UNIX PROGRAMMING LAB MANUAL

LAB CODE: 14ITL503/D

DEPARTMENT OF INFORMATION TECHNNOLOGY

BAPTLA ENGINEERING COLLEGE::BAPATLA

(AUTONOMOUS)

LAB COURSE DESCRIPTION

1.Course code: 14ITL503/D

2. Course Title: UNIX Programming lab

3. Core [ ] Elective [√]

4 .pre-requisites (if any): C Programming

5. Semester / year in which offered: 2019-2020 **III year I Sem**

6. No. of weeks of Instruction: 39 (sec 'A'), 39 (sec 'B')

7. No. of hours per week: 3

8. Evaluation procedure

INDEX	DESCRIPTION EVALUATION	TOTAL MARKS
A	Marks allotted for day-to-day lab work	20 M
B	Marks allotted for record	5 M
C	Marks Awarded for Lab Exam	15 M
D	CIE marks(A+B+C)	40 M
E	Semester End Examination Marks	60 M

LABCYCLE 1

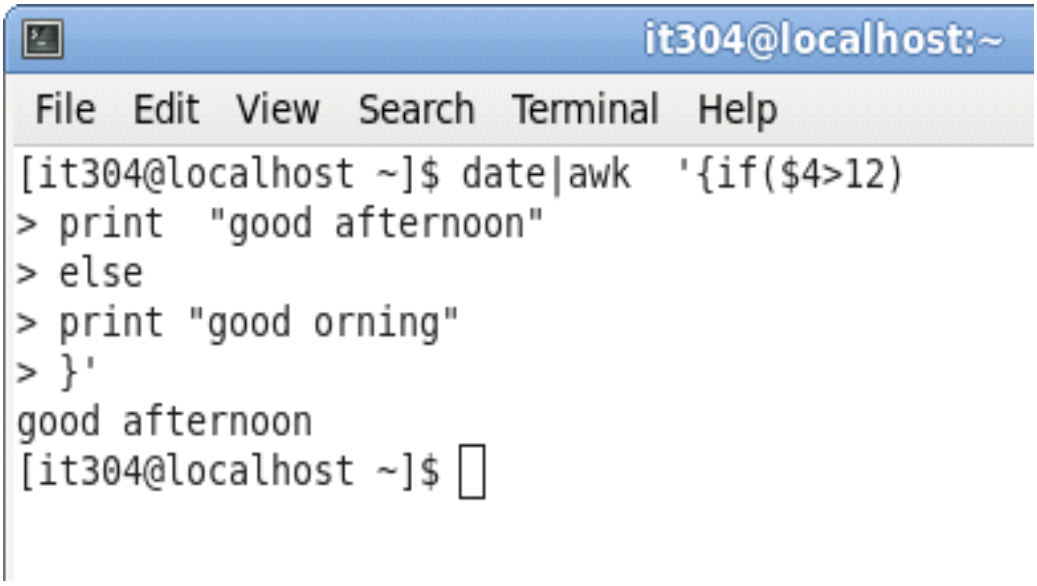
AWK Programming

1) Design a command "WISHME" that will greet you "goodmorning","good afternoon" according to current time.

Source code:

```
[it304@localhost ~]$ date|awk  '{
if($4>12)
print  "good afternoon"
else
print "good morning"
  }'
```

output:

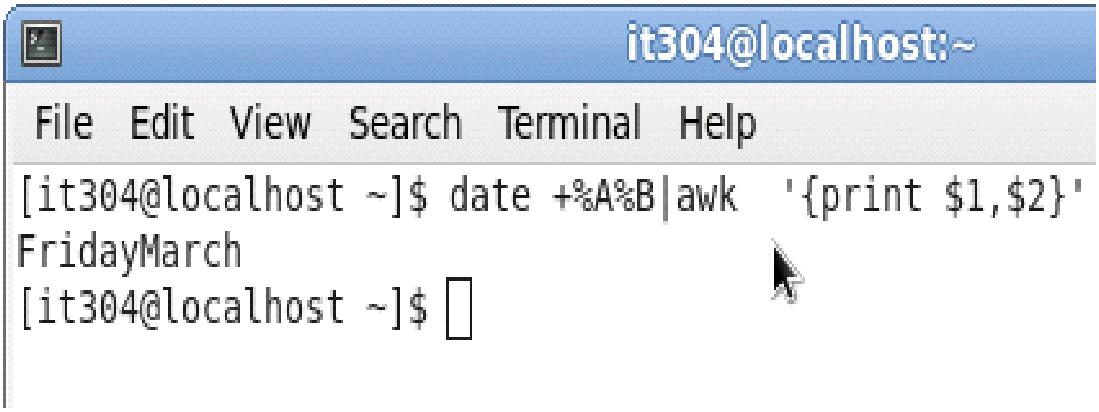


2) Design a command "VERBOSEDATE" that displays day and month completely spelled.

Source code:

```
[it304@localhost ~]$ date +%A%B|awk  '{print $1,$2}'
```

output:



3) Design a command ”fages” that will list the files and their ages to date

Source code:

```
ls -l | awk ' BEGIN{
no ["Jan"]=1
no["Feb"]=2
no["Mar"]=3
no["Apr"]=4
no["May"]=5
no["Jun"]=6
no["Jul"]=7
no["Aug"]=8
no["Sep"]=9
no["Oct"]=10
no["Nov"]=11
no["Dec"]=12
split(","'date'",",age) }
{
m=no[age[2]]-no[$6]
d=age[3]-$7
if(d<0)
{ d=d+30;
m--;
 }
printf("%s ->mon %d:%d\n",$9,m,d) }'
```

output:

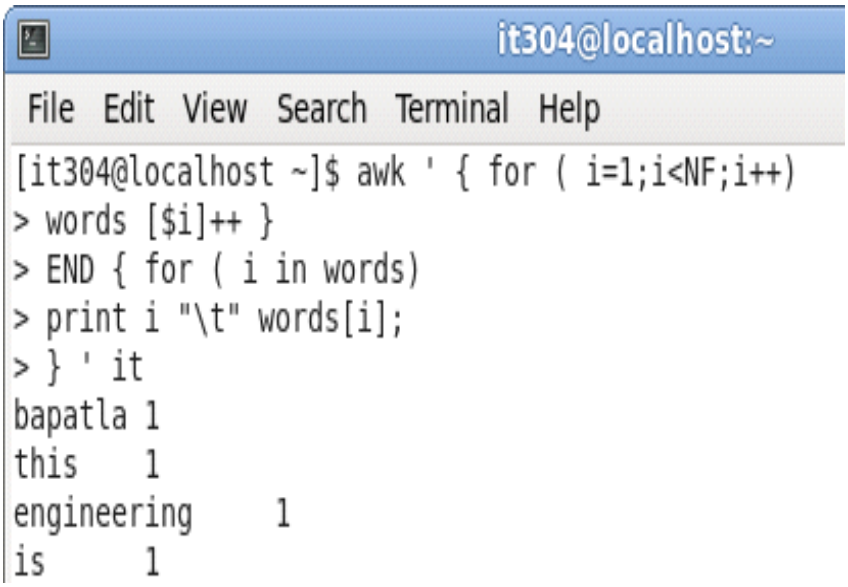
```
b~ ->mon -2:27
dlist ->mon -2:25
elist ->mon -2:25
emp2 ->mon -2:25
employee.lst ->mon -13:1
emp.lst ->mon -13:1
f1 ->mon -13:8
f2 ->mon -13:8
f3 ->mon -13:1
file ->mon -2:11
file1 ->mon -2:25
fileages ->mon -3:3
float ->mon -3:24
foo ->mon -2:11
gnome-terminal.desktop ->mon -13:8
include ->mon -2:11
instr.file ->mon -2:25
mlist ->mon -2:25
note.new~ ->mon -12:1
olist ->mon -2:11
path.sh ->mon -4:3
priya ->mon -13:8
remainder ->mon -3:3
Remainder ->mon -3:3
sales ->mon -3:3
UNIXCommands.doc ->mon -13:1
Unix_Lab_Manual_Uni14-15.doc ->mon -4:3
which.sh~ ->mon -2:6
```

4) Design a command “world freq" that will print the words and number of occurrences of that word in the given text.

Source code:

```
[it304@localhost ~]$ awk ' { for ( i=1;i<NF;i++)
words [$i]++ }
   END { for ( i in words)
printi "\t" words[i];
   } ' it
```

output:



5) Design a command: remainders" that will print the events happing today, where events and their dates are edited in the file "events"

Source code:

```
[it304@localhost ~]$ cat>remainder
todaynac visit
todayunix lab

[it304@localhost ~]$ awk  '{ split(","'"date"'",",d)
if(($2==d[2]) && ($3==d[3]))
print $0
} ' $remainder
```

output:

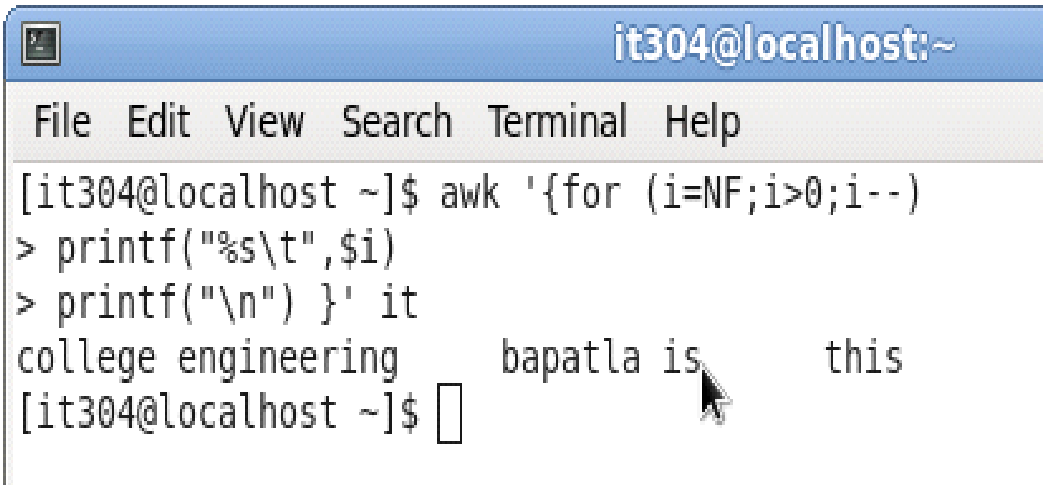
today nac visit
today unix lab

6) Design a command "backwords"that will prints the line and reverse order.

Source code:

```
[it304@localhost ~]$ awk '{for (i=NF;i>0;i--)
printf("%s\t",$i)
printf("\n") }' it
```

output:

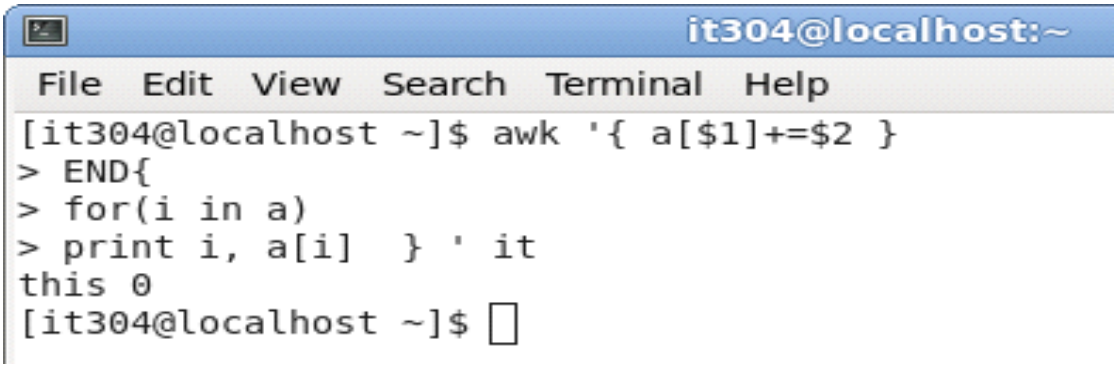


7) Design a command "sales_totals"that will consolidate the sales made by sales persons, from the file sales where each line contains the name of sales person and sales made.

Source code:

```
[it304@localhost ~]$ awk '{ a[$1]+=$2 }
END{
for(i in a)
printi, a[i]  } ' it
```

output:

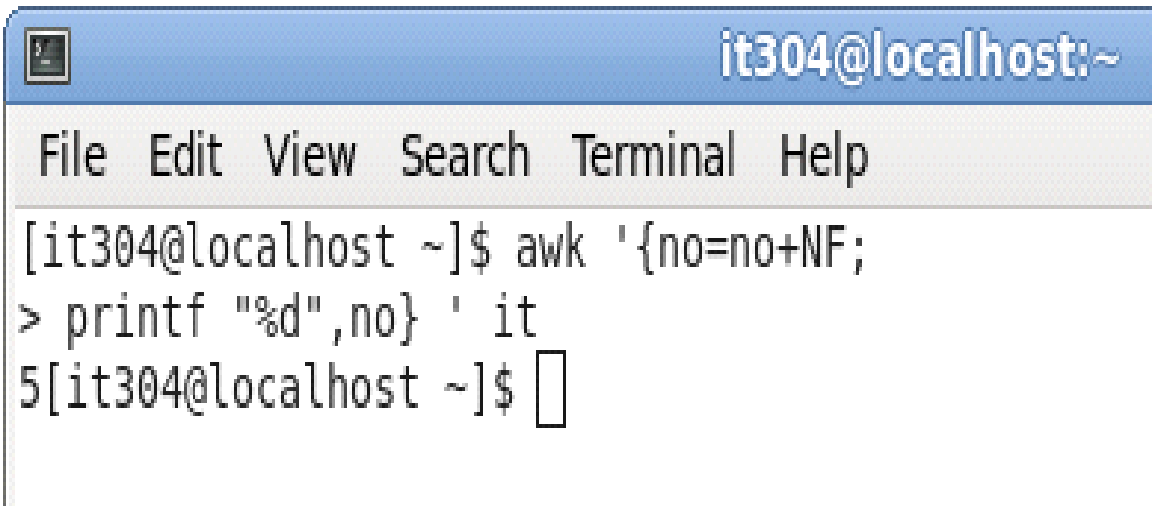


8) Design a command "wcount" that will count the number of words in a file.

Source code:

```
[it304@localhost ~]$ awk '{no=no+NF;
printf "%d", no} ' it
```

output:

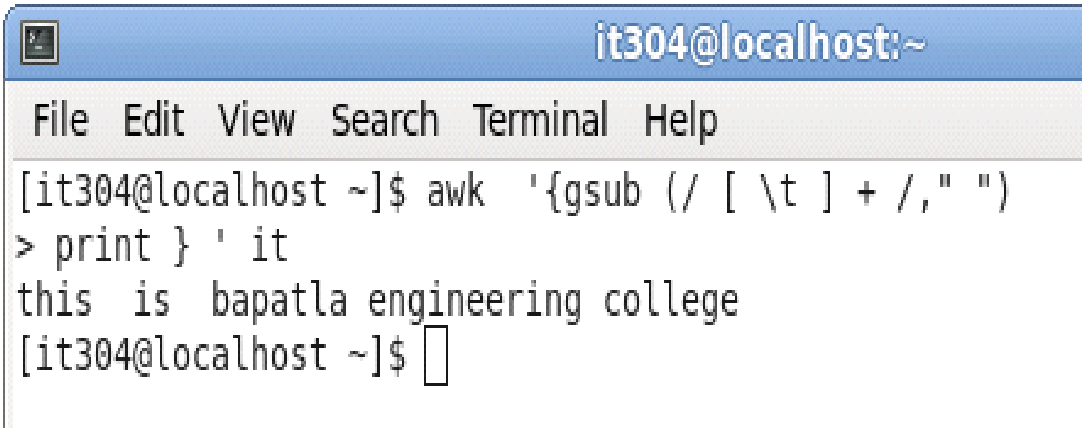


9) Design a command "squeeze" that will convert tabs or more than one blank space to one blank one blank space.

Source code:

```
[it304@localhost ~]$ awk  '{gsub (/ [ \t ] + /," ")
print } ' it
```

output:

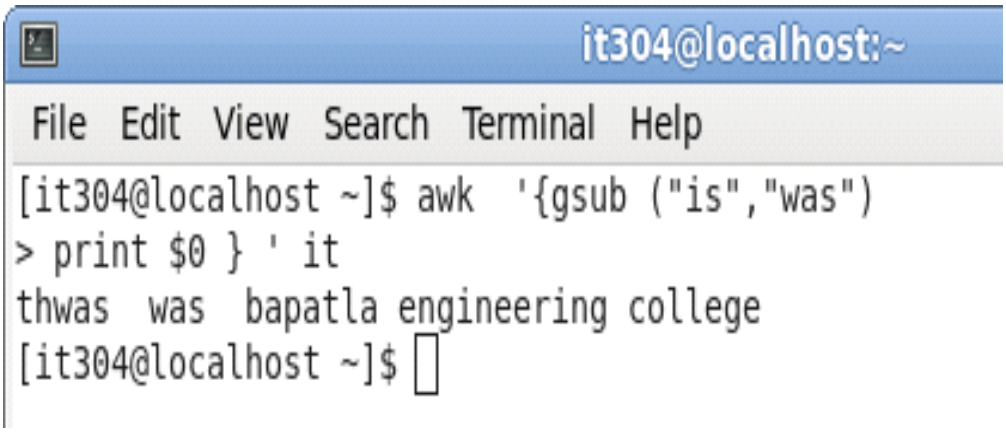


10) Design a command "replace over" that will replace the variable with the specified variable in a file.

Source code:

```
it304@localhost ~]$awk '{gsub ("is","was")
print $0 } ' it
```

output:



LABCYCLE 2

Shell scripts and programming

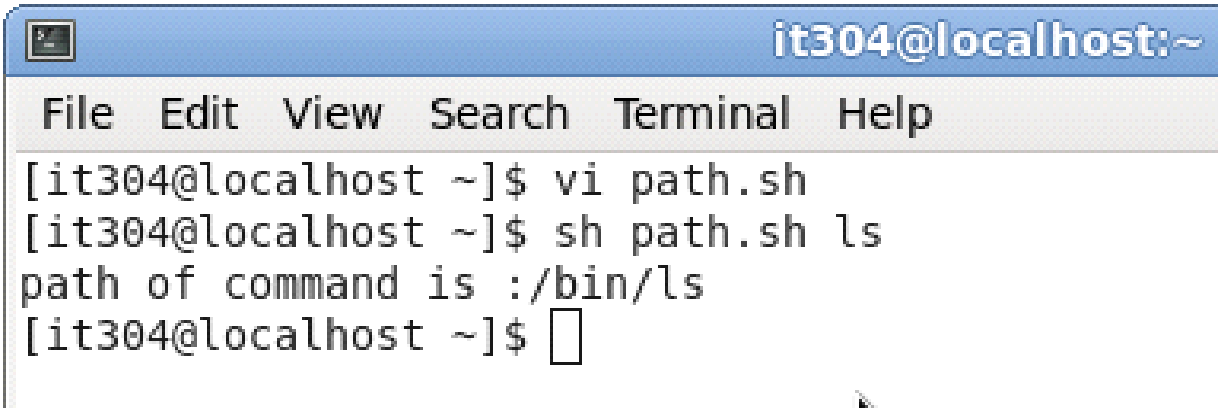
1) Design a command which, that prints the path of the command (file) given as argument.

Source code:

```
[it304@localhost ~]$ vi path.sh
c=0
for i in `echo $PATH|sed 's/:/ /g'`
do
if [ -f  $i/$1 ]
then
 c=`expr $c + 1`
echo "path of command is :$i/$1"
fi
done
if [ $c -eq 0 ]
then
echo "file not found"
fi
```

Execution: [it304@localhost ~]$ sh path.sh ls

output:



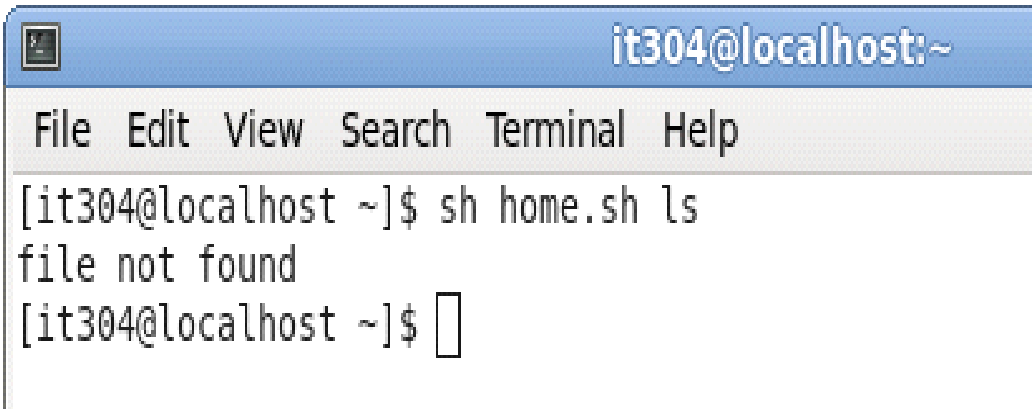
2) Design a command search that prints the path of the given as argument located in your home directory.

Source code:

```
[it304@localhost ~]$ vi home.sh
c=0
for i in `echo $HOME|sed 's/:/ /g'`
do
if [ -f  $i/$1 ]
then
c=`expr $c + 1`
echo "path of command is :$i/$1"
fi
done
if [ $c -eq 0 ]
then
echo "file not found"
fi
```

execution:[it304@localhost ~]$sh home.sh ls

output: file not found



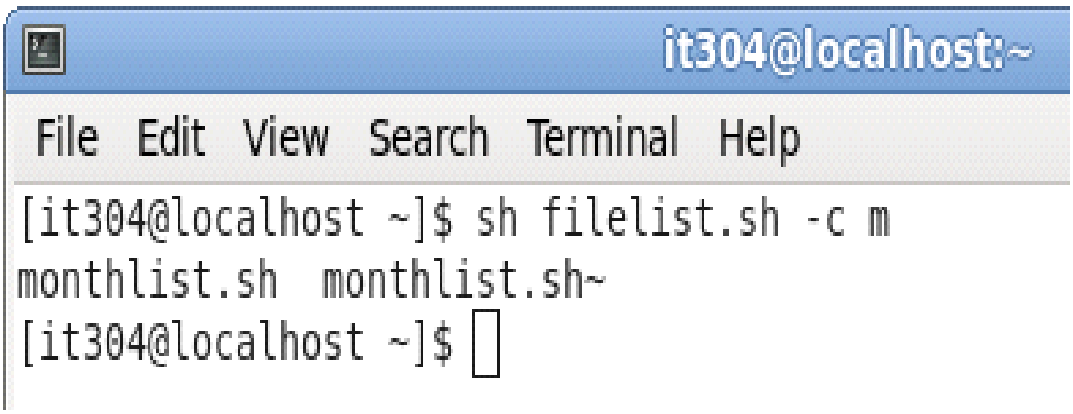
3) Design a command file list[-c <char>] which prints all filenames beginning with the character specified as argument to the command, if the option is not specified it should print all the file names.

Source code:

```
[it304@localhost ~]$ vi filelist.sh
if [ $# -ne 0 ]
then
if [ $1 == "-c" ]
then
ls $2*
else
echo "pass any valid argument"
fi
else
ls
fi
```

execution:[it304@localhost ~]$sh filelist.sh -c m

output:



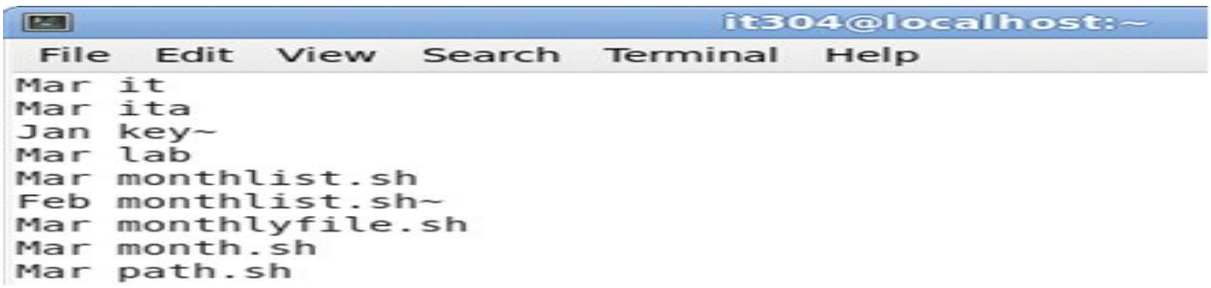
4)Design a command monthly file[-m <month>] which lists the files created in a given month where month is argument to be command if the option is not specified, it lists the files in all the months.

Source code:

```
 [it304@localhost ~]$ vi month.sh
if  [ $# -eq 0 ]
then
ls -l|awk  '{ print $6,$9 }'
elif  [ $# -eq 2 -a $1 == '-m'  ]
then
ls -l|awk  -v sm=$2 '{ if( $6 == sm )  print $6,$9 }'
else
echo "pass valid month name"
fi
fi
```

execution:[it304@localhost ~]$sh monthlyfile.sh

output:



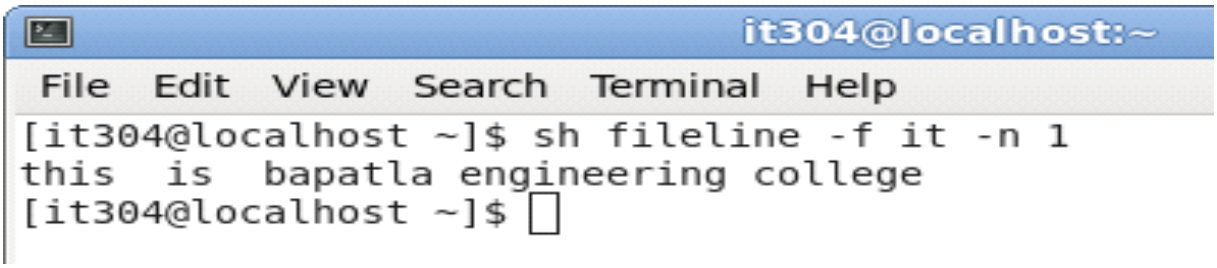
5) Design a command get line [-f <filename> -n <line no>] which prints the line number line no in the file specified with -f option .if the file specified with -f option. If the line no. is not specified it should list all the lines in the given file.

Source code:

```
  [it304@localhost ~]$ vi fileline
if [ $# -eq 4 -a  $1== "-f" -a $3=="-n" ]
then
sed -n $4p $2
elif [ $# -eq 2 -a $1== "-f" ]
then
cat $2
elif [ $# -eq 4 -a $1 == "-n" -a $3 == "-f" ]
then
sed -n $2 p $4
else
echo "Syntax Error"
fi
```

execution:[it304@localhost ~]$sh fileline -f it -n 1

output:



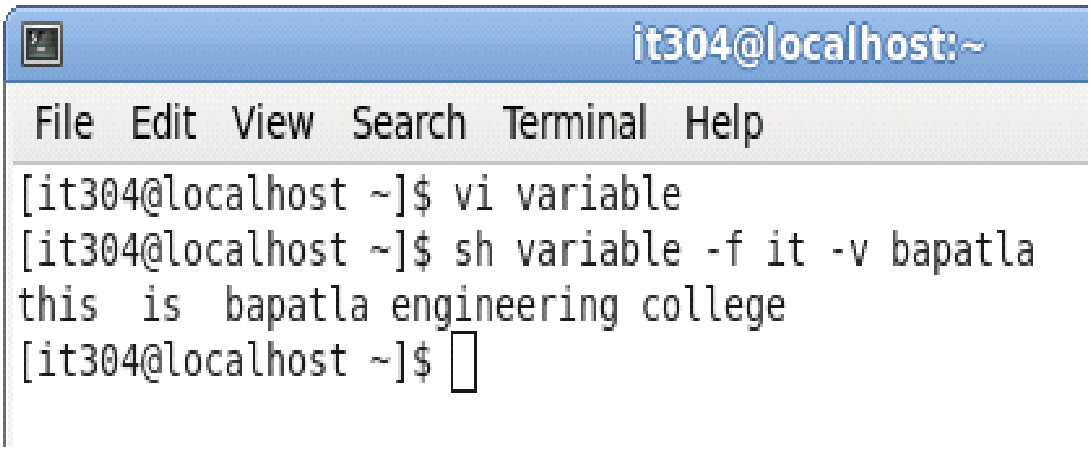
6) Design a command list lines [-f <filename> -v <variable>] which prints the line from the given file filaname,whichcontaing the variable varname.if variable is not specified it should list all the lines.

Source code:

```
[it304@localhost ~]$ vi variable
if [ $# -eq 4 -a $1 == "-f" -a $3 == "-v" ]
then grep $4 $2
elif [ $# -eq 2 -a $1 == "-f" ]
then      cat $2
elif [ $# -eq 4 -a $1 == "-v" -a $3 == "-f" ]
then grep $2 $4
else
echo "Syntax Error"
fi
```

execution:[it304@localhost ~]$sh variable -f it -v bapatla

output:



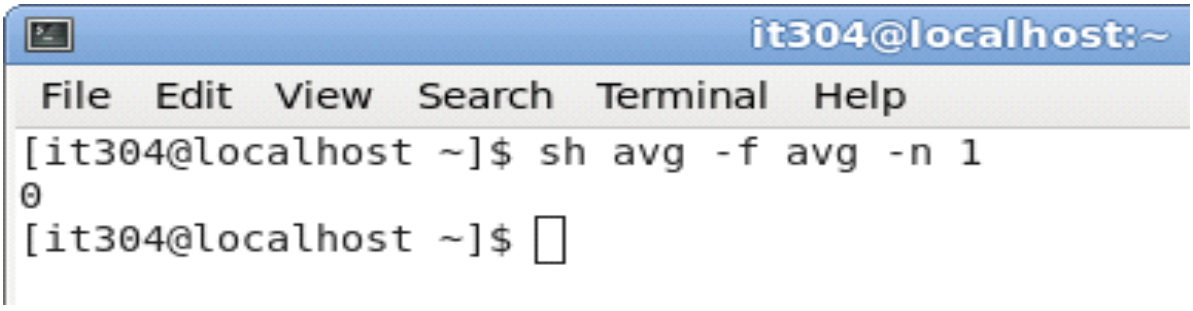
7) Design a command avg [-n <colon>-f<filename>] which prints the average of the given column in a file where colon and filename are arguments to the command.

Source code:

```
[it304@localhost ~]$ vi avg
if [ $# -eq 4 -a $1 == "-f" -a $3 == "-n" ]
then
cat $2 | awk -v c=$4 '{ s+=$c
    }
     END { print s/NR }'
elif [ $# -eq 4 -a $1 == "-n" -a $3 == "-f" ]
then
cat $4 | awk -v c=$2 '{ s+=$c
     }
     END { print s/NR }'
Els
echo "Syntax Error"
fi
```

execution:[it304@localhost ~]$sh avg -f avg -n 1

output:



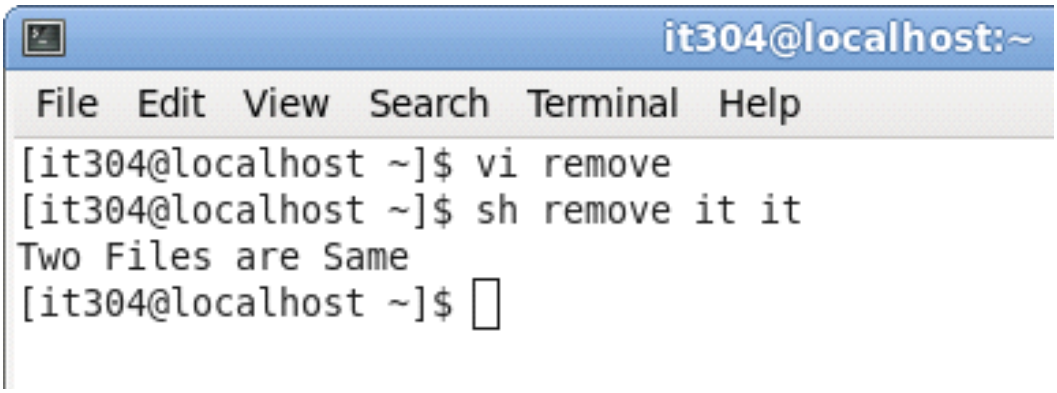
8) Program which takes two file names as arguments, if their contents are the same then remove the second file.

Source code:

```
[it304@localhost ~]$ vi remove
if [ $1 == $2 ]
then
echo "Two Files are Same"
else
     d=`diff $1 $2`
if [ -z $d ]
then
rm $2
echo "Second file Removed"
else
echo "The Contents of the files are not same"
fi
fi
```

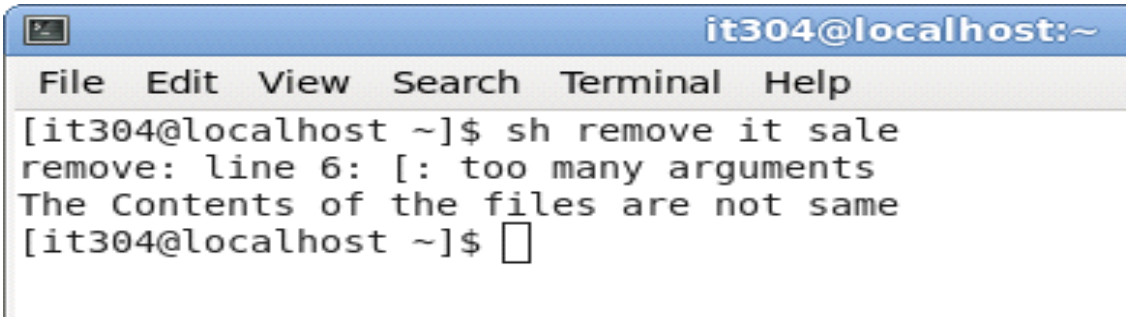
execution:[it304@localhost ~]$sh remove it it

output:



output:

[it304@localhost ~]$ sh remove it sale



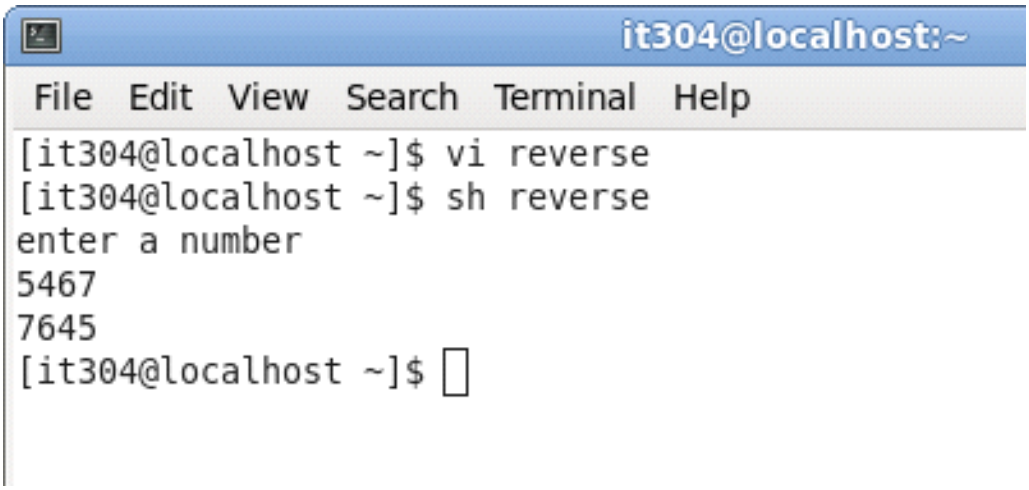
9) To print the given number in reversed order.

Source code:

```
[it304@localhost ~]$ vi reverse
#!/bin/sh
echo enter a number
c=0
read num
while [ $num -gt 0 ]
do
r=`expr $num % 10`
c=`expr  $c \* 10  + $r`
num=`expr $num / 10`
done
echo $c
```

execution:[it304@localhost ~]$sh reverse

output:



10) To print first 25 Fibonacci numbers.

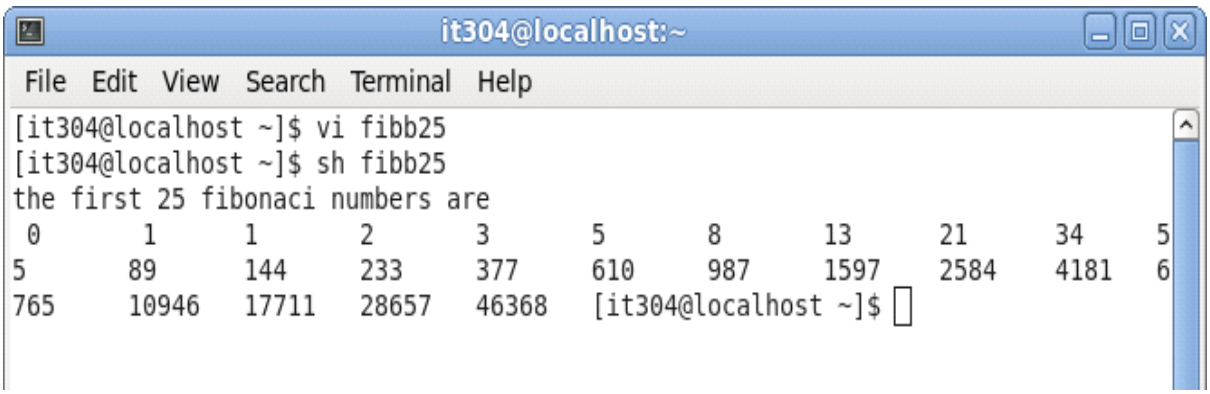
Source code:

```
[it304@localhost ~]$ vi fibb25
#!/bin/sh
echo the first 25 fibonaci numbers are
num1=0
num2=1
printf " $num1 \t $num2 \t"
counter=3
while [ $counter -le 25 ]
do
num3=`expr $num1 + $num2`
printf "$num3 \t"
```

```
num1=$num2
num2=$num3
counter=`expr $counter + 1`
done
```

Execution: [it304@localhost ~]$ sh fibb25

output:



11) To print the prime numbers between the specified range.

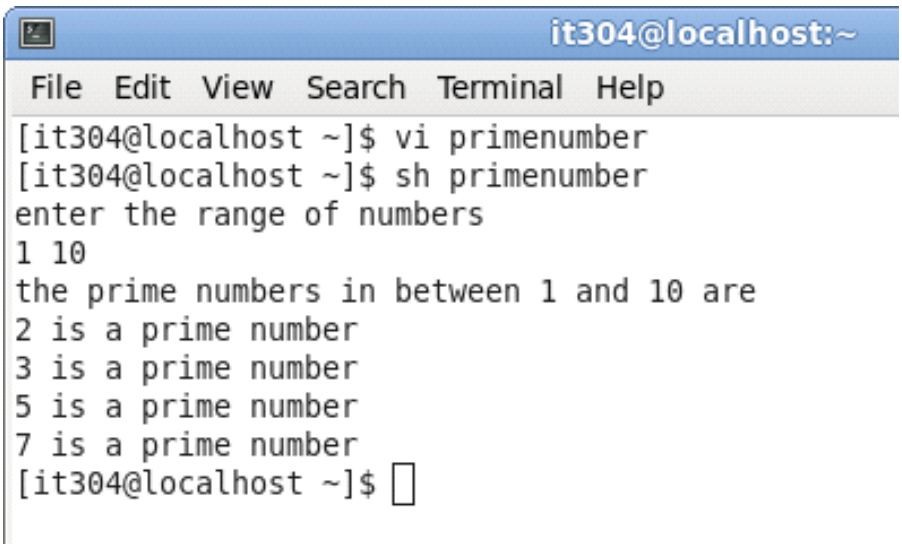
Source code:

```
[it304@localhost ~]$ vi primenumber
#!/bin/sh
echo enter the range of numbers
read num1 num2
echo the prime numbers in between $num1 and $num2 are
c=$num1
#count=0
while [ $c -le $num2 ]
do
count=0
d=2
if [ $c -le 1 ]
then
count=1
fi
while [ $d -le `expr $c / 2` ]
do
if [ `expr $c % $d` -eq 0 ]
then
count=`expr $count + 1`
fi
 d=`expr $d + 1`
done
```

```
if [ $count -eq 0 ]
then
echo $c is a prime number
fi
 c=`expr $c + 1`
done
```

execution:[it304@localhost ~]$sh primenumber.sh

output:



12) To print the first 50 prime numbers.

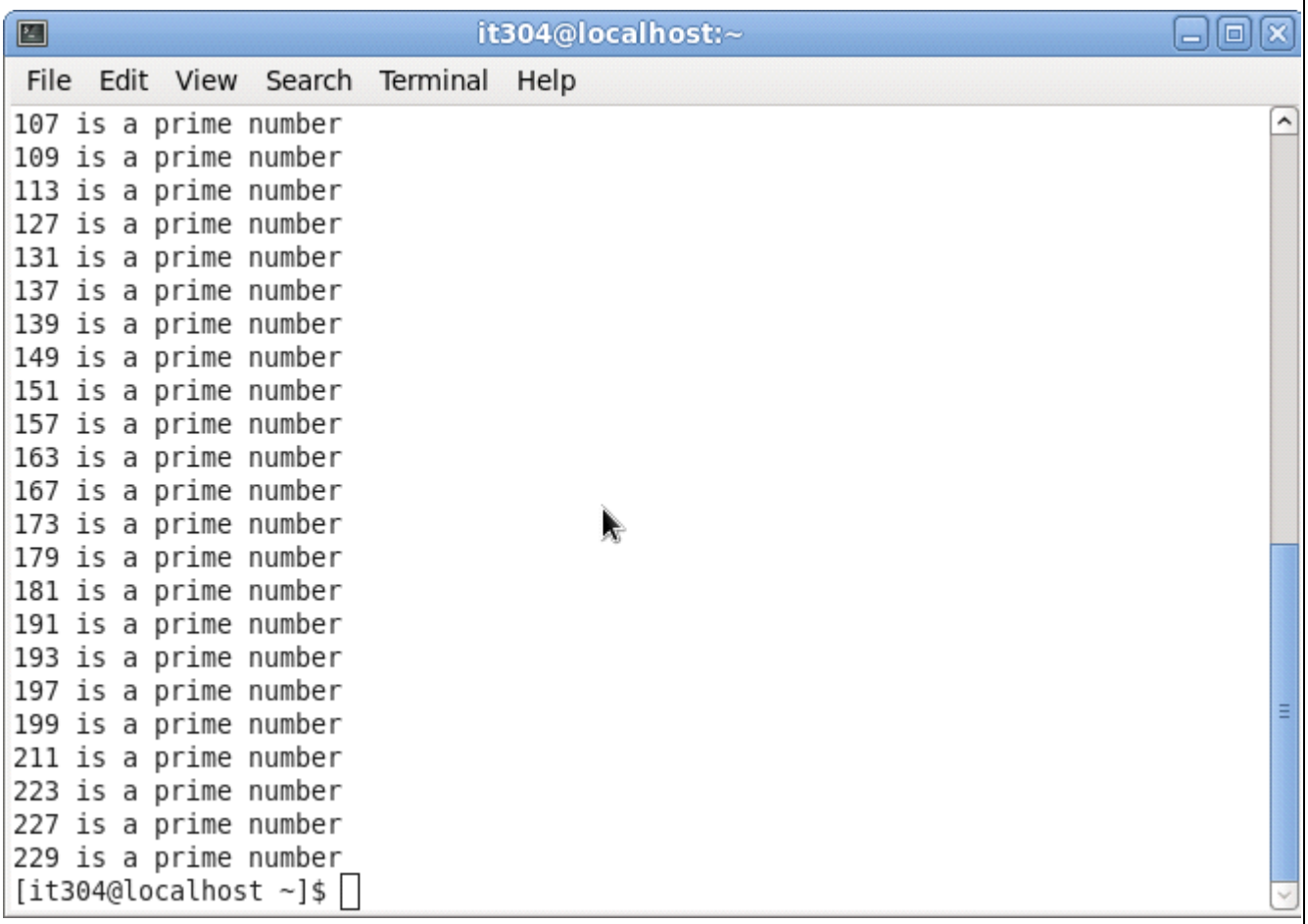
Source code:

```
[it304@localhost ~]$ vi prime50
#!/bin/sh
pcount=0
c=0
while [ $pcount -lt 50 ]
do
count=0
   d=2
if [ $c -le 1 ]
then
count=1
fi
while [ $d -le `expr $c / 2` ]
do
if [ `expr $c % $d` -eq 0 ]
then
count=`expr $count + 1`
fi
      d=`expr $d + 1`
done
if [ $count -eq 0 ]
```

```
then
echo $c is a prime number
pcount=`expr $pcount + 1`
fi
 c=`expr $c + 1`
done
```

execution:[it304@localhost ~]$sh prime50

output:



LABCYCLE 3

File & Process Management Programming

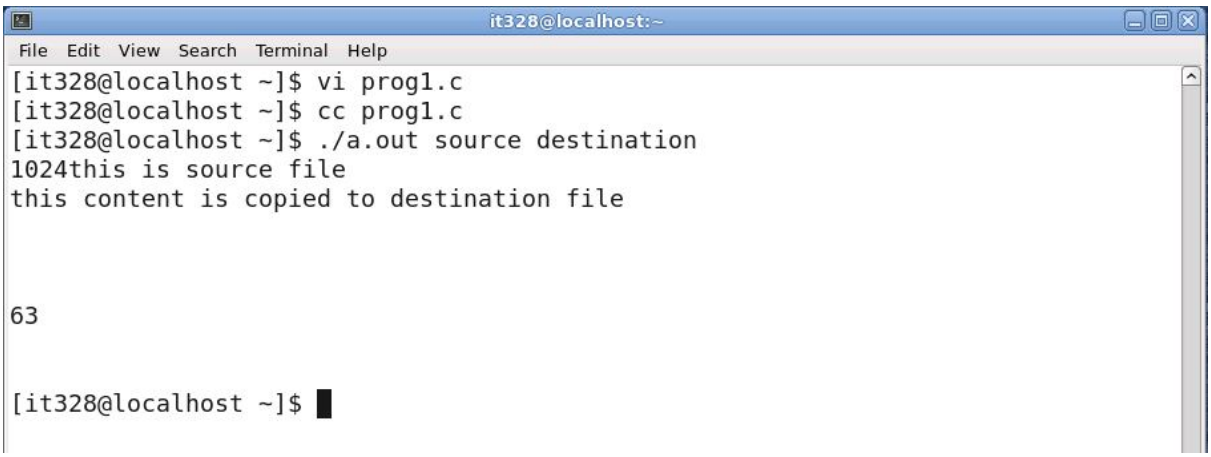
1) Write a C program for copy data from source file to destination file, where the file names are provided as command-line arguments.

Source code:

```
#include<stdio.h>
#include<fcntl.h>
#include<stdlib.h>
char buffer[1024];
main(int argc,char *argv[])
{
        Int fdold,fdnew;
        if(argc!=3)
        {
                printf("Needs two arguments\n");
                exit(1);
        }
        fdold=open(argv[1],O_RDONLY);
        if(fdold==-1)
        {
                printf("Can't open file %s\n",argv[1]);
                exit(2);
        }
        fdnew=creat(argv[2],0666);
        if(fdnew==-1)
        {
                printf("Can't create file %s\n",argv[2]);
```

```
                exit(3);
        }
        copy(fdold,fdnew);
        exit(0);
}
copy(int old,int new)
{
        int count;
        printf("%d",sizeof(buffer));
        while((count=read(old,buffer,sizeof(buffer)))>0){
        printf("%s",buffer);
        write(new,buffer,count);
        printf("\n\n\n%d\n\n\n",count);
}
```

output:



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

Source code:

```
#include<stdio.h>
#include<stdlib.h>
#include<fcntl.h>
main(int argc,char *argv[])
{
        Int fd,skval;
        char c;
        if(argc!=2)
        {
                printf("Need 1 argument\n");
                exit(1);
        }
        fd=open(argv[1],O_RDONLY);
        if(fd==-1)
        {
                printf("Can't open file %s\n",argv[1]);
                exit(2);
        }
        while((skval=read(fd,&c,1))==1)
        {
                printf("Char:%c\n",c);
                skval=lseek(fd,99,1);
                printf("New seek value is:%d\t",skval);
        }
        printf("\n");
        exit(0);    }
```

output:


it328@localhost:~
File Edit View Search Terminal Help
[it328@localhost ~]$ vi prog2.c
[it328@localhost ~]$ cc prog2.c
[it328@localhost ~]$ ./a.out prog2.c
Char:#
New seek value is:100 Char:
New seek value is:200 Char:i
New seek value is:300 Char:1
New seek value is:400 Char:
New seek value is:500
[it328@localhost ~]$

3. 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.

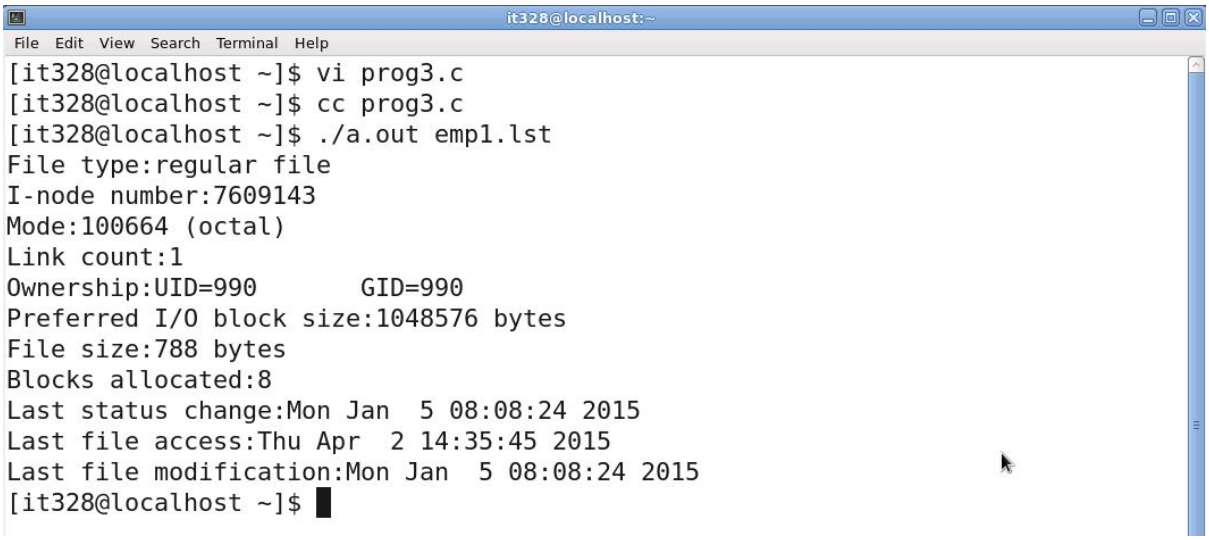
Source code:

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<time.h>
int main(int argc, char *argv[])
{
        struct stat sb;
if (argc != 2)
        {
                printf("Usage: %s <pathname>\n", argv[0]);
                exit(1);
        }

        if (stat(argv[1], &sb) == -1)
        {
                perror("stat");
                exit(2);
        }

        printf("File type:");
        switch (sb.st_mode& S_IFMT)
        {
                case S_IFBLK:
                        printf("block device\n");
```

```
                break;
        case S_IFCHR:
                printf("character device\n");
                break;
        case S_IFDIR:
                printf("directory\n");
                break;
        case S_IFIFO:
                printf("FIFO/pipe\n");
                break;
        case S_IFLNK:
                printf("symlink\n");
                break;
        case S_IFREG:
                printf("regular file\n");
                break;
        case S_IFSOCK:
                printf("socket\n");
                break;
        default:
                printf("unknown?\n");
                break;
}

printf("I-node number:%ld\n",(long) sb.st_ino);
printf("Mode:%lo (octal)\n",(unsigned long) sb.st_mode);
printf("Link count:%ld\n", (long) sb.st_nlink);
printf("Ownership:UID=%ld\tGID=%ld\n",(long)sb.st_uid,(long)sb.st_gid);
printf("Preferred I/O block size:%ld bytes\n",(long)sb.st_blksize);
printf("File size:%lld bytes\n",(long long)sb.st_size);
```

```
        printf("Blocks allocated:%lld\n",(long long)sb.st_blocks);
        printf("Last status change:%s",ctime(&sb.st_ctime));
        printf("Last file access:%s",ctime(&sb.st_atime));
        printf("Last file modification:%s",ctime(&sb.st_mtime));
        exit(0);
}
```

output:



4. Write a C program to display information about the file system.

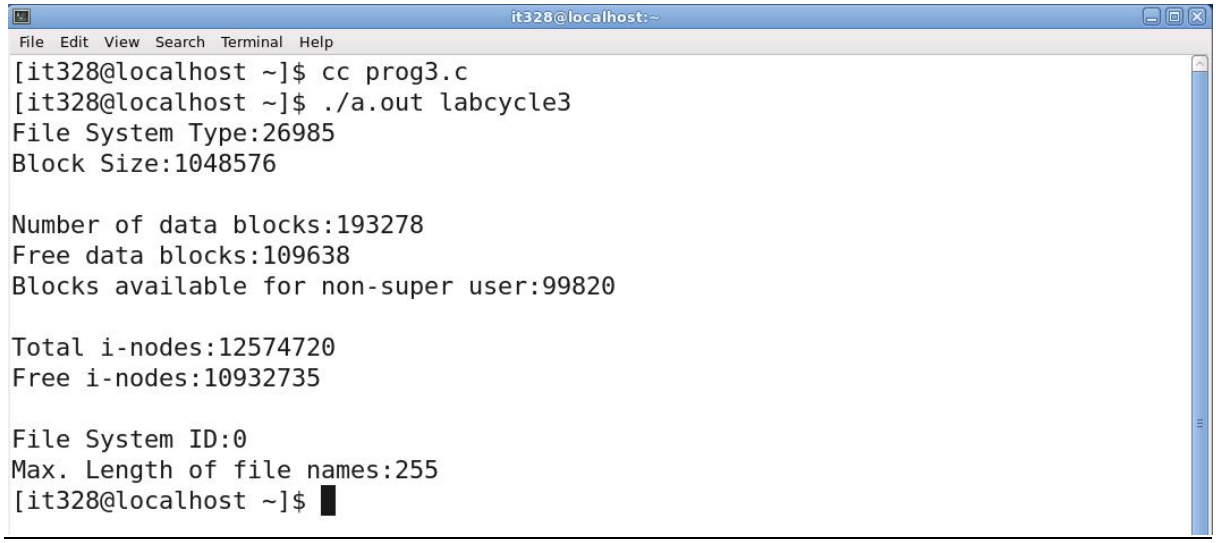
Source code:

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

```
#include<stdlib.h>
#include<sys/types.h>
#include<sys/vfs.h>

int main(intargc, char *argv[])
{
        structstatfsbuf;

        if (argc != 2)
        {
                printf("Usage: %s <devname/pathname>\n", argv[0]);
                exit(1);
        }

        if (statfs(argv[1], &buf) == -1)
        {
                perror("statfs");
                exit(2);
        }

        printf("File System Type:%ld\n",(long)buf.f_type);
        printf("Block Size:%ld\n\n",(long)buf.f_bsize);

        printf("Number of data blocks:%ld\n",(long)buf.f_blocks);
        printf("Free data blocks:%ld\n",(long)buf.f_bfree);
        printf("Blocks available for non-super user:%ld\n\n",(long)buf.f_bavail);

        printf("Total i-nodes:%ld\n",(long)buf.f_files);
        printf("Free i-nodes:%ld\n\n",(long)buf.f_ffree);
```

```
        printf("File System ID:%d\n",buf.f_fsid);
        printf("Max. Length of file names:%ld\n",(long) buf.f_namelen);


        exit(0);
}
```

output:



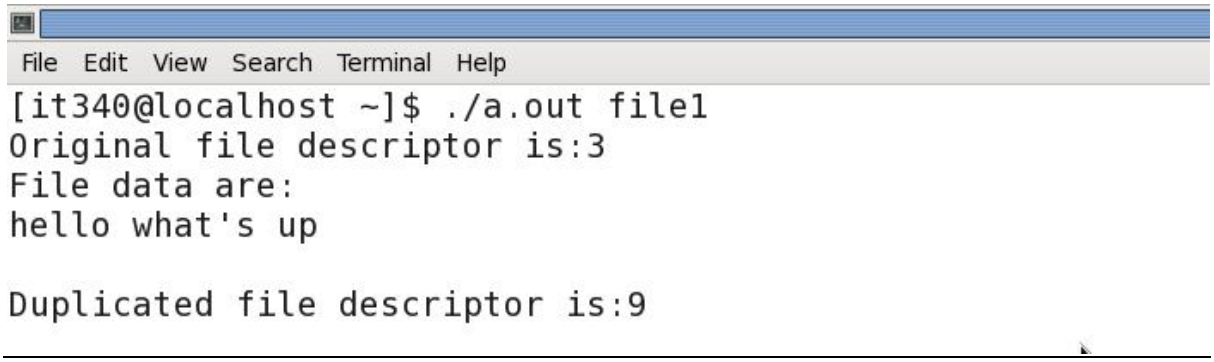
5. Write a C program for demonstrating dup and dup2 system calls

Source code:

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

main(intargc,char *argv[])
{
        inti,fd,newfd=9,number;
        char buffer[20];

        if(argc!=2)
        {
                printf("Need 1 argument\n");
                exit(1);
        }
        fd=open(argv[1],O_RDONLY);
        if(fd==-1)
        {
                printf("Can't open file %s",argv[1]);
                exit(2);
        }

        printf("Original file descriptor is:%d\n",fd);
        number=read(fd,buffer,20);
        printf("File data are:\n");
        for(i=0;i<number;i++)
                printf("%c",buffer[i]);

        newfd=dup2(fd,newfd);
        close(fd);
        printf("\nDuplicated file descriptor is:%d\n",newfd);
```

```
        number=read(newfd,buffer,20);
        for(i=0;i<number;i++)
                printf("%c",buffer[i]);

        printf("\n");
        close(newfd);
}
```

Output:



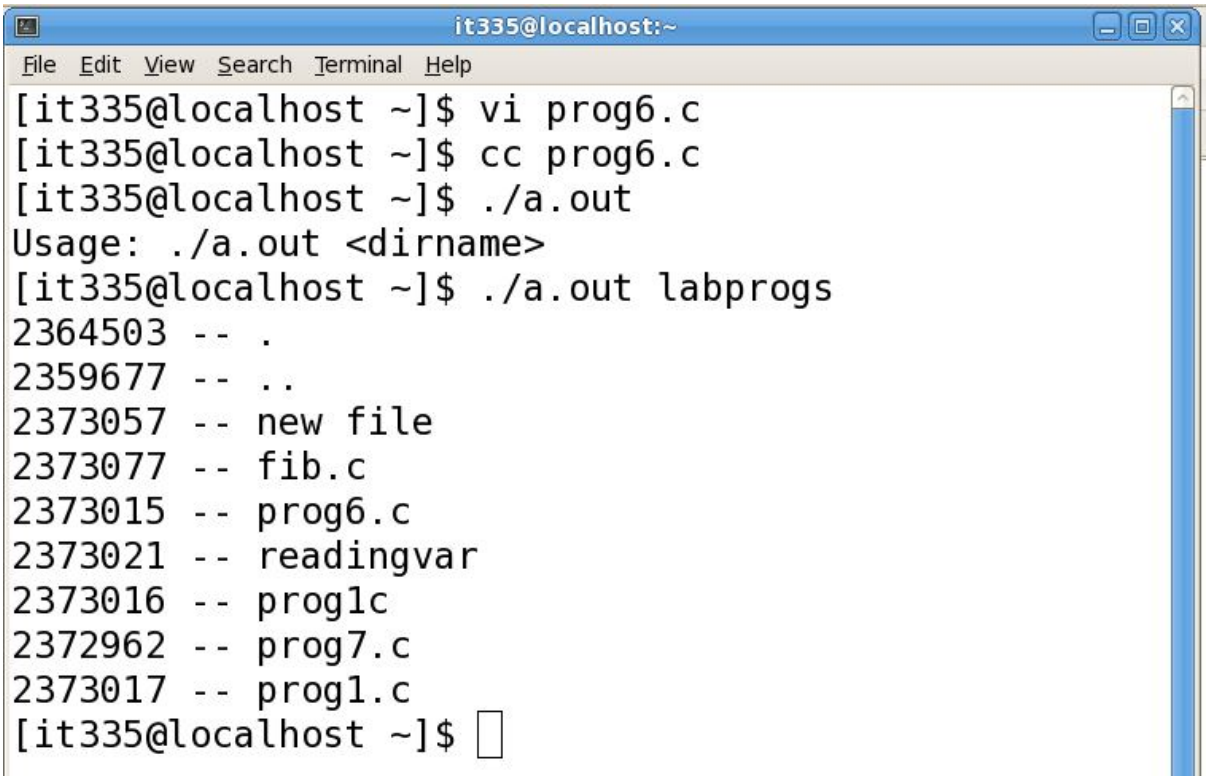
6. Write a C program that prints entries in a directory.

Source code:

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

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

main(intargc,char *argv[])
{
        DIR *dp;
        structdirent *dirp;
        inti;
        charstr[40];

        if(argc!=2)
        {
                printf("Usage: %s <dirname>\n",argv[0]);
                exit(1);
        }
        if((dp=opendir(argv[1]))==NULL)
        {
                printf("Directory open error\n");
                exit(2);
        }
        while((dirp=readdir(dp))!=NULL)
        {
                printf("%ld -- %s\n",dirp->d_ino,dirp->d_name);

        }
        exit(0);
}
```

Output:


it335@localhost:~
File Edit View Search Terminal Help
[it335@localhost ~]$ vi prog6.c
[it335@localhost ~]$ cc prog6.c
[it335@localhost ~]$ ./a.out
Usage: ./a.out <dirname>
[it335@localhost ~]$ ./a.out labprogs
2364503 -- .
2359677 -- ..
2373057 -- new file
2373077 -- fib.c
2373015 -- prog6.c
2373021 -- readingvar
2373016 -- prog1c
2372962 -- prog7.c
2373017 -- prog1.c
[it335@localhost ~]$

7. Write a C program that prints files recurcively in a given directory

Source code:

```
#include<stdio.h>
#include<stdlib.h>
#include<dirent.h>
#include<sys/stat.h>

voidprintdir(char *dir,int depth)
{
        DIR *dp;
        structdirent *dirp;
        struct stat buf;


        if((dp=opendir(dir))==NULL)
        {
                printf("Can't open directory:%s\t%s\n",dp,dir);
                exit(2);
        }

        chdir(dir);
        while((dirp=readdir(dp))!=NULL)
        {
                stat(dirp->d_name,&buf);
                if(S_ISDIR(buf.st_mode))
                {
                        if(strcmp(".",dirp->d_name)==0||strcmp("..",dirp->d_name)==0)
                                continue;
                        printf("%*s%s/\n",depth,"    ",dirp->d_name);
```

```
                depth=depth+4;
                if(dirp->d_name!=NULL)
                        printdir(dirp->d_name,depth);
            }
            else
                printf("%*s%s\n",depth,"    ",dirp->d_name);
        }
        chdir("..");
        closedir(dp);
}

main(intargc,char *argv[])
{
        if(argc!=2)
        {
            printf("Usage:%s <dirname>\n",argv[0]);
            exit(0);
        }
        printf("Directory scan of %s:\n",argv[1]);
        printdir(argv[1],0);
        printf("Scanning finished\n");
        exit(0);
}
```

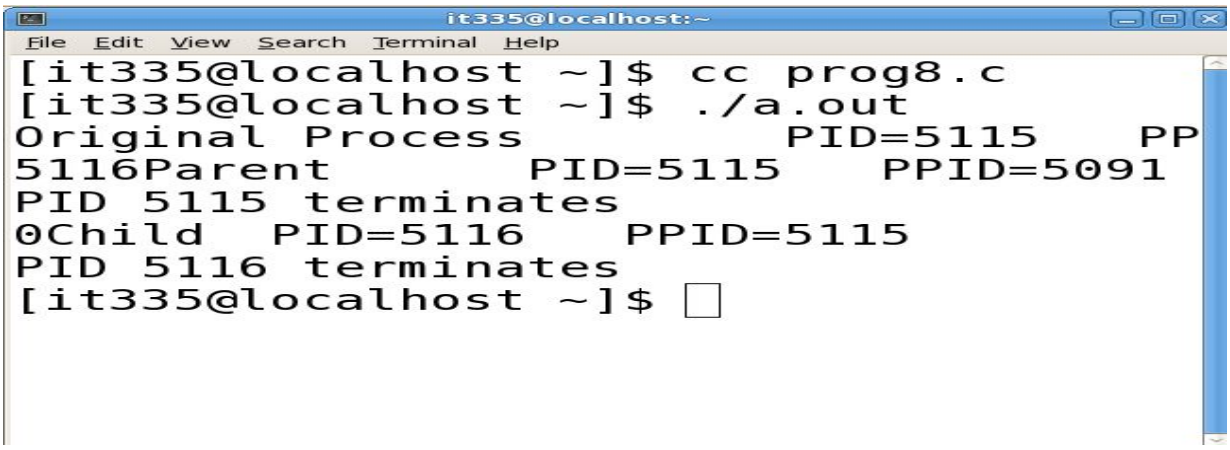
output:

8**. Write a C program to create a process by using fork () system call.**

Source code:

```
#include<stdio.h>
#include<stdlib.h>
main()
{
        intpid;
        printf("Original Process\tPID=%d   PPID=%d\n",getpid(),getppid());
        pid=fork();
        printf("%d",pid);
        if(pid==0)
        {
                printf("Child\tPID=%d   PPID=%d\n",getpid(),getppid());
        }
        else
        {
                printf("Parent\tPID=%d   PPID=%d\n",getpid(),getppid());
        }
        printf("PID %d terminates\n",getpid());
        exit(0);
}
```

output:

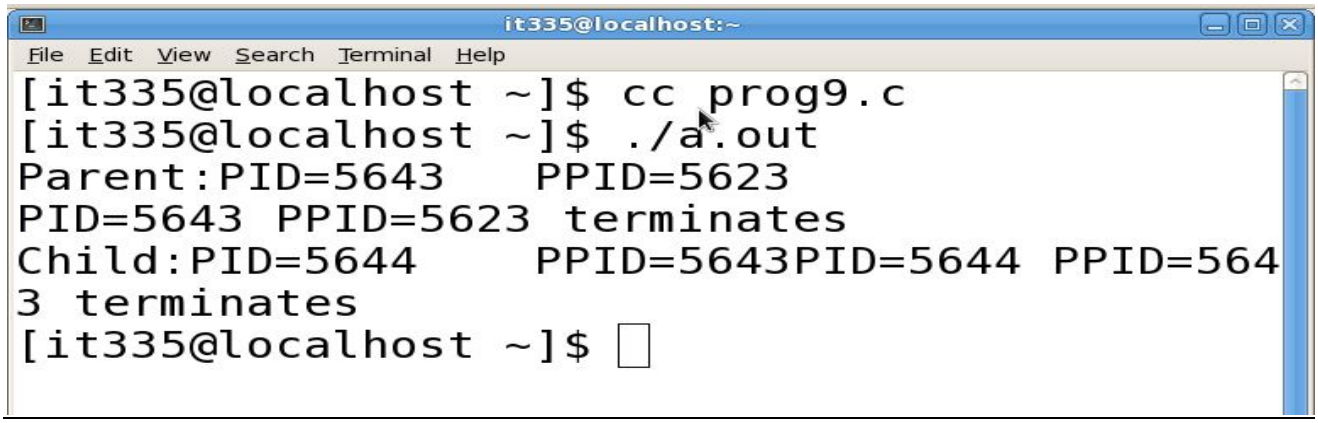


9) Write a C program to create an Orphan Process.

Source code:

```
#include<stdio.h>
#include<stdlib.h>
main()
{
        intpid;

        pid=fork();
        if(pid==0)
        {               // child process
                printf("Child:PID=%d   PPID=%d\n",getpid(),getppid());
                //sleep(5);
        }
        else
        {               //parent process
                printf("Parent:PID=%d   PPID=%d\n",getpid(),getppid());
                //exit(0);
        }
        printf("PID=%d PPID=%d terminates\n",getpid(),getppid());
}
```

output:

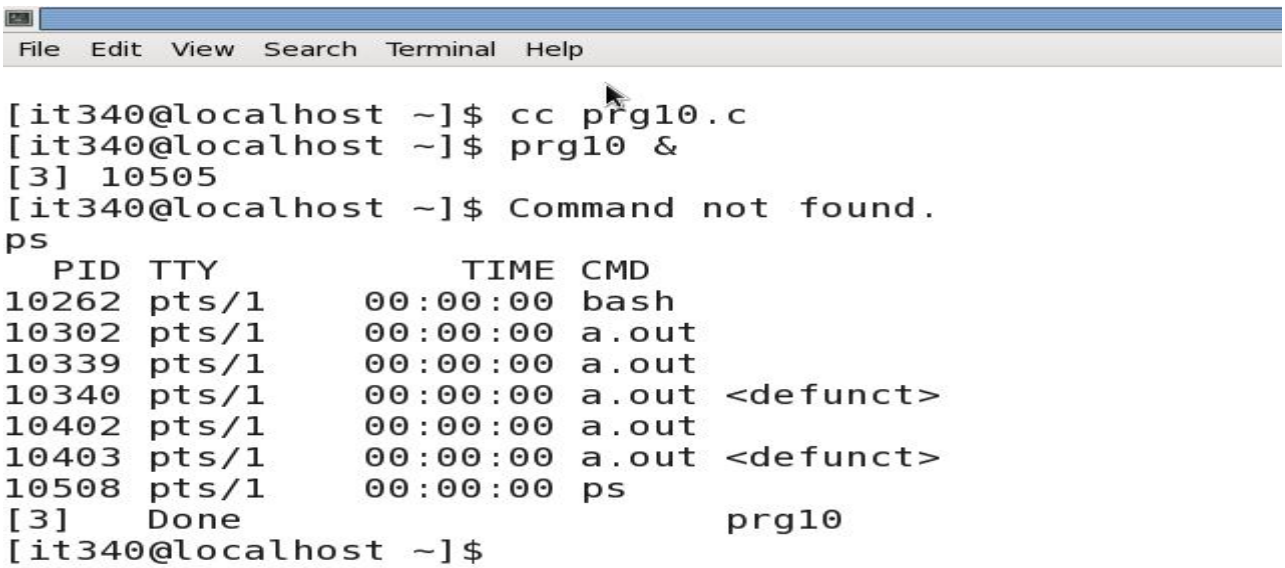


10. Write a C program to demonstrate Zombie process.

Source code:

```
#include<stdio.h>
#include<stdlib.h>
main()
{
        intpid;
        pid=fork();
        if(pid!=0)
        {
                while(1)
                        sleep(1000);
        }
        else
        {
                exit(42);
        }
}
```

output:

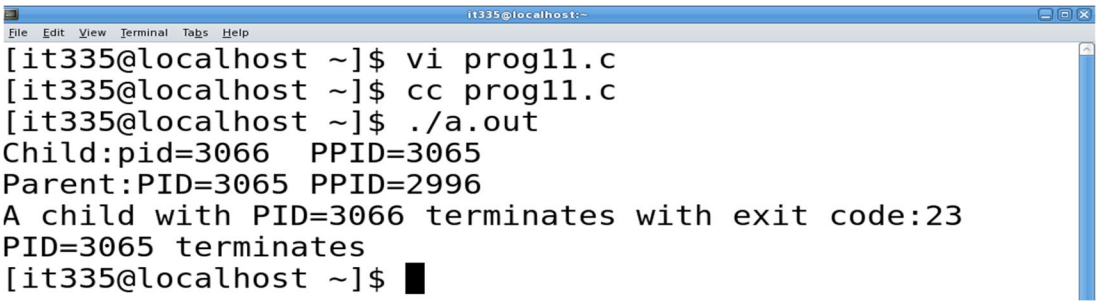


11. Write a C program to demonstrate a parent process that use wait () system call to catch child's exit code

Source code:

```
#include<stdio.h>
#include<stdlib.h>
main()
{
        intpid,childpid,status;
        pid=fork();
        if(pid==0)
        {
                printf("Child:PID=%d\tPPID=%d\n",getpid(),getppid());
                exit(23);
        }
        else
        {
                printf("Parent:PID=%d\tPPID%d\n",getpid(),getppid());
                childpid=wait(&status);
                printf("A child with PID=%d terminates with exit
code:%d\n",childpid,status>> 8);
        }
        printf("PID=%d terminates\n",getpid());
}
```

output:

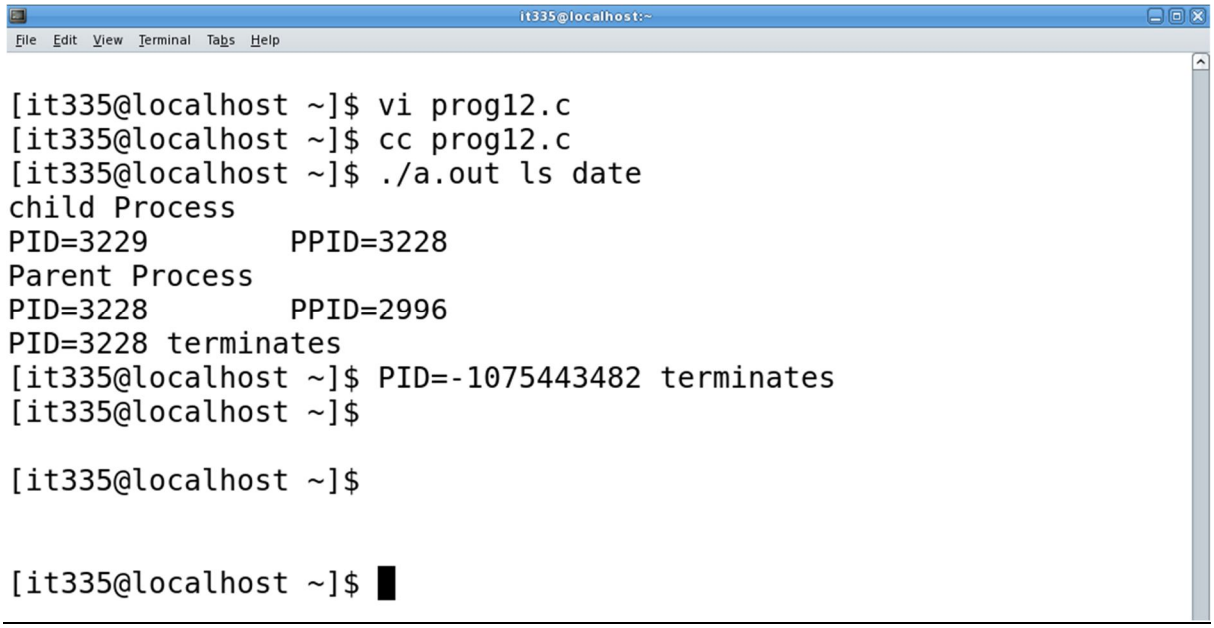


12. Write a C program to Overlay child address space by a program, where the program name is given as command-line argument.

Source code:

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

main(intargc,char *argv[])
{
        intpid;
        if(argc!=3)
        {
                printf("Usage:%s <pathname><prgname>\n",argv[0]);
                exit(1);
        }
        pid=fork();
        if(pid==0)
        {
                printf("child Process\n");
                printf("PID=%d\tPPID=%d\n",getpid(),getppid());
                execl(argv[1],argv[2],NULL);
        }
        else
        {
                printf("Parent Process\n");
                printf("PID=%d\tPPID=%d\n",getpid(),getppid());
        }
        printf("PID=%d terminates\n");
        exit(0);
}
```

output:



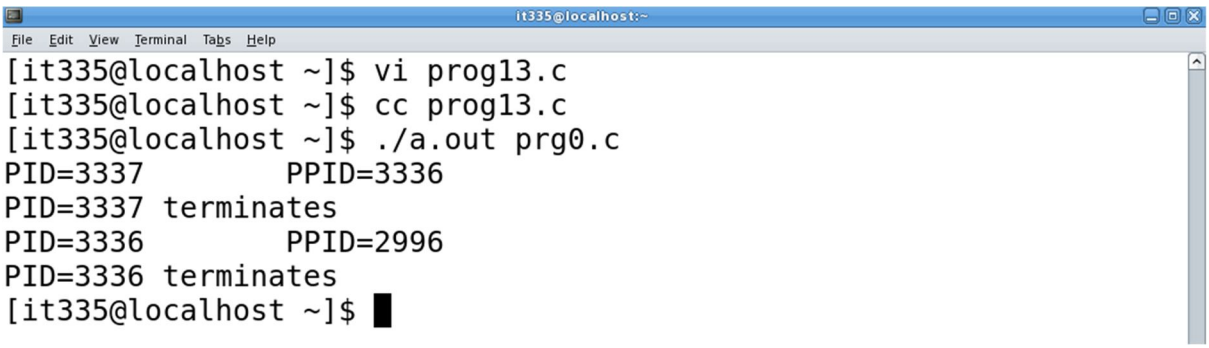
13. Program that demonstrates both child and parent processes writes data to the same file

Source code:

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

main(intargc,char *argv[])
{
        intfd,pid,num=0;
        if(argc!=2)
        {
                printf("Usage:%s <filename>\n",argv[0]);
                exit(1);
        }
        fd=open(argv[1],O_RDWR);
```

```
        if(fd==-1)
        {
                printf("Can't Open file %s",argv[1]);
                exit(2);
        }
        ++num;
        pid=fork();
        if(pid==0)              //child process
        {
                printf("PID=%d\tPPID=%d\n",getpid(),getppid());
                write(fd,&num,sizeof(num));
                num++;
        }
        else
        {
                printf("PID=%d\tPPID=%d\n",getpid(),getppid());
                write(fd,&num,sizeof(num));
                num++;
        }
        printf("PID=%d terminates\n",getpid());
        exit(0);
}
```

output:



LABCYCLE 4

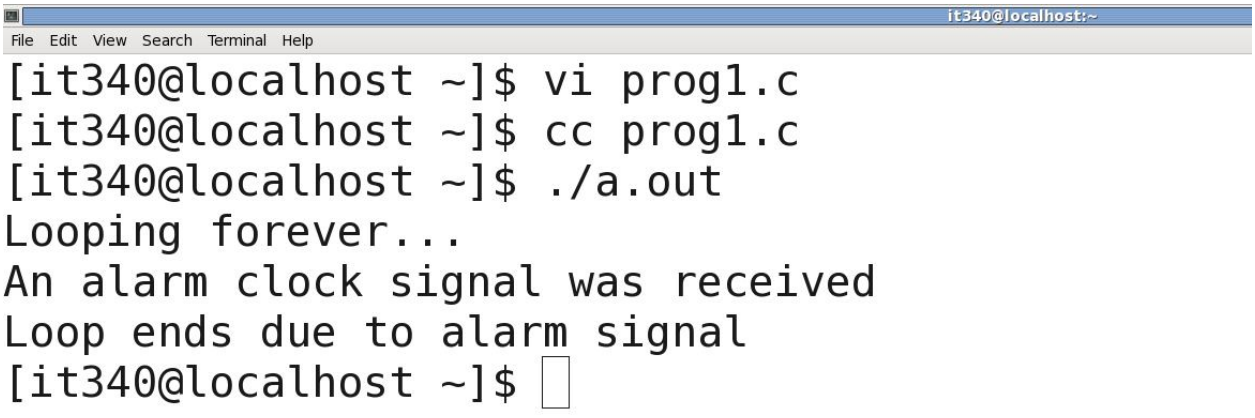
Signal & IPC Programming

1. Write a C program for Requesting an alarm signal to execute user defined alarm handler.

Source code:

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

intalarmflag=0;
voidalarmHandler();

main()
{

        signal(SIGALRM,alarmHandler);
        alarm(5);
        printf("Looping forever...\n");
        while(!alarmflag)
        {
                pause();
        }
        printf("Loop ends due to alarm signal\n");
}

voidalarmHandler()
{
        printf("An alarm clock signal was received\n");
        alarmflag=1;
}
```

Output:



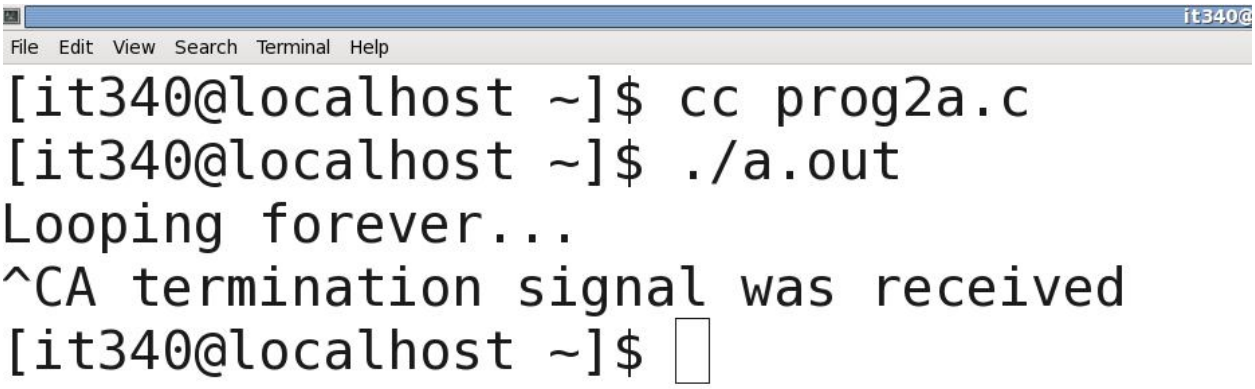
2. Write a C program to demonstrate terminal signals (control-c & control-z).

(control-c)

Source code:

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

voidmytermHandler();        //Forward declaration of termination handler
main()
{

        signal(SIGINT,mytermHandler);    //Install signal handler
        printf("Looping forever...\n");
        while(1);
        printf("This line should never be executed\n");
}

voidmytermHandler()
{
        printf("A termination signal was received\n");
        exit(0);
}
```

Output:



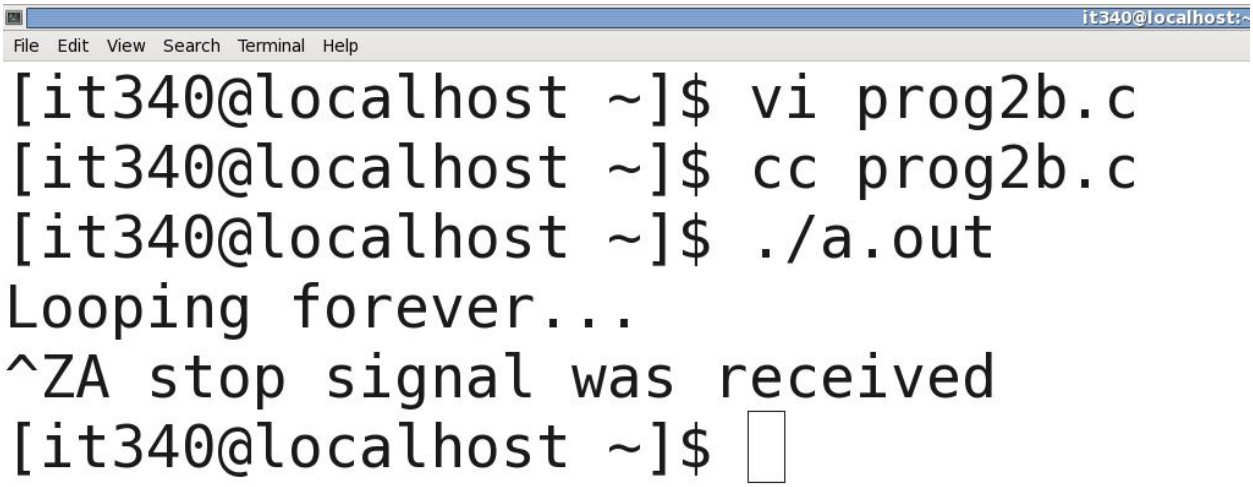
(control-z)

Source code:

```
#include<stdio.h>
#include<signal.h>
#include<stdlib.h>
 (control-z)

voidmystpHandler();
```

```
main()
{

        signal(SIGTSTP,mystpHandler);
        printf("Looping forever...\n");
        while(1);
        printf("This line should never be executed\n");
}

voidmystpHandler()
{
        printf("A stop signal was received\n");
        exit(0);
}
```

Output:



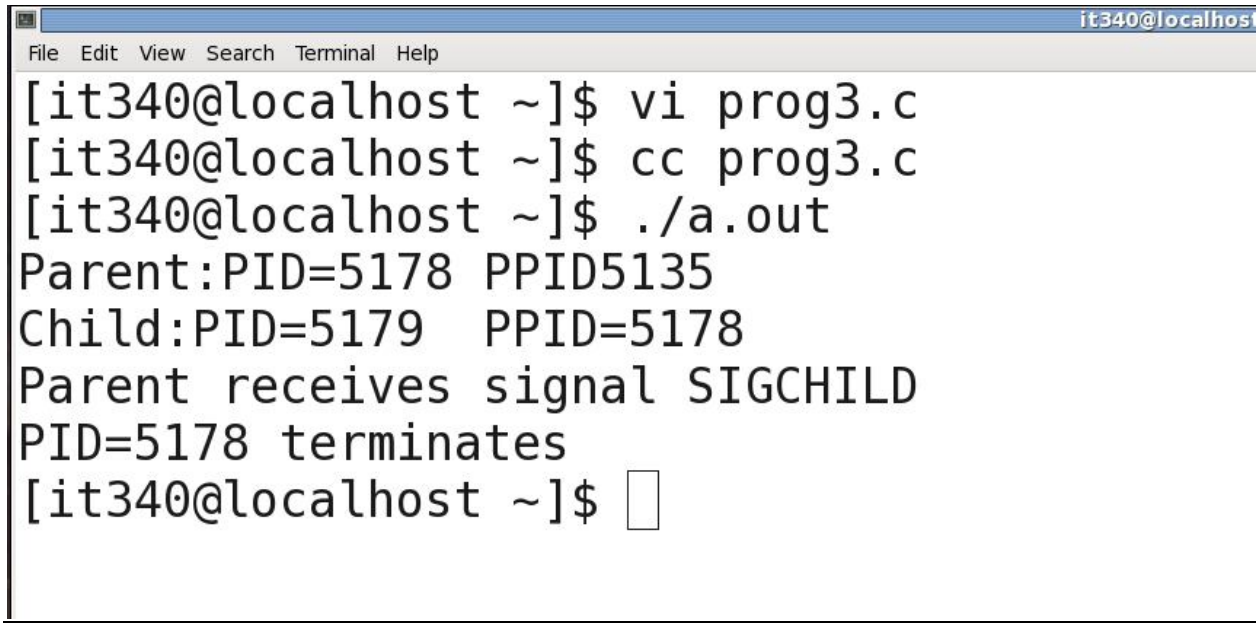
---

3. Write a C program to override child termination signal by the parent process.

Source code:

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

voidchildsigHandler();

main()
{
        intpid;

        signal(SIGCHLD,childsigHandler);
```

```
        pid=fork();
        if(pid==0)
        {
                printf("Child:PID=%d\tPPID=%d\n",getpid(),getppid());
                sleep(3);
                exit(0);
        }
        else
        {
                printf("Parent:PID=%d\tPPID%d\n",getpid(),getppid());
                wait();
        }
        printf("PID=%d terminates\n",getpid());
}

voidchildsigHandler()
{
        printf("Parent receives signal SIGCHILD\n");
}
```

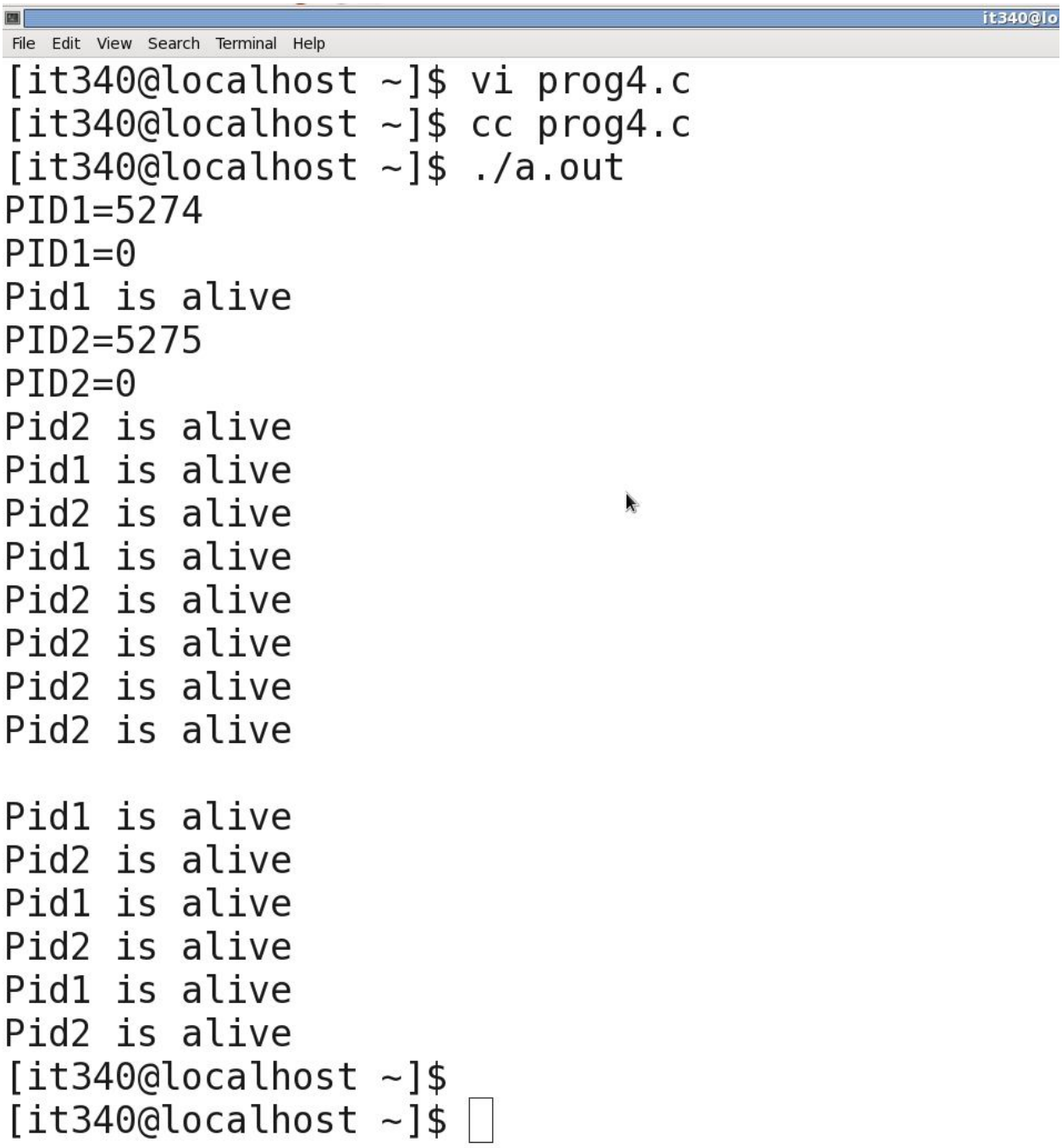


4**. Write a C program to demonstrate Suspending and Resuming Processes.**

Source code:

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

main()
{
        int pid1,pid2;

        pid1=fork();
        printf("PID1=%d\n",pid1);
        if(pid1==0)             //First child
        {
                while(1)
                {
                        printf("Pid1 is alive\n");
                        sleep(1);
                }
        }
        pid2=fork();
        printf("PID2=%d\n",pid2);
        if(pid2==0)             //Second child
        {
                while(1)
                {
                        printf("Pid2 is alive\n");
                        sleep(1);
                }
        }
        sleep(3);
        kill(pid1,SIGSTOP);  //Suspend first child
        sleep(3);
        kill(pid1,SIGCONT); //Resume first child
        sleep(3);
        kill(pid1,SIGINT);    //Kill first child
        kill(pid2,SIGINT);    //Kill second child
}
```

Output

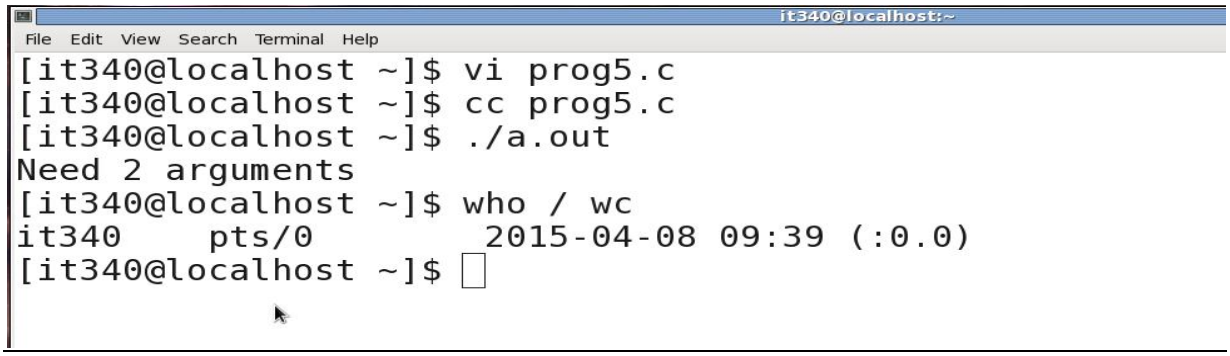

it340@lo
File Edit View Search Terminal Help
[it340@localhost ~]$ vi prog4.c
[it340@localhost ~]$ cc prog4.c
[it340@localhost ~]$ ./a.out
PID1=5274
PID1=0
Pid1 is alive
PID2=5275
PID2=0
Pid2 is alive
Pid1 is alive
Pid2 is alive
Pid1 is alive
Pid2 is alive
Pid2 is alive
Pid2 is alive
Pid2 is alive
Pid1 is alive
Pid2 is alive
Pid1 is alive
Pid2 is alive
Pid1 is alive
Pid2 is alive
[it340@localhost ~]$
[it340@localhost ~]$

5. Write a C program for Un-named pipes to send data from first process to the second process.

Source code:

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

#define READ 0
#define WRITE 1

main(intargc,char *argv[])
{
        intfd[2];
        if(argc!=3)
        {
                printf("Need 2 arguments\n");
                exit(1);
        }
        pipe(fd);               //Creating an unnamed pipe
        if(fork()==0)           //Child writer
        {
                close(fd[READ]);        //close unused end
                dup2(fd[WRITE],1);  //Duplicate used end to stdout
                close(fd[WRITE]);       //Close original used end
                execlp(argv[1],argv[1],NULL);       //Execute writer program
                perror("Connect");      //Should never execute
        }
        else                    //Parent reader
        {
                close(fd[WRITE]);       //Close unused end
                dup2(fd[READ],0);   //Duplicate used end to stdin
                close(fd[READ]);        //Close original used end
                execlp(argv[2],argv[2],NULL);       //Execute reader program
                perror("Connect");      //Should never execute
        }
        exit(0);
}
```

Output:



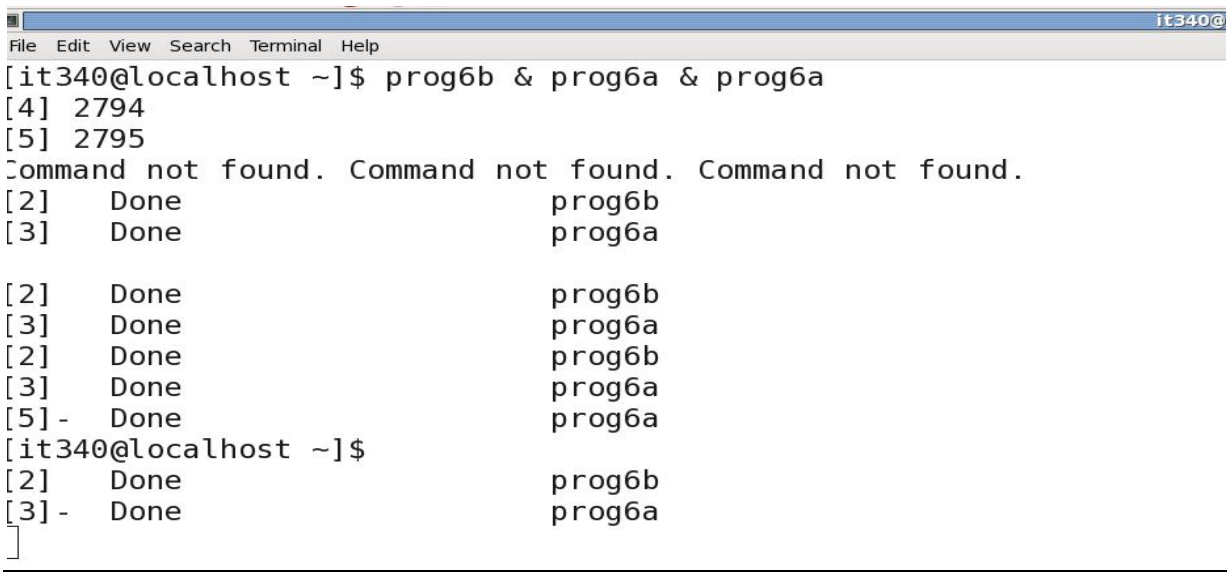
6. Write two C programs that demonstrates Named pipes, Reader and Writer Processes.

for Named pipes - Reader Process

Source code:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
main()
{
        intfd;
        charstr[100];
        unlink("apipe");                //Remove named pipe if it already exists
        mknod("apipe",S_IFIFO,0);  //Create named pipe
        chmod("apipe",0660);            //Change its permissions
        fd=open("apipe",O_RDONLY);      //Open pipe for reading
        while(readline(fd,str))                 //Display received message
                printf("%s\n",str);
        close(fd);
}
readline(intfd,char *str)
{
        int n;
        do                              //Read charecters until NULL or end of input
        {
                n=read(fd,str,1);       //Read one charecter
        }while(n>0 && *str++ !=NULL);

        return(n>0);                    //Return false if end of input
}
```

Output:

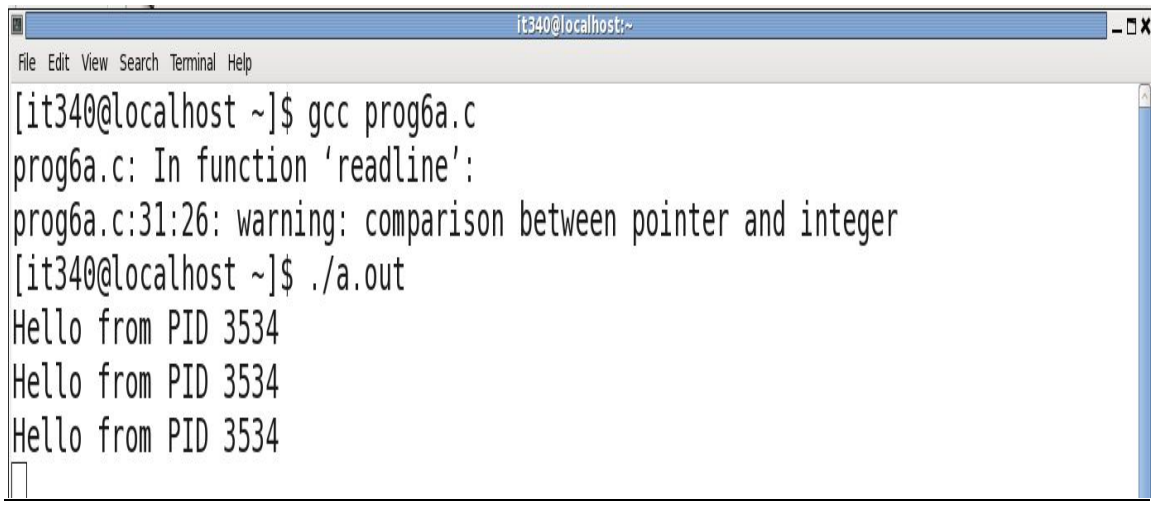


for Named pipes - Writer Process

Source code:

```
#include<stdio.h>
#include<fcntl.h>
#include<string.h>
main()
{
        intfd,messLen,i;
        char message[100];
        sprintf(message,"Hello from PID %d",getpid());
        messLen=strlen(message)+1;
        do                              //Keep trying to open the file
        {
                fd=open("apipe",O_WRONLY);      //open named piep for writing
                if(fd == -1)
                        sleep(1);               //Try again in 1 second
        }while(fd==-1);
        for(i=1;i<=3;i++)               //Send 3 messages
        {
                write(fd,message,messLen); //Write message down pipe
                sleep(2);                       //Pause a while
        }
        close(fd);
}
```

Output:



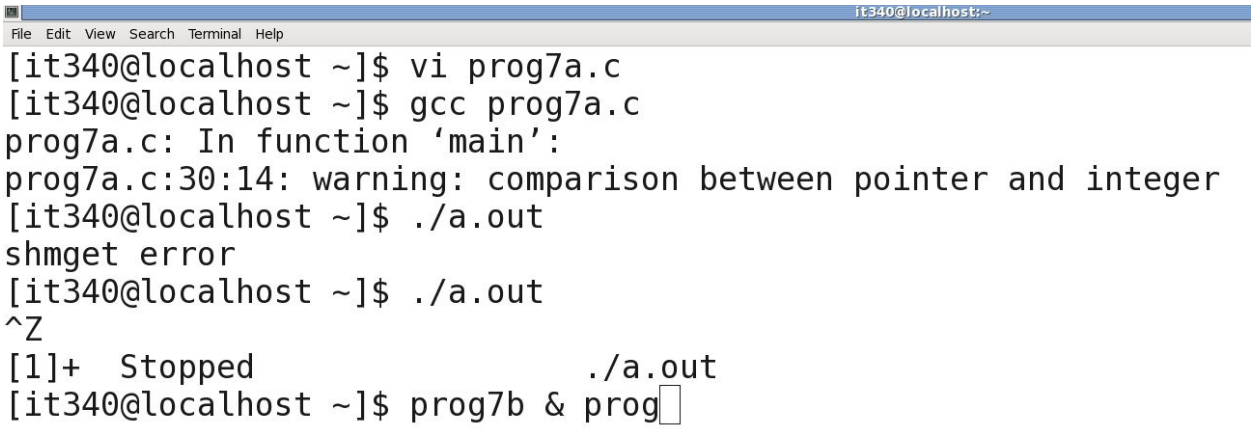
7. Write C program that demonstrates IPC through shared memory.

Demo program for shared memory - reader

Source code:

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>

#define SHMSZ 27

main()
{
        intshmid;
        key_t key;
        char *shm,*s;

        //Shared memory segment "5678", created by server
        key=5678;

        //Locate the segment
        shmid=shmget(key,SHMSZ,0666);
        if(shmid<0)
        {
                printf("shmget error\n");
                exit(1);
        }

        //Attach the segment to data space
        shm=shmat(shmid,NULL,0);
        if(shm == (char *) -1)
        {
                printf("Shared memory attach error\n");
                exit(2);
        }

        //Read the data
        for(s=shm;*s!=NULL;s++)
                putchar(*s);
        putchar('\n');

        //Change the first charecter of the segment
        //to '*', indicating that the client have read the segment
        *shm='*';

        exit(0);
}
```

Output:



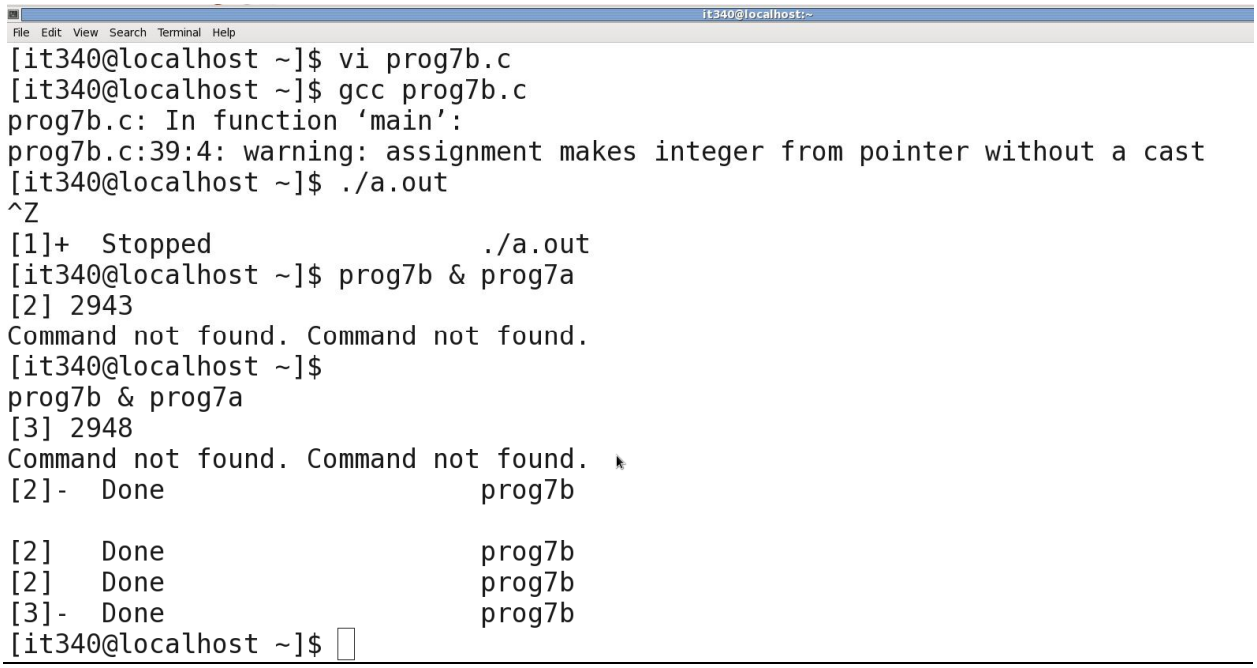
Demo program for shared memory – writer

Source code:

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>

#define SHMSZ 27

main()
{
        char c;
        intshmid;
        key_t key;
        char *shm,*s;

        //Name shared memory segment "5678"
        key=5678;

        //Create the segment
        shmid=shmget(key,SHMSZ,IPC_CREAT|0666);
        if(shmid<0)
        {
                printf("shmget error\n");
                exit(1);
        }

        //Attach the segment to data space
        shm=shmat(shmid,NULL,0);
```

```
        if(shm == (char *) -1)
        {
                printf("Shared memory attach error\n");
                exit(2);
        }

        //Place some data into the memory for other processes to read
        s=shm;

        for(c='a';c<='z';c++)
                *s++=c;
        *s=NULL;

        //Finally, server wait's until the other process change
        //the first charecter of our memory to '*', indicating
        //that it has read the data
        while(*shm != '*')
                sleep(1);

        exit(0);
}
```

Output:



Remove

Source code:

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>


#define SHMSZ 27

main()
{
        intshmid;
        key_t key;

        key=5678;


        shmid=shmget(key,SHMSZ,IPC_RMID);

        if(shmctl(shmid,IPC_RMID,0)==-1)
        {
                printf("Shared memory detach error\n");
                exit(1);
        }
        printf("Shared memory with key %d is detached.\n",key);
        exit(0);
}
```

Output: