Sarbani Bhattacharyya, A20383075, Chen En Lee, A20435695

Programming Assignment 3

PART 1 DOCUMENTATION

Part 1 consists of 2 questions. They are:
1. Write a program that allocates memory using malloc() but forgets to free it before exiting. What happens when this program runs? Can you use gdb to find any problems with it? How about valgrind (with the command: valgrind --leakcheck=yes null)?

2. Create other test cases for valgrind. Explain why you choose them and the expected results.

Response to Question 1

Valgrind categorizes leaks using these terms:

- *definitely lost:* heap-allocated memory that was never freed to which the program no longer has a pointer. Valgrind knows that you once had the pointer, but have since lost track of it. This memory is definitely orphaned.
- *indirectly lost:* heap-allocated memory that was never freed to which the only pointers to it also are lost.
- *possibly lost:* heap-allocated memory that was never freed to which valgrind cannot be sure whether there is a pointer or not.
- *still reachable:* heap-allocated memory that was never freed to which the program still has a pointer at exit.

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 //Response to Question 1 of Part 1 of PA3
4 int
5 main(void){
6
7         int *var = (int *)malloc(sizeof(int)); //memory is allocated here
8         var[0] = 2020; //value is stored in assigned memory
9         return 0;
10 }
```

Code for memleak.c

The file *memleak.c* contains the code that we used to answer this question. In this program, memory is allocated in Line 7 using malloc() but this memory is not freed before the program exits main method in Line 9.

The program compiles without any error or warning. Thus using gdb is of no use in this case, because gdb will not be able to catch the memory leak in this program. However, valgrind is used to detect memory leaks in this program. Valgrind reports that 4 bytes of memory is

definitely lost by this program. 4 bytes is the space assigned to an integer variable. In *memleak.c*, 4 bytes are assigned to the variable *var* but since they are never freed, this causes a memory leak in the program (aka., these 4 bytes are lost). So, the results we see in valgrind match our understanding of what is happening in *memleak.c*. Therefore, this is what we expect. The results that valgrind returns when it is run on *memleak.c* is shown below:

```
sarbani@sarbaniVM:~/cs450/PA3/part1$ valgrind --leak-check=yes ./memleak
==5073== Memcheck, a memory error detector
==5073== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==5073== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==5073== Command: ./memleak
==5073==
==5073==
==5073== HEAP SUMMARY:
==5073==     in use at exit: 4 bytes in 1 blocks
==5073==   total heap usage: 1 allocs, 0 frees, 4 bytes allocated
==5073==
==5073== 4 bytes in 1 blocks are definitely lost in loss record 1 of 1
==5073==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64
-linux.so)
==5073==    by 0x10865B: main (memleak.c:7)
==5073==
==5073== LEAK SUMMARY:
==5073==    definitely lost: 4 bytes in 1 blocks
==5073==    indirectly lost: 0 bytes in 0 blocks
==5073==      possibly lost: 0 bytes in 0 blocks
==5073==    still reachable: 0 bytes in 0 blocks
==5073==         suppressed: 0 bytes in 0 blocks
==5073==
==5073== For counts of detected and suppressed errors, rerun with: -v
==5073== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
sarbani@sarbaniVM:~/cs450/PA3/part1$
```

Valgrind result for memleak.c: Valgrind correctly points out the memory wasted, i.e., 4 bytes in Line 7

Response to Question 2

Equivalence Partitioning for Testing

The following equivalence partitions are needed in order to test C programs for memory leaks.

1. **Valid operations**
    a. Correctly allocating memory using malloc() and then freeing it properly using free() (test1.c)

2. **Invalid operations**

    a. Allocating memory but not freeing (test2.c & test3.c)
    b. Read a memory location after freeing it (test4.c)
    c. Writing to memory location after freeing it (test5.c)
    d. Allocating insufficient memory and then writing to it. (test6.c)
    e. Indirectly lost memory example (test7.c)

f. Possibly lost memory example (test8.c)
g. Memory leak due to use of uninitialized values (test9.c)
h. Changing the pointer's address to NULL (test10.c)
i. Local pointer from called function used in the caller function (test11.c)
j. Changing the pointer's address to NULL, example using array (test12.c)

test1.c : Tests valid operation case 1(a): Correcting allocating and freeing memory

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

void test1(){

        int *var = (int *)malloc(sizeof(*var));//allocate memory
        var[0] = 2020; //fill with some value
        free(var); //free allocated memory
}
int
main(void){

        test1();
        return 0;
}
```

test1.c code

test1.c shows how to correctly allocate memory using malloc() and then free the memory using free(). Valgrind produces the expected output -- no errors. See below:

```
sarbani@sarbaniVM:~/cs450/PA3/part1$ valgrind --leak-check=yes ./test1
==8544== Memcheck, a memory error detector
==8544== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==8544== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==8544== Command: ./test1
==8544==
==8544==
==8544== HEAP SUMMARY:
==8544==     in use at exit: 0 bytes in 0 blocks
==8544==   total heap usage: 1 allocs, 1 frees, 4 bytes allocated
==8544==
==8544== All heap blocks were freed -- no leaks are possible
==8544==
==8544== For counts of detected and suppressed errors, rerun with: -v
==8544== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
sarbani@sarbaniVM:~/cs450/PA3/part1$
```

Valgrind output from running test1.c

test2.c : Tests invalid operation case 2(a): Allocating memory but not freeing it

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

//Allocating memory with malloc() but not freeing it before exiting

void test2()
{
    int *a = malloc(sizeof(*a));
    *a = 10;
    printf("Value of a is: %d\n", *a);
}

int main()
{
    test2();
    return 0;
}
```

test2.c code

test2.c shows how to correctly allocate memory using malloc() but does not free it. This is similar to memleak.c where memory is not freed before the program exits. Valgrind produces the expected output -- 4 bytes are definitely lost at Line 8. See below:

```
sarbani@sarbaniVM:~/cs450/PA3/part1$ valgrind --leak-check=yes ./test2
==5133== Memcheck, a memory error detector
==5133== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==5133== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==5133== Command: ./test2
==5133==
Value of a is: 10
==5133==
==5133== HEAP SUMMARY:
==5133==     in use at exit: 4 bytes in 1 blocks
==5133==   total heap usage: 2 allocs, 1 frees, 1,028 bytes allocated
==5133==
==5133== 4 bytes in 1 blocks are definitely lost in loss record 1 of 1
==5133==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64
-linux.so)
==5133==    by 0x10869B: test2 (test2.c:8)
==5133==    by 0x1086D3: main (test2.c:15)
==5133==
==5133== LEAK SUMMARY:
==5133==    definitely lost: 4 bytes in 1 blocks
==5133==    indirectly lost: 0 bytes in 0 blocks
==5133==      possibly lost: 0 bytes in 0 blocks
==5133==    still reachable: 0 bytes in 0 blocks
==5133==         suppressed: 0 bytes in 0 blocks
==5133==
==5133== For counts of detected and suppressed errors, rerun with: -v
==5133== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
sarbani@sarbaniVM:~/cs450/PA3/part1$
```

Valgrind output from running test2.c

test3.c : Tests invalid operation case 2(a): Allocating memory but not freeing it

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

//Similar concept to test1: allocate memory but not free it
//Now trying this with different types of variables

void test3()
{
        long *a = (long *)malloc(sizeof(long)); //8 bytes
        float *b = (float *)malloc(sizeof(float)); //4 bytes
        char *c = (char *)malloc(sizeof(char)); //1 byte
        double *d = (double *)malloc(sizeof(double)); //8 bytes
}

int main()
{
    test3();
    return 0;
}
```

test3.c code

test3.c is similar to *test2.c* and *test1.c. test3.c* shows valgrind output when other types of primitive data types are allocated using malloc(). This test shows how much memory is allocated to each data type(float, double, long, etc.). Since memory is not freed before the program exits, this results in memory leaks. Valgrind produces the expected output -- 21 bytes

are definitely lost in total(8 bytes from long, 8 bytes from double, 1 byte from char, 4 bytes from float). Valgrind shows the memory lost from each allocation.

```
sarbani@sarbaniVM:~/cs450/PA3/part1$ valgrind --leak-check=yes ./test3
==8123== Memcheck, a memory error detector
==8123== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==8123== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==8123== Command: ./test3
==8123==
==8123==
==8123== HEAP SUMMARY:
==8123==     in use at exit: 21 bytes in 4 blocks
==8123==   total heap usage: 4 allocs, 0 frees, 21 bytes allocated
==8123==
==8123== 1 bytes in 1 blocks are definitely lost in loss record 1 of 4
==8123==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==8123==    by 0x108677: test3 (test3.c:11)
==8123==    by 0x10869A: main (test3.c:17)
==8123==
==8123== 4 bytes in 1 blocks are definitely lost in loss record 2 of 4
==8123==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==8123==    by 0x108669: test3 (test3.c:10)
==8123==    by 0x10869A: main (test3.c:17)
==8123==
==8123== 8 bytes in 1 blocks are definitely lost in loss record 3 of 4
==8123==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==8123==    by 0x10865B: test3 (test3.c:9)
==8123==    by 0x10869A: main (test3.c:17)
==8123==
==8123== 8 bytes in 1 blocks are definitely lost in loss record 4 of 4
==8123==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==8123==    by 0x108685: test3 (test3.c:12)
==8123==    by 0x10869A: main (test3.c:17)
==8123==
==8123== LEAK SUMMARY:
==8123==    definitely lost: 21 bytes in 4 blocks
==8123==    indirectly lost: 0 bytes in 0 blocks
==8123==      possibly lost: 0 bytes in 0 blocks
==8123==    still reachable: 0 bytes in 0 blocks
==8123==         suppressed: 0 bytes in 0 blocks
==8123==
==8123== For counts of detected and suppressed errors, rerun with: -v
==8123== ERROR SUMMARY: 4 errors from 4 contexts (suppressed: 0 from 0)
sarbani@sarbaniVM:~/cs450/PA3/part1$
```

Valgrind output from running test3.c

test4.c : Tests invalid operation case 2(b): Read a memory location after freeing it

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  //Invalid read operation: Trying to read value at memory after freeing it
5  void
6  test4()
7  {
8          int *d = malloc(sizeof(*d)); //allocating 4 byte of memory in heap
9          d[0] = 2020;
10         free(d);
11         printf("Value stored in d is: %d\n", d[0]); //illegal access because d has already been fr
   eed
12 }
13
14 int main()
15 {
16     test4();
17     return 0;
18 }
```

test4.c code

test4.c allocates memory, uses it, frees it but tries to access this memory after it has already been freed. In valgrind, this returns *Invalid read of size 4* error, since we tried to read 4 bytes of memory in Line 11 that has already been freed in Line 10. See below for valgrind output:

Sarbani Bhattacharyya, A20383075, Chen En Lee, A20435695

```
sarbani@sarbaniVM:~/cs450/PA3/part1$ valgrind --leak-check=yes ./test4
==8184== Memcheck, a memory error detector
==8184== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==8184== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==8184== Command: ./test4
==8184==
==8184== Invalid read of size 4
==8184==    at 0x10870A: test4 (test4.c:11)
==8184==    by 0x10872F: main (test4.c:16)
==8184==  Address 0x522d040 is 0 bytes inside a block of size 4 free'd
==8184==    at 0x4C30D3B: free (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==8184==    by 0x108705: test4 (test4.c:10)
==8184==    by 0x10872F: main (test4.c:16)
==8184==  Block was alloc'd at
==8184==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==8184==    by 0x1086EB: test4 (test4.c:8)
==8184==    by 0x10872F: main (test4.c:16)
==8184==
Value stored in d is: 2020
==8184==
==8184== HEAP SUMMARY:
==8184==     in use at exit: 0 bytes in 0 blocks
==8184==   total heap usage: 2 allocs, 2 frees, 1,028 bytes allocated
==8184==
==8184== All heap blocks were freed -- no leaks are possible
==8184==
==8184== For counts of detected and suppressed errors, rerun with: -v
==8184== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
sarbani@sarbaniVM:~/cs450/PA3/part1$
```

Valgrind output from running test4.c

test5.c : Tests invalid operation case 2(c): Writing to memory after freeing it

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 //Invalid write operation: Writing to memory after freeing it
5
6 void test5()
7 {
8         int *val = malloc(sizeof(*val));//allocates memory that's the size of whatever the type of
    variable val is(in this case int)
9         val[0] = 2019;
10        free(val);
11        val[0] = 2020;
12 }
13
14 int main()
15 {
16     test5();
17     return 0;
18 }
```

test5.c code

test5.c tries to write to memory that has already been freed. In valgrind, this returns *Invalid write of size 4* error, since we tried to write to 4 bytes of memory in Line 11 that has already been freed in Line 10. See below for valgrind output:

```
sarbani@sarbaniVM:~/cs450/PA3/part1$ valgrind --leak-check=yes ./test5
==8193== Memcheck, a memory error detector
==8193== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==8193== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==8193== Command: ./test5
==8193==
==8193== Invalid write of size 4
==8193==    at 0x1086BA: test5 (test5.c:11)
==8193==    by 0x1086D0: main (test5.c:16)
==8193==  Address 0x522d040 is 0 bytes inside a block of size 4 free'd
==8193==    at 0x4C30D3B: free (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==8193==    by 0x1086B5: test5 (test5.c:10)
==8193==    by 0x1086D0: main (test5.c:16)
==8193==  Block was alloc'd at
==8193==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==8193==    by 0x10869B: test5 (test5.c:8)
==8193==    by 0x1086D0: main (test5.c:16)
==8193==
==8193==
==8193== HEAP SUMMARY:
==8193==     in use at exit: 0 bytes in 0 blocks
==8193==   total heap usage: 1 allocs, 1 frees, 4 bytes allocated
==8193==
==8193== All heap blocks were freed -- no leaks are possible
==8193==
==8193== For counts of detected and suppressed errors, rerun with: -v
==8193== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
sarbani@sarbaniVM:~/cs450/PA3/part1$
```

Valgrind output from running test5.c

test6.c : Tests invalid operation case 2(d): Allocating insufficient memory and then writing to it.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 //Allocate insufficient memory and then try to write into it
5
6 void test6()
7 {
8         int *val = malloc(1);//insufficient because an int variable occupies 4 bytes
9         val[0] = 2020;
10        free(val);
11 }
12
13 int main()
14 {
15     test6();
16     return 0;
17 }
```

test6.c code

test6.c allocates memory but it is not sufficient. In Line 8, 1 byte of memory is allocated but tries to store an integer(that takes up 4 bytes of memory). Writing invalid data to allocated memory should cause a write error. This is verified when valgrind returns an *Invalid write of size 4* error. See below:

```
sarbani@sarbaniVM:~/cs450/PA3/part1$ valgrind --leak-check=yes ./test6
==8204== Memcheck, a memory error detector
==8204== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==8204== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==8204== Command: ./test6
==8204==
==8204== Invalid write of size 4
==8204==    at 0x1086A4: test6 (test6.c:9)
==8204==    by 0x1086C6: main (test6.c:15)
==8204==  Address 0x522d040 is 0 bytes inside a block of size 1 alloc'd
==8204==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64
-linux.so)
==8204==    by 0x10869B: test6 (test6.c:8)
==8204==    by 0x1086C6: main (test6.c:15)
==8204==
==8204==
==8204== HEAP SUMMARY:
==8204==     in use at exit: 0 bytes in 0 blocks
==8204==   total heap usage: 1 allocs, 1 frees, 1 bytes allocated
==8204==
==8204== All heap blocks were freed -- no leaks are possible
==8204==
==8204== For counts of detected and suppressed errors, rerun with: -v
==8204== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
sarbani@sarbaniVM:~/cs450/PA3/part1$
```

Valgrind output from running test6.c

test7.c : Tests invalid operation case 2(e): Indirectly lost memory example

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 //Indirect memory leak example.
 5 //The struct represents a binary tree.
 6
 7 int main(void)
 8 {
 9         typedef struct
10         {
11           int val;
12           struct Tree *left;
13           struct Tree *right;
14         } Tree;
15
16         Tree* t = (Tree*)malloc(sizeof(Tree));
17         t->left = malloc(sizeof(Tree));
18         t->right = malloc(sizeof(Tree));
19
20         ((Tree*)(t->left))->left = malloc(sizeof(Tree));
21
22         return 0;
23 }
```

test7.c code

In *test7.c,* we define a struct named Tree, which is a structure with dynamically allocated components. This struct(refer to Lines 9 - 14) represents a Binary Tree node, and therefore has a left node and a right node, and a place to store the value of the node(in integer variable *val*). We allocate memory to the parent and children nodes, but do not free the parent node(t). In doing so, the parent node, which has sole access to the children nodes, becomes unreachable and therefore lost. As a result, the children nodes of the parent become indirectly lost (72 bytes in 3 blocks). 24 bytes are definitely lost because we do not free any allocated memory blocks in this program. See below:

```
sarbani@sarbaniVM:~/cs450/PA3/part1$ valgrind --leak-check=yes ./test7
==8220== Memcheck, a memory error detector
==8220== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==8220== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==8220== Command: ./test7
==8220==
==8220==
==8220== HEAP SUMMARY:
==8220==     in use at exit: 96 bytes in 4 blocks
==8220==   total heap usage: 4 allocs, 0 frees, 96 bytes allocated
==8220==
==8220== 96 (24 direct, 72 indirect) bytes in 1 blocks are definitely lost in
oss record 4 of 4
==8220==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd6
-linux.so)
==8220==    by 0x10865C: main (test7.c:16)
==8220==
==8220== LEAK SUMMARY:
==8220==    definitely lost: 24 bytes in 1 blocks
==8220==    indirectly lost: 72 bytes in 3 blocks
==8220==      possibly lost: 0 bytes in 0 blocks
==8220==    still reachable: 0 bytes in 0 blocks
==8220==         suppressed: 0 bytes in 0 blocks
==8220==
==8220== For counts of detected and suppressed errors, rerun with: -v
==8220== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
sarbani@sarbaniVM:~/cs450/PA3/part1$
```

Valgrind output from running test7.c

test8.c : Tests invalid operation case 2(f): Possibly lost memory example

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 //Possible memory leak example.
 5
 6 void test8()
 7 {
 8         char *s = "Hello World";
 9         char *d = strdup(s);
10         d+=1;
11         exit(1);
12         d-=1;
13         free(d);
14
15 }
16
17 int main()
18 {
19     test8();
20     return 0;
21 }
```

test8.c code

test8.c shows how valgrind can detect possibly lost memory. Valgrind reports possibly lost memory when an array of characters (like a string) is allocated (see Line 8) and pointer arithmetic is used(see Line 10) to point into the middle of this block of characters. Then, the program exits before the free statement (see Line 13) leaving the pointer pointing to the middle of the allocated block of memory. Valgrind result is displayed below:

```
sarbani@sarbaniVM:~/cs450/PA3/part1$ valgrind --leak-check=yes ./test8
==8231== Memcheck, a memory error detector
==8231== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==8231== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==8231== Command: ./test8
==8231==
==8231==
==8231== HEAP SUMMARY:
==8231==     in use at exit: 12 bytes in 1 blocks
==8231==   total heap usage: 1 allocs, 0 frees, 12 bytes allocated
==8231==
==8231== 12 bytes in 1 blocks are possibly lost in loss record 1 of 1
==8231==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64
-linux.so)
==8231==    by 0x4ED99B9: strdup (strdup.c:42)
==8231==    by 0x1086A8: test8 (test8.c:9)
==8231==    by 0x1086C9: main (test8.c:19)
==8231==
==8231== LEAK SUMMARY:
==8231==    definitely lost: 0 bytes in 0 blocks
==8231==    indirectly lost: 0 bytes in 0 blocks
==8231==      possibly lost: 12 bytes in 1 blocks
==8231==    still reachable: 0 bytes in 0 blocks
==8231==         suppressed: 0 bytes in 0 blocks
==8231==
==8231== For counts of detected and suppressed errors, rerun with: -v
==8231== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
sarbani@sarbaniVM:~/cs450/PA3/part1$
```

Valgrind output from running test8.c

test9.c : Tests invalid operation case 2(g): Use of uninitialized values

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

//Memory check demonstrating error due to uninitialized value

void test9()
{
        int i;
        printf("i : %d\n", i);
}

int main()
{
    test9();
    return 0;
}
```

test9.c code

Line 9 in test9.c attempts to read and display an uninitialized value to console. Valgrind raises an error when we try to print the undefined value of variable i. See below:

```
==1594== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==1594== Command: ./test9
==1594==
==1594== Conditional jump or move depends on uninitialised value(s)
==1594==    at 0x4E988DA: vfprintf (vfprintf.c:1642)
==1594==    by 0x4EA0F25: printf (printf.c:33)
==1594==    by 0x108667: test9 (test9.c:9)
==1594==    by 0x108678: main (test9.c:14)
==1594==
==1594== Use of uninitialised value of size 8
==1594==    at 0x4E9486B: _itoa_word (_itoa.c:179)
==1594==    by 0x4E97F0D: vfprintf (vfprintf.c:1642)
==1594==    by 0x4EA0F25: printf (printf.c:33)
==1594==    by 0x108667: test9 (test9.c:9)
==1594==    by 0x108678: main (test9.c:14)
==1594==
==1594== Conditional jump or move depends on uninitialised value(s)
==1594==    at 0x4E94875: _itoa_word (_itoa.c:179)
==1594==    by 0x4E97F0D: vfprintf (vfprintf.c:1642)
==1594==    by 0x4EA0F25: printf (printf.c:33)
==1594==    by 0x108667: test9 (test9.c:9)
==1594==    by 0x108678: main (test9.c:14)
==1594==
==1594== Conditional jump or move depends on uninitialised value(s)
==1594==    at 0x4E98014: vfprintf (vfprintf.c:1642)
==1594==    by 0x4EA0F25: printf (printf.c:33)
==1594==    by 0x108667: test9 (test9.c:9)
==1594==    by 0x108678: main (test9.c:14)
==1594==
==1594== Conditional jump or move depends on uninitialised value(s)
==1594==    at 0x4E98B4C: vfprintf (vfprintf.c:1642)
==1594==    by 0x4EA0F25: printf (printf.c:33)
==1594==    by 0x108667: test9 (test9.c:9)
==1594==    by 0x108678: main (test9.c:14)
==1594==
1 : 0
==1594==
==1594== HEAP SUMMARY:
==1594==     in use at exit: 0 bytes in 0 blocks
==1594==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==1594==
==1594== All heap blocks were freed -- no leaks are possible
==1594==
==1594== For counts of detected and suppressed errors, rerun with: -v
==1594== Use --track-origins=yes to see where uninitialised values come from
==1594== ERROR SUMMARY: 5 errors from 5 contexts (suppressed: 0 from 0)
sarbani@sarbaniVM:~/cs450/PA3/part1$
```

Valgrind output from running test9.c

test10.c : Tests invalid operation case 2(h): Changing malloc returned address to NULL

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

//Allocate memory correctly using malloc(). Requested memory is assigned
//to specific pointer(*val in this case)
//But after that changing the assigned address of *val to NULL
//This loses track of the assigned memory as the pointer is redirected.
//so that now the assigned pointer points to NULL

void test10()
{
        int *val = malloc(sizeof(*val));
        val[0] = 2020;
        val = NULL; //2020 exists in user address space but no one can reac
h it now since the pointer to that address is lost.
}

int main()
{
    test10();
    return 0;
}
```

test10.c code

In *test10.c*, malloc is used to assign 4 bytes of memory to pointer *val. val is set to point to an integer value 2020. Then, val is set to point to NULL. By doing this, the malloc() allocated block of memory becomes unreachable. This causes a memory leak, detected by Valgrind.See below:

Sarbani Bhattacharyya, A20383075, Chen En Lee, A20435695

```
sarbani@sarbaniVM:~/cs450/PA3/part1$ valgrind --leak-check=yes ./test10
==1621== Memcheck, a memory error detector
==1621== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==1621== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==1621== Command: ./test10
==1621==
==1621==
==1621== HEAP SUMMARY:
==1621==     in use at exit: 4 bytes in 1 blocks
==1621==   total heap usage: 1 allocs, 0 frees, 4 bytes allocated
==1621==
==1621== 4 bytes in 1 blocks are definitely lost in loss record 1 of 1
==1621==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64
-linux.so)
==1621==    by 0x10865B: test10 (test10.c:12)
==1621==    by 0x108682: main (test10.c:19)
==1621==
==1621== LEAK SUMMARY:
==1621==    definitely lost: 4 bytes in 1 blocks
==1621==    indirectly lost: 0 bytes in 0 blocks
==1621==      possibly lost: 0 bytes in 0 blocks
==1621==    still reachable: 0 bytes in 0 blocks
==1621==         suppressed: 0 bytes in 0 blocks
==1621==
==1621== For counts of detected and suppressed errors, rerun with: -v
==1621== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
sarbani@sarbaniVM:~/cs450/PA3/part1$
```

Valgrind output from running test10.c

test11.c : Tests invalid operation case 2(i): Local pointer obtained from called function used in the caller function

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

//

char *justanotherfunc(void)
{
        char string[20] = "Year 2020";
        char *dup = string;
        return dup;

}
void test11()
{
        char *input = justanotherfunc();
        printf("Value returned from JustAnotherFunc(): %s", input);
}

int main()
{
    test11();
    return 0;
}
```

test11.c code

T*est11.c* checks to see if Valgrind will detect a local variable from a called function(here: justanotherfunc()) when it is used in the caller function(here: test11()). Valgrind is able to detect this error, and states that 4 bytes are definitely lost. See below Valgrind result:

```
sarbani@sarbaniVM:~/cs450/PA3/part1$ valgrind --leak-check=yes ./test11
==1635== Memcheck, a memory error detector
==1635== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==1635== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==1635== Command: ./test11
==1635==
==1635== Conditional jump or move depends on uninitialised value(s)
==1635==    at 0x4C32D39: __strlen_sse2 (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==1635==    by 0x4E994D2: vfprintf (vfprintf.c:1643)
==1635==    by 0x4EA0F25: printf (printf.c:33)
==1635==    by 0x108729: test11 (test11.c:16)
==1635==    by 0x10873A: main (test11.c:21)
==1635==
==1635== Conditional jump or move depends on uninitialised value(s)
==1635==    at 0x4C32D48: __strlen_sse2 (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==1635==    by 0x4E994D2: vfprintf (vfprintf.c:1643)
==1635==    by 0x4EA0F25: printf (printf.c:33)
==1635==    by 0x108729: test11 (test11.c:16)
==1635==    by 0x10873A: main (test11.c:21)
==1635==
==1635== Conditional jump or move depends on uninitialised value(s)
==1635==    at 0x4EC7A65: _IO_file_xsputn@@GLIBC_2.2.5 (fileops.c:1241)
==1635==    by 0x4E98FEA: vfprintf (vfprintf.c:1643)
==1635==    by 0x4EA0F25: printf (printf.c:33)
==1635==    by 0x108729: test11 (test11.c:16)
==1635==    by 0x10873A: main (test11.c:21)
==1635==
==1635== Conditional jump or move depends on uninitialised value(s)
==1635==    at 0x4EC7A77: _IO_file_xsputn@@GLIBC_2.2.5 (fileops.c:1241)
==1635==    by 0x4E98FEA: vfprintf (vfprintf.c:1643)
==1635==    by 0x4EA0F25: printf (printf.c:33)
==1635==    by 0x108729: test11 (test11.c:16)
==1635==    by 0x10873A: main (test11.c:21)
==1635== Syscall param write(buf) points to uninitialised byte(s)
==1635==    at 0x4F4C154: write (write.c:27)
==1635==    by 0x4EC71BC: _IO_file_write@@GLIBC_2.2.5 (fileops.c:1203)
==1635==    by 0x4EC8F50: new_do_write (fileops.c:457)
==1635==    by 0x4EC8F50: _IO_do_write@@GLIBC_2.2.5 (fileops.c:433)
==1635==    by 0x4ECB201: _IO_flush_all_lockp (genops.c:769)
==1635==    by 0x4ECB404: _IO_cleanup (genops.c:921)
==1635==    by 0x4E7F101: __run_exit_handlers (exit.c:130)
==1635==    by 0x4E7F139: exit (exit.c:139)
==1635==    by 0x4E5DB9D: (below main) (libc-start.c:344)
==1635==  Address 0x522d067 is 39 bytes inside a block of size 1,024 alloc'd
==1635==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==1635==    by 0x4EBA18B: _IO_file_doallocate (filedoalloc.c:101)
==1635==    by 0x4ECA378: _IO_doallocbuf (genops.c:365)
==1635==    by 0x4EC9497: _IO_file_overflow@@GLIBC_2.2.5 (fileops.c:759)
==1635==    by 0x4EC79EC: _IO_file_xsputn@@GLIBC_2.2.5 (fileops.c:1266)
==1635==    by 0x4E97533: vfprintf (vfprintf.c:1328)
==1635==    by 0x4EA0F25: printf (printf.c:33)
==1635==    by 0x108729: test11 (test11.c:16)
==1635==    by 0x10873A: main (test11.c:21)
==1635==
Value returned from JustAnotherFunc(): Year 2020==1635==
==1635== HEAP SUMMARY:
==1635==     in use at exit: 0 bytes in 0 blocks
==1635==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==1635==
==1635== All heap blocks were freed -- no leaks are possible
==1635==
==1635== For counts of detected and suppressed errors, rerun with: -v
==1635== Use --track-origins=yes to see where uninitialised values come from
==1635== ERROR SUMMARY: 20 errors from 5 contexts (suppressed: 0 from 0)
sarbani@sarbaniVM:~/cs450/PA3/part1$
```

Valgrind output from running test11.c

test12.c : Tests invalid operation case 2(j): Changing the pointer's address to NULL, example using array

Sarbani Bhattacharyya, A20383075, Chen En Lee, A20435695

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 //
5
6 void test12()
7 {
8         int i;
9         const int ARR_SIZE = 19;
10        int *arr = malloc(ARR_SIZE * sizeof(*arr));
11        for(i = 0; i<ARR_SIZE;i++)
12        {
13          if((arr = NULL)){
14                arr = malloc(ARR_SIZE*sizeof(*arr));
15                }
16
17        }
18        free(arr);
19 }
20
21 int main()
22 {
23     test12();
24     return 0;
25 }
```

test12.c code

The mistake in *test12.c* is in Line 13, inside the if-condition. Here, what the user meant to do is to check if the pointer *arr*'s value is NULL. Instead, the user sets the value of pointer *arr* to NULL, which causes a memory leak, as the program loses track of the malloc() allocated memory from Line 10. So, (19*4=76) bytes of memory are lost. Valgrind result is shown below:

```
sarbani@sarbaniVM:~/cs450/PA3/part1$ valgrind --leak-check=yes ./test12
==1649== Memcheck, a memory error detector
==1649== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==1649== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==1649== Command: ./test12
==1649==
==1649==
==1649== HEAP SUMMARY:
==1649==     in use at exit: 76 bytes in 1 blocks
==1649==   total heap usage: 1 allocs, 0 frees, 76 bytes allocated
==1649==
==1649== 76 bytes in 1 blocks are definitely lost in loss record 1 of 1
==1649==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==1649==    by 0x1086A9: test12 (test12.c:10)
==1649==    by 0x108703: main (test12.c:23)
==1649==
==1649== LEAK SUMMARY:
==1649==    definitely lost: 76 bytes in 1 blocks
==1649==    indirectly lost: 0 bytes in 0 blocks
==1649==      possibly lost: 0 bytes in 0 blocks
==1649==    still reachable: 0 bytes in 0 blocks
==1649==         suppressed: 0 bytes in 0 blocks
==1649==
==1649== For counts of detected and suppressed errors, rerun with: -v
==1649== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
sarbani@sarbaniVM:~/cs450/PA3/part1$
```

Valgrind output with test12.c