Assignment1

Write-up 2

Answer the questions in the comments in pointer.c. For example, why are

some of the statements valid and some are not?

1.What is the type of argv?

char *argv[]是一个指向字符串的指针数组，他的元素个数是argc，存放的是指向每一个参数的指针

2.printf("char d = %c\n", d); // What does this print?

char d = 6

3.pcp = argv; // Why is this assignment valid?

pcp是一个指向指向char类型指针的指针，argv也是一个指向指向char类型指针的指针，所以语句是合法的

4.What is the type of pcc2?

一个指向字符常量的指针

5.For each of the following, why is the assignment:

*pcc = ’7’; // invalid? 由于pcc是指向字符常量的指针，它指向的字符常量是不可改的，所以这条语句不合法

pcc = *pcp; // valid? *pcp是一个字符型的指针，将pcc的指针指向pcp所指向的字符，合法

pcc = argv[0]; // valid? argv[0]是一个字符串，将pcc指针更改为指向该字符串，合法

6.For each of the following, why is the assignment:

cp = *pcp; // invalid? cp是指向字符串的常指针，不能修改该指针的指向。而 *pcp是指向char类型的指针

cp = *argv; // invalid? *argv是指向char类型的指针，不能改变常指针的指向

*cp = ’!’; // valid?cp是指向字符串的常指针,可以改变它指向的值，即改为这里的'!'

7.For each of the following, why is the assignment:

cpc = *pcp; // invalid? cpc是一个指向字符串常量的常指针，不可以更改指针的指向

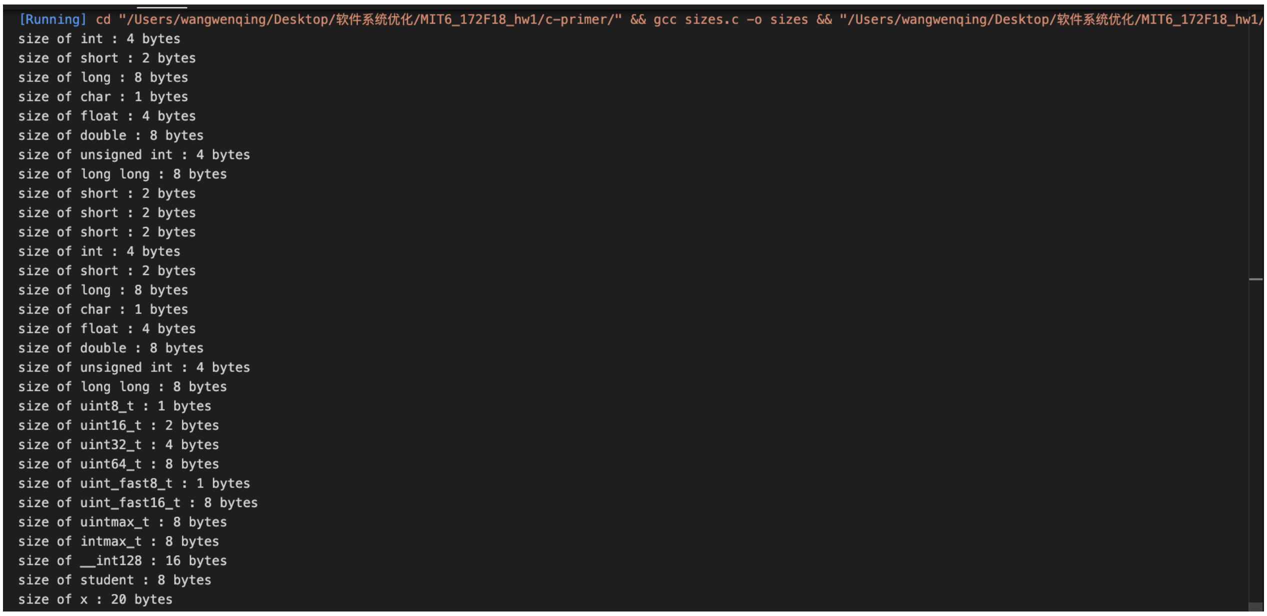
cpc = argv[0]; // invalid? cpc是一个指向字符串常量的常指针，不可以更改指向的值

*cpc = ’@’; // invalid? 不可以更改指向的字符串常量的值

Write-up 3

For each of the types in the sizes.c exercise above, print the size of a pointer to that type. Recall that obtaining the address of an array or struct requires the & operator. Provide the output of your program (which should include the sizes of both the actual type and a pointer to it) in the writeup.

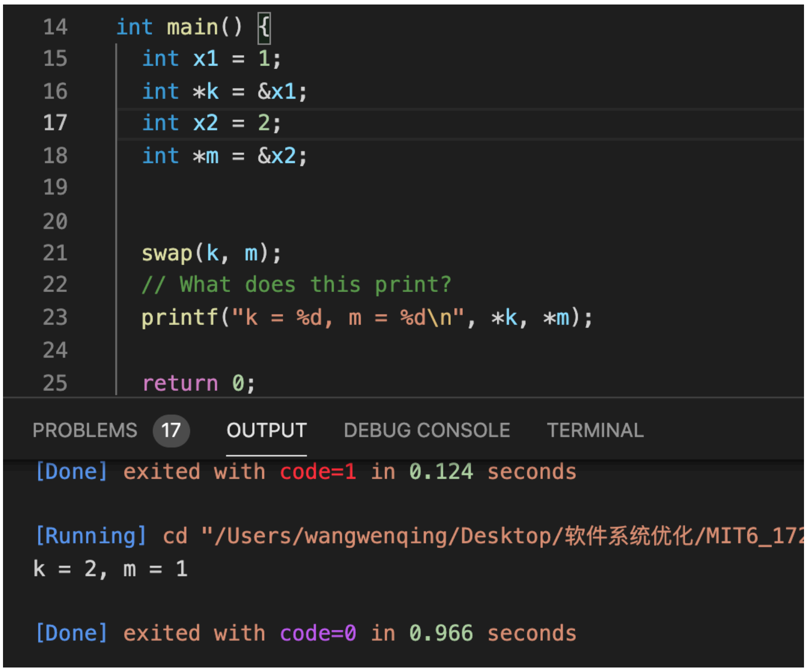
运行结果如图



```
[Running] cd "/Users/wangwenqing/Desktop/软件系统优化/MIT6_172F18_hw1/c-primer/" && gcc sizes.c -o sizes && "/Users/wangwenqing/Desktop/软件系统优化/MIT6_172F18_hw1/
size of int : 4 bytes
size of short : 2 bytes
size of long : 8 bytes
size of char : 1 bytes
size of float : 4 bytes
size of double : 8 bytes
size of unsigned int : 4 bytes
size of long long : 8 bytes
size of short : 2 bytes
size of short : 2 bytes
size of short : 2 bytes
size of int : 4 bytes
size of short : 2 bytes
size of long : 8 bytes
size of char : 1 bytes
size of float : 4 bytes
size of double : 8 bytes
size of unsigned int : 4 bytes
size of long long : 8 bytes
size of uint8_t : 1 bytes
size of uint16_t : 2 bytes
size of uint32_t : 4 bytes
size of uint64_t : 8 bytes
size of uint_fast8_t : 1 bytes
size of uint_fast16_t : 8 bytes
size of uintmax_t : 8 bytes
size of intmax_t : 8 bytes
size of __int128 : 16 bytes
size of student : 8 bytes
size of x : 20 bytes
```

Write-up 4

File swap.c contains the code to swap two integers. Rewrite the swap() function using pointers and make appropriate changes in main() function so that the values are swapped with a call to swap(). Compile the code with make swap and run the program with ./swap. Provide your edited code in the writeup. Verify that the results of both sizes.c and swap.c are correct by using the python script verifier.py.

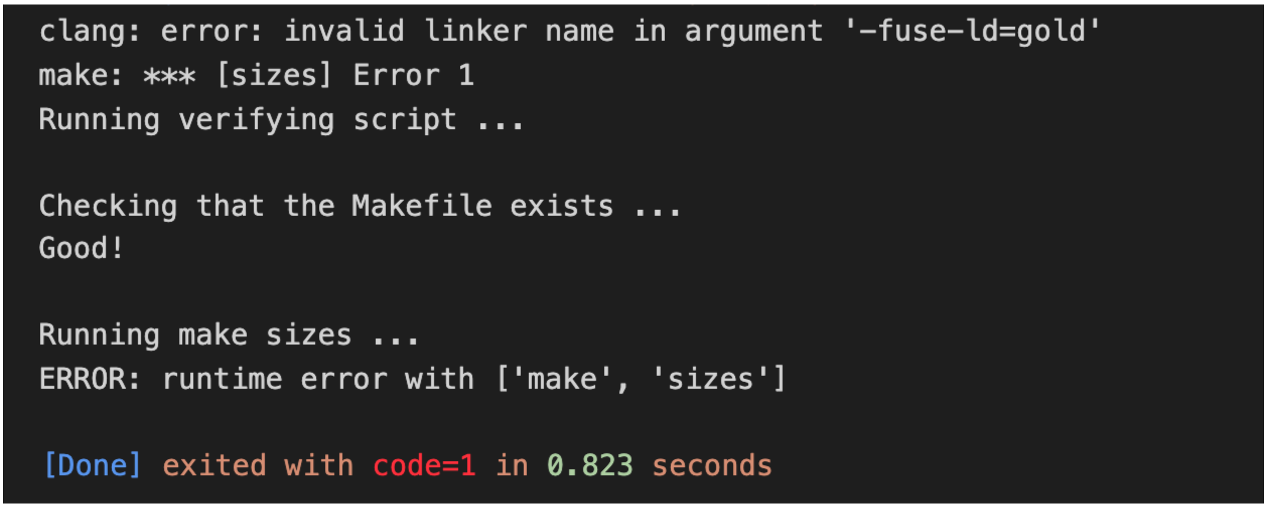


```c
int main() {
  int x1 = 1;
  int *k = &x1;
  int x2 = 2;
  int *m = &x2;


  swap(k, m);
  // What does this print?
  printf("k = %d, m = %d\n", *k, *m);

  return 0;
```

```
[Done] exited with code=1 in 0.124 seconds

[Running] cd "/Users/wangwenqing/Desktop/软件系统优化/MIT6_172
k = 2, m = 1

[Done] exited with code=0 in 0.966 seconds
```

verify运行报错的解决方案

运行报错如图所示



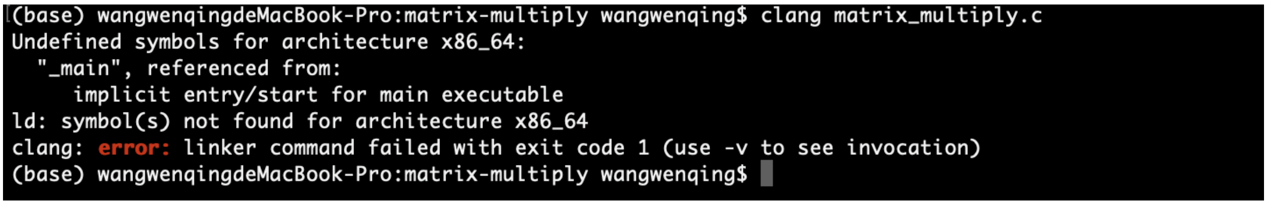
```
clang: error: invalid linker name in argument '-fuse-ld=gold'
make: *** [sizes] Error 1
Running verifying script ...

Checking that the Makefile exists ...
Good!

Running make sizes ...
ERROR: runtime error with ['make', 'sizes']

 [Done] exited with code=1 in 0.823 seconds
```

在makefile中删除对应的库即可解决

Write-up 5

Now, what do you see when you type make clean; make?

mac本地跑报错如下



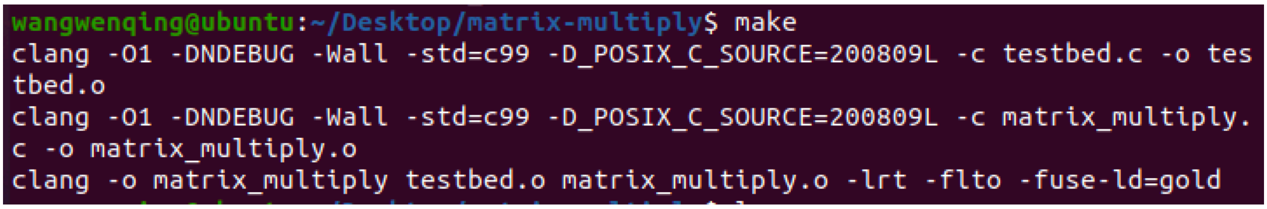
```
(base) wangwenqingdeMacBook-Pro:matrix-multiply wangwenqing$ clang matrix_multiply.c
Undefined symbols for architecture x86_64:
  "_main", referenced from:
     implicit entry/start for main executable
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see invocation)
(base) wangwenqingdeMacBook-Pro:matrix-multiply wangwenqing$
```

解决方案参考链接：https://stackoverflow.com/questions/30011041/ios-with-swift-main-referenced-from-implicit-entry-start-for-main-executab

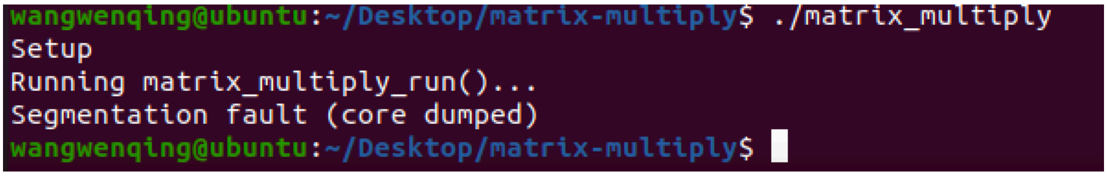
在makefile中将这里原先的-O1改为 CFLAGS_RELEASE := -O3 -DNDEBUG

并将这里CFLAGS_ASAN := -O1 -g -fsanitize=address 也改为-O3

运行结果



```
wangwenqing@ubuntu:~/Desktop/matrix-multiply$ make
clang -O1 -DNDEBUG -Wall -std=c99 -D_POSIX_C_SOURCE=200809L -c testbed.c -o tes
tbed.o
clang -O1 -DNDEBUG -Wall -std=c99 -D_POSIX_C_SOURCE=200809L -c matrix_multiply.
c -o matrix_multiply.o
clang -o matrix_multiply testbed.o matrix_multiply.o -lrt -flto -fuse-ld=gold
```

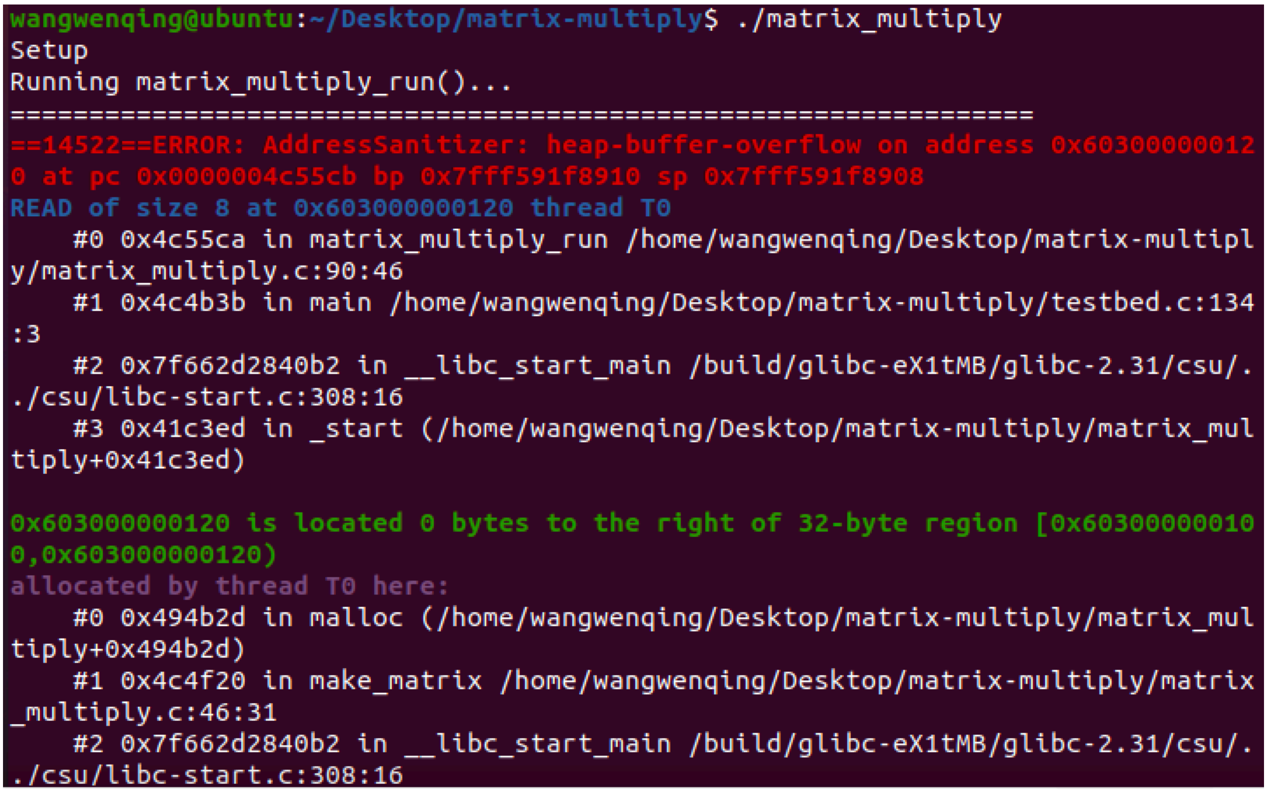


```
wangwenqing@ubuntu:~/Desktop/matrix-multiply$ ./matrix_multiply
Setup
Running matrix_multiply_run()...
Segmentation fault (core dumped)
wangwenqing@ubuntu:~/Desktop/matrix-multiply$
```

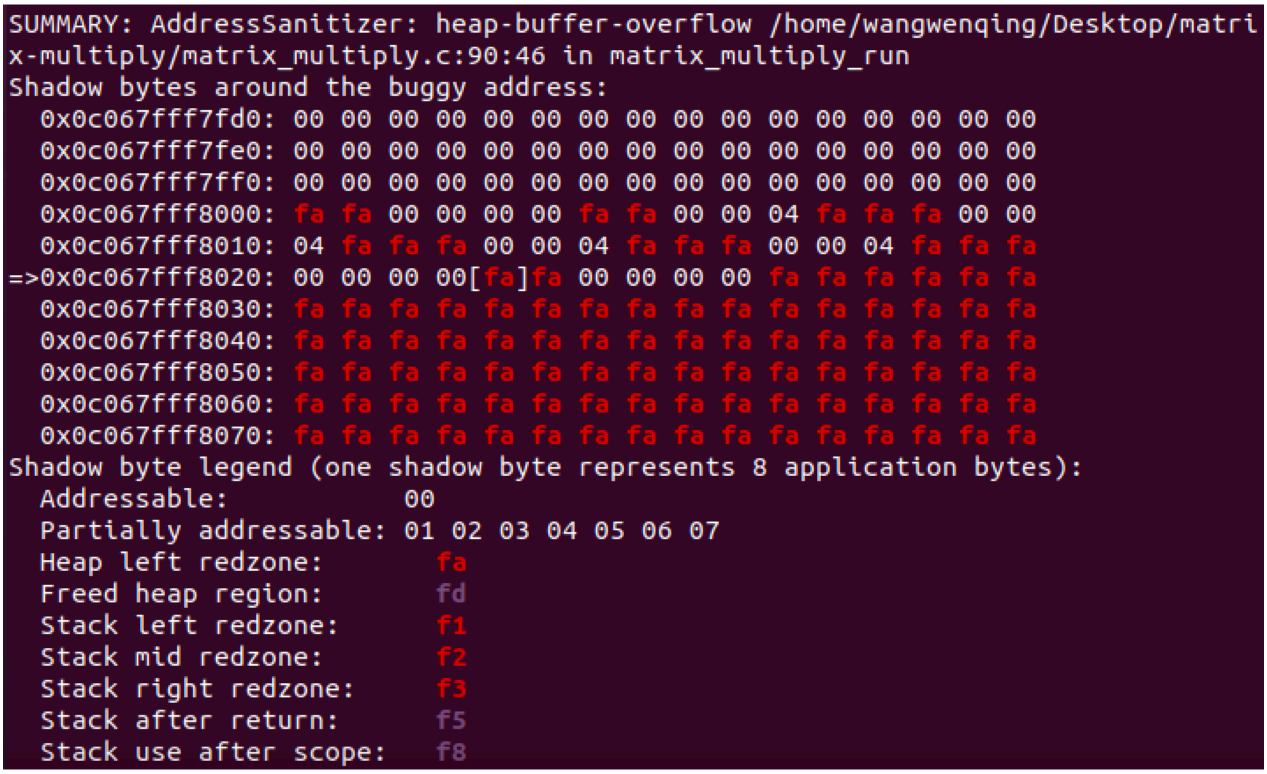
此外将CFLAGS_DEBUG := -g -DDEBUG -O0错误地改为了-O3会导致错误

Write-up 6

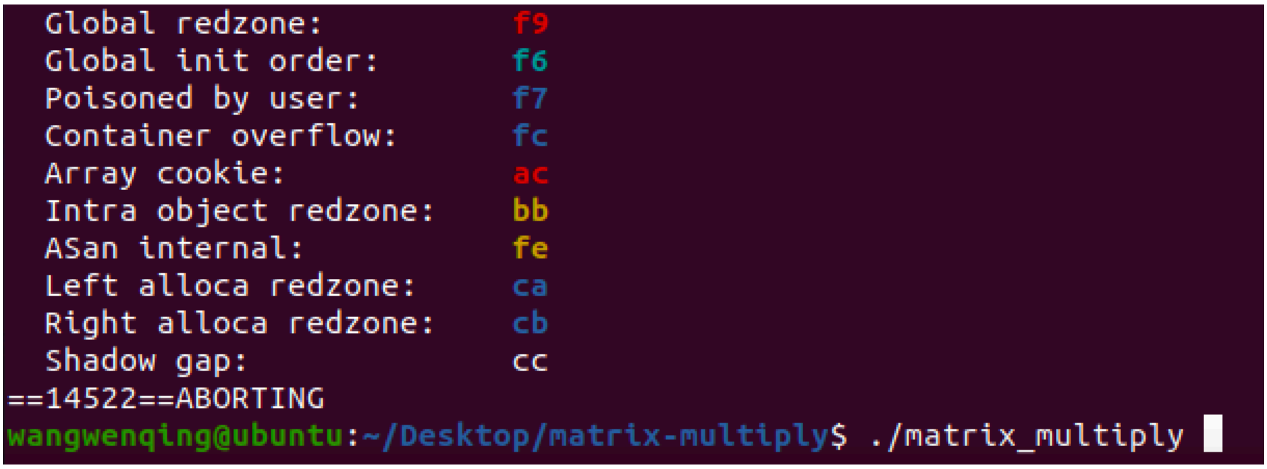
What output do you see from AddressSanitizer regarding the memory bug? Paste it into your writeup here.



```
wangwenqing@ubuntu:~/Desktop/matrix-multiply$ ./matrix_multiply
Setup
Running matrix_multiply_run()...
=================================================================
==14522==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x60300000012
0 at pc 0x0000004c55cb bp 0x7fff591f8910 sp 0x7fff591f8908
READ of size 8 at 0x603000000120 thread T0
    #0 0x4c55ca in matrix_multiply_run /home/wangwenqing/Desktop/matrix-multipl
y/matrix_multiply.c:90:46
    #1 0x4c4b3b in main /home/wangwenqing/Desktop/matrix-multiply/testbed.c:134
:3
    #2 0x7f662d2840b2 in __libc_start_main /build/glibc-eX1tMB/glibc-2.31/csu/.
./csu/libc-start.c:308:16
    #3 0x41c3ed in _start (/home/wangwenqing/Desktop/matrix-multiply/matrix_mul
tiply+0x41c3ed)

0x603000000120 is located 0 bytes to the right of 32-byte region [0x60300000010
0,0x603000000120)
allocated by thread T0 here:
    #0 0x494b2d in malloc (/home/wangwenqing/Desktop/matrix-multiply/matrix_mul
tiply+0x494b2d)
    #1 0x4c4f20 in make_matrix /home/wangwenqing/Desktop/matrix-multiply/matrix
_multiply.c:46:31
    #2 0x7f662d2840b2 in __libc_start_main /build/glibc-eX1tMB/glibc-2.31/csu/.
./csu/libc-start.c:308:16
```



```
SUMMARY: AddressSanitizer: heap-buffer-overflow /home/wangwenqing/Desktop/matri
x-multiply/matrix_multiply.c:90:46 in matrix_multiply_run
Shadow bytes around the buggy address:
  0x0c067fff7fd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c067fff7fe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c067fff7ff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c067fff8000: fa fa 00 00 00 00 fa fa 00 00 04 fa fa fa 00 00
  0x0c067fff8010: 04 fa fa fa 00 00 04 fa fa fa 00 00 04 fa fa fa
=>0x0c067fff8020: 00 00 00 00[fa]fa 00 00 00 00 fa fa fa fa fa fa
  0x0c067fff8030: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c067fff8040: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c067fff8050: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c067fff8060: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c067fff8070: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
```



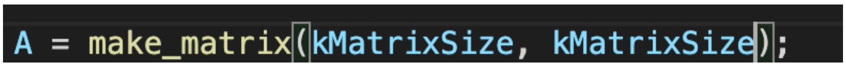
```
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==14522==ABORTING
wangwenqing@ubuntu:~/Desktop/matrix-multiply$ ./matrix_multiply 
```

Write-up 7

After you fix your program, run ./matrix_multiply -p. Paste the program output showing that the matrix multiplication is working correctly.

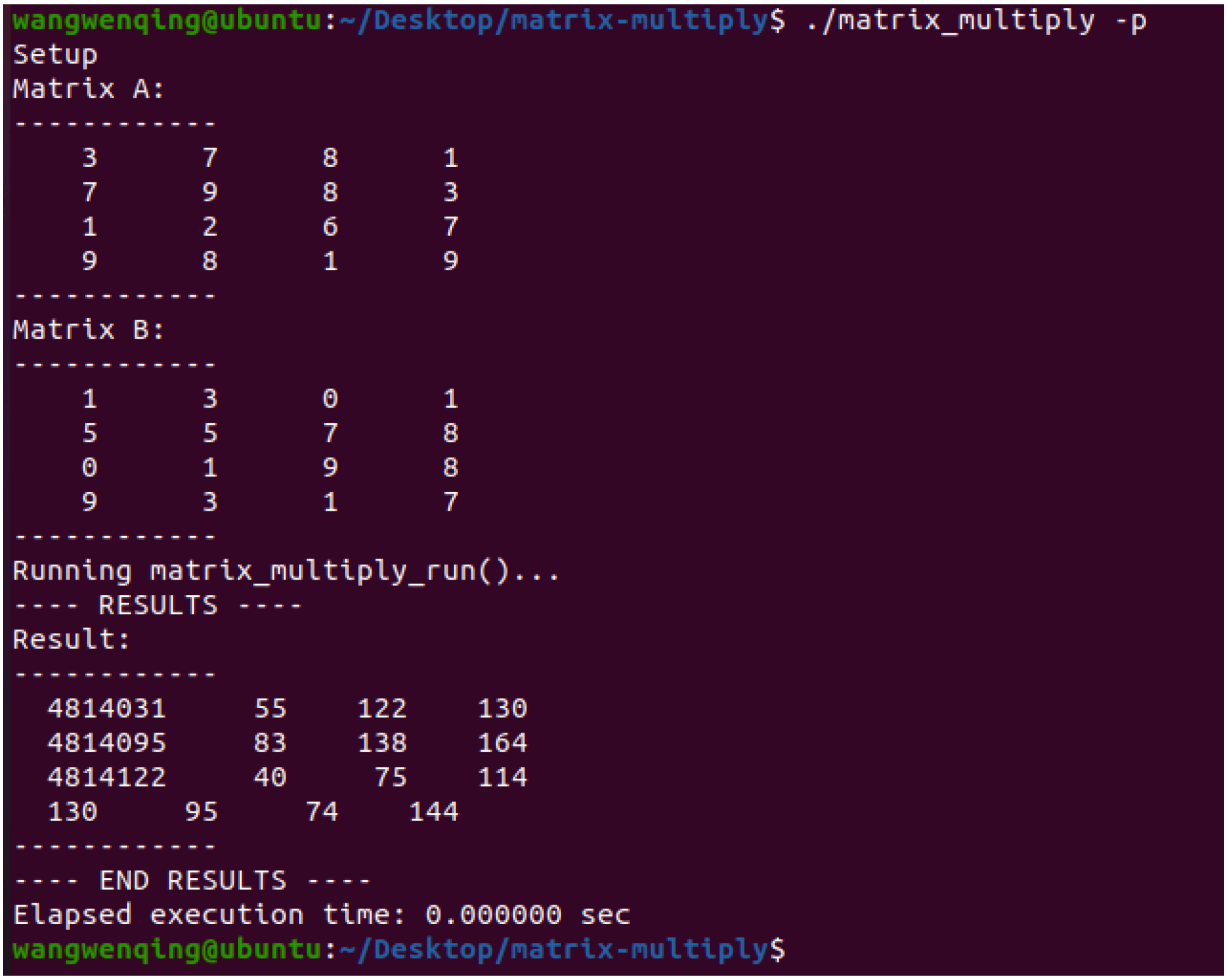
在testbed可以看到AB矩阵的维数不对应，无法相乘

这里将A矩阵的维数改为如图



```
A = make_matrix(kMatrixSize, kMatrixSize);
```

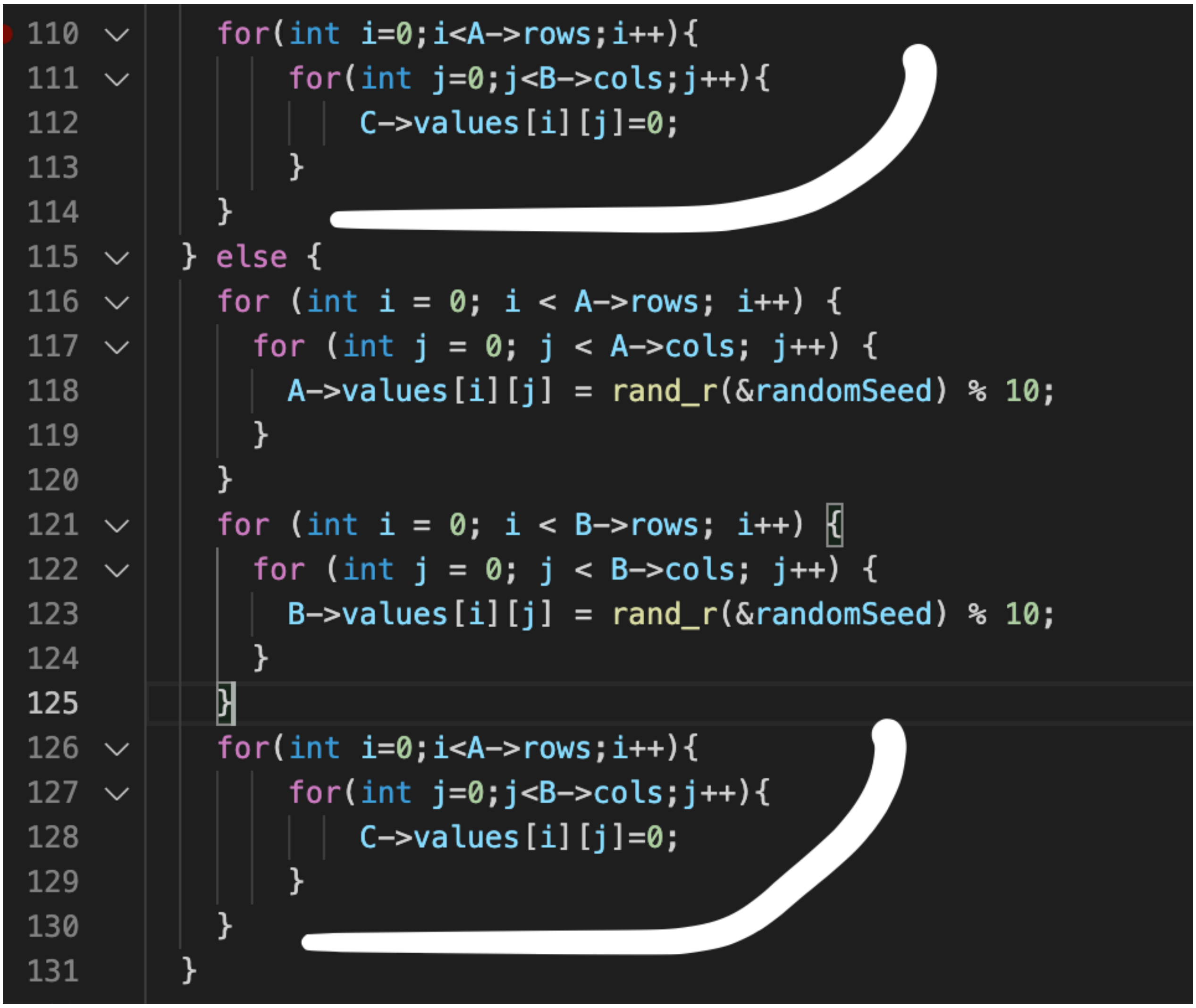
运行成功



```
wangwenqing@ubuntu:~/Desktop/matrix-multiply$ ./matrix_multiply -p
Setup
Matrix A:
------------
    3      7      8      1
    7      9      8      3
    1      2      6      7
    9      8      1      9
------------
Matrix B:
------------
    1      3      0      1
    5      5      7      8
    0      1      9      8
    9      3      1      7
------------
Running matrix_multiply_run()...
---- RESULTS ----
Result:
------------
  4814031     55    122    130
  4814095     83    138    164
  4814122     40     75    114
  130     95     74    144
------------
---- END RESULTS ----
Elapsed execution time: 0.000000 sec
wangwenqing@ubuntu:~/Desktop/matrix-multiply$
```

这里打印出来4814031是由于没有对C矩阵初始化导致的

增加对C矩阵初始化的代码如下

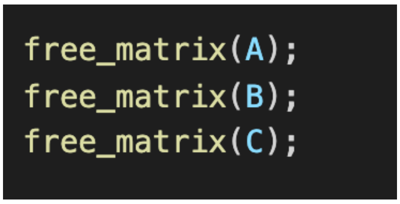


```
 for(int i=0;i<A->rows;i++){
   for(int j=0;j<B->cols;j++){
     C->values[i][j]=0;
   }
 }
} else {
 for (int i = 0; i < A->rows; i++) {
  for (int j = 0; j < A->cols; j++) {
   A->values[i][j] = rand_r(&randomSeed) % 10;
  }
 }
 for (int i = 0; i < B->rows; i++) {
  for (int j = 0; j < B->cols; j++) {
   B->values[i][j] = rand_r(&randomSeed) % 10;
  }
 }
 for(int i=0;i<A->rows;i++){
   for(int j=0;j<B->cols;j++){
     C->values[i][j]=0;
   }
 }
}
```

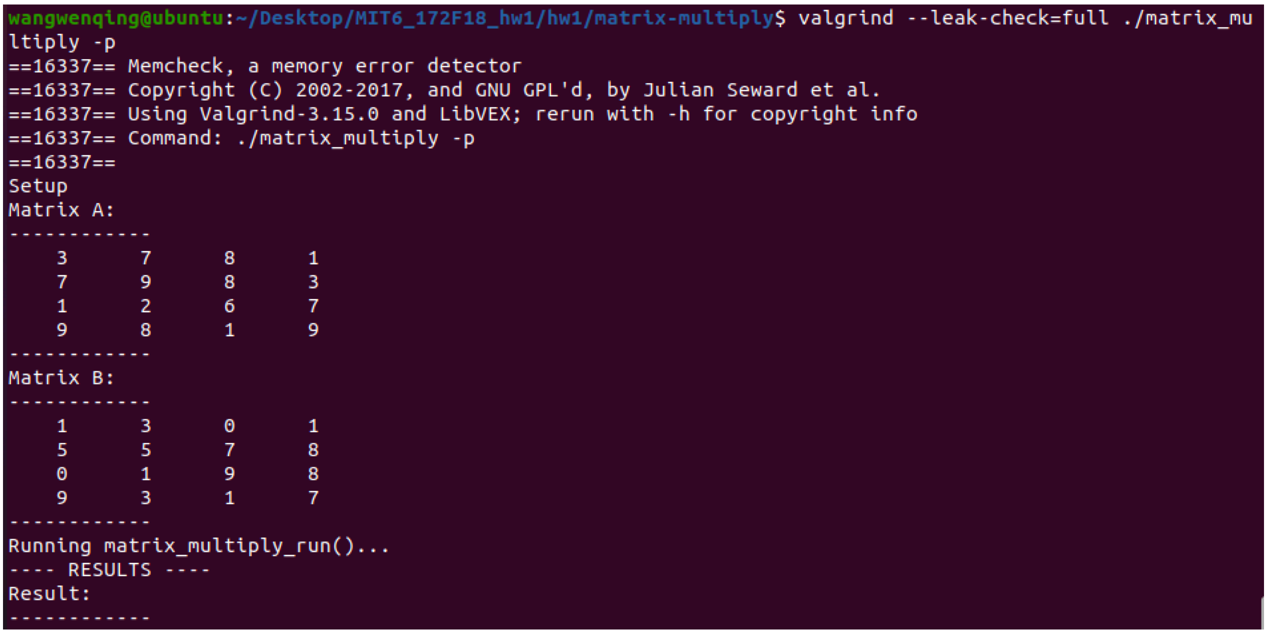
Write-up 8

Paste the output from Valgrind showing that there is no error in your program.

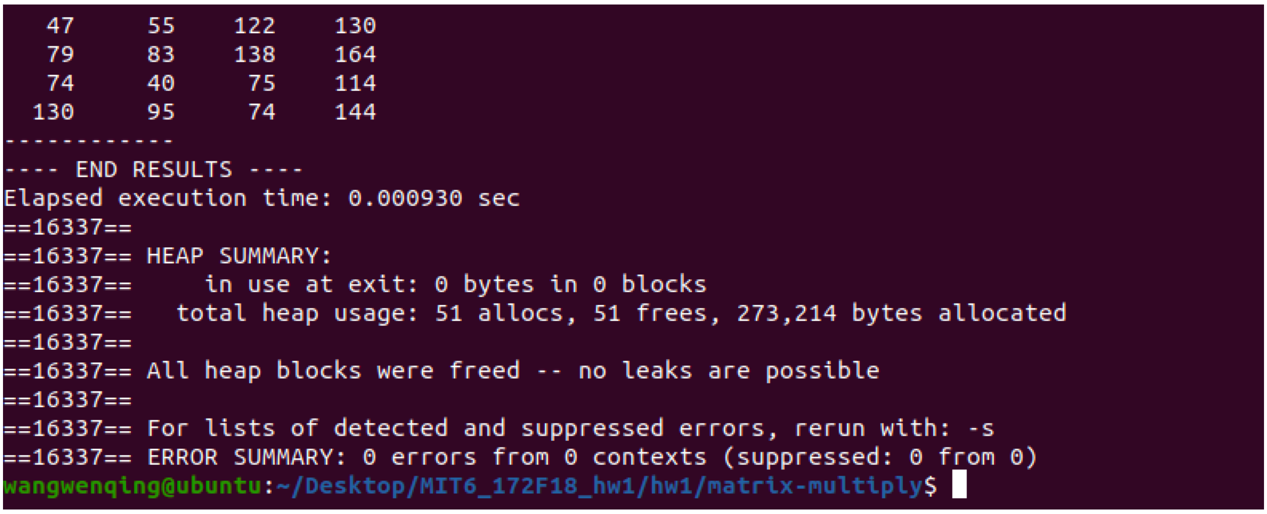
在matrix multiply之后增加free matrix操作



```
free_matrix(A);
free_matrix(B);
free_matrix(C);
```



```
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw1/hw1/matrix-multiply$ valgrind --leak-check=full ./matrix_mu
ltiply -p
==16337== Memcheck, a memory error detector
==16337== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==16337== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==16337== Command: ./matrix_multiply -p
==16337==
Setup
Matrix A:
------------
    3      7      8      1
    7      9      8      3
    1      2      6      7
    9      8      1      9
------------
Matrix B:
------------
    1      3      0      1
    5      5      7      8
    0      1      9      8
    9      3      1      7
------------
Running matrix_multiply_run()...
---- RESULTS ----
Result:
------------
```



```
   47     55    122    130
   79     83    138    164
   74     40     75    114
  130     95     74    144
------------
---- END RESULTS ----
Elapsed execution time: 0.000930 sec
==16337==
==16337== HEAP SUMMARY:
==16337==     in use at exit: 0 bytes in 0 blocks
==16337==   total heap usage: 51 allocs, 51 frees, 273,214 bytes allocated
==16337==
==16337== All heap blocks were freed -- no leaks are possible
==16337==
==16337== For lists of detected and suppressed errors, rerun with: -s
==16337== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw1/hw1/matrix-multiply$
```