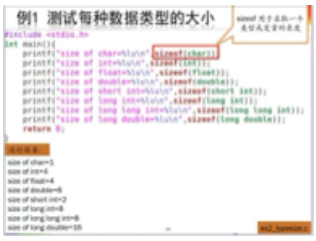
TCPL学习问题和解答2

问题：快排序需要用n次比较才能把数组分成两部分，只有正好把数组平分时才最快，最佳复杂度为nlogn，最坏复杂度为n^2，期望复杂度应该介于二者之间，为什么也是nlogn?

有点复杂。参见算法教材的证明。

问题：下面这样写太麻烦了，怎么定义一个类似数组的宏，把每种数据类型依次替换到每个 printf 中?



这里有个例子：GitHub.com/zedware/notebook/tree/master/samples/tcpl/src/misc/sizeof.c

// To see the expanded macros:

```
//   $ gcc -E sizeof.c
// To compile the program:
//   $ gcc -Wall sizeof.c
#include <stdio.h>

// 1. #x can concat strings
// 2. %lu for unsigned long int (aka. size_t)
// 3. \ to join the lines
#define SZ(x)  { \
    printf("sizeof(" #x ") = %lu\n", sizeof(x)); \
  }

void foo(void) {
    SZ(char);
    SZ(short);
    SZ(int);
    SZ(long);
    SZ(float);
    SZ(double);
}

// Count the number of elements in an array
x.
#define countof(x)  (sizeof(x)/sizeof(x[0]))

// Not work as expected.
```

```
void bar(void) {
    const char *types[] = {
        "char",
        "short",
        "int",
        "long",
        "float",
        "double"
    };

    // 1. sizeof(types) is the bytes of the array
types[].
    // 2. sizeof(types[0]) is the bytes of an
element of array: types[0]
    for (size_t i = 0; i < countof(types); i++) {
        SZ(types[i]);
    }

    // Not worked either.
    for (size_t i = 0; i < countof(types); i++) {
        printf("sizeof(%s) = %lu\n", types[i],
sizeof(types[i]));
    }
}

int main(void) {
```

```
    foo();
    bar();
    return 0;
}
```

注意：

1. 宏 SZ 可以起到减少重复代码的作用，但是还是需要一行行的去写。
2. bar() 函数的写法无效，可以通过观察 gcc -E sizeof.c 的输出理解。
3. sizeof 是个比较特殊的操作符 https://en.cppreference.com/w/c/language/sizeof

问题：不同数据类型之间的转换是截断还是符号位扩展?

关于数据类型之间的转换：https://github.com/zedware/notebook/blob/master/samples/tcpl/src/misc/type_convert.c。代码中有详细的注释。C 语言中规则的精确含义可以直接看汇编代码来准确理解。因为C语言是追求效率的，能省去不做的操作一定会省去；能用一条指令完成的绝不会用两条指令。

问题：编译器对源代码文件都做了哪些处理?

学究一点的说法是词法分析、语法分析、语义检查、中间代码生成、目标代码生成。从实用的角度出发，理解 gcc 的处理对初学者就够用了。

1. 预处理（preprocess）
2. 编译（compile）
3. 汇编（assemble）
4. 链接（link）

其中：

1. gcc -E 只做预处理，主要是处理那些以#打头的语句，包括#include、#define 等。输出的文件还是个 C 源代码。
2. gcc -S 只做预处理和编译，把源代码处理成一个汇编语言写成的文件。
3. gcc -c 会做预处理、编译和汇编，把源代码处理成一个特殊格式的目标文件。
4. 平常大家用的不加上述任何一个命令行开关的 gcc 命令会做预处理、编译、汇编和链接所有的事情，产生一个可执行文件。

例如：

1. gcc -E hello_world.c > hello_world.E
   # 默认输出到 stdout，所有需要重定向到文件
2. gcc -S hello_world.c # 默认输出到 *.s
3. gcc -c hello_world.c # 默认输出到 *.o
4. gcc hello_world.c   # 默认输出到 a.out

gcc 的常用命令行参数：https://gcc.gnu.org/onlinedocs/gcc/Overall-Options.html 。

完整的例子参看：https://github.com/zedware/notebook/tree/master/samples/tcpl/src/compiler 。

问题：scanf("%[^\n]", s) 为什么在循环中行为不符合预期?

同学总结的很好，也发现了一些问题。如果遵循工程化的方法可以节省一些时间，搞得更清楚一些。

1. 优先求之于手册

1）C/C++ 的在线手册

https://www.cplusplus.com/reference/cstdio/scanf/

2）scanf 函数的 man page

```
$ man scanf
<snipped>
RETURN VALUE
  On  success,  these  functions  return the number of input items successfully matched and assigned; this can be fewer than
  provided for, or even zero, in the event of an early matching failure.

  The value EOF is returned if the end of input is reached before either the first successful conversion or a matching fail-
  ure occurs.  EOF is also returned if a read error occurs, in which case the error indicator for the stream (see ferror(3))
  is set, and errno is set to indicate the error.
<snipped>
```

2. 检查函数返回值并做适当的打印输出

下述代码对返回值的判断有问题，所以它会形成一个死循环，每次打印出来的都是第一次输入的那一行。这一段代码因为用了打印语句，比同学给的例子要更直观一点。

```
#include <stdio.h>

int main(void) {
    char s[1024];
    while (scanf("%[^\n]", s) != EOF) {
        printf("s=%s\n", s);
    }
    return 0;
}
```

像上面每次都简单的 printf 一下，就很容易发现它死循环了。从上面 man 的输出看，返回值可能是 EOF 或 0 或一个正数。因此判断返回值的代码应该改成：

```
    while (scanf("%[^\n]", s) > 0) { }
```

下面的代码为了显示返回值的区别，用了一个单独的变量 rc 并且打印出来了。

```
$ cat scanf.c
#include <stdio.h>

int main(void) {
    char s[1024];
    int rc;
    while ((rc = scanf("%[^\n]", s)) != EOF) {
        printf("rc=%d\n", rc);
        printf("s=%s\n", s);
    }
    return 0;
}

$ gcc -Wall scanf.c
$ ./a.out
```

如果输入 abc 回车，输出类似这样：

```
rc=1
s=abc
rc=0
s=abc
rc=0
```

<snipped>

很明显，除了第一次，rc 的返回值都是 0。如果屏幕输出太快，不容易看到最初的几行输出，可以：

$ ./a.out > x

输入abc回车，然后 Ctrl+C 中断程序，再查看文件 x 的内容的前几行：

$ head x

当然也可以用 more 或 less 甚至 vim。

3. 怎么解决这个死循环

从上面的输出可以推断，除了第一次之外的每个 scanf 都没有等待用户输入，它很可能是把 \n 当做输入就直接返回了，而 \n 又不满足格式串，所以返回值为 0 （表示没有匹配到格式串）。所以一种直接的想法是把 \n 消化掉。例如：

$ cat scanf.c

```
#include <stdio.h>

int main(void) {
    char s[1024];
    char c;
    int rc;
    while ((rc = scanf("%[^\n]%c", s, &c)) !=
EOF) {
        printf("rc=%d\n", rc);
        printf("s=%s\n", s);
        printf("c=%d\n", c);
    }
    return 0;
}
```

4. 总结

1) 从学习和做作业的角度看，复用之前的作业代码比较好，因为代码自己很熟悉，不容易搞错。
2) 如果要引用现成的库函数，需要仔细阅读手册，尤其是注意其中的返回值、出错条件、使用限制。
3) 同学的实验和这里用到的都是黑盒测试的方法。实际上，很多软件问题是可以通过源代码获得直接的答案的：

Debugging the Infinite Loop in Scanf.
4) [] 是 scanf 函数的特殊功能，实际应用中用到的很少。%[^\n] 教学中
经常用来作为 gets() 函数的等价表示。但是 gets() 函数因为不能检查
目标字符串 s 的长度，很容易导致 s 的缓冲区溢出，从而引起安全漏洞，
所以这个函数已经被废弃了。上面例子中用 scanf 时候实际也没有检查 s
的长度，也存在类似的问题，因此实际代码也不能这么写。如果输入的字符
串长度大于 1024，程序也可能会异常。以后我们再来学怎么检查这些错误。

5. 补充

Scanf 失败的时候 stdin 里头留下的不仅仅是 \n，而是所有失败的字符。所以需要循环 getchar()，清理所有的字符。这个文档有详细的说明。

问题： for, while, if的特殊用法

for (;;) {} 表示死循环，有些程序中会遇到。
while (1) {} 也表示死循环。

do {} while (0) 表示循环体仅执行一次，常常用在宏定义里头。
#if 0 ... #endif 常常用来临时注释掉一大段代码。