Here’s a breakdown of the key C++ programming topics used throughout your CRN simulator, organized in a table with brief descriptions and code examples:

| **No.** | **Topic** | **Description** | **Example & Citation** |
| --- | --- | --- | --- |
| 1 | **Data Structures: struct & enum** | Custom POD types to hold PU/SU state (stPU, stSU) and an enum for activation/sensing modes. | struct stPU { … }; enum enActivationMethodsAndSensingMethods { … }; |
| 2 | **STL Containers & Algorithms** | Use of std::vector for dynamic arrays and std::sample, back\_inserter to randomly choose allocation bands. | vector<stPU> PUs(100); sample(...); |
| 3 | **Random Number Generation** | Pseudorandom streams via <random>: mt19937\_64, bernoulli\_distribution for coin flips, uniform\_int\_distribution for start‐time sampling. | FlipACoin(p); RandomNumberFrom0To99(); |
| 4 | **File I/O & CSV Output** | Writing per‐slot and per‐band results to CSV files with <fstream>: opening, streaming values with <<, then closing. | OutPerSlotValues.open("…csv"); OutPerSlotValues << …; |
| 5 | **Control Flow & Looping** | Nested for‑loops over time slots, PUs, SUs; if/else for state toggling, continue to skip inactive PUs, resetting counters each slot. | for (int TimeSlot = 0; …) { … if (…) continue; } |
| 6 | **Function Decomposition** | Encapsulation of each simulation step and metric into functions: LocalSensing(), MajoritySensing(), CalculateAverage…(), Interference(), etc. | void CalculateAverageUtilizationPerTimeSLot() { … } |
| 7 | **Simulation Logic** | Two PU activity models: deterministic toggling based on counters (PUs\_ActivationDeterminstic) and Markov chain transitions (Markov\_Chain()). | if (PU\_Activity==0 && OffCounter==T\_INACTIVE)… FlipACoin… |
| 8 | **Metric Computation** | Incrementing counters and computing averages for false alarms, miss detections, collisions, throughput, utilization, interference both per‐slot and per‐band. | AverageFalseAlarmPerTimeSlot\_Local = …/SUs.size(); |
| 9 | **Vector Management** | Dynamic resizing (.resize(), .reserve()), clearing (.clear()), inserting (.push\_back(), .insert()) to manage sensing results and band‐index lists. | SUs.at(i).SensingResults.resize(100); IndexOfAllAllocatedBands.clear(); |
| 10 | **Console UI & System Calls** | ASCII art screens (WaitingScreen(), EndScreen()) and simple system calls (system("Color 6F")) for user feedback. | cout << "Please Wait " << system("Color 6F"); |

**Small Sub-Projects Included Inside the Program**

1. **Primary User Activity Modeling**
   * Deterministic & Markov-based PU activation
2. **Spectrum Sensing Techniques**
   * Local sensing with false alarms/miss detections
   * Majority sensing for cooperative decisions
3. **Dynamic Spectrum Access**
   * Allocating empty bands to Secondary Users (SUs)
4. **Collision & Interference Detection**
   * Counts band collisions, interference events
5. **Throughput & Utilization Metrics**
   * Calculates network throughput and band utilization
6. **Performance Evaluation**
   * Per-slot and per-band reports for simulations
7. **Probability-based Decision Making**
   * Coin flips for probabilistic events
8. **Automation**
   * Loops for running 30,000 time slots
   * Automated resets and report generation
9. **User Interaction (Cosmetic)**
   * Waiting screens, colorized console outputs