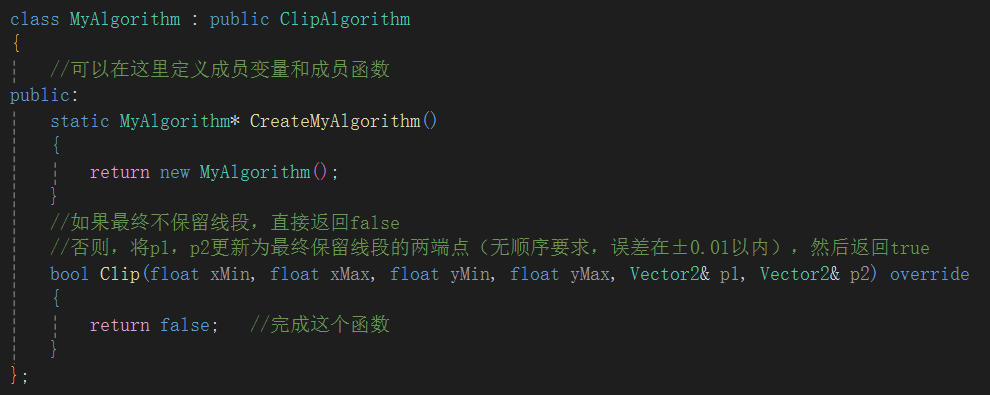
**1 裁剪算法和多边形扫描转换实验**

**李俊辉 2201 U202217182**

* 1. **实验目的**

1. **熟悉Cohen-sutherland算法和Liang-Barsky算法的思想，并掌握至少一种裁剪算法的实现**
2. **熟悉X-扫描线算法和边标志算法的思想，并掌握至少一种多边形扫描转换的实现**
3. **掌握一定的算法分析和优化的能力**
   1. **实验要求**

****

1. **完成实验项目中如上图所示的函数，以实现裁剪算法，并回答以下问题：**
2. **用自然语言简述你如何实现这一函数。**

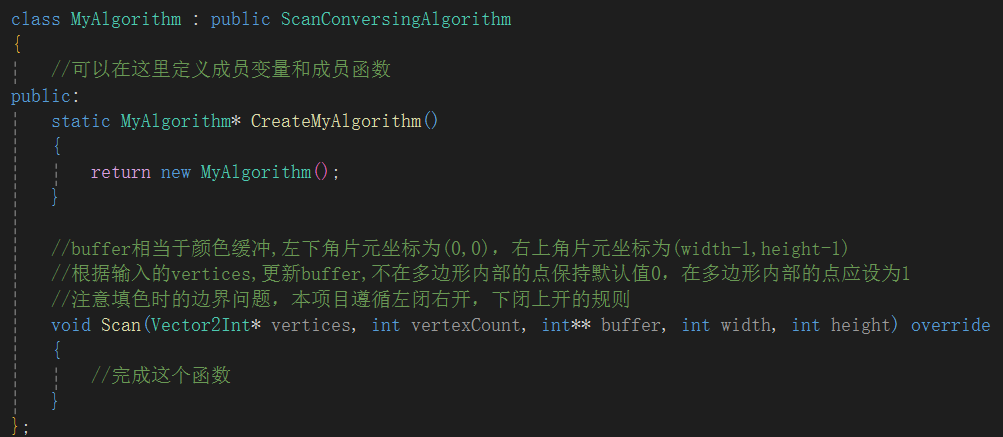
我主要用了Liang-Barsky算法的思想，首先进行输入和初始化参数（float xMin, float xMax, float yMin, float yMax, Vector2& p1, Vector2& p2）。然后进行边界检查，边界检查主要p，q这两个参数，p 表示线段与边界的相对方向（正负决定是“进入”还是“离开”窗口），q 表示线段起点到边界的距离。当p=0时，则线段与边界平行，当q<0,则为在窗口外。计算交点参数 t = q / p\*\*，如果 p < 0（线段从外向内进入窗口）： 更新 t\_in = max(t\_in, t)（确保取最晚的进入点）；如果 p > 0（线段从内向外离开窗口），更新 t\_out = min(t\_out, t)（确保取最早的离开点）。如果 t\_in > t\_out，说明线段完全在窗口外，返回 false。更新裁剪后的线段:如果 t\_in <= t\_out（线段部分可见）：p1.x = p1.x + t\_in \* dx,p1.y = p1.y + t\_in \* dy,p2.x = p1.x + t\_out \* dx,p2.y = p1.y + t\_out \* dy。返回结果:返回 true（线段可见，且 p1 和 p2 已更新为裁剪后的端点）。否则返回 false（线段完全不可见）。

1. **从你实现的函数出发，列出一个影响算法性能的点并简要分析。**

在 Liang-Barsky 算法中，每次边界检查都需要计算 t = q / p，而 ​除法运算（/）在 CPU 上的计算成本较高，尤其是在循环中执行多次时（如 4 次边界检查 × 100,000 次测试）。这会显著影响算法性能。优化点：用乘法代替除法​由于 t = q / p 等价于 t = q \* (1/p)，我们可以 ​预先计算 1/p（倒数）​，然后用乘法代替除法：优化原因​现代 CPU 对乘法优化更好；乘法指令（\*）的延迟通常比除法（/）低 3-5 倍。减少循环内的计算量​：

在边界检查的循环中，除法是主要瓶颈，替换为乘法可提升吞吐量。适用于大量重复计算​：

在测试框架中（如 times = 100,000），微小优化会被放大。

****

1. **完成实验项目中如上图所示的函数，以实现多边形扫描转换算法，并回答以下问题：**
2. **用自然语言简述你如何实现这一函数。**

首先初始化边表，有序边表（Ordered Edge Table, OET）：将多边形的所有边按 y\_min 排序存储，每条边记录以下信息：y\_max（边的最高端点 y 坐标）；x（当前扫描线与边的交点 x 坐标）；斜率倒数（1/m）（用于增量计算交点 x 坐标）。建立活性边表（Active Edge Table, AET）：动态维护当前扫描线（currentY）相交的边，初始时包含y\_min = currentY 的边。然后逐行扫描填充，先调用 activeEdges->MoveUp(intersectionXs) 将扫描线从下向上移动（currentY++），并执行：更新 AET​：移除 y\_max <= currentY 的边（已超出范围）。计算交点：对 AET 中剩余的边，计算与当前扫描线的交点 x 坐标，存入 intersectionXs 数组。排序交点：将 intersectionXs 按 x 坐标升序排列（保证每对交点形成区间）。接下来填充像素区间​：遍历 intersectionXs（每两个 x 值为一个区间）：区间范围​：[startX, endX)（左闭右开）。标记像素​：对区间内的每个 x，若在屏幕范围内（0 ≤ x < width，0 ≤ currentY < height），设置 buffer[x][currentY] = 1。更新 AET 边参数​：对 AET 中的每条边，根据斜率增量更新下一次交点的 x 值：  
x += 1/m（避免重复计算交点）。最后边界处理与内存管理​边界规则​：遵循 ​左闭右开（x）、下闭上开（y）的填充规则。释放内存：扫描完成后销毁AET和OET对象

1. **从你实现的函数出发，列出一个影响算法性能的点并简要分析。**

交点计算与排序的开销​在Scan 函数中，每行扫描时需对活性边表（AET）中的边计算交点并排序​（intersectionXs），这是算法的核心性能瓶颈：计算交点​：对每条活性边执行 x += 1/m（浮点运算）。排序交点​：每行需对交点数组 intersectionXs 进行排序（如快速排序），时间复杂度为 O(k log k)（k 为当前活性边数量）。

优化点：预排序活性边表 + 增量维护有序性​

预排序活性边表（AET）：在初始化 AET 时，按当前交点的 x 坐标排序，后续通过增量维护有序性。避免每行重新排序。排序规则：按 x 升序，若 x 相同则按 1/m 升序（避免交叉）。

增量更新交点：在 MoveUp 中更新活性边的 x 后，​仅局部调整顺序，而非完全重新排序。

* 1. **实验小结**

对于裁剪算法（Liang-Barsky）：通过参数化计算线段与裁剪窗口的交点，仅保留可见部分。核心在于高效处理边界条件和并行性优化（如用乘法代替除法）。关键点：边界检查的顺序、平行线段的特殊处理、参数化计算的数学技巧。扫描线填充算法：通过维护活性边表（AET）和有序边表（OET），逐行计算交点并填充区间。关键点：边表的构建与更新、交点排序优化、边界规则的严格遵循（左闭右开、下闭上开）。