Homework 1: Getting Started

杨浩然 10195501441

操作系统：ubuntu 20.04
编译器：gcc 9.3.0
IDE：Visual Studio Code

2021-9-6

Write-up 2

执行 `make pointer` 后，编译器报出如下错误：

```
younghojan@younghojan-XPS-15-7590:~/学习/软件系统优化/homework$ make pointer
cc     pointer.c   -o pointer
pointer.c: In function ‘main’:
pointer.c:12:5: warning: implicit declaration of function ‘printf’ [-Wimplicit-function-declaration]
   12 |     printf("char d = %c\n", d); // What does this print?
      |     ^~~~~~
pointer.c:12:5: warning: incompatible implicit declaration of built-in function ‘printf’
pointer.c:1:1: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’
  +++ |+#include <stdio.h>
    1 | int main(int argc, char *argv[]) {  // What is the type of argv?
pointer.c:24:10: error: assignment of read-only location ‘*pcc’
   24 |     *pcc = '7';  // invalid?
      |          ^
pointer.c:30:8: error: assignment of read-only variable ‘cp’
   30 |     cp = *pcp;  // invalid?
      |        ^
pointer.c:31:8: error: assignment of read-only variable ‘cp’
   31 |     cp = *argv; //invalid?
      |        ^
pointer.c:36:9: error: assignment of read-only variable ‘cpc’
   36 |     cpc = *pcp; // invalid?
      |         ^
pointer.c:37:9: error: assignment of read-only variable ‘cpc’
   37 |     cpc = argv[0];  // invalid?
      |         ^
pointer.c:38:10: error: assignment of read-only location ‘*cpc’
   38 |     *cpc = '@'; // invalid?
      |          ^
make: *** [<内置>: pointer] 错误 1
```

line 12: What does this print?

这里的语句是 `printf("char d = %c\n", d);`，输出结果为 `char d = 6`。
由 `char c[] = 6.172` 知，`c` 为指向数组第一个字符 `6` 的指针。`char *pc = c;` 声明一个 `char` 指针，它的值与 `c` 相等。`char d = *pc;` 声明了一个 `char` 型变量 `d`，使用解引用运算符 `*` 获取 `pc` 指向的对象，也就是 `c` 指向的对象即字符 `6`，并将其赋给 `d`。综上，`d` 的值为 `6`，类型为 `char`，该语句的输出结果为 `char d = 6`。

line 18: Why is this assignment valid?

这里的语句是 `pcp = argv;`。
首先看 `pcp` 和 `argv` 的声明。`pcp` 的声明类型为 `char **pcp`，是一个 `char` 二级指针。`pcp` 保存一级指针 `char *` 的地址（指向一级指针）。`argv` 的声明类型为 `char *argv[]`，由于 `[]` 的优先级高于 `*`，所以 `argv` 先和 `[]` 结合。`argv` 是一个数组，数组中的元素是 `char *`。
`argv` 为数组元素的首地址，和 `pcp` 一样保存的是一级指针 `char *` 的地址（指向一级指针），因此可以将 `argv` 赋给 `pcp`。

line 21: What is the type of pcc2?

这里的语句是 `char const *pcc2 = c;` 。
`pcc2` 是一个指向字符的指针常量。

line 24: invalid?

这里的语句是 `*pcc = '7';` 。
首先，`pcc` 是这样定义的：`const char *pcc = c` ，即定义了一个指向字符常量的指针 `pcc` 。这里 `pcc` 是一个指向 `char*` 类型的常量，不能用 `pcc` 来修改所指向的内容。因此 `*pcc = '7'` 是 invalid 的。

line 25: valid?

这里的语句是 `pcc = *pcp;` 。
尽管 `pcc` 是一个指向字符常量的指针，但其本身并非常量，所以可以通过重新赋值给 `pcc` 来修改所指向的值。而 `pcp` 是一个 `char` 二级指针，使用解引用运算符 `*` 获取 `pcp` 指向的对象，即是 `char *` ，与 `pcc` 类型相同，可以将其赋给 `pcc` 。

line 26: valid?

这里的语句是 `pcc = argv[0];` 。
原因同 line 25。

line 30: invalid?

这里的语句是 `cp = *pcp;`
首先，`cp` 是这样定义的：`char* const cp = c;` ，即定义了一个指向字符的指针常量 `cp` ，即 const 指针。修改 `cp` 的值是非法的，但可以修改 `cp` 指向的内容。

line 31: invalid?

这里的语句是 `cp = *argv;`
原因同上。

line 32: valid?

这里的语句是 `*cp = '!';`
即使 `cp` 是一个指向字符的指针常量，无法修改 `cp` 的值，但可以通过 `*` 修改其指向的内容。

line 36: invalid?

这里的语句是 `cpc = *pcp;`
首先，`cpc` 是这样定义的：`const char* const cpc = c;` ，这可以看作 `const char*` 和 `char* const` 的结合，即定义了一个指向字符常量的指针常量 `cpc` ，所以修改 `cpc` 的值是非法的。注意到，在这种情况下，修改 `cpc` 指向的内容也是非法的。

line 37: invalid?

这里的语句是 `cpc = *argv;`
原因同上。

line 38: invalid?

这里的语句是 `*cpc = '@';`
前面已经说过，用 `const char* const` 声明 `cpc`，这种情况下利用 `*` 修改 `cpc` 指向的内容也是非法的。

注释掉 invalid 的语句，重新编译，结果如下：

```
younghojan@younghojan-XPS-15-7590:~/学习/软件系统优化/homework$ make pointer
cc     pointer.c   -o pointer
pointer.c: In function ‘main’:
pointer.c:12:5: warning: implicit declaration of function ‘printf’ [-Wimplicit-function-declaration]
   12 |     printf("char d = %c\n", d); // What does this print?
      |     ^~~~~~
pointer.c:12:5: warning: incompatible implicit declaration of built-in function ‘printf’
pointer.c:1:1: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’
  +++ |+#include <stdio.h>
    1 | int main(int argc, char *argv[]) {  // What is the type of argv?
```

Write-up 3

编写宏来打印大小：

```
// My macro here.
#define PRINT_SIZE(type) printf("size of %s : %zu bytes\n", #type, sizeof(type));
```

输出结果如下：

```
younghojan@younghojan-XPS-15-7590:/media/younghojan/Study/undergraduate/junior/SEM1/软件系统优化/homework/hw1/MIT6_172F18_hw1/c-primer$ make sizes
clang -Wall -O1 -DNDEBUG  -c sizes.c
clang -o sizes sizes.o -lrt -flto -fuse-ld=gold
younghojan@younghojan-XPS-15-7590:/media/younghojan/Study/undergraduate/junior/SEM1/软件系统优化/homework/hw1/MIT6_172F18_hw1/c-primer$ ./sizes
size of int : 4 bytes
size of student : 8 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 long : 8 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 long* : 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
```

Write-up 4

代码作如下修改：

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

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

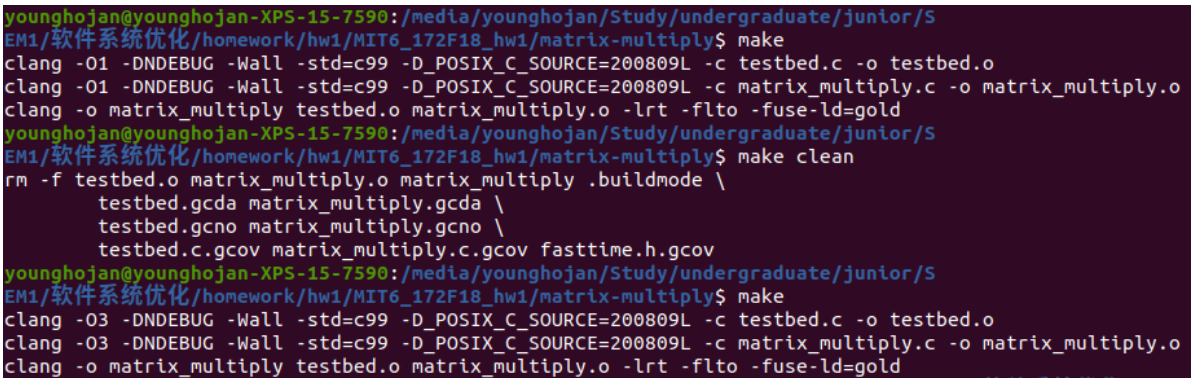
```c
#include <stdint.h>

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

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 `verifier.py`，校验通过。

需要注意的是，`verifier.py` 在 Python3 中无法运行，可以在 Anaconda 中创建一个安装 Python2 的虚拟环境，再运行 `verifier.py`。

Write-up 5

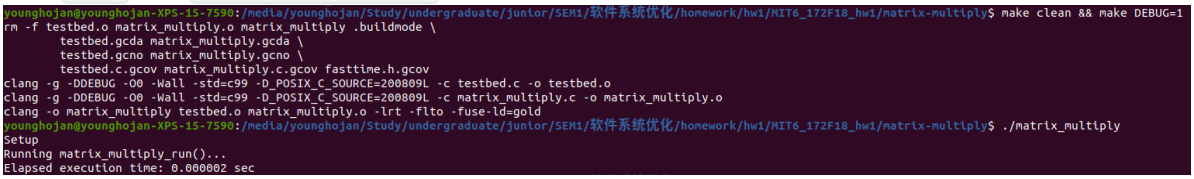
将 `CFLAGS_RELEASE` 项修改为 `-O3 -DNDEBUG`，于是程序将在 O3 条件下被编译，如图。



Write-up 6

fix `testbed.c`：
将 `A = make_matrix(kMatrixSize, kMatrixSize+1);` 改为 `A = make_matrix(kMatrixSize, kMatrixSize);` 即可。

重新 `make` 后，`./matrix_multiply` 的结果为：



使用 `make ASAN=1` 编译，运行结果如下：

Write-up 7

这里需要做的是在 `matrix_multiply.c` 中为矩阵分配空间时，对内存进行初始化。注意到，分配空间的代码为如下段：

```
    // Allocate a buffer big enough to hold the matrix.
    new_matrix->values = (int**)malloc(sizeof(int*) * rows);
    for (int i = 0; i < rows; i++) {
      new_matrix->values[i] = (int*)malloc(sizeof(int) * cols);
    }
```

`malloc()` 是不对分配的内存进行初始化的，这里初始化的方法不止一种，比如使用 `memset()` 进行初始化。我这里选择用 `calloc()` 替换 `malloc()` 。

`void *calloc(size_t nitems, size_t size)` 分配所需的内存空间，并返回一个指向它的指针。`malloc()` 和 `calloc()` 之间的不同点是，`malloc()` 不会设置内存为零，而 `calloc()` 会设置分配的内存为零。

此时，矩阵乘法的结果正确。

```
younghojan@younghojan-XPS-15-7590:/media/younghojan/Study/undergraduate/junior/SEM1/软件系统优化/homework/hw1/MIT6_172F18_hw1/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:
------------
   47     55    122    130
   79     83    138    164
   74     40     75    114
  130     95     74    144
------------
---- END RESULTS ----
Elapsed execution time: 0.000001 sec
younghojan@younghojan-XPS-15-7590:/media/younghojan/Study/undergraduate/junior/SEM1/软件系统优化/homework/hw1/MIT6_172F18_hw1/matrix-multiply$ ./matrix_multiply -pz
Setup
Matrix A:
------------
    0      0      0      0
    0      0      0      0
    0      0      0      0
    0      0      0      0
------------
Matrix B:
------------
    0      0      0      0
    0      0      0      0
    0      0      0      0
    0      0      0      0
------------
Running matrix_multiply_run()...
---- RESULTS ----
Result:
------------
    0      0      0      0
    0      0      0      0
    0      0      0      0
    0      0      0      0
------------
---- END RESULTS ----
Elapsed execution time: 0.000001 sec
```

Write-up 8

使用 `free_matrix()` 对矩阵的内存进行释放，再使用 Valgrind 得到以下输出。此时 Valgrind 不返回错误信息。

```
younghojan@younghojan-XPS-15-7590:/media/younghojan/Study/undergraduate/junior/SEM1/软件系统优化/homework/hw1/MIT6_172F18_hw1/matrix-multiply$ valgrind --leak-check=full ./matrix_multiply -p
==9665== Memcheck, a memory error detector
==9665== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==9665== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==9665== Command: ./matrix_multiply -p
==9665==
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: 0.000599 sec
==9665==
==9665== HEAP SUMMARY:
==9665==     in use at exit: 0 bytes in 0 blocks
==9665==   total heap usage: 39 allocs, 39 frees, 1,680 bytes allocated
==9665==
==9665== All heap blocks were freed -- no leaks are possible
==9665==
==9665== For lists of detected and suppressed errors, rerun with: -s
==9665== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```