**1. Program Selected:**

#include <stdio.h>

float getAverageTemp(float readings[], int n) {

float sum = 0;

for (int i = 0; i < n; i++) {

sum += readings[i];

}

return sum / n;

}

void controlHeater(float currentTemp, float desiredTemp) {

if (currentTemp < desiredTemp - 2)

printf("Heater ON - Increasing temperature...\n");

else if (currentTemp > desiredTemp + 2)

printf("Heater OFF - Room is warm enough.\n");

else

printf("Temperature is stable. No action needed.\n");

}

int main() {

float readings[5], desiredTemp;

printf("Enter desired room temperature: ");

scanf("%f", &desiredTemp);

printf("Enter 5 temperature readings:\n");

for (int i = 0; i < 5; i++) {

printf("Reading %d: ", i + 1);

scanf("%f", &readings[i]);

}

float avgTemp = getAverageTemp(readings, 5);

printf("Average Temperature: %.2f°C\n", avgTemp);

controlHeater(avgTemp, desiredTemp);

return 0;

}

**2. Structured Constructs Identified:**

| **Construct** | **Example** | **Description** |
| --- | --- | --- |
| **Sequence** | sum += readings[i];, printf() | Basic instruction flow |
| **Selection** | if (currentTemp < desiredTemp - 2) | Decision logic |
| **Iteration** | for (int i = 0; i < 5; i++) | Collect readings repeatedly |

Each block can be “boxed” as a separate logical structure — clear and modular.

**3. Violation Check:**

* ✅ No goto
* ✅ Proper modular decomposition into functions
* ✅ No nested confusion or unstructured branching

So, there are **no violations**.

**4. Observation:**

Every box drawn around the constructs shows **clean nesting** — the hallmark of structured programming.  
Each function has a single purpose (e.g., reading temperature, controlling heater).

**5. Conclusion:**

The Smart Home Temperature Regulator follows all principles of **structured programming** —  
it is modular, clear, and easy to maintain. The flow of control remains predictable and efficient.