reference - y
//           * pointer - p
//
// 1) Create a function that takes three parameters
//    Modify B.h and B.cpp
// 2) Add to the bottom B function
//    B_result.SetTestData(&x, &y, &p);
// 3) Spawn the thread in Unit Test
//    See AZUL_INSERT_CODE_HERE
//---------------------------------------------------------------
```

    - Trick here… passing the 3 different parameters

- Thread constructor actually copies data
  - So verify the connection between main thread and the new thread
  - Make sure the reference and pointer are actually from main thread and not a copy

```
// ---------------------------------------------------------------
// C) Spawn thread tC
//    With a function object (functor)
//         With no parameters, no return
//
// 2) Add a function call operator to Class C.
//         Modify C.h and C.pp
// 2) Add to the bottom C function call operator
//    Add C_result.SetTestData(*this);
// 3) Spawn the thread in Unit Test
//    See AZUL_INSERT_CODE_HERE
//----------------------------------------------------------------
```

- Functors can have data in the holding class.. that's ok
- Now you need to instantiated the object first before spawning the thread

```
// ---------------------------------------------------------------
// D) Spawn thread tD
//    With a function object (functor)
//    With three parameters, no return
//        * value - x
//        * reference - y
//        * pointer - p
//
// 1) Add a function call operator to Class D.
//    Modify D.h and D.pp
// 2) Add to the bottom D function call operator
//    Add     D_result.SetTestData(&x, &y, &p);
// 3) Spawn the thread in Unit Test
//    See AZUL_INSERT_CODE_HERE
//----------------------------------------------------------------
```

- Adding a little more complexity… now additional parameters to pass

```
// ---------------------------------------------------------------
// E) Write lambda_E
//    With lambda with empty capture clause
//        With no input parameters, no return
//
// 1) Write lambda inside Unit Test
//    With lambda with empty capture clause
//        With no input parameters
//    Inside Lambda:
//        Add function:
//            E_result.SetTestData(&x);
//            used for testing
// 2) Spawn thread tE
//----------------------------------------------------------------
```

- In a word Lambdas… Simple one
- Trick – create a stand-alone lambda then pass it to the thread

```
// ------------------------------------------------------------------
// F) Write lambda_F
//    With lambda with empty capture clause
//    With at three parameters, no return
//          * value - x
//          * reference - y
//          * pointer = p
//
// 1) Write lambda inside Unit Tests
//    With at three parameters
//          * value - x
//          * reference - y
//          * pointer = p
//    Inside Lambda:
//        Add function:
//            F_result.SetTestData(&x,&y,&p);
//            used for testing
// 2) Spawn thread tF
//------------------------------------------------------------------
```

- Passing data without using the capture clause
- Took me a while to figure out… but now it's easy.

```
// ------------------------------------------------------------------
// G) Write lambda_G
//    With lambda copy by value capture
//    With no parameters, no return
//
// 1) Write lambda inside Unit Tests
//    With lambda copy by value capture
//    With no parameters
//    Inside Lambda:
//        Add function:
//            G_result.SetTestData(&x,&y,&p);
//            used for testing
// 2) Spawn thread tG
//------------------------------------------------------------------
```

- Now you need a capture clause (copy by value)

```
// ------------------------------------------------------------------
// G) Write lambda_H
//    With lambda copy by reference capture
//    With no parameters, no return
//
// 1) Write lambda inside Unit Tests
//    With lambda copy by reference capture
//    With no parameters
//    Inside Lambda:
//        Add function:
//            H_result.SetTestData(&x,&y,&p);
//            used for testing
// 2) Spawn thread tG
//------------------------------------------------------------------
```

- Now you need a capture clause (copy by reference)

```
// ------------------------------------------------------------------
// I) Spawn thread I
//     With function pointers
//     With no parameters, no return
//
// 1) Add a function with no parameters to Class I
//    Modify I.h and I.pp
// 2) Add to the bottom I function call operator
//    Add  I_result.SetTestData();
// 3) Spawn the thread in Unit Test
//    See AZUL_INSERT_CODE_HERE
//------------------------------------------------------------------
```

- Function pointers… yikes
- Create the pointer first, then pass it in

```
// ------------------------------------------------------------------
// J) Spawn thread J
//     With function pointers
//     With at three parameters, no return
//        * value
//        * reference
//        * pointer
//
// 1) Add a function with three parameters to Class J
//    Modify J.h and J.pp
// 2) Add to the bottom I function call operator
//    Add     J_result.SetTestData(&x, &y, &p);
// 3) Spawn the thread in Unit Test
//    See AZUL_INSERT_CODE_HERE
//------------------------------------------------------------------
```

- Arguments in function pointers are dangerous
- No safety… you'll see

```
// ------------------------------------------------------------------
// K) Spawn thread K
//    With Member function
//    With no parameters, no return
//
// 1) Add a member function with three parameters to Class K
//    Modify K.h and K.pp
// 2) Add to the bottom K member function
//    Add     K_result.SetTestData();
// 3) Spawn the thread in Unit Test
//    See AZUL_INSERT_CODE_HERE
//------------------------------------------------------------------
```

- Member functions
- Need to create the object first then pass to thread

```
// ---------------------------------------------------------------------
// L) Spawn thread L
//    With Member function
//    With three parameters, no return
//        * value
//        * reference
//        * pointer
//
// 1) Add a member function with three parameters to Class L
//    Modify L.h and L.pp
// 2) Add to the bottom L member function
//    Add     L_result.SetTestData(&x, &y, &p);
// 3) Spawn the thread in Unit Test
//    See AZUL_INSERT_CODE_HERE
//---------------------------------------------------------------------
```

- Crazy sauce… more parameters
- YUM

3. ***Sample output***
   a. You don't need to copy this… but it's here to provide a sample or a reference

```
--------------------------------
    Memory Tracking: start()
--------------------------------

thread( 4864) ---MAIN---: begin()
------------------- Testing DEBUG ------------------------

A_Thread_Test: start

                (---A---):   (entry): A()
                (---A---):      (id): 0x3224
                (---A---):  (thread): ---A---
                (---A---):    (exit): A()

A_Thread_Test: end

 PASSED: A_Thread_Test

B_Thread_Test: start


        (---MAIN---):  x(0137F9B4): 5
        (---MAIN---):  y(0137F9A8): 33.299999
        (---MAIN---):  p(0137F99C): 0156D0B0 ABCD

                (---B---):  (entry): B()
                (---B---):     (id): 0x35BC
                (---B---): (thread): ---B---
                (---B---):  x(0179FE68): 5
                (---B---):  y(0137F9A8): 33.299999
                (---B---):  p(0179FE70): 0156D0B0 ABCD
                (---B---):   (exit): B()

B_Thread_Test: end

 PASSED: B_Thread_Test

C_Thread_Test: start

                (---C---):   (entry): C()
                (---C---):      (id): 0x41EC
                (---C---):  (thread): ---C---
```

```
                (---C---):         x: 99
                (---C---):    (exit): C()

C_Thread_Test: end

 PASSED: C_Thread_Test

D_Thread_Test: start

        (---MAIN---):  x(0137F9A8): 55
        (---MAIN---):  y(0137F99C): 111.099998
        (---MAIN---):  p(0137F990): 0156D2E0 rick

                (---D---):  (entry): D()
                (---D---):     (id): 0x3614
                (---D---): (thread): ---D---
                (---D---):  x(0179FB58): 55
                (---D---):  y(0137F99C): 111.099998
                (---D---):  p(0179FB60): 0156D2E0 rick
                (---D---):   (exit): D()

D_Thread_Test: end

 PASSED: D_Thread_Test

E_Thread_Test: start

        (---MAIN---):  x(0137F9B4): 33

                (---E---):   (entry): E()
                (---E---):      (id): 0x47E0
                (---E---):  (thread): ---E---
                (---E---):     x(0179FD34): 777
                (---E---):    (exit): E()

E_Thread_Test: end

 PASSED: E_Thread_Test

F_Thread_Test: start

        (---MAIN---):  x(0137F9B4): 44
        (---MAIN---):  y(0137F9A8): 222.199997
        (---MAIN---):  p(0137F99C): 0156D1C8 bird

                (---F---):      (entry): F()
                (---F---):         (id): 0x1AEC
                (---F---):     (thread): ---F---
                (---F---):  x(0179FAE0): 44
                (---F---):  y(0137F9A8): 222.199997
                (---F---):  p(0179FAE8): 0156D1C8 bird
                (---F---):       (exit): F()

F_Thread_Test: end

 PASSED: F_Thread_Test

G_Thread_Test: start

        (---MAIN---):  x(0137F9B4): 88
        (---MAIN---):  y(0137F9A8): 999.900024
        (---MAIN---):  p(0137F99C): 0156D2E0 pill

                (---G---):      (entry): G()
                (---G---):         (id): 0x1614
                (---G---):     (thread): ---G---
                (---G---):  x(0156D238): 88
                (---G---):  y(0156D23C): 999.900024
```

```
                (---G---):  p(0156D240): 0156D2E0 pill
                (---G---):       (exit): G()

G_Thread_Test: end

 PASSED: G_Thread_Test

H_Thread_Test: start

        (---MAIN---):  x(0137F9B4): 22
        (---MAIN---):  y(0137F9A8): 222.199997
        (---MAIN---):  p(0137F99C): 0156D238 cats

                (---H---):      (entry): H()
                (---H---):         (id): 0x46F8
                (---H---):     (thread): ---H---
                (---H---):  x(0137F9B4): 22
                (---H---):  y(0137F9A8): 222.199997
                (---H---):  p(0137F99C): 0156D238 cats
                (---H---):       (exit): H()

H_Thread_Test: end

 PASSED: H_Thread_Test
I_Thread_Test: start

                (---I---):      (entry): I()
                (---I---):         (id): 0x3758
                (---I---):     (thread): ---I---
                (---I---):       (exit): I()

I_Thread_Test: end

 PASSED: I_Thread_Test

J_Thread_Test: start

        (---MAIN---):  x(0137F9B4): 33
        (---MAIN---):  y(0137F9A8): 333.299988
        (---MAIN---):  p(0137F99C): 0156D1C8 dogs

                (---J---):      (entry): J()
                (---J---):         (id): 0x4604
                (---J---):     (thread): ---J---
                (---J---):  x(0179FDC4): 33
                (---J---):  y(0137F9A8): 333.299988
                (---J---):  p(0179FDCC): 0156D1C8 dogs
                (---J---):       (exit): J()

J_Thread_Test: end

 PASSED: J_Thread_Test

J_Thread_Test: start

                (---K---):      (entry): K()
                (---K---):         (id): 0x3878
                (---K---):     (thread): ---K---
                (---K---):       (exit): K()

K_Thread_Test: end

 PASSED: K_Thread_Test

L_Thread_Test: start

        (---MAIN---):  x(0137F9A8): 88
        (---MAIN---):  y(0137F99C): 888.799988
```

```
(---MAIN---):  p(0137F990): 0156D0B0 jump

        (---L---):      (entry): L()
        (---L---):        (id): 0x27A8
        (---L---):    (thread): ---L---
        (---L---):  x(0179FD24): 88
        (---L---):  y(0137F99C): 888.799988
        (---L---):  p(0179FD2C): 0156D0B0 jump
        (---L---):       (exit): L()

L_Thread_Test: end

 PASSED: L_Thread_Test

  --- Tests Results ---

      Ignored: 0
       Passed: 12
       Failed: 0

   Test Count: 12
 Indiv Checks: 201
         Mode: x86 Debug

-----------------
thread( 4864) ---MAIN---: end()

--------------------------------
    Memory Tracking: passed
--------------------------------
    Memory Tracking: end()
--------------------------------
```

4. ***Make sure it builds for all configurations***
    a. Suggestion: Implement and develop on Debug/x86
    b. After that configuration works → verify all configurations:
        i. Debug x86
        ii. Release x86

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*** in all configurations without crashing?
- Is the submission report filled in and submitted to perforce?
- Follow the verification process for perforce
    - 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.

- Do many little check-ins
  - Iteration is easy and it helps.
  - Perforce is good at it.
- READ the book
  - Many good ideas in there.
- I had to do a lot of googling and web searching
  - Not make examples out there.
  - Dig into it you'll get it