//fcfs

```
#include<stdio.h>
#include<stdlib.h>

typedef struct{
        int pid,bt,wt,tt;     //can consider all struct array
}sp;  //sp=stack pointer(header)

int main(){
        int i,j,n,tott=0,towt=0,tbm=0;
        sp *p,t; //p ,t=ptr(for which space is allocated)
        printf("FCFS scheduling\n");
        printf("enter the no of processes: ");
        scanf("%d",&n);
        p=(sp *)malloc(n * sizeof(sp));  //
        for(i=0;i<n;i++){
                p[i].pid=i+1;
                printf("\nenter burst time of process id %d: ",p[i].pid);
                scanf("%d",&p[i].bt);
        }
        //calculation
        printf("\nprocess scheduling\n");
        printf("process\tburst\twaiting\tturnaround\n");
        for(i=0;i<n;i++){
           tbm+=p[i].bt;
           p[i].tt=tbm;
           p[i].wt=tbm-p[i].bt;
           towt+=p[i].wt;
           tott+=p[i].tt;
           printf("%d\t\t%d\t%d\t%d\n",p[i].pid,p[i].bt,p[i].wt,p[i].tt);
        }
```

```
	printf("avg waiting time:%.2f\n",(float)towt/n);
	printf("avg turn around time:%.2f\n",(float)tott/n);
	free(p);
}
```

//fcfs with at

```
#include<stdio.h>
#include<stdlib.h>

typedef struct{
	int pid,bt,wt,tt,at;     //can consider all struct array
}sp;  //sp=stack pointer(header)

int main(){
	int i,j,n,tott=0,towt=0,tbm=0;
	sp *p,t; //p ,t=ptr(for which space is allocated)
	printf("FCFS scheduling with at\n");
	printf("enter the no of processes: ");
	scanf("%d",&n);
	p=(sp *)malloc(n * sizeof(sp));  //
	for(i=0;i<n;i++){
		p[i].pid=i+1;
		printf("\nenter burst time of process id %d: ",p[i].pid);
		scanf("%d",&p[i].bt);
		printf("\nenter arrival time of process id %d: ",p[i].pid);
		scanf("%d",&p[i].at);
	}
	//sort processes by their at
	for(i=0;i<n;i++){
		for(j=i+1;j<n;j++){
			if(p[i].at>p[j].at){
```

```
                                t=p[i];
                                p[i]=p[j];
                                p[j]=t;
                        }
                }
        }
        //calculation
        printf("\nprocess scheduling\n");
        printf("process\tburst\tarrival\twaiting\tturnaround\n");
        for(i=0;i<n;i++){
           tbm+=p[i].bt;
           p[i].tt=tbm-p[i].at;
           p[i].wt=tbm-p[i].bt-p[i].at;
           towt+=p[i].wt;
           tott+=p[i].tt;
           printf("%d\t\t%d\t%d\t%d\t%d\n",p[i].pid,p[i].bt,p[i].at,p[i].wt,p[i].tt);
        }
        printf("avg waiting time:%.2f\n",(float)towt/n);
        printf("avg turn around time:%.2f\n",(float)tott/n);
        free(p);
}
```

//Preemptive SJF scheduling

```
#include<stdio.h>
#include<stdlib.h>
#include <limits.h>

typedef struct {
   int pid, bt, wt, tt, at, remaining_time;
} sp;
```

```
int main() {
    int i, j, n, tbm = 0, towt = 0, tott = 0;

    printf("Preemptive SJF scheduling with arrival time\n");
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    sp *p, t;
    p = (sp *)malloc(n * sizeof(sp));

    for (i = 0; i < n; i++) {
        p[i].pid = i + 1;
        printf("\nEnter burst time of process id %d: ", p[i].pid);
        scanf("%d", &p[i].bt);
        printf("Enter arrival time of process id %d: ", p[i].pid);
        scanf("%d", &p[i].at);
        p[i].remaining_time = p[i].bt;
    }

    // Sort processes by their arrival time
    for (i = 0; i < n; i++) {
        for (j = i + 1; j < n; j++) {
            if (p[i].at > p[j].at) {
                t = p[i];
                p[i] = p[j];
                p[j] = t;
            }
        }
    }

    printf("\nProcess scheduling\n");
```

```
printf("Process\tBurst\tArrival\tWaiting\tTurnaround\n");

int current_time = 0;

while (1) {
   int idx = -1, min_bt = INT_MAX;

   // Find the process with the shortest remaining burst time at the current time
   for (i = 0; i < n; i++) {
      if (p[i].at <= current_time && p[i].remaining_time > 0 && p[i].remaining_time < min_bt) {
         min_bt = p[i].remaining_time;
         idx = i;
      }
   }

   if (idx == -1) {
      // No processes remaining
      break;
   }

   // Update current time
   current_time++;

   // Update waiting and turnaround times
   if (p[idx].remaining_time == p[idx].bt) {
      // First time the process is executed
      p[idx].wt = current_time - p[idx].at - 1;
   }

   // Update remaining time
   p[idx].remaining_time--;
```

```
        // Check if the process is completed
        if (p[idx].remaining_time == 0) {
            p[idx].tt = current_time - p[idx].at;  //when p[idx].remaining_time,then
current_time==p[i].bt(idx==i)
            tbm += p[idx].bt;
            towt += p[idx].wt;
            tott += p[idx].tt;

            printf("%d\t\t%d\t%d\t%d\t%d\n", p[idx].pid, p[idx].bt, p[idx].at, p[idx].wt, p[idx].tt);
        }
    }

    printf("\nAverage waiting time: %.2f\n", (float)towt / n);
    printf("Average turnaround time: %.2f\n", (float)tott / n);

    free(p);
}
```

//sjf non preemptive

```
#include<stdio.h>
#include<stdlib.h>

typedef struct{
        int pid,bt,wt,tt;     //can consider all struct array
}sp;  //sp=stack pointer(header)

int main(){
        int i,j,n,tott=0,towt=0,tbm=0;
        sp *p,t; //p ,t=ptr(for which space is allocated)
        printf("SJF scheduling\n");
```

```
printf("enter the no of processes: ");
scanf("%d",&n);
p=(sp *)malloc(n * sizeof(sp));  //
for(i=0;i<n;i++){
        p[i].pid=i+1;
        printf("\nenter burst time of process id %d: ",p[i].pid);
        scanf("%d",&p[i].bt);
}
//sort processes by their bt
for(i=0;i<n;i++){
        for(j=i+1;j<n;j++){
                if(p[i].bt>p[j].bt){
                        t=p[i];
                        p[i]=p[j];
                        p[j]=t;
                }
        }
}
//calculation
printf("\nprocess scheduling\n");
printf("process\tburst\twaiting\tturnaround\n");
for(i=0;i<n;i++){
   tbm+=p[i].bt;
   p[i].tt=tbm;
   p[i].wt=tbm-p[i].bt;
   towt+=p[i].wt;
   tott+=p[i].tt;
   printf("%d\t\t%d\t%d\t%d\n",p[i].pid,p[i].bt,p[i].wt,p[i].tt);
}
printf("avg waiting time:%.2f\n",(float)towt/n);
printf("avg turn around time:%.2f\n",(float)tott/n);
```

```
        free(p);
}
```

//rr

```
#include <stdio.h>

int main() {
    int n;
    printf("Enter Total Number of Processes: ");
    scanf("%d", &n);

    int wt = 0, tt = 0;
    int burst[n], rburst[n];
    int q;

    for (int i = 0; i < n; i++) {
        printf("Enter Burst Time for Process %d: ", i + 1);
        scanf("%d", &burst[i]);
        rburst[i] = burst[i];
    }

    printf("Enter Time Slice (Quantum): ");
    scanf("%d", &q);

    int total_time = 0;
    int completed_processes = 0;

    printf("\nProcess ID\tBurst Time\tTurnaround Time\tWaiting Time\n");

    int curp = 0;
```

```
while (completed_processes < n) {
    if (rburst[curp] > 0) {
        int exet;
        if (rburst[curp] > q) {
            exet = q;
        } else {
            exet = rburst[curp];
        }

        total_time += exet;
        rburst[curp] -= exet;

        if (rburst[curp] == 0) {
            completed_processes++;

            int turnaround_time = total_time;
            int waiting_time = turnaround_time - burst[curp];

            printf("%d\t\t%d\t\t%d\t\t%d\n", curp + 1, burst[curp], turnaround_time, waiting_time);

            wt += waiting_time;
            tt += turnaround_time;
        }

        curp = (curp + 1) % n;
    } else {
        curp = (curp + 1) % n;
    }
}

float avg_waiting_time = (float)wt / n;
```

```
    float avg_turnaround_time = (float)tt / n;

    printf("\nAverage Waiting Time: %f", avg_waiting_time);
    printf("\nAverage Turnaround Time: %f\n", avg_turnaround_time);
}
```

//rr with at

```
#include <stdio.h>

int main() {
    int n;
    printf("Enter Total Number of Processes: ");
    scanf("%d", &n);

    int wt = 0, tt = 0;
    int arrt[n],burst[n], rburst[n];
    int q;

    for (int i = 0; i < n; i++) {
        printf("\nEnter Burst Time for Process %d: ", i + 1);
        scanf("%d", &burst[i]);
        printf("\nEnter arrival Time for Process %d: ", i + 1);
        scanf("%d", &arrt[i]);
        rburst[i] = burst[i];
    }

    printf("Enter Time Slice (Quantum): ");
    scanf("%d", &q);

    int total_time = 0;
    int completed_processes = 0;
```

```
    printf("\nProcess ID\tBurst Time\tArrival Time\tTurnaround Time\tWaiting Time\n");

    int curp = 0;

    while (completed_processes < n) {
        if (rburst[curp] > 0) {
            int exet;
            if (rburst[curp] > q) {
                exet = q;
            } else {
                exet = rburst[curp];
            }

            total_time += exet;
            rburst[curp] -= exet;

            if (rburst[curp] == 0) {
                completed_processes++;

                int turnaround_time = total_time-arrt[curp];
                int waiting_time = turnaround_time - burst[curp];

                printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n", curp + 1, burst[curp], arrt[curp],turnaround_time,
waiting_time);

                wt += waiting_time;
                tt += turnaround_time;
            }

            curp = (curp + 1) % n;
```

```
        } else {
            curp = (curp + 1) % n;
        }
    }

    float avg_waiting_time = (float)wt / n;
    float avg_turnaround_time = (float)tt / n;

    printf("\nAverage Waiting Time: %f", avg_waiting_time);
    printf("\nAverage Turnaround Time: %f\n", avg_turnaround_time);
}
```