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?

```c
int main(int argc, char* argv[]) {   // What is the type of argv?
  int i = 5;
  // The & operator here gets the address of i and stores it into pi
  int* pi = &i;
  // The * operator here dereferences pi and stores the value -- 5 --
  // into j.
  int j = *pi;

  char c[] = "6.172";
  char* pc = c;   // Valid assignment: c acts like a pointer to c[0] here.
  char d = *pc;
  printf("char d = %c\n", d);  // What does this print?

  // compound types are read right to left in C.
  // pcp is a pointer to a pointer to a char, meaning that
  // pcp stores the address of a char pointer.
  char** pcp;
  pcp = argv;  // Why is this assignment valid?

  const char* pcc = c;   // pcc is a pointer to char constant
  char const* pcc2 = c;   // What is the type of pcc2?

  // For each of the following, why is the assignment:
  *pcc = '7';  // invalid?
  pcc = *pcp;  // valid?
  pcc = argv[0];  // valid?

  char* const cp = c;   // cp is a const pointer to char
  // For each of the following, why is the assignment:
  cp = *pcp;  // invalid?
  cp = *argv;  // invalid?
  *cp = '!';  // valid?

  const char* const cpc = c;   // cpc is a const pointer to char const
  // For each of the following, why is the assignment:
  cpc = *pcp;  // invalid?
  cpc = argv[0];  // invalid?
  *cpc = '@';  // invalid?

  return 0;
}
```

Figure 5: An example of valid and invalid pointer usage in C.

(1)argv的数据类型是指向字符串数组的指针

(2)printf("char d = %c\n", d);输出了d的值，即pc指向的地址存储的值，即c[0]

(3)pcp是一个指向字符的指针的指针，本质上是地址；argv是指向数组的指针，也是地址，二者可以赋值

(4)pcc2 is a const pointer to char，是一个字符的常量指针

(5)pcc指向的值是一个字符常量，不能改变常量的值，因此*pcc='7'无效

(6)pcc是一个指向字符的指针，pcp是指向字符的指针的指针，则*pcp是指向字符的指针，赋值有效

(7)pcc是一个指向字符的指针，argv[0]是指向数组第一个元素的地址，也是字符串第一个字符的指针，赋值有效

(8)cp是一个常量指针，不可以修改值

(9)cp是一个常量指针，不可以修改值

(10)cp指向的字符可以修改

(11)cpc是一个指向常量字符的常量指针，只要是常量都不可以修改，因此最后三行赋值语句都无效

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.

使用的代码如下：

```
// Copyright (c) 2012 MIT License by 6.172 Staff

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#define PRINT_SIZE(str,a) printf("size of %s : %zu bytes \n", str, sizeof(a));

int main() {
  int a;
  short b;
  long c;
  char d;
  float e;
  double f;
  unsigned int g;
  long long h;
  uint8_t i;
  uint16_t j;
  uint32_t k;
  uint64_t l;
  uint_fast8_t m;
  uint_fast16_t n;
  uintmax_t o;
  intmax_t p;
  __int128 q;
  // Please print the sizes of the following types:
  // int, short, long, char, float, double, unsigned int, long long
  // uint8_t, uint16_t, uint32_t, and uint64_t, uint_fast8_t,
  // uint_fast16_t, uintmax_t, intmax_t, __int128, and student

  // Here's how to show the size of one type. See if you can define a macro
  // to avoid copy pasting this code.
  //printf("size of %s : %zu bytes \n", "int", sizeof(int));
  // e.g. PRINT_SIZE("int", int);
  //      PRINT_SIZE("short", short);
  PRINT_SIZE("int", &a);
  PRINT_SIZE("short", &b);
  PRINT_SIZE("long", &c);
  PRINT_SIZE("char", &d);
  PRINT_SIZE("float", &e);
  PRINT_SIZE("double", &f);
  PRINT_SIZE("unsigned int", &g);
```

```
  PRINT_SIZE("long long", &h);
  PRINT_SIZE("uint8_t", &i);
  PRINT_SIZE("uint16_t", &j);
  PRINT_SIZE("uint32_t", &k);
  PRINT_SIZE("uint64_t", &l);
  PRINT_SIZE("uint_fast8_t m", &m);
  PRINT_SIZE("uint_fast16_t m", &n);
  PRINT_SIZE("uintmax_t", &o);
  PRINT_SIZE("intmax_t", &p);
  PRINT_SIZE("__int128", &q);
  // Alternatively, you can use stringification
  // (https://gcc.gnu.org/onlinedocs/cpp/Stringification.html) so that
  // you can write
  // e.g. PRINT_SIZE(int);
  //      PRINT_SIZE(short);

  // Composite types have sizes too.
  typedef struct {
    int id;
    int year;
  } student;

  student you;
  you.id = 12345;
  you.year = 4;


  // Array declaration. Use your macro to print the size of this.
  int x[5];

  // You can just use your macro here instead: PRINT_SIZE("student", you);
  //printf("size of %s : %zu bytes \n", "student", sizeof(you));
    PRINT_SIZE("student", &you);
    PRINT_SIZE("int[]", &x);
  return 0;
}
```

```
zhangkeer@zhangkeer-virtual-machine:~$ make sizes && ./sizes
clang -Wall -O1 -DNDEBUG  -c sizes.c
clang -o sizes sizes.o -lrt -flto -fuse-ld=gold
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
size of int* : 8 bytes
size of short* : 8 bytes
size of long* : 8 bytes
size of char* : 8 bytes
size of float* : 8 bytes
size of double* : 8 bytes
size of unsigned int* : 8 bytes
size of long long* : 8 bytes
size of uint8_t* : 8 bytes
size of uint16_t* : 8 bytes
size of uint32_t* : 8 bytes
size of uint64_t* : 8 bytes
size of uint_fast8_t* : 8 bytes
size of uint_fast16_t* : 8 bytes
size of uintmax_t* : 8 bytes
size of intmax_t* : 8 bytes
size of __int128* : 8 bytes
size of student* : 8 bytes
size of &x : 8 bytes
zhangkeer@zhangkeer-virtual-machine:~$
```

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.

修改后的代码：

```c
// Copyright (c) 2012 MIT License by 6.172 Staff

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

void swap(int* i, int* j) {
  int temp = *i;
  *i = *j;
  *j = temp;
}
```

```c
int main() {
  int k = 1;
  int m = 2;
  swap(&k, &m);
  // What does this print?
  printf("k = %d, m = %d\n", k, m);

  return 0;
}
```

运行结果：

```
zhangkeer@zhangkeer-virtual-machine:~$ make swap
cc      swap.c    -o swap
zhangkeer@zhangkeer-virtual-machine:~$ ./swap
k = 2, m = 1
zhangkeer@zhangkeer-virtual-machine:~$
```

运行脚本verifier.py：

```
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 uint32_t : 4 bytes
size of uint64_t : 8 bytes
size of student : 8 bytes
size of x : 20 bytes
size of int* : 8 bytes
size of short* : 8 bytes
size of long* : 8 bytes
size of char* : 8 bytes
size of float* : 8 bytes
size of double* : 8 bytes
size of unsigned int* : 8 bytes
size of long long* : 8 bytes
size of uint8_t* : 8 bytes
size of uint16_t* : 8 bytes
size of uint32_t* : 8 bytes
size of uint64_t* : 8 bytes
size of uint_fast8_t* : 8 bytes
size of uint_fast16_t* : 8 bytes
size of uintmax_t* : 8 bytes
size of intmax_t* : 8 bytes
size of __int128* : 8 bytes
size of uint32_t* : 8 bytes
size of uint64_t* : 8 bytes
size of student* : 8 bytes
size of &x : 8 bytes
Ok!

Running make pointer ...
Ok!

Checking that pointer was built ...
Ok!

Running make swap ...
Ok!

Checking that swap was built ...
Ok!
Checking output of swap ...
Ok!
LGTM
zhangkeer@zhangkeer-virtual-machine:~$
```

Write-up 5: Now, what do you see when you type make clean; make?

修改前：

```
zhangkeer@zhangkeer-virtual-machine:~/a1/matrix$ make
clang -O1 -DNDEBUG -Wall -std=c99 -D_POSIX_C_SOURCE=200809L -c testbed.c -o testbed.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
zhangkeer@zhangkeer-virtual-machine:~/a1/matrix$
```

修改后:

```
zhangkeer@zhangkeer-virtual-machine:~/a1/matrix$ make clean;make
rm -f testbed.o matrix_multiply.o matrix_multiply .buildmode \
        testbed.gcda matrix_multiply.gcda \
        testbed.gcno matrix_multiply.gcno \
        testbed.c.gcov matrix_multiply.c.gcov fasttime.h.gcov
clang -O3 -DNDEBUG -Wall -std=c99 -D_POSIX_C_SOURCE=200809L -c testbed.c -o testbed.o
clang -O3 -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
zhangkeer@zhangkeer-virtual-machine:~/a1/matrix$
```

Write-up 6: What output do you see from AddressSanitizer regarding the memory bug? Paste it into your writeup here.

```
zhangkeer@zhangkeer-virtual-machine:~/a1/matrix$ make clean
rm -f testbed.o matrix_multiply.o matrix_multiply .buildmode \
        testbed.gcda matrix_multiply.gcda \
        testbed.gcno matrix_multiply.gcno \
        testbed.c.gcov matrix_multiply.c.gcov fasttime.h.gcov
zhangkeer@zhangkeer-virtual-machine:~/a1/matrix$ make ASAN=1
clang -O1 -g -fsanitize=address -Wall -std=c99 -D_POSIX_C_SOURCE=200809L -c testbed.c -o testbed.o
clang -O1 -g -fsanitize=address -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 -fsanitize=address
/usr/bin/ld.gold: warning: Cannot export local symbol '__asan_extra_spill_area'
zhangkeer@zhangkeer-virtual-machine:~/a1/matrix$ ./matrix_multiply
Setup
Running matrix_multiply_run()...
Elapsed execution time: 0.000001 sec

=================================================================
==20787==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 48 byte(s) in 3 object(s) allocated from:
    #0 0x494b2d in malloc (/home/zhangkeer/a1/matrix/matrix_multiply+0x494b2d)
    #1 0x4c4ed9 in make_matrix /home/zhangkeer/a1/matrix/matrix_multiply.c:39:24
    #2 0x7f33a1d470b2 in __libc_start_main /build/glibc-eX1tMB/glibc-2.31/csu/../csu/libc-start.c:308:16

Indirect leak of 192 byte(s) in 12 object(s) allocated from:
    #0 0x494b2d in malloc (/home/zhangkeer/a1/matrix/matrix_multiply+0x494b2d)
    #1 0x4c4f67 in make_matrix /home/zhangkeer/a1/matrix/matrix_multiply.c:48:35

Indirect leak of 96 byte(s) in 3 object(s) allocated from:
    #0 0x494b2d in malloc (/home/zhangkeer/a1/matrix/matrix_multiply+0x494b2d)
    #1 0x4c4f20 in make_matrix /home/zhangkeer/a1/matrix/matrix_multiply.c:46:31
    #2 0x7f33a1d470b2 in __libc_start_main /build/glibc-eX1tMB/glibc-2.31/csu/../csu/libc-start.c:308:16

SUMMARY: AddressSanitizer: 336 byte(s) leaked in 18 allocation(s).
zhangkeer@zhangkeer-virtual-machine:~/a1/matrix$
```

Write-up 7: After you fix your program, run ./matrix_multiply -p. Paste the program output showing that the matrix multiplication is working correctly.

```
zhangkeer@zhangkeer-virtual-machine:~/a1/matrix$ ./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:
------------
   47      55     122     130
   79      83     138     164
   74      40      75     114
  130      95      74     144
------------
---- END RESULTS ----
Elapsed execution time: 0.000000 sec
zhangkeer@zhangkeer-virtual-machine:~/a1/matrix$
```

Write-up 8: Paste the output from Valgrind showing that there is no error in your program.

```
zhangkeer@zhangkeer-virtual-machine:~/a1/matrix$ make DEBUG=1
clang -g -DDEBUG -O0 -Wall -std=c99 -D_POSIX_C_SOURCE=200809L -c testbed.c -o testbed.o
clang -g -DDEBUG -O0 -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
zhangkeer@zhangkeer-virtual-machine:~/a1/matrix$ valgrind --leak-check=full ./matrix_multiply -p
==20921== Memcheck, a memory error detector
==20921== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==20921== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==20921== Command: ./matrix_multiply -p
==20921==
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: 0.000879 sec
==20921==
==20921== HEAP SUMMARY:
==20921==     in use at exit: 0 bytes in 0 blocks
==20921==   total heap usage: 39 allocs, 39 frees, 1,680 bytes allocated
==20921==
==20921== All heap blocks were freed -- no leaks are possible
==20921==
==20921== For lists of detected and suppressed errors, rerun with: -s
==20921== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
zhangkeer@zhangkeer-virtual-machine:~/a1/matrix$
```