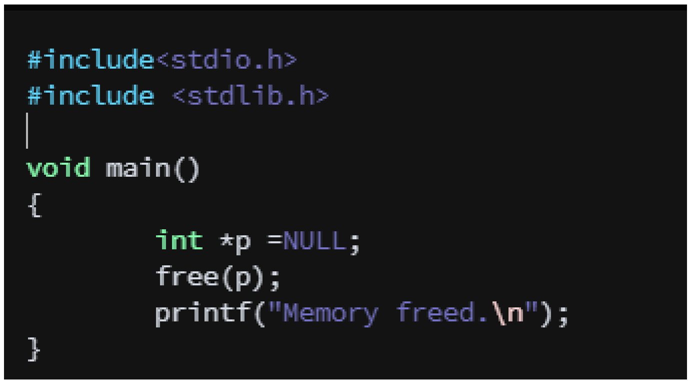
作业二

1．首先，编写一个名为 null.c 的简单程序，它创建一个指向整数的指针，将其设置为NULL，然后尝试对其进行释放内存操作。把它编译成一个名为 null 的可执行文件。当你运 行这个程序时会发生什么?



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

void main()
{
        int *p =NULL;
        free(p);
        printf("Memory freed.\n");
}
```

该程序将指针 `ptr` 设置为 NULL，然后尝试使用 `free()` 函数释放它指向的内存。由于 `ptr` 为 NULL，`free()` 函数不会执行任何操作，因此该程序不会引起内存泄漏或崩溃
输出

```
Memory freed.
```

2．接下来，编译该程序，其中包含符号信息（使用-g 标志）。这样做可以将更多信息放入可执行文件中，使调试器可以但问有关变量名称等的本多有用信息。通过输入 gdb null，在调试器下运行该程序，然后，一旦 gdb 运行，输入 run。gdb 显示什么信息?

我们输入

```
gcc -g null.c -o null
gdb null

(gdb) run

Starting program: /home/ubuntu/null

Memory freed.

[Inferior 1 (process 1992109) exited with code 016]
```

3、编写一个使用 malloc()来分配内存的简单程序，但在退出之前忘记释放它。这个程序运行时会发生什么？你可以用 gdb 来查找它的任何问题吗？用 valgrind 呢（再次使用--leak-check=yes 标志）？

编写一个`leak.c`程序

```
#include <stdlib.h>
#include <stdio.h>
int main() {
int* ptr = (int*)malloc(sizeof(int));
*ptr = 10; printf("%d\n", *ptr);
return 0;
}
```

用gdb调试

```
Reading symbols from leak...
(gdb) r
Starting program: /home/ubuntu/leak
10
[Inferior 1 (process 1996434) exited normally]
```

无法查找出问题

利用valgrind查看

输入 `valgrind --leak-check=yes ./leak`
输出如下

```
==1996842== HEAP SUMMARY:

==1996842== in use at exit: 4 bytes in 1 blocks

==1996842== total heap usage: 2 allocs, 1 frees, 1,028 bytes allocated

==1996842==

==1996842== 4 bytes in 1 blocks are definitely lost in loss record 1 of 1

==1996842== at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)

==1996842== by 0x10917E: main (leak.c:5)
```

```
==1996842==

==1996842== LEAK SUMMARY:

==1996842== definitely lost: 4 bytes in 1 blocks

==1996842== indirectly lost: 0 bytes in 0 blocks

==1996842== possibly lost: 0 bytes in 0 blocks

==1996842== still reachable: 0 bytes in 0 blocks

==1996842== suppressed: 0 bytes in 0 blocks

==1996842==

==1996842== For lists of detected and suppressed errors, rerun with: -s

==1996842== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

HEAP SUMMARY 部分给出了程序在退出时使用堆（heap）的情况，包括分配的内存块数、已分配内存总量、仍在使用中的内存块数、已释放内存块数和已释放内存总量。在这个例子中，程序只分配了一个大小为4字节的内存块，并且在程序退出时这个内存块仍在使用中。

LEAK SUMMARY 部分给出了内存泄漏的情况，包括所有的内存泄漏信息，以及内存泄漏的总量。在这个例子中，Valgrind 检测到程序在退出时有一个内存块（大小为4字节）没有被释放，被称为“definitely lost”。

在最后几行，Valgrind 给出了错误总结，指出有1个错误（definitely lost）在一个上下文中被检测到，并且没有被抑制

4、编写一个程序，使用 malloc 创建一个名为 data、大小为 100 的整数数组。然后，将data[100]设置为 0。当你运行这个程序时会发生什么？当你使用 valgrind 运行这个程序时会 发生什么？程序是否正确？

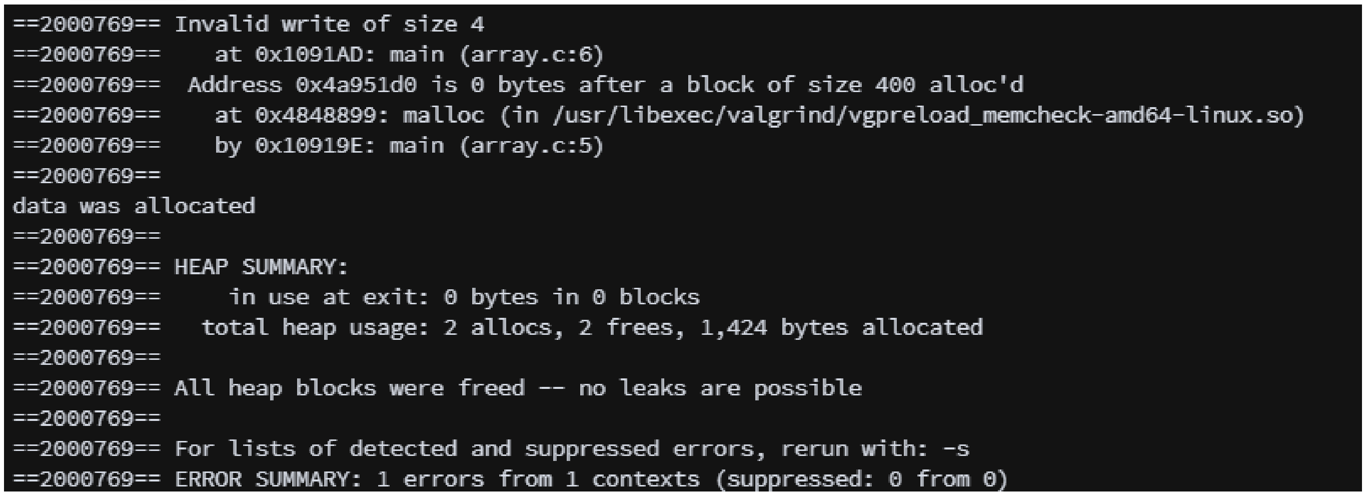
编写一个`array.c` 程序

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

编译运行后输出

```
data was allocated
```

使用`valgrind`发现越界



```
==2000769== Invalid write of size 4
==2000769==    at 0x1091AD: main (array.c:6)
==2000769==  Address 0x4a951d0 is 0 bytes after a block of size 400 alloc'd
==2000769==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==2000769==    by 0x10919E: main (array.c:5)
==2000769==
data was allocated
==2000769==
==2000769== HEAP SUMMARY:
==2000769==     in use at exit: 0 bytes in 0 blocks
==2000769==   total heap usage: 2 allocs, 2 frees, 1,424 bytes allocated
==2000769==
==2000769== All heap blocks were freed -- no leaks are possible
==2000769==
==2000769== For lists of detected and suppressed errors, rerun with: -s
==2000769== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

5、创建一个分配整数数组的程序（如上所述），释放它们，然后尝试打印数组中某个元素的值。程序会运行吗？当你使用 valgrind 时会发生什么？

程序

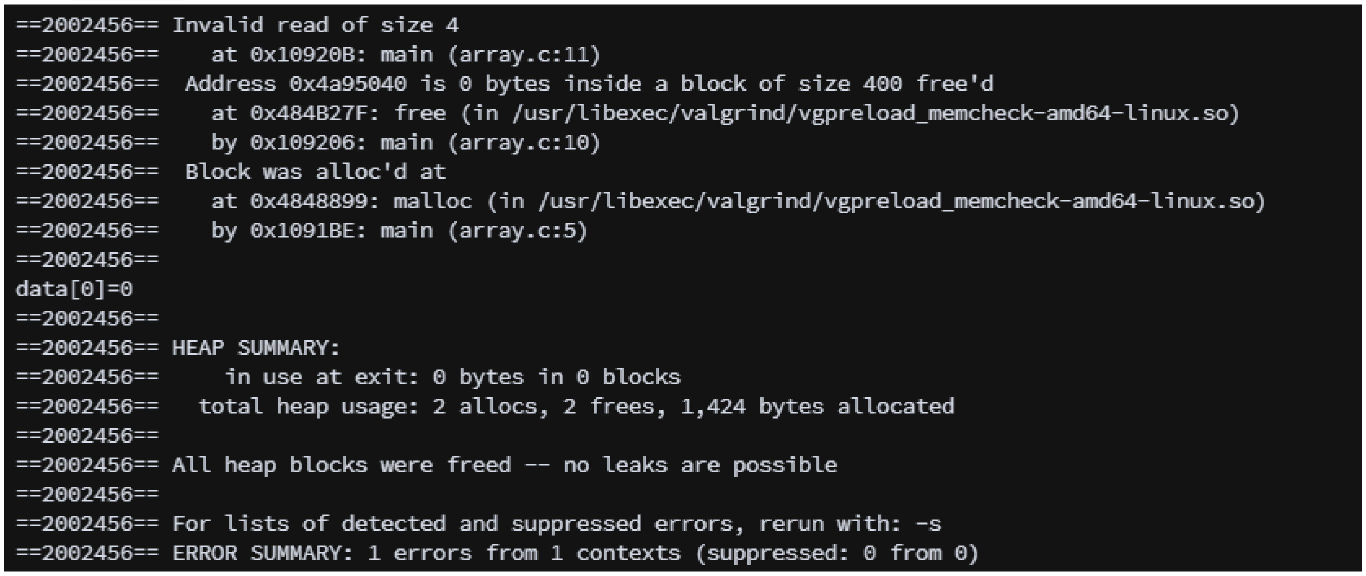
```
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int *data = malloc(100*sizeof(int));
        for (int i = 0; i < 100; i++) {
        data[i] = i;
    }
        printf("data was allocated\n" )
        free(data);
        printf("data[0]=%d\n",data[0]);
        return 0;
}
```

运行结果

```
data was allocated
data[0]=1671536644
```

程序会运行但是结果不正确

使用`valgrind`发现无效的读错误



```
==2002456== Invalid read of size 4
==2002456==    at 0x10920B: main (array.c:11)
==2002456==  Address 0x4a95040 is 0 bytes inside a block of size 400 free'd
==2002456==    at 0x484B27F: free (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==2002456==    by 0x109206: main (array.c:10)
==2002456==  Block was alloc'd at
==2002456==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==2002456==    by 0x1091BE: main (array.c:5)
==2002456==
data[0]=0
==2002456==
==2002456== HEAP SUMMARY:
==2002456==     in use at exit: 0 bytes in 0 blocks
==2002456==   total heap usage: 2 allocs, 2 frees, 1,424 bytes allocated
==2002456==
==2002456== All heap blocks were freed -- no leaks are possible
==2002456==
==2002456== For lists of detected and suppressed errors, rerun with: -s
==2002456== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```