AMAL BABU
20222016
CS A S6

SHELL SCRIPTING

SUM

```
echo "Enter two digits:"
read a b
sum=$((a + b))
echo "Sum = $sum"
```

OUTPUT

```
Enter two digits:
5 7
Sum = 12
```

ODD EVEN

```
echo "Enter number:"
read a
b=`expr $a % 2`
if [ $b -eq 0 ]
then
        echo "$a is even"
else
        echo "$a is odd"
fi
```

OUTPUT

```
Enter number:
153
153 is odd

Enter number:
22
22 is even
```

AREA OF CIRCLE

```
echo "Enter the radius of circle:"
read r
c=3.14
a=`echo $c \* $r \* $r|bc`
echo "Area = $a"
```

OUTPUT

```
Enter the radius of circle:
2
Area = 12.56
```

PRIME NUMBER

```
echo "Enter the number:"
read a
i=2
f=0
b=`echo $a/2|bc`
if [ $a -eq 0 -o $a -eq 1 ]
then
        echo "Not a prime number"
else
        while [ $i -le $b ]
        do
                if [ `expr $a % $i` -eq 0 ]
                then
                        f=1
                        break
                fi
                i=`expr $i + 1`
        done
        if [ $f -eq 0 ]
        then
                echo "Prime Number"
        else
                echo "Not a prime number"
        fi
fi
```

OUTPUT

```
Enter the number:
17
Prime Number
```

```
Enter the number:
22
Not a prime number
```

SWAP TWO NUMBERS

```
echo "Enter a and b:"
read a b
echo "Before swap a:$a and b:$b"
temp=$a
a=$b
b=$temp
echo "After swap a:$a and b:$b"
```

OUTPUT

```
echo "Enter a and b:"
read a b
echo "Before swap a:$a and b:$b"
temp=$a
a=$b
b=$temp
echo "After swap a:$a and b:$b"
```

LARGEST OF TWO NUMBERS

```
echo "Enter two numbers:"
read a b
if [ $a -gt $b ]
then
        echo "$a is greater than $b"
else
        echo "$b is greater than $a"
fi

#OUTPUT
#Enter two numbers:
#4 5
#5 is greater than 4

#Enter two numbers:
#8 2
#8 is greater than 2
```

SUM OF DIGITS

```
echo "Enter digit:"
read a
sum=0
while [ $a -gt 0 ]
do
        b=`expr $a % 10`
        sum=`expr $sum + $b`
        a=`expr $a / 10`
done
echo "Sum=$sum"
```

OUTPUT

```
Enter digit:
154
Sum=10
```

LARGEST OF THREE NUMBERS

```
echo "Enter three numbers:"
read a b c
if [ $a -gt $b ]
then
        if [ $a -gt $c ]
        then
                echo "$a is greater"
        else
                echo "$c is greater"
        fi
else
        if [ $b -gt $c ]
        then
                echo "$b is greater"
        else
                echo "$c is greater"
        fi
fi

#OUTPUT
#Enter three numbers:
#22 20 21
#22 is greater

#Enter three numbers:
#55 60 54
#60 is greater
```

CALCULATOR

```
echo "Enter two numbers:"
read a b
echo "Enter the operation:1)Addition 2)Subtraction 3)Multiplication 4)Division"
read opt
case "$opt" in
	"1")r=$((a+b));;
	"2")r=$((a-b));;
	"3")r=$((a*b));;
	"4")r=`echo "scale=2; $a/$b " |bc`;;
esac
echo "Result = $r"
```

OUTPUT

```
Enter two numbers:
12 13
Enter the operation:1)Addition 2)Subtraction 3)Multiplication 4)Division
1
Result = 25

Enter two numbers:
2 3
Enter the operation:1)Addition 2)Subtraction 3)Multiplication 4)Division
3
Result = 6
```

LEAP YEAR

```
echo "Enter year:"
read y
if [ `expr $y % 4` -eq 0 ]
then
	if [ `expr $y % 100` -eq 0 ]
	then
		if [ `expr $y % 400` -eq 0 ]
		then
			echo "$y is a leap year"
		else
			echo "$y is not a leap year"
		fi
	else
		echo "$y is a leap year"
	fi
else
```

```
        echo "$y is not a leap year"
fi
```

OUTPUT

```
Enter year:
2016
2016 is a leap year

Enter year:
2022
2022 is not a leap year
```

ARMSTRONG NUMBER

```
echo "Enter the number:"
read num
a=$num
s=0
while [ $a -gt 0 ]
do
        b=$((a%10))
        c=$b
        c=$((c*c))
        c=$((c*b))
        s=$((s+c))
        a=$((a/10))
done
if [ $s -eq $num ]
then
        echo "Armstrong Number"
else
        echo "Not an Armstrong Number"
fi
```

OUTPUT

```
Enter the number:
153
Armstrong Number

Enter the number:
156
Not an Armstrong Number
```

FACTORIAL

```
echo "Enter the number:"
read num
fact=1
while [ $num -gt 1 ]
do
        fact=$((fact*num))
        num=$((num-1))
done
echo "Factorial = $fact"
```

OUTPUT

```
Enter the number:
5
Factorial = 120
```

STRING CONCATENATION

```
echo "Enter first string:"
read str1
echo "Enter second string:"
read str2
concat="$str1$str2"
echo "Concatenated string: $concat"
```

OUTPUT

```
Enter first string:
Hello
Enter second string:
World
Concatenated string: HelloWorld
```

STRING REVERSAL

```
echo "Enter a string:"
read str
rev=""
len=${#str}
```

```
for (( i=$len-1; i>=0; i-- ))
do
   rev="$rev${str:$i:1}"
done

echo "Reversed string: $rev"
```

OUTPUT

```
Enter a string:
Hello
Reversed string: olleH
```

CHARACTER COUNT

```
echo "Enter a string:"
read str
count=${#str}
echo "Number of characters: $count"
```

OUTPUT

```
Enter a string:
Shell
Number of characters: 5
```

STRING REVERSAL

```
#!/bin/bash

while true
do
   echo ""
   echo "-------- STRING OPERATIONS MENU --------"
   echo "1. Convert to Uppercase"
   echo "2. Convert to Lowercase"
   echo "3. Replace a Substring"
   echo "4. Find Position of a Substring"
   echo "5. Exit"
   echo "---------------------------------------"
   echo -n "Enter your choice: "
   read choice

   case $choice in
      1)
      echo -n "Enter the string: "
      read str
      upper=$(echo "$str" | tr '[:lower:]' '[:upper:]')
      echo "Uppercase: $upper"
      ;;
    2)
      echo -n "Enter the string: "
      read str
      lower=$(echo "$str" | tr '[:upper:]' '[:lower:]')
      echo "Lowercase: $lower"
      ;;
    3)
      echo -n "Enter the original string: "
      read str
      echo -n "Enter the substring to replace: "
      read old
      echo -n "Enter the new substring: "
      read new
      replaced=${str//$old/$new}
      echo "Modified string: $replaced"
      ;;
    4)
      echo -n "Enter the main string: "
      read str
      echo -n "Enter the substring to find: "
      read sub
      pos=$(expr index "$str" "$sub")
      if [ $pos -eq 0 ]; then
        echo "Substring not found."
      else
        echo "Substring found at position: $pos"
      fi
      ;;
    5)
      echo "Exiting..."
      break
      ;;
    *)
      echo "Invalid choice. Please try again."
      ;;
  esac
done
```

OUTPUT

```
-------- STRING OPERATIONS MENU --------
1. Convert to Uppercase
2. Convert to Lowercase
3. Replace a Substring
4. Find Position of a Substring
5. Exit
---------------------------------------
```

```
Enter your choice: 1
Enter the string: hello world
Uppercase: HELLO WORLD

-------- STRING OPERATIONS MENU --------
Enter your choice: 2
Enter the string: HeLLo WoRLD
Lowercase: hello world

-------- STRING OPERATIONS MENU --------
Enter your choice: 3
Enter the original string: shell scripting is fun
Enter the substring to replace: fun
Enter the new substring: powerful
Modified string: shell scripting is powerful

-------- STRING OPERATIONS MENU --------
Enter your choice: 4
Enter the main string: scripting
Enter the substring to find: rip
Substring found at position: 4

-------- STRING OPERATIONS MENU --------
Enter your choice: 5
Exiting...
```

EQUAL STRING

```
echo "Enter first string:"
read str1
echo "Enter second string:"
read str2

if [ "$str1" = "$str2" ]
then
   echo "Strings are equal"
else
   echo "Strings are not equal"
fi
```

OUTPUT

```
Enter first string:
code
Enter second string:
code
Strings are equal
```

REMOVAL OF SPACES

```
echo “enter string”
read input
trimmed=$( echo “$input” | sed ‘s/^ *//;s/ *$//’)
echo “trimmed string:  ‘$trimmed’ ”
```

OUTPUT

```
enter string:
            Hello
trimmed string : ‘Hello’
```

SYSTEM CALLS

FORK

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main() {
    pid_t pid;

    pid = fork();
    if (pid < 0) {
        perror("Fork failed");
        return 1;
    } else if (pid == 0) {
         printf("This is the child process.\n");
        printf("Child PID: %d\n", getpid());
        printf("Parent PID: %d\n", getppid());
    } else {
        printf("This is the parent process.\n");
        printf("Parent PID: %d\n", getpid());
        printf("Child PID: %d\n", pid);
    }

    return 0;
}
```

OUTPUT

```
This is the parent process.
Parent PID: 3456
Child PID: 3457

This is the child process.
Child PID: 3457
Parent PID: 3456
```

SLEEP & WAIT

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    pid_t pid;

    pid = fork();
    if (pid < 0) {
        perror("Fork failed");
        return 1;
    } else if (pid == 0) {
        printf("Child process started (PID: %d)\n", getpid());
        sleep(3);
        printf("Child process completed\n");
        exit(0);
    } else {
        printf("Parent process waiting for child to finish...\n");
        wait(NULL);
        printf("Parent process resumes after child terminates\n");
    }

    return 0;
}
```

OUTPUT

```
Parent process waiting for child to finish...
Child process started (PID: 4587)
Child process completed
Parent process resumes after child terminates
```

COPY FILE

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

int main(int argc, char *argv[])
{
    int fd1, fd2;
    char buffer[1024];
    long int n1;

    if(((fd1 = open(argv[1], O_RDONLY)) == -1) ||
((fd2=open(argv[2],O_CREAT|O_WRONLY|O_TRUNC, 0700)) == -1)){
        perror("file problem");
        exit(1);
    }

    while((n1=read(fd1, buffer, 1024)) > 0){
        if(write(fd2, buffer, n1) != n1){
            perror("writing problem ");
            exit(3);
        }
    }
    close(fd1);
    close(fd2);
}
```

OUTPUT

```
OLD CONTENT OF F1:HELLO WORLD
OLD CONTENT OF F2:

NEW CONTENT OF F1:HELLO WORLD
NEW CONTENT OF F2:HELLO WORLD
```

SCHEDULING PROGRAMS

FCFS

```
#include <stdio.h>

struct Process {
    int id;
    int arrival;
    int burst;
    int completion;
    int turnAround;
    int waiting;
};

void FCFS(struct Process p[], int n) {
    int totalWaiting = 0, totalTurnaround = 0;

    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (p[i].arrival > p[j].arrival) {
                struct Process temp = p[i];
                p[i] = p[j];
                p[j] = temp;
            }
        }
    }

    p[0].completion = p[0].completion + p[0].burst;
    for (int i = 1; i < n; i++) {
        p[i].completion = p[i - 1].completion + p[i].burst;
    }
    for (int i = 0; i < n; i++) {
        p[i].turnAround = p[i].completion - p[i].arrival;
        p[i].waiting = p[i].turnAround - p[i].burst;
        totalWaiting += p[i].waiting;
        totalTurnaround += p[i].turnAround;
    }

    printf("\nGantt Chart:\n");
    for (int i = 0; i < n; i++) {
        printf("| P%d ", p[i].id);
    }
    printf("|\n");

    printf("  ");
    for (int i = 0; i < n; i++) {
        printf("  %d ", p[i].completion);
    }
    printf("\n");

    printf("\nProcess ID\tArrival Time\tBurst Time\tCompletion Time\tTurnaround
Time\tWaiting Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", p[i].id, p[i].arrival, p[i].burst,
p[i].completion, p[i].turnAround, p[i].waiting);
    }

    printf("\nAverage Waiting Time: %.2f", (float)totalWaiting / n);
    printf("\nAverage Turnaround Time: %.2f", (float)totalTurnaround / n);
}

int main() {
    int n;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    struct Process p[n];

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

    FCFS(p, n);
    return 0;
}
```

OUTPUT

```
Enter the number of processes:3

Enter the details of process 1
Burst Time:1
Arrival Time:2

Enter the details of process 2
Burst Time:4
Arrival Time:5

Enter the details of process 3
Burst Time:2
Arrival Time:3
Processes   Burst time   Arrival time   Waiting time   Turn around time
   1          1          2            0          1
   2          4          5            0          5
   3          2          3            1          6
Average waiting time = 0.333333
Average turn around time = 4.000000

Gantt Chart:
--------------------------------
   P1      P2      P3
--------------------------------
0    3        9        11
```

SJFS

```
#include <stdio.h>

struct Process {
    int id;
    int arrival;
    int burst;
    int completion;
    int turnAround;
    int waiting;
    int remaining;
};

void FCFS(struct Process p[], int n) {
    int totalWaiting = 0, totalTurnaround = 0;
    int finished=0,time=0,minIndex,minBurst;
    int store[n];
    int k=0;

    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (p[i].arrival > p[j].arrival) {
                struct Process temp = p[i];
                p[i] = p[j];
```

```
          p[j] = temp;
        }
      }
    }

    for(int i=0;i<n;i++){
    p[i].remaining=p[i].burst;
    }
    while(finished<n)
       {
       minIndex=-1;
       minBurst=9999;
       for(int i=0;i<n;i++)
            {
            if(p[i].arrival<=time&&p[i].remaining>0&&p[i].remaining<minBurst)
                 {
                 minBurst=p[i].remaining;
                 minIndex=i;
                 }
            }
       if(minIndex==-1)
            {
            time++;
            continue;
            }
       store[k++]=minIndex;
       p[minIndex].remaining=p[minIndex].burst;
       time += p[minIndex].remaining;
       p[minIndex].completion=time;
       p[minIndex].turnAround=p[minIndex].completion-p[minIndex].arrival;
       p[minIndex].waiting=p[minIndex].turnAround-p[minIndex].burst;
       p[minIndex].remaining=0;
       finished++;
       }



    printf("\nGantt Chart:\n");
    for (int i = 0; i < n; i++) {
      printf("| P%d ", p[store[i]].id);
    }
    printf("|\n");

    printf("  ");
    for (int i = 0; i < n; i++) {
      printf("  %d ", p[store[i]].completion);
    }
    printf("\n");

    printf("\nProcess ID\tArrival Time\tBurst Time\tCompletion Time\tTurnaround
  Time\tWaiting Time\n");
    for (int i = 0; i < n; i++) {
       totalWaiting += p[store[i]].waiting;
       totalTurnaround += p[store[i]].turnAround;
      printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", p[store[i]].id, p[store[i]].arrival,
p[store[i]].burst, p[store[i]].completion, p[store[i]].turnAround, p[store[i]].waiting);
    }

    printf("\nAverage Waiting Time: %.2f", (float)totalWaiting / n);
    printf("\nAverage Turnaround Time: %.2f", (float)totalTurnaround / n);
}

int main() {
    int n;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    struct Process p[n];

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

    FCFS(p, n);
    return 0;
}
```

OUTPUT

```
Enter number of processes: 3
Enter Arrival Time and Burst Time for Process 1: 2
5
Enter Arrival Time and Burst Time for Process 2: 1
3
Enter Arrival Time and Burst Time for Process 3: 6
8

Gantt Chart:
| P2 | P1 | P3 |
   4   9   17
Process ID       Arrival Time      Burst Time       Completion Time    Turnaround Time     Waiting
Time
2               1              3              4              3              0
1               2              5              9              7              2
3               6              8              17              11              3

Average Waiting Time: 1.67
Average Turnaround Time: 7.00
```

SJF PREMPTIVE

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

struct Process {
    int id,at, bt, ct, tat, wt, remaining_bt;
```

```
};

void swap(int *a, int *b)
{
int temp=*a;
*a=*b;
*b=temp;
}
void sjf_preemptive( struct Process p[],int n)
{
int completed=0,time=0,min_index,ttat=0,twt=0;
int gantt[100],gantt_time[100],gantt_size=0;
float atat,awt;

while(completed < n)
{
min_index=-1;
int min_bt=INT_MAX;

for(int i=0;i<n;i++)
{
if(p[i].at<= time && p[i].remaining_bt > 0 && p[i].remaining_bt < min_bt)
{
min_bt=p[i].remaining_bt;
min_index=i;
}
}
if(min_index==-1)
{
time++;
continue;
}
p[min_index].remaining_bt--;

gantt[gantt_size++]=min_index;

if(p[min_index].remaining_bt ==0)
{
p[min_index].ct=time+1;
p[min_index].tat=p[min_index].ct - p[min_index].at;
p[min_index].wt=p[min_index].tat - p[min_index].bt;
completed++;
}
time++;
}
printf("\nProcess\tAT\tBT\tCT\tTAT\tWT\n");
for (int i=0;i<n;i++)
{
printf("P%d\t%d\t%d\t%d\t%d\t%d\n",p[i].id,p[i].at,p[i].bt,p[i].ct,p[i].tat,p[i].wt);
}
printf("\nGantt chart:");
   for (int i = 0; i < gantt_size; i++) {
      printf("|P%d", p[gantt[i]].id);
   }
   printf("|\n");
```

```
    for(int i=0;i<n;i++)
    {
    ttat+=p[i].tat;
    twt+=p[i].wt;
    }
    atat=(float)ttat/n;
    awt=twt/n;
    printf("Average turn around time=%f",atat);
    printf("\nAverage waiting time=%f",awt);
}

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

    struct Process p[n];

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

    sjf_preemptive(p, n);

    return 0;
}
```

OUTPUT

```
Enter number of processes: 6
Enter the process id, arrival time and burst time: 1 0 7
Enter the process id, arrival time and burst time: 2 1 5
Enter the process id, arrival time and burst time: 3 2 3
Enter the process id, arrival time and burst time: 4 3 1
Enter the process id, arrival time and burst time: 5 4 2
Enter the process id, arrival time and burst time: 6 5 1

Process   AT      BT      CT      TAT     WT
P1      0       7       19      19      12
P2      1       5       13      12      7
P3      2       3       6       4       1
P4      3       1       4       1       0
P5      4       2       9       5       3
P6      5       1       7       2       1

Gantt chart:|P1|P2|P3|P4|P3|P3|P6|P5|P5|P2|P2|P2|P2|P1|P1|P1|P1|P1|P1|
Average turn around time=7.166667
Average waiting time=4.000000
```

NON PREEMPTIVE PRIORITY

```
#include<stdio.h>

struct Process
{
   char name[5];
   int at, bt, ct, tat, wt, priority, completed;
};

void sortByArrival(struct Process p[], int n) {
   struct Process temp;
   for (int i = 0; i < n - 1; i++) {
      for (int j = i + 1; j < n; j++) {
         if (p[i].at > p[j].at) {
            temp = p[i];
            p[i] = p[j];
            p[j] = temp;
         }
      }
   }
}
void nonpreemptivePriority(struct Process p[], int n)
{
   int currentTime = 0, completed = 0;
   printf("\nGantt chart: ");

   while (completed < n)
   {
    int highestPriority = -1;
      for (int i = 0; i < n; i++)
      {
         if (!p[i].completed && p[i].at <= currentTime)
         {
            if (highestPriority == -1 || p[i].priority < p[highestPriority].priority)
               highestPriority = i;
         }
      }

      if (highestPriority == -1)
      {
         currentTime++;
         continue;
      }

      printf("|%s", p[highestPriority].name);
      currentTime+=p[highestPriority].bt;
      p[highestPriority].ct=currentTime;
      p[highestPriority].tat=p[highestPriority].ct - p[highestPriority].at;
      p[highestPriority].wt=p[highestPriority].tat - p[highestPriority].bt;
      p[highestPriority].completed=1;
      completed++;
```

```
        }

    printf("|\n");
}

void display(struct Process p[], int n)
{
int ttat=0,twt=0;
float atat,awt;
    printf("\nProcess\tAT\tBT\tCT\tTAT\tWT\n");
    for (int i = 0; i < n; i++)
        printf("%s\t%d\t%d\t%d\t%d\t%d\n", p[i].name, p[i].at, p[i].bt, p[i].ct, p[i].tat,
p[i].wt);
          printf("|\n");
        for(int i=0;i<n;i++)
    {
    ttat+=p[i].tat;
    twt+=p[i].wt;
    }
    atat=(float)ttat/n;
    awt=(float)twt/n;
    printf("Average turn around time=%f",atat);
    printf("\nAverage waiting time=%f",awt);
}

int main()
{
    int n;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    struct Process p[n];

    for (int i = 0; i < n; i++)
    {
        printf("Enter process name:\n");
        scanf("%s", p[i].name);
        printf("Enter arrival time, burst time, priority:\n");
        scanf("%d", &p[i].at);
        scanf("%d", &p[i].bt);
        scanf("%d", &p[i].priority);
        p[i].completed = 0;
    }
    sortByArrival(p,n);
    nonpreemptivePriority(p, n);
    display(p, n);
    return 0;
}
```

```
Enter number of processes: 5
Enter process name:
p1
Enter arrival time, burst time, priority:
0 4 2
Enter process name:
p2
```

```
Enter arrival time, burst time, priority:
1 3 3
Enter process name:
p3
Enter arrival time, burst time, priority:
2 1 6
Enter process name:
p4
Enter arrival time, burst time, priority:
3 5 5
Enter process name:
p5
Enter arrival time, burst time, priority:
4 2 5

Gantt chart: |p1|p2|p4|p5|p3|

Process   AT      BT      CT      TAT     WT
p1       0       4       4       4       0
p2       1       3       7       6       3
p3       2       1       15      13      12
p4       3       5       12      9       4
p5       4       2       14      10      8
|
Average turn around time=8.400000
Average waiting time=5.400000
```

PRIORITY PREMPTIVE

```
#include<stdio.h>

struct Process
{
   char name[5];
   int at, bt, ct, tat, wt, priority, remaining_bt, completed;
};

void preemptivePriority(struct Process p[], int n)
{
   int currentTime = 0, completed = 0, minPriority;
   int ganttChart[100];
   int ganttIndex = 0;

   while (completed < n)
   {
      minPriority = -1;
      for (int i = 0; i < n; i++)
      {
         if (p[i].at <= currentTime && p[i].remaining_bt > 0)
```

```
            {
                if (minPriority == -1 || p[i].priority < p[minPriority].priority)
                    minPriority = i;
            }
        }

        if (minPriority == -1)
        {
            currentTime++;
            continue;
        }


        if (ganttIndex == 0 || ganttChart[ganttIndex - 1] != minPriority) {
            ganttChart[ganttIndex++] = minPriority;
        }

        p[minPriority].remaining_bt--;

        if (p[minPriority].remaining_bt == 0)
        {
            p[minPriority].ct = currentTime + 1;
            p[minPriority].tat = p[minPriority].ct - p[minPriority].at;
            p[minPriority].wt = p[minPriority].tat - p[minPriority].bt;
            completed++;
        }

        currentTime++;
    }


    printf("\nGantt chart: ");
    for (int i = 0; i < ganttIndex; i++)
        printf("|%s", p[ganttChart[i]].name);
    printf("|\n");
}

void display(struct Process p[], int n)
{
    printf("\nProcess\tAT\tBT\tCT\tTAT\tWT\n");
    for (int i = 0; i < n; i++)
        printf("%s\t%d\t%d\t%d\t%d\t%d\n", p[i].name, p[i].at, p[i].bt, p[i].ct, p[i].tat,
p[i].wt);
}

int main()
{
    int n;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    struct Process p[n];

    for (int i = 0; i < n; i++)
    {
        printf("Enter process name:\n");
        scanf("%s", p[i].name);
        printf("Enter arrival time, burst time, priority:\n");
        scanf("%d", &p[i].at);
        scanf("%d", &p[i].bt);
        scanf("%d", &p[i].priority);
        p[i].remaining_bt = p[i].bt;
        p[i].completed = 0;
    }

    preemptivePriority(p, n);
    display(p, n);
    return 0;
}
```

OUTPUT

```
Enter number of processes: 6
Enter process name:
p1
Enter arrival time, burst time, priority:
0 10 3
Enter process name:
p2
Enter arrival time, burst time, priority:
1 1 1
Enter process name:
p3
Enter arrival time, burst time, priority:
2 2 4
Enter process name:
p4
Enter arrival time, burst time, priority:
3 1 2
Enter process name:
p5
Enter arrival time, burst time, priority:
4 5 1
Enter process name:
p6
Enter arrival time, burst time, priority:
6 3 3

Gantt chart: |p1|p2|p1|p4|p5|p1|p6|p3|

Process   AT      BT      CT      TAT     WT
p1      0      10      17      17      7
p2      1      1       2       1       0
p3      2      2       22      20      18
p4      3      1       4       1       0
p5      4      5       9       5       0
p6      6      3       20      14      11
```

ROUND ROBIN

```
#include <stdio.h>

struct Process {
    char name[5];
    int at, bt, ct, tat, wt, remaining_bt;
};

void roundRobin(struct Process p[], int n, int tq) {
    int currentTime = 0, completed = 0;
    int ganttChart[100];
    int ganttIndex = 0;

    while (completed < n) {
        for (int i = 0; i < n; i++) {
            if (p[i].remaining_bt > 0) {
                ganttChart[ganttIndex++] = i;

                if (p[i].remaining_bt > tq) {
                    currentTime += tq;
                    p[i].remaining_bt -= tq;
                } else {
                    currentTime += p[i].remaining_bt;
                    p[i].remaining_bt = 0;
                    p[i].ct = currentTime;
                    p[i].tat = p[i].ct - p[i].at;
                    p[i].wt = p[i].tat - p[i].bt;
                    completed++;
                }
            }
        }
    }


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


    printf("\nGantt chart: ");
    for (int i = 0; i < ganttIndex; i++) {
        printf("|%s", p[ganttChart[i]].name);
    }
    printf("|\n");
}

int main() {
    int n, tq;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    struct Process p[n];
    for (int i = 0; i < n; i++) {
        printf("Enter process name:\n");
        scanf("%s", p[i].name);
        printf("Enter arrival time, burst time:\n");
        scanf("%d", &p[i].at);
        scanf("%d", &p[i].bt);
        p[i].remaining_bt = p[i].bt;
    }

    printf("Enter the time quantum: ");
    scanf("%d", &tq);

    roundRobin(p, n, tq);
    return 0;
}
```

OUTPUT

```
Enter number of processes: 3
Enter process name:
1
Enter arrival time, burst time:
1
5
Enter process name:
2
Enter arrival time, burst time:
2
6
Enter process name:
3
Enter arrival time, burst time:
3
8
Enter the time quantum: 2

Process  AT     BT     CT     TAT    WT
1        1      5      13     12     7
2        2      6      15     13     7
3        3      8      19     16     8

Gantt chart: |1|2|3|1|2|3|1|2|3|3|
```