Lab4

Lab4 git link

Lab4 Part IV - output_golden.txt updated

Attached Files: output_golden.txt (3.639 KB)

The output_golden.txt has been updated here to get rid of the ^M at the end of each line (refer to the discussion forum "Always get a ^M in the end of line?") Sorry for this inconvenience.

Lab 4

EE 599 Lab4
Spring 2019 Nazarian **100 Points**

Student ID: __________________________ **Name: __________________________**

Assigned: Wednesday Jan. 30, 2019
Due: Wednesday Feb. 6 at 11:59pm

Late submissions will be accepted for two days after the deadline with a maximum penalty of 15% per day. For each day, submissions between 12am and 1am: 2% penalty; between 1 and 2am: 4%; 2-3am: 8%; and after 3am: 15%. No submissions accepted after Friday Feb. 8 at 4am.

Check the general Lab Guidelines.

What You Will Practice

You are going to get familiar with some useful tools to help you debug your programs, including GNU debugger (GDB) and Address Sanitizer.

You will also practice Fibonacci numbers and Virtual Functions along with the continued Object-Oriented Design in C++. You are encouraged to get use of the debuggers learnt from Part I & II as practice.

Part I: GDB

Part I - GNU Debugger

Starting up GDB

GDB is a popular standard debugger in the GNU operating system. In this lab we first review the basic commands and options of gdb which is the executable file of GDB. To compile a program with debugger symbols, use the *-g* flag when you compile. For programs with a Makefile, we'll usually include it for you.

Once that is done you can debug the program. The command to run gdb is

gdb EXECUTABLE_NAME

Here you can run the executable using by entering *run* or *r*. If you need to add command line arguments, you can add them when you run the executable. For example: *run ARG_1 ARG_2 ...*

Breakpoints & Printing Info

If your program has a segmentation fault, memory error, or other program-terminating problem, *gdb* will halt and let you know that you have hit an error. It is useful to stop the execution of your program at an arbitrary point. This can be accomplished by setting breakpoints. To set a breakpoint, use the *break* command. The break requires a line number to be specified, and if there are multiple source files, you must specify that too. For example:

- break LINE_NUMBER
- break sourcefile.cpp:LINE_NUMBER

Once you hit a breakpoint, you can use the *print* and *backtrace* commands. When the program stops, there are a couple commands that might be helpful.

- *print VARIABLE_NAME* will print the value of a given variable name. This can be used to see what value is incorrect and is usually a good point to start looking for bugs.
- *backtrace* or *bt* will print the stack trace; this will show you the functions that have been called up to the point of the crash.

To proceed with program execution, there are two commands you can use.

-

continue or *c* to run the program until it encounters the next breakpoint or the end of execution.

- *step* or *s* to step through the program line by line.
- *next* or *n* to step through the program, but skip over function calls (i.e. treat the function call as one line of code rather than stepping through each subsequent line of the function call)

To delete a breakpoint: *delete BREAKPOINT_NUMBER*
To list all breakpoints: *info breakpoints*
To quit gdb: quit

Summary

gdb is a powerful tool, and we've only outlined the basics. Practice of additional command and options is highly recommended.

Assignment: Download the brokengdb1.cpp file and find out the bugs present in the code.
Dump the terminal contents in a file for each debug session for the cpp and submit it with correct code.
Name your correct code as brokengdb1_*studentID*.cpp. Replace *studentID* with your own ID. For log file, name as brokengdb1_*studentID.log*
You can do that by either using

- In the gnome-terminal menu, **Edit > Select All** and then **Edit > Copy**. (Or use your favorite keyboard shortcut for the copy.)
- Using the command gdb <filename> | tee <filename>.log

Part II: Address Sanitizer

Part II - Address Sanitizer

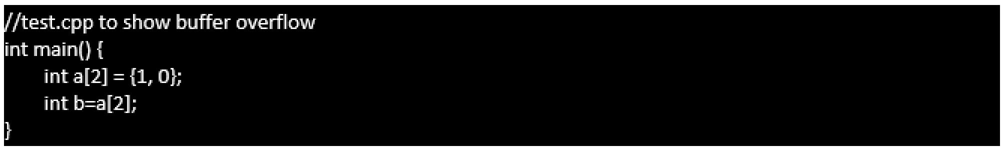
Recent versions of the compilers llvm and gcc have received a powerful tool to spot memory access bugs. It is called Address Sanitizer (ASan) and it can be enabled at compile time. Memory access bugs, including buffer overflows and uses of freed heap memory, remain a serious problem for programming languages lik C and C++.

Address Sanitizer is a memory error detector.

- Instruments each memory access.
- Checks if it' s OK to access memory at that address.
- Emits an error when accessing a non-valid address.
- libsanitizer runtime library replaces malloc/free functions.
- Addresses of freed memory are marked as being non-valid aka "poisoned" .

To use Address Sanitizer, we need to add the parameter **-fsanitize=address** to our compiler flags.
To make debugging easier we will also add **-ggdb**.

Example:



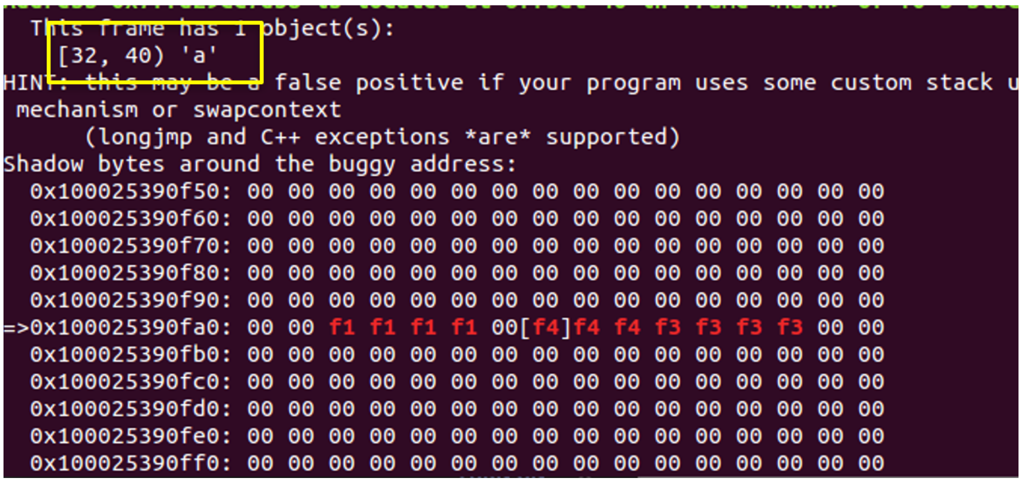
```
//test.cpp to show buffer overflow
int main() {
      int a[2] = {1, 0};
      int b=a[2];
}
```

//test. cpp to show buffer overflow

Use: **g++ -fsanitize=address -ggdb -o test test.cpp**
./test

- The following picture is the result after running the ASAN tool.
- The red number zone is the allocated memory locations for this code.
- The [] circuited out which memory location is illegal (out of range).
- In this example, since we only declared memory location a[0] & a[1], but we are accessing a[2], which is out of the declared range.
-

The yellow area shows which variable (here is ‘a’) causes the leakage (means out of range accessing).



```
  This frame has 1 object(s):
    [32, 40) 'a'
HINT: this may be a false positive if your program uses some custom stack u
 mechanism or swapcontext
      (longjmp and C++ exceptions *are* supported)
Shadow bytes around the buggy address:
  0x100025390f50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100025390f60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100025390f70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100025390f80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100025390f90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x100025390fa0: 00 00 f1 f1 f1 f1 00[f4]f4 f4 f3 f3 f3 f3 00 00
  0x100025390fb0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100025390fc0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100025390fd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100025390fe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100025390ff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Links for more reference:

- https://github.com/google/sanitizers/wiki/AddressSanitizer
- https://en.wikipedia.org/wiki/AddressSanitizer

The erroneous code named “BrokenASan.cpp” is attached, using ASan try to debug the code. State how you use ASAN to debug the code in your readme. Submit the correct code and debug log.

Part III - Fibonacci Series

Part III – Fibonacci Series

Each number in the Fibonacci series is the sum of the preceding two numbers, the first few Fibonacci numbers are:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

In mathematical terms, the sequence Fn of Fibonacci numbers is defined by the recurrence relation $F_n = F_{n-1} + F_{n-2}$ with seed values $F_0 = 0$ and $F_1 = 1$.

Now you are given a nonnegative integer N. Please print the Nth Fibonacci Number.

You are required to implement this function using two ideas: (i) recursive approach, that results in exponential complexity O(2^n), and (ii) dynamic programming, which is polynomial complexity O(n). You should store them separately in two files, namely, *fibonacci_recursive.cpp*, *fibonacci_dp.cpp*. Also compare their run time complexities, and present that the execution times of recursive method increases rapidly with the growth of input N.

Part IV - Virtual Functions

Part IV – Virtual Functions

You have learnt how to define Class and member functions in C++ in Lab3. Actually, there can be different "types" of classes, including "Base Class" and "Derived Class". A class can be used as the base class for a derived new class. The derived class inherits the public properties of the base class. You may want to make a new version of some member functions from the base class in your derived class. Then you'll need virtual functions.

A virtual function is a member function which is declared within base class and is re-defined by derived class. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the function.

Now you will practice using derived class and virtual functions.

Problem Description - "Pancakes with Fibonacci Sizes"

The chef who we thought was sloppy in Lab3 actually works for a spying agency. His next task is to pass a secret message to an agent who will dine in the restaurant. The chef makes a rectangular pan of pancakes. There are X*Y piles, with Z pancakes in each. The pancakes are already prepared by the spying agency to be of different sizes and randomly ordered. The burnt sides of pancakes in the interior piles are randomly faced, but in the exterior piles they are already facing up.

The chef is trained (it took him 20 years to master this :D) to measure pancake sizes by eye. That means he can get the value of diameter of a pancake by only looking at it. He does the following:

1) For each **interior** pile, orders the pancakes **ascendingly** by the size, with all burnt sides facing **down**;

2) For each **exterior** pile, bring all the pancakes with size as a Fibonacci number to the top of the pile, then sort **only those** pancakes **descendingly** (i.e. pancake with the largest Fibonacci size is on the top). The ones with size as a non-Fibonacci number should remain as the original order.

3) Next, for each pile, he adds the Fibonacci numbers that are at the top of the pile, also 3rd from the top, 5th from the top, etc., all the way to the last pancake with an odd rank starting from the top of the pancake. He repeats this for every **exterior** pile.

4) He now has the summation of the odd-ranked Fibonacci numbers for each pile. Next, to calculate the final encoded message which is a Code number to a secret mission, he adds up all those sum values and make a mod operation (mod base 512).

5) Now that he has the encoded message, Code, he would find a safe way to communicate that Code with the agent.

You are responsible to help the chef with the implementation of steps 1 to 4. You are not responsible for step 5!

Note: Similarly to lab3, please wash your hands before working on lab4. Please also don't eat any of the pancake, because you may eat one with a

Fibonacci size, and that would compromise the mission :P

Visual Functions and Object-oriented Design Requirements

You should design and implement the following four classes:
1) Class ***Pancake*** that includes and integer variable *Size* and a bool variable *Burnt*. *Burnt* = 1 means the top side is burnt, while 0 means the down side is burnt. A method *flip_pancake()* should be included that flips the pancake, i.e., inverts Burnt.
2) Class ***PancakePile*** should be defined (as the **public base class**). It should include an array of size Z of pancake objects of type Pancake. This class should include two functions doing the original sorting and rearranging: a) *pancake_sort_ascending_burnt_down()*; b) *pancake_sort_ descending_burnt_up()*. Note that function b) should be virtual.
3) Class ***FibPancakePile*** should be defined as the **derived class** from PancakePile. Prefix "Fib-" means Fibonacci. You should re-define your function *pancake_sort_descending_burnt_up()* in this class.
4) Class ***MPancakePile*** should include X*Y objects of class PancakePile, but you need to **use pointers** to point each object to the derived class FibPancakePile. It should include two functions: a) *sort_interior()*; b) *sort_exterior().*
5) Code is an external value (to classes) that you would define in main.

Input File Format

You will be given a few text files named "input.txt" one at a time. This file contains two pieces of information: (i) *size* of the 3D pancake volume, i.e. number of rows (X), number of columns (Y), and height of each pile (Z), separated by a space character; (ii) *size* and *burnt* side position of each pancake, separated by comma. The format of input files is defined: The first line is the size of the pancake volume. Starting from second line, each line represents a pancake pile, and it consists of Z pairs of integer numbers representing the properties of each pancake, Size followed by Burnt. Burnt=1 means the burnt side is facing up (and 0 mean the burnt side is facing down). Pancakes within each pancake pile is separated by a space character. The pancake piles are listed row by row in the X-Y dimension. Here is an example of "input.txt". The first line tells us X=3, Y=3 and Z=10. Line 2 to Line 4 represent the first row of 3 piles of pancakes, where each pile has 10 pancakes. For the following illustration, the number in red means the size of that data is a Fibonacci number.

In "input.txt" file:

```
3 3 10
60,1 5,1 20,1 49,1 85,1 41,1 47,1 96,1 89,1 76,1
98,1 74,1 58,1 96,1 19,1 92,1 10,1 59,1 95,1 6,1
49,1 5,1 71,1 52,1 30,1 27,1 75,1 57,1 59,1 66,1
84,1 70,1 52,1 22,1 9,1 89,1 49,1 9,1 98,1 93,1
```

17,1 15,0 62,1 93,1 70,1 46,1 12,0 4,1 49,0 9,0
20,1 8,1 34,1 54,1 77,1 41,1 48,1 61,1 61,1 66,1
89,1 20,1 29,1 30,1 53,1 82,1 81,1 68,1 51,1 92,1
18,1 95,1 24,1 66,1 52,1 55,1 72,1 43,1 66,1 48,1
100,1 38,1 52,1 13,1 5,1 16,1 63,1 42,1 71,1 59,1

Output File

1. After obtaining the result of soring the pancakes, you should write your results to a text files with name “**output.txt**”. The format should be the same as the input file. You should write your result to files instead of printing in terminal. For example, the desired output for above example will be as follows:

---

3 3 10
89,1 5,1 60,1 20,1 49,1 85,1 41,1 47,1 96,1 76,1
98,1 74,1 58,1 96,1 19,1 92,1 10,1 59,1 95,1 6,1
5,1 49,1 71,1 52,1 30,1 27,1 75,1 57,1 59,1 66,1
89,1 84,1 70,1 52,1 22,1 9,1 49,1 9,1 98,1 93,1
4,0 9,0 12,0 15,0 17,0 46,0 49,0 62,0 70,0 93,0
34,1 8,1 20,1 54,1 77,1 41,1 48,1 61,1 61,1 66,1
89,1 20,1 29,1 30,1 53,1 82,1 81,1 68,1 51,1 92,1
55,1 18,1 95,1 24,1 66,1 52,1 72,1 43,1 66,1 48,1
13,1 5,1 100,1 38,1 52,1 16,1 63,1 42,1 71,1 59,1

---

2. Print the Code you got in step 4) on the screen. The result will be checked by running your code.

Notes

- X>1, Y>1, and Z>=1. The value of X, Y, or Z will be at most 512.
- Pancakes in the same pile may have repeated sizes.
- Your code should be written in only **ONE** file named “**EE599_Lab4_<STUDENT-ID>.cpp**” where you should implement all the classes and the main driver function. Please replace <STUDENT-ID> with your own 10 digit USC-ID, the name of the file would be something like: EE599_Lab4_1234567890.cpp
-

An “input.txt” and correct “output_golden.txt” files are included in the starter repo for your reference. You can use them to test your code, but there will be other inputs for grading your code.

Summary

- You are **NOT** allowed to use STL in this lab. You can only use standard libraries, such as iostream, fstream, string, etc. For other unsure libraries, please ask in discussion forum first before you use it.

- You have to follow the exact formats and syntax for generating the required results. Even adding an extra white line in the beginning of your output file or an extra space, etc., will result in losing the whole score of this lab.

- There should be only one code file “**EE599_Lab4_<STUDENT-ID>.cpp**”, where you implement all the classes and the main driver function. Compiling and executing it will generate one output file.

- You can use shell script to check whether your result is same as the golden answer, which is similar to what you do in Lab1.

- You should also write a README file. You can write a .txt file or .md file. A README file typically contains information of author’s name and email, a brief program summary, references and instructions. Besides, include any information that you think the course staff, especially the grader should know while grading your assignment: references, any non-working part, any concerns, etc.

    1. Any non-working part should be clearly stated.

    2. The citations should be done carefully and clearly, e.g.: “to write my code, lines 27 to 65, I used the Diijkstra's shortest path algorithm c++ code from the following website: www.SampleWebsite.com/ ...”

    3. The Readme file content of labs cannot be hand-written and should only be typed.

Submission

Submit your code files and a README file using GitHub. Please replace <STUDENT-ID> with your own 10 digit USC-ID, the name of the file would be something like: EE599_Lab4_1234567890.cpp

Extra Credits for lab4

1. Search and flip optimization. (Max of 10) Use minimum steps as you can.

2 Fibonacci Memory optimization. (Max of 5)

3 Other optimization you think is necessary. (Max of 10)

For extra credit, write your answer with details in a new file "Lab4_Report_USCID.pdf"

FAQs

1. Virtual Function (VF)

Only 12 students did virtual function correctly and I gave them 10 extra points. You will not lose any points because of virtual function. For virtual function, you need to define a base class pointer then points to derived class.

```
e.g. PancakePile **piles;
 piles = new PancakePile*[x](or FibPancakePile);
  for (int i = 0; i < x; i++) {
   piles[i] = new FibPancakePile[y];
  }
```

Two common problems:

a. Some of you define both Pan object and FibPan objects and use FibPan for operation of Fib sort and flip.

b. Define Fibpancake directly which should be PancakePile type object.

To see virtual function, you need to define base class type and points to derived class (Usually it uses function in base class but with virtual function it will use the same name function in derived class). So this Lab is a practice of using pointers on multi-dimensional arrays. Some of you may think virtual function is useless that we can just change function name to avoid failures. It's true but pure virtual function (virtual f() = 0;) is very useful. You can search online for details.

2. Data structure problems

a. Please do as we required in lab description. i.e. "Class ***MPancakePile*** should include X*Y objects of class PancakePile, but you need to **use**

pointers to point each object to the derived class FibPancakePile. It should include two functions: a) *sort_interior()*; b) *sort_exterior().* " Please implement the objects of Pancake and two function in class Mpan instead of define some int * in main, do all things in main with two blank function or something else.

b.

Pancake, PancakePile and Mpan are three individual classes, they are not friend class or one derived from others. Only FibPancakepile is derived from PancakePile.

Data structure problems will lose a little point.

3. Extra Parts (EP)

EP1 has two part (5 for search & 5 for flip optimization), EP2 (is actually the optimization for fib_dp, three variables are enough for it instead an array). EP3 you need to provide novel idea (also EP1) to get points. Our grading policy is very strict on EP, you can only get 0 or 5 for each part.

4. Try to free memory if you apply dynamic array. Double check your code on Ubuntu before you submit it. Few of you still print lots of numbers in command line, this time you will lose 5 points because of this since it's very difficult to find your mission code.