Exercise 2.1
Write a shell script that prints “Shell scripting is Fun!” on the screen

```
echo "Shell scripting is fun!"
```

exercise 2.2
modify the shell script from exercise 1 to include a variable. The variable will hold the contents of the message “Shell scripting is Fun!”

```
a="Shell Scripting is Fun!"
echo $a
```

exercise 2.3
write a shell script to add two numbers and print the result. The two number should be input by the user.

```
read n1
read n2
echo $(($n1+$n2))
```

ex 2.4
write a shell script to find the given number is an odd number or even number

```
read a
if [ $a -gt 0 ]
then
   if [ $((a%2)) -eq 0 ]
   then
      echo "Even"
   else
      echo "Odd"
   fi
else
   echo "Neither even nor odd"
fi
```

ex 2.5
write a shell script to print the multiplication table of a given number up to 15 times

```
read n
i = 1
echo " $n "
echo "output"
while [ $i -le 15 ]
do
      echo "$((n*i))"
      i = $(( i + 1 ))
done
```

ex 2.6
store the output of the command “hostname” in a variable. Display “this script is running on where”- “is the output of the “hostname” command.

```
HOSTNAME = $(hostname)
echo This script is running on $HOSTNAME
```

ex 2.7
write a shell script that displays “man”, “bear”, “pig”, “dog”, “cat”, and “sheep”, on the screen with each appearing on a separate line. Try to do this in as few lines as possible

```
echo man
echo bear
echo pig
echo dog
echo cat
echo sheep
```

ex 2.8
Write a shell script to change the current directory to "/" directory by executing the command **"cd /"**. Execute **"ls"** command to list the content of the "/" directory. Lastly get a filename or directory name from the user and reports if it is a regular file, a directory, or another type of file. Also perform an ls command against the file or directory with the long listing option.

```
cd /
read filen
if [[ -f $filen ]]
then
  echo "$filen is a regular file"
  ls -l $filen
elif [[ -d $filen ]]
then
   echo "$filen is a directory"
   ls -l $filen
else
    echo "file not found"
fi
```

ex 2.9
write a shell script to change the current directory to “/bin” and display the number of files in the present working directory.

```
cd ..
cd ..
cd bin
pwd
ls -l . | egrep -c '^-'
echo "1"
```

ex 2.10
write a shell script that displays the number of files in the specified directory. The directory name should be given as input by the user.

```
cd ..
cd ..
cd bin
pwd
ls -l . | egrep -c '^-'
```

ex 2.11
Write the shell script that renames all files in the current directory that end in ".txt" to begin with today's date in the following format: YYYY-MM-DD. For example, if a file a.txt was in the current directory and today was March 17,2021 it would change name from "a.txt" to "2021–03–17-a.txt

```
DAY=$(date +%F)s
read NAME

for NAME in *.txt
 do
   echo "Renaming $NAME to ${DAY}-${NAME}"
   mv $NAME ${DAY}-${NAME}
 done
```

Exercise 3.1
Five persons are in a queue to book railway tickets. There is only one counter and all of them arrived at 9.30 AM. The counter will be open by 10.00 AM. The processing time for each person to book the tickets are 10 minutes, 20 minutes, 5 minutes, 7 minutes and 13 minutes respectively. Identify the scheduling algorithm and write a C program for the same. Calculate the waiting time of each individual persons and average waiting time for five persons. Similarly calculate the turn around time for each person and average turn around time for all.

```
#include<stdio.h>
int main(){
    int arr[5],i,j,arr1[6],sum=0;
    arr1[0]=sum;
    for(i=0;i<5;i++){
        scanf("%d",&arr[i]);
        sum=sum+arr[i];
        arr1[i+1]=sum;
    }
    float avg;
    for(j=0;j<2;j++){
        avg=0;
    for(i=j;i<j+5;i++){
            printf("%d\n",arr1[i]);
            avg=avg+arr1[i];
        }
        avg=avg/5.0;
        printf("%.1f\n",avg);
    }
}
```

Exercise 3.2
Three patients are in a Hospital to visit the doctor. The first patient is suffering from high temperature, the second patient is in unconscious state and the third patient is suffering from cold and cough. The time of each patient is 10 minutes, 30 minutes and 5 minutes respectively. Identify the scheduling algorithm and write a C program for the same.

```
#include<stdio.h>
struct sc{
        int wait;
        int pr;
        int sum;
    }sc[4];
int pri(int pr){
    int i;
    for(i=0;i<4;i++){
        if(pr==sc[i].pr){
            return sc[i].wait;
        }
    }

}
int main(){
    sc[0].pr=0;
    sc[0].wait=0;
    int i,j,sum1,k=0,z=0;
    for(i=1;i<4;i++){
        scanf("%d",&sc[i].wait);
    }
    for(i=1;i<4;i++){
        scanf("%d",&sc[i].pr);
    }
    ss:
    sum1=0;
   for(i=1;i<4;i++){
      for(j=1;j<4;j++){
          if(i==sc[j].pr){
              sc[j].sum=sum1+pri(z);
              sum1=sc[j].sum;
              z+=1;
          }
       }
    }
    float avg=0.0;
    for(i=1;i<4;i++){
       printf("%d\n",sc[i].sum);
       avg=avg+sc[i].sum;
    }
    avg=avg/3.0;
    printf("%.2f\n",avg);
    if(k==0){
       k=1;
       z=1;
       goto ss;
    }

}
```

Exercise 3.3

Group discussion is conducted for 5 students in a campus recruitment drive. The time each student had prepared is ten minutes. But the GD conductor will be giving only 3 minutes for each student. Identify the scheduling algorithm and write a C program for the same.

```
#include<stdio.h>
int main(){
   int i,a[5],n,v[5],j,temp=0;
   for(i=0;i<5;i++){
      scanf("%d",&a[i]);
   }
   scanf("%d",&n);
   for(i=0;i<3;i++){
      for(j=0;j<5;j++){
         a[i]=temp+3;
         temp=temp+3;
      }
   }
   for(i=0;i<5;i++){
      a[i]=temp+1;
      temp=temp+1;
      v[i]=a[i]-10;
   }
   int avg1=0;
   for(i=0;i<5;i++){
      printf("%d\n",v[i]);
      avg1=avg1+v[i];
   }
   printf("%d\n",avg1/5);
   avg1=0;
   for(i=0;i<5;i++){
      printf("%d\n",a[i]);
      avg1=avg1+a[i];
   }
   printf("%d\n",avg1/5);
}
```

Exercise 3.4

Six persons are in a queue to book railway tickets. There is only one counter and all of them arrived at 9.30 AM. The counter will be open by 10.00 AM. The processing time for each person to book the tickets are 10 minutes, 20 minutes, 5 minutes, 7 minutes, 2 minutes and 13 minutes respectively. The person with the minimum processing time has to be given preference. Identify the scheduling algorithm and write a C program for the same

```
#include<stdio.h>
int main(){
   int i,j,arr[6],sum1=30;
   struct pr{
      int value;
      int pri;
      int sum;
   }s[7];
   for(i=0;i<6;i++){
      scanf("%d",&s[i].value);
      arr[i]=s[i].value;
   }
   for(i=0;i<5;i++){
      for(j=0;j<5;j++){
         if(arr[j]>arr[j+1]){
            int temp;
            temp=arr[j];
            arr[j]=arr[j+1];
            arr[j+1]=temp;
         }
      }
   }
   for(i=0;i<6;i++){
      for(j=0;j<6;j++){
         if(arr[i]==s[j].value){
            s[j].pri=i;
         }
      }
   }
   int k=0,z=0;
   ss:
   if(k==1){
      sum1=32;
      z=1;}
   for(i=0;i<6;i++){
      for(j=0;j<6;j++){
            if(s[j].pri==i){
                s[j].sum=sum1;
                sum1=s[j].sum+arr[z];
                z+=1;
            }
        }
    }
    s[j].sum=sum1;
    float avg=0.0;
    for(j=0;j<6;j++){
            printf("%d\n",s[j].sum);
            avg=s[j].sum+avg;
    }
    avg=avg/6.0;
    printf("%.1f\n",avg);
    if(k==0){
        k+=1;
        goto ss;
    }
}
```

C program to copy file into another file

```
#include <stdio.h>
#include <stdlib.h> // For exit()

int main()
{
    FILE *fptr1, *fptr2;
    char filename[100], c;

    printf("Enter the filename to open for reading \n");
    scanf("%s", filename);

    // Open one file for reading
    fptr1 = fopen(filename, "r");
    if (fptr1 == NULL)
    {
        printf("Cannot open file %s \n", filename);
        exit(0);
    }

    printf("Enter the filename to open for writing \n");
    scanf("%s", filename);

    // Open another file for writing
    fptr2 = fopen(filename, "w");
    if (fptr2 == NULL)
    {
        printf("Cannot open file %s \n", filename);
        exit(0);
    }

    // Read contents from file
    c = fgetc(fptr1);
    while (c != EOF)
    {
        fputc(c, fptr2);
        c = fgetc(fptr1);
    }

    printf("\nContents copied to %s", filename);

    fclose(fptr1);
    fclose(fptr2);
    return 0;
}
```

C program in producer and consumer

/ C program for the above approach

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

// Initialize a mutex to 1
int mutex = 1;

// Number of full slots as 0
int full = 0;

// Number of empty slots as size
// of buffer
int empty = 10, x = 0;

// Function to produce an item and
// add it to the buffer
void producer()
{
    // Decrease mutex value by 1
    --mutex;

    // Increase the number of full
    // slots by 1
    ++full;

    // Decrease the number of empty
    // slots by 1
    --empty;
    // Item produced
    x++;
    printf("\nProducer produces"
           "item %d",
           x);

    // Increase mutex value by 1
    ++mutex;
}

// Function to consume an item and
// remove it from buffer
void consumer()
{
    // Decrease mutex value by 1
    --mutex;

    // Decrease the number of full
    // slots by 1
    --full;

    // Increase the number of empty
    // slots by 1
    ++empty;
    printf("\nConsumer consumes "
           "item %d",
           x);
    x--;

    // Increase mutex value by 1
    ++mutex;
}

// Driver Code
int main()
{
    int n, i;
    printf("\n1. Press 1 for Producer"
           "\n2. Press 2 for Consumer"
           "\n3. Press 3 for Exit");

// Using '#pragma omp parallel for'
// can  give wrong value due to
// synchronization issues.

// 'critical' specifies that code is
// executed by only one thread at a
// time i.e., only one thread enters
// the critical section at a given time
#pragma omp critical

    for (i = 1; i > 0; i++) {

        printf("\nEnter your choice:");
        scanf("%d", &n);

        // Switch Cases
        switch (n) {
        case 1:

            // If mutex is 1 and empty
            // is non-zero, then it is
            // possible to produce
            if ((mutex == 1)
                && (empty != 0)) {
                producer();
            }

            // Otherwise, print buffer
            // is full
            else {
                printf("Buffer is full!");
            }
            break;

        case 2:

            // If mutex is 1 and full
            // is non-zero, then it is
            // possible to consume
            if ((mutex == 1)
                && (full != 0)) {
                consumer();
            }

            // Otherwise, print Buffer
            // is empty
            else {
                printf("Buffer is empty!");
            }
            break;

        // Exit Condition
        case 3:
            exit(0);
            break;
        }
    }
}
```

Menu driven program for OS CPU scheduling

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

```
#include<stdlib.h>

typedef struct process {
char name[5];
int bt;
int at;
int prt;
int wt, ta;
int flag;
}
processes;

void bubble_sort(processes proc[], int n) {
processes t;
int i, j;
for (i = 1; i < n; i++)
for (j = 0; j < n - i; j++) {
if (proc[j].at > proc[j + 1].at) {
t = proc[j];
proc[j] = proc[j + 1];
proc[j + 1] = t;
}
}
}

int get_Processes(processes P[]) {
int i, n;
printf("\n Enter total no. of processes : ");
scanf("%d", & n);
for (i = 0; i < n; i++) {
printf("\n PROCESS [%d]", i + 1);
printf(" Enter process name : ");
scanf("%s", & P[i].name);
printf(" Enter burst time : ");
scanf("%d", & P[i].bt);
printf(" Enter arrival time : ");
scanf("%d", & P[i].at);
printf(" Enter priority : ");
scanf("%d", & P[i].prt);
}
printf("\n PROC.\tB.T.\tA.T.\tPRIORITY");
for (i = 0; i < n; i++)
printf("\n %s\t%d\t%d\t%d", P[i].name, P[i].bt,
P[i].at, P[i].prt);
return n;
}

// FCFS Algorithm
void FCFS(processes P[], int n) {
processes proc[10];
int sumw = 0, sumt = 0;
int x = 0;
float avgwt = 0.0, avgta = 0.0;
int i, j;
for (i = 0; i < n; i++)
proc[i] = P[i];

bubble_sort(proc, n);

printf("\n\n PROC.\tB.T.\tA.T.");
for (i = 0; i < n; i++)
printf("\n %s\t%d\t%d", proc[i].name, proc[i].bt,
proc[i].at);

sumw = proc[0].wt = 0;
sumt = proc[0].ta = proc[0].bt - proc[0].at;

for (i = 1; i < n; i++) {
proc[i].wt = (proc[i - 1].bt + proc[i - 1].at + proc[i -
1].wt) - proc[i].at;;
proc[i].ta = (proc[i].wt + proc[i].bt);
sumw += proc[i].wt;
sumt += proc[i].ta;
}
avgwt = (float) sumw / n;
avgta = (float) sumt / n;
printf("\n\n PROC.\tB.T.\tA.T.\tW.T\tT.A.T");
for (i = 0; i < n; i++)
printf("\n %s\t%d\t%d\t%d\t%d", proc[i].name,
proc[i].bt, proc[i].at, proc[i].wt, proc[i].ta);

printf("0\t");
for (i = 1; i <= n; i++) {
x += proc[i - 1].bt;
printf("%d ", x);
}
printf("\n\n Average waiting time = %0.2f\n
Average turn-around = %0.2f.", avgwt, avgta);
}

//Shortest Job First - Pre-emptive
void SJF_P(processes P[], int n) {
int i, t_total = 0, tcurr, b[10], min_at, j, x, min_bt;
int sumw = 0, sumt = 0;
float avgwt = 0.0, avgta = 0.0;
processes proc[10], t;

for (i = 0; i < n; i++) {
proc[i] = P[i];
t_total += P[i].bt;
}

bubble_sort(proc, n);

for (i = 0; i < n; i++)
b[i] = proc[i].bt;
```

```
i = j = 0;
avgwt = (float) sumw / n;
avgta = (float) sumt / n;
printf("\n\n Average waiting time = %0.2f\n
Average turn-around = %0.2f.", avgwt, avgta);
}

//SJF Non Pre-emptive
void SJF_NP(processes P[], int n) {
processes proc[10];
processes t;
int sumw = 0, sumt = 0;
int x = 0;
float avgwt = 0.0, avgta = 0.0;
int i, j;

for (i = 0; i < n; i++)
proc[i] = P[i];

bubble_sort(proc, n);

for (i = 2; i < n; i++)
for (j = 1; j < n - i + 1; j++) {
if (proc[j].bt > proc[j + 1].bt) {
t = proc[j];
proc[j] = proc[j + 1];
proc[j + 1] = t;
}
}

printf("\n\n PROC.\tB.T.\tA.T.");
for (i = 0; i < n; i++)
printf("\n %s\t%d\t%d", proc[i].name, proc[i].bt,
proc[i].at);

sumw = proc[0].wt = 0;
```

```
sumt = proc[0].ta = proc[0].bt - proc[0].at;

for (i = 1; i < n; i++) {
proc[i].wt = (proc[i - 1].bt + proc[i - 1].at + proc[i -
1].wt) - proc[i].at;;
proc[i].ta = (proc[i].wt + proc[i].bt);
sumw += proc[i].wt;
sumt += proc[i].ta;
}
avgwt = (float) sumw / n;
avgta = (float) sumt / n;
printf("\n\n PROC.\tB.T.\tA.T.\tW.T\tT.A.T");
for (i = 0; i < n; i++)
printf("\n %s\t%d\t%d\t%d\t%d", proc[i].name,
proc[i].bt, proc[i].at, proc[i].wt, proc[i].ta);

printf("0\t");
for (i = 1; i <= n; i++) {
x += proc[i - 1].bt;
printf("%d ", x);
}
printf("\n\n Average waiting time = %0.2f\n
Average turn-around = %0.2f.", avgwt, avgta);
}
// Priority - Preemptive
void Priority_P(processes P[], int n) {
int i, t_total = 0, tcurr, b[10], j, x, min_pr;
int sumw = 0, sumt = 0;
float avgwt = 0.0, avgta = 0.0;
processes proc[10], t;

for (i = 0; i < n; i++) {
proc[i] = P[i];
t_total += P[i].bt;
}
bubble_sort(proc, n);

for (i = 0; i < n; i++)
b[i] = proc[i].bt;

i = j = 0;
printf(" %d", tcurr);
avgwt = (float) sumw / n;
avgta = (float) sumt / n;
printf("\n\n Average waiting time = %0.2f\n
Average turn-around = %0.2f.", avgwt, avgta);
}

//Priority Non Pre-emptive
void Priority_NP(processes P[], int n) {
processes proc[10];
processes t;
int sumw = 0, sumt = 0;
float avgwt = 0.0, avgta = 0.0;
int i, j;
int x = 0;

for (i = 0; i < n; i++)
proc[i] = P[i];

bubble_sort(proc, n);

for (i = 2; i < n; i++)
for (j = 1; j < n - i + 1; j++) {
if (proc[j].prt > proc[j + 1].prt) {
t = proc[j];
proc[j] = proc[j + 1];
proc[j + 1] = t;
}
}
```

```
printf("\n\n PROC.\tB.T.\tA.T.");
for (i = 0; i < n; i++)
printf("\n %s\t%d\t%d", proc[i].name, proc[i].bt,
proc[i].at);

sumw = proc[0].wt = 0;
sumt = proc[0].ta = proc[0].bt - proc[0].at;

for (i = 1; i < n; i++) {
proc[i].wt = (proc[i - 1].bt + proc[i - 1].at + proc[i -
1].wt) - proc[i].at;;
proc[i].ta = (proc[i].wt + proc[i].bt);
sumw += proc[i].wt;
sumt += proc[i].ta;
}
avgwt = (float) sumw / n;
avgta = (float) sumt / n;
printf("\n\n PROC.\tB.T.\tA.T.\tW.T\tT.A.T");
for (i = 0; i < n; i++)
printf("\n %s\t%d\t%d\t%d\t%d", proc[i].name,
proc[i].bt, proc[i].at, proc[i].wt, proc[i].ta);

printf("0\t");
for (i = 1; i <= n; i++) {
x += proc[i - 1].bt;
printf("%d ", x);
}
printf("\n\n Average waiting time = %0.2f\n
Average turn-around = %0.2f.", avgwt, avgta);
}

//Round Robin Scheduling
void RR(processes P[], int n) {
int pflag = 0, t, tcurr = 0, k, i, Q = 0;
int sumw = 0, sumt = 0;
float avgwt = 0.0, avgta = 0.0;
processes proc1[10], proc2[10];

for (i = 0; i < n; i++)
proc1[i] = P[i];

bubble_sort(proc1, n);

for (i = 0; i < n; i++)
proc2[i] = proc1[i];

printf("\n Enter quantum time : ");
scanf("%d", & Q);

for (k = 0;; k++) {
if (k > n - 1)
k = 0;
if (proc1[k].bt > 0)
printf(" %d %s", tcurr, proc1[k].name);
t = 0;
while (t < Q && proc1[k].bt > 0) {
t++;
tcurr++;
proc1[k].bt--;
}
if (proc1[k].bt <= 0 && proc1[k].flag != 1) {
proc1[k].wt = tcurr - proc2[k].bt - proc1[k].at;
proc1[k].ta = tcurr - proc1[k].at;
pflag++;
proc1[k].flag = 1;
sumw += proc1[k].wt;
sumt += proc1[k].ta;
}
if (pflag == n)
break;
}
printf(" %d", tcurr);
```

```
avgwt = (float) sumw / n;

avgta = (float) sumt / n;

printf("\n\n Average waiting time = %0.2f\n
Average turn-around = %0.2f.", avgwt, avgta);

}

int main() {

processes P[10];

int ch, n;

do {

printf("\n\n SIMULATION OF CPU
SCHEDULING ALGORITHMS\n");

printf("\n Options:");

printf("\n 0. Enter process data.");

printf("\n 1. FCFS");

printf("\n 2. SJF (Pre-emptive)");

printf("\n 3. SJF (Non Pre-emptive)");

printf("\n 4. Priority Scheduling (Pre-emptive)");

printf("\n 5. Priority Scheduling (Non Pre-
emptive)");

printf("\n 6. Round Robin");

printf("\n 7. Exit\n Select : ");

scanf("%d", & ch);

switch (ch) {

case 0:

n = get_Processes(P);

break;

case 1:

FCFS(P, n);

break;

case 2:

SJF_P(P, n);

break;

case 3:

SJF_NP(P, n);

break;

case 4:

Priority_P(P, n);

break;

case 5:

Priority_NP(P, n);

break;

case 6:

RR(P, n);

break;

case 7:

exit(0);

}

} while (ch != 7);

return 0;

}
```

Implementation of bankers algorithm

```
// C Program to Simulate Banker's Algorithm -
CODE BY Nived Kannada
#include<stdio.h>
void main()
{   //Here we need 3 arrays namely Allocation, Max
and Available
    // arrays alloc and max are 2D arrays. array
'available' is a 1D array.
    int n, m, i, j, k, alloc[20][20], max[20][20],
available[20];
    int f[20],ans[20], ind=0, need[20][20]; //We need
the Need matrix.

    //Reading the number of Processes from the
input.
    printf("Enter number of processes: ");
    scanf("%d",&n);

    //Reading the number of Resources from the
input.
    printf("Enter the number of Resources: ");
    scanf("%d",&m);
```

```
   //Reading the Allocation Values to the Matrix
'alloc[][]'
   printf("Enter the Values of Allocation Matrix:
\n");
   for(i=0;i<n;i++)
   {
      for(j=0;j<m;j++)
      {
         printf("Enter value at position (%d%d)
:",i+1,j+1);
         scanf("%d",&alloc[i][j]);
      }
   }

   //Reading the Max values to the matrix 'max[][]'
      printf("Enter the Values of Max Matrix: \n");
   for(i=0;i<n;i++)
   {
      for(j=0;j<m;j++)
      {
         printf("Enter value at position (%d%d)
:",i+1,j+1);
         scanf("%d",&max[i][j]);
      }
   }

   //Reading the values of array available[]
   printf("Enter the values in available: \n");
   for(j=0;j<m;j++)
   {
      printf("Enter value at position (%d) :",j+1);
      scanf("%d",&available[j]);
   }

   //We are using an array f to represent the
finished status of each process.
   //Initially setting all processes as not finished. ie.
Setting f[i]=0 for each process i.
   for(k=0;k<n;k++)
   {
      f[k]=0;
   }

   //Calculating values of the NEED MATRIX using
its equation, for all processes
   //Equation is need[i][j] = max[i][j] -
allocation[i][j]
   for(i=0;i<n;i++)    //For each process
   {
      for(j=0;j<m;j++)    //For each resource
      {
         need[i][j] = max[i][j] - alloc[i][j];
      }
   }

   //Finding safe sequence
   int y=0;
// ans[] array will be used to store the SAFE
SEQUENCE in the end.
   for(k=0;k<n;k++)
   {
      for(i=0;i<n;i++)    //For each process
      {
         if(f[i]==0)
         {
            int flag = 0;   //setting flag as 0 or false.
            for(j=0;j<m;j++)    //For each Resource
            {
               if(need[i][j] > available[j])    //If Need
greater than Available, then
               {
                  flag=1; //Setting flag as true or 1.
                  //flag=1 means the Need is greater
than what is Available for that particular resource.
                  break; //Breaking out of this loop if
need > available
               }
            }

            if(flag==0)
            {
               ans[ind++] = i;
               for(y=0;y<m;y++)    //For each
Resource
               {
                  available[y] = available[y] +
alloc[i][y];  //Setting availability to current
availability + allocation
               }
               f[i]=1; //Declaring the current process
as FINISHED.
            }
         }
      }
   }

  //Displaying the SAFE SEQUENCE.
   printf("The SAFE SEQUENCE is: \n");
   for(i=0;i<n-1;i++)  //Here loop ends at n-1
because we don't want to printf the arrowmark(->)
at the end.printf(" P%d", answer[n-1)
   {
      printf(" P%d ->", ans[i]);
   }
```

```
    printf(" P%d", ans[n-1]);    //Printing the final
state in safe sequence without printing the
arrowmark.

}


/* CODE ENDS HERE */
/* Code written and explicitly commented by Nived
Kannada */
```