Due: See website for due date.

What to submit: See website.

The theme of this exercise is automatic memory management, leak detection and virtual memory.

1. Understanding valgrind's leak checker

Valgrind is a tool that can aid in finding memory leaks in C programs. To that end, it performs an analysis similar to the "mark" phase of a traditional mark-and-sweep garbage collector right before a program exits and identifies still reachable objects and leaks.

For leaks, it uses the definition prevalent in C: objects that have been allocated but not yet freed, and there is no possible way for a legal program to access them in the future.

Read Section 4.2.8 Memory leak detection in the Valgrind Manual [URL] and construct a C program `leak.c` that, when run with

```
valgrind --leak-check=full --show-leak-kinds=all ./leak
```

produces the following output:

```
==1898512== Memcheck, a memory error detector
==1898512== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==1898512== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==1898512== Command: ./leak
==1898512==
==1898512==
==1898512== HEAP SUMMARY:
==1898512==     in use at exit: 1,008 bytes in 5 blocks
==1898512==   total heap usage: 5 allocs, 0 frees, 1,008 bytes allocated
==1898512==
==1898512== 100 bytes in 1 blocks are possibly lost in loss record 1 of 5
==1898512==    at 0x4C37135: malloc (vg_replace_malloc.c:381)
==1898512==    by 0x4005DA: main (leak.c:17)
==1898512==
==1898512== 200 bytes in 1 blocks are definitely lost in loss record 2 of 5
==1898512==    at 0x4C37135: malloc (vg_replace_malloc.c:381)
==1898512==    by 0x4005EF: main (leak.c:18)
==1898512==
==1898512== 300 bytes in 1 blocks are indirectly lost in loss record 3 of 5
==1898512==    at 0x4C37135: malloc (vg_replace_malloc.c:381)
==1898512==    by 0x4005C6: main (leak.c:14)
==1898512==
==1898512== 308 (8 direct, 300 indirect) bytes in 1 blocks are definitely
                                            lost in loss record 4 of 5
==1898512==    at 0x4C37135: malloc (vg_replace_malloc.c:381)
==1898512==    by 0x4005B8: main (leak.c:13)
==1898512==
==1898512== 400 bytes in 1 blocks are still reachable in loss record 5 of 5
```

```
==1898512==    at 0x4C37135: malloc (vg_replace_malloc.c:381)
==1898512==    by 0x4005A7: main (leak.c:11)
==1898512==
==1898512== LEAK SUMMARY:
==1898512==    definitely lost: 208 bytes in 2 blocks
==1898512==    indirectly lost: 300 bytes in 1 blocks
==1898512==      possibly lost: 100 bytes in 1 blocks
==1898512==    still reachable: 400 bytes in 1 blocks
==1898512==         suppressed: 0 bytes in 0 blocks
==1898512==
==1898512== For lists of detected and suppressed errors, rerun with: -s
==1898512== ERROR SUMMARY: 3 errors from 3 contexts (suppressed: 0 from 0)
```

(the line numbers need not match, but the LEAK summary should.)

2. Reverse Engineering A Memory Leak

In this part of the exercise, you will be given a post-mortem dump of a JVM's heap that was obtained when running a program with a memory leak. The dump was produced at the point in time when the program ran out of memory because its live heap size exceeded the maximum, which can be accomplished as shown in this log:

```
$ java -XX:+HeapDumpOnOutOfMemoryError -Xmx64m OOM
java.lang.OutOfMemoryError: Java heap space
Dumping heap to java_pid1895653.hprof ...
Heap dump file created [69532530 bytes in 0.044 secs]
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
        at OOM$Blurb.<init>(OOM.java:6)
        at OOM.main(OOM.java:16)
```

Your task is to examine the heap dump (oom.hprof) and reverse engineer the leaky program.

To that end, you must install the Eclipse Memory Analyzer on your computer. It can be downloaded from this URL. Open the heap dump.

Requirements

- Your program must run out of memory when run as shown above. You should double-check that the created heap dump matches the provided dump, where "matches" is defined as follows.
- The structure of the reachability graph of the subcomponent with the largest retained size should be similar in your heap dump as in the provided heap dump. (Other information such as the content of arrays may differ.)

- You will need to write one or more classes and write code that allocates these objects and creates references between them. You should choose the **same field and class names** in your program as in the heap dump, and no extra ones (we will check this). Think of field names as edge labels in the reachability graph.
- You should investigate whether classes from Java's standard library are involved in the leak.

Hints

- The program that was used to create the heap dump is 19 lines long (without comments, and including the main function), though your line numbers may differ.
- Static inner classes are separated with a dollar sign $. For instance, `A$B` is the name of a static inner class called `B` nested in `A`. (Your solution should use the same class names as in the heap dump.)
- Start with the "Leak Suspects" report, then look in Details. Use the "List Objects ... with outgoing references" feature to find a visualization of the objects that were part of the heap when the program ran out of memory.
- The "dominator tree" option can also give you insight into the structure of the object graph. Zoom in on the objects that have the largest "Retained Heap" quantity.
- Use the Java Tutor website to write small test programs and trace how the reachability graph changes over time.
- Do not forget the `-Xmx64m` switch when running your program, or else your program may run for several minutes before running out of memory, even if implemented correctly.
- Do not access the `oom.hprof` file through a remote file system path such as a mapped Google drive or similar. Students in the past have reported runtime errors in Eclipse MAT when trying to do that. Instead, copy it to your local computer's file system first as a binary file. The SHA256 sum of `oom.hprof` is

  `38c344f51fd7ca0bab767506c35fb417661689199a63ae2badd4ba8621b63849`

3. Using mmap to list the entries in a ZIP file

Write a short program `zipdir` that displays the list of entries inside a zip file whose name is passed to the program as its first argument. A sample use would be:

```
$ ./zipdir heap.zip
heap1.dot
heap1.in
```

```
heap1.out
heap1.png
heap2.dot
heap2.in
heap2.out
heap2.png
heap3.dot
heap3.in
heap3.out
heap3.png
```

Your program should use only the `open(2)`, `stat(2)`, and `mmap(2)` system calls (plus any system calls needed to output the result, such as `write(1)`.

Do not use `read(2)` (or higher-level functions such as `fread(3)`, etc. that call `read()` internally).

The ZIP file format is described, among others, on the WikiPedia page https://en.wikipedia.org/wiki/ZIP_(file_format)

Use the following algorithm:

- use `stat(2)` to determine the length of the file.
- open the file with `open(2)` in read-only mode.
- use `mmap(2)` to map the entire file into memory in a read-only way.
- scan from the back of the file until you find the beginning marker of the End of Central Directory Record (EOCD).
- extract the number of central directory records on this disk and the start offset of the central directory.
- Then, starting from the start offset of the central directory, examine each central directory file header and output the filename contained in it.
- Skip forward to the next central directory record by advancing $46 + m + n + k$ bytes where n is the length of the filename, m is the extra field length, and k is the file comment length contained in each central file directory header.

Simplifying assumptions/hints:

- All multibyte integers in a ZIP file are stored in little-endian order, and — for the purposes of this exercise — you may assume that the host byte order of the machine on which your program runs is little endian as well.
- You may use pointer arithmetic on `void *` pointers, which advances by 1 byte. To access 16-bit or 32-bit values, use `uint16_t *` and `uint32_t *`, respectively under the assumption of little endian host byte order.

- If the given file is not a well-formed ZIP archive then the behavior of your program can be undefined.