HW#3
Advanced Operating Systems, Spring 2023

MengXian,Du (CBB108047)
Department of Computer Science and Information Engineering
National Pingtung University

1. First, write a simple program called null.c that creates a pointer to an integer, sets it to NULL, and then tries to dereference it. Compile this into an executable called null. What happens when you run this program?

   Solution: Please refer to List 1 (null.c):

   Listing 1: null.c

```c
/*
First, write a simple program called null.c that creates a pointer
to an integer, sets it to NULL, and then tries to dereference it.
Compile this into an executable called null. What happens when you
run this program?
*/

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

int main()
{
    int *p = NULL;
    printf("Start\n");
    printf("The address of p is %p\n", p);
    printf("The value of p is %d\n", *p);
    printf("End\n");
    return 0;
}
```

   Its execution results are as follows:

```
Start
Segmentation fault (core dumped)
```

   As we can see the program is terminated with a segmentation fault.

   So we know that if a variable doesn't allocated memory space, it will cause a segmentation fault when it is dereferenced.

   But if we need to reference it. It's ok to reference it.

2. Next, compile this program with symbol information included (with the -g flag). Doing so let's put more information into the executable, enabling the debugger to access more useful information about variable names and the like. Run the program under the debugger by typing gdb null and then, once gdb is running, typing run. What does gdb show you?

   Solution:

   Its execution results are as follows:

```
(gdb) run
Starting program: /home/it/Desktop/Advanced_Operation_Systen/hw3/null
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Start
The address of p is (nil)
```

```

Program received signal SIGSEGV, Segmentation fault.
0x00005555555551ab in main () at null.c:16
16          printf("The␣value␣of␣p␣is␣%d\n", *p);
```

The gdb shows that the program is terminated with a segmentation fault.

3. Finally, use the valgrind tool on this program. We'll use the memcheck tool that is a part of valgrind to analyze what happens. Run this by typing in the following: valgrind –leak-check=yes null. What happens when you run this? Can you interpret the output from the tool?

   Solution:

   Its execution results are as follows:

```
==20372== Memcheck, a memory error detector
==20372== Copyright (C) 2002−2017, and GNU GPL'd,␣by␣Julian␣Seward␣et␣al.
==20372==␣Using␣Valgrind-3.18.1␣and␣LibVEX;␣rerun␣with␣-h␣for␣copyright␣info
==20372==␣Command:␣./null
==20372==
Start
The␣address␣of␣p␣is␣(nil)
==20372==␣Invalid␣read␣of␣size␣4
==20372==␣␣␣␣at␣0x1091AB:␣main␣(null.c:16)
==20372==␣␣Address␣0x0␣is␣not␣stack'd, malloc'd␣or␣(recently)␣free'd
==20372==
==20372==
==20372== Process terminating with default action of signal 11 (SIGSEGV)
==20372==  Access not within mapped region at address 0x0
==20372==    at 0x1091AB: main (null.c:16)
==20372==  If you believe this happened as a result of a stack
==20372==  overflow in your program's␣main␣thread␣(unlikely␣but
==20372==␣␣possible),␣you␣can␣try␣to␣increase␣the␣size␣of␣the
==20372==␣␣main␣thread␣stack␣using␣the␣--main-stacksize=␣flag.
==20372==␣␣The␣main␣thread␣stack␣size␣used␣in␣this␣run␣was␣8388608.
==20372==
==20372==␣HEAP␣SUMMARY:
==20372==␣␣␣␣␣in␣use␣at␣exit:␣1,024␣bytes␣in␣1␣blocks
==20372==␣␣␣total␣heap␣usage:␣1␣allocs,␣0␣frees,␣1,024␣bytes␣allocated
==20372==
==20372==␣LEAK␣SUMMARY:
==20372==␣␣␣␣definitely␣lost:␣0␣bytes␣in␣0␣blocks
==20372==␣␣␣␣indirectly␣lost:␣0␣bytes␣in␣0␣blocks
==20372==␣␣␣␣␣␣possibly␣lost:␣0␣bytes␣in␣0␣blocks
==20372==␣␣␣␣still␣reachable:␣1,024␣bytes␣in␣1␣blocks
==20372==␣␣␣␣␣␣␣␣␣suppressed:␣0␣bytes␣in␣0␣blocks
==20372==␣Reachable␣blocks␣(those␣to␣which␣a␣pointer␣was␣found)␣are␣not␣shown.
==20372==␣To␣see␣them,␣rerun␣with:␣--leak-check=full␣--show-leak-kinds=all
==20372==
==20372==␣For␣lists␣of␣detected␣and␣suppressed␣errors,␣rerun␣with:␣-s
==20372==␣ERROR␣SUMMARY:␣1␣errors␣from␣1␣contexts␣(suppressed:␣0␣from␣0)
Segmentation␣fault␣(core␣dumped)
```

   When I run 'null' with valgrind, it shows that it can't read the memory that it's size is 4.

   Address 0x0 is not stack'd, malloc'd or (recently) free'd means that the address 0x0 is not allocated and recently freed. Also the program is terminated with a SIGSEGV signal which means a process executes a invalid memory reference.

And we can see at line 23 shows that the program has a memory block in use at exit.

More see :`https://stackoverflow.com/questions/3840582/still-reachable-leak-detected-by-val`

4. Write a simple 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 (again with the –leak-check=yes flag)?

   Solution: Please refer to List 1 (q4.c):

   Listing 2: q4.c

```c
/*
First, write a simple program called null.c that creates a pointer
to an integer, sets it to NULL, and then tries to dereference it.
Compile this into an executable called null. What happens when you
run this program?
*/

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

int main()
{
    int *p = malloc(sizeof(int));
    printf("Start\n");
    printf("The address of p is %p\n", p);
    printf("The value of p is %d\n", *p);
    printf("End\n");
    return 0;
}
```

   Its execution results are as follows:

```
Start
The address of p is 0x55f62934c2a0
The value of p is 0
End
```

   The result use gdb is as follows:

```
(gdb) run
Starting program: /home/it/Desktop/Advanced_Operation_Systen/hw3/q4
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Start
The address of p is 0x5555555592a0
The value of p is 0
End
[Inferior 1 (process 22758) exited normally]
```

   The result use valgrind is as follows:

```
==22981== Memcheck, a memory error detector
==22981== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==22981== Copyright (C) 2002-2017, and GNU GPL'd,␣by␣Julian␣Seward␣et␣al.
==22981==␣Command:␣./q4
==22981==
Start
The␣address␣of␣p␣is␣0x4a96040
==22981==␣Conditional␣jump␣or␣move␣depends␣on␣uninitialised␣value(s)
==22981==␣␣␣␣at␣0x48E1B56:␣__vfprintf_internal␣(vfprintf-internal.c:1516)
==22981==␣␣␣␣by␣0x48CB81E:␣printf␣(printf.c:33)
==22981==␣␣␣␣by␣0x1091E8:␣main␣(in␣/home/it/Desktop/Advanced_Operation_Systen/hw3/q4)
==22981==
==22981==␣Use␣of␣uninitialised␣value␣of␣size␣8
==22981==␣␣␣␣at␣0x48C533B:␣_itoa_word␣(_itoa.c:177)
==22981==␣␣␣␣by␣0x48E0B3D:␣__vfprintf_internal␣(vfprintf-internal.c:1516)
==22981==␣␣␣␣by␣0x48CB81E:␣printf␣(printf.c:33)
==22981==␣␣␣␣by␣0x1091E8:␣main␣(in␣/home/it/Desktop/Advanced_Operation_Systen/hw3/q4)
==22981==
==22981==␣Conditional␣jump␣or␣move␣depends␣on␣uninitialised␣value(s)
==22981==␣␣␣␣at␣0x48C534C:␣_itoa_word␣(_itoa.c:177)
==22981==␣␣␣␣by␣0x48E0B3D:␣__vfprintf_internal␣(vfprintf-internal.c:1516)
==22981==␣␣␣␣by␣0x48CB81E:␣printf␣(printf.c:33)
==22981==␣␣␣␣by␣0x1091E8:␣main␣(in␣/home/it/Desktop/Advanced_Operation_Systen/hw3/q4)
==22981==
==22981==␣Conditional␣jump␣or␣move␣depends␣on␣uninitialised␣value(s)
==22981==␣␣␣␣at␣0x48E1643:␣__vfprintf_internal␣(vfprintf-internal.c:1516)
==22981==␣␣␣␣by␣0x48CB81E:␣printf␣(printf.c:33)
==22981==␣␣␣␣by␣0x1091E8:␣main␣(in␣/home/it/Desktop/Advanced_Operation_Systen/hw3/q4)
==22981==
==22981==␣Conditional␣jump␣or␣move␣depends␣on␣uninitialised␣value(s)
==22981==␣␣␣␣at␣0x48E0C85:␣__vfprintf_internal␣(vfprintf-internal.c:1516)
==22981==␣␣␣␣by␣0x48CB81E:␣printf␣(printf.c:33)
==22981==␣␣␣␣by␣0x1091E8:␣main␣(in␣/home/it/Desktop/Advanced_Operation_Systen/hw3/q4)
==22981==
The␣value␣of␣p␣is␣0
End
==22981==
==22981==␣HEAP␣SUMMARY:
==22981==␣␣␣␣␣in␣use␣at␣exit:␣4␣bytes␣in␣1␣blocks
==22981==␣␣␣total␣heap␣usage:␣2␣allocs,␣1␣frees,␣1,028␣bytes␣allocated
==22981==
==22981==␣4␣bytes␣in␣1␣blocks␣are␣definitely␣lost␣in␣loss␣record␣1␣of␣1
==22981==␣␣␣␣at␣0x4848899:␣malloc␣(in␣/usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==22981==␣␣␣␣by␣0x10919E:␣main␣(in␣/home/it/Desktop/Advanced_Operation_Systen/hw3/q4)
==22981==
==22981==␣LEAK␣SUMMARY:
==22981==␣␣␣␣definitely␣lost:␣4␣bytes␣in␣1␣blocks
==22981==␣␣␣␣indirectly␣lost:␣0␣bytes␣in␣0␣blocks
==22981==␣␣␣␣␣␣possibly␣lost:␣0␣bytes␣in␣0␣blocks
==22981==␣␣␣␣still␣reachable:␣0␣bytes␣in␣0␣blocks
==22981==␣␣␣␣␣␣␣␣␣suppressed:␣0␣bytes␣in␣0␣blocks
==22981==
==22981==␣Use␣--track-origins=yes␣to␣see␣where␣uninitialised␣values␣come␣from
==22981==␣For␣lists␣of␣detected␣and␣suppressed␣errors,␣rerun␣with:␣-s
==22981==␣ERROR␣SUMMARY:␣6␣errors␣from␣6␣contexts␣(suppressed:␣0␣from␣0)

␣␣␣␣
```

1. What happens when this program runs?

Ans : The program executes normally seems not big deal.

2. Can you use gdb to find any problems with it?

Ans : No, gdb shows the program is terminated normally.

3. How about valgrind (again with the –leak-check=yes flag)?

Ans : Yes, valgrind shows that the program has a memory leak.

5. Write a program that creates an array of integers called data of size 100 using malloc; then, set data[100] to zero. What happens when you run this program? What happens when you run this program using valgrind? Is the program correct?

Solution: Please refer to List 1 (q5.c):

Listing 3: q5.c

```c
/*
Write a program that creates an array of integers called data
of size 100 using malloc; then, set data[100] to zero. What happens
when you run this program? What happens when you run this
program using valgrind? Is the program correct?
*/


#include <stdio.h>
#include <stdlib.h>
int main()
{
    int *data = (int *)malloc(100 * sizeof(int));
    data[100] = 0;
    printf("data[100] = %d\n", data[100]);
    return 0;
}
```

Its execution results are as follows:

```
data[100] = 0
```

Valgrind's execution results are as follows:

```
==24012== Memcheck, a memory error detector
==24012== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==24012== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==24012== Command: ./q5
==24012==
==24012== Invalid write of size 4
==24012==    at 0x10918D: main (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q5)
==24012==  Address 0x4a961d0 is 0 bytes after a block of size 400 alloc'd
==24012==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-
    linux.so)
==24012==    by 0x10917E: main (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q5)
==24012==
==24012== Invalid read of size 4
```

```
==24012==    at 0x10919D: main (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q5)
==24012==  Address 0x4a961d0 is 0 bytes after a block of size 400 alloc'd
==24012==␣␣␣␣at␣0x4848899:␣malloc␣(in␣/usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==24012==␣␣␣␣by␣0x10917E:␣main␣(in␣/home/it/Desktop/Advanced_Operation_Systen/hw3/q5)
==24012==
data[100]␣=␣0
==24012==
==24012==␣HEAP␣SUMMARY:
==24012==␣␣␣␣␣in␣use␣at␣exit:␣400␣bytes␣in␣1␣blocks
==24012==␣␣␣total␣heap␣usage:␣2␣allocs,␣1␣frees,␣1,424␣bytes␣allocated
==24012==
==24012==␣400␣bytes␣in␣1␣blocks␣are␣definitely␣lost␣in␣loss␣record␣1␣of␣1
==24012==␣␣␣␣at␣0x4848899:␣malloc␣(in␣/usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==24012==␣␣␣␣by␣0x10917E:␣main␣(in␣/home/it/Desktop/Advanced_Operation_Systen/hw3/q5)
==24012==
==24012==␣LEAK␣SUMMARY:
==24012==␣␣␣␣definitely␣lost:␣400␣bytes␣in␣1␣blocks
==24012==␣␣␣␣indirectly␣lost:␣0␣bytes␣in␣0␣blocks
==24012==␣␣␣␣␣␣possibly␣lost:␣0␣bytes␣in␣0␣blocks
==24012==␣␣␣␣still␣reachable:␣0␣bytes␣in␣0␣blocks
==24012==␣␣␣␣␣␣␣␣␣suppressed:␣0␣bytes␣in␣0␣blocks
==24012==
==24012==␣For␣lists␣of␣detected␣and␣suppressed␣errors,␣rerun␣with:␣-s
==24012==␣ERROR␣SUMMARY:␣3␣errors␣from␣3␣contexts␣(suppressed:␣0␣from␣0)
␣␣␣␣␣␣␣␣
```

1. What happens when you run this program?

Ans : The program executes normally seems good no big problems.

2. What happens when you run this program using valgrind?

Ans : Valgrind shows that the program has a memory leak and invalid read and write.

3. Is the program correct?

Ans : No, the program is not correct. Becaues it use the memory that is not allocated.

6. Create a program that allocates an array of integers (as above), frees them, and then tries to print the value of one of the elements of the array. Does the program run? What happens when you use valgrind on it?

Solution: Please refer to List 1 (q6.c):

Listing 4: q6.c

```c
/*
Create a program that allocates an array of integers (as above), frees
them, and then tries to print the value of one of the elements of
the array. Does the program run? What happens when you use
valgrind on it?
*/


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

```c
int main()
{
    int *data = (int *)malloc(100 * sizeof(int));
    free(data);
    printf("data[0]␣=␣%d\n", data[0]);
    return 0;
}
```

Its execution results are as follows:

```
data[0] = 1570806496
```

Valgrind's execution results are as follows:

```
==24375== Command: ./q6
==24375==
==24375== Invalid read of size 4
==24375==    at 0x1091B3: main (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q6)
==24375==  Address 0x4a96040 is 0 bytes inside a block of size 400 free'd
==24375==␣␣␣␣at␣0x484B27F:␣free␣(in␣/usr/libexec/valgrind/vgpreload_memcheck-amd64-
    linux.so)
==24375==␣␣␣␣by␣0x1091AE:␣main␣(in␣/home/it/Desktop/Advanced_Operation_Systen/hw3/q6)
==24375==␣␣Block␣was␣alloc'd at
==24375==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-
    linux.so)
==24375==    by 0x10919E: main (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q6)
==24375==
data[0] = 0
==24375==
==24375== HEAP SUMMARY:
==24375==     in use at exit: 0 bytes in 0 blocks
==24375==   total heap usage: 2 allocs, 2 frees, 1,424 bytes allocated
==24375==
==24375== All heap blocks were freed -- no leaks are possible
==24375==
==24375== For lists of detected and suppressed errors, rerun with: -s
==24375== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

1. Does the program run?

Ans : Yes, The program can be run but the value is not controllable and predictable.

2. What happens when you use valgrind on it?

Ans : Valgrind shows that the program has no memory leak but there have a memory block has invalid read and it's size is 4 bytes.

7. Now pass a funny value to free (e.g., a pointer in the middle of the array you allocated above). What happens? Do you need tools to find this type of problem?

Solution: Please refer to List 1 (q7.c):

Listing 5: q7.c

```c
/*
Now pass a funny value to free (e.g., a pointer in the middle of the
array you allocated above). What happens? Do you need tools to
find this type of problem?
*/

#include <stdio.h>
#include <stdlib.h>
int main()
{
    int *data = (int *)malloc(3 * sizeof(int));
    data[0] = 1;

    free(data + 1);
    printf("data[0] = %d\n", data[0]);
    return 0;
}
```

Its execution results are as follows:

```
free(): invalid pointer
Aborted (core dumped)
```

Valgrind's execution results are as follows:

```
==25997== Command: ./q7
==25997==
==25997== Invalid free() / delete / delete[] / realloc()
==25997==    at 0x484B27F: free (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==25997==    by 0x1091BC: main (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q7)
==25997==  Address 0x4a96044 is 4 bytes inside a block of size 12 alloc'd
==25997==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==25997==    by 0x10919E: main (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q7)
==25997==
data[0] = 1
==25997==
==25997== HEAP SUMMARY:
==25997==     in use at exit: 12 bytes in 1 blocks
==25997==   total heap usage: 2 allocs, 2 frees, 1,036 bytes allocated
==25997==
==25997== 12 bytes in 1 blocks are definitely lost in loss record 1 of 1
==25997==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==25997==    by 0x10919E: main (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q7)
==25997==
==25997== LEAK SUMMARY:
==25997==    definitely lost: 12 bytes in 1 blocks
==25997==    indirectly lost: 0 bytes in 0 blocks
==25997==      possibly lost: 0 bytes in 0 blocks
==25997==    still reachable: 0 bytes in 0 blocks
==25997==         suppressed: 0 bytes in 0 blocks
==25997==
==25997== For lists of detected and suppressed errors, rerun with: -s
==25997== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
    
```

1. What happens?

Ans : The program is terminated with a SIGABRT signal which means a process aborts.

2. Do you need tools to find this type of problem?

Ans : Yes ,I use valgrind to find this type of problem. We can see the output of valgrind that we do a invalid free() operation. It seems we can't free a memory block that in the middle of a series of memory that allocated.

8. Try out some of the other interfaces to memory allocation. For example, create a simple vector-like data structure and related routines that use realloc() to manage the vector. Use an array to store the vectors elements; when a user adds an entry to the vector, use realloc() to allocate more space for it. How well does such a vector perform? How does it compare to a linked list? Use valgrind to help you find bugs.

   Solution: Please refer to List 1 (q8.c):

   Listing 6: q8.c

```c
/*
Try out some of the other interfaces to memory allocation.
For example, create a simple vector-like data structure and
related routines that use realloc() to manage the vector.
Use an array to store the vectors elements; when a user adds
an entry to the vector, use realloc() to allocate more space
for it. How well does such a vector perform? How does it compare
to a linked list? Use valgrind to help you find bugs
*/
#include <stdio.h>
#include <stdlib.h>

typedef struct node
{
    int data;
    struct node *next;
} node;

typedef struct vector
{
    int *array;
    int size;
    int capacity;
} vector;

void vinit(vector *v)
{
    // initialize the vector
    v->size = 0;
    v->capacity = 1;
    v->array = (int *)malloc(sizeof(int));
}
void vpop(vector *v)
{
    // pop the last element
    if (v->size > 0)
    {
        v->size--;
    }
    v->array = (int *)realloc(v->array, v->size * sizeof(int));
}
void vpush(vector *v, int element)
{
    // push the element to the end of the vector
    if (v->size == v->capacity)
    {
        v->capacity *= 2;
        v->array = (int *)realloc(v->array, v->capacity * sizeof(int));
    }
    v->array[v->size] = element;
    v->size++;
}
void vclear(vector *v)
{
    // clear the vector
    free(v->array);
}
void vshow(vector *v)
{
    printf("size: %d, capacity: %d => The Value in vector are ", v->size, v->capacity
        );
    // show the vector
    for (int i = 0; i < v->size; i++)
    {
        printf("%d ", v->array[i]);
    }
    printf("\n");
}

void llinit(node *head)
{
    // initialize the linked list
    head->next = NULL;
}
void llshow(node *head)
{
    // show the linked list
    printf("Linked List: ");
    node *temp = head;
    while (temp != NULL)
    {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
void llpush(node *head, int element)
{
    // insert the element to the end of the linked list
    node *temp = head;
    while (temp->next != NULL)
    {
        temp = temp->next;
    }
    node *new_node = (node *)malloc(sizeof(node));
    new_node->data = element;
    new_node->next = NULL;
    temp->next = new_node;
}
```

```c
void llpop(node *head)
{
    // pop the last element of the linked list
    node *temp = head;
    while (temp->next->next != NULL)
    {
        temp = temp->next;
    }
    free(temp->next);
    temp->next = NULL;
}
void llclear(node *head)
{
    // clear the linked list
    node *temp = head;
    while (temp != NULL)
    {
        node *temp2 = temp;
        temp = temp->next;
        free(temp2);
    }
}
int main()
{
    node *n = (node *)malloc(sizeof(node));
    n->data = 1;
    llshow(n);
    llpush(n, 2);
    llshow(n);
    llpush(n, 3);
    llshow(n);
    llpop(n);
    llshow(n);
    llpop(n);
    llshow(n);
    llclear(n);
    vector v;
    vinit(&v);
    vpush(&v, 1);
    vshow(&v);
    vpush(&v, 2);
    vshow(&v);
    vpush(&v, 3);
    vshow(&v);
    vclear(&v);
    vshow(&v);
}
```

Its execution results are as follows:

```
Linked List: 1
Linked List: 1 2
Linked List: 1 2 3
Linked List: 1 2
Linked List: 1
size: 1, capacity: 1 => The Value in vector are 1
size: 2, capacity: 2 => The Value in vector are 1 2
size: 3, capacity: 4 => The Value in vector are 1 2 3
size: 3, capacity: 4 => The Value in vector are 1790712579 21891 -529391540
```

Valgrind's execution results are as follows:

```
==31906== Conditional jump or move depends on uninitialised value(s)
==31906==    at 0x10942C: llshow (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q8)
==31906==    by 0x109537: main (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q8)
==31906==
Linked List: 1
==31906== Conditional jump or move depends on uninitialised value(s)
==31906==    at 0x10946F: llpush (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q8)
==31906==    by 0x109548: main (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q8)
==31906==
Linked List: 1 2
Linked List: 1 2 3
Linked List: 1 2
Linked List: 1
size: 1, capacity: 1 => The Value in vector are 1
size: 2, capacity: 2 => The Value in vector are 1 2
size: 3, capacity: 4 => The Value in vector are 1 2 3
==31906== Invalid read of size 4
==31906==    at 0x109381: vshow (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q8)
==31906==    by 0x10961C: main (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q8)
==31906==  Address 0x4a96610 is 0 bytes inside a block of size 16 free'd
==31906==    at 0x484B27F: free (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==31906==    by 0x10932D: vclear (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q8)
==31906==    by 0x109610: main (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q8)
==31906==  Block was alloc'd at
==31906==    at 0x484DCD3: realloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==31906==    by 0x1092D7: vpush (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q8)
==31906==    by 0x1095F8: main (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q8)
==31906==
size: 3, capacity: 4 => The Value in vector are 1 2 3
==31906==
==31906== HEAP SUMMARY:
==31906==     in use at exit: 16 bytes in 1 blocks
==31906==   total heap usage: 7 allocs, 6 frees, 1,100 bytes allocated
==31906==
==31906== 16 bytes in 1 blocks are definitely lost in loss record 1 of 1
==31906==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==31906==    by 0x10951D: main (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q8)
==31906==
==31906== LEAK SUMMARY:
==31906==    definitely lost: 16 bytes in 1 blocks
==31906==    indirectly lost: 0 bytes in 0 blocks
==31906==      possibly lost: 0 bytes in 0 blocks
==31906==    still reachable: 0 bytes in 0 blocks
==31906==         suppressed: 0 bytes in 0 blocks
==31906==
==31906== Use --track-origins=yes to see where uninitialised values come from
==31906== For lists of detected and suppressed errors, rerun with: -s
==31906== ERROR SUMMARY: 6 errors from 4 contexts (suppressed: 0 from 0)
```

---

1. How well does such a vector perform?

Ans : It's perform I is very well, It easy to use and clear to know. This is a good way to store data.

2. How does it compare to a linked list?

Ans : It compare to linked list is faster then linked list. If we need to add a new element to the end of vector and linked list, the vector is faster than linked list. Becaues of linked list need to find the last element of linked list, but the vector just need to add the new element to the end of vector. and it has stored the size and capacity of vector.

3. Use valgrind to help you find bugs