

## Unit 0

2014302580341 卓越二班 余璞轩

### Exercise1

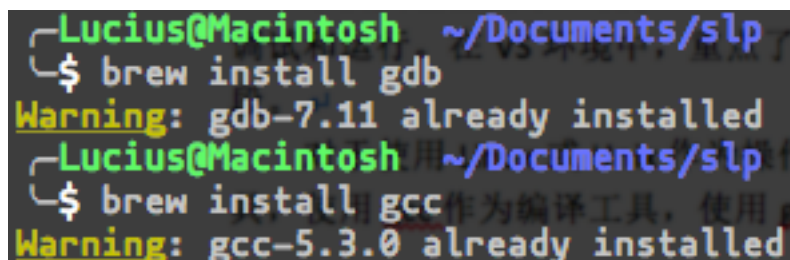
1983 年，美国国家标准化组织（ANSI）成立了 C 语言工作小组，开始了 C 语言的标准化工作。小组所处理的主要事务是确认 C 语言的常用特性，但对语言本身也做了一些修改。尽管当时世界上大约有 5000 万台 PC，而且它是当时应用范围最广的 C 语言实现平台，但标准仍然认为不应该通过修改语言来处理某个特定平台所存在的限制。

### Exercise2

Mac 系统上采用 homebrew 的方式安装 gcc 和 gdb。

```
$ brew install gcc
```

```
$ brew install gdb
```



```
Lucius@Macintosh ~/Documents/slp  
$ brew install gdb  
Warning: gdb-7.11 already installed  
Lucius@Macintosh ~/Documents/slp  
$ brew install gcc  
Warning: gcc-5.3.0 already installed
```

## Exercise3

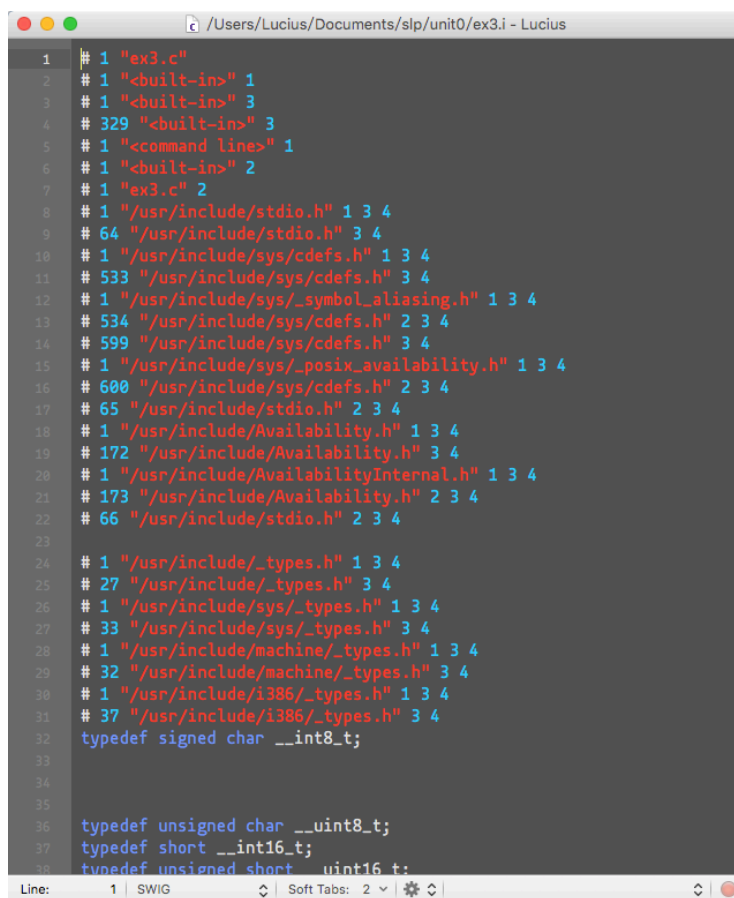
原文件 ex3.c:

```
1  #include <stdio.h>
2
3  int main (int argc, const char* argv[]) {
4      printf("Goodbye, Dannis!\n");
5      return 0;
6  }
```

用 gcc 将其预处理为 ex3.i 文件

```
$ gcc -E ex3.c -o ex3.i
```

得到的 ex3.i 文件: (节选)



```
1  # 1 "ex3.c"
2  # 1 "<built-in>" 1
3  # 1 "<built-in>" 3
4  # 329 "<built-in>" 3
5  # 1 "<command line>" 1
6  # 1 "<built-in>" 2
7  # 1 "ex3.c" 2
8  # 1 "/usr/include/stdio.h" 1 3 4
9  # 64 "/usr/include/stdio.h" 3 4
10 # 1 "/usr/include/sys/cdefs.h" 1 3 4
11 # 533 "/usr/include/sys/cdefs.h" 3 4
12 # 1 "/usr/include/sys/_symbol_aliasing.h" 1 3 4
13 # 534 "/usr/include/sys/cdefs.h" 2 3 4
14 # 599 "/usr/include/sys/cdefs.h" 3 4
15 # 1 "/usr/include/sys/_posix_availability.h" 1 3 4
16 # 600 "/usr/include/sys/cdefs.h" 2 3 4
17 # 65 "/usr/include/stdio.h" 2 3 4
18 # 1 "/usr/include/Availability.h" 1 3 4
19 # 172 "/usr/include/Availability.h" 3 4
20 # 1 "/usr/include/AvailabilityInternal.h" 1 3 4
21 # 173 "/usr/include/Availability.h" 2 3 4
22 # 66 "/usr/include/stdio.h" 2 3 4
23
24 # 1 "/usr/include/_types.h" 1 3 4
25 # 27 "/usr/include/_types.h" 3 4
26 # 1 "/usr/include/sys/_types.h" 1 3 4
27 # 33 "/usr/include/sys/_types.h" 3 4
28 # 1 "/usr/include/machine/_types.h" 1 3 4
29 # 32 "/usr/include/machine/_types.h" 3 4
30 # 1 "/usr/include/i386/_types.h" 1 3 4
31 # 37 "/usr/include/i386/_types.h" 3 4
32 typedef signed char __int8_t;
33
34
35
36 typedef unsigned char __uint8_t;
37 typedef short __int16_t;
38 typedef unsigned short __uint16_t;
```

用 gcc 将 ex3.i 编译为 ex3.s

```
$ gcc -S ex3.i -o ex3.s
```

得到的 ex3.s 文件



```
1 | .section __TEXT,__text,regular,pure_instructions
2 | .macosx_version_min 10, 11
3 | .globl _main
4 | .align 4, 0x90
5 | _main:                                ## @main
6 | .cfi_startproc
7 | ## BB#0:
8 | pushq %rbp
9 | Ltmp0:
10 | .cfi_def_cfa_offset 16
11 | Ltmp1:
12 | .cfi_offset %rbp, -16
13 | movq %rsp, %rbp
14 | Ltmp2:
15 | .cfi_def_cfa_register %rbp
16 | subq $32, %rsp
17 | leaq L_.str(%rip), %rax
18 | movl $0, -4(%rbp)
19 | movl %edi, -8(%rbp)
20 | movq %rsi, -16(%rbp)
21 | movq %rax, %rdi
22 | movb $0, %al
23 | callq _printf
24 | xorl %ecx, %ecx
25 | movl %eax, -20(%rbp)      ## 4-byte Spill
26 | movl %ecx, %eax
27 | addq $32, %rsp
28 | popq %rbp
29 | retq
30 | .cfi_endproc
31 |
32 | .section __TEXT,__cstring,cstring_literals
33 | L_.str:                    ## @.str
34 | .asciz "Goodbye, Dannis!\n"
35 |
36 |
37 | .subsections via symbols
```

用 gcc 将 ex3.s 汇编成 ex3.o

```
$ gcc -c ex3.s -o ex3.o
```

最后进行链接

```
$ gcc ex3.o -o ex3.exe
```

## Exercise4

1. 在赋值 `ccp = cp` 中, 左操作数是一个指向有 `const` 限定符的 `char` 的指针, 右操作数是一个指向没有限定符的 `char` 的指针。左边具有右边的全部限定符 (无), 所以可以赋值。反之则不可以。

```
/*test 1*/  
char* cp;  
const char* ccp;  
ccp = cp;  
//cp = ccp;
```

2. `limit = 20` 不可以编译通过, 因为 `limit` 被声明为 `const`, 所以 `limit` 这个符号不能被赋值。

`limitp = &i` 可以编译通过, 因为指针本身的值是可以改变的。

```
const int limit = 10;  
const int* limitp = &limit;  
int i = 27;  
limitp = &i;  
//limit = 20;
```

3. 赋值操作失败, 因为 `char**` 是指向指向 `char` 指针的指针, 无限定符; 而 `const char**` 是指向指向 `const char` 的指针的指针, 同样无限定符。所以两边都有限定符不可以传值。
4. `sizeof()` 函数返回无符号数, 所以 `TOTAL_ELEMENTS` 定义的值是 `unsigned int` 类型。If 语句中 `-1` 与一个无符号数做比较, `-1` 也变成无符号整型, 变得巨大使条件为否。

5. case 语句后没有接 break 语句，导致 case2 执行后将后面的语句一并执

行。[C 语言将 fall through 作为 switch 的缺省行为是一个失误。]

```
switch(2) {  
    case 1: printf("case 1.\n");  
    case 2: printf("case 2.\n");  
    case 3: printf("case 3.\n");  
    case 4: printf("case 4.\n");  
    default: printf("default.\n");  
}
```

6. 原意是想让 break 语句跳出，但事实上 break 跳出的是最近的循环语句或

者 switch 语句，所以导致初始化工作没有进行，导致后面的故障。

```
network code() {  
    switch(line) {  
        case THING1: { doit1(); break; }  
        case THING2: {  
            if (x == STUFF) {  
                do_first_stuff();  
                if (y == OTHER_STUFF) { break; }  
                do_later_stuff();  
            } /* meant to jump to here */  
            initialize_modes_pointer();  
            break;  
        default: { processing(); }  
        } /* but it jumped to here */  
        use_modes_pointer(); /* but modes_pointer wasn't  
initialized */  
    }  
}
```

7. 这段代码利用了静态变量只初始化一次的特性，后续的初始化均没有效果，

所以可以实现出类似：string1, string2, string3, …等的字符串。

```
generate_initializer(char* string) {  
    static char separator = ' ';  
    printf("%c %s \n", separator, string);  
    separator = ',';  
}
```

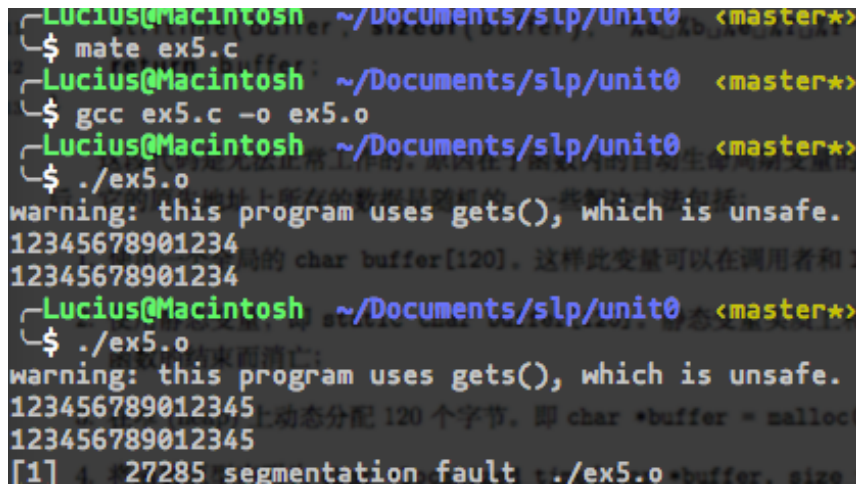
## Exercise5

1.

编写的测试代码:

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

int main() {
    char string[3];
    gets(string);
    puts(string);
    return 0;
}
```



```
Lucius@Macintosh ~/Documents/slp/unit0 <master*>
$ mate ex5.c
Lucius@Macintosh ~/Documents/slp/unit0 <master*>
$ gcc ex5.c -o ex5.o
Lucius@Macintosh ~/Documents/slp/unit0 <master*>
$ ./ex5.o
warning: this program uses gets(), which is unsafe.
12345678901234
12345678901234
Lucius@Macintosh ~/Documents/slp/unit0 <master*>
$ ./ex5.o
warning: this program uses gets(), which is unsafe.
123456789012345
123456789012345
[1] 27285 segmentation fault ./ex5.o
```

使用 gets 函数的时候, 输入 14 个字符不会出现问题, 而输入 15 个字符就会导致程序出现中断。原因是 gets() 函数不检查缓冲区的空间, 多出来的字符覆盖了原先的内容。

2.

```
/* Convert the source file timestamp into a localized
date string */
char * localized_time(char * filename)
{
    struct tm *tm_ptr;
    struct stat stat_block;
    char buffer[120];
    /* get the sourcefile's timestamp in time_t format */
    stat(filename, &stat_block);
    /* convert UNIX time_t into a struct tm holding local
time */
    tm_ptr = localtime(&stat_block.st_mtime);}
    /* convert the tm struct into a string in local format */
    strftime(buffer, sizeof(buffer), "%a %b %e %T %Y", tm_ptr);
    return buffer;
}
```

最后一行 `return buffer` 其实是返回了一个自动分配内存的数组，是函数的局部变量。函数结束时变量被销毁，无法确定该指针指向的地址的内容是什么。

解决方法有：

- a. 使用全局声明的数组，但是任何人有可能修改这个全局数组。
- b. 使用静态数组。函数的下一次调用会覆盖这个数组，而且浪费内存空间。
- c. 显式分配空间，保存返回值，但是需要手动去管理内存。
- d. 分配内存来保存函数返回值，并且用类似 `fgets()` 的方法来指定缓冲区大小。
- e. 返回字符串常量。