在 RTKLIB 的典型工作流程中，pntpos (单点定位, SPP) 和 relpos (相对定位, RTK/PPK) 扮演着不同的角色：

1. **pntpos (单点定位)**：首先被调用，用于计算一个**绝对的、但精度较低（米级）**的初始位置。
2. **relpos (相对定位)**：随后被调用，它利用 pntpos 得到的大致位置作为**初始值**，并结合基准站的数据进行差分处理和模糊度固定（lambda），从而计算出**高精度（厘米级）**的相对位置。

以下是这两个函数在您提供的代码文件中的具体工作流程。

---

1. pntpos 函数流程 (单点定位)

此函数在 pntpos.c 文件中定义。其目的是单独使用一个接收机（流动站）的观测数据和导航电文，计算其位置、速度和钟差（PVT）。

输入：

- obs：(obsd_t 数组) 流动站的观测数据集。
- n：观测到的卫星数量。
- nav：(nav_t 结构体) 包含所有卫星星历和电离层参数的导航数据。
- opt：(prcopt_t 结构体) 处理选项，例如 rtknavi.c 中配置的 PMODE_KINEMA。
- sol：(sol_t 结构体) 用于存储解算结果（在第一次调用时，其内部的 sol->rr 通常为 0 或上一个历元的解，用作初始值）。

输出：

- sol：填充了米级精度的位置 (sol->rr)、速度 (sol->rr+3) 和钟差 (sol->dtr)。
- azel：(double 数组) 每颗卫星的方位角和仰角。
- vsat：(int 数组) 标记每颗卫星在本次解算中是否有效（1 为有效）。
- resp：(double 数组) 每颗有效卫星的伪距残差。

内部具体流程：

1. **获取卫星状态 (satposs)**
    - pntpos 首先调用 satposs 来获取所有 n 颗卫星在信号**发射时刻**的精确状态。

- satposs 内部循环每颗卫星：
  - 从 obs[i].P[j] 读取伪距 pr，计算信号发射时间 (time[i] = obs[i].time - pr/CLIGHT)。
  - 调用 ephclk 计算一个初步的卫星钟差 dt，并用 dt 再次修正发射时间 time[i]。
  - 调用 satpos，satpos 根据配置（opt->sateph）选择：
    - **广播星历 (BRDC)**：调用 ephpos，后者会从 nav->eph (GPS/GAL 等) 或 nav->geph (GLO) 中选择星历，并调用 eph2pos 或 geph2pos 计算卫星位置、速度和钟差。
    - **精密星历 (PREC)**：调用 peph2pos，后者使用 SP3 和 CLK 文件数据（通过 pephpos 和 pephclk）进行插值计算。
- **satposs 输出**：rs (卫星位置/速度数组), dts (卫星钟差/钟漂数组), var (方差), svh (健康状态)。

2. **估计接收机位置 (estpos)**
   - 这是核心的**伪距最小二乘迭代**解算。
   - **迭代开始**（最多 10 次）：
     - **A. 计算残差 (rescode)**：
       - 使用上一次迭代的位置 x（sol->rr 作为初值）计算到每颗卫星的几何距离 r 和视线向量 e。
       - 计算卫星方位角和仰角 azel (satazel 函数)。
       - **剔除卫星**：剔除低仰角（opt->elmin）或低信噪比（snrmask）的卫星。
       - **应用改正**：
         - **电离层**：调用 ionocorr，若 opt->ionoopt 为 IONOOPT_BRDC，则使用 nav->ion_gps (Klobuchar)；若为 IONOOPT_TEC，则调用 iontec 使用 nav->tec (IONEX)。
         - **对流层**：调用 tropcorr，若 opt->tropopt 为 TROPOPT_SAAS，则使用 tropmodel

(Saastamoinen)。

- **获取伪距**：调用 prange，从 obs->P 读取伪距，并应用 TGD/DCB 改正（TGD 来自 nav->eph[i].tgd，DCB 来自 nav->cbias）。
- **计算残差 v**：v = P_corrected - (r + rcv_clk_bias - sat_clk_bias + d_iono + d_tropo)。
- **构建设计矩阵 H**：计算残差 v 对未知数 x（x, y, z，接收机钟差，系统间钟差）的偏导数。
- **计算权重 var**：根据仰角、广播星历 URA 等计算观测噪声方差。

- **B. 求解**：调用 lsq（最小二乘法，库函数）求解状态修正量 dx。
- **C. 更新状态**：x = x + dx。
- **D. 检查收敛**：如果 dx 足够小，则迭代成功。

- **迭代结束**。
- **A. 检验解 (valsol)**：通过 GDOP 和卡方检验（vv > chisqr）判断当前解是否可靠。
- **B. 存储解**：将解算出的 x（包含位置 rr 和钟差 dtr）存入 sol 结构体。

3. **估计接收机速度 (estvel)**
   - 如果位置解算成功，则调用 estvel。
   - 这是一个独立的最小二乘迭代，用于求解速度（vx, vy, vz）和钟漂。
   - 它调用 resdop，resdop 使用 obs->D[0]（多普勒观测值）来构建残差方程 v 和设计矩阵 H，然后求解速度状态 x。
   - **存储解**：将速度解存入 sol->rr+3。
4. **(可选) 完好性监测 (raim_fde)**
   - 如果 estpos 失败，但 opt->posopt[4] (RAIM) 开启，此函数会被调用。
   - 它会尝试依次**排除**每一颗卫星，并重新调用 estpos。
   - 如果排除某颗卫星后 estpos 成功，并且残差的均方根（RMS）最小，

则认为该卫星是故障卫星，采用排除它之后的解作为最终解。

---

2. relpos 函数流程 (相对定位 - RTK/PPK)

注意：relpos.c 文件未在您提供的列表中，但 rtknavi.c 和 lambda.c 的存在明确指向了 relpos 的功能。以下流程是基于 RTKLIB 标准库结构推导的。

目标： 计算流动站相对于基准站的高精度基线向量。

输入：

- obsr：流动站观测数据（rtksvr.obs[0]）。
- obsb：基站观测数据（rtksvr.obs[1]）。
- nav：导航电文（rtksvr.nav）。
- stap：基站坐标（来自 rtksvr.sta[1]，由 MT 1005/1006 解码得到）。
- sol：来自 pntpos 的**初始 SPP 解**（sol->rr）。

输出：

- sol：更新为高精度的 RTK 解（SOLQ_FIX 或 SOLQ_FLOAT）。

内部具体流程（推测）：

1. **数据准备**：
    - 使用 pntpos 提供的 sol->rr 作为流动站的先验位置。
    - 计算两站（基站、流动站）到所有卫星的几何距离、方位角/仰角。
    - 计算卫星位置（同 pntpos，调用 satposs）。
2. **差分计算**：
    - 对基站和流动站的观测数据（obsr 和 obsb）进行**双差（Double-Difference, DD）**处理。
    - **载波相位双差**： DD_L = (L_r,i - L_b,i) - (L_r,j - L_b,j)
    - **伪距双差**： DD_P = (P_r,i - P_b,i) - (P_r,j - P_b,j)
    - （注：i 和 j 是卫星，r 和 b 是接收机）。
    - 这个过程极大地消除了卫星钟差、接收机钟差、大气延迟（电离层/对流层）和星历误差。
3. **浮点解估算 (Float Solution)**：

- 建立双差观测方程。未知数包括：
  - 基线向量（dx, dy, dz）。
  - 双差模糊度 N（此时为浮点数，如 10.3 周）。
- 使用卡尔曼滤波器或最小二乘法求解这些未知数，得到**浮点解 (Float Solution)**。
- **传递**：将浮点解得到的模糊度向量 (a) 及其协方差矩阵 (Q) 传递给 lambda 函数。

4. **模糊度固定 (Integer Ambiguity Resolution)**
   - 调用 lambda.c: lambda() 函数。
   - **lambda 内部流程**：
     - **LD 分解**：调用 LD()，对协方差矩阵 Q 进行分解 (Q = L' * D * L)。
     - **缩减 (Reduction)**：调用 reduction()，通过 Z 变换（整数高斯变换 gauss 和置换 perm）对 Q 进行降相关，使得模糊度更易搜索。
     - **搜索 (Search)**：调用 search() (即 MLAMBDA 算法)。在变换后的空间中搜索**最优**和**次优**的整数模糊度候选值（E），并计算它们的残差平方和 (s[0] 和 s[1])。
     - **反变换**：调用 solve()，将整数候选值 E 转换回原始的模糊度域 F。
   - **lambda 输出**：返回 m 组最佳整数模糊度 F 及其对应的残差 s。
5. **解的检验 (Ratio Test)**
   - 比较最佳解的残差 s[0] 和次优解的残差 s[1]。
   - 计算比率 ratio = s[1] / s[0]。
   - 检查 ratio 是否大于 prcopt->thresar（在 rtknavi.c 中设为 3.0）。
6. **固定解 (Fixed Solution)**
   - **If (Ratio Test 通过)**：
     - 认为最佳整数模糊度 F[0] 可靠。
     - 将 F[0] 作为已知值（约束）代入双差方程。

        - 再次使用最小二乘法，仅求解基线向量（dx, dy, dz）。
        - **存储解**：sol->stat = SOLQ_FIX（固定解），精度为厘米级。
    - **If (Ratio Test 未通过)**:
        - 不使用整数解。
        - **存储解**：sol->stat = SOLQ_FLOAT（浮点解），精度为分米级。
7. **输出**：返回高精度的 sol 结构体。