# 结构体（struct）

## 【题目背景】

在 C++ 等高级语言中，除了 `int` 和 `float` 等基本类型外，通常还可以自定义结构体类型。在本题当中，你需要模拟一种类似 C++ 的高级语言的结构体定义方式，并计算出相应的内存占用等信息。

## 【题目描述】

在这种语言中，基本类型共有 4 种：`byte`，`short`，`int`，`long`，分别占据 $1, 2, 4, 8$ 字节的空间。

定义一个结构体**类型**时，需要给出**类型名**和**成员**，其中每个成员需要按顺序给出**类型**和**名称**。类型可以为基本类型，也可以为**先前定义过**的结构体类型。注意，定义结构体**类型**时不会定义具体元素，即不占用内存。

定义一个**元素**时，需要给出元素的**类型**和**名称**。元素将按照以下规则占据内存：

- 元素内的所有成员将按照**定义时给出的顺序**在内存中排布，对于类型为结构体的成员同理。
- 为了保证内存访问的效率，元素的地址占用需要满足**对齐规则**，即任何类型的**大小**和该类型元素在内存中的**起始地址**均应对齐到该类型对齐要求的**整数倍**。具体而言：
  - 对于基本类型：对齐要求等于其占据空间大小，如 `int` 类型需要对齐到 4 字节，其余同理。
  - 对于结构体类型：对齐要求等于其成员的对齐要求的**最大值**，如一个含有 `int` 和 `short` 的结构体类型需要对齐到 4 字节。

以下是一个例子（以 C++ 语言的格式书写）：

```
struct d {
    short a;
    int b;
    short c;
};
d e;
```

该代码定义了结构体类型 `d` 与元素 `e`。元素 `e` 包含三个成员 `e.a`，`e.b`，`e.c`，分别占据第 $0 \sim 1$，$4 \sim 7$，$8 \sim 9$ 字节的地址。由于类型 `d` 需要对齐到 4 字节，因此 `e` 占据了第 $0 \sim 11$ 字节的地址，大小为 12 字节。

你需要处理 $n$ 次操作，每次操作为以下四种之一：

1. 定义一个结构体类型。具体而言，给定正整数 $k$ 与字符串 $s, t_1, n_1, \ldots, t_k, n_k$，其中 $k$ 表示该类型的成员数量，$s$ 表示该类型的类型名，$t_1, t_2, \ldots, t_k$ 按顺序分别表示每个成员的类型，$n_1, n_2, \ldots, n_k$ 按顺序分别表示每个成员的名称。你需要输出该结构体类型的大小和对齐要求，用一个空格分隔。
2. 定义一个元素，具体而言，给定字符串 $t$，$n$ 分别表示该元素的类型与名称。所有被定义的元素将按顺序，从内存地址为 0 开始依次排开，并需要满足地址对齐规则。你需要输出新定义的元素的起始地址。
3. 访问某个元素。具体而言，给定字符串 $s$，表示所访问的元素。与 C++ 等语言相同，采用 `.` 来访问结构体类型的成员。如 `a.b.c`，表示 `a` 是一个已定义的元素，它是一个结构体类型，有一个名称为 `b` 的成员，它也是一个结构体类型，有一个名称为 `c` 的成员。你需要输出如上被访问的**最内层**元素的起始地址。
4. 访问某个内存地址。具体而言，给定非负整数 $addr$，表示所访问的地址，你需要判断是否存在一个**基本类型**的元素占据了该地址。若是，则按操作 3 中的访问元素格式输出该元素；否则输出 `ERR`。

## 【输入格式】

从文件 ***struct.in*** 中读入数据。

第 1 行：一个正整数 $n$，表示操作的数量。

接下来若干行，依次描述每个操作，每行第一个正整数 $op$ 表示操作类型：

- 若 $op = 1$，首先输入一个字符串 $s$ 与一个正整数 $k$，表示类型名与成员数量，接下来 $k$ 行每行输入两个字符串 $t_i$，$n_i$，依次表示每个成员的类型与名称。
- 若 $op = 2$，输入两个字符串 $t$，$n$，表示该元素的类型与名称。
- 若 $op = 3$，输入一个字符串 $s$，表示所访问的元素。
- 若 $op = 4$，输入一个非负整数 $addr$，表示所访问的地址。

## 【输出格式】

输出到文件 ***struct.out*** 中。

输出 $n$ 行，依次表示每个操作的输出结果，输出要求如题目描述中所述。

## 【样例 1 输入】

```
5
1 a 2
short aa
int ab
1 b 2
a ba
```

```
long bb
2 b x
3 x.ba.ab
4 10
```

【样例 1 输出】

```
8 4
16 8
0
4
x.bb
```

【样例 1 解释】

结构体类型 a 中，int 类型的成员 aa 占据第 0 ～ 3 字节地址，short 类型的成员 ab 占据第 4 ～ 5 字节地址。又由于其对齐要求为 4 字节，可得其大小为 8 字节。由此可同理计算出结构体类型 b 的大小为 16 字节，对齐要求为 8 字节。

【样例 2】

见选手目录下的 ***struct/struct2.in*** 与 ***struct/struct2.ans***。

【样例 2 解释】

第二个操作 4 中，访问的内存地址恰好在为了地址对齐而留下的“洞”里，因此没有基本类型元素占据它。

【样例 3】

见选手目录下的 ***struct/struct3.in*** 与 ***struct/struct3.ans***。

【数据范围】

对于全部数据，满足 $1 \leq n \leq 100$，$1 \leq k \leq 100$，$0 \leq addr \leq 10^{18}$。

所有定义的结构体类型名、成员名称和定义的元素名称均由不超过 10 个字符的小写字母组成，且都不是 byte,short,int,long（即不与基本类型重名）。

所有定义的结构体类型名和元素名称互不相同，同一结构体内成员名称互不相同。但不同的结构体可能有相同的成员名称，某结构体内的成员名称也可能与定义的结构体或元素名称相同。

保证所有操作均符合题目所述的规范和要求，即结构体的定义不会包含不存在的类型、不会访问不存在的元素或成员等。

保证任意结构体大小及定义的元素占据的最高内存地址均不超过 $10^{18}$。

| 测试点编号 | 特殊性质 |
|---|---|
| 1 | A、D |
| $2 \sim 3$ | A |
| $4 \sim 5$ | B、D |
| $6 \sim 8$ | B |
| $9 \sim 10$ | C、D |
| $11 \sim 13$ | C |
| $14 \sim 16$ | D |
| $17 \sim 20$ | 无 |

特殊性质 A：没有操作 1；

特殊性质 B：只有一个操作 1；

特殊性质 C：所有操作 1 中给出的成员类型均为基本类型；

特殊性质 D：基本类型只有 `long`。

## 【提示】

对于结构体类型的对齐要求和大小，形式化的定义方式如下：

- 设该结构体内有 $k$ 个成员，其大小分别为 $s_1, ..., s_k$，对齐要求分别为 $a_1, ..., a_k$；
- 则该结构体的对齐要求为 $a = \max\{a_1, ..., a_k\}$；
- 再设这些成员排布时的**地址偏移量**分别为 $o_1, ..., o_k$，则：
  - $o_1 = 0$；
  - 对于 $i = 2, ..., k$，$o_i$ 为满足 $o_{i-1} + s_{i-1} \le o_i$ 且 $a_i$ 整除 $o_i$ 的最小值；
  - 则该结构体的大小 $s$ 为满足 $o_k + s_k \le s$ 且 $a$ 整除 $s$ 的最小值；

对于定义元素时的内存排布，形式化的定义方式如下：

- 设第 $i$ 个被定义的元素大小为 $s_i$，对齐要求为 $a_i$，起始地址为 $b_i$；
- 则 $b_1 = 0$，对于 $2 \le i$，$b_i$ 为满足 $b_{i-1} + s_{i-1} \le b_i$ 且 $a_i$ 整除 $b_i$ 的最小值。