1.按照要求，编写如下 null.c 程序

```c
#include <stdio.h>

int main(){
    int *a = NULL;
    *a = 1;
    return 0;
}
```

编译运行得到报错：Segmentation fault，如下：

```
[2020202279@work122 ~]$ touch null.c
[2020202279@work122 ~]$ gcc -o null  null.c
[2020202279@work122 ~]$ ./null
Segmentation fault
```

2. gdb 展示的信息：程序接受 SIGSEGV 信号终止

```
(gdb) run
Starting program: /mnt/ics1-2020/2020202279/null

Program received signal SIGSEGV, Segmentation fault.
0x00000000004004fd in main () at null.c:5
5           *a = 1;
(gdb)
```

3.按要求输入命令，得到如下输出：

```
[2020202279@work122 ~]$ valgrind --leak-check=yes ./null
==91341== Memcheck, a memory error detector
==91341== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==91341== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==91341== Command: ./null
==91341==
==91341== Invalid write of size 4
==91341==    at 0x4004FD: main (null.c:5)
==91341==  Address 0x0 is not stack'd, malloc'd or (recently) free'd
==91341==
==91341==
==91341== Process terminating with default action of signal 11 (SIGSEGV)
==91341==  Access not within mapped region at address 0x0
==91341==    at 0x4004FD: main (null.c:5)
==91341==  If you believe this happened as a result of a stack
==91341==  overflow in your program's main thread (unlikely but
==91341==  possible), you can try to increase the size of the
==91341==  main thread stack using the --main-stacksize= flag.
==91341==  The main thread stack size used in this run was 8388608.
==91341==
==91341== HEAP SUMMARY:
==91341==     in use at exit: 0 bytes in 0 blocks
==91341==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==91341==
==91341== All heap blocks were freed -- no leaks are possible
==91341==
==91341== For lists of detected and suppressed errors, rerun with: -s
==91341== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
Segmentation fault
```

其中，Invalid write of size 4 指出了程序的错误：在 null.c 程序的第五行，尝试为指针 a 指向

的空间赋值，但 a 是空指针，对应的地址 0x0 不是可访问（写入）的空间，因此产生错误。
下一段则说明程序收到了信号 11（SIGSEGV）而终止，因为访问不在地址 0x0 的映射区域内。并给出针对可能出现的栈溢出情况给出提醒和建议。
下一段给出了堆区的总结，包括使用情况（多少 allocs、多少 frees 等）和释放情况。
最后则是错误总结。

4.按要求改写程序，malloc 但不 free

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

int main(){
    int *a = (int *)malloc(sizeof(int));
    *a = 1;
    return 0;
}
```

编译运行没有报错，gdb 中进程也正常退出，所以不能用 gdb 找到问题

```
[2020202279@work122 ~]$ gcc -g -o null null.c
[2020202279@work122 ~]$ ./null
[2020202279@work122 ~]$ gdb null
GNU gdb (GDB) 8.0.1
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from null...done.
(gdb) run
Starting program: /mnt/ics1-2020/2020202279/null
[Inferior 1 (process 203008) exited normally]
```

而 valgrind 的输出则不同，在对堆区进行总结时，指出进行了 1 个 alloc 和 0 个 free，并给出了对应的未释放的 malloc 指令的位置和所在函数。此外，还对内存泄露的总情况进行了总结和描述，指出了程序的错误。

```
[2020202279@work122 ~]$ valgrind --leak-check=yes ./null
==213363== Memcheck, a memory error detector
==213363== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==213363== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==213363== Command: ./null
==213363==
==213363==
```

```
==213363== HEAP SUMMARY:
==213363==     in use at exit: 4 bytes in 1 blocks
==213363==   total heap usage: 1 allocs, 0 frees, 4 bytes allocated
==213363==
==213363== 4 bytes in 1 blocks are definitely lost in loss record 1 of 1
==213363==    at 0x4C29F73: malloc (vg_replace_malloc.c:309)
==213363==    by 0x40053E: main (null.c:5)
==213363==
==213363== LEAK SUMMARY:
==213363==    definitely lost: 4 bytes in 1 blocks
==213363==    indirectly lost: 0 bytes in 0 blocks
==213363==      possibly lost: 0 bytes in 0 blocks
==213363==    still reachable: 0 bytes in 0 blocks
==213363==         suppressed: 0 bytes in 0 blocks
==213363==
==213363== For lists of detected and suppressed errors, rerun with: -s
==213363== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

5.

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

int main(){
    int *a = (int *)malloc(sizeof(int)*100);
    a[100] = 0;
    return 0;
}
```

编译运行不报错，valgrind 则提示访问 a[100]是不合法的，并且未进行 free 导致内存泄漏。因此，这个程序是不正确的，它访问了超出 malloc 分配空间的 a[100]，而且未释放分配的空间。

```
[2020202279@work122 ~]$ ./null
[2020202279@work122 ~]$ valgrind --leak-check=yes ./null
==28278== Memcheck, a memory error detector
==28278== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==28278== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==28278== Command: ./null
==28278==
==28278== Invalid write of size 4
==28278==    at 0x40054D: main (null.c:6)
==28278==  Address 0x52051d0 is 0 bytes after a block of size 400 alloc'd
==28278==    at 0x4C29F73: malloc (vg_replace_malloc.c:309)
==28278==    by 0x40053E: main (null.c:5)
==28278==
==28278==
==28278== HEAP SUMMARY:
==28278==     in use at exit: 400 bytes in 1 blocks
==28278==   total heap usage: 1 allocs, 0 frees, 400 bytes allocated
==28278==
==28278== 400 bytes in 1 blocks are definitely lost in loss record 1 of 1
==28278==    at 0x4C29F73: malloc (vg_replace_malloc.c:309)
==28278==    by 0x40053E: main (null.c:5)
==28278==
==28278== LEAK SUMMARY:
==28278==    definitely lost: 400 bytes in 1 blocks
==28278==    indirectly lost: 0 bytes in 0 blocks
==28278==      possibly lost: 0 bytes in 0 blocks
==28278==    still reachable: 0 bytes in 0 blocks
==28278==         suppressed: 0 bytes in 0 blocks
==28278==
==28278== For lists of detected and suppressed errors, rerun with: -s
==28278== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

6.

```c
int main(){
    int *a = (int *)malloc(sizeof(int)*100);
    free(a);
    printf("%d\n", a[1]);
    return 0;
}
```

该程序可以运行并且输出 0，但是 valgrind 工具则报错，显示它进行了非法读取，因为对应的空间已经 free 释放。不过此次由于申请的空间已经释放，所以没有内存泄漏。

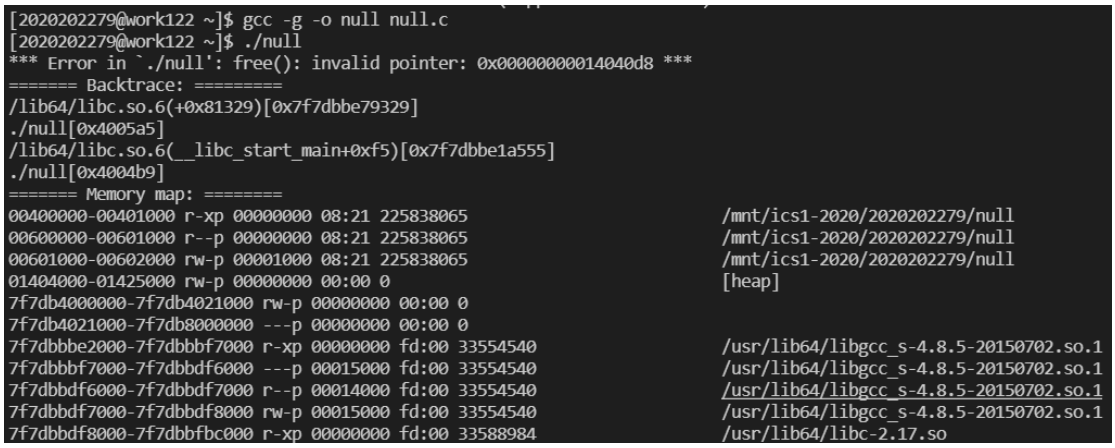
```
0[2020202279@work122 ~]$ gcc -g -o null null.c
[2020202279@work122 ~]$ ./null
0
[2020202279@work122 ~]$ valgrind --leak-check=yes ./null
==59269== Memcheck, a memory error detector
==59269== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==59269== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==59269== Command: ./null
==59269==
==59269== Invalid read of size 4
==59269==    at 0x4005E7: main (null.c:7)
==59269==  Address 0x5205044 is 4 bytes inside a block of size 400 free'd
==59269==    at 0x4C2B06D: free (vg_replace_malloc.c:540)
==59269==    by 0x4005DE: main (null.c:6)
==59269==  Block was alloc'd at
==59269==    at 0x4C29F73: malloc (vg_replace_malloc.c:309)
==59269==    by 0x4005CE: main (null.c:5)
==59269==
0
==59269==
==59269== HEAP SUMMARY:
==59269==     in use at exit: 0 bytes in 0 blocks
==59269==   total heap usage: 1 allocs, 1 frees, 400 bytes allocated
==59269==
==59269== All heap blocks were freed -- no leaks are possible
==59269==
==59269== For lists of detected and suppressed errors, rerun with: -s
==59269== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

7.

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

int main(){
    int *a = (int *)malloc(sizeof(int)*100);
    free(a+50);
    return 0;
}
```

编译运行直接报错，所以不需要 gdb 或 valgrind 工具即可发现错误，即不能用指向数组中间值的指针来进行释放。



8.经过实验，用 realloc 为向量扩容的方法只适合动态数组（如 malloc 分配的数组），而不适合静态数组。并且由于 realloc 在其后空间不足时，需要为原数组分配新的空间并将其复制过去，所以在数据规模较大或需要多次扩容时，该种向量的性能会有所下降。
与链表相比，这种向量要求存储空间必须是连续的，而链表则对空间的连续性没有要求，因此不需要对原数组进行复制。