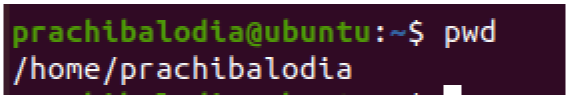
Prachi Balodia

20BDS0177

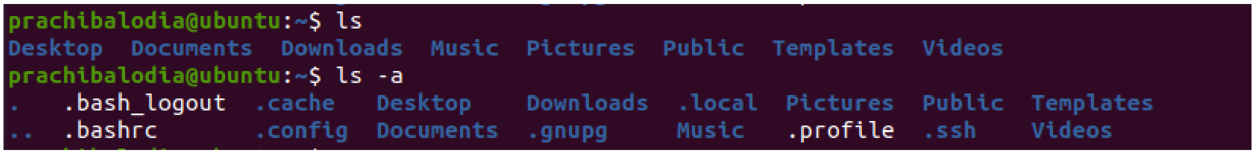
Basic Linux Commands

1. pwd: To know which directory we are in, we can use the “pwd” command. It gives us the absolute path, which means the path that starts from the root.



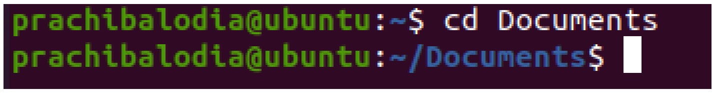
```
prachibalodia@ubuntu:~$ pwd
/home/prachibalodia
```

2. ls — We use the "ls" command to know what files are in the directory we are in. You can see all the hidden files by using the command “ls -a”.



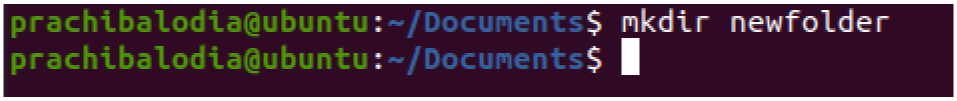
```
prachibalodia@ubuntu:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
prachibalodia@ubuntu:~$ ls -a
.   .bash_logout  .cache   Desktop    Downloads  .local  Pictures  Public  Templates
..  .bashrc       .config  Documents  .gnupg     Music   .profile  .ssh    Videos
```

3. cd — We use the "cd" command to go to a directory.



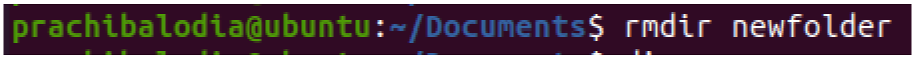
```
prachibalodia@ubuntu:~$ cd Documents
prachibalodia@ubuntu:~/Documents$
```

4. mkdir— We use the mkdir command when we need to create a folder or a directory.



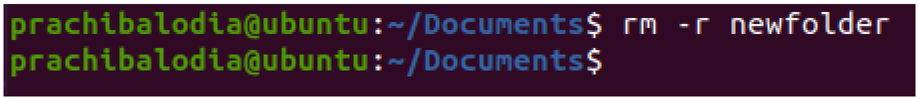
```
prachibalodia@ubuntu:~/Documents$ mkdir newfolder
prachibalodia@ubuntu:~/Documents$
```

5. rmdir- We use rmdir to delete a directory. But rmdir can only be used to delete an empty directory.

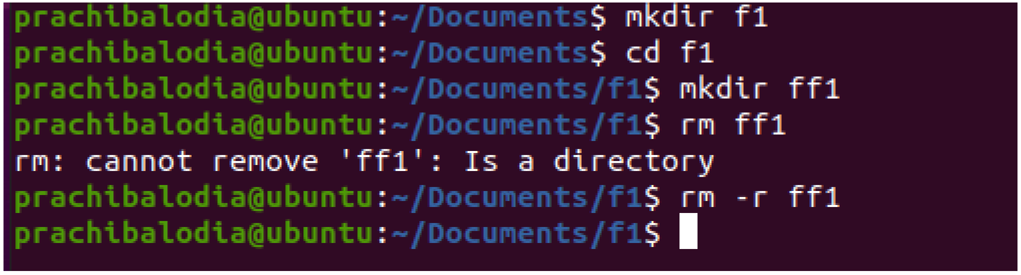


```
prachibalodia@ubuntu:~/Documents$ rmdir newfolder
```

6. rm - Use the rm command to delete files and directories. Use "rm -r" to delete just the directory. It deletes both the folder and the files it contains when using only the rm command.

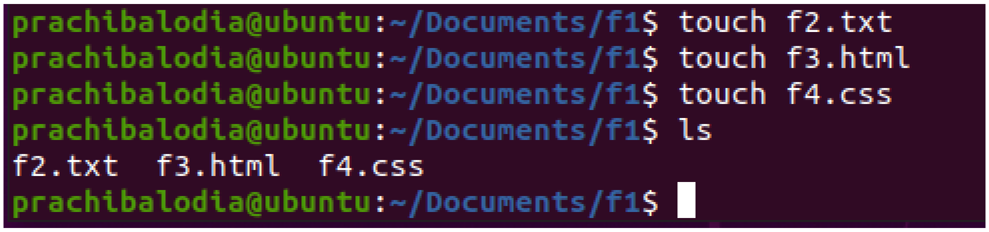


```
prachibalodia@ubuntu:~/Documents$ rm -r newfolder
prachibalodia@ubuntu:~/Documents$
```



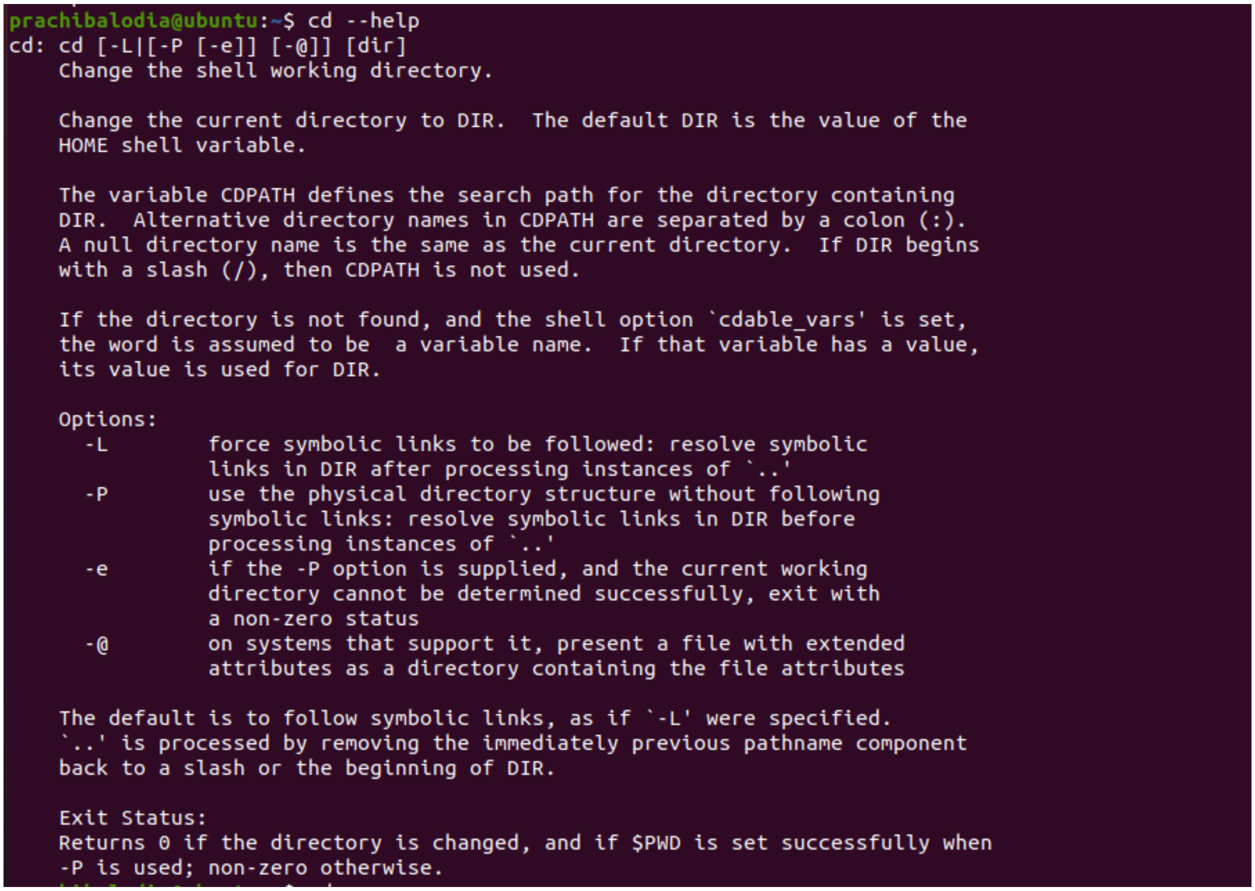
```
prachibalodia@ubuntu:~/Documents$ mkdir f1
prachibalodia@ubuntu:~/Documents$ cd f1
prachibalodia@ubuntu:~/Documents/f1$ mkdir ff1
prachibalodia@ubuntu:~/Documents/f1$ rm ff1
rm: cannot remove 'ff1': Is a directory
prachibalodia@ubuntu:~/Documents/f1$ rm -r ff1
prachibalodia@ubuntu:~/Documents/f1$
```

7. Touch - The touch command is used to create a file. It can be anything, from an empty txt file to an empty zip file.



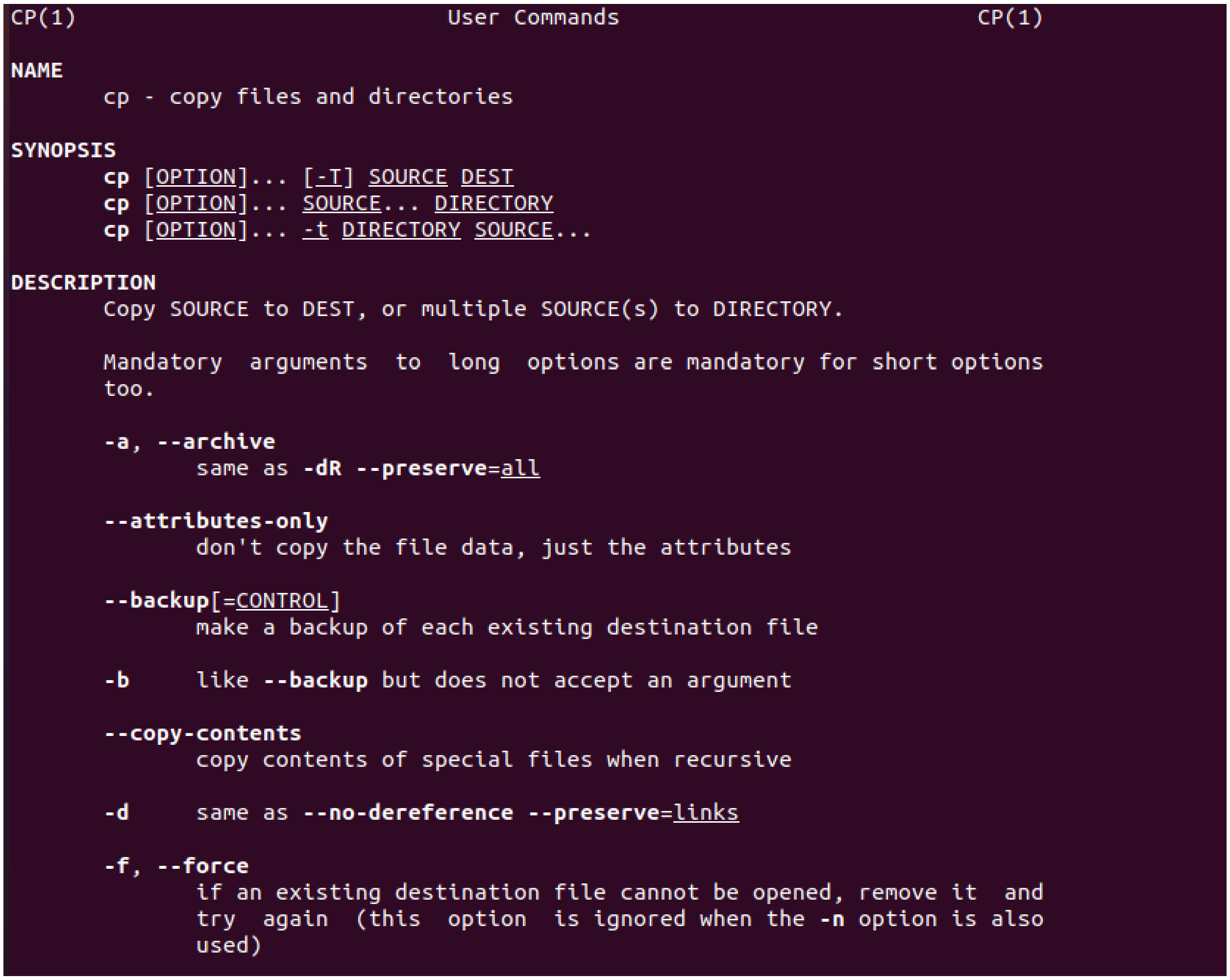
```
prachibalodia@ubuntu:~/Documents/f1$ touch f2.txt
prachibalodia@ubuntu:~/Documents/f1$ touch f3.html
prachibalodia@ubuntu:~/Documents/f1$ touch f4.css
prachibalodia@ubuntu:~/Documents/f1$ ls
f2.txt  f3.html  f4.css
prachibalodia@ubuntu:~/Documents/f1$ 
```

8. --help — To know more about a command and how to use it, use the man command. It shows the manual pages of the command.



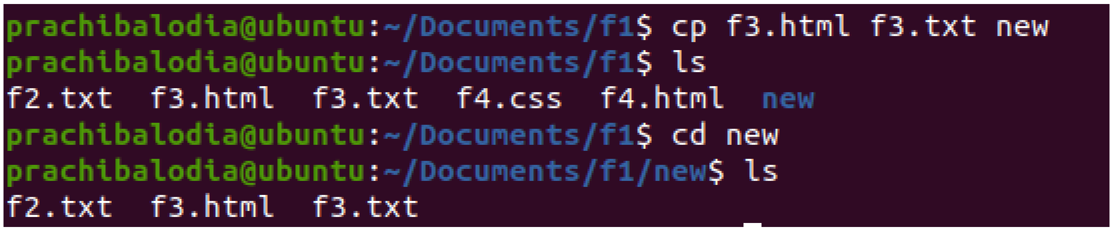
```
prachibalodia@ubuntu:~$ cd --help
cd: cd [-L|[-P [-e]] [-@]] [dir]
    Change the shell working directory.

    Change the current directory to DIR.  The default DIR is the value of the
    HOME shell variable.

    The variable CDPATH defines the search path for the directory containing
    DIR.  Alternative directory names in CDPATH are separated by a colon (:).
    A null directory name is the same as the current directory.  If DIR begins
    with a slash (/), then CDPATH is not used.

    If the directory is not found, and the shell option `cdable_vars' is set,
    the word is assumed to be  a variable name.  If that variable has a value,
    its value is used for DIR.

    Options:
      -L        force symbolic links to be followed: resolve symbolic
                links in DIR after processing instances of `..'
      -P        use the physical directory structure without following
                symbolic links: resolve symbolic links in DIR before
                processing instances of `..'
      -e        if the -P option is supplied, and the current working
                directory cannot be determined successfully, exit with
                a non-zero status
      -@        on systems that support it, present a file with extended
                attributes as a directory containing the file attributes

    The default is to follow symbolic links, as if `-L' were specified.
    `..' is processed by removing the immediately previous pathname component
    back to a slash or the beginning of DIR.

    Exit Status:
    Returns 0 if the directory is changed, and if $PWD is set successfully when
    -P is used; non-zero otherwise.
```

9. man- It helps us to know more about a command.



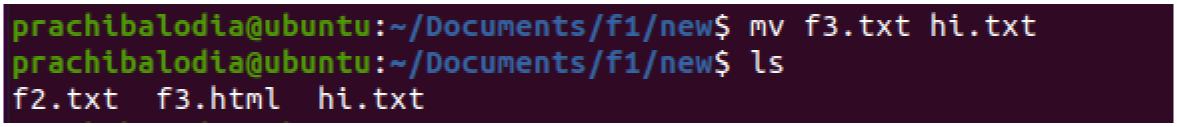
```
CP(1)                            User Commands                           CP(1)

NAME
       cp - copy files and directories

SYNOPSIS
       cp [OPTION]... [-T] SOURCE DEST
       cp [OPTION]... SOURCE... DIRECTORY
       cp [OPTION]... -t DIRECTORY SOURCE...

DESCRIPTION
       Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY.

       Mandatory  arguments  to  long  options are mandatory for short options
       too.

       -a, --archive
              same as -dR --preserve=all

       --attributes-only
              don't copy the file data, just the attributes

       --backup[=CONTROL]
              make a backup of each existing destination file

       -b     like --backup but does not accept an argument

       --copy-contents
              copy contents of special files when recursive

       -d     same as --no-dereference --preserve=links

       -f, --force
              if an existing destination file cannot be opened, remove it  and
              try  again  (this  option  is ignored when the -n option is also
              used)
```

10. cp — We use the cp command to copy files through the command line. It takes two arguments: The first is the location of the file to be copied, the second is where to copy.



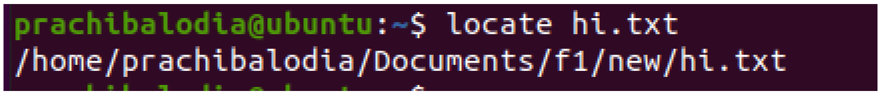
```
prachibalodia@ubuntu:~/Documents/f1$ cp f3.html f3.txt new
prachibalodia@ubuntu:~/Documents/f1$ ls
f2.txt  f3.html  f3.txt  f4.css  f4.html  new
prachibalodia@ubuntu:~/Documents/f1$ cd new
prachibalodia@ubuntu:~/Documents/f1/new$ ls
f2.txt  f3.html  f3.txt
```

11. mv — We use the mv command to move files through the command line. We can also use the mv command to rename a file.



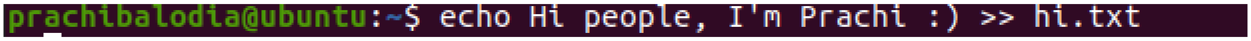
```
prachibalodia@ubuntu:~/Documents/f1/new$ mv f3.txt hi.txt
prachibalodia@ubuntu:~/Documents/f1/new$ ls
f2.txt  f3.html  hi.txt
```

12. locate — The locate command is used to locate a file in a Linux system, just like the search command in Windows. This command is useful when you don't know where a file is saved or the actual name of the file.



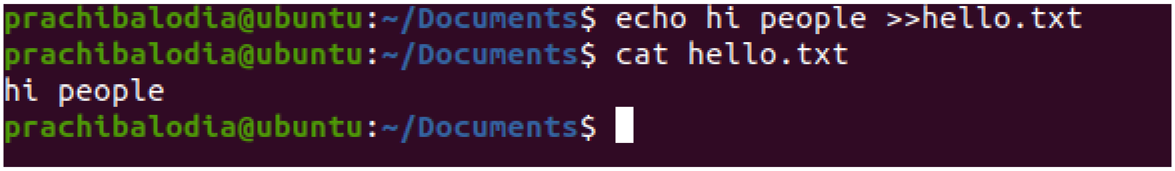
```
prachibalodia@ubuntu:~$ locate hi.txt
/home/prachibalodia/Documents/f1/new/hi.txt
```

13. echo — The "echo" command helps us move some data, usually text into a file.



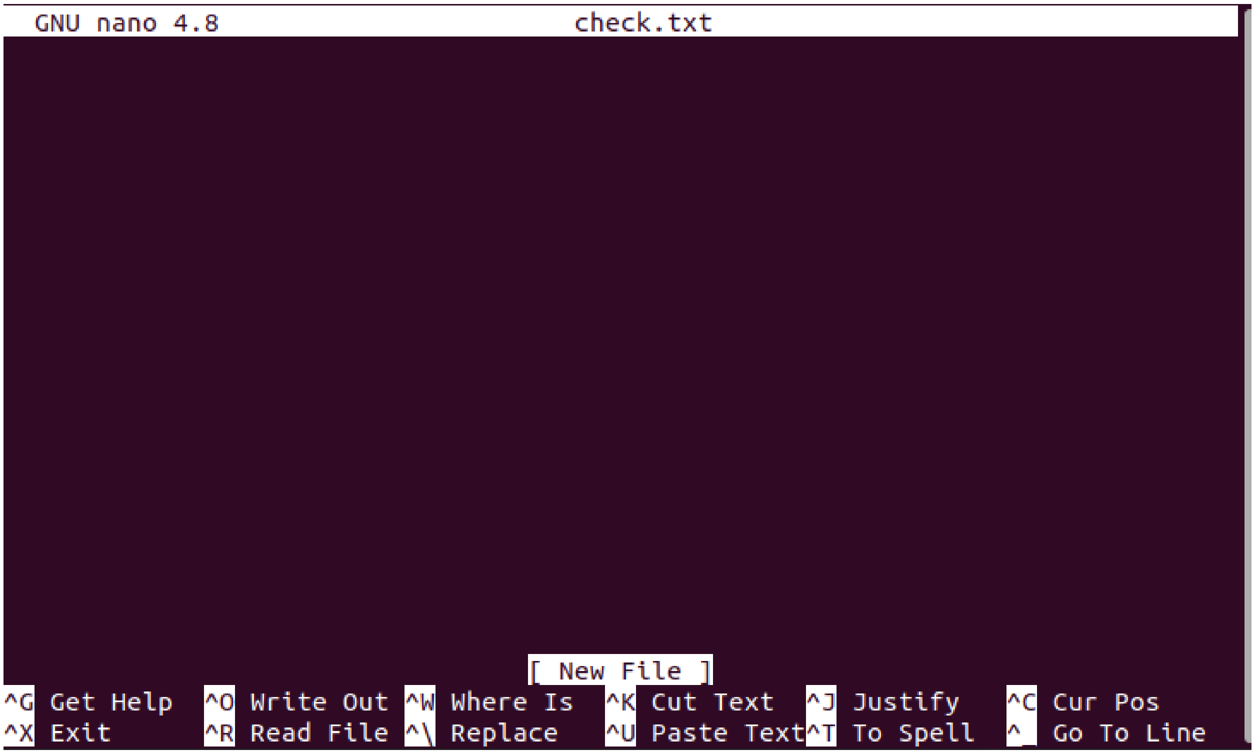
```
prachibalodia@ubuntu:~$ echo Hi people, I'm Prachi :) >> hi.txt
```

14. cat — Use the cat command to display the contents of a file. It is usually used to easily view programs.



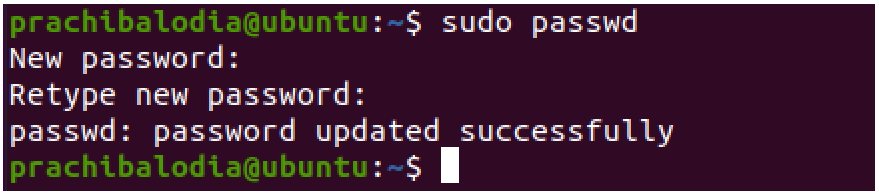
```
prachibalodia@ubuntu:~/Documents$ echo hi people >>hello.txt
prachibalodia@ubuntu:~/Documents$ cat hello.txt
hi people
prachibalodia@ubuntu:~/Documents$
```

15. nano- We can use it for create new files.



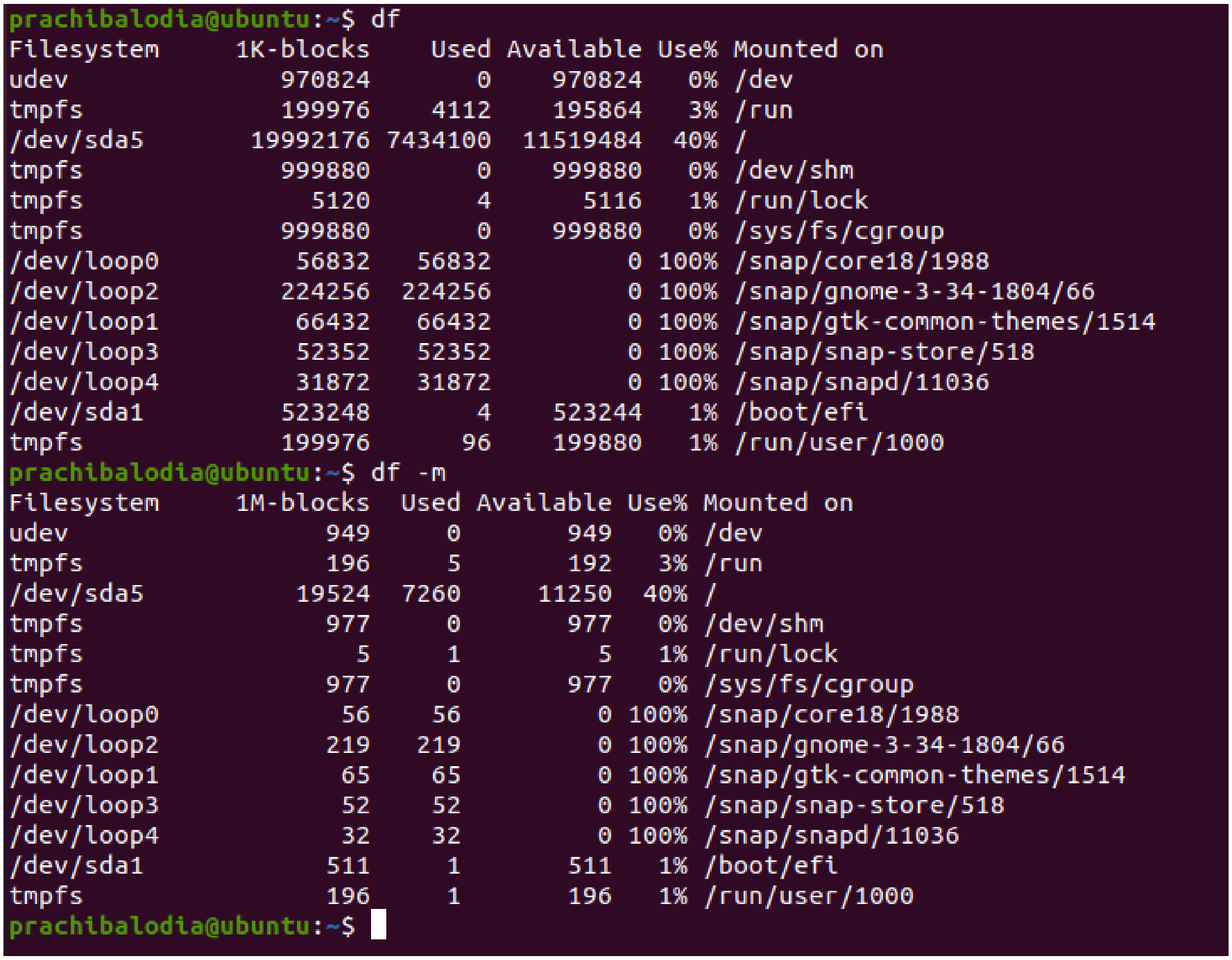
```
  GNU nano 4.8                          check.txt











                                   [ New File ]
^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos
^X Exit        ^R Read File   ^\ Replace     ^U Paste Text^T To Spell      ^_ Go To Line
```

16. sudo — A widely used command in the Linux command line, sudo stands for "SuperUser Do". So, if we want any command to be done with administrative or root privileges, you can use the sudo command.



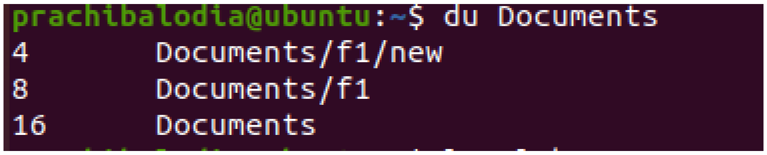
```
prachibalodia@ubuntu:~$ sudo passwd
New password:
Retype new password:
passwd: password updated successfully
prachibalodia@ubuntu:~$
```

17. df — We use the df command to see the available disk space in each of the partitions in your system. You can just type in df in the command line and you can see each mounted partition and their used/available space in % and in KBs. If we want it shown in megabytes, you can use the command “df -m”.



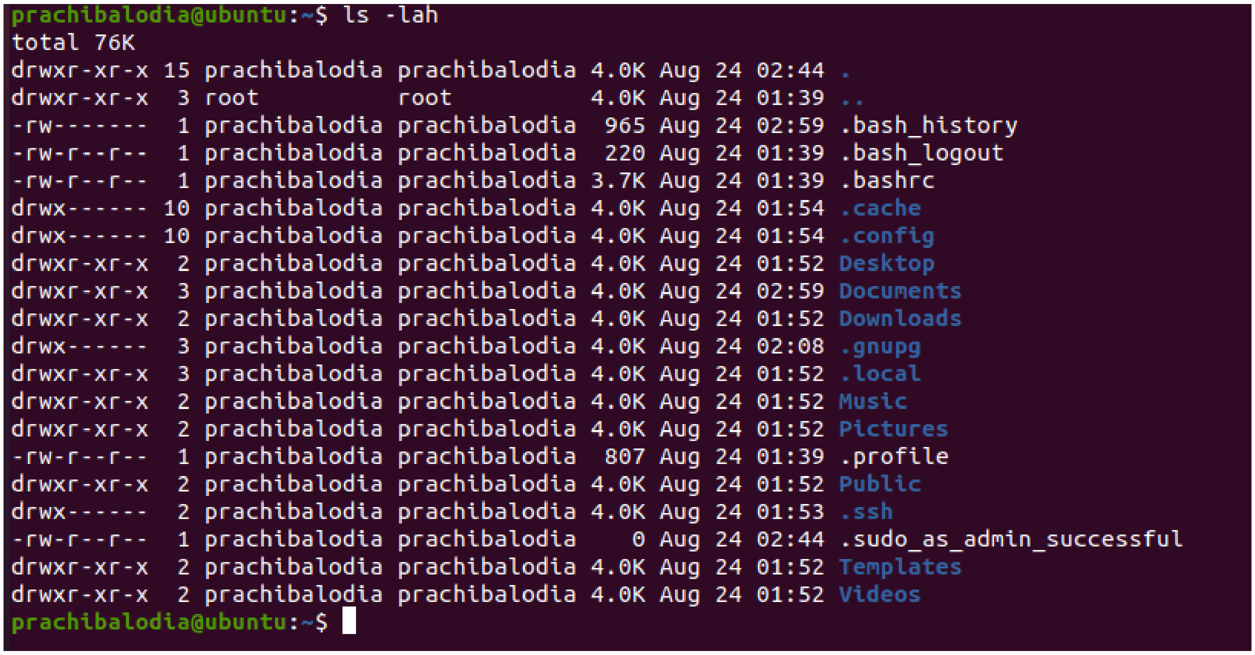
```
prachibalodia@ubuntu:~$ df
Filesystem     1K-blocks    Used Available Use% Mounted on
udev              970824       0    970824   0% /dev
tmpfs             199976    4112    195864   3% /run
/dev/sda5       19992176 7434100  11519484  40% /
tmpfs             999880       0    999880   0% /dev/shm
tmpfs               5120       4      5116   1% /run/lock
tmpfs             999880       0    999880   0% /sys/fs/cgroup
/dev/loop0         56832   56832         0 100% /snap/core18/1988
/dev/loop2        224256  224256         0 100% /snap/gnome-3-34-1804/66
/dev/loop1         66432   66432         0 100% /snap/gtk-common-themes/1514
/dev/loop3         52352   52352         0 100% /snap/snap-store/518
/dev/loop4         31872   31872         0 100% /snap/snapd/11036
/dev/sda1         523248       4    523244   1% /boot/efi
tmpfs             199976      96    199880   1% /run/user/1000
prachibalodia@ubuntu:~$ df -m
Filesystem     1M-blocks  Used Available Use% Mounted on
udev                 949     0       949   0% /dev
tmpfs                196     5       192   3% /run
/dev/sda5          19524  7260     11250  40% /
tmpfs                977     0       977   0% /dev/shm
tmpfs                  5     1         5   1% /run/lock
tmpfs                977     0       977   0% /sys/fs/cgroup
/dev/loop0            56    56         0 100% /snap/core18/1988
/dev/loop2           219   219         0 100% /snap/gnome-3-34-1804/66
/dev/loop1            65    65         0 100% /snap/gtk-common-themes/1514
/dev/loop3            52    52         0 100% /snap/snap-store/518
/dev/loop4            32    32         0 100% /snap/snapd/11036
/dev/sda1            511     1       511   1% /boot/efi
tmpfs                196     1       196   1% /run/user/1000
prachibalodia@ubuntu:~$ 
```

18. du — Use du to know the disk usage of a file in your system.



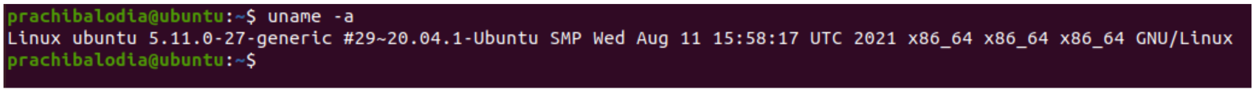
```
prachibalodia@ubuntu:~$ du Documents
4       Documents/f1/new
8       Documents/f1
16      Documents
```

19. ls -lah – l is used to view the file sizes of all the files in a folder.



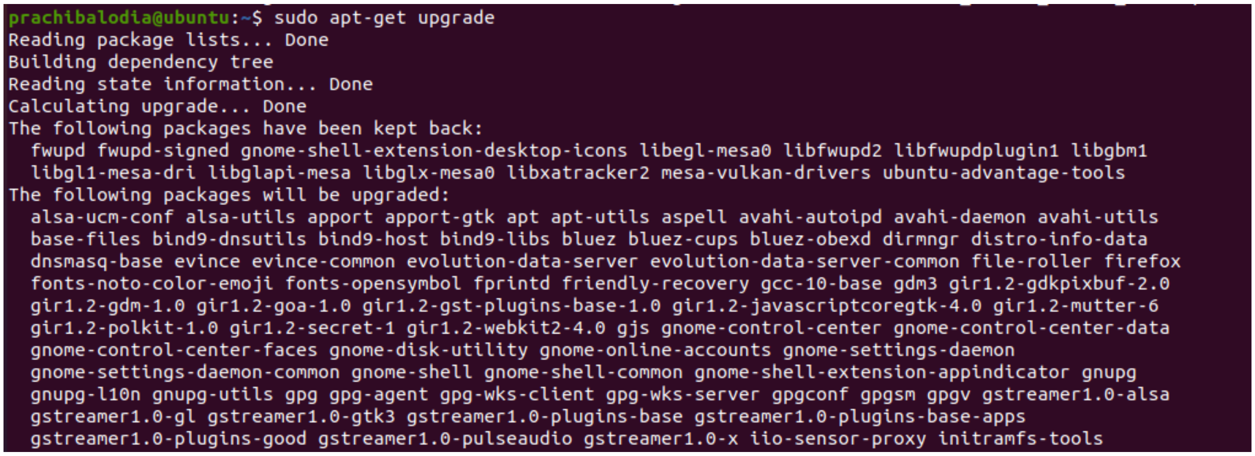
```
prachibalodia@ubuntu:~$ ls -lah
total 76K
drwxr-xr-x 15 prachibalodia prachibalodia 4.0K Aug 24 02:44 .
drwxr-xr-x  3 root          root          4.0K Aug 24 01:39 ..
-rw-------  1 prachibalodia prachibalodia  965 Aug 24 02:59 .bash_history
-rw-r--r--  1 prachibalodia prachibalodia  220 Aug 24 01:39 .bash_logout
-rw-r--r--  1 prachibalodia prachibalodia 3.7K Aug 24 01:39 .bashrc
drwx------ 10 prachibalodia prachibalodia 4.0K Aug 24 01:54 .cache
drwx------ 10 prachibalodia prachibalodia 4.0K Aug 24 01:54 .config
drwxr-xr-x  2 prachibalodia prachibalodia 4.0K Aug 24 01:52 Desktop
drwxr-xr-x  3 prachibalodia prachibalodia 4.0K Aug 24 02:59 Documents
drwxr-xr-x  2 prachibalodia prachibalodia 4.0K Aug 24 01:52 Downloads
drwx------  3 prachibalodia prachibalodia 4.0K Aug 24 02:08 .gnupg
drwxr-xr-x  3 prachibalodia prachibalodia 4.0K Aug 24 01:52 .local
drwxr-xr-x  2 prachibalodia prachibalodia 4.0K Aug 24 01:52 Music
drwxr-xr-x  2 prachibalodia prachibalodia 4.0K Aug 24 01:52 Pictures
-rw-r--r--  1 prachibalodia prachibalodia  807 Aug 24 01:39 .profile
drwxr-xr-x  2 prachibalodia prachibalodia 4.0K Aug 24 01:52 Public
drwx------  2 prachibalodia prachibalodia 4.0K Aug 24 01:53 .ssh
-rw-r--r--  1 prachibalodia prachibalodia    0 Aug 24 02:44 .sudo_as_admin_successful
drwxr-xr-x  2 prachibalodia prachibalodia 4.0K Aug 24 01:52 Templates
drwxr-xr-x  2 prachibalodia prachibalodia 4.0K Aug 24 01:52 Videos
prachibalodia@ubuntu:~$ 
```

20. tar — We use tar to work with tarballs (or files compressed in a tarball archive) in the Linux command line.
21. . zip, unzip — We use zip to compress files into a zip archive, and unzip to extract files from a zip archive.
22. uname — We use uname to show the information about the system your Linux distro is running. Using the command “uname -a” prints most of the information about the system. This prints the kernel release date, version, processor type, etc.



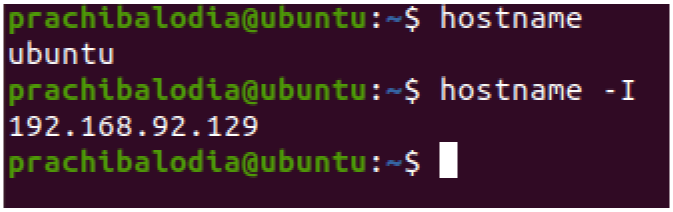
```
prachibalodia@ubuntu:~$ uname -a
Linux ubuntu 5.11.0-27-generic #29~20.04.1-Ubuntu SMP Wed Aug 11 15:58:17 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux
prachibalodia@ubuntu:~$
```

23. apt-get — We use apt to work with packages in the Linux command line. Use apt-get to install packages. This requires root privileges, so use the sudo command with it.



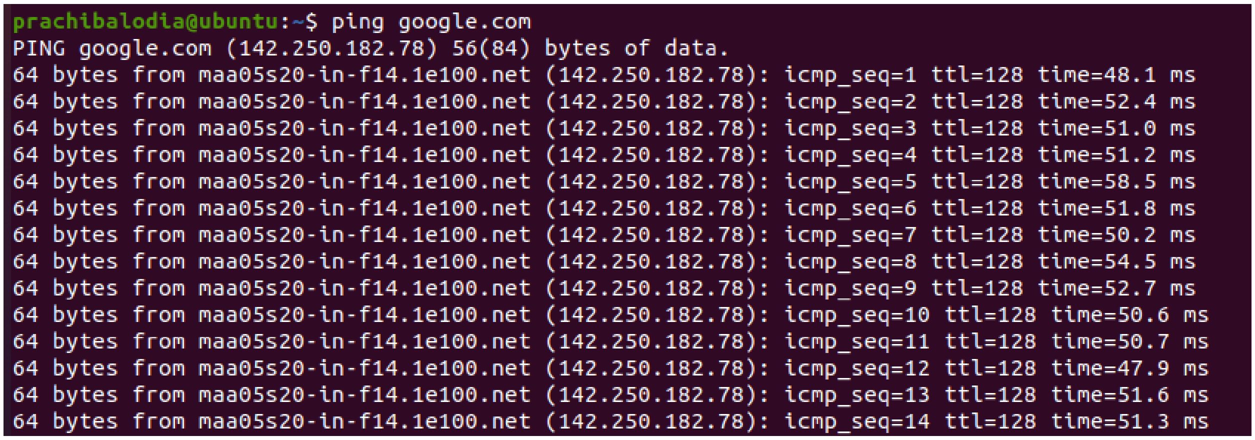
```
prachibalodia@ubuntu:~$ sudo apt-get upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
The following packages have been kept back:
  fwupd fwupd-signed gnome-shell-extension-desktop-icons libegl-mesa0 libfwupd2 libfwupdplugin1 libgbm1
  libgl1-mesa-dri libglapi-mesa libglx-mesa0 libxatracker2 mesa-vulkan-drivers ubuntu-advantage-tools
The following packages will be upgraded:
  alsa-ucm-conf alsa-utils apport apport-gtk apt apt-utils aspell avahi-autoipd avahi-daemon avahi-utils
  base-files bind9-dnsutils bind9-host bind9-libs bluez bluez-cups bluez-obexd dirmngr distro-info-data
  dnsmasq-base evince evince-common evolution-data-server evolution-data-server-common file-roller firefox
  fonts-noto-color-emoji fonts-opensymbol fprintd friendly-recovery gcc-10-base gdm3 gir1.2-gdkpixbuf-2.0
  gir1.2-gdm-1.0 gir1.2-goa-1.0 gir1.2-gst-plugins-base-1.0 gir1.2-javascriptcoregtk-4.0 gir1.2-mutter-6
  gir1.2-polkit-1.0 gir1.2-secret-1 gir1.2-webkit2-4.0 gjs gnome-control-center gnome-control-center-data
  gnome-control-center-faces gnome-disk-utility gnome-online-accounts gnome-settings-daemon
  gnome-settings-daemon-common gnome-shell gnome-shell-common gnome-shell-extension-appindicator gnupg
  gnupg-l10n gnupg-utils gpg gpg-agent gpg-wks-client gpg-wks-server gpgconf gpgsm gpgv gstreamer1.0-alsa
  gstreamer1.0-gl gstreamer1.0-gtk3 gstreamer1.0-plugins-base gstreamer1.0-plugins-base-apps
  gstreamer1.0-plugins-good gstreamer1.0-pulseaudio gstreamer1.0-x iio-sensor-proxy initramfs-tools
```

24. hostname — Use hostname to know your name in your host or network. Basically, it displays your hostname and IP address. Just typing “hostname” gives the output. Typing in “hostname -I” gives you your IP address in your network.



```
prachibalodia@ubuntu:~$ hostname
ubuntu
prachibalodia@ubuntu:~$ hostname -I
192.168.92.129
prachibalodia@ubuntu:~$ 
```

25. ping — We use ping to check your connection to a server.



```
prachibalodia@ubuntu:~$ ping google.com
PING google.com (142.250.182.78) 56(84) bytes of data.
64 bytes from maa05s20-in-f14.1e100.net (142.250.182.78): icmp_seq=1 ttl=128 time=48.1 ms
64 bytes from maa05s20-in-f14.1e100.net (142.250.182.78): icmp_seq=2 ttl=128 time=52.4 ms
64 bytes from maa05s20-in-f14.1e100.net (142.250.182.78): icmp_seq=3 ttl=128 time=51.0 ms
64 bytes from maa05s20-in-f14.1e100.net (142.250.182.78): icmp_seq=4 ttl=128 time=51.2 ms
64 bytes from maa05s20-in-f14.1e100.net (142.250.182.78): icmp_seq=5 ttl=128 time=58.5 ms
64 bytes from maa05s20-in-f14.1e100.net (142.250.182.78): icmp_seq=6 ttl=128 time=51.8 ms
64 bytes from maa05s20-in-f14.1e100.net (142.250.182.78): icmp_seq=7 ttl=128 time=50.2 ms
64 bytes from maa05s20-in-f14.1e100.net (142.250.182.78): icmp_seq=8 ttl=128 time=54.5 ms
64 bytes from maa05s20-in-f14.1e100.net (142.250.182.78): icmp_seq=9 ttl=128 time=52.7 ms
64 bytes from maa05s20-in-f14.1e100.net (142.250.182.78): icmp_seq=10 ttl=128 time=50.6 ms
64 bytes from maa05s20-in-f14.1e100.net (142.250.182.78): icmp_seq=11 ttl=128 time=50.7 ms
64 bytes from maa05s20-in-f14.1e100.net (142.250.182.78): icmp_seq=12 ttl=128 time=47.9 ms
64 bytes from maa05s20-in-f14.1e100.net (142.250.182.78): icmp_seq=13 ttl=128 time=51.6 ms
64 bytes from maa05s20-in-f14.1e100.net (142.250.182.78): icmp_seq=14 ttl=128 time=51.3 ms
```

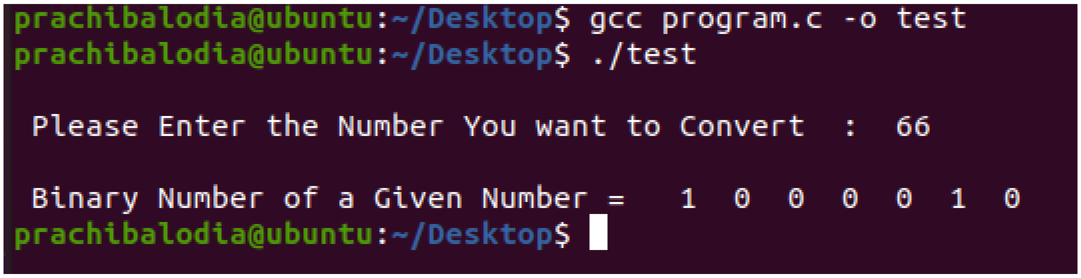
Implementation of C Programming with LINUX OS

1.(a) Program to convert decimal to binary.

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

int main()

{

   int a[10], number, i, j;

   printf("\n Please Enter the Number You want to Convert  :  ");

   scanf("%d", &number);

   for(i = 0; number > 0; i++)

   {

      a[i] = number % 2;

      number = number / 2;

   }

   printf("\n Binary Number of a Given Number =  ");

   for(j = i - 1; j >= 0; j--)  {

      printf(" %d ", a[j]);

   }

   printf("\n");

   return 0;

}
```

Algorithm:

- Divide the number by 2 through % (modulus operator) and store the remainder in array
- Divide the number by 2 through / (division operator)
- Repeat the step 2 until number is greater than 0



```
prachibalodia@ubuntu:~/Desktop$ gcc program.c -o test
prachibalodia@ubuntu:~/Desktop$ ./test

 Please Enter the Number You want to Convert  :  66

 Binary Number of a Given Number =   1  0  0  0  0  1  0
prachibalodia@ubuntu:~/Desktop$ 
```

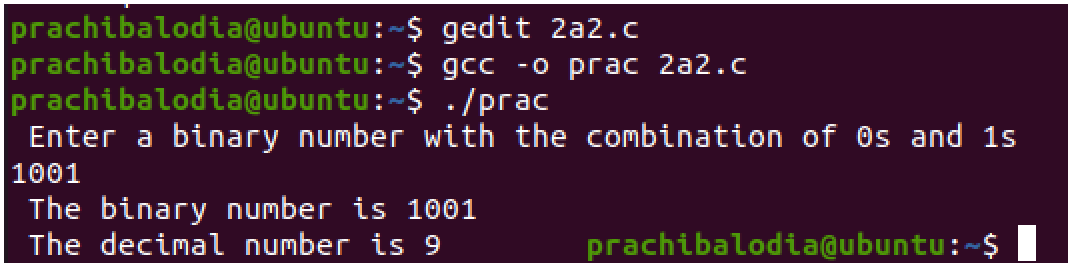
1.(b) Program to convert binary to decimal.

Code:

```
#include <stdio.h>
int main()
{

  int num, binary_num, decimal_num = 0, base = 1, rem;
  printf (" Enter a binary number with the combination of 0s and 1s \n");
  scanf (" %d", &num);

  binary_num = num;

  while ( num > 0)
  {
    rem = num % 10;
    decimal_num = decimal_num + rem * base;
    num = num / 10;
    base = base * 2;
  }

  printf ( " The binary number is %d \t", binary_num);
  printf (" \n The decimal number is %d \t", decimal_num);
  return 0;
}
```

Algorithm:

- Take a binary number as the input.
- Divide the number by 10 and store the remainder into variable rem.
- decimal_num = decimal_num + rem * base;
- Initially, the decimal_num is 0, and the base is 1, where the rem variable stores the remainder of the number.
- Divide the quotient of the original number by 10.
- Multiply the base by 2.
- Print the decimal of the binary number.



```
prachibalodia@ubuntu:~$ gedit 2a2.c
prachibalodia@ubuntu:~$ gcc -o prac 2a2.c
prachibalodia@ubuntu:~$ ./prac
 Enter a binary number with the combination of 0s and 1s
1001
 The binary number is 1001
 The decimal number is 9        prachibalodia@ubuntu:~$ 
```

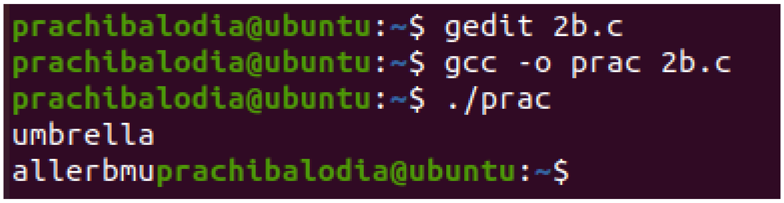
2. **C Program to print the Reverse of a string using user defined function.**

Code:

```
#include<stdio.h>

#include<string.h>

void revstr(char *s)

{

for(int i=strlen(s);i>=0;i--)

   {

      printf("%c",s[i]);

   }

}


int main()

{

   char s[20];

   scanf("%s",s);
   revstr(s);
   return 0;
}
```

Algorithm:

- Input a string
- Reverse the string using revstr function



```
prachibalodia@ubuntu:~$ gedit 2b.c
prachibalodia@ubuntu:~$ gcc -o prac 2b.c
prachibalodia@ubuntu:~$ ./prac
umbrella
allerbmuprachibalodia@ubuntu:~$
```

3. Prime number between two interval.

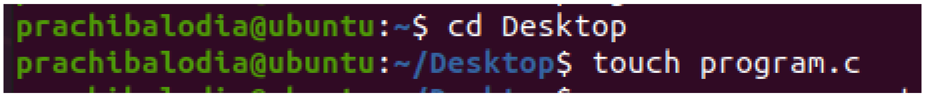
Code:

```
#include <stdio.h>
int main() {
  int low, high, i, flag;
  printf("Enter two numbers(intervals): ");
  scanf("%d %d", &low, &high);
  printf("Prime numbers between %d and %d are: ", low, high);
  while (low < high) {
    flag = 0;
    if (low <= 1) {
      ++low;
      continue;
    for (i = 2; i <= low / 2; ++i) {
    if (low % i == 0) {
        flag = 1;
```

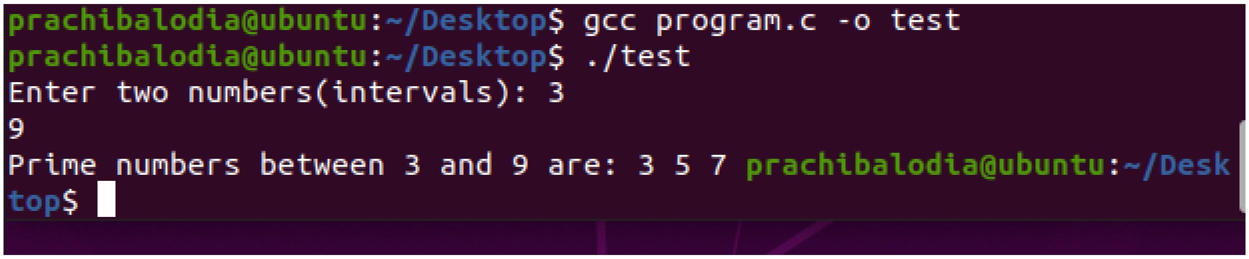
```
          break;
        }
      }

      if (flag == 0)
        printf("%d ", low);
      ++low;
    }
  return 0;
}
```

Algorithm:

- Input the range
- Check whether the number is prime or not
- Increment the number to check the next number
- The process is repeated till the range value



```
prachibalodia@ubuntu:~$ cd Desktop
prachibalodia@ubuntu:~/Desktop$ touch program.c
```



```
prachibalodia@ubuntu:~/Desktop$ gcc program.c -o test
prachibalodia@ubuntu:~/Desktop$ ./test
Enter two numbers(intervals): 3
9
Prime numbers between 3 and 9 are: 3 5 7 prachibalodia@ubuntu:~/Desk
top$ 
```

4. C Program to count the number of Vowels and Consonants and so on.

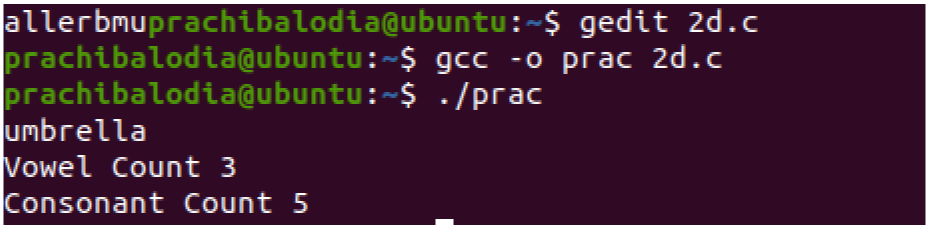
Code:

```
#include<stdio.h>

#include <string.h>

int main()

{

   char s[20];
```

```
    scanf("%s",s);


int vc=0,cc=0;
    for(int i=0;i<strlen(s);i++)
    {
        if(s[i]=='a'||s[i]=='e'||s[i]=='i'||s[i]=='o'||s[i]=='u')
        {
            vc++;


        }
        else
        {
            cc++;
        }
    }
 printf("Vowel Count %d\n",vc);
 printf("Consonant Count %d\n",cc);
    return 0;
}
```

Algorithm:

- Input the string
- The characters are compared with the vowels and vowel count is increased if it matches, or else the consonant count is increased
- The respective count is printed



```
allerbmuprachibalodia@ubuntu:~$ gedit 2d.c
prachibalodia@ubuntu:~$ gcc -o prac 2d.c
prachibalodia@ubuntu:~$ ./prac
umbrella
Vowel Count 3
Consonant Count 5
```

5. C Program to store information of Students using Structures.

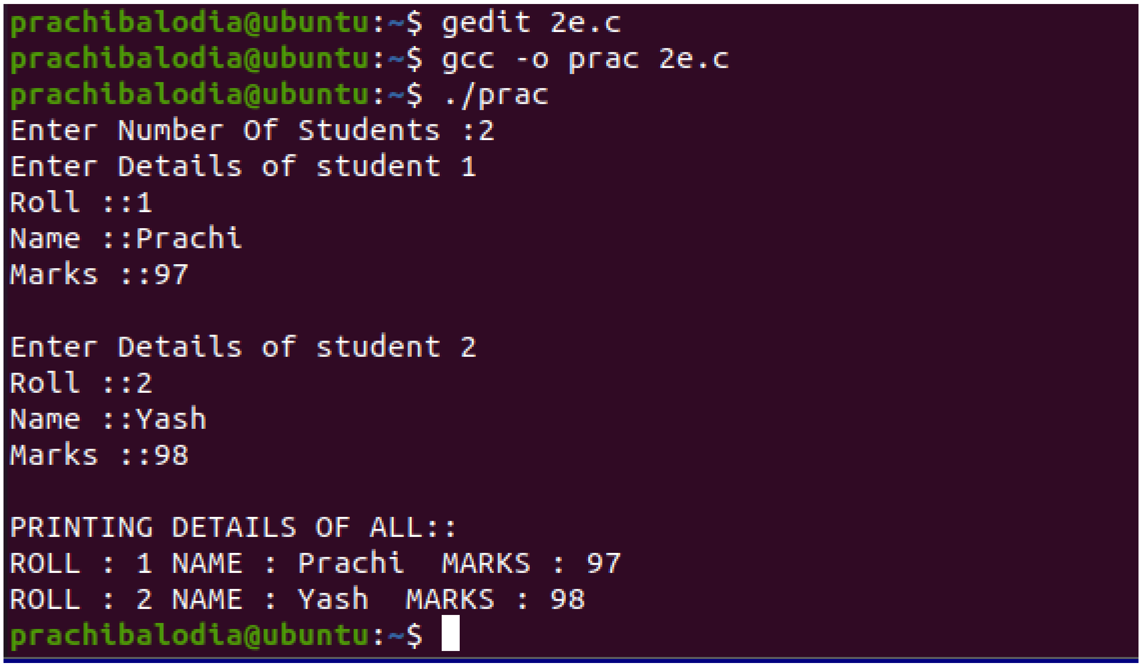
Code:

```
#include <stdio.h>

struct student
{
    int roll;
    char name[10];
    int marks;
};

int main()
{
    int x;
    printf("Enter Number Of Students :");
    scanf("%d", &x);

    struct student s[x];

    for (int i = 0; i < x; i++)
    {
        printf("Enter Details of student %d\n", i + 1);
        printf("Roll ::");
        scanf("%d", &s[i].roll);
        printf("Name ::");
        scanf("%s", s[i].name);
        printf("Marks ::");
        scanf("%d", &s[i].marks);
        printf("\n");
    }
```

```
  printf("PRINTING DETAILS OF ALL::\n");
  for (int i = 0; i < x; i++)
  {
    printf("ROLL : %d ", s[i].roll);
    printf("NAME : %s  ", s[i].name);
    printf("MARKS : %d  \n", s[i].marks);
  }
 return 0;
```

Algorithm:

- Get the number of Students whose details are to be stored. Here we are taking 5 students for simplicity.
- Create a variable of Student structure to access the records.
- Get the data of x students and store it in student's fields with the help of dot (.) operator
- After all the data is stored, print the records of each students using the dot (.) operator and loop



```
prachibalodia@ubuntu:~$ gedit 2e.c
prachibalodia@ubuntu:~$ gcc -o prac 2e.c
prachibalodia@ubuntu:~$ ./prac
Enter Number Of Students :2
Enter Details of student 1
Roll ::1
Name ::Prachi
Marks ::97

Enter Details of student 2
Roll ::2
Name ::Yash
Marks ::98

PRINTING DETAILS OF ALL::
ROLL : 1 NAME : Prachi  MARKS : 97
ROLL : 2 NAME : Yash  MARKS : 98
prachibalodia@ubuntu:~$ 
```

```
}
```

Process and Thread Management

1. Create Parent and Child process using fork( ) and exec() system call

a. Checking the Process Identifier

b. Assigning new task to child

c. Providing the path name and program name to exec()

d. Synchronizing Parent and child process using wait()

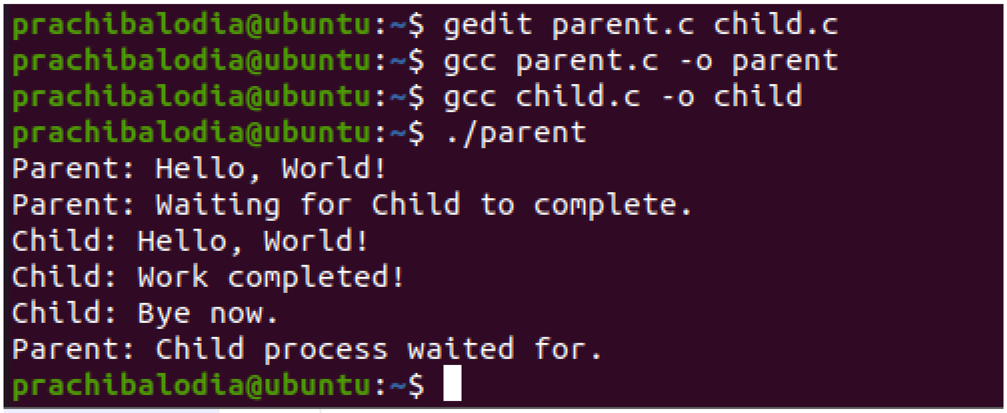
Code:

Parent program:

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

int main (int argc, char **argv)
{
   int i = 0;
   long sum;
   int pid;
   int status, ret;
   char *myargs [] = { NULL };
   char *myenv [] = { NULL };
   printf ("Parent: Hello, World!\n");
   pid = fork ();
   if (pid == 0) {
      execve ("child", myargs, myenv);
   }
   printf ("Parent: Waiting for Child to complete.\n");
   if ((ret = waitpid (pid, &status, 0)) == -1)
       printf ("parent:error\n");
   if (ret == pid)
      printf ("Parent: Child process waited for.\n");
}
```

Child program:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define A 500
#define B 600
#define C 700
   int main (int argc, char **argv)
{
   int i, j;
   long sum;
   printf ("Child: Hello, World!\n");
   for (j = 0; j < 30; j++ ) {
      for (i =0; i < 900000; i++) {
         sum = A * i + B * i * i + C;
         sum %= 543;
      }
   }
   printf ("Child: Work completed!\n");
   printf ("Child: Bye now.\n");
   exit (0);
}
```



```
prachibalodia@ubuntu:~$ gedit parent.c child.c
prachibalodia@ubuntu:~$ gcc parent.c -o parent
prachibalodia@ubuntu:~$ gcc child.c -o child
prachibalodia@ubuntu:~$ ./parent
Parent: Hello, World!
Parent: Waiting for Child to complete.
Child: Hello, World!
Child: Work completed!
Child: Bye now.
Parent: Child process waited for.
prachibalodia@ubuntu:~$ 
```

2. Write a program to create a thread and perform the following

Create a thread runner function

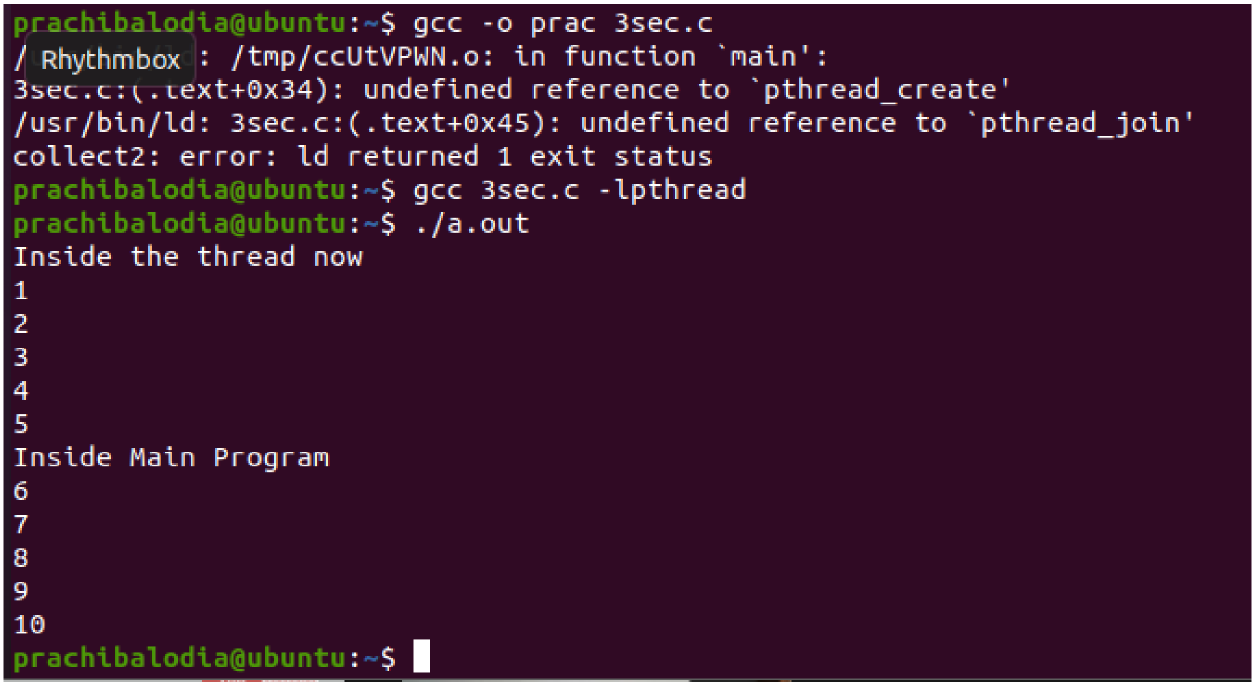
Set the thread attributes

Join the parent and thread

Wait for the thread to complete

Code:

```
 #include<stdio.h>

#include<unistd.h>

#include<pthread.h>


void *thread_function(void *arg);

int i,n,j;

int main(){

pthread_t t; //thread declearation

pthread_create(&t, NULL , thread_function,NULL); // thread is created

pthread_join ( t,NULL); // process waits for thread to finish.

printf("Inside Main Program\n");

for( j=6;j<11;j++){

        printf("%d\n",j);

        sleep(1);
}
return 0;
}

void *thread_function(void *arg){
printf("Inside the thread now \n");
for(i=1;i<6;i++){
    printf("%d\n",i);
    sleep(1);
}
}
```



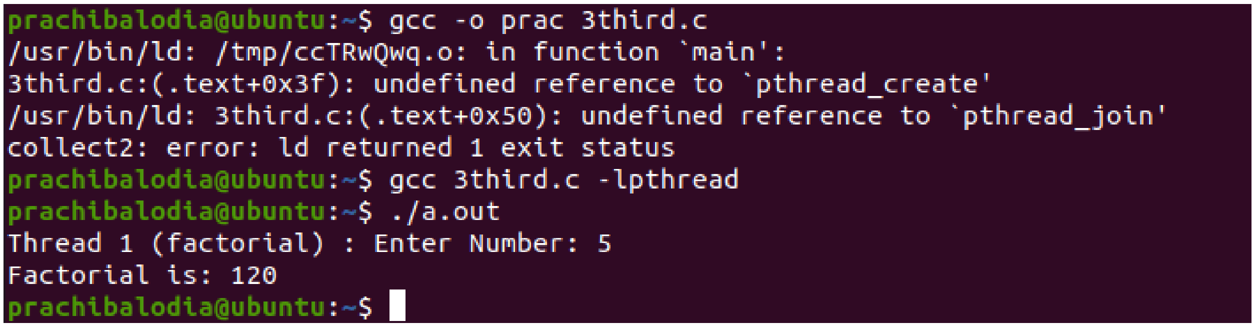
```
prachibalodia@ubuntu:~$ gcc -o prac 3sec.c
/ Rhythmbox : /tmp/ccUtVPWN.o: in function `main':
3sec.c:(.text+0x34): undefined reference to `pthread_create'
/usr/bin/ld: 3sec.c:(.text+0x45): undefined reference to `pthread_join'
collect2: error: ld returned 1 exit status
prachibalodia@ubuntu:~$ gcc 3sec.c -lpthread
prachibalodia@ubuntu:~$ ./a.out
Inside the thread now
1
2
3
4
5
Inside Main Program
6
7
8
9
10
prachibalodia@ubuntu:~$ 
```

3. Write a program to create a Thread to find the Factorial of a natural number ‘n’.

Code:

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

```
void *factorial(void *p);
int fact(int n);
int main(){
pthread_t tid1;
pthread_attr_t attr; // set of thread attributes
pthread_attr_init(&attr);
pthread_create(&tid1,&attr,factorial,NULL);
pthread_join(tid1,NULL);

}
int fact(int n){
if(n==0 || n==1)
return 1;
else
return n*fact(n-1);
}
void *factorial(void *p){
int i,num1;
printf("Thread 1 (factorial) : ");
printf("Enter Number: ");
scanf("%d",&num1);
printf("Factorial is: %d\n",fact(num1));
pthread_exit(0);
}
```



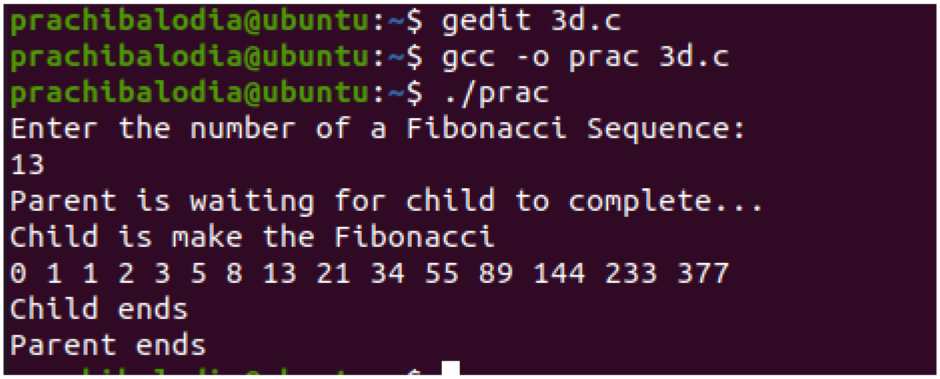
```
prachibalodia@ubuntu:~$ gcc -o prac 3third.c
/usr/bin/ld: /tmp/ccTRwQwq.o: in function `main':
3third.c:(.text+0x3f): undefined reference to `pthread_create'
/usr/bin/ld: 3third.c:(.text+0x50): undefined reference to `pthread_join'
collect2: error: ld returned 1 exit status
prachibalodia@ubuntu:~$ gcc 3third.c -lpthread
prachibalodia@ubuntu:~$ ./a.out
Thread 1 (factorial) : Enter Number: 5
Factorial is: 120
prachibalodia@ubuntu:~$ 
```

4. Write a program to create a Process to generate the Fibonacci sequence and display the same.

Code:

```
#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

#include <sys/wait.h>

int main()

{

int a=0, b=1, n=a+b,i;

printf("Enter the number of a Fibonacci Sequence:\n");

scanf("%d", &i);


pid_t pid = fork();

if (pid == 0)

{

   printf("Child is make the Fibonacci\n");

   printf("0 %d ",n);

   while (i>0) {

      n=a+b;

      printf("%d ", n);
```

```
        a=b;
        b=n;
        i--;
        if (i == 0) {
            printf("\nChild ends\n");
        }
    }
}
    else
    {
        printf("Parent is waiting for child to complete...\n");
        waitpid(pid, NULL, 0);
        printf("Parent ends\n");
    }
    return 0;
}
```



```
prachibalodia@ubuntu:~$ gedit 3d.c
prachibalodia@ubuntu:~$ gcc -o prac 3d.c
prachibalodia@ubuntu:~$ ./prac
Enter the number of a Fibonacci Sequence:
13
Parent is waiting for child to complete...
Child is make the Fibonacci
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
Child ends
Parent ends
```

Operating Systems

Lab DA-2

1. C program for FCFS Scheduling (Non-Pre-emptive) in Linux.

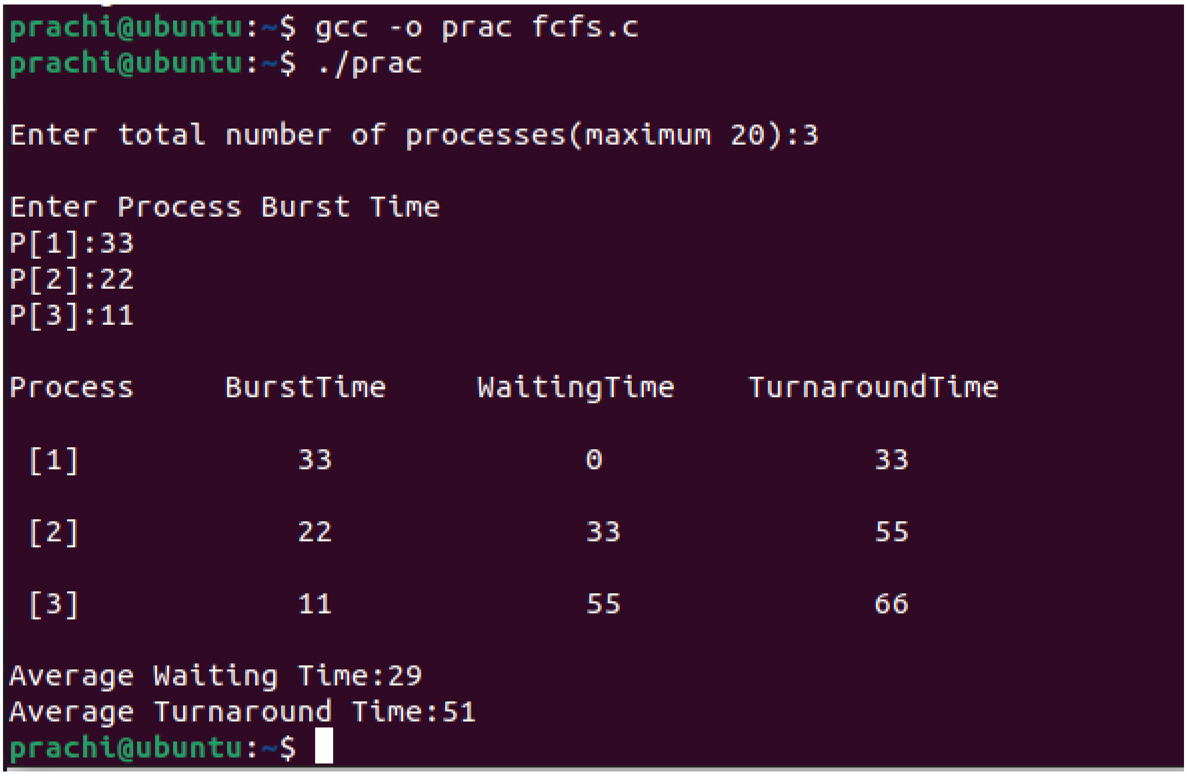
Algorithm:

- Input the processes along with their burst time (bt).
- Find waiting time (wt) for all processes.
- As first process that comes need not to wait so waiting time for process 1 will be 0.
- For waiting time for all other processes i.e. for process
  i -> wt[i] = bt[i-1] + wt[i-1] .
- For turnaround time = waiting_time + burst_time for all processes.
- For average waiting time = total_waiting_time / no_of_processes.
- Similarly, for average turnaround time = total_turn_around_time / no_of_processes.

Code:

```
#include<stdio.h>
 int main()
 {
   int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;
   printf("\nEnter total number of processes(maximum 20):");
   scanf("%d",&n);
   printf("\nEnter Process Burst Time\n");
   for(i=0;i<n;i++)
   {
      printf("P[%d]:",i+1);
      scanf("%d",&bt[i]);
   }
    wt[0]=0;
    for(i=1;i<n;i++)
   {
      wt[i]=0;
      for(j=0;j<i;j++)
          wt[i]+=bt[j];
   }
    printf("\nProcess     BurstTime     WaitingTime    TurnaroundTime\n");
    for(i=0;i<n;i++)
   {
      tat[i]=bt[i]+wt[i];
      avwt+=wt[i];
      avtat+=tat[i];
```

```
        printf("\n [%d]               %d               %d               %d\n
",i+1,bt[i],wt[i],tat[i]);
    }
     avwt/=i;
    avtat/=i;
    printf("\nAverage Waiting Time:%d",avwt);
    printf("\nAverage Turnaround Time:%d\n",avtat);
     return 0;
}
```



```
prachi@ubuntu:~$ gcc -o prac fcfs.c
prachi@ubuntu:~$ ./prac

Enter total number of processes(maximum 20):3

Enter Process Burst Time
P[1]:33
P[2]:22
P[3]:11

Process     BurstTime     WaitingTime    TurnaroundTime

 [1]            33              0              33

 [2]            22              33             55

 [3]            11              55             66

Average Waiting Time:29
Average Turnaround Time:51
prachi@ubuntu:~$ 
```

2. C program for SJF Scheduling in Linux.

(i). Non-Pre-emptive

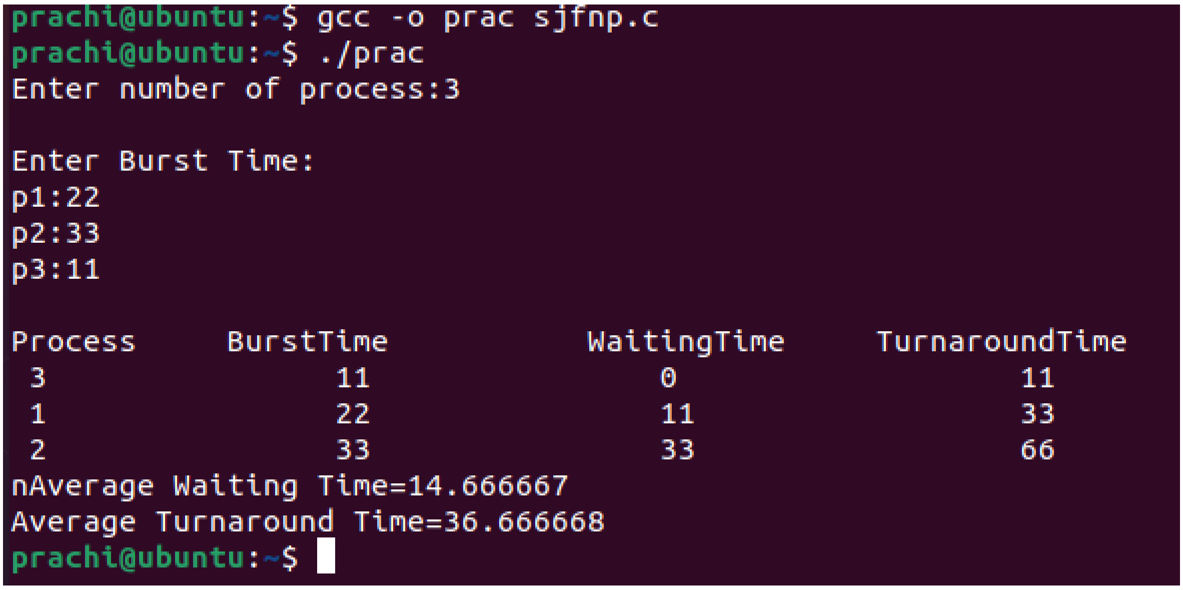
Algorithm:

- Sort all the processes according to their arrival time.
- The process which arrives at 0 is completed first.
- Now, the burst time of the processes is compared which has arrived meanwhile the completion of process with arrival time=0.
- The process with shortest burst time is executed and the same procedure is repeated until all the processes is executed.

Code:

```
#include<stdio.h>

 int main()

{

   int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;

   float avg_wt,avg_tat;

   printf("Enter number of process:");

   scanf("%d",&n);

    printf("\nEnter Burst Time:\n");

   for(i=0;i<n;i++)

   {

      printf("p%d:",i+1);

      scanf("%d",&bt[i]);

      p[i]=i+1;

   }

   for(i=0;i<n;i++)

   {

      pos=i;

      for(j=i+1;j<n;j++)

      {

         if(bt[j]<bt[pos])

            pos=j;
    }
      temp=bt[i];
    bt[i]=bt[pos];
    bt[pos]=temp;
      temp=p[i];
    p[i]=p[pos];
    p[pos]=temp;
}
   wt[0]=0;
     for(i=1;i<n;i++)
{
    wt[i]=0;
    for(j=0;j<i;j++)
        wt[i]+=bt[j];
      total+=wt[i];
}
  avg_wt=(float)total/n;
total=0;

printf("\nProcess         BurstTime     WaitingTime     TurnaroundTime");
for(i=0;i<n;i++)
{
    tat[i]=bt[i]+wt[i];
    total+=tat[i];
    printf("\n %d                     %d                      %d
         %d",p[i],bt[i],wt[i],tat[i]);
}
  avg_tat=(float)total/n;
printf("\nnAverage Waiting Time=%f",avg_wt);
printf("\nAverage Turnaround Time=%f\n",avg_tat);
```

}



```
prachi@ubuntu:~$ gcc -o prac sjfnp.c
prachi@ubuntu:~$ ./prac
Enter number of process:3

Enter Burst Time:
p1:22
p2:33
p3:11

Process      BurstTime            WaitingTime      TurnaroundTime
 3                 11                 0                   11
 1                 22                 11                  33
 2                 33                 33                  66
nAverage Waiting Time=14.666667
Average Turnaround Time=36.666668
prachi@ubuntu:~$ 
```

(ii). Pre-emptive

Algorithm:

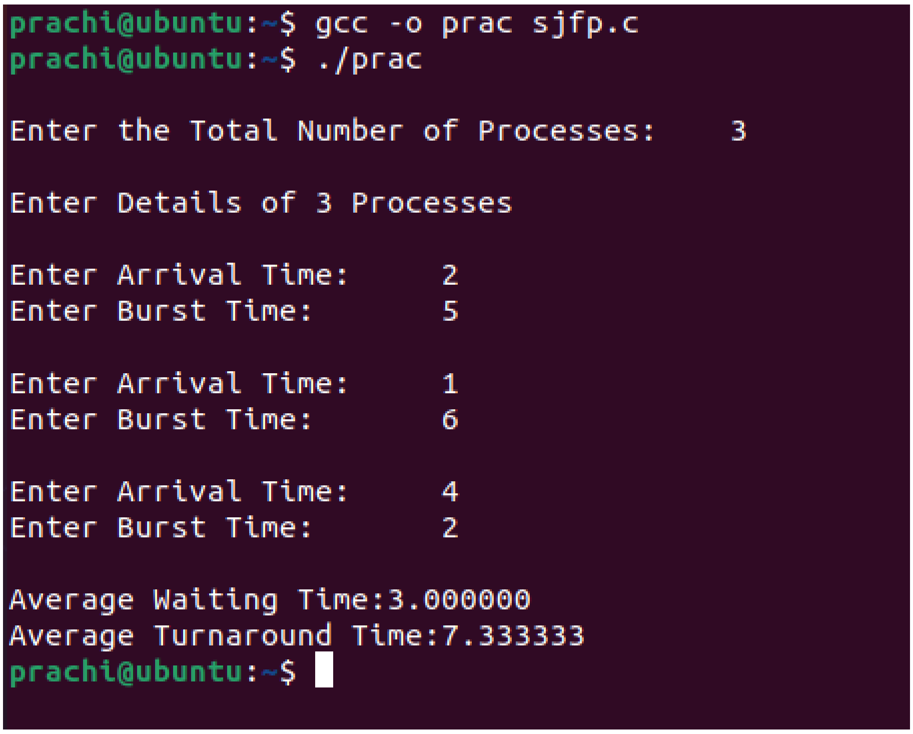
- At first, jobs are put into the ready queue as they come.
- A process with shortest burst time begins execution.
- If a process with even a shorter burst time arrives, the current process is removed or pre-empted from execution, and the shorter job is allocated CPU cycle.
- The same procedure is repeated until all the processes are executed.

Code:

```
#include <stdio.h>

 int main()

{

    int at[10], bt[10], temp[10];

    int i, smallest, count = 0, time, n;

    double wt, tat, end,total;

    float avgwt, avgtat;

    printf("\nEnter the Total Number of Processes:   ");

    scanf("%d", &n);

    printf("\nEnter Details of %d Processes\n", n);

    for(i = 0; i < n; i++)
```

```
{
    printf("\nEnter Arrival Time:    ");
    scanf("%d", &at[i]);
    printf("Enter Burst Time:        ");
    scanf("%d", &bt[i]);
    temp[i] = bt[i];
}
bt[9] = 9999;
for(time = 0; count != n; time++)
{
    smallest = 9;
    for(i = 0; i < n; i++)
    {
        if(at[i] <= time && bt[i] < bt[smallest] && bt[i] > 0)
        {
            smallest = i;
        }
    }
    bt[smallest]--;
    if(bt[smallest] == 0)
    {
        count++;
        end = time + 1;
        wt = wt + end - at[smallest] - temp[smallest];
        tat = tat + end - at[smallest];
    }
}
avgwt = wt / n;
avgtat = tat / n;
```

```
    printf("\nAverage Waiting Time:%lf \n", avgwt);

    printf("Average Turnaround Time:%lf \n", avgtat);

    return 0;

}
```



```
prachi@ubuntu:~$ gcc -o prac sjfp.c
prachi@ubuntu:~$ ./prac

Enter the Total Number of Processes:    3

Enter Details of 3 Processes

Enter Arrival Time:     2
Enter Burst Time:       5

Enter Arrival Time:     1
Enter Burst Time:       6

Enter Arrival Time:     4
Enter Burst Time:       2

Average Waiting Time:3.000000
Average Turnaround Time:7.333333
prachi@ubuntu:~$ 
```

Prachi Balodia

20BDS0177

OS Lab DA-3

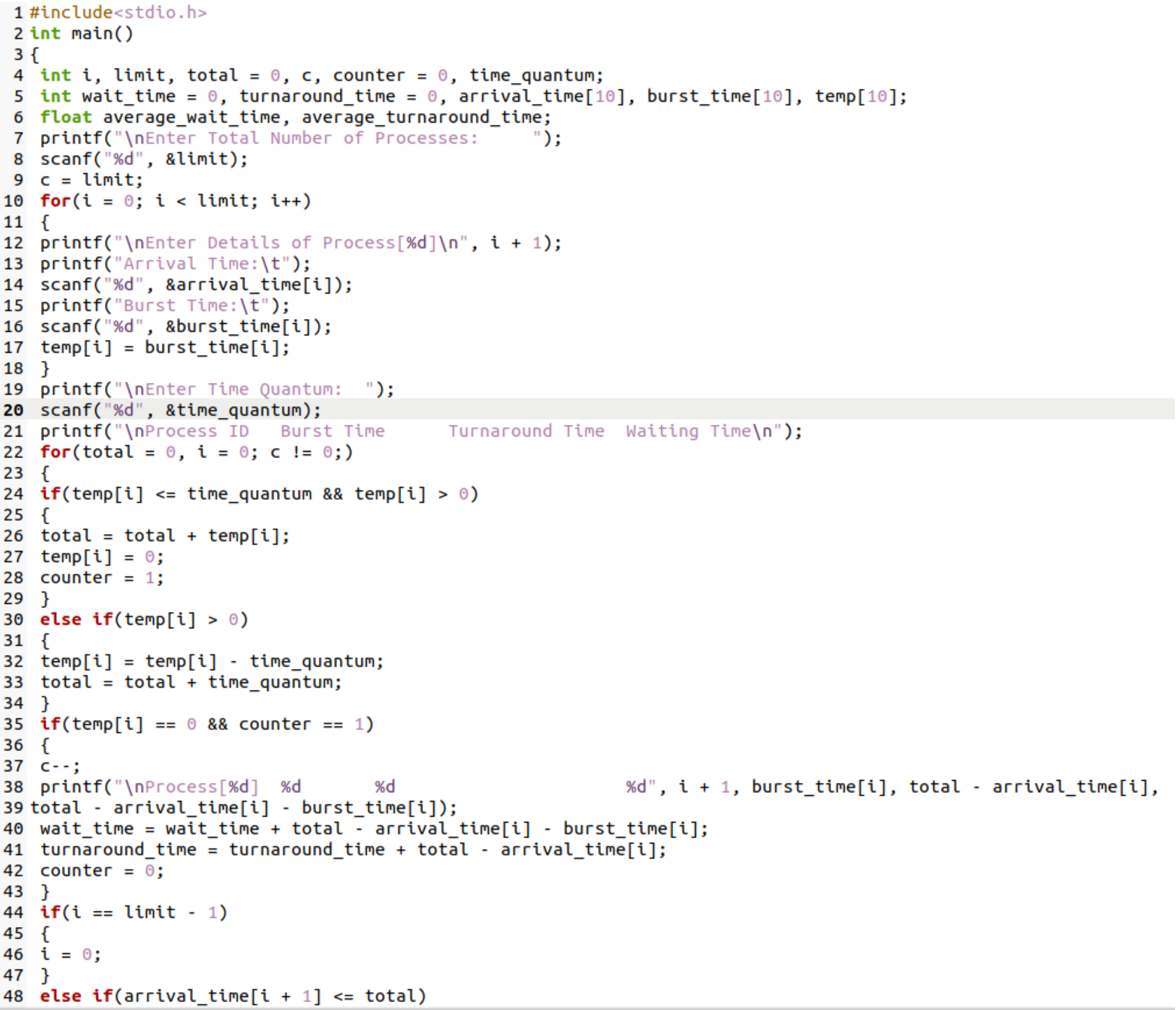
1. Round Robin Scheduling (Pre-emptive)

Algorithm:

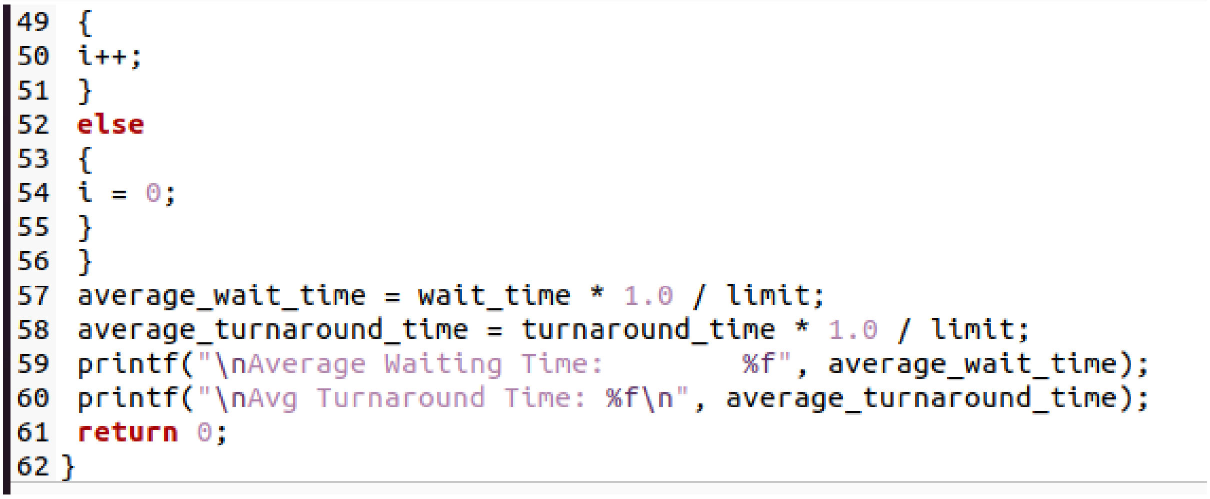
- Round robin is a pre-emptive algorithm.
- In Round-robin scheduling, each ready task runs turn by turn only in a cyclic queue for a limited time slice.

Code:

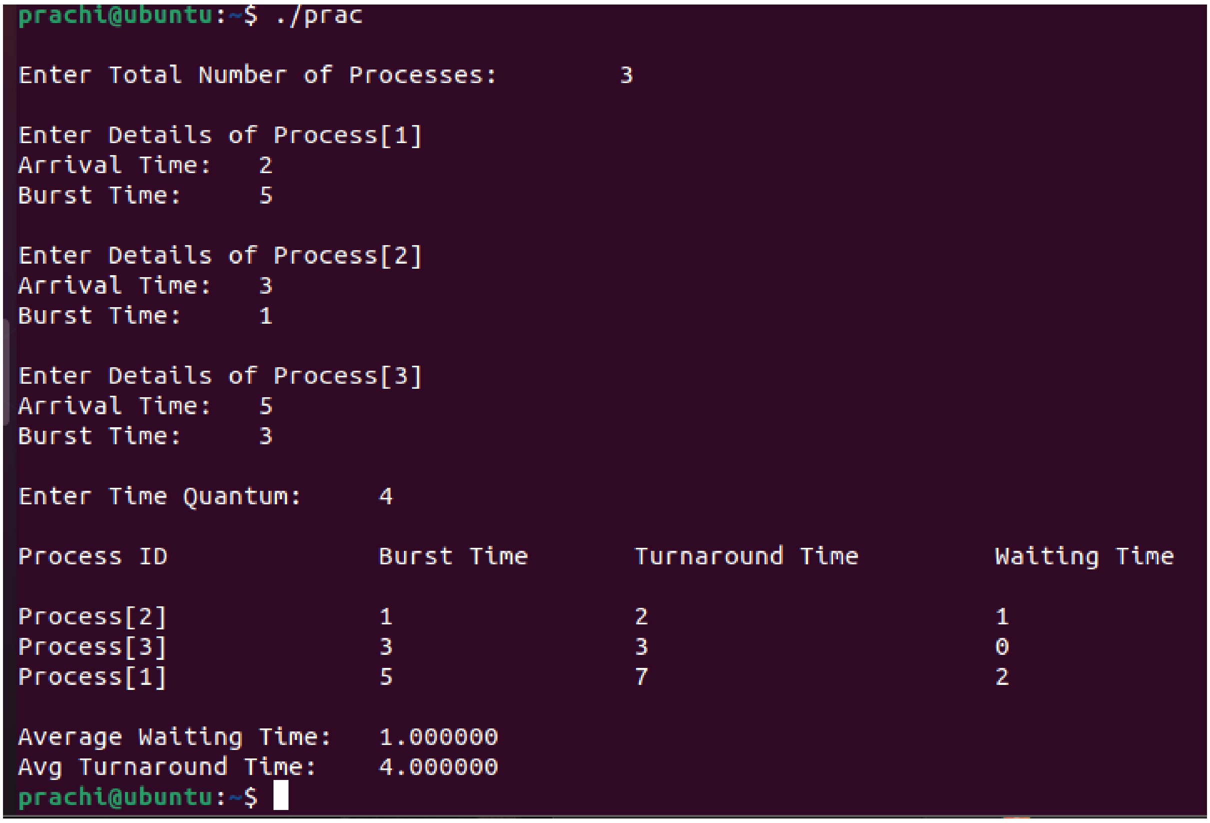
```
#include<stdio.h>
int main()
{
 int i, limit, total = 0, c, counter = 0, time_quantum;
 int wait_time = 0, turnaround_time = 0, arrival_time[10], burst_time[10], temp[10];
 float average_wait_time, average_turnaround_time;
 printf("\nEnter Total Number of Processes: ");
 scanf("%d", &limit);
 c = limit;
 for(i = 0; i < limit; i++)
 {
 printf("\nEnter Details of Process[%d]\n", i + 1);
 printf("Arrival Time:\t");
 scanf("%d", &arrival_time[i]);
 printf("Burst Time:\t");
 scanf("%d", &burst_time[i]);
 temp[i] = burst_time[i];
 }
 printf("\nEnter Time Quantum:        ");
 scanf("%d", &time_quantum);
 printf("\nProcess ID   Burst Time      Turnaround Time         Waiting Time\n");
 for(total = 0, i = 0; c != 0;)
 {
 if(temp[i] <= time_quantum && temp[i] > 0)
 {
 total = total + temp[i];
 temp[i] = 0;
 counter = 1;
 }
 else if(temp[i] > 0)
 {
 temp[i] = temp[i] - time_quantum;
 total = total + time_quantum;
 }
 if(temp[i] == 0 && counter == 1)
 {
 c--;
 printf("\nProcess[%d]          %d        %d                              %d", i + 1, burst_time[i],
total - arrival_time[i],
total - arrival_time[i] - burst_time[i]);
 wait_time = wait_time + total - arrival_time[i] - burst_time[i];
 turnaround_time = turnaround_time + total - arrival_time[i];
 counter = 0;
 }
 if(i == limit - 1)
 {
 i = 0;
 }
 else if(arrival_time[i + 1] <= total)
 {
 i++;
 }
 else
 {
 i = 0;
 }
 }
 average_wait_time = wait_time * 1.0 / limit;
 average_turnaround_time = turnaround_time * 1.0 / limit;
 printf("\nAverage Waiting Time:      %f", average_wait_time);
 printf("\nAvg Turnaround Time:       %f\n", average_turnaround_time);
 return 0;
}
```



```c
#include<stdio.h>
int main()
{
int i, limit, total = 0, c, counter = 0, time_quantum;
int wait_time = 0, turnaround_time = 0, arrival_time[10], burst_time[10], temp[10];
float average_wait_time, average_turnaround_time;
printf("\nEnter Total Number of Processes:     ");
scanf("%d", &limit);
c = limit;
for(i = 0; i < limit; i++)
{
printf("\nEnter Details of Process[%d]\n", i + 1);
printf("Arrival Time:\t");
scanf("%d", &arrival_time[i]);
printf("Burst Time:\t");
scanf("%d", &burst_time[i]);
temp[i] = burst_time[i];
}
printf("\nEnter Time Quantum:  ");
scanf("%d", &time_quantum);
printf("\nProcess ID   Burst Time       Turnaround Time  Waiting Time\n");
for(total = 0, i = 0; c != 0;)
{
if(temp[i] <= time_quantum && temp[i] > 0)
{
total = total + temp[i];
temp[i] = 0;
counter = 1;
}
else if(temp[i] > 0)
{
temp[i] = temp[i] - time_quantum;
total = total + time_quantum;
}
if(temp[i] == 0 && counter == 1)
{
c--;
printf("\nProcess[%d]  %d       %d                    %d", i + 1, burst_time[i], total - arrival_time[i],
total - arrival_time[i] - burst_time[i]);
wait_time = wait_time + total - arrival_time[i] - burst_time[i];
turnaround_time = turnaround_time + total - arrival_time[i];
counter = 0;
}
if(i == limit - 1)
{
i = 0;
}
else if(arrival_time[i + 1] <= total)
```



```c
{
i++;
}
else
{
i = 0;
}
}
average_wait_time = wait_time * 1.0 / limit;
average_turnaround_time = turnaround_time * 1.0 / limit;
printf("\nAverage Waiting Time:        %f", average_wait_time);
printf("\nAvg Turnaround Time: %f\n", average_turnaround_time);
return 0;
}
```



```
prachi@ubuntu:~$ ./prac

Enter Total Number of Processes:        3

Enter Details of Process[1]
Arrival Time:   2
Burst Time:     5

Enter Details of Process[2]
Arrival Time:   3
Burst Time:     1

Enter Details of Process[3]
Arrival Time:   5
Burst Time:     3

Enter Time Quantum:     4

Process ID              Burst Time       Turnaround Time         Waiting Time

Process[2]              1                2                       1
Process[3]              3                3                       0
Process[1]              5                7                       2

Average Waiting Time:   1.000000
Avg Turnaround Time:    4.000000
prachi@ubuntu:~$ 
```

2. Priority Scheduling

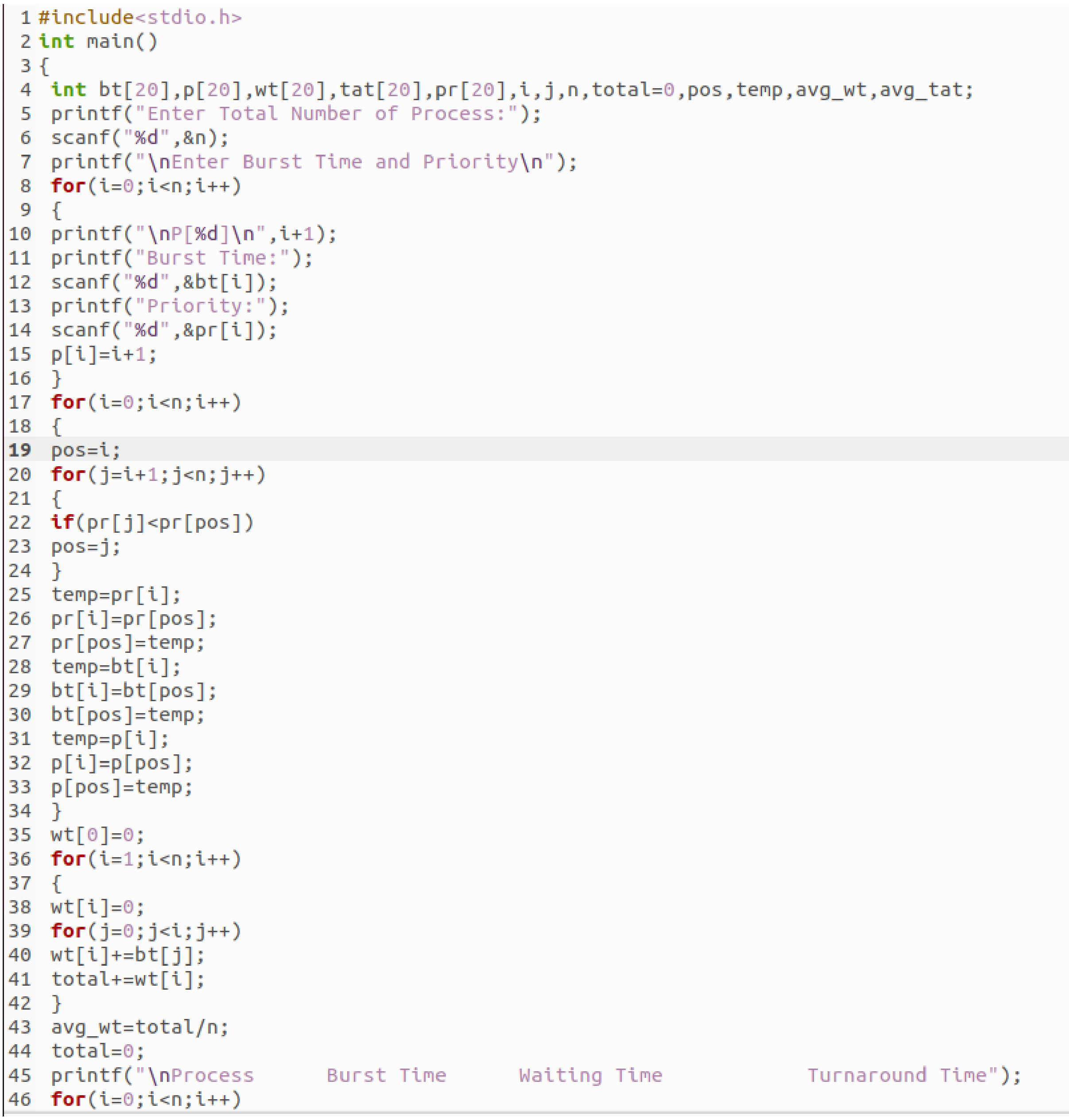
(i) Non-Pre-emptive

Algorithm:

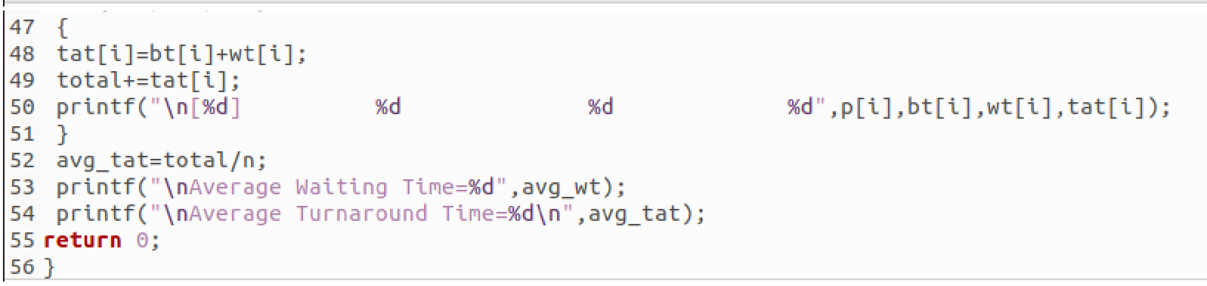
1. In the non pre-emptive priority scheduling algorithm, the priority of the process is compared with the priority of other processes which has arrived and are in the ready queue.
2. Now, the process which has the highest priority will be assigned to the CPU and will be executed first.
3. Now, the job scheduled will run till the completion.

Code:

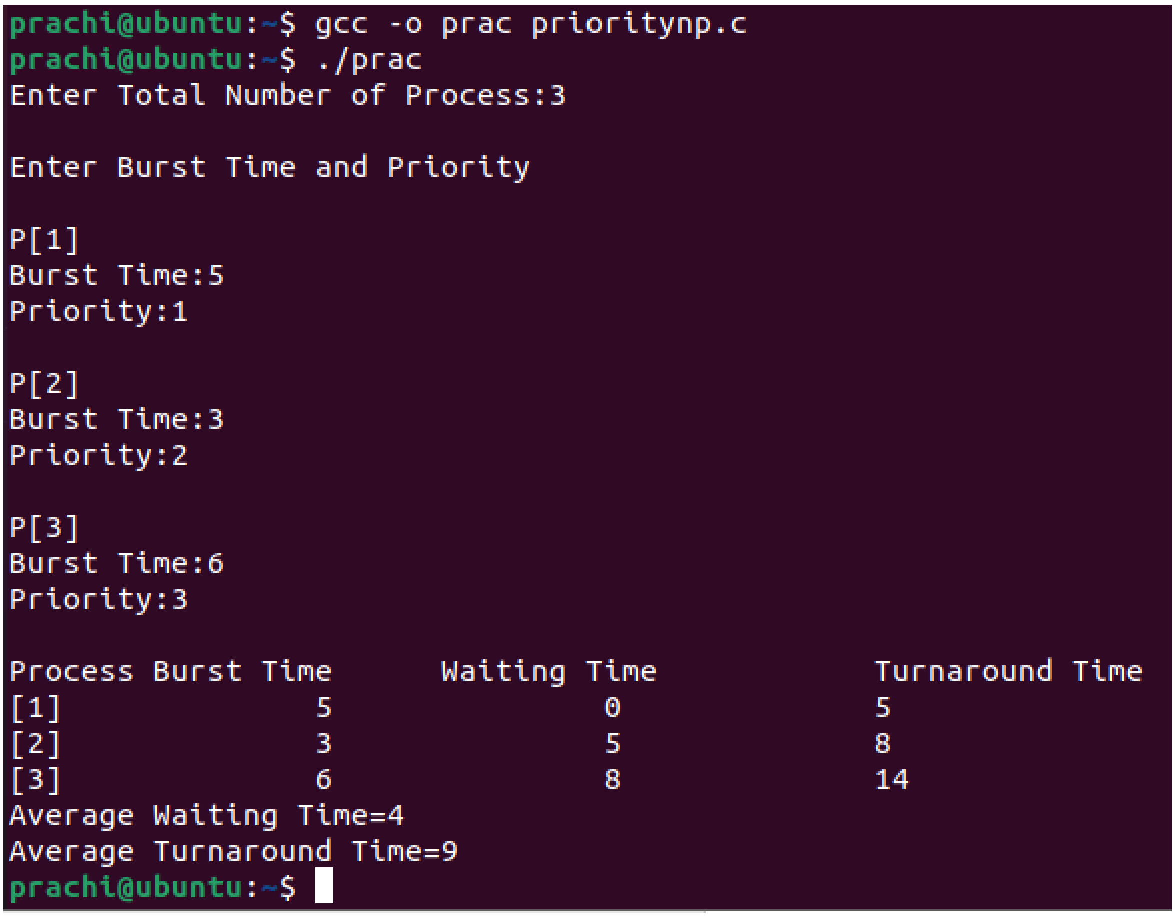
```
#include<stdio.h>
int main()
{
 int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_tat;
 printf("Enter Total Number of Process:");
 scanf("%d",&n);
 printf("\nEnter Burst Time and Priority\n");
 for(i=0;i<n;i++)
 {
 printf("\nP[%d]\n",i+1);
 printf("Burst Time:");
 scanf("%d",&bt[i]);
 printf("Priority:");
 scanf("%d",&pr[i]);
 p[i]=i+1;
 }
 for(i=0;i<n;i++)
 {
 pos=i;
 for(j=i+1;j<n;j++)
 {
 if(pr[j]<pr[pos])
 pos=j;
 }
 temp=pr[i];
 pr[i]=pr[pos];
 pr[pos]=temp;
 temp=bt[i];
 bt[i]=bt[pos];
 bt[pos]=temp;
 temp=p[i];
 p[i]=p[pos];
 p[pos]=temp;
 }
 wt[0]=0;
 for(i=1;i<n;i++)
 {
 wt[i]=0;
 for(j=0;j<i;j++)
 wt[i]+=bt[j];
 total+=wt[i];
 }
 avg_wt=total/n;
 total=0;
 printf("\nProcess       Burst Time      Waiting Time            Turnaround 
Time");
 for(i=0;i<n;i++)
 {
 tat[i]=bt[i]+wt[i];
 total+=tat[i];
 printf("\n[%d]           %d                %d                 %d",p[i],bt[i],wt[i],tat[i]);
 }
 avg_tat=total/n;
 printf("\nAverage Waiting Time=%d",avg_wt);
 printf("\nAverage Turnaround Time=%d\n",avg_tat);
return 0;
}
```



```
#include<stdio.h>
int main()
{
 int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_tat;
 printf("Enter Total Number of Process:");
 scanf("%d",&n);
 printf("\nEnter Burst Time and Priority\n");
 for(i=0;i<n;i++)
 {
 printf("\nP[%d]\n",i+1);
 printf("Burst Time:");
 scanf("%d",&bt[i]);
 printf("Priority:");
 scanf("%d",&pr[i]);
 p[i]=i+1;
 }
 for(i=0;i<n;i++)
 {
 pos=i;
 for(j=i+1;j<n;j++)
 {
 if(pr[j]<pr[pos])
 pos=j;
 }
 temp=pr[i];
 pr[i]=pr[pos];
 pr[pos]=temp;
 temp=bt[i];
 bt[i]=bt[pos];
 bt[pos]=temp;
 temp=p[i];
 p[i]=p[pos];
 p[pos]=temp;
 }
 wt[0]=0;
 for(i=1;i<n;i++)
 {
 wt[i]=0;
 for(j=0;j<i;j++)
 wt[i]+=bt[j];
 total+=wt[i];
 }
 avg_wt=total/n;
 total=0;
 printf("\nProcess      Burst Time      Waiting Time            Turnaround Time");
 for(i=0;i<n;i++)
```



```
 {
 tat[i]=bt[i]+wt[i];
 total+=tat[i];
 printf("\n[%d]          %d               %d              %d",p[i],bt[i],wt[i],tat[i]);
 }
 avg_tat=total/n;
 printf("\nAverage Waiting Time=%d",avg_wt);
 printf("\nAverage Turnaround Time=%d\n",avg_tat);
return 0;
}
```



```
prachi@ubuntu:~$ gcc -o prac prioritynp.c
prachi@ubuntu:~$ ./prac
Enter Total Number of Process:3

Enter Burst Time and Priority

P[1]
Burst Time:5
Priority:1

P[2]
Burst Time:3
Priority:2

P[3]
Burst Time:6
Priority:3

Process Burst Time      Waiting Time            Turnaround Time
[1]             5               0               5
[2]             3               5               8
[3]             6               8               14
Average Waiting Time=4
Average Turnaround Time=9
prachi@ubuntu:~$ 
```

(ii) Pre-emptive

Algorithm:

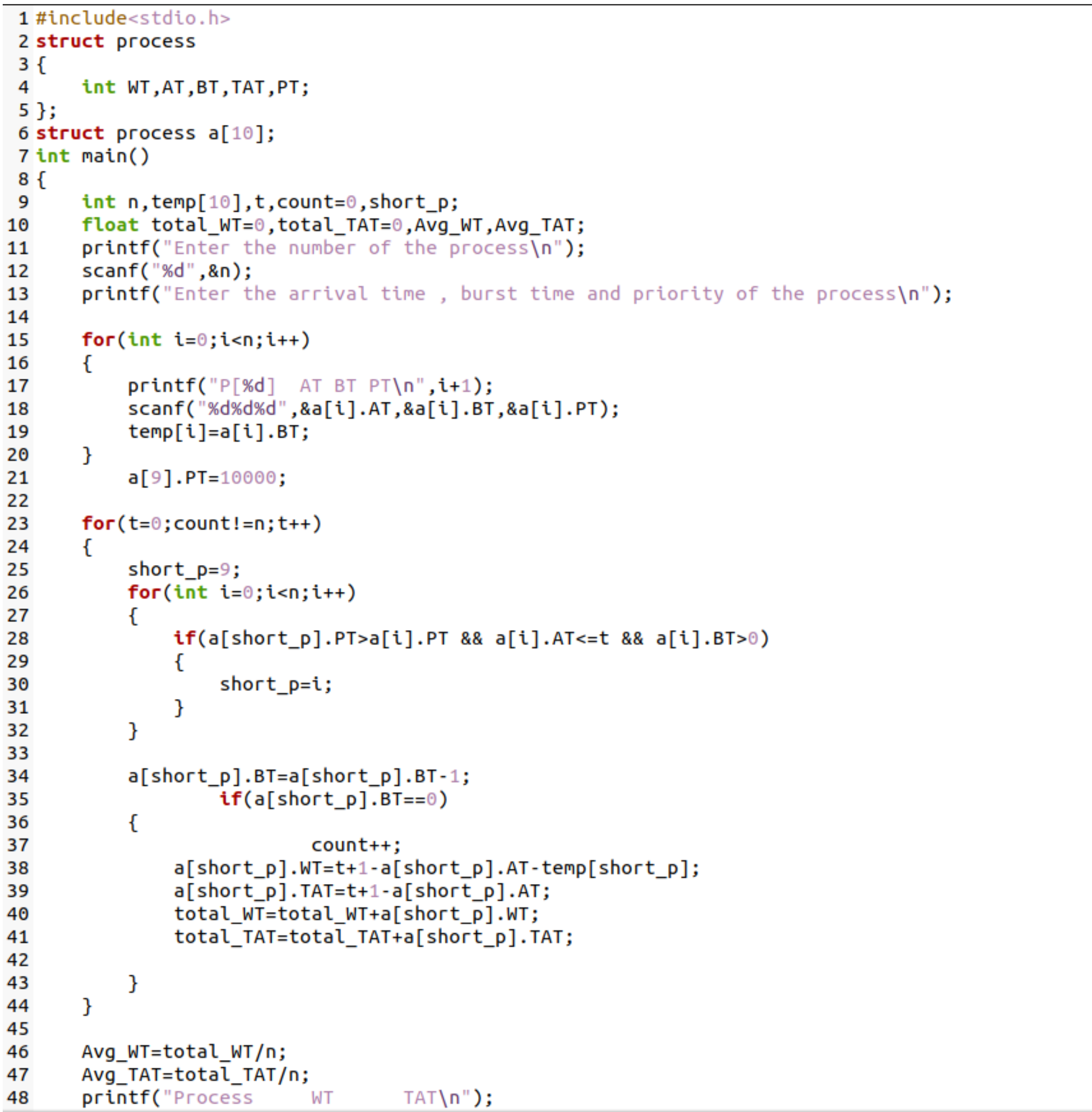
1. In the pre-emptive priority scheduling algorithm, the priority
of the process is compared with the priority of other processes which has arrived and are in the ready queue.
2. Now, the process which has the highest priority will be assigned to the CPU and will be executed first.
3. And all the other processes are scheduled in the ready queue based on their priority and arrival time.
4. Now, that all the processes get available in the ready queue,
the algorithm will behave as non-pre-emptive priority scheduling, which means the job scheduled will run till the

completion and no preemption will be done.

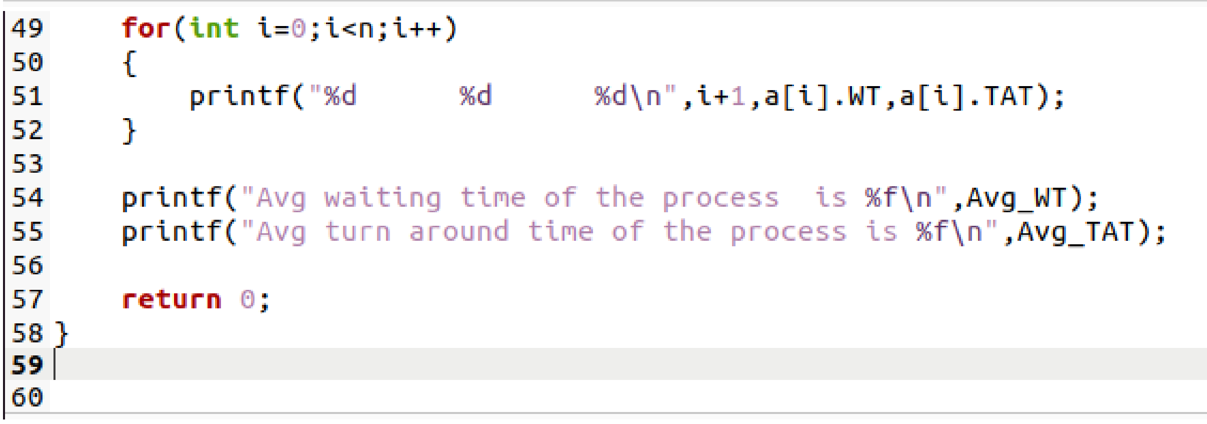
Code:

```
#include<stdio.h>
struct process
{
   int WT,AT,BT,TAT,PT;
};
struct process a[10];
int main()
{
   int n,temp[10],t,count=0,short_p;
   float total_WT=0,total_TAT=0,Avg_WT,Avg_TAT;
   printf("Enter the number of the process\n");
   scanf("%d",&n);
   printf("Enter the arrival time , burst time and priority of the process\n");

   for(int i=0;i<n;i++)
   {
      printf("P[%d]  AT BT PT\n",i+1);
      scanf("%d%d%d",&a[i].AT,&a[i].BT,&a[i].PT);
      temp[i]=a[i].BT;
   }
      a[9].PT=10000;

   for(t=0;count!=n;t++)
   {
      short_p=9;
      for(int i=0;i<n;i++)
      {
         if(a[short_p].PT>a[i].PT && a[i].AT<=t && a[i].BT>0)
         {
            short_p=i;
         }
      }

      a[short_p].BT=a[short_p].BT-1;
            if(a[short_p].BT==0)
      {
                  count++;
         a[short_p].WT=t+1-a[short_p].AT-temp[short_p];
         a[short_p].TAT=t+1-a[short_p].AT;
```

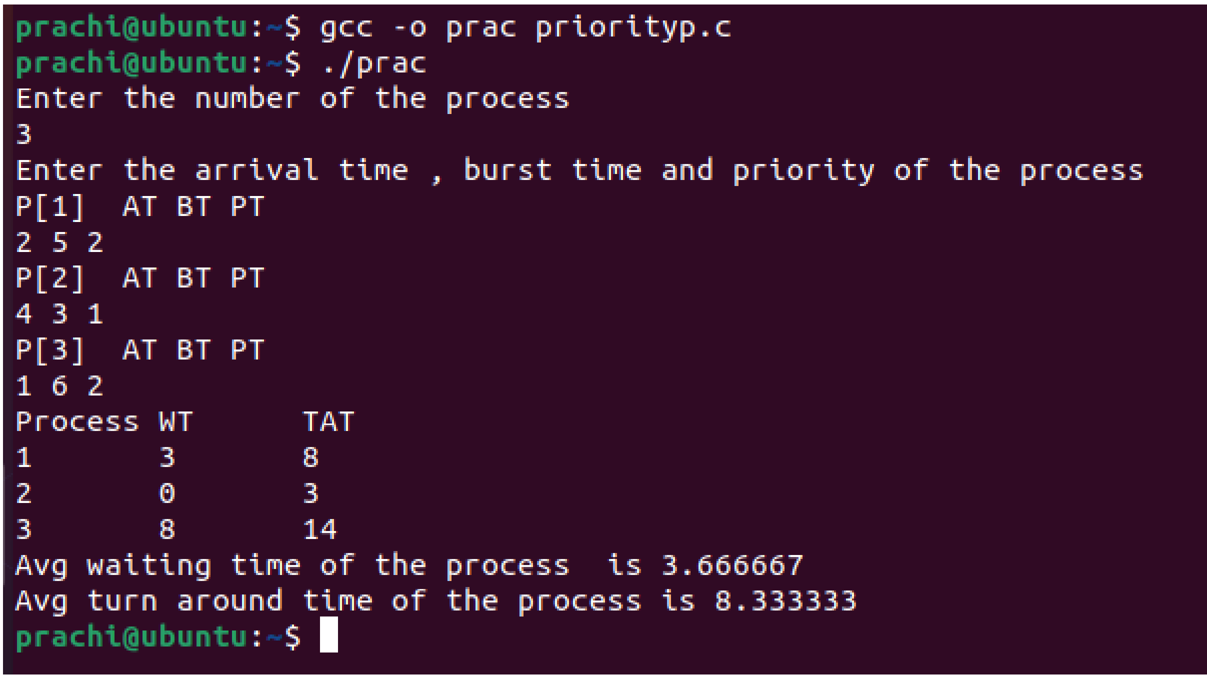
```
            total_WT=total_WT+a[short_p].WT;
            total_TAT=total_TAT+a[short_p].TAT;

        }
    }

    Avg_WT=total_WT/n;
    Avg_TAT=total_TAT/n;
    printf("Process        WT       TAT\n");
    for(int i=0;i<n;i++)
    {
        printf("%d            %d       %d\n",i+1,a[i].WT,a[i].TAT);
    }

    printf("Avg waiting time of the process  is %f\n",Avg_WT);
    printf("Avg turn around time of the process is %f\n",Avg_TAT);

    return 0;
}
```



```c
#include<stdio.h>
struct process
{
    int WT,AT,BT,TAT,PT;
};
struct process a[10];
int main()
{
    int n,temp[10],t,count=0,short_p;
    float total_WT=0,total_TAT=0,Avg_WT,Avg_TAT;
    printf("Enter the number of the process\n");
    scanf("%d",&n);
    printf("Enter the arrival time , burst time and priority of the process\n");

    for(int i=0;i<n;i++)
    {
        printf("P[%d]  AT BT PT\n",i+1);
        scanf("%d%d%d",&a[i].AT,&a[i].BT,&a[i].PT);
        temp[i]=a[i].BT;
    }
        a[9].PT=10000;

    for(t=0;count!=n;t++)
    {
        short_p=9;
        for(int i=0;i<n;i++)
        {
            if(a[short_p].PT>a[i].PT && a[i].AT<=t && a[i].BT>0)
            {
                short_p=i;
            }
        }

        a[short_p].BT=a[short_p].BT-1;
                if(a[short_p].BT==0)
        {
                        count++;
            a[short_p].WT=t+1-a[short_p].AT-temp[short_p];
            a[short_p].TAT=t+1-a[short_p].AT;
            total_WT=total_WT+a[short_p].WT;
            total_TAT=total_TAT+a[short_p].TAT;

        }
    }

    Avg_WT=total_WT/n;
    Avg_TAT=total_TAT/n;
    printf("Process     WT      TAT\n");
```



```c
    for(int i=0;i<n;i++)
    {
        printf("%d      %d      %d\n",i+1,a[i].WT,a[i].TAT);
    }

    printf("Avg waiting time of the process  is %f\n",Avg_WT);
    printf("Avg turn around time of the process is %f\n",Avg_TAT);

    return 0;
}


```



```
prachi@ubuntu:~$ gcc -o prac priorityp.c
prachi@ubuntu:~$ ./prac
Enter the number of the process
3
Enter the arrival time , burst time and priority of the process
P[1]  AT BT PT
2 5 2
P[2]  AT BT PT
4 3 1
P[3]  AT BT PT
1 6 2
Process WT      TAT
1       3       8
2       0       3
3       8       14
Avg waiting time of the process  is 3.666667
Avg turn around time of the process is 8.333333
prachi@ubuntu:~$ 
```

3. Memory Allocation Strategies:

(i) First Fit:

Algorithm:

-Read the number of processes and number of the block from the user

-Read the size of each block and the size of all the process requests.

-Start allocating the processes

-Display the results as shown below

-Stop

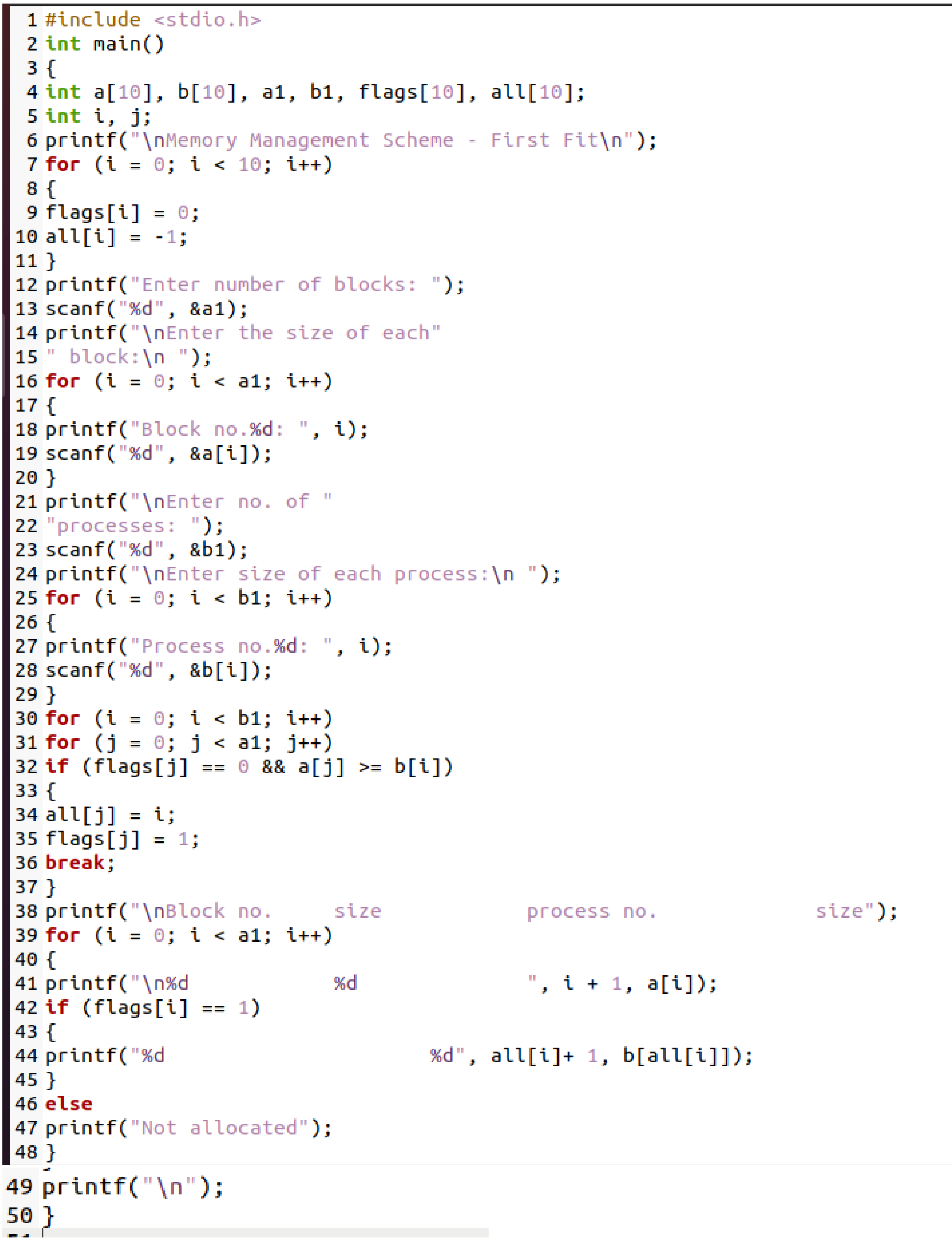
Code:

```
#include <stdio.h>
int main()
{
int a[10], b[10], a1, b1, flags[10], all[10];
int i, j;
printf("\nMemory Management Scheme - First Fit\n");
for (i = 0; i < 10; i++)
{
flags[i] = 0;
all[i] = -1;
}
```

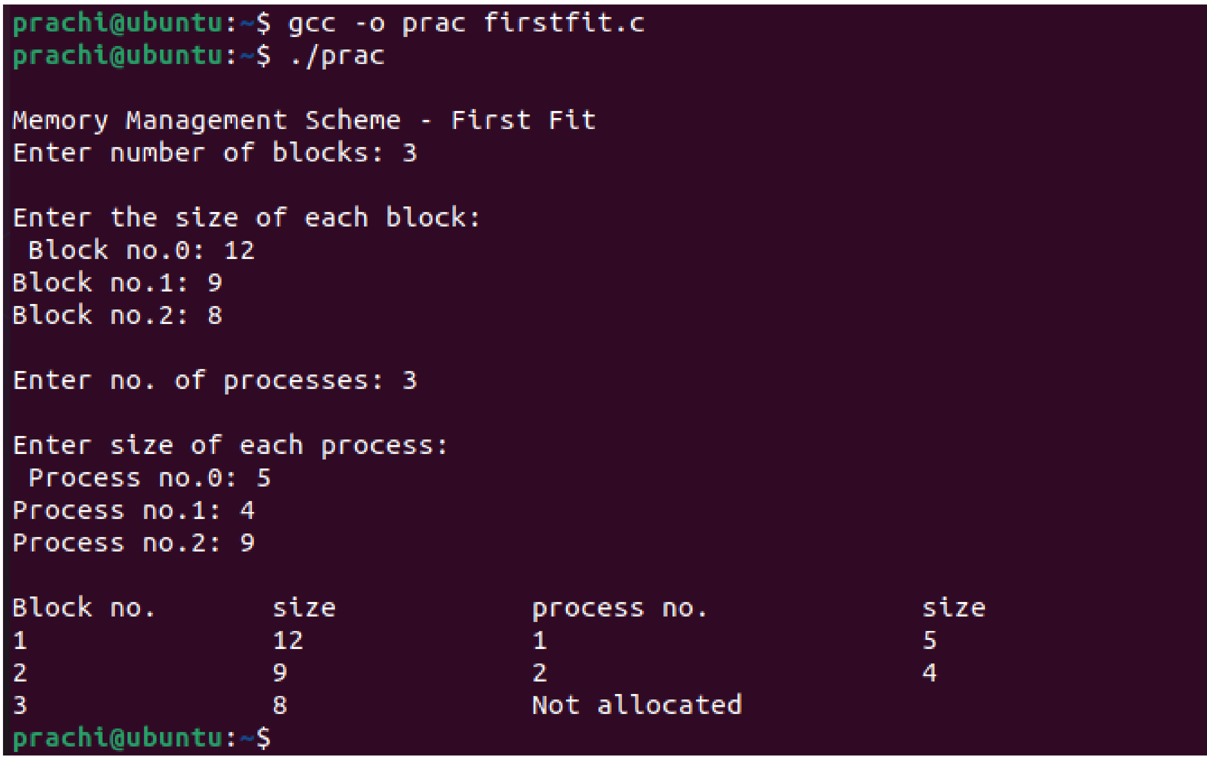
```
printf("Enter number of blocks: ");
scanf("%d", &a1);
printf("\nEnter the size of each"
" block:\n ");
for (i = 0; i < a1; i++)
{
printf("Block no.%d: ", i);
scanf("%d", &a[i]);
}
printf("\nEnter no. of "
"processes: ");
scanf("%d", &b1);
printf("\nEnter size of each process:\n ");
for (i = 0; i < b1; i++)
{
printf("Process no.%d: ", i);
scanf("%d", &b[i]);
}
for (i = 0; i < b1; i++)
for (j = 0; j < a1; j++)
if (flags[j] == 0 && a[j] >= b[i])
{
all[j] = i;
flags[j] = 1;
break;
}
printf("\nBlock no. size            process no.          size");
for (i = 0; i < a1; i++)
{
printf("\n%d         %d              ", i + 1, a[i]);
if (flags[i] == 1)
{
printf("%d                     %d", all[i]+ 1, b[all[i]]);
}
else
printf("Not allocated");
}
```

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

Output:



```c
#include <stdio.h>
int main()
{
int a[10], b[10], a1, b1, flags[10], all[10];
int i, j;
printf("\nMemory Management Scheme - First Fit\n");
for (i = 0; i < 10; i++)
{
flags[i] = 0;
all[i] = -1;
}
printf("Enter number of blocks: ");
scanf("%d", &a1);
printf("\nEnter the size of each"
" block:\n ");
for (i = 0; i < a1; i++)
{
printf("Block no.%d: ", i);
scanf("%d", &a[i]);
}
printf("\nEnter no. of "
"processes: ");
scanf("%d", &b1);
printf("\nEnter size of each process:\n ");
for (i = 0; i < b1; i++)
{
printf("Process no.%d: ", i);
scanf("%d", &b[i]);
}
for (i = 0; i < b1; i++)
for (j = 0; j < a1; j++)
if (flags[j] == 0 && a[j] >= b[i])
{
all[j] = i;
flags[j] = 1;
break;
}
printf("\nBlock no.     size            process no.              size");
for (i = 0; i < a1; i++)
{
printf("\n%d            %d              ", i + 1, a[i]);
if (flags[i] == 1)
{
printf("%d                     %d", all[i]+ 1, b[all[i]]);
}
else
printf("Not allocated");
}
printf("\n");
}
```



```
prachi@ubuntu:~$ gcc -o prac firstfit.c
prachi@ubuntu:~$ ./prac

Memory Management Scheme - First Fit
Enter number of blocks: 3

Enter the size of each block:
 Block no.0: 12
Block no.1: 9
Block no.2: 8

Enter no. of processes: 3

Enter size of each process:
 Process no.0: 5
Process no.1: 4
Process no.2: 9

Block no.       size            process no.             size
1               12              1                       5
2               9               2                       4
3               8               Not allocated
prachi@ubuntu:~$
```

(ii) **Best Fit:**

Algorithm:

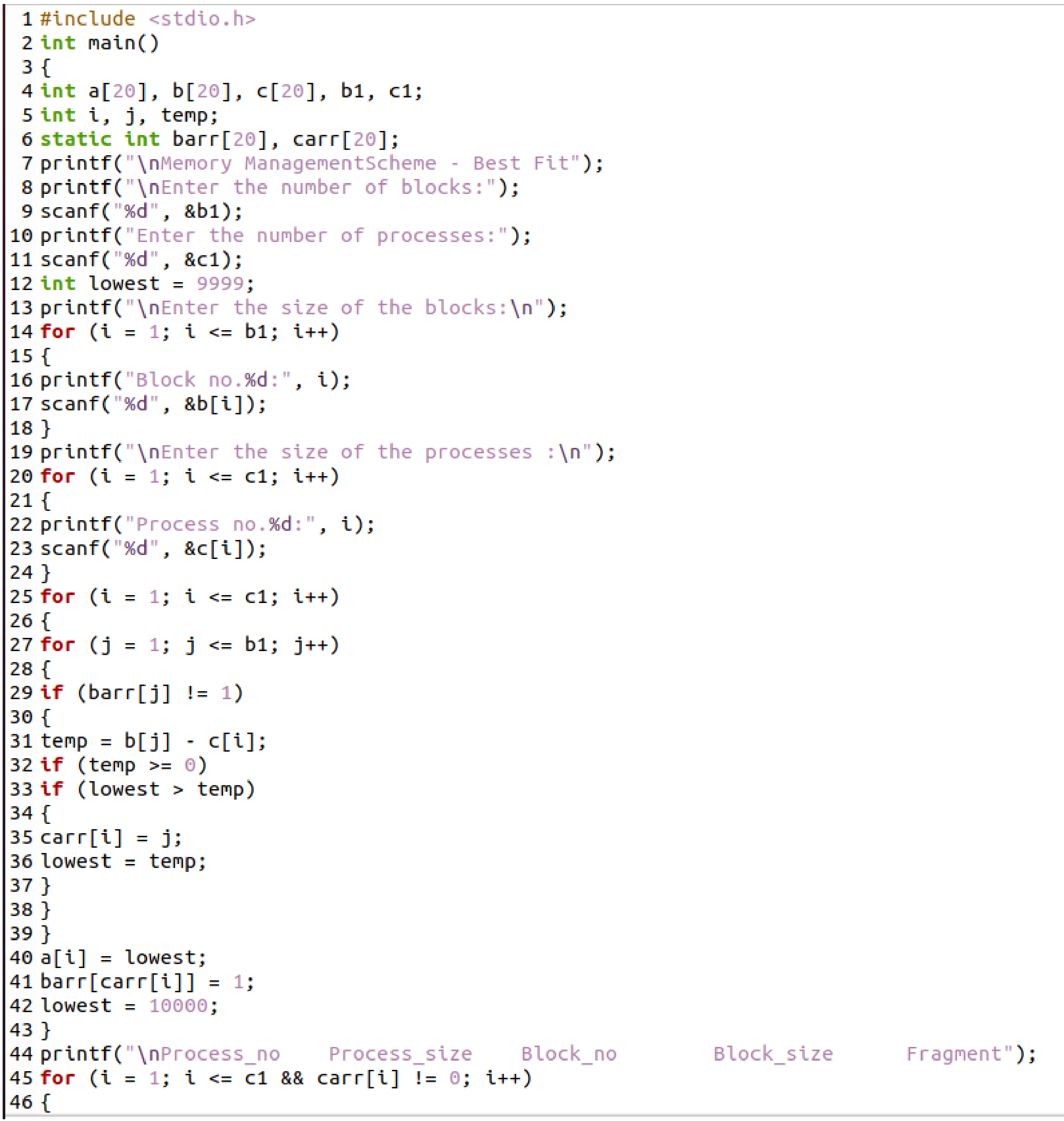
-Read the number of processes and number of blocks from the user
-Get the size of each block and process requests
-Then select the best memory block
-Display the result as shown below
-The fragmentation column will keep track of wasted memory
-Stop

Code:

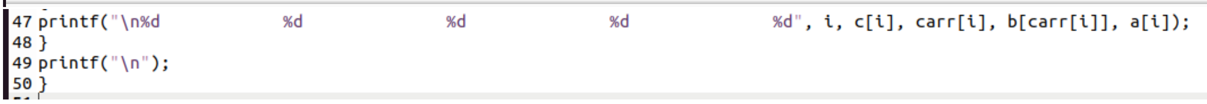
```
#include <stdio.h>
int main()
{
int a[20], b[20], c[20], b1, c1;
int i, j, temp;
static int barr[20], carr[20];
printf("\nMemory ManagementScheme - Best Fit");
printf("\nEnter the number of blocks:");
scanf("%d", &b1);
printf("Enter the number of processes:");
scanf("%d", &c1);
int lowest = 9999;
```

```
printf("\nEnter the size of the blocks:\n");
for (i = 1; i <= b1; i++)
{
printf("Block no.%d:", i);
scanf("%d", &b[i]);
}
printf("\nEnter the size of the processes :\n");
for (i = 1; i <= c1; i++)
{
printf("Process no.%d:", i);
scanf("%d", &c[i]);
}
for (i = 1; i <= c1; i++)
{
for (j = 1; j <= b1; j++)
{
if (barr[j] != 1)
{
temp = b[j] - c[i];
if (temp >= 0)
if (lowest > temp)
{
carr[i] = j;
lowest = temp;
}
}
}
a[i] = lowest;
barr[carr[i]] = 1;
lowest = 10000;
}
printf("\nProcess_no      Process_sizeBlock_no      Block_size
         Fragment");
for (i = 1; i <= c1 && carr[i] != 0; i++)
{
printf("\n%d         %d            %d            %d            %d", i,
c[i], carr[i], b[carr[i]], a[i]);
}
printf("\n");
}
```

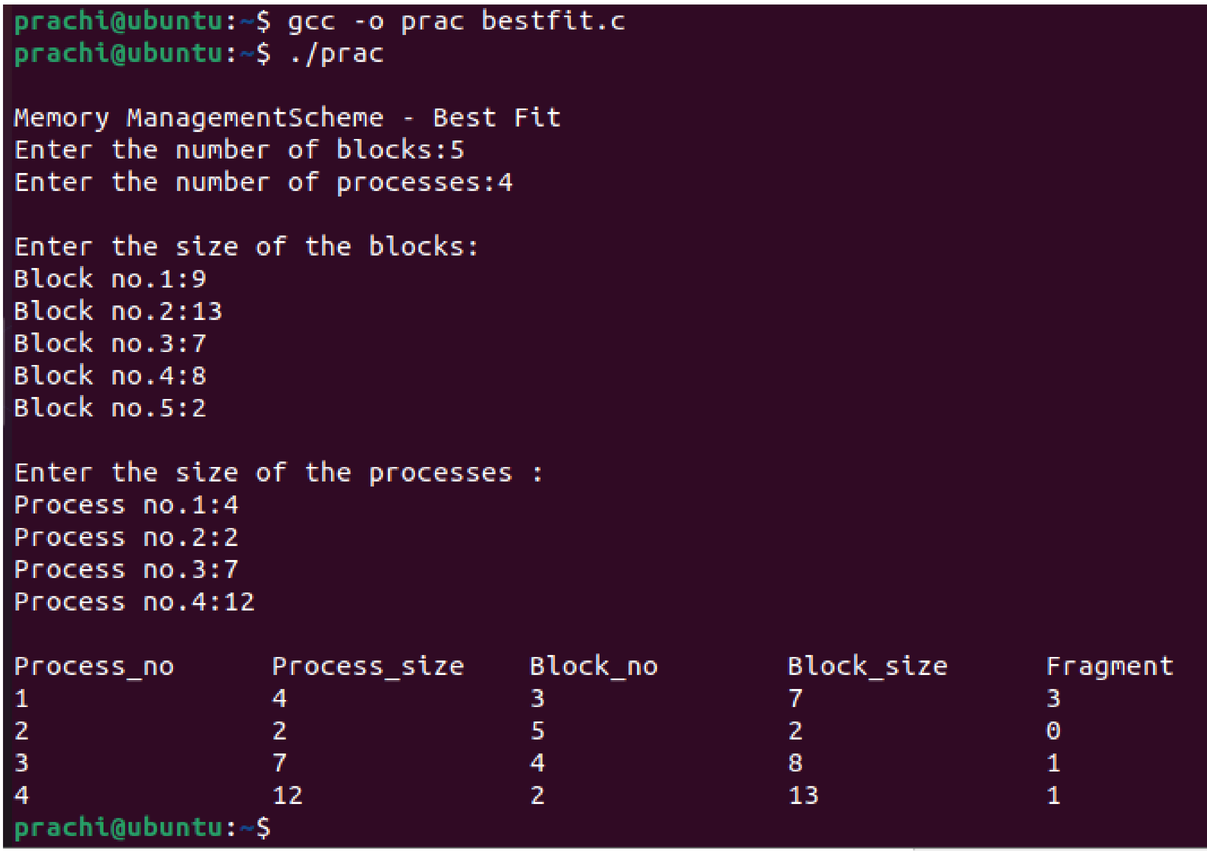
Output:



```
#include <stdio.h>
int main()
{
int a[20], b[20], c[20], b1, c1;
int i, j, temp;
static int barr[20], carr[20];
printf("\nMemory ManagementScheme - Best Fit");
printf("\nEnter the number of blocks:");
scanf("%d", &b1);
printf("Enter the number of processes:");
scanf("%d", &c1);
int lowest = 9999;
printf("\nEnter the size of the blocks:\n");
for (i = 1; i <= b1; i++)
{
printf("Block no.%d:", i);
scanf("%d", &b[i]);
}
printf("\nEnter the size of the processes :\n");
for (i = 1; i <= c1; i++)
{
printf("Process no.%d:", i);
scanf("%d", &c[i]);
}
for (i = 1; i <= c1; i++)
{
for (j = 1; j <= b1; j++)
{
if (barr[j] != 1)
{
temp = b[j] - c[i];
if (temp >= 0)
if (lowest > temp)
{
carr[i] = j;
lowest = temp;
}
}
}
a[i] = lowest;
barr[carr[i]] = 1;
lowest = 10000;
}
printf("\nProcess_no    Process_size    Block_no        Block_size      Fragment");
for (i = 1; i <= c1 && carr[i] != 0; i++)
{
```



```
printf("\n%d            %d            %d            %d            %d", i, c[i], carr[i], b[carr[i]], a[i]);
}
printf("\n");
}
```



```
prachi@ubuntu:~$ gcc -o prac bestfit.c
prachi@ubuntu:~$ ./prac

Memory ManagementScheme - Best Fit
Enter the number of blocks:5
Enter the number of processes:4

Enter the size of the blocks:
Block no.1:9
Block no.2:13
Block no.3:7
Block no.4:8
Block no.5:2

Enter the size of the processes :
Process no.1:4
Process no.2:2
Process no.3:7
Process no.4:12

Process_no      Process_size    Block_no        Block_size      Fragment
1               4               3               7               3
2               2               5               2               0
3               7               4               8               1
4               12              2               13              1
prachi@ubuntu:~$
```

(iii) Worst Fit:

Algorithm:

-Read total number of block and files

-Get the size of each block and the files from the user

-Start from the first process and find the maximum block size that can be assigned to the current process, if found then assign it to the current process.

-If not found then leave that process and move ahead to check the rest of the processes

-Display the result as shown below

-The fragmentation column keeps the track of wasted memory

-Stop

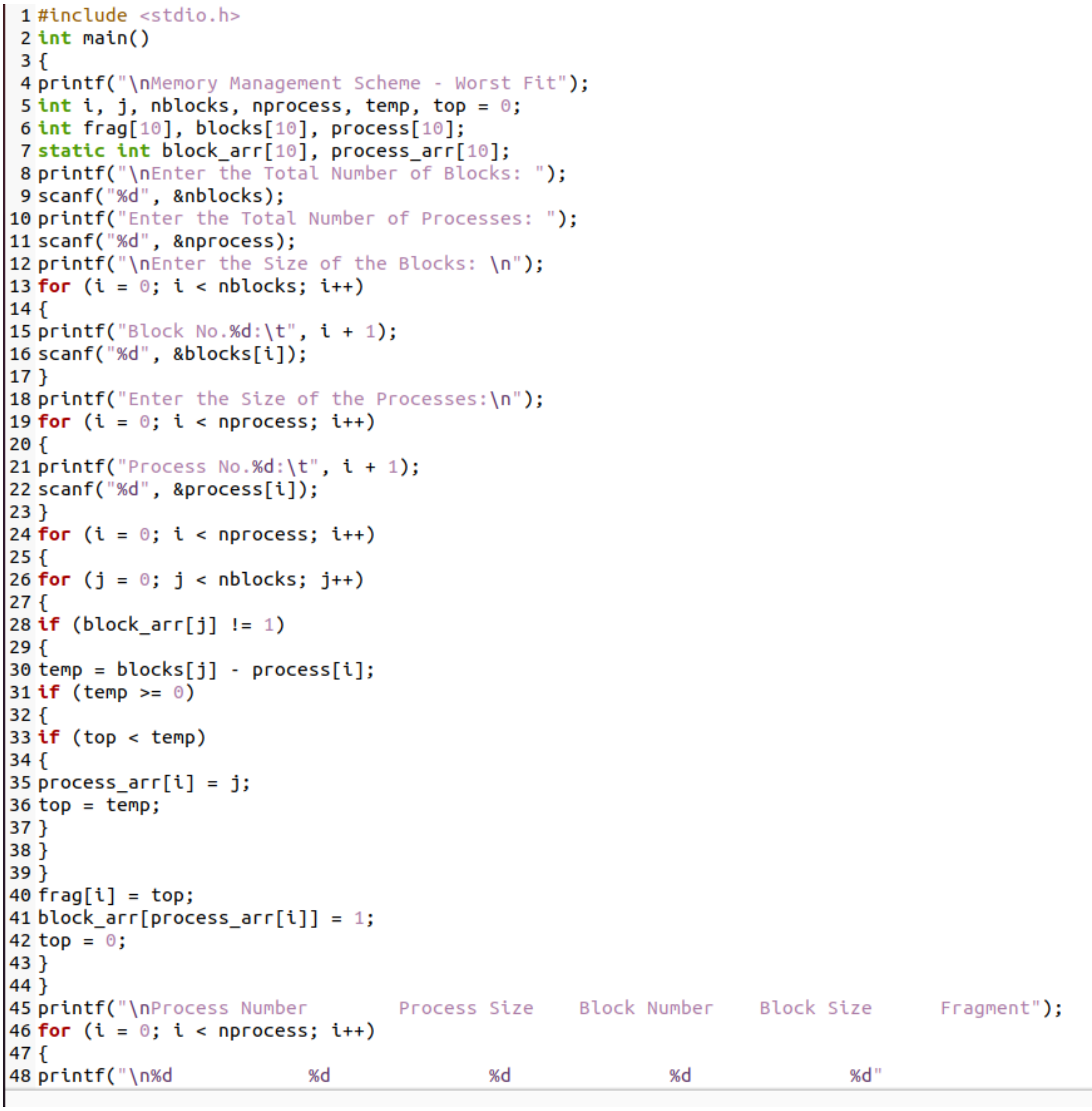
Code:

```
#include <stdio.h>
int main()
{
printf("\nMemory Management Scheme - Worst Fit");
int i, j, nblocks, nprocess, temp, top = 0;
int frag[10], blocks[10], process[10];
static int block_arr[10], process_arr[10];
```

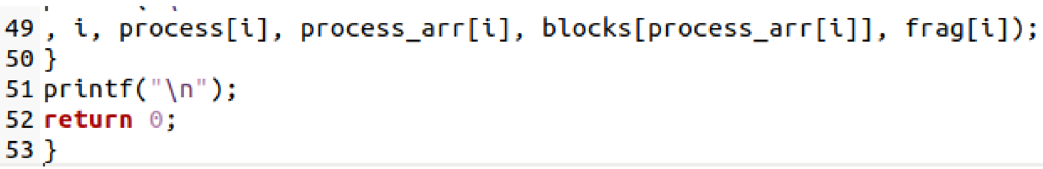
```
printf("\nEnter the Total Number of Blocks: ");
scanf("%d", &nblocks);
printf("Enter the Total Number of Processes: ");
scanf("%d", &nprocess);
printf("\nEnter the Size of the Blocks: \n");
for (i = 0; i < nblocks; i++)
{
printf("Block No.%d:\t", i + 1);
scanf("%d", &blocks[i]);
}
printf("Enter the Size of the Processes:\n");
for (i = 0; i < nprocess; i++)
{
printf("Process No.%d:\t", i + 1);
scanf("%d", &process[i]);
}
for (i = 0; i < nprocess; i++)
{
for (j = 0; j < nblocks; j++)
{
if (block_arr[j] != 1)
{
temp = blocks[j] - process[i];
if (temp >= 0)
{
if (top < temp)
{
process_arr[i] = j;
top = temp;
}
}
}
frag[i] = top;
block_arr[process_arr[i]] = 1;
top = 0;
}
}
```

```
printf("\nProcess Number        Process Size Block Number
          Block Size     Fragment");
for (i = 0; i < nprocess; i++)
{
printf("\n%d         %d            %d             %d             %d"
, i, process[i], process_arr[i], blocks[process_arr[i]], frag[i]);
}
printf("\n");
return 0;
}
```

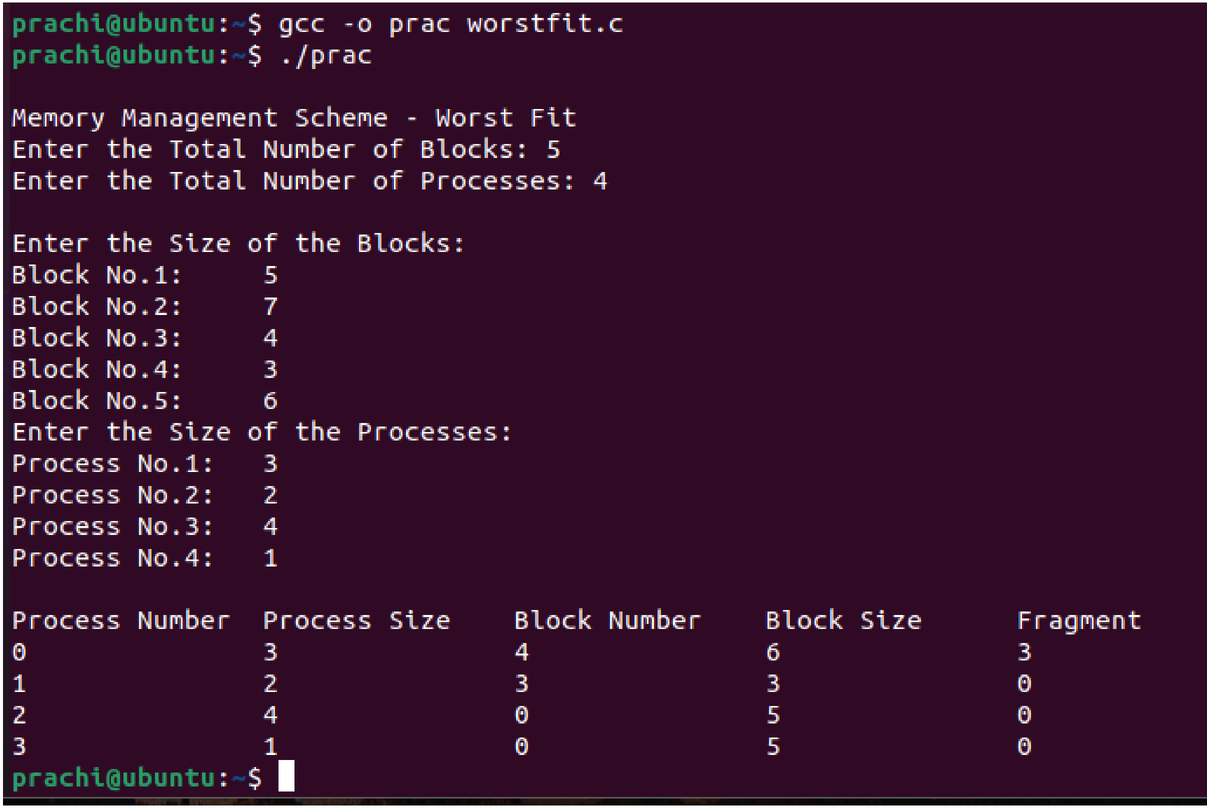
Output:



```
#include <stdio.h>
int main()
{
printf("\nMemory Management Scheme - Worst Fit");
int i, j, nblocks, nprocess, temp, top = 0;
int frag[10], blocks[10], process[10];
static int block_arr[10], process_arr[10];
printf("\nEnter the Total Number of Blocks: ");
scanf("%d", &nblocks);
printf("Enter the Total Number of Processes: ");
scanf("%d", &nprocess);
printf("\nEnter the Size of the Blocks: \n");
for (i = 0; i < nblocks; i++)
{
printf("Block No.%d:\t", i + 1);
scanf("%d", &blocks[i]);
}
printf("Enter the Size of the Processes:\n");
for (i = 0; i < nprocess; i++)
{
printf("Process No.%d:\t", i + 1);
scanf("%d", &process[i]);
}
for (i = 0; i < nprocess; i++)
{
for (j = 0; j < nblocks; j++)
{
if (block_arr[j] != 1)
{
temp = blocks[j] - process[i];
if (temp >= 0)
{
if (top < temp)
{
process_arr[i] = j;
top = temp;
}
}
}
frag[i] = top;
block_arr[process_arr[i]] = 1;
top = 0;
}
}
printf("\nProcess Number        Process Size    Block Number    Block Size      Fragment");
for (i = 0; i < nprocess; i++)
{
printf("\n%d            %d              %d              %d              %d"
```



```
, i, process[i], process_arr[i], blocks[process_arr[i]], frag[i]);
}
printf("\n");
return 0;
}
```



```
prachi@ubuntu:~$ gcc -o prac worstfit.c
prachi@ubuntu:~$ ./prac

Memory Management Scheme - Worst Fit
Enter the Total Number of Blocks: 5
Enter the Total Number of Processes: 4

Enter the Size of the Blocks:
Block No.1:     5
Block No.2:     7
Block No.3:     4
Block No.4:     3
Block No.5:     6
Enter the Size of the Processes:
Process No.1:   3
Process No.2:   2
Process No.3:   4
Process No.4:   1

Process Number  Process Size    Block Number    Block Size      Fragment
0               3               4               6               3
1               2               3               3               0
2               4               0               5               0
3               1               0               5               0
prachi@ubuntu:~$ 
```

(iv) Next Fit:

Algorithm:

-Input the number of memory blocks and their sizes and initializes all the blocks as free.

-Input the number of processes and their sizes.

-Start by picking each process and check if it can be assigned to the current block, if yes, allocate it the required memory and check for next process but from the block where we left not from starting.

-If the current block size is smaller then keep checking the further blocks.

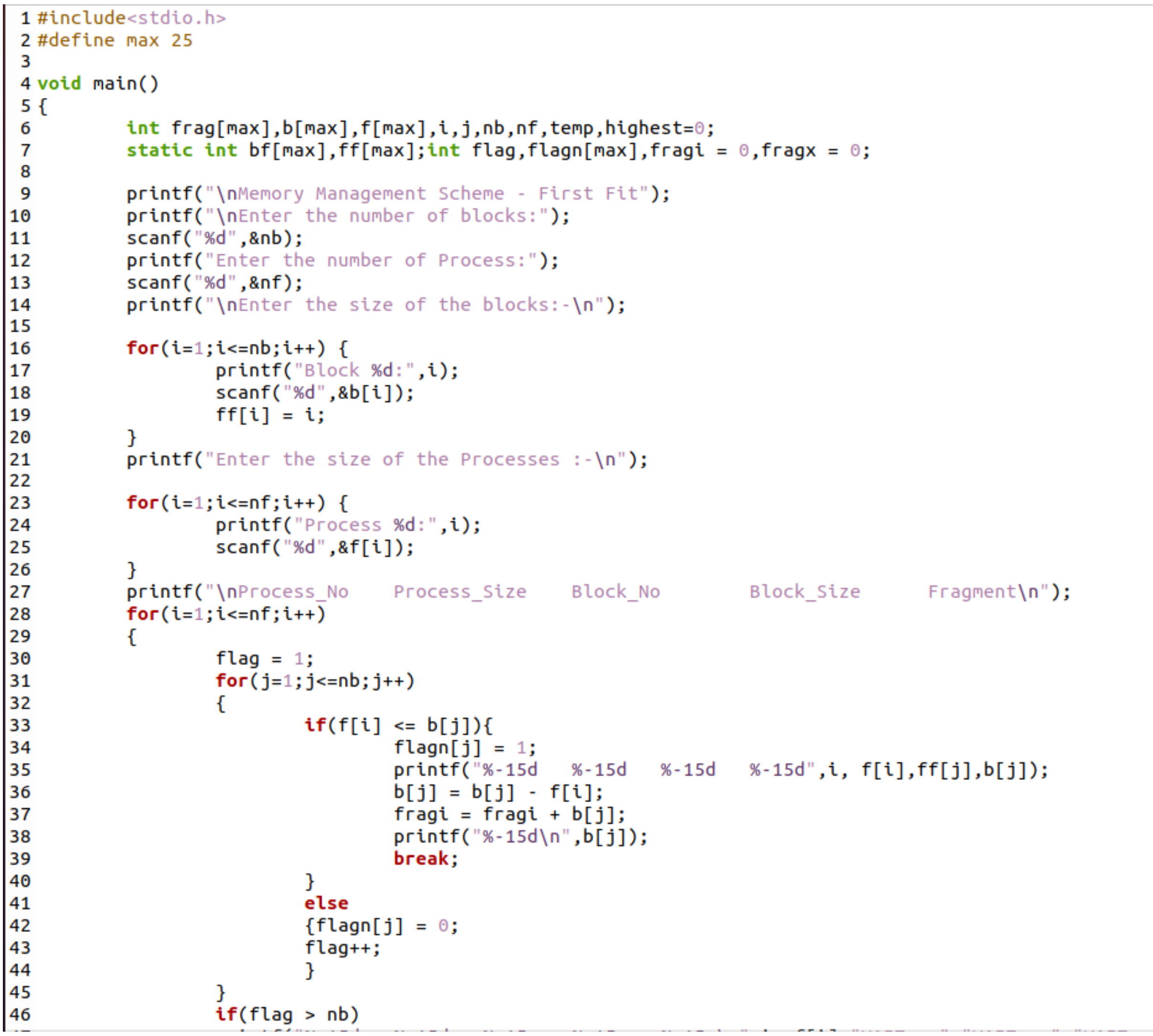
Code:

```
#include<stdio.h>
#define max 25

void main()
{
        int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;
```

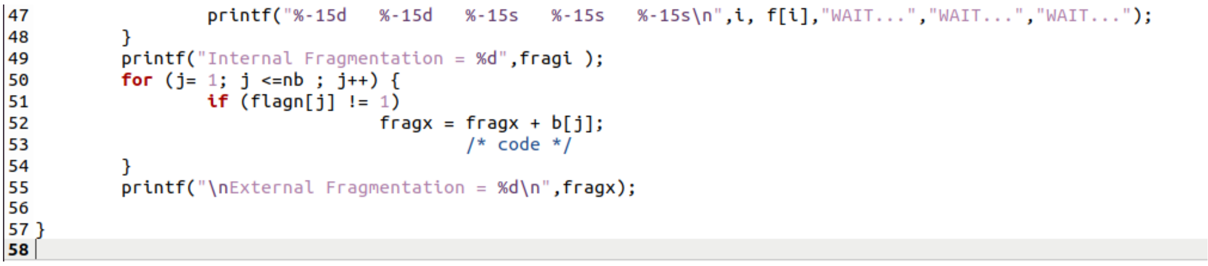
```
    static int bf[max],ff[max];int flag,flagn[max],fragi = 0,fragx =
0;

    printf("\nMemory Management Scheme - First Fit");
    printf("\nEnter the number of blocks:");
    scanf("%d",&nb);
    printf("Enter the number of Process:");
    scanf("%d",&nf);
    printf("\nEnter the size of the blocks:-\n");

    for(i=1;i<=nb;i++) {
        printf("Block %d:",i);
        scanf("%d",&b[i]);
        ff[i] = i;
    }
    printf("Enter the size of the Processes :-\n");

    for(i=1;i<=nf;i++) {
        printf("Process %d:",i);
        scanf("%d",&f[i]);
    }
    printf("\nProcess_No      Process_Size        Block_No
    Block_Size    Fragment\n");
    for(i=1;i<=nf;i++)
    {
        flag = 1;
        for(j=1;j<=nb;j++)
        {
            if(f[i] <= b[j]){
                flagn[j] = 1;
                printf("%-15d        %-15d%-15d%-15d",i,
f[i],ff[j],b[j]);
                b[j] = b[j] - f[i];
                fragi = fragi + b[j];
                printf("%-15d\n",b[j]);
                break;
            }
```

```
                        else
                        {flagn[j] = 0;
                        flag++;
                        }
                }
                if(flag > nb)
                printf("%-15d        %-15d%-15s %-15s %-15s\n",i,
f[i],"WAIT...","WAIT...","WAIT...");
        }
        printf("Internal Fragmentation = %d",fragi );
        for (j= 1; j <=nb ; j++) {
                if (flagn[j] != 1)
                                fragx = fragx + b[j];
                                        /* code */
        }
        printf("\nExternal Fragmentation = %d\n",fragx);

}
```

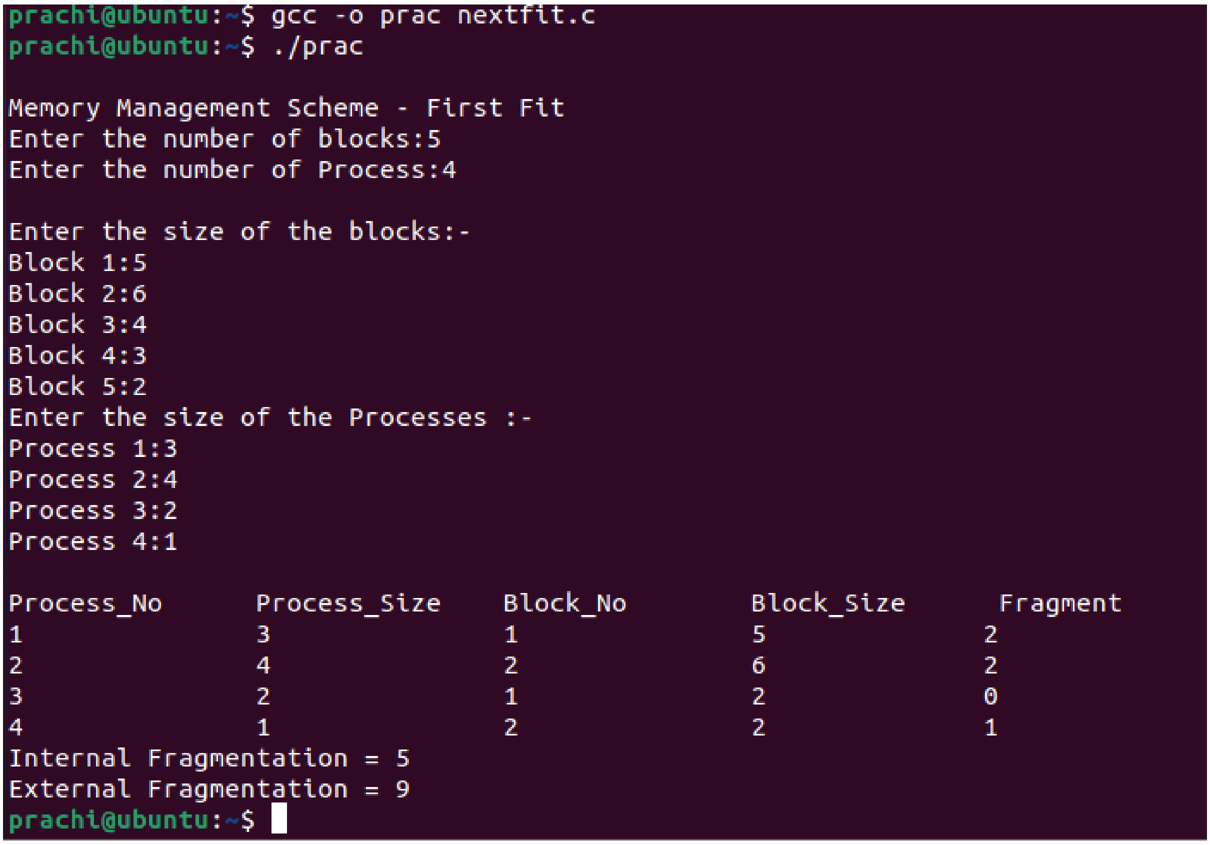
Output:



```c
#include<stdio.h>
#define max 25

void main()
{
        int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;
        static int bf[max],ff[max];int flag,flagn[max],fragi = 0,fragx = 0;

        printf("\nMemory Management Scheme - First Fit");
        printf("\nEnter the number of blocks:");
        scanf("%d",&nb);
        printf("Enter the number of Process:");
        scanf("%d",&nf);
        printf("\nEnter the size of the blocks:-\n");

        for(i=1;i<=nb;i++) {
                printf("Block %d:",i);
                scanf("%d",&b[i]);
                ff[i] = i;
        }
        printf("Enter the size of the Processes :-\n");

        for(i=1;i<=nf;i++) {
                printf("Process %d:",i);
                scanf("%d",&f[i]);
        }
        printf("\nProcess_No     Process_Size     Block_No         Block_Size       Fragment\n");
        for(i=1;i<=nf;i++)
        {
                flag = 1;
                for(j=1;j<=nb;j++)
                {
                        if(f[i] <= b[j]){
                                flagn[j] = 1;
                                printf("%-15d   %-15d   %-15d   %-15d",i, f[i],ff[j],b[j]);
                                b[j] = b[j] - f[i];
                                fragi = fragi + b[j];
                                printf("%-15d\n",b[j]);
                                break;
                        }
                        else
                        {flagn[j] = 0;
                        flag++;
                        }
                }
                if(flag > nb)
```



```c
                printf("%-15d   %-15d   %-15s   %-15s   %-15s\n",i, f[i],"WAIT...","WAIT...","WAIT...");
        }
        printf("Internal Fragmentation = %d",fragi );
        for (j= 1; j <=nb ; j++) {
                if (flagn[j] != 1)
                                fragx = fragx + b[j];
                                        /* code */
        }
        printf("\nExternal Fragmentation = %d\n",fragx);

}

```



```
prachi@ubuntu:~$ gcc -o prac nextfit.c
prachi@ubuntu:~$ ./prac

Memory Management Scheme - First Fit
Enter the number of blocks:5
Enter the number of Process:4

Enter the size of the blocks:-
Block 1:5
Block 2:6
Block 3:4
Block 4:3
Block 5:2
Enter the size of the Processes :-
Process 1:3
Process 2:4
Process 3:2
Process 4:1

Process_No      Process_Size    Block_No        Block_Size       Fragment
1               3               1               5               2
2               4               2               6               2
3               2               1               2               0
4               1               2               2               1
Internal Fragmentation = 5
External Fragmentation = 9
prachi@ubuntu:~$
```

Prachi Balodia

20BDS0177

Operating Systems LAB DA-4

1. **Banker's Algorithm**

   Safety Algorithm

   The algorithm for finding out whether or not a system is in a safe state can be described as follows:

   1) Let Work and Finish be vectors of length 'm' and 'n'
   respectively. Initialize: Work = Available
   Finish[i] = false; for i=1, 2, 3, 4….n
   2) Find an i such that both
   a) Finish[i] = false
   b) Needi <= Work
   if no such i exists goto step (4)
   3) Work = Work + Allocation[i]
   Finish[i] = true
   goto step (2)
   4) ifFinish[i]=trueforall i then the system is in a safe state

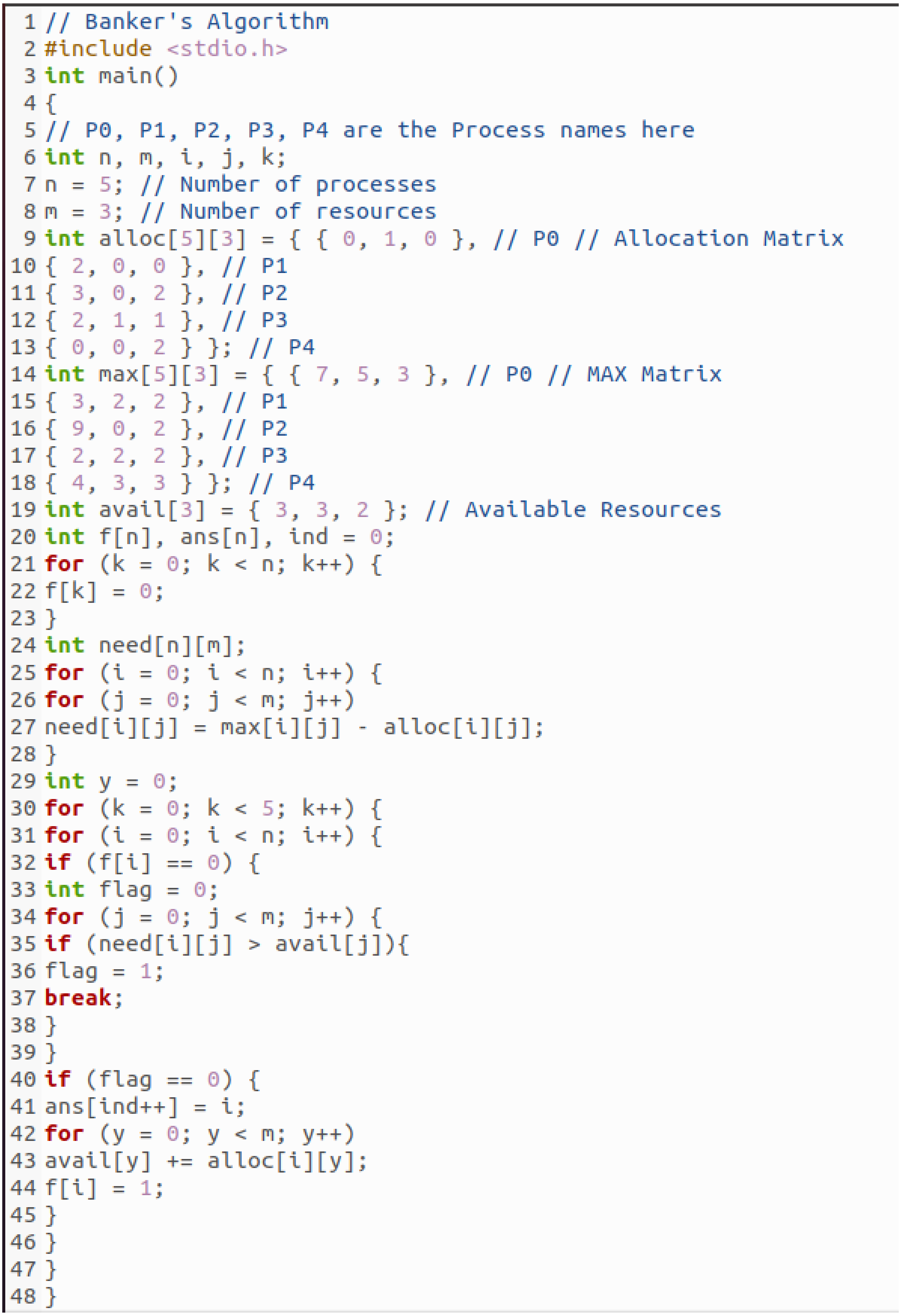
   Resource-Request Algorithm

   Let Requesti be the request array for process Pi. Requesti [j] = k means process Pi wants k instances of resource type Rj. When a request for resources is made by process Pi, the following actions are taken:

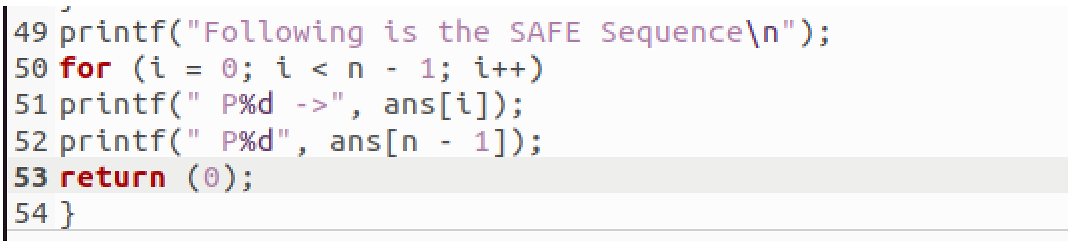
   1) If Requesti <= Needi
   Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.
   2) If Requesti <= Available
   Goto step (3); otherwise, Pi must wait, since the resources are not available.
   3) Have the system pretend to have allocated the requested resources to
   process Pi by modifying the state as
   follows:
   Available = Available – Requesti
   Allocationi = Allocationi + Requesti
   Needi = Needi– Requesti

Code:

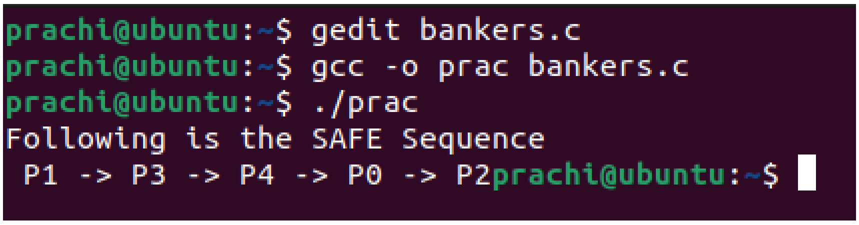
```
// Banker's Algorithm
#include <stdio.h>
int main()
{
// P0, P1, P2, P3, P4 are the Process names here
int n, m, i, j, k;
n = 5; // Number of processes
m = 3; // Number of resources
int alloc[5][3] = { { 0, 1, 0 }, // P0 // Allocation Matrix
{ 2, 0, 0 }, // P1
{ 3, 0, 2 }, // P2
{ 2, 1, 1 }, // P3
{ 0, 0, 2 } }; // P4
int max[5][3] = { { 7, 5, 3 }, // P0 // MAX Matrix
{ 3, 2, 2 }, // P1
{ 9, 0, 2 }, // P2
{ 2, 2, 2 }, // P3
{ 4, 3, 3 } }; // P4
int avail[3] = { 3, 3, 2 }; // Available Resources
int f[n], ans[n], ind = 0;
for (k = 0; k < n; k++) {
f[k] = 0;
}
int need[n][m];
for (i = 0; i < n; i++) {
for (j = 0; j < m; j++)
need[i][j] = max[i][j] - alloc[i][j];
}
int y = 0;
for (k = 0; k < 5; k++) {
for (i = 0; i < n; i++) {
if (f[i] == 0) {
int flag = 0;
for (j = 0; j < m; j++) {
if (need[i][j] > avail[j]){
flag = 1;
break;
}
}
if (flag == 0) {
ans[ind++] = i;
for (y = 0; y < m; y++)
avail[y] += alloc[i][y];
f[i] = 1;
}
}
}
}
printf("Following is the SAFE Sequence\n");
for (i = 0; i < n - 1; i++)
printf(" P%d ->", ans[i]);
printf(" P%d", ans[n - 1]);
return (0);
}
```



```c
// Banker's Algorithm
#include <stdio.h>
int main()
{
// P0, P1, P2, P3, P4 are the Process names here
int n, m, i, j, k;
n = 5; // Number of processes
m = 3; // Number of resources
int alloc[5][3] = { { 0, 1, 0 }, // P0 // Allocation Matrix
{ 2, 0, 0 }, // P1
{ 3, 0, 2 }, // P2
{ 2, 1, 1 }, // P3
{ 0, 0, 2 } }; // P4
int max[5][3] = { { 7, 5, 3 }, // P0 // MAX Matrix
{ 3, 2, 2 }, // P1
{ 9, 0, 2 }, // P2
{ 2, 2, 2 }, // P3
{ 4, 3, 3 } }; // P4
int avail[3] = { 3, 3, 2 }; // Available Resources
int f[n], ans[n], ind = 0;
for (k = 0; k < n; k++) {
f[k] = 0;
}
int need[n][m];
for (i = 0; i < n; i++) {
for (j = 0; j < m; j++)
need[i][j] = max[i][j] - alloc[i][j];
}
int y = 0;
for (k = 0; k < 5; k++) {
for (i = 0; i < n; i++) {
if (f[i] == 0) {
int flag = 0;
for (j = 0; j < m; j++) {
if (need[i][j] > avail[j]){
flag = 1;
break;
}
}
if (flag == 0) {
ans[ind++] = i;
for (y = 0; y < m; y++)
avail[y] += alloc[i][y];
f[i] = 1;
}
}
}
}
```



```
printf("Following is the SAFE Sequence\n");
for (i = 0; i < n - 1; i++)
printf(" P%d ->", ans[i]);
printf(" P%d", ans[n - 1]);
return (0);
}
```



```
prachi@ubuntu:~$ gedit bankers.c
prachi@ubuntu:~$ gcc -o prac bankers.c
prachi@ubuntu:~$ ./prac
Following is the SAFE Sequence
 P1 -> P3 -> P4 -> P0 -> P2prachi@ubuntu:~$
```

2. Page Replacement Algorithm- FIFO, LRU, OPTIMAL

Algorithm:

Start the program

Obtain the number of sequences, number of frames and sequence string from the user

Now when a page is not in the frame comes, increment the number of page fault and

remove the page that come in the first in FIFO algorithm

In LRU algorithm, when a page fault occurs, the page which most recently used is

removed

5.In Optimal algorithm when a page fault occurs, the page which will not be used in

near future is removed.

Display the number of faults.

Stop the program

Code:

```
//page replacement
#include<stdio.h>
int n,nf;
int in[100];
int p[50];
int hit=0;
int i,j,k;
int pgfaultcnt=0;
void getData()
{
printf("\n Enter length of page reference sequence:");
```

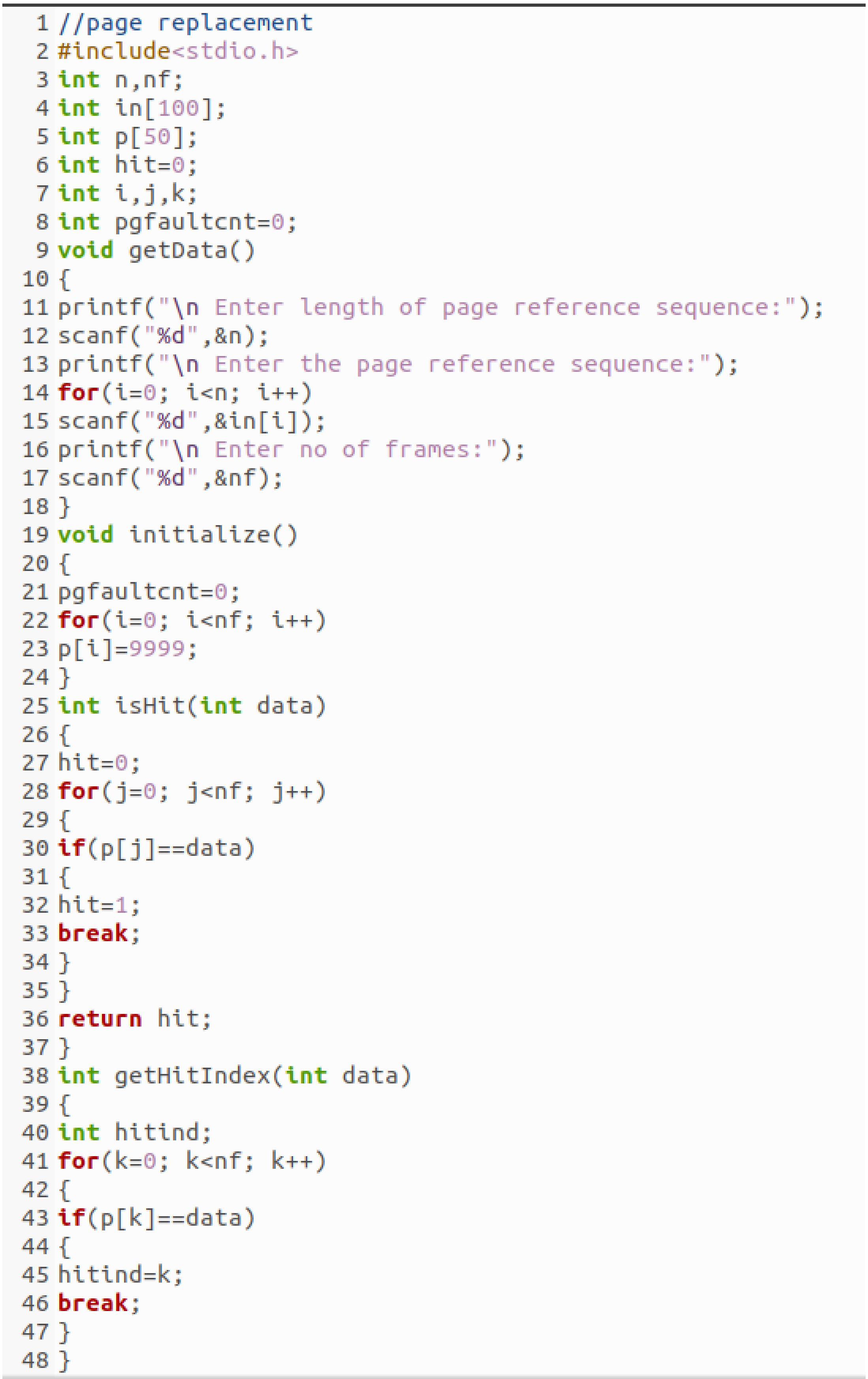
```
scanf("%d",&n);
printf("\n Enter the page reference sequence:");
for(i=0; i<n; i++)
scanf("%d",&in[i]);
printf("\n Enter no of frames:");
scanf("%d",&nf);
}
void initialize()
{
pgfaultcnt=0;
for(i=0; i<nf; i++)
p[i]=9999;
}
int isHit(int data)
{
hit=0;
for(j=0; j<nf; j++)
{
if(p[j]==data)
{
hit=1;
break;
}
}
return hit;
}
int getHitIndex(int data)
{
int hitind;
for(k=0; k<nf; k++)
{
if(p[k]==data)
{
hitind=k;
break;
}
}
```

```
return hitind;
}
void dispPages()
{
for (k=0; k<nf; k++)
{
if(p[k]!=9999)
printf(" %d",p[k]);
}
}
void dispPgFaultCnt()
{
printf("\n Total no of page faults:%d",pgfaultcnt);
}
void fifo()
{
initialize();
for(i=0; i<n; i++)
{
printf("\n For %d :",in[i]);
if(isHit(in[i])==0)
{
for(k=0; k<nf-1; k++)
p[k]=p[k+1];
p[k]=in[i];
pgfaultcnt++;
dispPages();
}
else
printf("No page fault");
}
dispPgFaultCnt();
}
void optimal()
{
initialize();
int near[50];
```

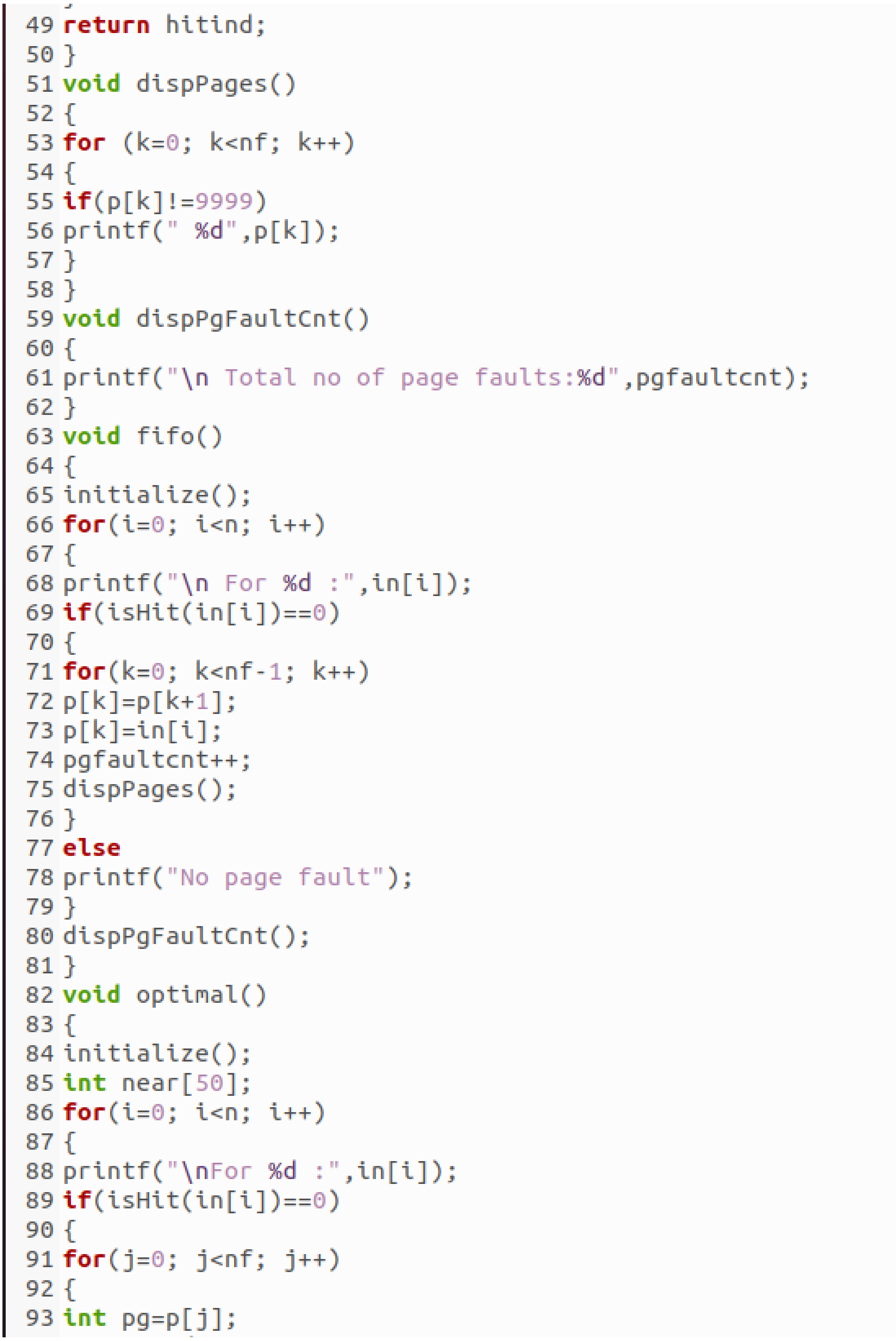
```
for(i=0; i<n; i++)
{
printf("\nFor %d :",in[i]);
if(isHit(in[i])==0)
{
for(j=0; j<nf; j++)
{
int pg=p[j];
int found=0;
for(k=i; k<n; k++)
{
if(pg==in[k])
{
near[j]=k;
found=1;
break;
}
else
found=0;
}
if(!found)
near[j]=9999;
}
int max=-9999;
int repindex;
for(j=0; j<nf; j++)
{
if(near[j]>max)
{
max=near[j];
repindex=j;
}
}
p[repindex]=in[i];
pgfaultcnt++;
dispPages();
}
```

```
else
printf("No page fault");
}
dispPgFaultCnt();
}
void lru()
{
initialize();
int least[50];
for(i=0; i<n; i++)
{
printf("\n For %d :",in[i]);
if(isHit(in[i])==0)
{
for(j=0; j<nf; j++)
{
int pg=p[j];
int found=0;
for(k=i-1; k>=0; k--)
{
if(pg==in[k])
{
least[j]=k;
found=1;
break;
}
else
found=0;
}
if(!found)
least[j]=-9999;
}
int min=9999;
int repindex;
for(j=0; j<nf; j++)
{
if(least[j]<min)
{
min=least[j];
repindex=j;
}
}
p[repindex]=in[i];
pgfaultcnt++;
dispPages();
}
else
printf("No page fault!");
}
dispPgFaultCnt();
}
int main()
{
int choice;
while(1)
{
printf("Page Replacement Algorithms \n 1.Enter data \n 2.FIFO \n
3.Optimal \n 4.LRU \n Enter your choice:");
scanf("%d",&choice);
switch(choice)
{
case 1:
getData();
break;
case 2:
fifo();
break;
case 3:
optimal();
break;
case 4:
lru();
break;
default:
```

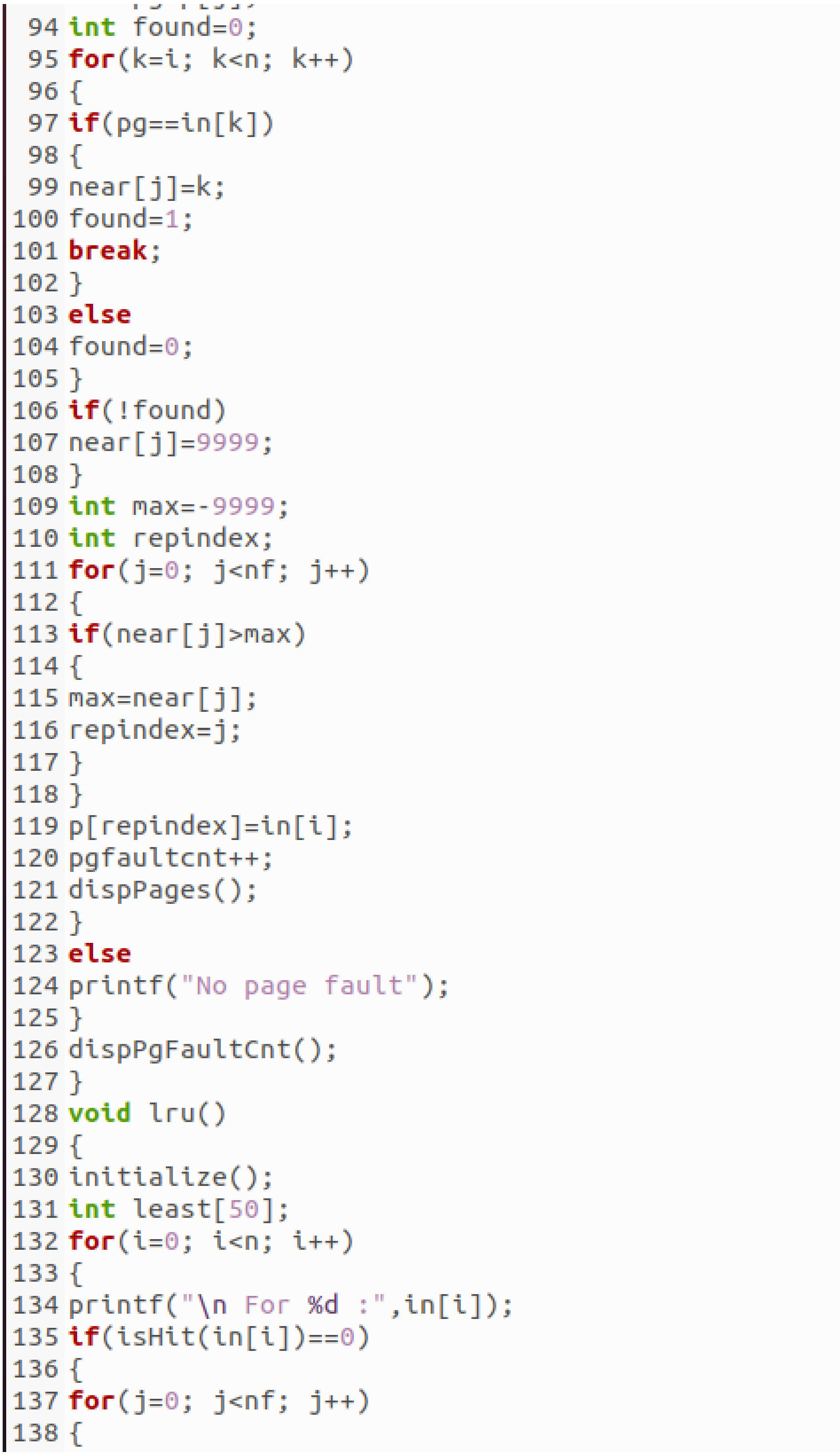
```
return 0;
break;
}
}
}
```



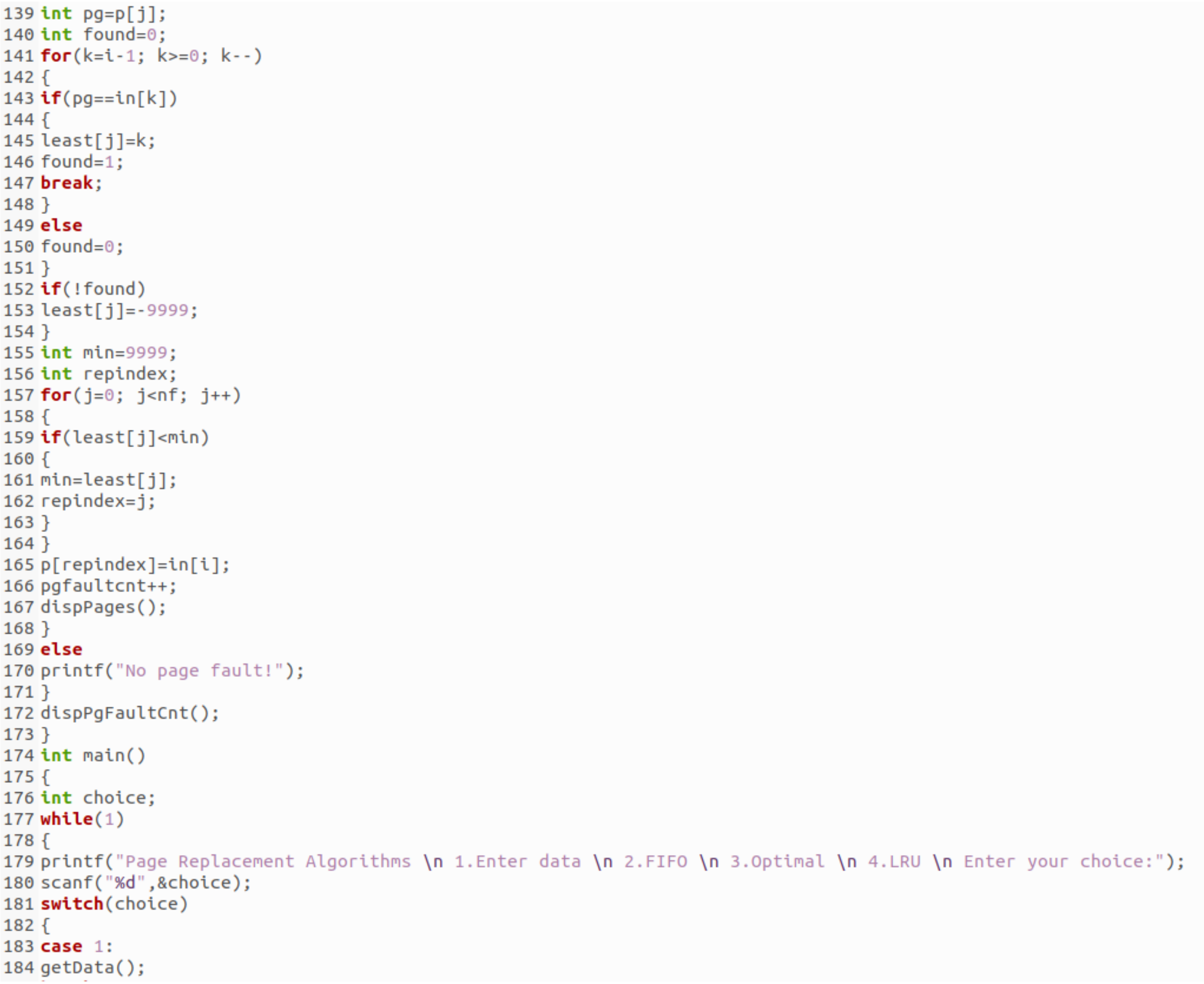
```c
//page replacement
#include<stdio.h>
int n,nf;
int in[100];
int p[50];
int hit=0;
int i,j,k;
int pgfaultcnt=0;
void getData()
{
printf("\n Enter length of page reference sequence:");
scanf("%d",&n);
printf("\n Enter the page reference sequence:");
for(i=0; i<n; i++)
scanf("%d",&in[i]);
printf("\n Enter no of frames:");
scanf("%d",&nf);
}
void initialize()
{
pgfaultcnt=0;
for(i=0; i<nf; i++)
p[i]=9999;
}
int isHit(int data)
{
hit=0;
for(j=0; j<nf; j++)
{
if(p[j]==data)
{
hit=1;
break;
}
}
return hit;
}
int getHitIndex(int data)
{
int hitind;
for(k=0; k<nf; k++)
{
if(p[k]==data)
{
hitind=k;
break;
}
}
```



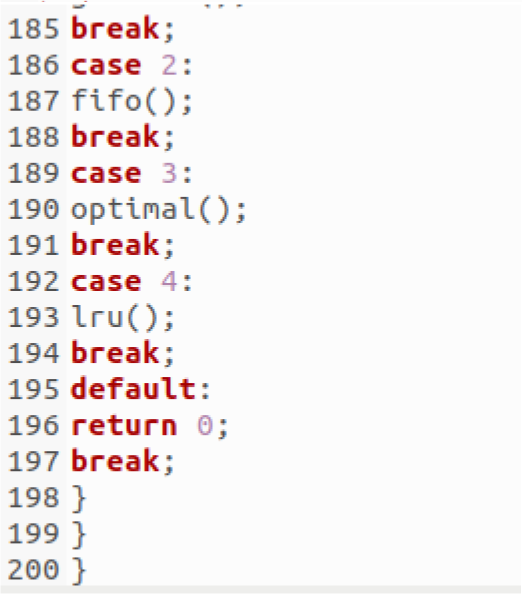
```
return hitind;
}
void dispPages()
{
for (k=0; k<nf; k++)
{
if(p[k]!=9999)
printf(" %d",p[k]);
}
}
void dispPgFaultCnt()
{
printf("\n Total no of page faults:%d",pgfaultcnt);
}
void fifo()
{
initialize();
for(i=0; i<n; i++)
{
printf("\n For %d :",in[i]);
if(isHit(in[i])==0)
{
for(k=0; k<nf-1; k++)
p[k]=p[k+1];
p[k]=in[i];
pgfaultcnt++;
dispPages();
}
else
printf("No page fault");
}
dispPgFaultCnt();
}
void optimal()
{
initialize();
int near[50];
for(i=0; i<n; i++)
{
printf("\nFor %d :",in[i]);
if(isHit(in[i])==0)
{
for(j=0; j<nf; j++)
{
int pg=p[j];
```



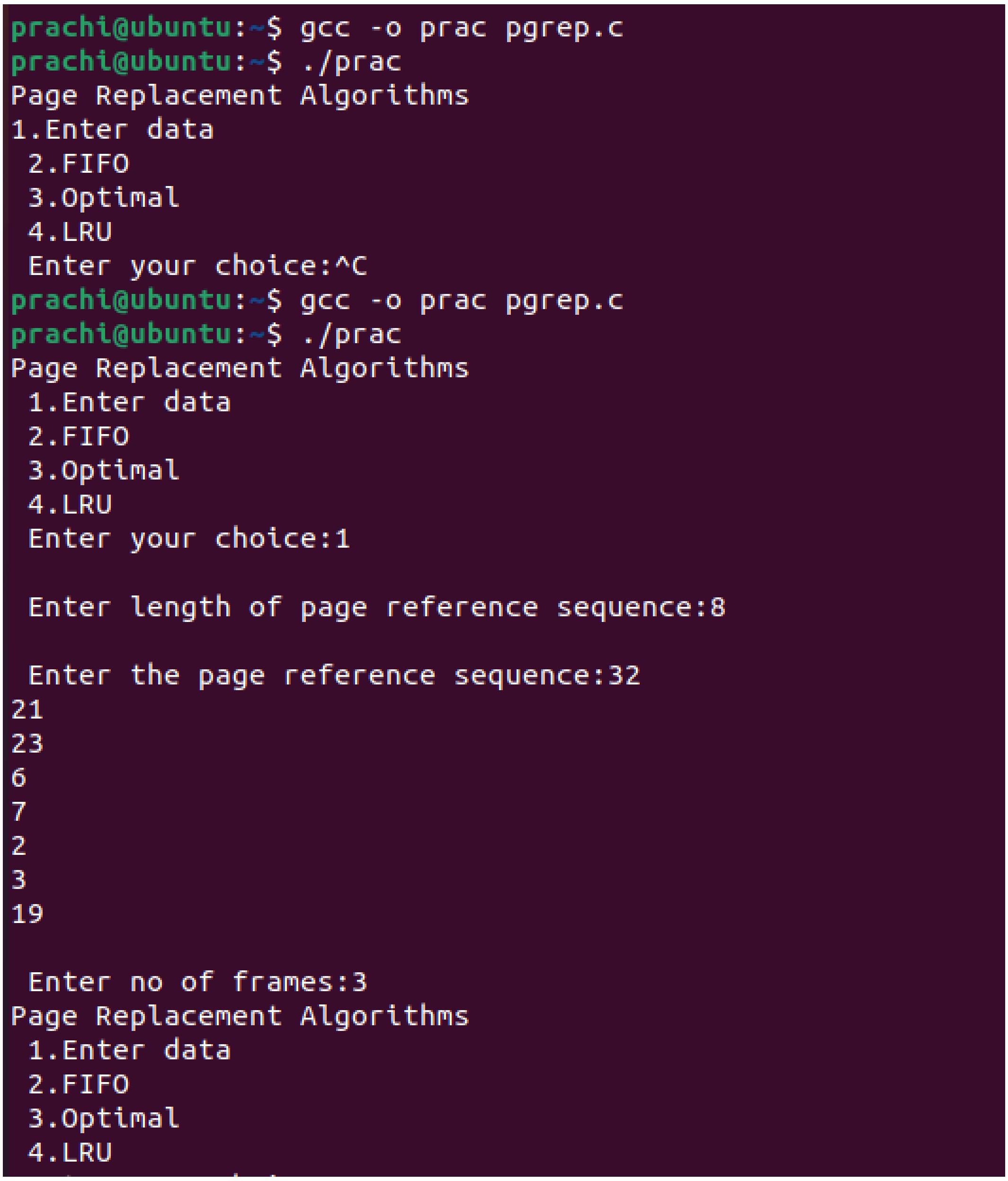
```c
int found=0;
for(k=i; k<n; k++)
{
if(pg==in[k])
{
near[j]=k;
found=1;
break;
}
else
found=0;
}
if(!found)
near[j]=9999;
}
int max=-9999;
int repindex;
for(j=0; j<nf; j++)
{
if(near[j]>max)
{
max=near[j];
repindex=j;
}
}
p[repindex]=in[i];
pgfaultcnt++;
dispPages();
}
else
printf("No page fault");
}
dispPgFaultCnt();
}
void lru()
{
initialize();
int least[50];
for(i=0; i<n; i++)
{
printf("\n For %d :",in[i]);
if(isHit(in[i])==0)
{
for(j=0; j<nf; j++)
{
```



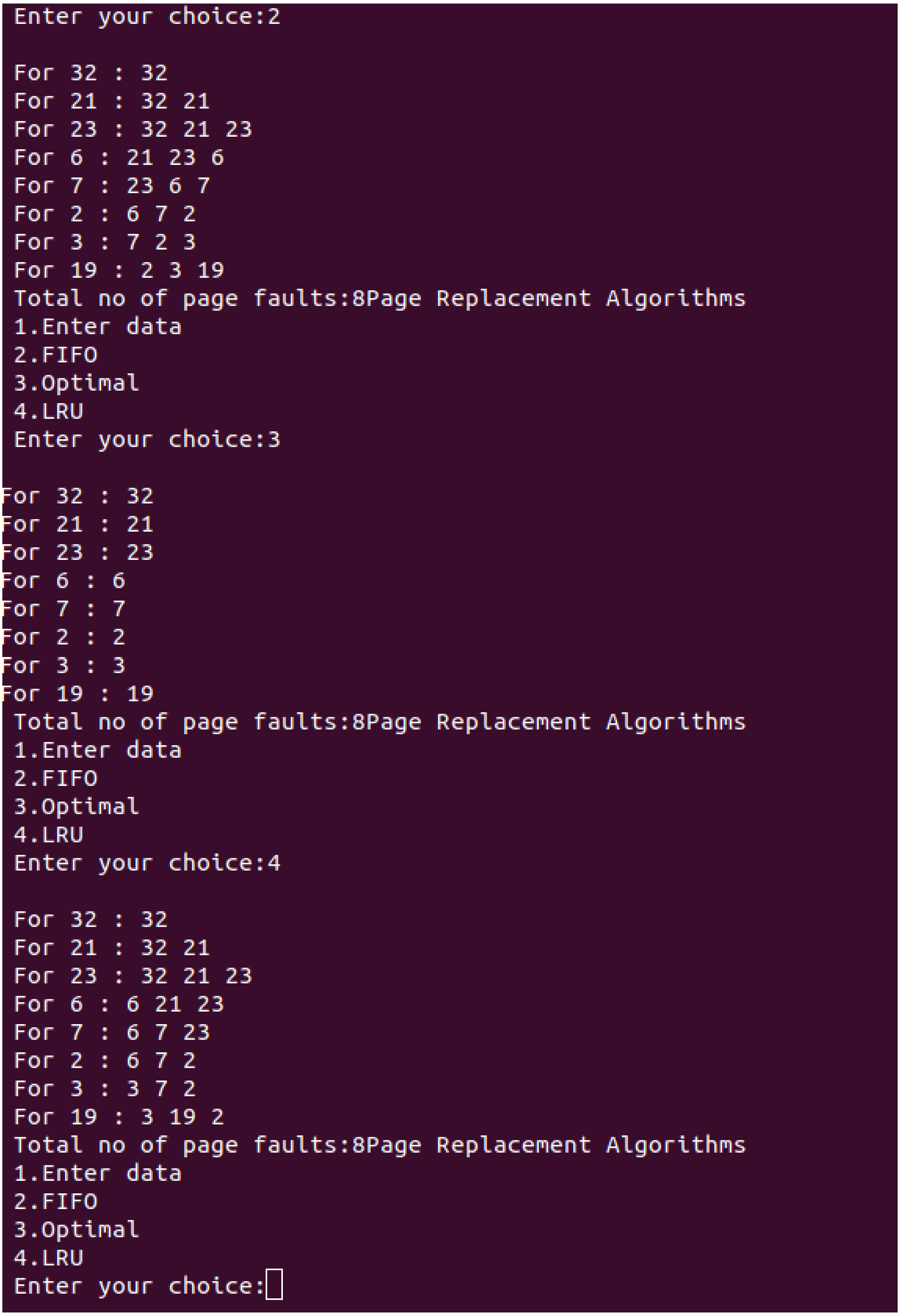
```
int pg=p[j];
int found=0;
for(k=i-1; k>=0; k--)
{
if(pg==in[k])
{
least[j]=k;
found=1;
break;
}
else
found=0;
}
if(!found)
least[j]=-9999;
}
int min=9999;
int repindex;
for(j=0; j<nf; j++)
{
if(least[j]<min)
{
min=least[j];
repindex=j;
}
}
p[repindex]=in[i];
pgfaultcnt++;
dispPages();
}
else
printf("No page fault!");
}
dispPgFaultCnt();
}
int main()
{
int choice;
while(1)
{
printf("Page Replacement Algorithms \n 1.Enter data \n 2.FIFO \n 3.Optimal \n 4.LRU \n Enter your choice:");
scanf("%d",&choice);
switch(choice)
{
case 1:
getData();
```



```
break;
case 2:
fifo();
break;
case 3:
optimal();
break;
case 4:
lru();
break;
default:
return 0;
break;
}
}
}
```



```
prachi@ubuntu:~$ gcc -o prac pgrep.c
prachi@ubuntu:~$ ./prac
Page Replacement Algorithms
1.Enter data
 2.FIFO
 3.Optimal
 4.LRU
 Enter your choice:^C
prachi@ubuntu:~$ gcc -o prac pgrep.c
prachi@ubuntu:~$ ./prac
Page Replacement Algorithms
 1.Enter data
 2.FIFO
 3.Optimal
 4.LRU
 Enter your choice:1

 Enter length of page reference sequence:8

 Enter the page reference sequence:32
21
23
6
7
2
3
19

 Enter no of frames:3
Page Replacement Algorithms
 1.Enter data
 2.FIFO
 3.Optimal
 4.LRU
```



```
Enter your choice:2

For 32 : 32
For 21 : 32 21
For 23 : 32 21 23
For 6 : 21 23 6
For 7 : 23 6 7
For 2 : 6 7 2
For 3 : 7 2 3
For 19 : 2 3 19
Total no of page faults:8Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
Enter your choice:3

For 32 : 32
For 21 : 21
For 23 : 23
For 6 : 6
For 7 : 7
For 2 : 2
For 3 : 3
For 19 : 19
Total no of page faults:8Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
Enter your choice:4

For 32 : 32
For 21 : 32 21
For 23 : 32 21 23
For 6 : 6 21 23
For 7 : 6 7 23
For 2 : 6 7 2
For 3 : 3 7 2
For 19 : 3 19 2
Total no of page faults:8Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
Enter your choice:
```

3. Classical problem of synchronization

(i). Producer Consumer Problem

Algorithm:

The producer-consumer problem is an example of a multi-process synchronization problem. The problem describes two processes, the producer and the consumer that shares a common fixed-size buffer use it as a queue.

The producer’s job is to generate data, put it into the buffer, and start again.

At the same time, the consumer is consuming the data (i.e., removing it from the buffer), one piece at a time.

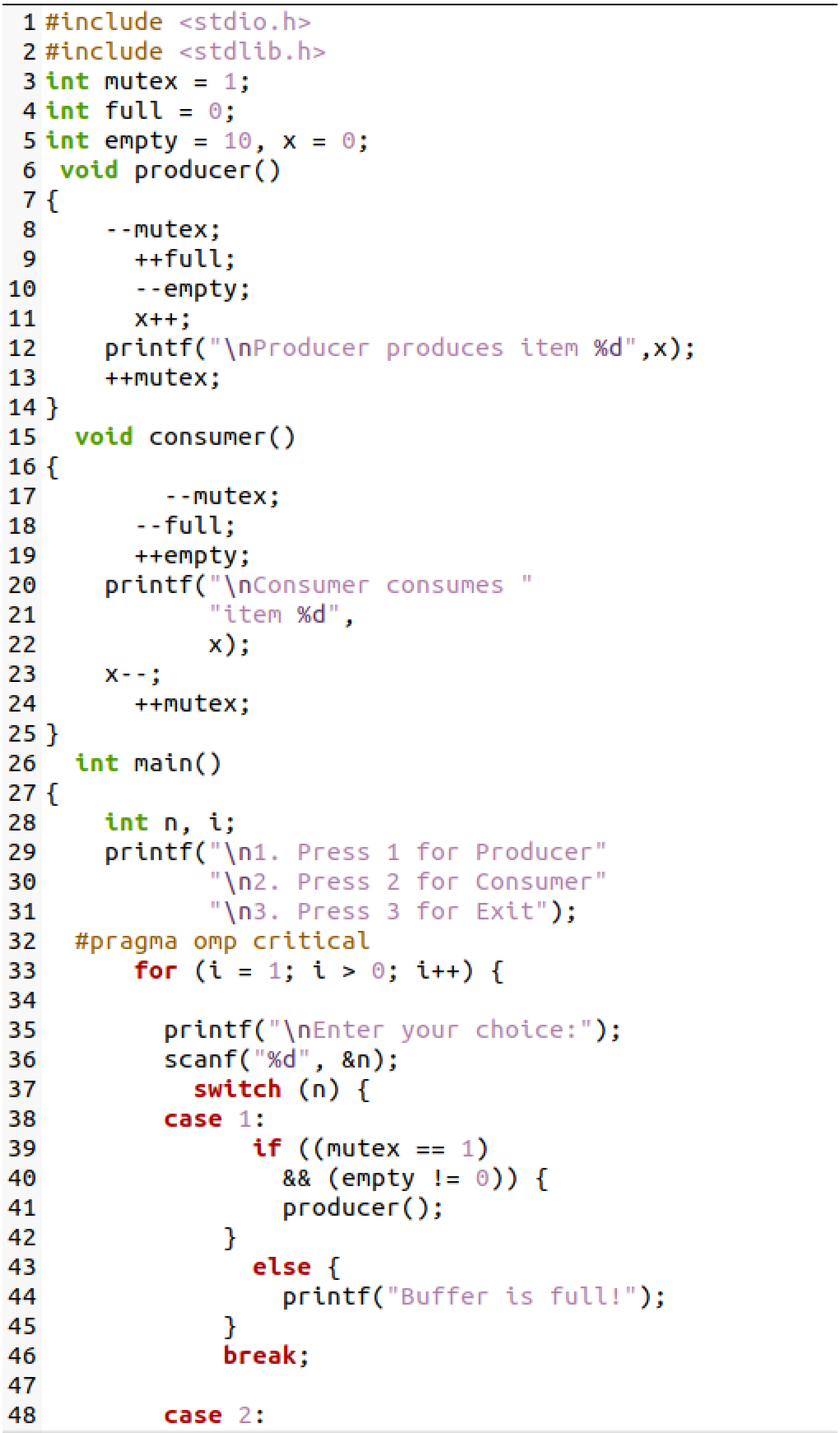
The producer is to either go to sleep or discard data if the buffer is full. The next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again.

In the same manner, the consumer can go to sleep if it finds the buffer to be empty. The next time the producer puts data into the buffer, it wakes up the sleeping consumer.

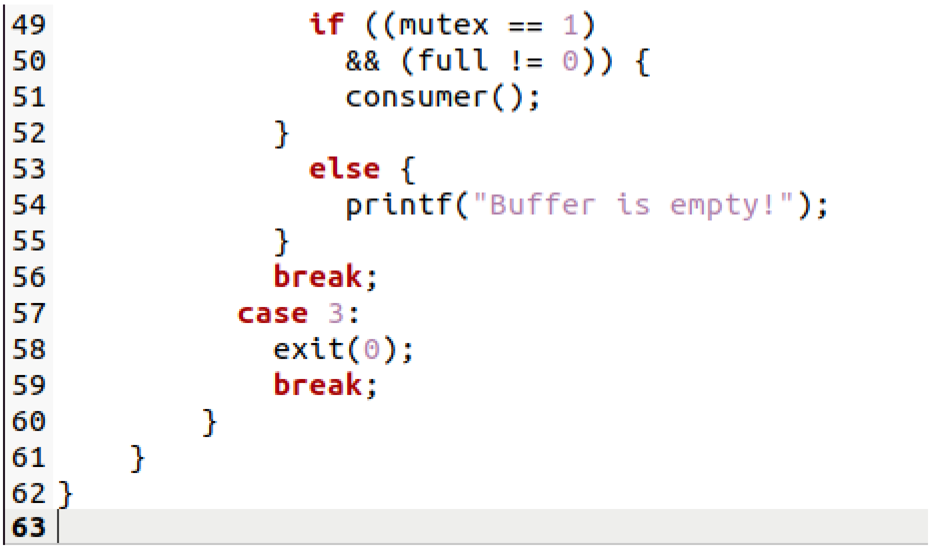
Code:

```
#include <stdio.h>
#include <stdlib.h>
int mutex = 1;
int full = 0;
int empty = 10, x = 0;
 void producer()
{
   --mutex;
     ++full;
     --empty;
     x++;
   printf("\nProducer produces item %d",x);
   ++mutex;
}
  void consumer()
{
       --mutex;
     --full;
     ++empty;
   printf("\nConsumer consumes "
        "item %d",
        x);
  x--;
   ++mutex;
}
 int main()
{
  int n, i;
  printf("\n1. Press 1 for Producer"
      "\n2. Press 2 for Consumer"
      "\n3. Press 3 for Exit");
 #pragma omp critical
   for (i = 1; i > 0; i++) {

    printf("\nEnter your choice:");
    scanf("%d", &n);
     switch (n) {
    case 1:
       if ((mutex == 1)
        && (empty != 0)) {
        producer();
      }
       else {
        printf("Buffer is full!");
      }
      break;

    case 2:
       if ((mutex == 1)
        && (full != 0)) {
        consumer();
      }
       else {
        printf("Buffer is empty!");
      }
      break;
     case 3:
      exit(0);
      break;
```

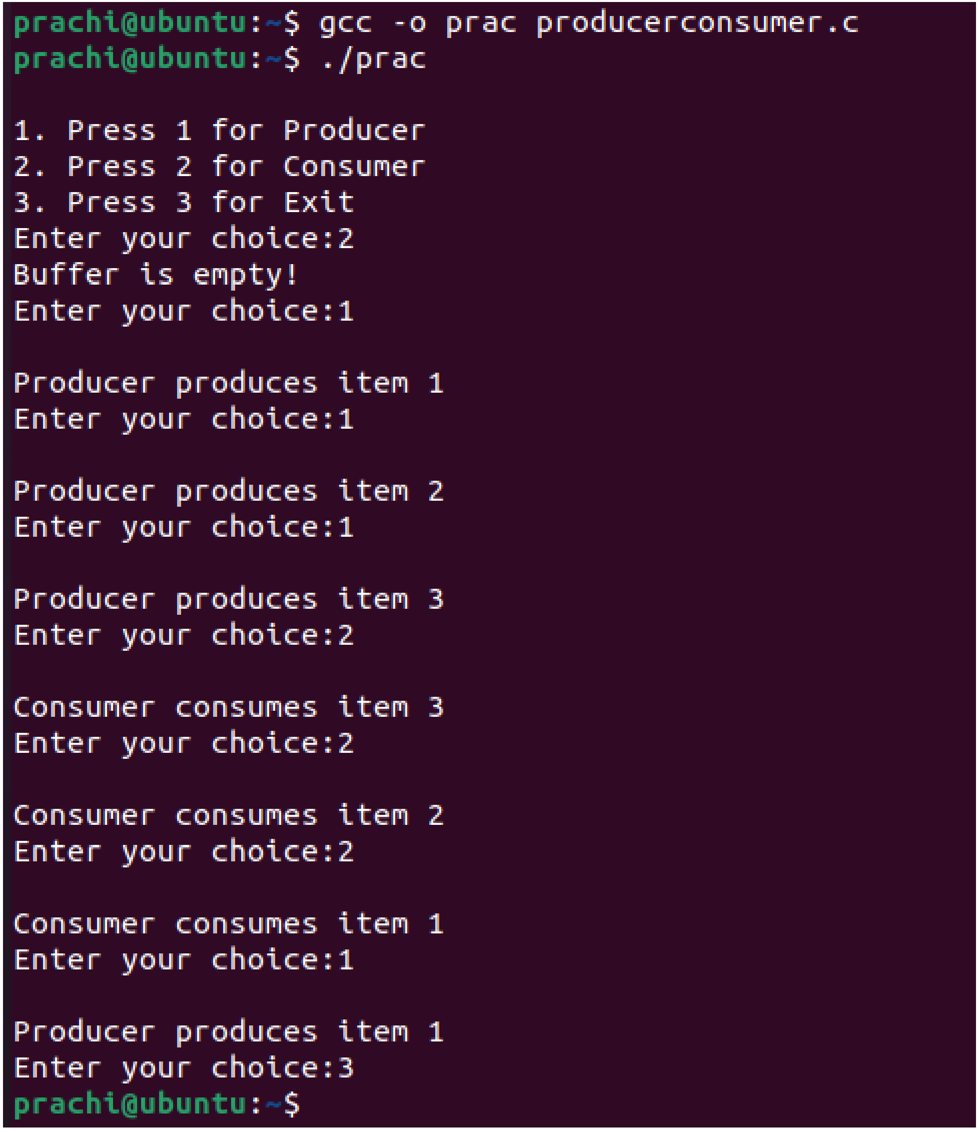
```
        }
    }
}
```



```
#include <stdio.h>
#include <stdlib.h>
int mutex = 1;
int full = 0;
int empty = 10, x = 0;
 void producer()
{
    --mutex;
      ++full;
      --empty;
      x++;
    printf("\nProducer produces item %d",x);
    ++mutex;
}
  void consumer()
{
        --mutex;
      --full;
      ++empty;
    printf("\nConsumer consumes "
           "item %d",
           x);
    x--;
      ++mutex;
}
  int main()
{
    int n, i;
    printf("\n1. Press 1 for Producer"
           "\n2. Press 2 for Consumer"
           "\n3. Press 3 for Exit");
  #pragma omp critical
      for (i = 1; i > 0; i++) {

        printf("\nEnter your choice:");
        scanf("%d", &n);
          switch (n) {
        case 1:
              if ((mutex == 1)
                && (empty != 0)) {
                producer();
            }
              else {
                printf("Buffer is full!");
            }
            break;

        case 2:
```



```
       if ((mutex == 1)
        && (full != 0)) {
        consumer();
      }
       else {
        printf("Buffer is empty!");
      }
      break;
     case 3:
      exit(0);
      break;
    }
  }
}
```



```
prachi@ubuntu:~$ gcc -o prac producerconsumer.c
prachi@ubuntu:~$ ./prac

1. Press 1 for Producer
2. Press 2 for Consumer
3. Press 3 for Exit
Enter your choice:2
Buffer is empty!
Enter your choice:1

Producer produces item 1
Enter your choice:1

Producer produces item 2
Enter your choice:1

Producer produces item 3
Enter your choice:2

Consumer consumes item 3
Enter your choice:2

Consumer consumes item 2
Enter your choice:2

Consumer consumes item 1
Enter your choice:1

Producer produces item 1
Enter your choice:3
prachi@ubuntu:~$
```

(ii). Reader Writer Problem(Semaphore)

Algorithm:

One set of data is shared among a number of processes.

Once a writer is ready, it performs its write. Only one writer may write at a time.

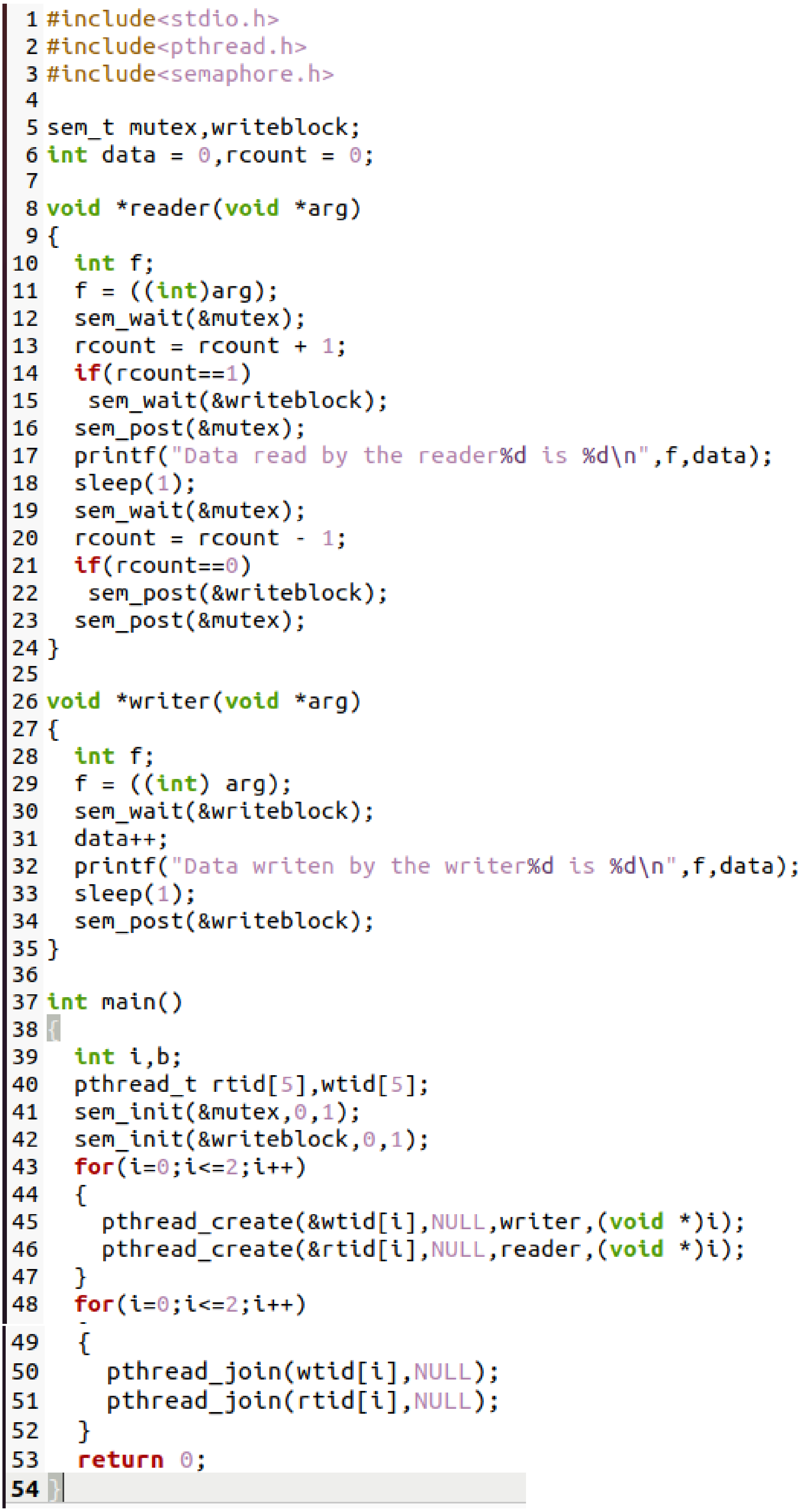
If a process is writing, no other process can read it.

If at least one reader is reading, no other process can write.

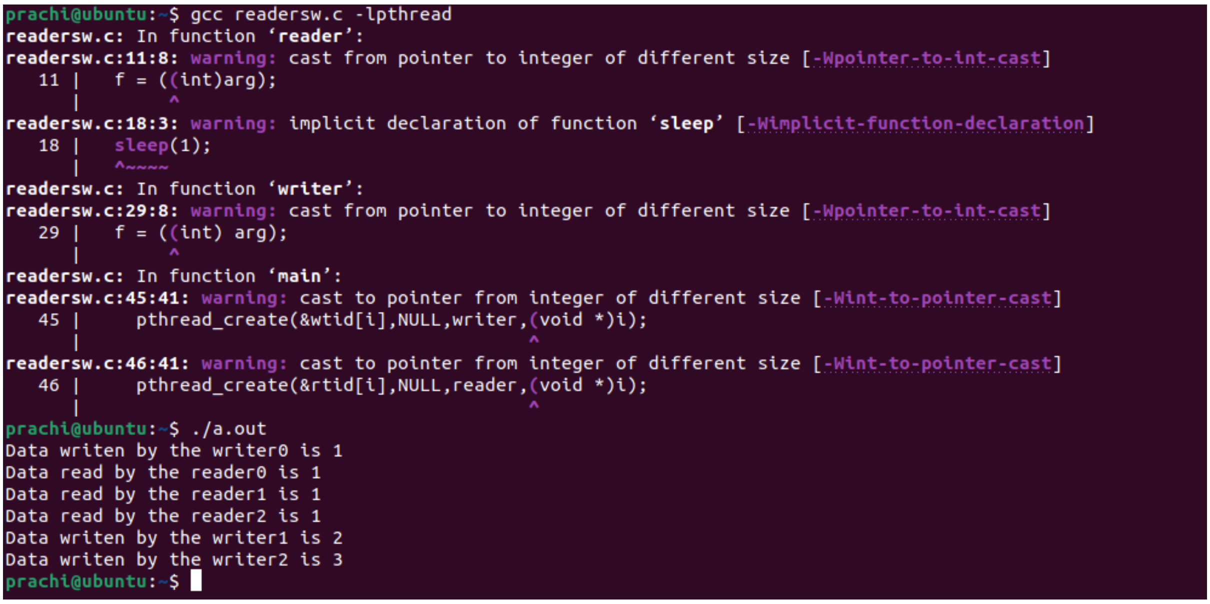
Readers may not write and only read.

Code:

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

sem_t mutex,writeblock;
int data = 0,rcount = 0;

void *reader(void *arg)
{
  int f;
  f = ((int)arg);
  sem_wait(&mutex);
  rcount = rcount + 1;
  if(rcount==1)
   sem_wait(&writeblock);
  sem_post(&mutex);
  printf("Data read by the reader%d is %d\n",f,data);
  sleep(1);
  sem_wait(&mutex);
  rcount = rcount - 1;
  if(rcount==0)
   sem_post(&writeblock);
  sem_post(&mutex);
}

void *writer(void *arg)
{
  int f;
  f = ((int) arg);
  sem_wait(&writeblock);
  data++;
  printf("Data writen by the writer%d is %d\n",f,data);
  sleep(1);
  sem_post(&writeblock);
}

int main()
{
  int i,b;
  pthread_t rtid[5],wtid[5];
  sem_init(&mutex,0,1);
  sem_init(&writeblock,0,1);
  for(i=0;i<=2;i++)
  {
    pthread_create(&wtid[i],NULL,writer,(void *)i);
    pthread_create(&rtid[i],NULL,reader,(void *)i);
  }
  for(i=0;i<=2;i++)
  {
    pthread_join(wtid[i],NULL);
    pthread_join(rtid[i],NULL);
  }
  return 0;
}
```



```c
#include<stdio.h>
#include<pthread.h>
#include<semaphore.h>

sem_t mutex,writeblock;
int data = 0,rcount = 0;

void *reader(void *arg)
{
  int f;
  f = ((int)arg);
  sem_wait(&mutex);
  rcount = rcount + 1;
  if(rcount==1)
   sem_wait(&writeblock);
  sem_post(&mutex);
  printf("Data read by the reader%d is %d\n",f,data);
  sleep(1);
  sem_wait(&mutex);
  rcount = rcount - 1;
  if(rcount==0)
   sem_post(&writeblock);
  sem_post(&mutex);
}

void *writer(void *arg)
{
  int f;
  f = ((int) arg);
  sem_wait(&writeblock);
  data++;
  printf("Data writen by the writer%d is %d\n",f,data);
  sleep(1);
  sem_post(&writeblock);
}

int main()
{
  int i,b;
  pthread_t rtid[5],wtid[5];
  sem_init(&mutex,0,1);
  sem_init(&writeblock,0,1);
  for(i=0;i<=2;i++)
  {
    pthread_create(&wtid[i],NULL,writer,(void *)i);
    pthread_create(&rtid[i],NULL,reader,(void *)i);
  }
  for(i=0;i<=2;i++)
  {
    pthread_join(wtid[i],NULL);
    pthread_join(rtid[i],NULL);
  }
  return 0;
}
```



```
prachi@ubuntu:~$ gcc readersw.c -lpthread
readersw.c: In function ‘reader’:
readersw.c:11:8: warning: cast from pointer to integer of different size [-Wpointer-to-int-cast]
   11 |    f = ((int)arg);
      |         ^
readersw.c:18:3: warning: implicit declaration of function ‘sleep’ [-Wimplicit-function-declaration]
   18 |    sleep(1);
      |    ^~~~~
readersw.c: In function ‘writer’:
readersw.c:29:8: warning: cast from pointer to integer of different size [-Wpointer-to-int-cast]
   29 |    f = ((int) arg);
      |         ^
readersw.c: In function ‘main’:
readersw.c:45:41: warning: cast to pointer from integer of different size [-Wint-to-pointer-cast]
   45 |      pthread_create(&wtid[i],NULL,writer,(void *)i);
      |                                          ^
readersw.c:46:41: warning: cast to pointer from integer of different size [-Wint-to-pointer-cast]
   46 |      pthread_create(&rtid[i],NULL,reader,(void *)i);
      |                                          ^
prachi@ubuntu:~$ ./a.out
Data writen by the writer0 is 1
Data read by the reader0 is 1
Data read by the reader1 is 1
Data read by the reader2 is 1
Data writen by the writer1 is 2
Data writen by the writer2 is 3
prachi@ubuntu:~$ 
```