**Unit 0**

**2014302580341 卓越二班 余璞轩**

**Exercise1**

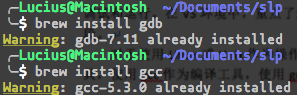
1983年，美国国家标准化组织（ANSI）成立了C语言工作小组，开始了C语言的标准化工作。小组所处理的主要事务是确认C语言的常用特性，但对语言本身也做了一些修改。尽管当时世界上大约有5000万台PC，而且它是当时应用范围最广的C语言实现平台，但标准仍然认为不应该通过修改语言来处理某个特定平台所存在的限制。

**Exercise2**

Mac系统上采用homebrew 的方式安装gcc 和 gdb。

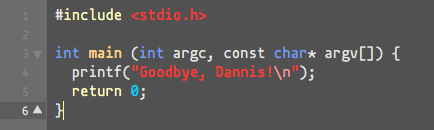
$ brew install gcc

$ brew install gdb



**Exercise3**

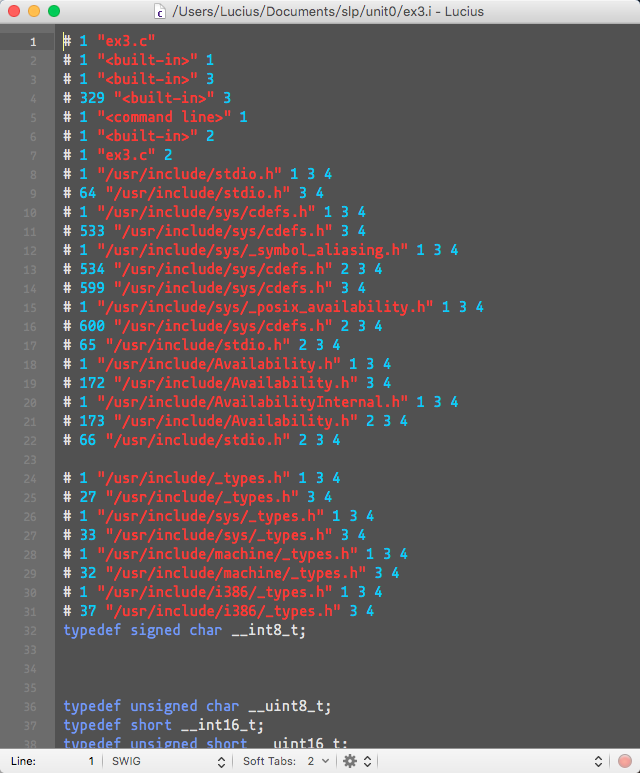
原文件ex3.c:



用gcc将其预处理为ex3.i文件

$ gcc –E ex3.c –o ex3.i

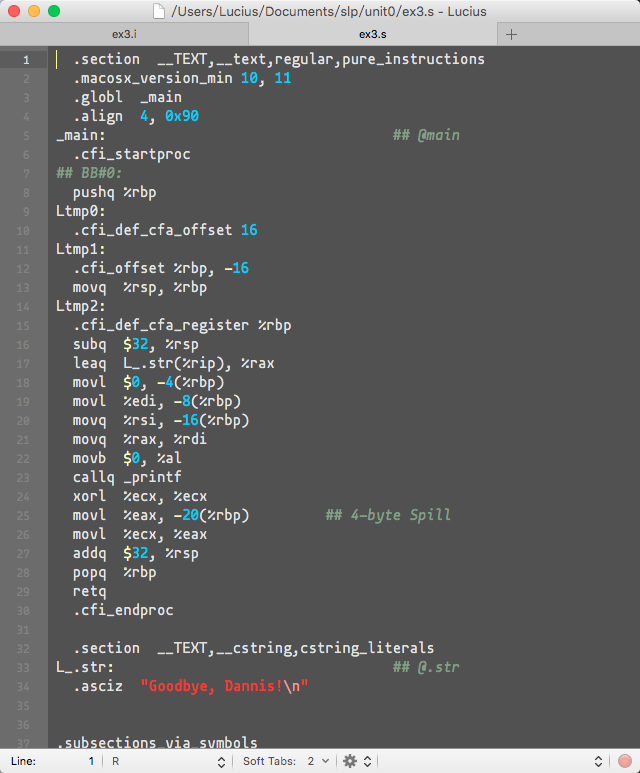
得到的ex3.i文件：（节选）



用gcc将ex3.i 编译为 ex3.s

$ gcc –S ex3.i –o ex3.s

得到的ex3.s文件



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;

1. 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;

1. 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。所以两边都没有限定符不可以传值。
2. 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。
3. 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");

}

1. 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 \*/

}

}

1. 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**

编写的测试代码：

#include <stdio.h>

#include <stdlib.h>

int main() {

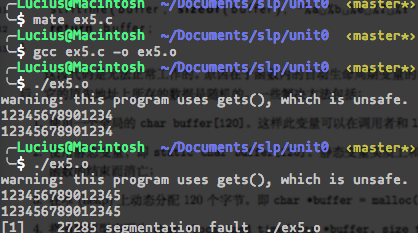
char string[3];

gets(string);

puts(string);

return 0;

}



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。

/\* 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：

1. 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。
2. 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。
3. 显式分配空间，保存返回值，但是需要手动去管理内存。
4. 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。
5. 返回字符串常量。