9-more-pointers 题解

> 由战犯 Tilnel 贡献本次题解。
>
> 如果有你没见过的函数（用法），请尽量弄懂它。花10分钟学一下，省掉考场上一点自己造轮子的时间。

A-不太简单的词法分析器 (tokenizer.c)

> 本题数据的特性就是，不存在一个连续的字符串里有多个分号的情况。例如：`return;return;return;return;return;`
>
> 所以对于分号，你可以当作只有：
>
> 1、在开头
> 2、在末尾
> 3、在中间有一个
> 4、没有
> 5、字符串里只有分号
>
> 这几种情形

根据题目注解写的思路：

```
while (scanf("%s", s) != EOF) {
  char *position = strchr(s, ';');   // if there's no ';' in s, it returns NULL.
  if (position == NULL) process(s);
  else {
    process(s 中分号前的部分);
    在输出末尾加个换行;
    process(s 中分号后的部分);
  }
}
```

非常简单地想到我们可以直接逐个处理，并把输出记录到字符串当中。

由于我们要求，要么输出完整的结果，要么只输出 Compile Error，而完整的结果是逐词累积的。如何在一个字符串的末尾去 append 新的内容呢？答案是：`strcat()`，或 `sprintf()`。

> NAME
>
> strcat, strncat - concatenate two strings
>
> SYNOPSIS
>
> #include <string.h>
>
> ```
> char *strcat(char *dest, const char *src);
> char *strncat(char *dest, const char *src, size_t n);
> ```

`strcat()` 的做法是把 src 拼到 dest 的后面。需要 dest 有足够的空间。并且只能操作纯字符串。而 `sprintf()` 可以在指定的位置放置一个格式化输出的字符串。同学们可以自行 RTFM。

那么，我们就开一个大的字符串去存放输出结果，每次更新。如果出错就输出报错，然后直接返回；否则在最后把结果输出。

如何分割字符串？以 "abcd;efgh" 为例：

```
char s[] = "abcd;efgh";
s[4] = 0;
printf("%s, %s\n", s, s + 5);
```

它的输出结果是

```
abcd, efgh
```

> 注意：
>
> ```
> char *s = "abcd;efgh";
> s[4] = 0;
> ```
>
> 这段代码将会给出 segmentation fault。
>
> 在 C 语言中，"" 所给出的字符串字面量都将是常量，也就是说它会放在只读的内存空间中。
> `char *s = "a string";` 的作用，是将这段只读内存空间的指针赋给 `s`。因此你不能重新对 `s` 中的字符进行修改。
> 而 `char s[] = "a string";` 是分配一段可读写的内存空间，并将其中的值初始化为该字符串。

在 C 中，字符串是以 '\0' 结尾的。因此我们只需要让我们所需要的字符串部分的后一位为 '\0'，即可得到字符串的这一局部。因此想要将字符串按照 ';' 割成两段，非常简单：

```
char *semi = strchr(s, ';');   // find position of semicolon
if (semi) {   // strchr() returns NULL if there's no semicolon
  *semi = '\0';
  process(s);
  process(semi + 1);    // what's after semicolon is another part
} else
  process(s);
```

对于 `process()`，我们的要求则是：

- 如果一个词符合五种规则，则追加一条解析
- 如果不符合，则直接输出 "Compile Error" 并退出

直接退出程序：

```
exit(EXIT_SUCCESS);
```

不同于 return，它直接将整个程序正常结束。

那么，剩下最后一件事：解析规则。根据规则，一个字符串只可能属于其中的一种。故对五个规则依次判断。要注意的是，保留字均符合变量的条件，因此要先判出保留字，再去判变量。

保留字和运算符

策略很简单：把所有保留字（运算符）放在一个字符串数组里，碰到以后逐个 `strcmp()`

```
char **reserved = malloc(16 * sizeof(char *));   // char **, 实质上是一个 (char *) 的数组
*(reserved + 0) = "const";
*(reserved + 1) = "int";
...

for (int i = 0; i < 16; i++) {
    if (!strcmp(s, *(reserved + i))) {
        return true;   // is a reserved
    }
}
return false;  // is not
```

运算符同理

其实还有一个偷懒的写法：

```
char *operators = " + - * / < > = == >= <= != "; // why there is space?
char *tmp = malloc(4096);
sprintf(tmp, " %s ", s);   // add space in the front and rear of s
if (strstr(operators, tmp)) return true; // !!
return false;
```

保留字是同理。

变量

对于开头，检查是否是字母或下划线；对于后续，检测是否是字母、数字或下划线。

活用 `isalpha()`, `isdigit()`...

```
if (!(isalpha(*s) || *s == '_')) return false;
for (s++; *s != 0; s++) {
  if (!(isalpha(*s) || isdigit(*s) || s == '_')) return false;
}
return true;
```

整数

`isdigit()` 过一遍就好

浮点数

`isdigit()` 过一遍，但有一个位置可以是 '.'

STD

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

char *reserved = " break char const double enum extern float goto int long
return static struct typedef union void ";
char *operators = " + - * / > < >= <= == = != ";
char *tmp, *output;

int isinteger(char *s) {
    for (; *s; s++) if (!isdigit(*s)) return 0;
    return 1;
}

int isfloat(char *s) {
    int cnt = 0;
    for (; *s; s++)
        if (!isdigit(*s)) {
            if (*s == '.') cnt++;
            else return 0;
        }
    return cnt == 1 ? 1 : 0;
}

int isreserved(char *s) {
    sprintf(tmp, " %s ", s);
    return strstr(reserved, tmp) ? 1 : 0;
}

int isvariable(char *s) {
    if (!(isalpha(*s) || *s == '_')) return 0;
    for (s++; *s != 0; s++) {
        if (!(isalpha(*s) || isdigit(*s) || *s == '_')) return 0;
    }
    return 1;
}

int isoperator(char *s) {
    sprintf(tmp, " %s ", s);
    return strstr(operators, tmp) ? 1 : 0;
}

void process(char *s) {
    if (!*s) return;  // empty string while splitted by semicolon
    if (isinteger(s)) strcat(output, "integer ");
    else if (isreserved(s)) strcat(output, "reserved ");
    else if (isoperator(s)) strcat(output, "operator ");
    else if (isfloat(s)) strcat(output, "float ");
    else if (isvariable(s)) strcat(output, "variable ");
```

```
    else {
        printf("Compile Error\n");
        exit(EXIT_SUCCESS);
    }
}

int main() {
    char *s = malloc(4096);
    output = malloc(8192);
    tmp = malloc(4096);
    while (scanf("%s", s) != EOF) {
        char *semi = strchr(s, ';');
        if (!semi) process(s);
        else {
            *semi = 0;
            process(s);
            strcat(output, "\n");
            process(semi + 1);
        }
    }
    printf("%s", output);
    return 0;
}
```

B-字符串拼接 (stringcat.c)

从长到短找出最长的公共部分。注意 A 有可能比 B 长，有的同学对于

```
ababacd ababa
```

这样的一组，在寻找公共部分的时候，从 A 的第一个字符找起。显然 5 个字符全匹配了，但匹配部分并不是 A 的后缀。

偷懒：

对于一个字符串 B 的前缀 P, 总有

$$strstr(B, P) == B$$

那么...

STD

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

int main() {
    int T;
    scanf("%d", &T);
    char *a = malloc(1001), *b = malloc(1001);
    while (T--) {
        scanf("%s%s", a, b);
```

```
        int la = strlen(a);
        int lb = strlen(b);
        for (int i = (la > lb) ? la - lb : 0; i <= la; i++) {   // 为什么是 "<=" ?
            if (strstr(b, a + i) == b) {
                printf("%s%s\n", a, b + la - i);
                break;
            }
        }
    }
    return 0;
}
```

思考：为什么这里是 "<="

思考：`strstr(s, "")` 的结果

C-参数传递 (parse.c)

> Saki 放上去的参考代码里，有一份我的原实现。那是纯纯的超纲写法。
> 不需要 getopt()。那是 GNUC 的玩意。
> 也不需要 dup2()，这纯纯的 Linux 系统调用。

思路是，从前往后扫就完事了。

有的同学发现，option requires an argument 只会在最后出现，于是先去看最后一个参数。

考虑：

```
b:c:
program -b -c
```

只看 -c，你可能已经想报错了。但这个 -c 其实应当作为 -b 的值。

输出与报错的实现思路，和 A 是一致的。

> 嘿，可以学一下 `sprintf()` 的使用！

STD

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

int main() {
    char *output = malloc(4096), *rules = malloc(128);
    char *name = malloc(1024), *arg = malloc(1024);
    char *value = malloc(1024);
    int pos = 0;
    scanf("%s%s", rules, name);
    pos += sprintf(output + pos, "%s\n", name);
    while ((scanf("%s", arg) != EOF)) {
        if (*arg != '-') continue;
        char *rule = strchr(rules, *(arg + 1));
        if (rule != NULL) {
            if (*(rule + 1) == ':') {
```

```
                if (scanf("%s", value) == EOF) {
                    printf("%s: option requires an argument -- '%c'\n", name,
*rule);
                    return 0;
                } else
                    pos += sprintf(output + pos, "%c=%s\n", *rule, value);
            } else
                pos += sprintf(output + pos, "%c\n", *rule);
        } else {
            printf("%s: invalid option -- '%c'\n", name, *(arg + 1));
            return 0;
        }
    }
    printf("%s", output);
    return 0;
}
```

Takeaways (TL;DR)

- 字符串如何终止
  - null byte
- 学习简单的库函数让你在编程时偷懒
  - 但是要搞清楚这些函数的行为
  - 用好了能节省大量时间
  - 所以前提是学习的时候不偷懒
- 指针难写
  - 去学断点调试
  - 去学 GDB
- 还不知道如何处理 SIGSEGV / Segmentation fault？？？
  - STFW+自己调试，或惨挂