Part – A

P1: Write a shell program that creates a database file which consists of sid, sname, dept, display only student name, department and frequency of occurrences of students belonging to respective departments.

program p1.sh

```
clear
cat > db.txt # Create the File, after the file name enter the
             #sample Input as seen above then finally press ^d,
             #entered details are stored in the file db.txt
echo -e "Student_Name \t Department"
cut -d ',' -f 2,3 db.txt
echo -e "No. of Students Department"
cut -d ',' -f 3 db.text | sort | uniq -c
```

P2: Write a shell program which takes two file names as arguments, if their contents are the same then remove the second file.

```
# -----Sample Inputs/Files-----
#  $sh program2.sh a.txt b.txt
# -----Sample Output------
# Files are identical. b.txt is removed.
```

#program p2.sh

```
clear
#Compare Two files , contets are same result is Zero or else 1
#Error redirection
cmp $1 $2 2>>Err
if [ $? -eq 0 ]
then
    echo Files are identical. the file $2 is removed.
    rm $2
else
    echo Files are not identical. No action performed.
    echo Contents of file $1 are:
    cat $1
    echo Contents of file $2 are:
    cat $2
fi
```

p3. Write a shell program that takes a command line argument and reports on whether it is directory, a file, or something else.

```
#$p3.sh FileName or PathName
```

#Program p3.sh

```
clear
if [ -f  $1 ]
then
    echo $1 is a regular file.
elif [ -d  $1 ]
then
    echo $1 is a directory.
elif [ -L $1 ]
```

```
then
    echo $1 is a link.
elif [ -c $1 ]
then
   echo $1 is a character device.
elif [ -b $1 ]
then
    echo $1 is a block device.
else
    echo $1 is unrecognized.
fi
```

P4: Write a shell program that accepts one or more file names as arguments and converts all to uppercase provided, they exist in the current directory.

```
# $sh program6.sh a.txt b.txt
# ---Sample Output---
# Directory before:
# a.txt b.txt abc.cpp a.out
# Directory after:
# A.TXT B.TXT abc.cpp a.out
#program p4.sh
clear
echo Directory before:
ls            # List out the contents of the current directory before.
for var in $*
do
  if [ -e $var ]
   then
        new=`echo $var | tr "[a-z]" "[A-Z]"`
        mv $var $new
    else
        echo $var does not exist in the current directory.
    fi
done
echo Directory after:
ls
```

P5: Write a shell script that accepts two file names as arguments and checks if the file permissions are identical. If they are identical print the common permission else output each file name followed by its permissions.

```
# $1 - The first command line argument passed to the script.
# $2 - The second command line argument passed to the script.
#chmod used for changing the file permission  usage: chmod
user(s) operation permission(s)
#user - u user, g group, o others, a all, operation + or - ,
permission- r(read), w (write) and x (Execute)
```

```
#Program p5.sh
clear
# select and Store the permissions of the first(file) argument
p1=`ls -l $1 | cut -c 2-10`
# select Store the permissions of the second (file) argument.
p2=`ls -l $2 | cut -c 2-10`
if [ "$p1" = "$p2" ]
then
    echo File permissions of $1 and $2 are identical.
    echo $p1  # Print the common permission of both files.
else
    echo File permissions of $1 and $2 are not identical.
    echo "$1 permissions are:  $p1"
    echo "$2 permissions are:  $p2
fi
```

P6: Write a shell program to do the followings on strings, Find length of a string, wether string is NULL , and two strings are equal or not.

```
#Program P6.sh
clear
echo Enter first string:
read str1        # Read the first string.
echo Enter second string:
read str2        # Read the second string.
echo Length of string 1: ${#str1} # Print the length of str1
using ${#str1}
echo Length of string 2: ${#str2} # Print the length of str2
using ${#str2}
if [ -z "$str1" ] # Check if str1 is null.
then
    echo $str1 is NULL.
else
    echo $str1 is not NULL.
fi

if [ -z "$str2" ] # Check if str2 is null.
then
    echo $str2 is NULL.
else
    echo $str2 is not NULL.
fi

if [ "$str1" = "$str2" ] # Check if str1 is equivalent to
str2.
then
    echo $str1 and $str2 are equal.
else
    echo $str1 and $str2 are not equal.
fi
```

P7. Write a shell script to display command line arguments in reverse order

```
# $ revarg.sh A b C output should be C b A
```

```
#program p7.sh
clear
echo Given arguments are: $*
rev=" "
for var in $*      # Iterate over the command-line arguments stored in $*.
do
     rev=$rev" "$var
done
echo Arguments in reversed order are: $rev
```

P8: Write a shell program to print the given number in reverse order.

```
#Program p8.sh
clear
echo Enter the number:
read n
rev=0
nc=$n
while [ $n -gt 0 ]
do
    dig=`expr $n % 10`
    rev=`expr $rev * 10 + $dig`
    n=`expr $n / 10`
done
echo Reverse of $nc is $rev
```

P9 : Write a shell program to print the first 25 fibonacci numbers.

```
#Program p9.sh
clear
f0=0
f1=1
i=0
while [ $i -lt 23 ]
do
    f2=`expr $f0 + $f1`
    ans=$ans"  "$f0
    f0=$f1          # Move the variables as per logic.
    f1=$f2
    i=`expr $i + 1` # Increment the iterator
done
echo First 25 Fibonacci Numbers are: $ans
```

P10: Write a shell program to print the prime numbers between the specified range.

```
#Program 10
clear
echo Enter the starting number:
read start       # Read the starting number.
echo Enter the ending number:
read end         # Read the ending number.
echo Prime numbers in the range  $start  to $end are:
while [ $start -lt $end ]
do
    i=2
    flag=0
    while [ $i -lt $start ]
    do
        if [ `expr $start % $i` -eq 0 ]
        then
            flag=1
            break
        fi
        i=`expr $i + 1` # Increment the iterator.
    done
    if [ $flag -eq 0 ] # print prime number
    then
        echo $start
    fi
    start=`expr $start + 1` # Increment $start.
done
```

Q11: Write a shell program to search the given key element using linear search.

```
#Program p11.sh
clear
echo Enter the number of elements:
read n
i=0                 # Initialize the iterator to $i.
while [ $i -lt $n ] # Loop until $i < $n.
do
    echo Enter the element at `expr $i + 1`:
    read a[i]
    i=`expr $i + 1`    # Increment the iterator.
Done
# Print the contents of the array.
echo The array elements are: ${a[@]}
# Print the size of the array.
echo The number of elements in array is ${#a[@]}
echo Enter the key to search for:
read key
i=0
flag=1
```

```
while [ $i -lt $n ]
do
    if [ "${a[$i]}" = "$key" ]
    then    #this is done to accommodate for non-numeric arrays
        flag=0
        break
    fi
    i=`expr $i + 1`
done
if [ $flag -eq 0 ]
then
      echo $key Found at index $i.
else
      echo Could not find $key in the array.
fi
```

P12 Write a shell program to find the largest of three numbers using a function.

```
#program p12.sh
clear
max3(){      # Define a function named max3.
   max=$1    # Set max variable to the first parameter ($1).
   if [ $1 -lt $2 ]
   then
        max=$2
   fi
    if [ $max -lt $3 ]
    then
        max=$3
    fi
    return $max
}
# Call the max3 function just like a command
#using 1, 2 and 3 as parameters.
max3 1 2 3
# Print the result of the max3 function.
echo Max of 1, 2 and 3 is $?.
echo enter 3 numbers
read n1
read n2
read n3
max3 $n1 $n2 $n3   # Call the max3 function just like a
command using 1, 2 and 3 as parameters.
echo Max of $n1, $n2 and $n3 is $?
```

Or

```
max3(){

if [ $1 -gt $2 -a $1 -gt $3 ]
then
    return $1
elif [ $2 -gt $1 -a $2 -gt $3 ]
then
    return $2
else
    return $3
fi
}
echo "Enter three Integers:"
read a b c
large $a $b $c
echo " Largest of $a $b and $c is $? "
```

Part – B

1. **Write a C/C++ program to implement UNIX commands ln, mv, rm commands using APIs.**

The UNIX ln command is implemented using the link API..

```
#include <iostream>
#include <stdio.h>
#include <unistd.h>
using namespace std;

int main ( int argc, char* argv[] )
{
  if (argc != 3)
  {
    cout<<"usage: ./a.out Source_FileName Link_FileName\n”;
    exit(0);
  }

   if (( link ( argv[1], argv[2 ] )) == -1 )
            perror( "link error" );
   else
       cout<<“Link has been created with the name”<<argv[2]);
return 0;
}
```

rm command to delete all the files on the command line using ulink API..

```
 int main(int argc, char *argv[])

{

      int i;
     if (argc <=1)    {
        cout<<"No files to delete”
```

```
             <<”usage: ./a.out FileName1 FileName2 …..\n”;
         exit(1);
       }

     for(i=1;i<argc;i++)
     {
            if(unlink(argv[i])==-1)
                 perror("Error in deleting...\n");
           else
                cout<<”the file ”<<argv[i]<<” is deleted...\n";
     }
 return 0;
}
```

The UNIX mv command can be implemented using link and unlink APIs is shown below.

```
#include<stdio.h>
#include<unistd.h>
#include <iostream>
using namespace std;
int main(int  argc, char *argv[])
{
  if (argc != 3)
  {
    cout<<"usage: ./a.out Source_FileName Link_FileName\n”;
    exit(1);
}


     if((link(argv[1],argv[2]))==-1)
        perror("Link eror\n");
     else
          unlink(argv[1]); //deletes first file
     return 0;
}
```

2. **Write a program in C/C++ to display the contents of a named file on standard output device., Also Write a program to copy the contents of one file to another.**

```
#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>
#include<sys/stat.h>
#include <iostream>
using namespace std;

int main(int argc, char *argv[])
{
      int n,fd1, fd2;
     char buf[10];
```

```
      if (argc != 3)
      {
      cout<<"usage: ./a.out Source_FileName Destination_FileName\n";
      exit(1);
      }
      if((fd1= open(argv[1], O_RDONLY))==-1)
      {
      cout<<"Can't open file" <<argv[1]<<" for Reading\n";
      exit(0);
      }

      if((fd2= open(argv[2],O_WRONLY | O_CREAT | O_TRUNC ,744))==-1)
      {
      cout<<"Can't open file "<<argv[2]<<" for Writing\n";
      exit(0);
      }
      while((n = read(fd1, buf, 1))>0)
      {
      write(1, buf,n);   // write to terminal
      write(fd2, buf,n); // write to file
      }
      close (fd1);
      close (fd2);
    return 0;
  }
```

3. **Write a C program that reads every 100th byte from the file, where the file name is given as command -line argument**

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <iostream>
using namespace std;

int main(int argc,char *argv[])
{
 int fd,n,skval;
 char c;
 if(argc!=2)
 {
   cout<<"usage: ./a.out FileName\n";
   exit(1);
  }
  if((fd= open(argv[1], O_RDONLY))==-1)
  {
   cout<<"Can't open file "<<argv[1]<<" for Reading\n";
   exit(0);
   }
while((n=read(fd,&c,1))==1)
{
 cout<<"\nChar: "<<c;
 skval=lseek(fd,99,1);
 cout<<"\nNew seek value is:"<<skval;
}
cout<<endl;
exit(0);
}
```

4. **Write a C program to display information of a given file which determines type of file and Inode information, where the file name is given as command line argument.**

```
#include<stdio.h>
#include<stdlib.h>
#include <unistd.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<time.h>
#include <iostream>
using namespace std;

int main(int argc, char *argv[])
{
 struct stat sb;
 if (argc != 2)
 {
  cout<<"Usage: ./a.out  FileName \n";
  exit(0);
 }
if (stat(argv[1], &sb) == -1)
 {
   perror("stat");
   exit(0);
 }
 cout<<"File: "<<argv[1]<<" is ";
 switch (sb.st_mode& S_IFMT)
 {
  case S_IFBLK: cout<<"Block device file \n";
                break;
  case S_IFCHR: cout<<"Character device file\n";
                break;
  case S_IFDIR: cout<<"Directory file\n";
                break;
  case S_IFIFO: cout<<"FIFO/pipe file\n";
                break;
  case S_IFLNK: cout<<"Symbolic link file \n";
                break;
  case S_IFREG: cout<<"Regular file \n";
                break;
  default:     cout<<"Unknown file? \n";
 }
cout<<"and its Inode number is : "<< sb.st_ino<<endl;
return 0;
}
```

5. Write a C program to create a process by using fork () and vfork() system call

```
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
using namespace std;

int main( )
{
    pid_t     pid;
    cout<<"\n Original Process:”<<getpid()
         <<”\tParent Process:”<<getppid()<<endl;
    if ((pid = fork()) ==-1)
      {
            perror("Fork Error");
            exit(0);
      }
    if (pid == 0) {                      /* child */
        cout<<"\n Child Process:”<<getpid()
            <<”\tParent Process:”<<getppid()<<endl;
      }
    if (pid > 0) {                      /* parent */
            cout<<"\n Original Process:”<<getpid()
                 <<”\tParent Process:”<<getppid()<<endl;
          }
    cout<<”\n end of Main\n”;
 return 0;
}
```

Using Vfork

```
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
using namespace std;

int main( )
{
    pid_t     pid;
    cout<<"\n Original Process:”<<getpid()
         <<”\tParent Process:”<<getppid()<<endl;
    if ((pid = vfork()) ==-1)
      {
            perror("Fork Error");
            exit(0);
      }
    if (pid == 0) {                      /* child */
        cout<<"\n Child Process:”<<getpid()
            <<”\tParent Process:”<<getppid()<<endl;
      }
    if (pid > 0) {                      /* parent */
            cout<<"\n Original Process:”<<getpid()
                 <<”\tParent Process:”<<getppid()<<endl;
            }
     cout<<”\n end of Main\n”;
     return 0;
    }
```

6. Write a program to demonstrate the process is Zombie, and to avoid Zombie process.
Write a program that creates a zombie and then call system to execute the PS Command to verify that the process is Zombie.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <iostream>
using namespace std;

int    main(void)
{
    pid_t    pid;
    if ((pid = fork()) ==-1)
    {
    perror("Fork Error");
    exit(0);
     }
     if (pid == 0)                /* child */
     {
    cout<<"\nChild :"<<getpid()
          <<" Waiting for parent to retrieve its exit status\n";
          exit(0);
    }                  /* parent */
    if (pid > 0)
    {
          sleep(5);
          cout<<"\nParent:"<<getpid();
           system("ps u");
    }
    return(0);
}
```

Program to avoid zombie using wait and then call system to execute the PS Command to verify that the process is Zombie or not.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <iostream>
using namespace std;

int    main()
{
    pid_t    pid;
    int status;
    if ((pid = fork()) ==-1)
    {
           perror("Fork Error");
      return 0;
      }
     if (pid == 0){         /* child */
          cout<<"\nChild :"<<getpid()
           <<" Waiting for parent to retrieve its exit status\n";
           exit(0);
     }
     if (pid  > 0)          /* parent */
     {
          wait(&status);
         cout<<"\nParent:"<<getpid();
          system("ps u");
      }
   return(0);
}
```

7. Write a C program to create an Orphan Process.

```
include<stdio.h>
#include<stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <iostream>
using namespace std;
int main()
{
    pid_t pid;
    if ((pid = fork()) ==-1)
    {
      perror("Fork Error");
     exit(0);
      }
    if (pid == 0)                /* child */
    {
      sleep(5);
      cout<<"\n Child :"<<getpid()
        <<"\tOrphan's Parent :"<<getppid()<<endl;
     }
    if (pid > 0)                 /* parent */
     {
        cout<<"\n Original Parent:"<<getpid()<<endl;
        exit(0);
     }
   return 0;
}
```

8. Write a C program to demonstrate a parent process that uses wait ( ) system call to catch the child 's exit code.

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <iostream>
using namespace std;
void pr_exit(int status)
{
   if (WIFEXITED(status))
     cout<<"\n Normal termination, exit status ="
          << WEXITSTATUS(status);
   else if (WIFSIGNALED(status))
     cout <<"\nAbnormal termination, signal number = "
          <<WTERMSIG(status);
  else if (WIFSTOPPED(status))
      cout<<"\nChild stopped, signal number = "
           <<WSTOPSIG(status);
}
int main()
{
   pid_t pid, childpid;
   int status;
```

```
      if ((pid = fork()) ==-1){
        perror("\nFork Error");
        return 0;
     }
      if(pid==0)
        exit(23);
        childpid=wait(&status);
        pr_exit(status);

     if ((pid = fork()) ==-1){
        perror("\nFork Error");
        return 0;
     }
      if(pid==0)
        abort();
      childpid=wait(&status);
      pr_exit(status);

       if ((pid = fork()) ==-1){
        perror("\nFork Error");
        return 0;
       }
      if(pid==0)
         int res=5/0;
      childpid=wait(&status);
      pr_exit(status);
return 0;
}
```

9. Write a Program to demonstrate race condition.

```
 #include <unistd.h>
 #include <sys/types.h>
 #include <stdio.h>
 #include <stdlib.h>
 #include <iostream>
 using namespace std;

void charatatime(char *str)
{
  char *ptr;
  int c;
  setbuf(stdout, NULL); /* set unbuffered */
  for (ptr = str; (c = *ptr++) != 0; )
     putc(c, stdout);
}
int main(int argc , char *argv[])
{
    pid_t    pid;
    if ((pid = fork()) ==-1)
    {
          perror("Fork Error");
          return 0;
     }
    if (pid == 0)
     charatatime("CHILD PROCESS WRITING DATA TO THE TERMINAL\n");
    if (pid > 0)
     charatatime("parent process writing the data to terminal\n");
    return 0;
}
```

10. Write a program to implement UNIX system (), using APIs.

```
#include <sys/types.h>
#include <sys/wait.h>
#include <errno.h>
#include <unistd.h>
#include <stdlib.h>
#include <iostream>
using namespace std;

int status;
int system(const char *cmd)
{
    pid_t pid;

    if (cmd == NULL)
          return(1); /* always a command processor with Unix */

    if ( (pid = fork()) < 0) {
            status = -1;           /* probably out of processes */
            }
        else if (pid == 0) {                              /* child */
            execl("/bin/sh", "sh", "-c", cmd ,(char *) 0);
            _exit(127);                      /* execl error */
           }
        else {                               /* parent */
           while (waitpid(pid, &status, 0) < 0)
             if (errno != EINTR)
             {
               status = -1; /* error other than EINTR from waitpid() */
               break;
              }
       }
       return(status);
   }

   int  main(void)
   {
       if ( (status = system("date; exit 0")) < 0)
            perror("system error");

       if ( (status = system("daaate")) < 0)
            perror("system error");

       if ( (status = system("who; exit 44")) < 0)
            perror("system error");

       exit(0);
   }
```

11. Write a C/C++ program to catch, ignore and accept the signal, SIGINT.

//Catch the signal

```
#include <signal.h>
#include <iostream.h>
#include <unistd.h>
#include <stdlib.h>
#include <iostream>
using namespace std;

void handler(int signo)
{
  cout<<"\nsignal handlar : catched signal num is =>"
      <<signo<<endl;
  exit(0);
}
int main()
{
    signal(SIGINT,handler) ;
    while(1)
        cout<<"hello \t  ";
  }
```

2. **Taking default action**

```
#include <signal.h>
#include <unistd.h>
#include <iostream>
using namespace std;

int main()
{
    signal(SIGINT, SIG_DFL) ;
    while(1)
      cout<<"hello\t";
  }
```

3. **Ignore the Signal**

```
#include <signal.h>
#include <iostream.h>
#include <unistd.h>

int main()
{
    signal(SIGINT,SIG_IGN) ;
    while(1)
    cout<<"hello\t  ";
  }
```

12. Write a program to create, writes to, and reads from a pipe. Also Write a program to create a pipe from the parent to child and send data down the pipe.

Program creates, writes to, and reads from a pipe.

```
#include <stdio.h>
#include <stdlib.h>
 #include <errno.h>
#include <unistd.h>
#include <iostream>
using namespace std;

    int main()
    {
    int pfds[2];
    char buf[30];
    if (pipe(pfds) == -1)
    {
         perror("pipe");
         exit(1);
    }
    cout<<"writing to file descriptor #"<<pfds[1]<<endl;
    write(pfds[1], "test", 5);
```

```
     cout<<"reading from file descriptor #"<<pfds[0]<<endl;
     read(pfds[0], buf, 5);
     cout<<"read : "<<buf<<endl;
     return 0;
 }
```

Program to create a pipe from the parent to child and send data down the pipe.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <iostream>
using namespace std;

     int main()
     {
     int  n , pfds[2];
     char  buf[30];
     pid_t  pid;

     if (pipe(pfds) == -1)
     {
          perror("pipe");
          exit(1);
     }

     if ((pid = fork()) ==-1)
     {
          perror("Fork Errror");
          return 0;
      }
     if (pid>0)    //parent
     {
          close(pfds[0]);
          write(pfds[1], "hello\n",7);
     }
     if(pid==0)             //child
     {
       close(pfds[1]);
       n=read(pfds[0],buf, 7);
       write(1, buf,n);
     }
  return 0;
}
```