```
#include<bits/stdc++.h>
#define MAX 5
using namespace std;

class node{
public:
    int process[MAX];
    int arrival[MAX];
    int burst[MAX];
    int completion[MAX];
    int turn_arround[MAX];
    int waiting[MAX];
    int remaining_burst[MAX];
    node* next;
};

class cll{
public:
    node* head;
    node* tail;
    int time_quantum;

    cll(){
        head = NULL;
        tail = NULL;
    }

    void accept_time_quantum();
    void accept();
    void calculate_completion();
    void calculate_turn_arround();
    void calculate_waiting();
    void calculate_avg();
    void display_table();
};

// Accept the time quantum
void cll::accept_time_quantum(){
    cout << "Enter Time Quantum = ";
    cin >> time_quantum;
}

// Accept process information (arrival time and burst time)
void cll::accept(){
    node* temp = new node;
    cout << "Enter process details (arrival and burst time):" << endl;
    for(int i = 0; i < MAX; i++){
        cout << "Process " << i + 1 << " arrival time: ";
        cin >> temp->arrival[i];
        cout << "Process " << i + 1 << " burst time: ";
        cin >> temp->burst[i];
        temp->process[i] = i + 1;
        temp->remaining_burst[i] = temp->burst[i];  // For round-robin
scheduling
    }
    head = temp;
    tail = temp;
    tail->next = head;  // Circular link
```

```
}

// Calculate completion times using Round-Robin scheduling
void cll::calculate_completion(){
    int time = 0;
    bool done;
    while (true){
        done = true;
        for(int i = 0; i < MAX; i++){
            if(head->remaining_burst[i] > 0){
                done = false;
                if(head->remaining_burst[i] > time_quantum){
                    time += time_quantum;
                    head->remaining_burst[i] -= time_quantum;
                }
                            else{
                    time += head->remaining_burst[i];
                    head->completion[i] = time;
                    head->remaining_burst[i] = 0;
                }
            }
        }
        if(done) break;
    }
}

// Calculate turn around times
void cll::calculate_turn_arround(){
    for(int i = 0; i < MAX; i++){
        head->turn_arround[i] = head->completion[i] - head->arrival[i];
    }
}

// Calculate waiting times
void cll::calculate_waiting(){
    for(int i = 0; i < MAX; i++){
        head->waiting[i] = head->turn_arround[i] - head->burst[i];
    }
}

// Calculate and display average times
void cll::calculate_avg(){
    int total_completion = 0, total_turn_arround = 0, total_waiting =
0;

    for(int i = 0; i < MAX; i++){
        total_completion += head->completion[i];
        total_turn_arround += head->turn_arround[i];
        total_waiting += head->waiting[i];
    }
       cout << "Average Completion Time = " << (float)total_completion
/ MAX << endl;
    cout << "Average Turnaround Time = " << (float)total_turn_arround /
MAX << endl;
    cout << "Average Waiting Time = " << (float)total_waiting / MAX <<
endl;
}
```

```
void cll::display_table(){
    cout << "\n+---------+------------+-----------+----------------
+-----------------+--------------+" << endl;
    cout << "| Process | Arrival Time| Burst Time | Completion Time |
Turnaround Time | Waiting Time |" << endl;
    cout << "+---------+------------+-----------+----------------+--
---------------+--------------+" << endl;
    for (int i = 0; i < MAX; i++) {
        cout << "|    " << head->process[i]
             << "    |      " << head->arrival[i]
             << "     |     " << head->burst[i]
             << "     |       " << head->completion[i]
             << "       |        " << head->turn_arround[i]
             << "       |      " << head->waiting[i]
             << "      |" << endl;
    }
    cout << "+---------+------------+-----------+----------------+--
---------------+--------------+" << endl;
}

int main(){
    cll robin;
    robin.accept_time_quantum();
    robin.accept();
    robin.calculate_completion();
    robin.calculate_turn_arround();
    robin.calculate_waiting();
    robin.display_table();
    robin.calculate_avg();
    return 0;
}
```