	SVKM's NMIMS School of Technology Management & Engineering Navi Mumbai Campus
	Department of Computer Engineering

Name: Rahul Purohit	**SAP ID:** 70562100106
Semester: III	**Year**: II
Subject: Operating systems	**Roll No**.: A022
Batch: 2	**Date**:

Round Robin algorithm:

Round robin is a CPU scheduling algorithm that allows each process to execute for a certain amount of time, called a time quantum, before it is preempted and another process is given a chance to run. The time quantum is typically very short, such as 10 milliseconds or less.
In round robin scheduling, all processes are placed in a queue and the first process in the queue is given the CPU for the time quantum. When the time quantum expires, the process is preempted and the next process in the queue is given the CPU. This process continues until all processes have had a chance to run.
Round robin scheduling is a simple and fair algorithm that ensures that all processes get a chance to run. However, it can also be inefficient if there are many short processes, as each process will only get a small amount of CPU time before it is preempted.

Code:

```cpp
#include<iostream>
#include <algorithm>
#include <queue>
#include<iomanip>
#include<climits>
using namespace std;

struct process_struct
{
  int pid;
  int at;
  int bt;
  int ct,wt,tat,rt,start_time;
  int bt_remaining;
}ps[100];

bool comparatorAT(struct process_struct a,struct process_struct b)
{
   int x =a.at;
   int y =b.at;
   return x < y;
}

bool comparatorPID(struct process_struct a,struct process_struct b)
{
   int x =a.pid;
   int y =b.pid;
   return x < y;
}

int main()
{

    int n,index;
    int cpu_utilization;
    queue<int> q;
    bool visited[100]={false},is_first_process=true;
    int current_time = 0,max_completion_time;
    int completed = 0,tq, total_idle_time=0,length_cycle;
    cout<<"Enter total number of processes: ";
    cin>>n;
    float sum_tat=0,sum_wt=0,sum_rt=0;

    cout << fixed << setprecision(2);

    for(int i=0;i<n;i++)
    {
        cout<<"\nEnter Process " <<i<< " Arrival Time: ";
        cin >> ps[i].at;
        ps[i].pid=i;
    }

    for(int i=0;i<n;i++)
    {
        cout<<"\nEnter Process " <<i<< " Burst Time: ";
        cin >> ps[i].bt;
        ps[i].bt_remaining= ps[i].bt;
    }

    cout<<"\nEnter time quanta: ";
    cin>>tq;

    sort(ps,ps+n,comparatorAT);

    q.push(0);
    visited[0] = true;

    while(completed != n)
    {
      index = q.front();
      q.pop();

      if(ps[index].bt_remaining == ps[index].bt)
      {
            ps[index].start_time = max(current_time,ps[index].at);
            total_idle_time += (is_first_process == true) ? 0 : ps[index].start_time - current_time;
            current_time =  ps[index].start_time;
            is_first_process = false;

      }

      if(ps[index].bt_remaining-tq > 0)
      {
            ps[index].bt_remaining -= tq;
            current_time += tq;
      }
      else
      {
            current_time += ps[index].bt_remaining;
            ps[index].bt_remaining = 0;
            completed++;

            ps[index].ct = current_time;
            ps[index].tat = ps[index].ct - ps[index].at;
            ps[index].wt = ps[index].tat - ps[index].bt;
            ps[index].rt = ps[index].start_time - ps[index].at;

            sum_tat += ps[index].tat;
            sum_wt += ps[index].wt;
            sum_rt += ps[index].rt;
      }

      for(int i = 1; i < n; i++)
      {
          if(ps[i].bt_remaining > 0 && ps[i].at <= current_time && visited[i] == false)
          {
            q.push(i);
            visited[i] = true;
          }
      }
      if( ps[index].bt_remaining> 0)
          q.push(index);
      if(q.empty())
      {
          for(int i = 1; i < n; i++)
          {
            if(ps[i].bt_remaining > 0)
            {
              q.push(i);
              visited[i] = true;
              break;
            }
          }
      }
   }
  max_completion_time = INT_MIN;

  for(int i=0;i<n;i++)
        max_completion_time = max(max_completion_time,ps[i].ct);
  sort(ps, ps+n , comparatorPID);

  //Output
  cout<<"\nProcess No.\tAT\tCPU Burst Time\tStart Time\tCT\tTAT\tWT\n";
  for(int i=0;i<n;i++)
    cout<<i<<"\t\t"<<ps[i].at<<"\t"<<ps[i].bt<<"\t\t"<<ps[i].start_time<<"\t\t"<<ps[i].ct<<"\t"<<ps[i].tat<<"\t"<<ps[i].wt<<endl;
  cout<<endl;

  cout<<"\nAverage Turn Around time= "<< (float)sum_tat/n;
  cout<<"\nAverage Waiting Time= "<<(float)sum_wt/n;
  cout<<"\nAverage Response Time= "<<(float)sum_rt/n;
  return 0;
}
```

Output:

```
Enter total number of processes: 6

Enter Process 0 Arrival Time: 0

Enter Process 1 Arrival Time: 1

Enter Process 2 Arrival Time: 2

Enter Process 3 Arrival Time: 3

Enter Process 4 Arrival Time: 4

Enter Process 5 Arrival Time: 5

Enter Process 0 Burst Time: 5

Enter Process 1 Burst Time: 6

Enter Process 2 Burst Time: 3

Enter Process 3 Burst Time: 1

Enter Process 4 Burst Time: 5

Enter Process 5 Burst Time: 4

Enter time quanta: 4

Process No.     AT      CPU Burst Time  Start Time      CT      TAT     WT
0               0       5               0               17      17      12
1               1       6               4               23      22      16
2               2       3               8               11      9       6
3               3       1               11              12      9       8
4               4       5               12              24      20      15
5               5       4               17              21      16      12


Average Turn Around time= 15.50
Average Waiting Time= 11.50
```