CPU SCHEDULING PROGRAMS

1. FCFS without process structure

```c
#include <stdio.h>

int swap(int *a, int *b)
{
        int temp = *a;
        *a = *b;
        *b = temp;
}


void sort_by_at(int p[], int at[], int bt[], int n)
{
        for (int i = 0; i < n - 1; i++)
        {
                for (int j = 0; j < n - i - 1; j++)
                {
                        if (at[j] > at[j + 1])
                        {
                                swap(&at[j], &at[j + 1]);
                                swap(&bt[j], &bt[j + 1]);
                                swap(&p[j], &p[j + 1]);
                        }
                }
        }
}

void fcfs(int at[], int bt[], int wt[], int tat[], int n)
{
        int completion_time = 0;
        for (int i = 0; i < n; i++)
        {
                if (at[i] > completion_time)
                {
                        // If the process hasn't arrived yet, wait for it to
arrive
                        completion_time = at[i];
                }
                wt[i] = completion_time - at[i];
                tat[i] = wt[i] + bt[i];
                completion_time += bt[i];
        }
}

void findAvgTime(int p[], int at[], int bt[], int n)
{
        int wt[100], tat[100];
        int total_wt = 0, total_tat = 0;

        fcfs(at, bt, wt, tat, n);

        for (int i = 0; i < n; i++)
        {
                total_wt += wt[i];
                total_tat += tat[i];
        }

        printf("P\tAT\tBT\tWT\tTAT\n");
        for (int i = 0; i < n; i++)
        {
                printf("P%d\t%d\t%d\t%d\t%d\n", p[i], at[i], bt[i], wt[i],
tat[i]);
        }

        printf("\nAverage Waiting Time = %.2f", (float)total_wt / (float)n);
        printf("\nAverage Turn Around Time = %.2f \n", (float)total_tat /
(float)n);
}

int main()
{
        int n;
        int p[100], bt[100], at[100];
        printf("Enter the number of processes:");
        scanf("%d", &n);
        for (int i = 0; i < n; i++)
        {
                printf("Process %d (Burst Time):", i + 1);
                scanf("%d",&bt[i]);
                at[i]=0;
                p[i] = i + 1;
        }
        sort_by_at(p, at, bt, n);
        findAvgTime(p, at, bt, n);
        return 0;
}
```

2. FCFS using process structures and dynamic memory allocation

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

typedef struct Process
{
        int p_num;
        int bt;
        int at;
        int wt;
        int tat;
} Process;


void swap(Process *xp, Process *yp)
{
        Process temp = *xp;
        *xp = *yp;
        *yp = temp;
}


void sort_by_at(Process *p, int n)
{
        for (int i = 0; i < n - 1; i++)
        {
                for (int j = 0; j < n - i - 1; j++)
                {
                        if (p[j].at > p[j + 1].at)
                        {
                                swap(&p[j], &p[j + 1]);
                        }
                }
        }
}

void fcfs(Process *p, int n)
{
        int completion_time = 0;
        for (int i = 0; i < n; i++)
        {
                if (p[i].at > completion_time)
                {
                        completion_time = p[i].at;
                }
                p[i].wt = completion_time - p[i].at;
                p[i].tat = p[i].wt + p[i].bt;
                completion_time += p[i].bt;
        }
}

void findAvgTime(Process *p, int n)
{
        fcfs(p, n);
        int total_wt = 0;
        int total_tat = 0;
        for (int i = 0; i < n; i++)
        {
                total_wt += p[i].wt;
                total_tat += p[i].tat;
        }
        printf("P\tAT\tBT\tWT\tTAT\n");
        for (int i = 0; i < n; i++)
        {
                printf("P%d\t%d\t%d\t%d\t%d\n", p[i].p_num, p[i].at, p[i].bt,
p[i].wt, p[i].tat);
        }

        printf("\nAverage Waiting Time = %.2f", (float)total_wt / (float)n);
        printf("\nAverage Turn Around Time = %.2f \n", (float)total_tat /
(float)n);
}

int main()
{
        int n;
        printf("Enter the number of processes:");
        scanf("%d", &n);
        Process *p;
        p = (Process *)malloc(sizeof(Process) * n);
        for (int i = 0; i < n; i++)
        {
                printf("Process %d (Burst Time):", i + 1);
                scanf("%d",&p[i].bt);
                p[i].at=0;
                p[i].p_num = i + 1;
        }
        sort_by_at(p, n);
        findAvgTime(p, n);
        free(p);
        return 0;
}
```

3. non-preemptive SJF without process structure

```c
#include <stdio.h>

int main()
{
    int at[10], bt[10], p[10];
    int n, i, j, temp, exit_time = 0, count, over = 0, sum_wait = 0,
sum_turnaround = 0, start;
    float avg_wait, avg_turn;

    printf("Enter the number of processes: ");
    scanf("%d", &n);

    for(int i=0; i<n; i++)
    {
        printf("Enter the arrival time and execution time for process %d\n",
i+1);
        scanf("%d %d", &at[i], &bt[i]);
        p[i] = i+1;
    }

    for(int i=0; i<n; i++)
    {
        for(int j=i+1; j<n; j++)
        {
            if(at[i] > at[j])
            {
                temp = at[i]; at[i] = at[j]; at[j] = temp;
                temp = bt[i]; bt[i] = bt[j]; bt[j] = temp;
                temp = p[i]; p[i] = p[j]; p[j] = temp;
            }
        }
    }

    printf("\nProcess\t\tAT\t\tBT\t\tStart\t\tWT\t\tTAT\n");

    while (over < n)
    {
        count = 0;
        for (int i=over; i<n; i++)
        {
            if (at[i] <= exit_time)
            {
                count++;
            }
            else
            {
                break;
            }
        }

        if (count > 1)
        {
            for (i=over; i<over+count-1; i++)
            {
                for (j=i+1; j<over+count; j++)
                {
                    if (bt[i] > bt[j])
                    {
                        temp = at[i]; at[i] = at[j]; at[j] = temp;
                        temp = bt[i]; bt[i] = bt[j]; bt[j] = temp;
                        temp = p[i]; p[i] = p[j]; p[j] = temp;
                    }
                }
            }
        }

        start = exit_time;
        exit_time += bt[over];
        int waiting_time = start - at[over];
        int turnaround_time = exit_time - at[over];

        printf("p[%d]\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", p[over], at[over],
bt[over], start, waiting_time, turnaround_time);

        sum_wait += waiting_time;
        sum_turnaround += turnaround_time;
        over++;
    }

    avg_wait = (float)sum_wait / (float)n;
    avg_turn = (float) sum_turnaround / (float)n;
    printf("Average waiting time is %f\n", avg_wait);
    printf("Average turnaround time is %f\n", avg_turn);
    return 0;
}
```

4. non-preemptive SJF using process structure and dynamic memory allocation

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

struct Process {
    int pid;
    int arrival_time;
    int burst_time;
};

int main()
{
    struct Process *processes;
    int n, temp, exit_time = 0, count, over = 0, sum_wait = 0, sum_turnaround =
0, start;
    float avg_wait, avg_turn;

    printf("Enter the number of processes: ");
    scanf("%d", &n);

    processes = (struct Process *)malloc(n * sizeof(struct Process));

    for(int i = 0; i < n; i++)
    {
        processes[i].pid = i + 1;
        printf("Enter the arrival time and execution time for process %d\n", i +
1);
        scanf("%d %d", &processes[i].arrival_time, &processes[i].burst_time);
    }

    for(int i = 0; i < n; i++)
    {
        for(int j = i + 1; j < n; j++)
        {
            if(processes[i].arrival_time > processes[j].arrival_time)
            {
```

```c
                struct Process temp_proc = processes[i];
                processes[i] = processes[j];
                processes[j] = temp_proc;
            }
        }
    }

    printf("\nProcess\t\tAT\t\tBT\t\tStart\t\tWT\t\tTAT\n");

    while (over < n)
    {
        count = 0;
        for (int i = over; i < n; i++)
        {
            if (processes[i].arrival_time <= exit_time)
            {
                count++;
            }
            else
            {
                break;
            }
        }

        if (count > 1)
        {
            for (int i = over; i < over + count - 1; i++)
            {
                for (int j = i + 1; j < over + count; j++)
                {
                    if (processes[i].burst_time > processes[j].burst_time)
                    {
                        struct Process temp_proc = processes[i];
                        processes[i] = processes[j];
                        processes[j] = temp_proc;
                    }
                }
            }
        }

        start = exit_time;
        exit_time += processes[over].burst_time;
        int waiting_time = start - processes[over].arrival_time;
        int turnaround_time = exit_time - processes[over].arrival_time;

        printf("p[%d]\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", processes[over].pid, processes[over].arrival_time, processes[over].burst_time, start, waiting_time, turnaround_time);

        sum_wait += waiting_time;
        sum_turnaround += turnaround_time;
        over++;
    }

    avg_wait = (float)sum_wait / (float)n;
    avg_turn = (float)sum_turnaround / (float)n;
    printf("Average waiting time is %f\n", avg_wait);
    printf("Average turnaround time is %f\n", avg_turn);

    free(processes);
    return 0;
}
```

5. round robin without process structure

```c
#include <stdio.h>

void main()
{
        int i=0, sum = 0, wt = 0, tat = 0, count = 0, quant, at[10], bt[10],
temp[10], NOP, y = 0, time = 0;
        printf("Enter the number of processes: ");
        scanf("%d", &NOP);
        y = NOP;

        for(int i=0; i<NOP; i++)
        {
                printf("Enter the Arrival time, Burst time for process: %d\n",
i+1);
                printf("Enter the Arrival Time: ");
                scanf("%d", &at[i]);
                printf("Enter the Burst Time: ");
                scanf("%d", &bt[i]);
                temp[i] = bt[i];
        }

        printf("Enter the time quantum: ");
        scanf("%d", &quant);
        printf("\nProcess\t\t\tArrival Time\t\tBurst Time\t\tWaiting Time\t\tTurn
Around Time\n");
        for(sum=0, i=0; y!=0;)
        {
                if(temp[i]<=quant && temp[i]>0)
                {
                        time = time + temp[i];
                        temp[i] = 0;
                        count = 1;
                }
                else if(temp[i]>0)
                {
                        temp[i] = temp[i] - quant;
                        time = time + quant;
                }
                if(temp[i]==0 && count==1)
                {
                        y--;
                        printf("PID[%d]\t\t\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d\n",
i+1, at[i], bt[i], time-at[i]-bt[i], time-at[i]);
                        wt = wt + (time - at[i] - bt[i]);
                        tat = tat + (time - at[i]);
                        count = 0;
                }

                if(i==NOP-1)
                {
                        i=0;
                }
                else if(at[i+1]<=time)
                {
                        i++;
                }
                else
                {
                        i=0;
                }

        }
        printf("Average Waiting Time: %d\n", wt/NOP);
        printf("Average Turn Around Time: %d\n", tat/NOP);
}
```

6. round robin with process structure and dynamic memory allocation

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

struct process{
        int at;
        int bt;
        int wt;
        int tat;
};
```

```
void main()
{
        int i = 0, sum = 0, count = 0, quant, NOP, y = 0, time = 0, total_wt = 0,
total_tat = 0;
        printf("Enter the number of processes: ");
        scanf("%d", &NOP);
        y = NOP;

        struct process *p, *temp;
        p = (struct process *)malloc(NOP * sizeof(struct process));
        temp = (struct process *)malloc(NOP * sizeof(struct process));

        for(int i=0; i<NOP; i++)
        {
                printf("Enter the Arrival time, Burst time for process: %d\n",
i+1);
                printf("Enter the Arrival Time: ");
                scanf("%d", &p[i].at);
                printf("Enter the Burst Time: ");
                scanf("%d", &p[i].bt);
                temp[i].bt = p[i].bt;
        }

        printf("Enter the time quantum: ");
        scanf("%d", &quant);
        printf("\nProcess\t\t\tArrival Time\t\tBurst Time\t\tWaiting Time\t\tTurn
Around Time\n");
        for(sum=0, i=0; y!=0;)
        {
                if(temp[i].bt<=quant && temp[i].bt>0)
                {
                        time = time + temp[i].bt;
                        temp[i].bt = 0;
                        count = 1;
                }
                else if(temp[i].bt>0)
                {
                        temp[i].bt = temp[i].bt - quant;
                        time = time + quant;
                }

                if(temp[i].bt==0 && count==1)
                {
                        y--;
                        printf("PID[%d]\t\t\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d\n",
i+1, p[i].at, p[i].bt, time-p[i].at-p[i].bt, time-p[i].at);
                        total_wt = total_wt + p[i].wt + (time - p[i].at -
p[i].bt);
                        total_tat = total_tat + p[i].tat + (time - p[i].at);
                        count = 0;
                }

                if(i==NOP-1)
                {
                        i=0;
                }
                else if(p[i+1].at<=time)
                {
                        i++;
                }
                else
                {
                        i=0;
                }

        }
        printf("Average Waiting Time: %d\n", total_wt/NOP);
        printf("Average Turn Around Time: %d\n", total_tat/NOP);
}
```

7. non-preemptive priority without process structure

```c
#include <stdio.h>

void swap(int *a, int *b)
{
        int temp = *a;
        *a = *b;
        *b = temp;
}

int main()
{
        int p[100], bt[100], index[100];
        int n;
        printf("Enter the no. of processes: ");
        scanf("%d", &n);
        for(int i=0; i<n; i++)
        {
                printf("Enter the burst time and priority value for the 
process %d\n", i+1);
                scanf("%d %d", &bt[i], &p[i]);
                index[i]=i+1;
        }

        for(int i=0; i<n; i++)
        {
                int a=p[i], m=i;
                for(int j=i+1; j<n; j++)
                {
                        if(p[j]<a)
                        {
                                a = p[j];
                                m = j;
                        }
                }

                swap(&p[i], &p[m]);
                swap(&bt[i], &bt[m]);
                swap(&index[i], &index[m]);
        }
        int t=0;
        printf("\nORDER OF PROCESS EXECUTION\n");
        for(int i=0; i<n; i++)
        {
                printf("p%d is executed from %d to %d \n", index[i], t, t+bt[i]);
                t+=bt[i];
        }

        printf("\npid \t\t BT \t\t WT \t\t TAT\n");
        int wt = 0;
        for(int i=0; i<n; i++)
        {
                printf("%d \t\t %d \t\t %d \t\t %d\n", index[i], bt[i], wt,
wt+bt[i]);
                wt+=bt[i];
        }
        return 0;
}
```

8. non-preemptive priority with process structure and dynamic memory allocation

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

typedef struct
{
    int pid;
    int bt;
    int priority;
    int wt;
    int tat;
} Process;

void swap(Process *a, Process *b)
{
    Process temp = *a;
    *a = *b;
    *b = temp;
}

int main()
{
    Process *p;
    int n;

    printf("Enter the number of processes: ");
    scanf("%d", &n);

    p = (Process *)malloc(n * sizeof(Process));

    for (int i = 0; i < n; i++)
    {
        printf("Enter the burst time and priority value for process %d: ", i +
1);
        scanf("%d %d", &p[i].bt, &p[i].priority);
        p[i].pid = i + 1;
    }



    for (int i = 0; i < n; i++)
    {
        int min_priority = p[i].priority, m = i;
        for (int j = i + 1; j < n; j++)
        {
            if (p[j].priority < min_priority)
            {
                min_priority = p[j].priority;
                m = j;
            }
        }

        swap(&p[i], &p[m]);
    }

    int t = 0;
    printf("\nORDER OF PROCESS EXECUTION\n");
    for (int i = 0; i < n; i++)
    {
        printf("Process %d is executed from %d to %d\n", p[i].pid, t, t +
p[i].bt);
        t += p[i].bt;
    }

    printf("\nPID\tBT\tWT\tTAT\n");
    int wt = 0;
```

```c
    for (int i = 0; i < n; i++)
    {
        p[i].wt = wt;
        p[i].tat = wt + p[i].bt;
        printf("%d\t%d\t%d\t%d\n", p[i].pid, p[i].bt, p[i].wt, p[i].tat);
        wt += p[i].bt;
    }

    free(p);

    return 0;
}
```