typedef & malloc <=>free

typedef

第一种方式：标签名和别名都定义

用例：定义一个二叉树节点

```
typedef struct TreeNode {
    int value;
    struct TreeNode* left;
    struct TreeNode* right;
} TreeNode;
```

在这个例子中，我们使用标签名 `struct TreeNode` 来定义一个递归结构体，因为我们需要在结构体内部引用相同类型的结构体成员（`left` 和 `right` 指针）。

使用：

```
TreeNode* root = malloc(sizeof(TreeNode));
root->value = 10;
root->left = malloc(sizeof(TreeNode));
root->left->value = 5;
root->right = malloc(sizeof(TreeNode));
root->right->value = 15;
```

第二种方式：只定义标签名

用例：定义一个图形界面的窗口结构

```
// 在头文件中
struct Window;

// 在源文件中
struct Window {
    int width;
    int height;
    char* title;
};
```

在这种情况下，我们可能不想让库的使用者直接访问结构体的内部成员。他们只能通过 API 函数来操作 `struct Window`。

使用：

```
struct Window* createWindow(int width, int height, const char* title);
void resizeWindow(struct Window* win, int newWidth, int newHeight);
```

第三种方式：只定义别名

用例：定义一个点的坐标

```
typedef struct {
    int x;
    int y;
} Point;
```

这里，我们没有给结构体定义一个标签名，因为我们不需要在结构体内部引用它。

使用：

```
Point p1;
p1.x = 1;
p1.y = 2;

Point p2 = {3, 4};
```

`malloc` 关键字

`malloc`（Memory Allocation）是 C 语言标准库中的一个函数，用于在堆（Heap）上动态分配内存。它返回一个指向分配的内存块的指针。

语法：

```
void* malloc(size_t size);
```

示例：

```
int *arr = (int*) malloc(10 * sizeof(int));  // 分配一个包含10个整数的数组
```

堆（Heap）与栈（Stack）

- **堆（Heap）**：是一个用于动态内存分配的内存区域。当你使用 `malloc`、`calloc` 或 `new`（在 C++ 中）时，分配的内存来自堆。
- **栈（Stack）**：是一个用于存储局部变量和函数调用信息的内存区域。当你在函数中声明一个变量时，它的内存通常来自栈。

堆与栈的区别：

1. **生命周期**：堆上的内存需要手动管理，而栈上的内存是自动管理的。
2. **大小限制**：堆通常比栈大得多。
3. **访问速度**：栈上的内存访问通常比堆上的快。
4. **碎片化**：堆内存容易碎片化，而栈不会。

使用注意事项：

内存泄漏：使用 `malloc` 分配的内存必须通过 `free` 函数释放，以防止内存泄漏。

```
free(arr);
```

空指针检查：`malloc` 可能会返回 `NULL`，特别是当系统内存不足时。

```
if (arr == NULL) {
    // 处理内存分配失败
}
```

1. **未初始化的内存**：`malloc` 不会初始化分配的内存。如果你需要一个初始化为零的内存块，可以使用 `calloc`。
2. **数组大小**：当使用动态数组时，确保不要越界访问。
3. **多次释放**：确保不要多次释放同一块内存。
4. **悬挂指针**：释放内存后，确保不要再使用该块内存的指针。