Lab 4 Introduction to Valgrind CIS2500 Winter 2021

Due Friday March 26th 11:59pm

This lab is about a command-line tool tool called Valgrind. It is a very useful tool to detect memory errors and to detect memory leaks. **You are required to do this lab on the school server (linux.socs.uoguelph.ca)**. To run valgrind, first you must compile your code and create an executable. Now you can run valgrind on the executable. For example, if a.out is the name of my executable, I will run vagrind as:

$ valgrind ./a.out

You may also run it using flags such as -g and others. To learn more on the syntax of valgrind, you can run the man command:

$ man valgrind

In general, it reports a heap summary, a leak summary and an error summary.

Part I (ungraded part): Read more on valgrind and understand the leak summary, especially how it calculates the "definitely lost" and "indirectly lost" bytes. Note that some memory leak occurs when we use standard libraries such as stdio.h.

1. Example report on heap, leak and error report:
For example, it reports the following for the "Hello world" program given in the textbox on the right-hand side:

```
chaturvr@ginny:~/2500/for_test$ valgrind ./a.out
==27826== Memcheck, a memory error detector
==27826== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==27826== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==27826== Command: ./a.out
==27826==
Hello world
==27826==
==27826== HEAP SUMMARY:
==27826==     in use at exit: 0 bytes in 0 blocks
==27826==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==27826==
==27826== All heap blocks were freed -- no leaks are possible
==27826==
==27826== For counts of detected and suppressed errors, rerun with: -v
==27826== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

```c
#include <stdio.h>

int main ( )
{
  printf ("Hello world\n");
  return 0;
}
```

2. Example on initialization error:
Valgrind is also useful in finding initialization errors. For example, it reports the following for the program given in the textbox on the right-hand side. As you can see line 8 is attempting to print arr [1] that has not been initialized – and valgrind reports this error, including the line number.

```
chaturvr@ginny:~/2500/for_test$ valgrind ./a.out
==3804== Memcheck, a memory error detector
==3804== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==3804== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==3804== Command: ./a.out
==3804==
==3804== Conditional jump or move depends on uninitialised value(s)
==3804==    at 0x48B1029: vfprintf (vfprintf.c:1637)
==3804==    by 0x48B8605: printf (printf.c:33)
==3804==    by 0x109187: main (test2.c:8)
==3804==
==3804== Use of uninitialised value of size 8
==3804==    at 0x48ACD1E: _itoa_word (_itoa.c:179)
==3804==    by 0x48B05F3: vfprintf (vfprintf.c:1637)
==3804==    by 0x48B8605: printf (printf.c:33)
==3804==    by 0x109187: main (test2.c:8)
==3804==
==3804== Conditional jump or move depends on uninitialised value(s)
==3804==    at 0x48ACD29: _itoa_word (_itoa.c:179)
==3804==    by 0x48B05F3: vfprintf (vfprintf.c:1637)
==3804==    by 0x48B8605: printf (printf.c:33)
==3804==    by 0x109187: main (test2.c:8)
==3804==
==3804== Conditional jump or move depends on uninitialised value(s)
==3804==    at 0x48B1213: vfprintf (vfprintf.c:1637)
==3804==    by 0x48B8605: printf (printf.c:33)
==3804==    by 0x109187: main (test2.c:8)
==3804==
==3804== Conditional jump or move depends on uninitialised value(s)
==3804==    at 0x48B075D: vfprintf (vfprintf.c:1637)
==3804==    by 0x48B8605: printf (printf.c:33)
==3804==    by 0x109187: main (test2.c:8)
==3804==
arr [1] = 0==3804==
==3804== HEAP SUMMARY:
==3804==     in use at exit: 0 bytes in 0 blocks
==3804==   total heap usage: 2 allocs, 2 frees, 1,064 bytes allocated
==3804==
==3804== All heap blocks were freed -- no leaks are possible
==3804==
==3804== For counts of detected and suppressed errors, rerun with: -v
==3804== Use --track-origins=yes to see where uninitialised values come from
==3804== ERROR SUMMARY: 5 errors from 5 contexts (suppressed: 0 from 0)
```

```c
#include <stdio.h>
#include <stdlib.h>

int main ( )
{
    int *arr= malloc (sizeof(int)*10);

    printf("arr [1] = %d", arr[1]);

    free (arr);

    return 0;
}
```

Part II (graded part):

You are given 3 source program files. You must run each file using valgrind, understand the report it generates and fix the memory / initialization issues with it.

For each given source file, you must submit

(a) the output generated by valgrind on the given source file

To save the output to a file, use valgrind with `--log-file` flag to log the report to a file (e.g. `logFile1.txt`), instead of displaying it on standard output.

```
valgrind --log-file=logFile1.txt ./a.out
```

Note that you are allowed to use other flags with valgrind.

(b) an improved C program file such that

- "definitely lost" and "indirectly lost" bytes is zero.
- source file is free of any initialization error

Part III (Submission):

There are a total of 6 files to submit (2 for each source program file). You must create a tar file that consists of these 6 files. A tar file can be created using the following command. Here flag c is for create, v for verbose and f for filename.

The following command will compress all files in the current folder to a file called lab3.tar.

```
tar –cvf lab3.tar *
```

The following command will compress files `file1.c` and `log_file1.txt` to a file called lab3.tar.

```
tar –cvf lab3.tar file1.c log_file1.txt
```

To learn more on the tar command, man it!

At the end, submit the tar file `lab3.tar` to Gitlab.