8-pointer 题解

本周均属于简单的编程题。可以先用数组写了再转成指针写法。

A-内存解释器 (interpreter.c)

> 不需要思考小端存储的事，不需要考虑浮点数非数的情况。
> 这是一道简单题，**别想复杂**，想得越复杂，越难做对。
> 也就是说，代码**不应该超过10行**，不然很可能只能拿部分分……别怪我没提醒。
> 当然，如果你足够强，可以化简为繁，挑战100行的写法。

思路是很显然的，不过在那之前，我们可以稍微 dip into 一下 C 的类型系统，尤其是指针类型系统。

常规类型

`int, float, double, long, long long...`

每一种类型都对应了相应的**字长**。如 `int` 字长为 4，`long long` 字长为 8，而 `long` 的字长是机器相关的。在 32 位机上，它是 4 字，而在 64 位机上，它就是 8 字。机器相关的好处就在于，你可以编写可移植的代码。

当你在函数中写下 `int a;` 的时候，编译器可以在程序的栈空间里留下 4 个连续的字节空间用以存放这一整数。

指针类型

`int *, float *, char *...`

在 C 里，所有的指针类型都是机器相关的。指针长度与机器位宽相等，用来指示内存当中的某个位置。

对于 `int *` 变量进行解引用，如对 `int *a` 进行 `*a` 的操作，则 `*a` 的类型会自动还原为 `int`。其他常规类型同理。

`void *`

这种类型没有指定其解引用后的原类型。但它依然是一个指针。

指针类型的强制类型转换

对于任意类型的指针，由于它们的字长是相同的，因此它们之间可以无阻碍地进行强制转换。如对于某个 `int *a`，可以通过 `(double *)a` 来获取一个 `double *` 类型的指针。那么毫无疑问，如果你对这个指针进行解引用，即 `*(double *)a`，则机器会按照 `double` 的类型约定，来解析从指针位置开始的 8 字节作为解引用的值。于是题目就迎刃而解了。

STD

```
#include <stdio.h>
int main(){
    int a;
    scanf("%x", &a);
    printf("%d\n%u\n%f", a, a, *(float*)&a);
}
```

B-子串的位置 (substr.c)

认真阅读 C 库函数的手册，

> NAME
>
> strstr, strcasestr - locate a substring
>
> SYNOPSIS
>
> #include <string.h>
>
> char *strstr(const char *haystack, const char *needle);
>
> DESCRIPTION
>
> The strstr() function finds the first occurrence of the substring needle in the string haystack. The terminating null bytes ('\0') are not compared

.....

现成的

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

int main() {
    char *s = malloc(100005), *t = malloc(100005), *tmp = s;
    scanf("%s%s", s, t);
    while ((tmp = strstr(tmp, t)) != NULL) {
        printf("%d ", tmp - s);
        tmp++;
    }
    return 0;
}
```

如果没有接触过 `strstr()`，那么至少也要想到，把**从字符串里找到一个子串的位置**这个功能封装为一个函数，毕竟要多次调用。

C-搞个大新闻 (title.c)

> 敬爱的江泽民同志永垂不朽

既然说了是逐词以空格分开，那就可以这样的方式读入（一定要学会！）：

```
while ((scanf("%s", s) != EOF)) {
   // process(s);
}
```

要么就在第 9 次作业再吃一次苦

C 还有两个库函数：

```
#include <ctype.h>

int toupper(int c);
int tolower(int c);
```

只需要对第一个无脑 `toupper()`，剩下的全都 `tolower()` 就完事了...

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

void convert(char *s) {
  *s = toupper(*s);
  for (s++; *s != 0; s++)
    *s = tolower(*s);
}

int main() {
  char *s = (char *)malloc(4096);
  while (scanf("%s", s) != EOF) {
    convert(s);
    printf("%s ", s);
  }
  return 0;
}
```