电子词典（C语言版本）

词典在文件里面的存储格式

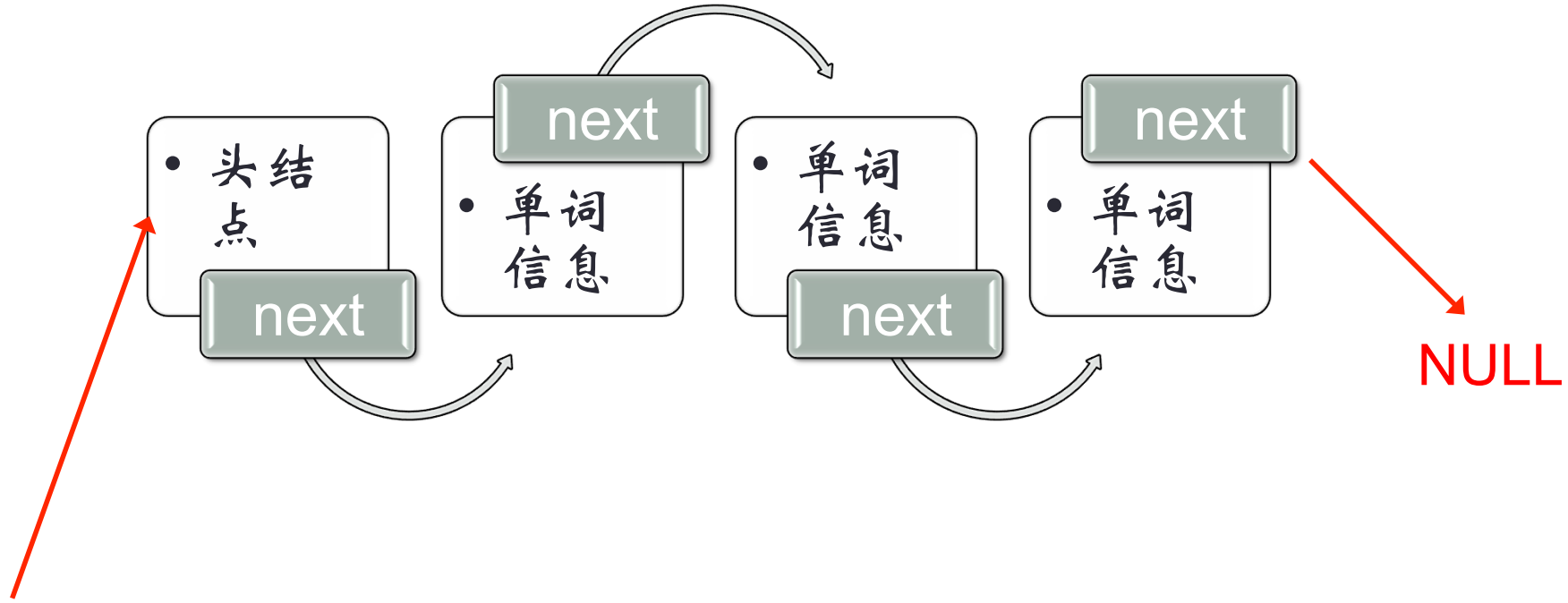
一个单词占2行，第一行存储的是单词名称，以"#"开头

第二行是单词的解释，以"Trans:"开头，如果有多个解释，则解释之间以"@"分割

```
#a
Trans:art. 一;字母A
#a.m.
Trans:n. 上午
#a/c
Trans:n. 往来帐户@往来:come - and - go; contact;
intercourse@n. 往来帐户
```

怎样把这些数据格式化存储起来

我们用链表来保存，每个节点存储一个单词的信息

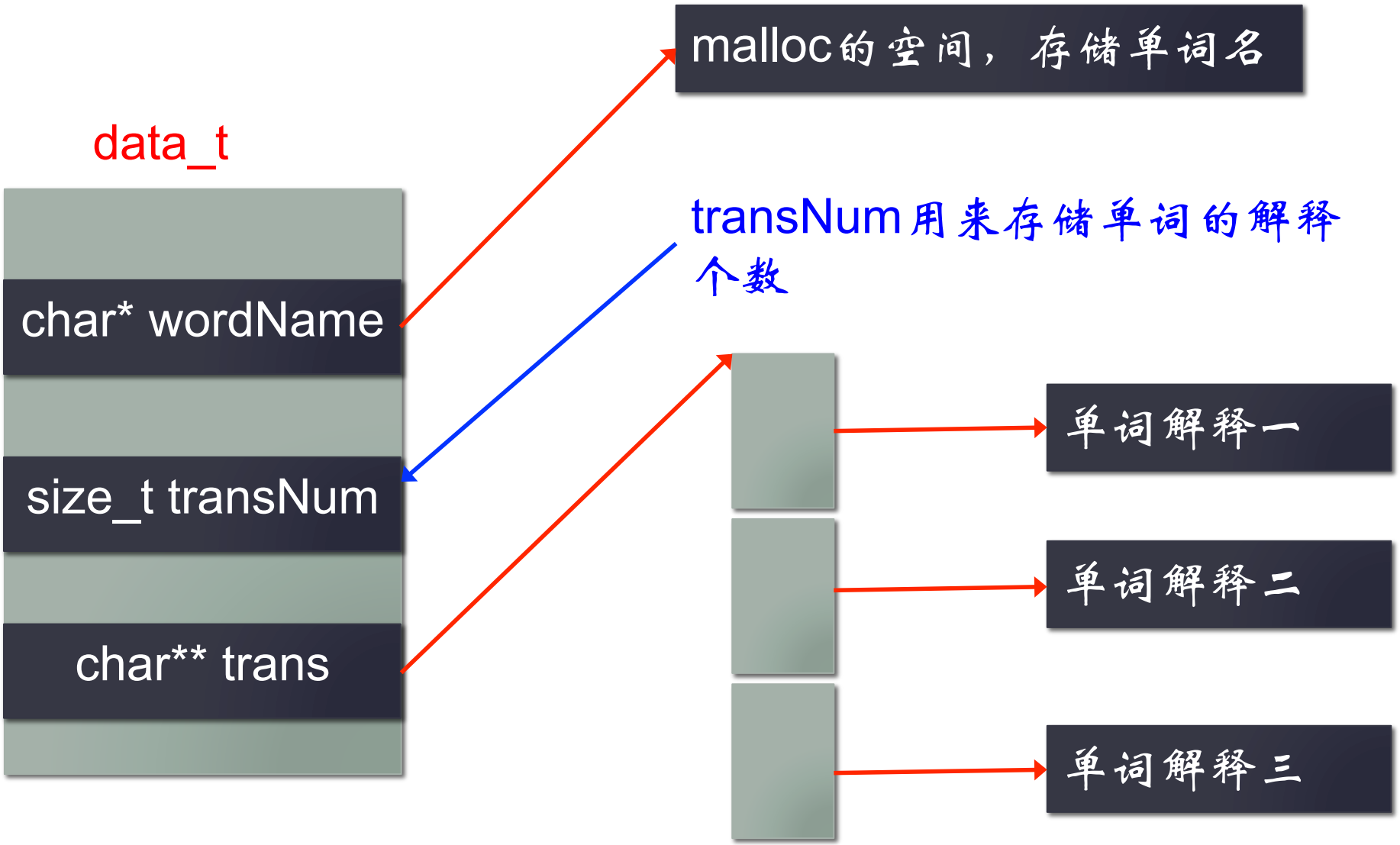


node_t的结构

```
typedef struct node
{
    data_t data;
    struct node* next;
}node_t;
```

data_t又是什么结构呢？

data_t的内存示意图



data_t的结构

```
typedef struct
{
    //用来保存单词名称，不包括'#'和'\n'，需动态开辟空间
  char* wordName;

    //解释的个数
  size_t transNum;

    //保存字符指针数组的首地址，字符指针数组是动态开辟
  char** trans;
}data_t;
```

操作链表的相关函数

```
//初始化一个链表
node_t* initList(void);

//向链表里面插入节点
void insertNode(node_t* listHead, node_t*
dataNode);

//根据要查找的单词名，返回对应节点，没有返回NULL
node_t* searchNode(node_t* listHead, const
char* wordName);

//删除链表，清理所有资源，难度最大的函数
void destoryList(node_t* listHead);
```

main函数代码

```
int main (int argc, const char * argv[])
{
    if (argc!=2) {
        printf("argments wrong.\n");
        exit(-1);
    }

    FILE* fp=openFile(argv[1], "r");
    node_t* listHead=initDictionary(fp);
    runTask(listHead);

    return 0;
}
```

其它辅助函数

```
//封装fopen，加入出错处理
FILE* openFile(const char* fileName, const char* mode);

//统计每行有多少个字符，包含‘\n’，统计后位置指示标志和先前一样
size_t getNumberOfLineCharacters(FILE* fp);

//统计字符串里面有几个@符号
size_t getNumberOfAtCharacter(const char* str);

//封装一个读取一行的函数
void myGetLine(char* buf, FILE* fp);

//把字典文件格式化，得到一个链表，难度最大
node_t* initDictionary(FILE* fp);

void showTrans(char* trans[], size_t num);

void runTask(node_t* listHead);
```