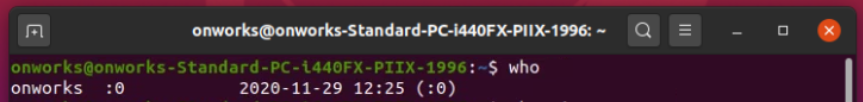
Linux Commands

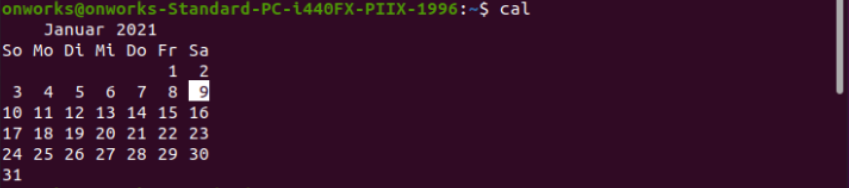
**Who:** Print all usernames currently logged in

**Whoami:** Print the current user id and name ('id -un')

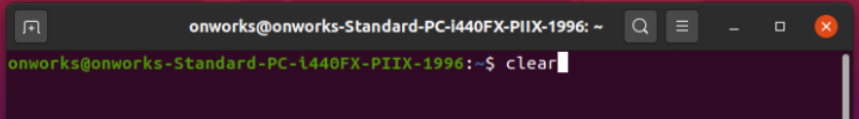


**Date:** Display or change the date & time

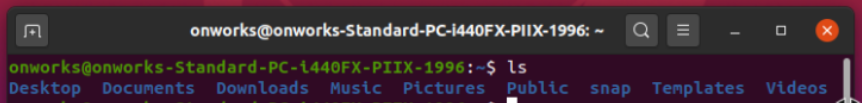
**Cal:** Display a calendar



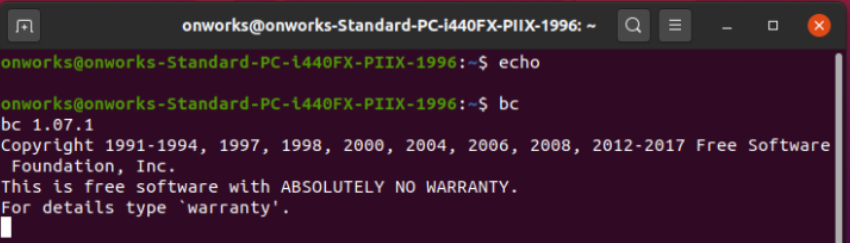
**Clear:** Clear terminal screen



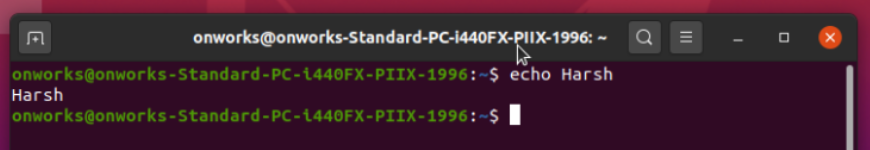
**Ls:** List information about file(s)



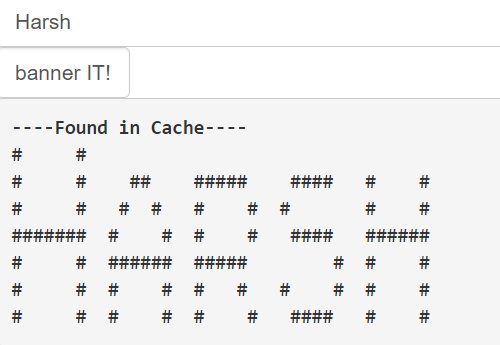
**Bc:** Arbitrary precision calculator language



**Echo:** Display message on screen



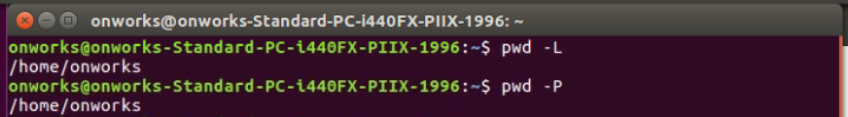
**Banner**: To print the ASCII character string in large letter to standard output.



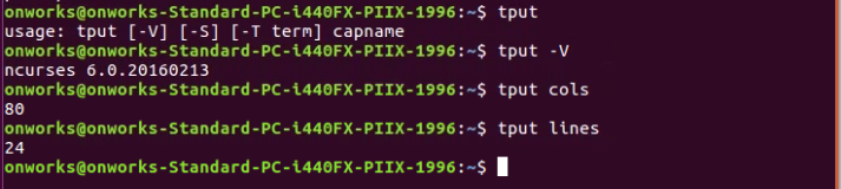
**Pwd**: pwd stands for Print Working Directory. It prints the path of the working directory, starting from the root.

**pwd -P**: Prints the actual path.

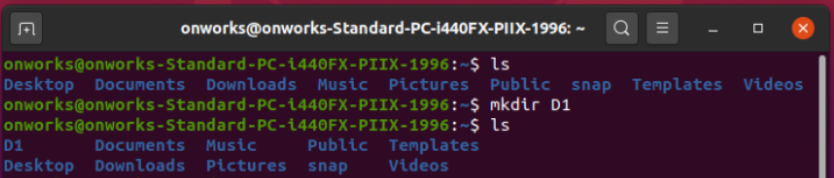
**pwd -L:** Prints the symbolic path.



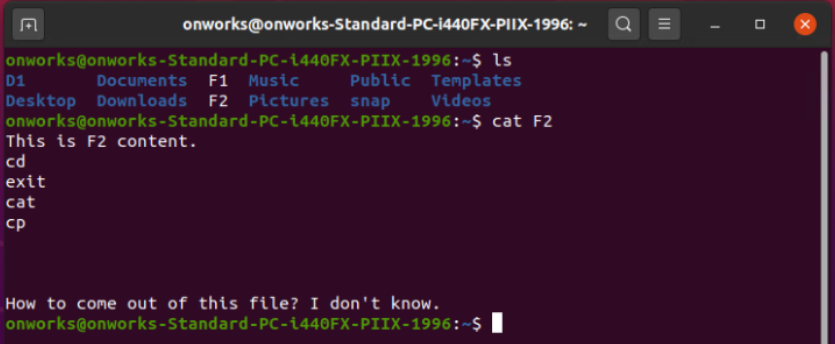
**Tput:** Set terminal-dependent capabilities, color, position traceroute Trace Route to Host



**Mkdir:** Create new folder(s)

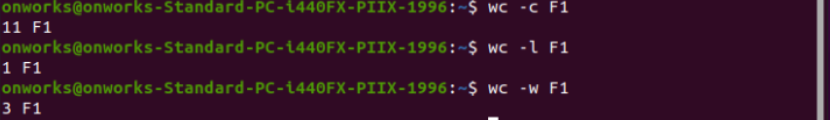


**Cat:** It reads data from the file and gives their content as output. It helps us to create, view, concatenate files.

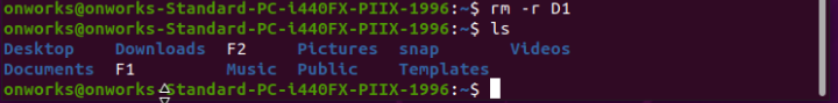


**Wc:** Print byte, word, and line counts





**Rm:** Remove files

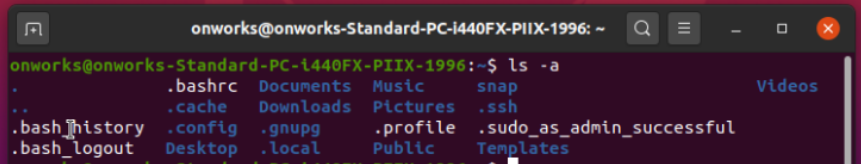


**Cd:** Change Directory



**Ls -a:** list all files including hidden file starting with '.'

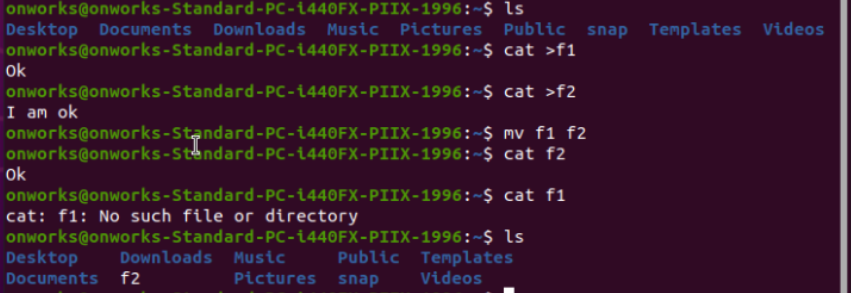




**Mv:** mv stands for move. mv is used to move one or more files or directories from one place to another in file system like UNIX. It has two distinct functions:

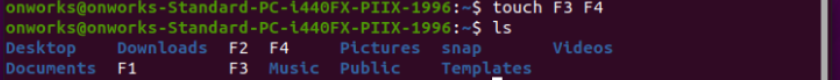
(i) It rename a file or folder.

(ii) It moves group of files to different directory.



**Cp:** cp stands for copy. This command is used to copy files or group of files or directory. It creates an exact image of a file on a disk with different file name. cp command require at least two filenames in its arguments.

**Touch:** It is used to create a file without any content. The file created using touch command is empty.



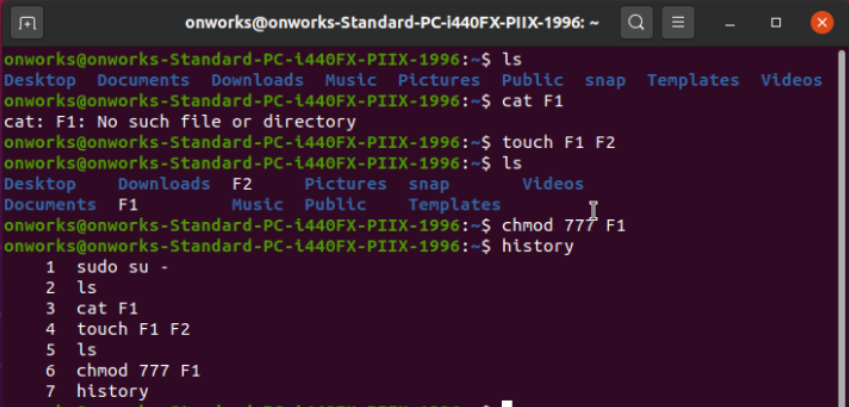
**Login:** The login command initializes the user environment from the user database, from the command line, and from the /etc/environment configuration file; changes the current directory to the user's home directory (normally); and runs the user's initial program.

**Logout:** logout command allows you to programmatically logout from your session.

**Passwd:** passwd command in Linux is used to change the user account passwords. The root user reserves the privilege to change the password for any user on the system, while a normal user can only change the account password for his or her own account.

**Exit:** exit command in linux is used to exit the shell where it is currently running.

**History:** history command is used to view the previously executed command.



**Chmod**: chmod is the command and system call which is used to change the access permissions of file system objects.

**Types of users:**

1. User/Owner represented by ‘u’
2. Group of People represented by ‘g’
3. Others/ public represented by ‘o’

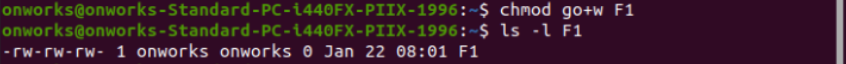
**Types of Permission:**

1.Read represented by ‘r’

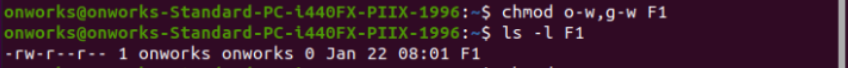
2. Write represented by ‘w’

3. Execute represented by ‘x’

**Syntax:** chmod [user category] [+/-] [Permission type] [File name]









**In number mode:**

1. Read is represented by ‘4’
2. Write is represented by ‘2’
3. Execute is represented by ‘1’

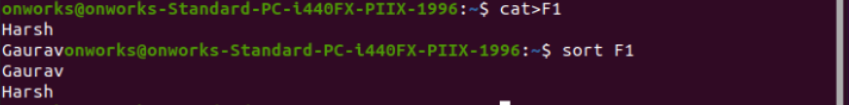
**Places of Significance:** chmod [User][Group of People][Public]



**Sort:** sort command is used to sort a file, arranging the records in a particular order. By default, the sort command sorts file assuming the contents are ASCII.

**-o Option :** Unix also provides us with special facilities like if you want to write the output to a new file, output.txt, redirects the output like this or you can also use the built-in sort option -o, which allows you to specify an output file.Using the -o option is functionally the same as redirecting the output to a file.

**-r Option:** Sorting In Reverse Order : You can perform a reverse-order sort using the -r flag. the -r flag is an option of the sort command which sorts the input file in reverse order i.e. descending order by default.



**Grep:** The grep filter searches a file for a particular pattern of characters, and displays all lines that contain that pattern.

**-c :** This prints only a count of the lines that match a pattern

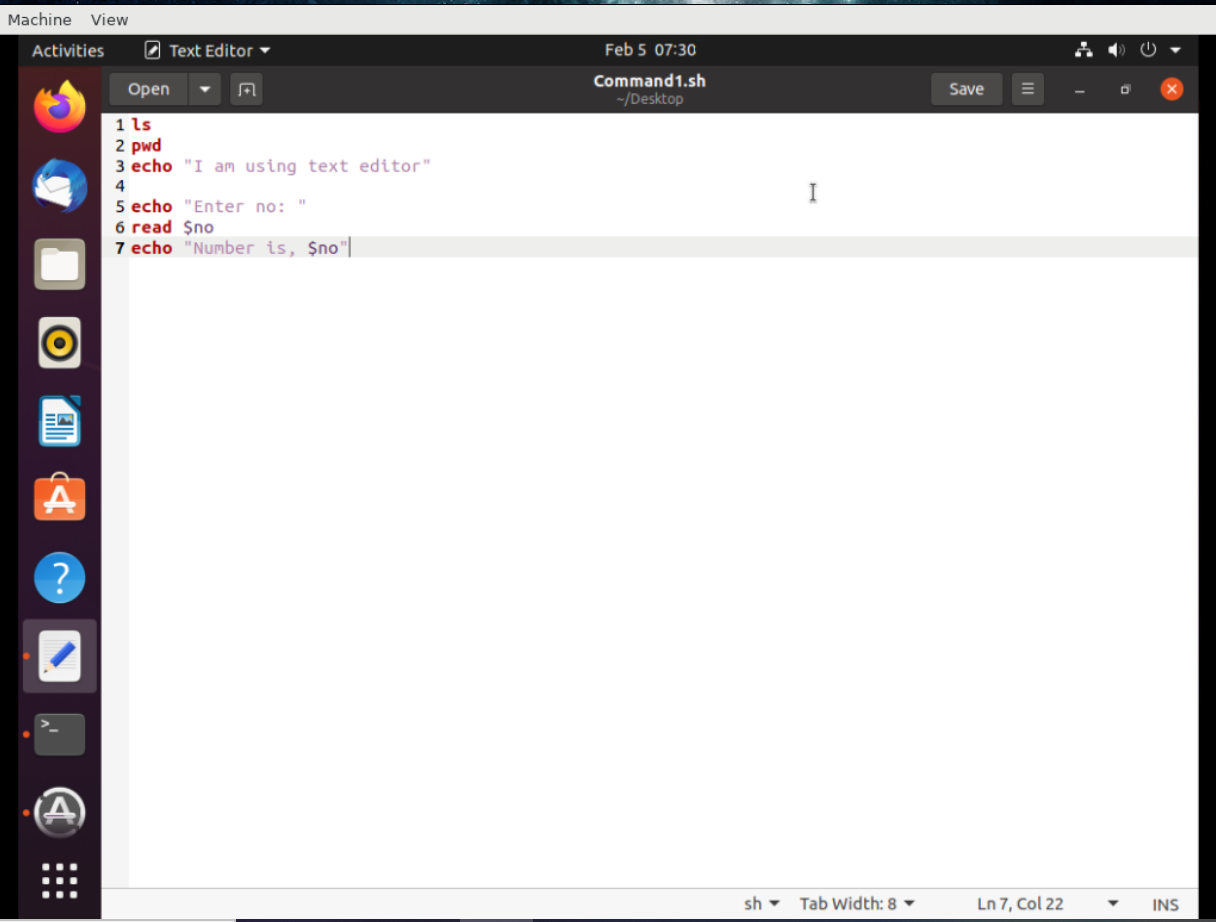
**-h :** Display the matched lines, but do not display the filenames.

**-i :** Ignores, case for matching



Shell Scripting

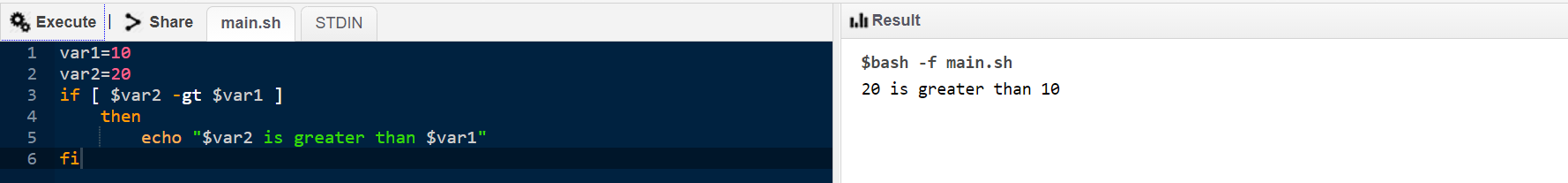
**Program 1:** Echoing Number



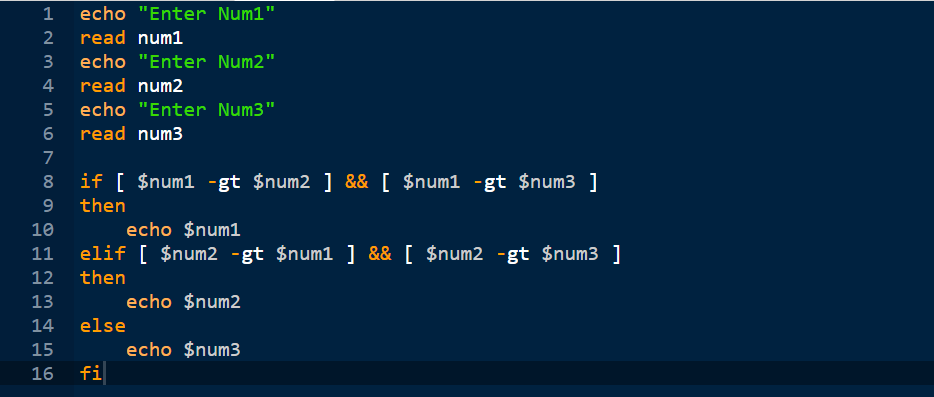


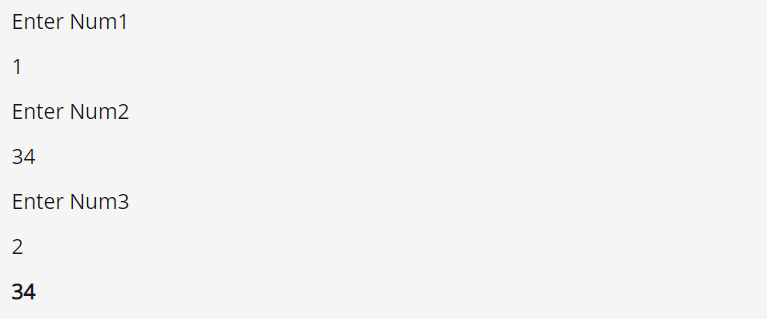


**Program 2:** Compare the value of 2 numbers

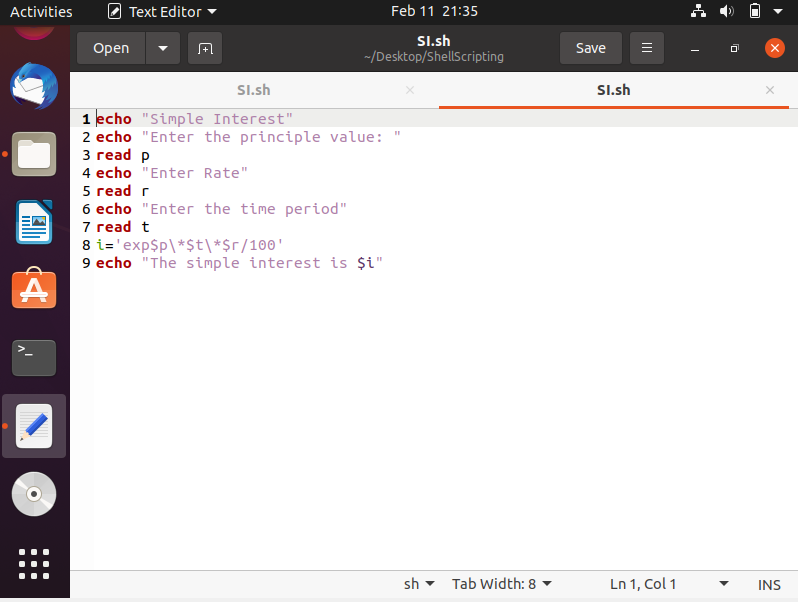


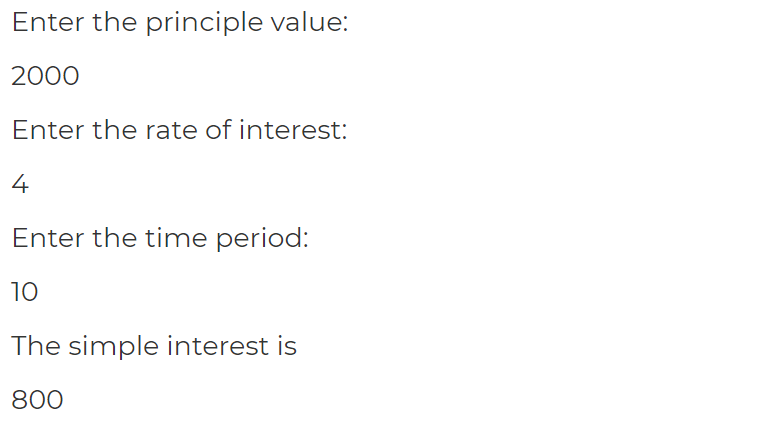
**Program 3:** Compare the value of 3 numbers





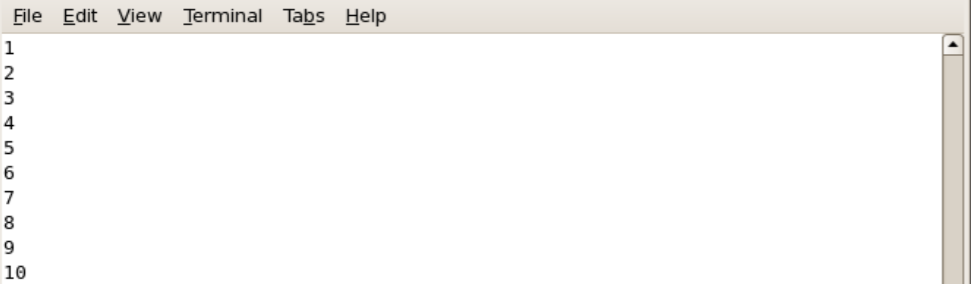
**Program 4:** Calculate simple interest



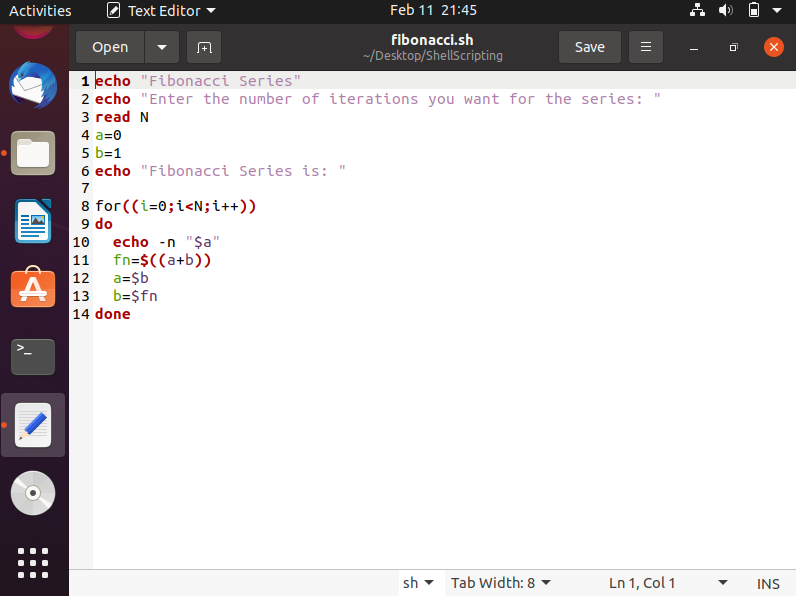


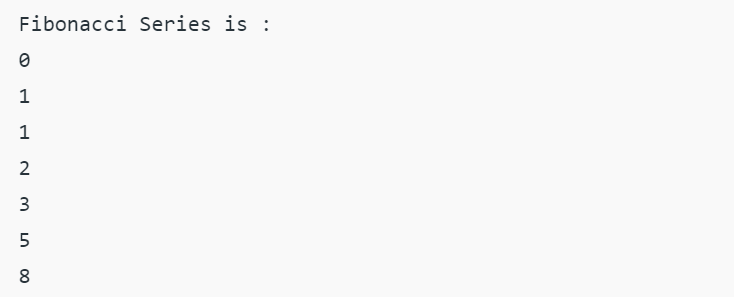
**Program 5:** Print the first 10 natural numbers using For loop



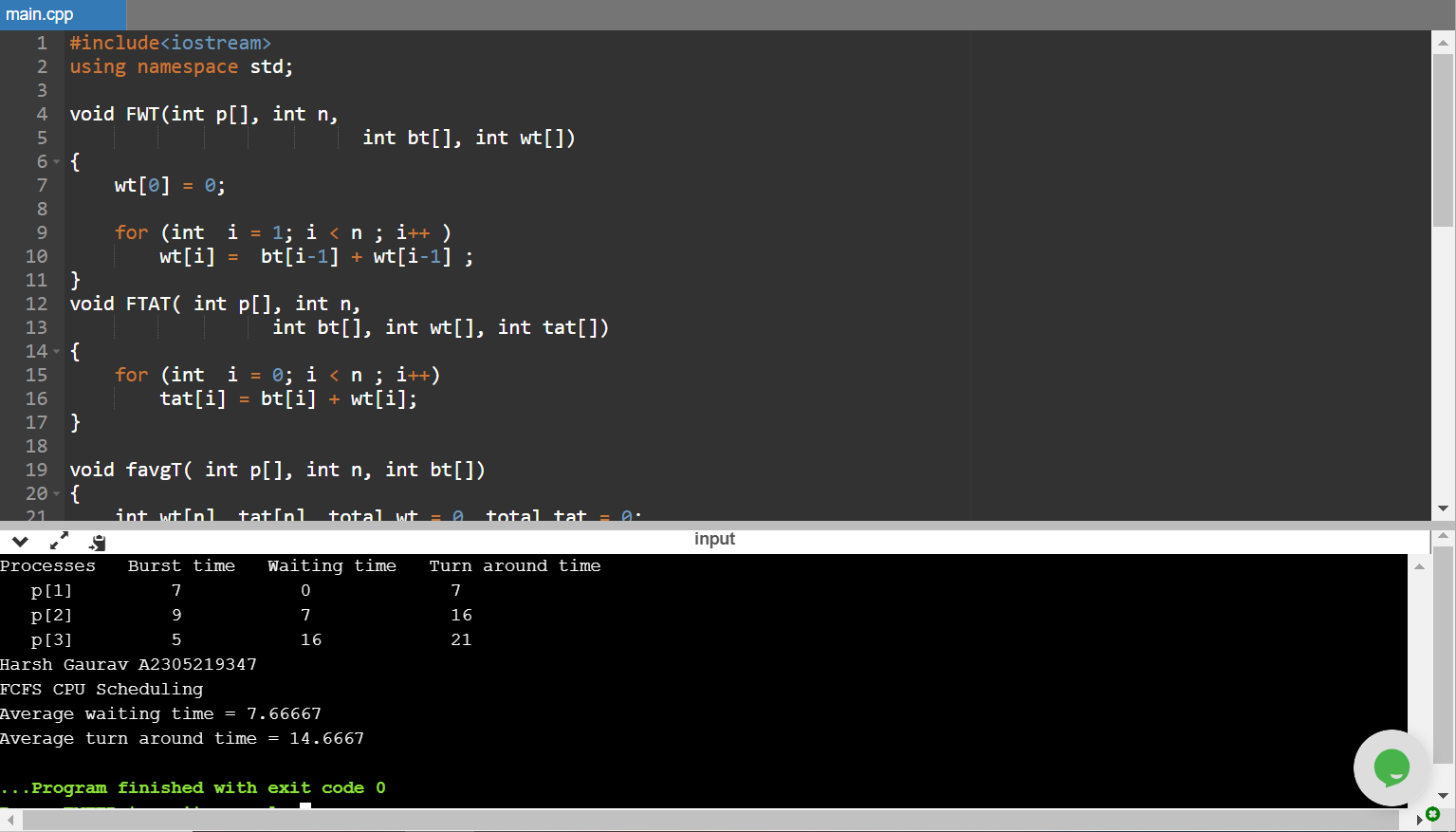


**Program 6:** Generate Fibonacci series





FCFS CPU Scheduling



#include<iostream>

using namespace std;

void FWT(int p[], int n,

int bt[], int wt[])

{

wt[0] = 0;

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

wt[i] = bt[i-1] + wt[i-1] ;

}

void FTAT( int p[], int n,

int bt[], int wt[], int tat[])

{

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

tat[i] = bt[i] + wt[i];

}

void favgT( int p[], int n, int bt[])

{

int wt[n], tat[n], total\_wt = 0, total\_tat = 0;

FWT(p, n, bt, wt);

FTAT(p, n, bt, wt, tat);

cout << "Processes "<< " Burst time "

<< " Waiting time " << " Turn around time\n";

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

{

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

cout << " p[" << i+1 << "]\t\t" << bt[i] <<"\t "

<< wt[i] <<"\t\t " << tat[i] <<endl;

}

cout<<"Harsh Gaurav A2305219347 \nFCFS CPU Scheduling\n";

cout << "Average waiting time = "

<< (float)total\_wt / (float)n;

cout << "\nAverage turn around time = "

<< (float)total\_tat / (float)n;

}

int main()

{

int p[] = { 1, 2, 3};

int n = sizeof p / sizeof p[0];

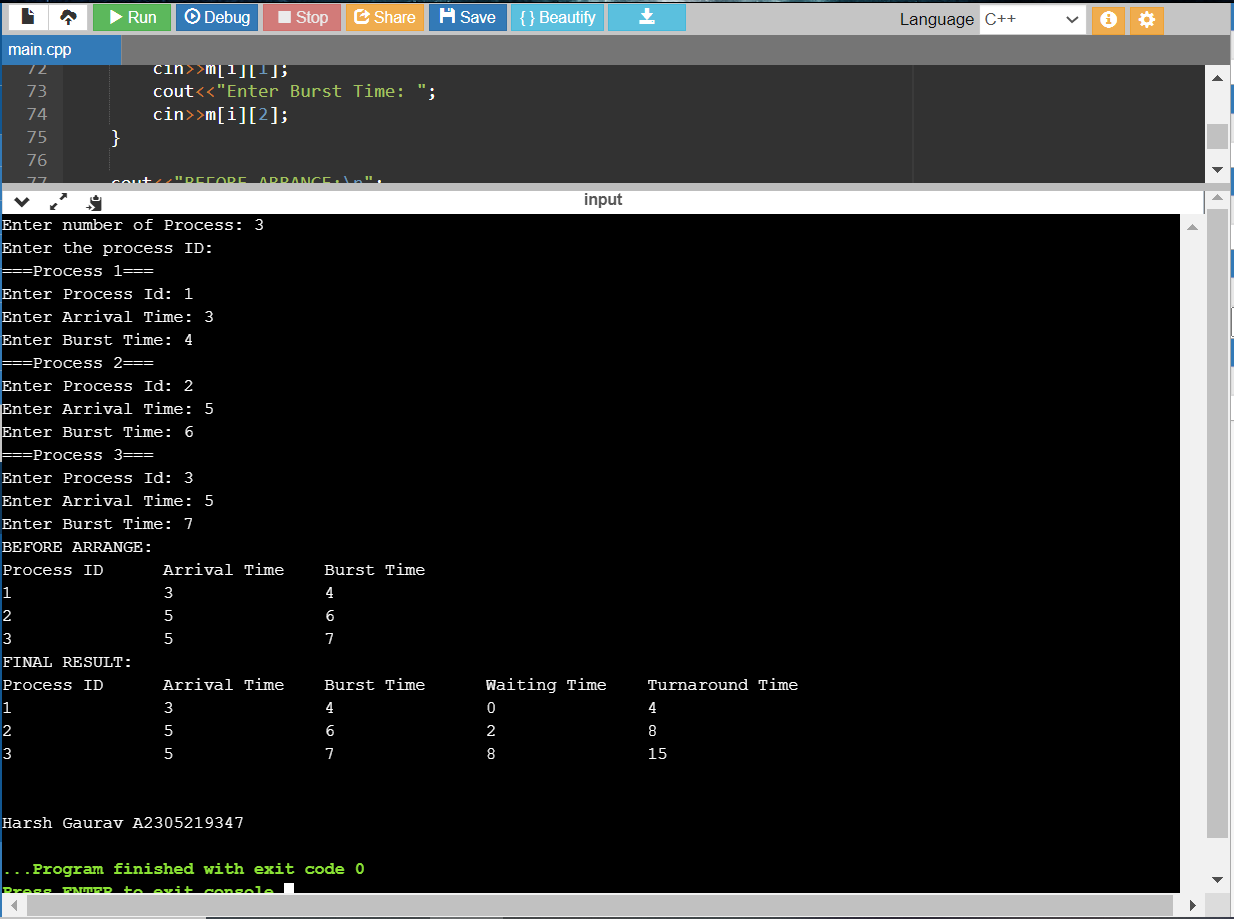
int burst\_time[] = {7 , 9, 5};

favgT(p, n, burst\_time);

return 0;

}

Shortest Job First (SJF)



#include<iostream>

using namespace std;

int m[10][6];

void swap(int \*a, int \*b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void arrangeArrival(int num, int m[][6])

{

for(int i=0; i<num; i++)

{

for(int j=0; j<num-i-1; j++)

{

if(m[j][1] > m[j+1][1])

{

for(int k=0; k<5; k++)

{

swap(m[j][k], m[j+1][k]);

}

}

}

}

}

void completionTime(int num, int m[][6])

{

int temp, val;

m[0][3] = m[0][1] + m[0][2];

m[0][5] = m[0][3] - m[0][1];

m[0][4] = m[0][5] - m[0][2];

for(int i=1; i<num; i++)

{

temp = m[i-1][3];

int low = m[i][2];

for(int j=i; j<num; j++)

{

if(temp >= m[j][1] && low >= m[j][2])

{

low = m[j][2];

val = j;

}

}

m[val][3] = temp + m[val][2];

m[val][5] = m[val][3] - m[val][1];

m[val][4] = m[val][5] - m[val][2];

for(int k=0; k<6; k++)

{

swap(m[val][k], m[i][k]);

}

}

}

int main()

{

int num, temp;

cout<<"Enter number of Process: ";

cin>>num;

cout<<"Enter the process ID: \n";

for(int i=0; i<num; i++)

{

cout<<"===Process "<<i+1<<"===\n";

cout<<"Enter Process Id: ";

cin>>m[i][0];

cout<<"Enter Arrival Time: ";

cin>>m[i][1];

cout<<"Enter Burst Time: ";

cin>>m[i][2];

}

cout<<"BEFORE ARRANGE:\n";

cout<<"Process ID\tArrival Time\tBurst Time\n";

for(int i=0; i<num; i++)

{

cout<<m[i][0]<<"\t\t"<<m[i][1]<<"\t\t"<<m[i][2]<<"\n";

}

arrangeArrival(num, m);

completionTime(num, m);

cout<<"FINAL RESULT:\n";

cout<<"Process ID\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n";

for(int i=0; i<num; i++)

{

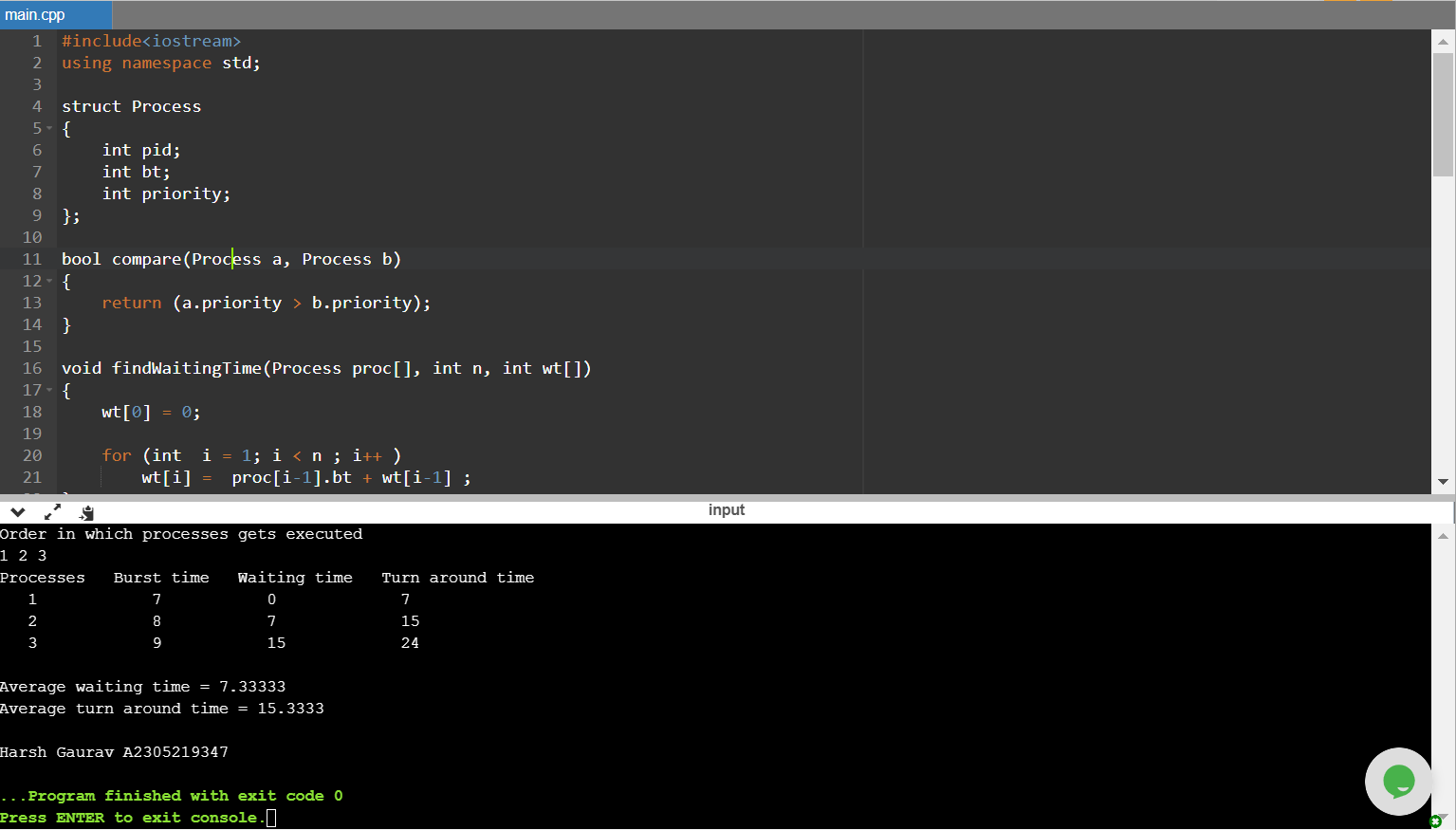
cout<<m[i][0]<<"\t\t"<<m[i][1]<<"\t\t"<<m[i][2]<<"\t\t"<<m[i][4]<<"\t\t"<<m[i][5]<<"\n";

}

cout<<"\n\nHarsh Gaurav A2305219347";

}

Priority Based Scheduling (PBS)



#include<iostream>

using namespace std;

struct Process

{

int pid;

int bt;

int priority;

};

bool compare(Process a, Process b)

{

return (a.priority > b.priority);

}

void findWaitingTime(Process proc[], int n, int wt[])

{

wt[0] = 0;

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

wt[i] = proc[i-1].bt + wt[i-1] ;

}

void findTurnAroundTime(Process proc[], int n, int wt[], int tat[])

{

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

tat[i] = proc[i].bt + wt[i];

}

void findavgTime(Process proc[], int n)

{

int wt[n], tat[n], total\_wt = 0, total\_tat = 0;

findWaitingTime(proc, n, wt);

findTurnAroundTime(proc, n, wt, tat);

cout << "\nProcesses "<< " Burst time "

<< " Waiting time " << " Turn around time\n";

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

{

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

cout << " " << proc[i].pid << "\t\t"

<< proc[i].bt << "\t " << wt[i]

<< "\t\t " << tat[i] <<endl;

}

cout << "\nAverage waiting time = "

<< (float)total\_wt / (float)n;

cout << "\nAverage turn around time = "

<< (float)total\_tat / (float)n;

}

void PS(Process proc[], int n)

{

cout<< "Order in which processes gets executed \n";

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

cout << proc[i].pid <<" " ;

findavgTime(proc, n);

}

int main()

{

Process proc[] = {{1, 7, 2}, {2, 8, 0}, {3, 9, 1}};

int n = sizeof proc / sizeof proc[0];

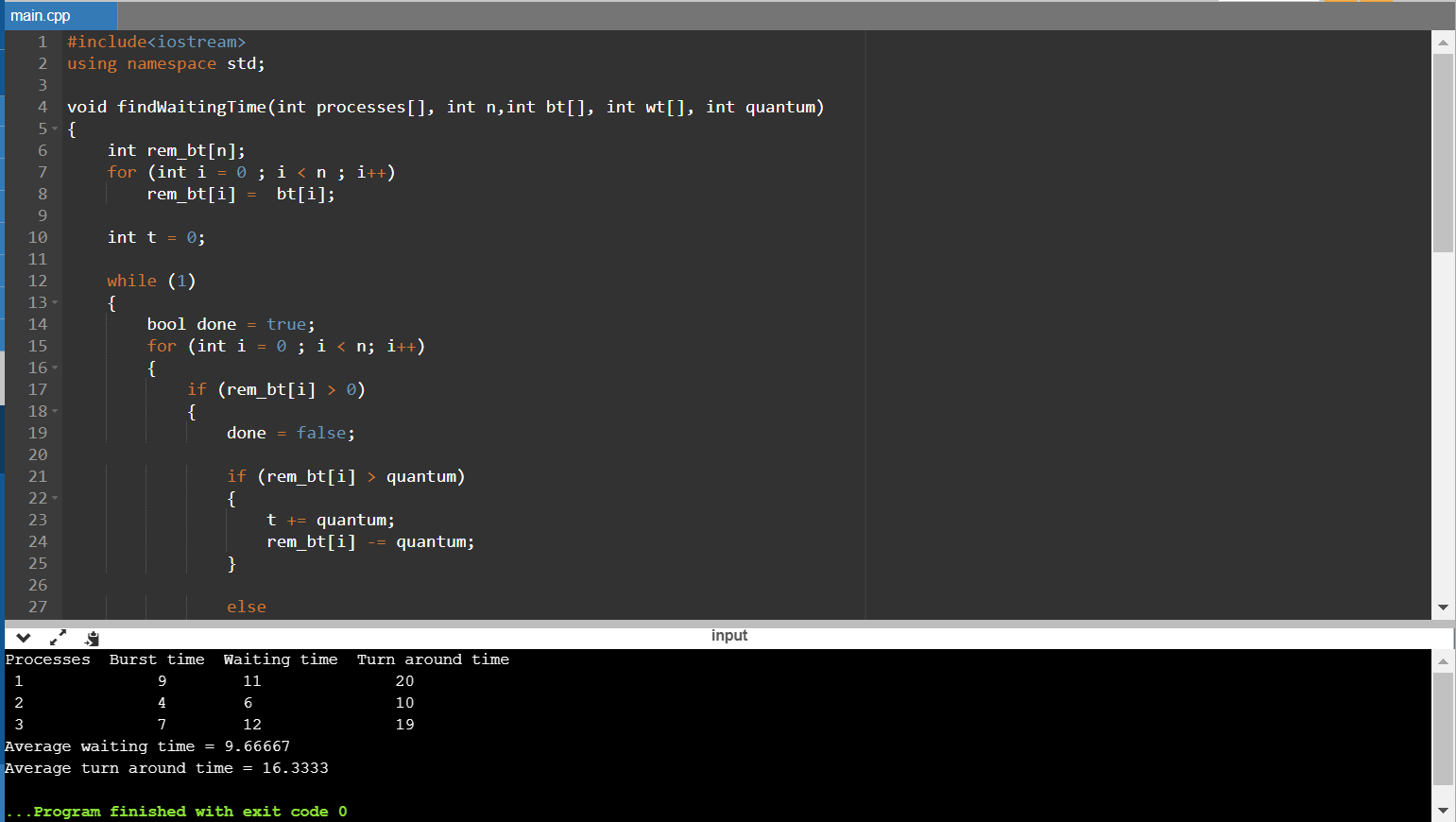
PS(proc, n);

cout<<"\n\nHarsh Gaurav A2305219347";

return 0;

}

Round Robin Scheduling (RRS)



#include<iostream>

using namespace std;

void findWaitingTime(int processes[], int n,int bt[], int wt[], int quantum)

{

int rem\_bt[n];

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

rem\_bt[i] = bt[i];

int t = 0;

while (1)

{

bool done = true;

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

{

if (rem\_bt[i] > 0)

{

done = false;

if (rem\_bt[i] > quantum)

{

t += quantum;

rem\_bt[i] -= quantum;

}

else

{

t = t + rem\_bt[i];

wt[i] = t - bt[i];

rem\_bt[i] = 0;

}

}

}

if (done == true)

break;

}

}

void findTurnAroundTime(int processes[], int n,int bt[], int wt[], int tat[])

{

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

tat[i] = bt[i] + wt[i];

}

void findavgTime(int processes[], int n, int bt[], int quantum)

{

int wt[n], tat[n], total\_wt = 0, total\_tat = 0;

findWaitingTime(processes, n, bt, wt, quantum);

findTurnAroundTime(processes, n, bt, wt, tat);

cout << "Processes "<< " Burst time "

<< " Waiting time " << " Turn around time\n";

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

{

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

cout << " " << i+1 << "\t\t" << bt[i] <<"\t "

<< wt[i] <<"\t\t " << tat[i] <<endl;

}

cout << "Average waiting time = "

<< (float)total\_wt / (float)n;

cout << "\nAverage turn around time = "

<< (float)total\_tat / (float)n;

}

int main()

{

int processes[] = { 1, 2, 3};

int n = sizeof processes / sizeof processes[0];

int burst\_time[] = {9, 4, 7};

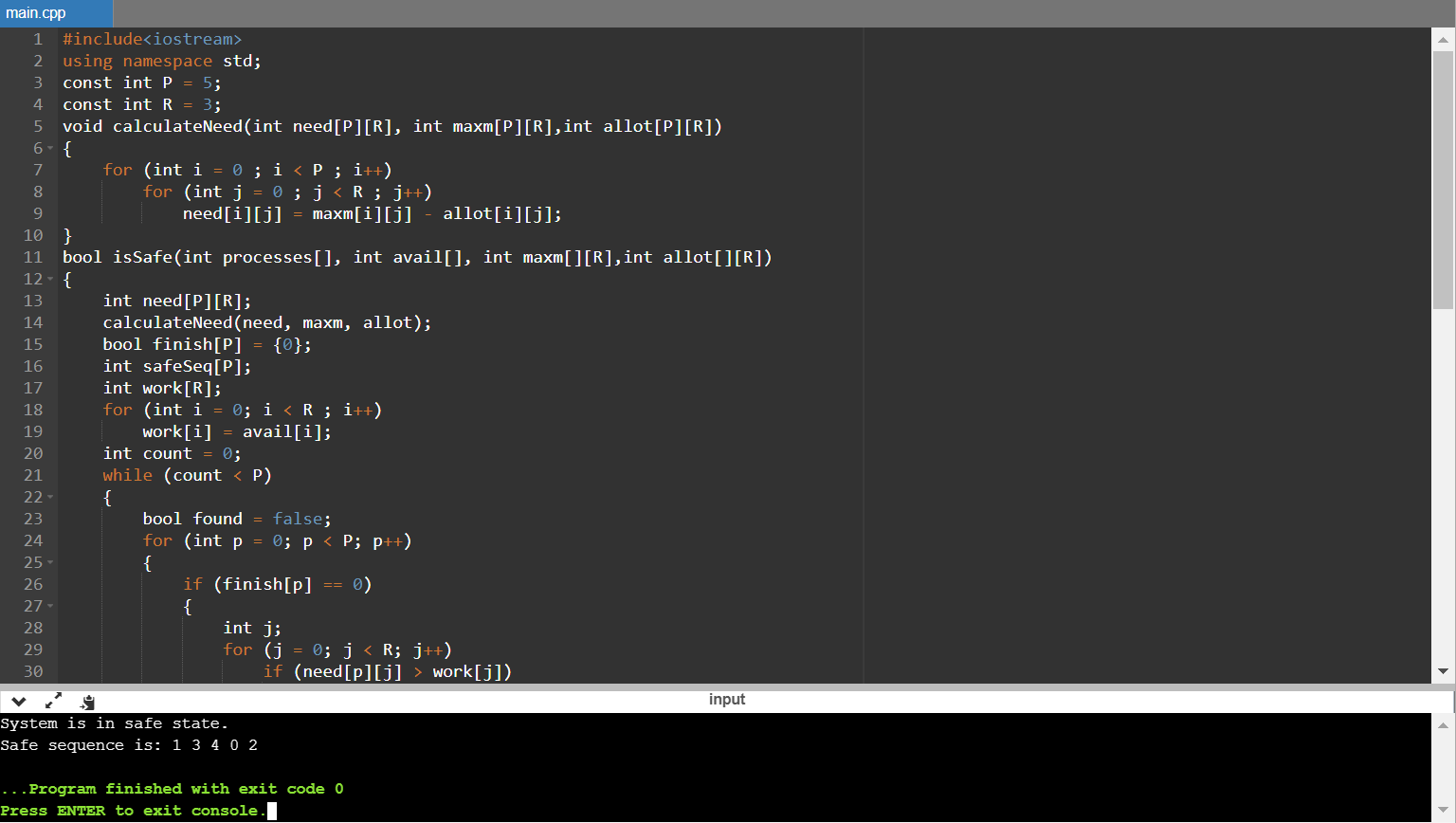
int quantum = 2;

findavgTime(processes, n, burst\_time, quantum);

return 0;

}

Banker’s Algorithm



#include<iostream>

using namespace std;

const int P = 5;

const int R = 3;

void calculateNeed(int need[P][R], int maxm[P][R],int allot[P][R])

{

for (int i = 0 ; i < P ; i++)

for (int j = 0 ; j < R ; j++)

need[i][j] = maxm[i][j] - allot[i][j];

}

bool isSafe(int processes[], int avail[], int maxm[][R],int allot[][R])

{

int need[P][R];

calculateNeed(need, maxm, allot);

bool finish[P] = {0};

int safeSeq[P];

int work[R];

for (int i = 0; i < R ; i++)

work[i] = avail[i];

int count = 0;

while (count < P)

{

bool found = false;

for (int p = 0; p < P; p++)

{

if (finish[p] == 0)

{

int j;

for (j = 0; j < R; j++)

if (need[p][j] > work[j])

break;

if (j == R)

{

for (int k = 0 ; k < R ; k++)

work[k] += allot[p][k];

safeSeq[count++] = p;

finish[p] = 1;

found = true;

}

}

}

if (found == false)

{

cout << "System is not in safe state";

return false;

}

}

cout << "System is in safe state.\nSafe"

" sequence is: ";

for (int i = 0; i < P ; i++)

cout << safeSeq[i] << " ";

return true;

}

int main()

{

int processes[] = {0, 1, 2, 3, 4};

int avail[] = {3, 3, 2};

int maxm[][R] = {{7, 5, 3},

{3, 2, 2},

{9, 0, 2},

{2, 2, 2},

{4, 3, 3}};

int allot[][R] = {{0, 1, 0},

{2, 0, 0},

{3, 0, 2},

{2, 1, 1},

{0, 0, 2}};

isSafe(processes, avail, maxm, allot);

return 0;

}

VI Editor

There are many ways to edit files in Unix. Editing files using the screen-oriented text editor vi is one of the best ways. This editor enables us to edit lines in context with other lines in the file.

vi is generally considered the de facto standard in Unix editors because −

* It's usually available on all the flavours of Unix system.
* Its implementations are very similar across the board.
* It requires very few resources.
* It is more user-friendly than other editors such as the ed or the ex.

|  |  |
| --- | --- |
| 1 | **vi filename**  Creates a new file if it already does not exist, otherwise opens an existing file. |
| 2 | **vi -R filename**  Opens an existing file in the read-only mode. |
| 3 | **view filename**  Opens an existing file in the read-only mode. |

**Moving within a File**

To move around within a file without affecting your text, we must be in the command mode (press Esc twice). The following table lists out a few commands we can use to move around one character at a time:

|  |  |
| --- | --- |
| 1 | **k**  Moves the cursor up one line |
| 2 | **j**  Moves the cursor down one line |
| 3 | **h**  Moves the cursor to the left one character position |
| 4 | **l**  Moves the cursor to the right one character position |

**Editing Files**

To edit the file, we need to be in the insert mode. There are many ways to enter the insert mode from the command mode:

|  |  |
| --- | --- |
| 1 | **i**  Inserts text before the current cursor location |
| 2 | **I**  Inserts text at the beginning of the current line |
| 3 | **a**  Inserts text after the current cursor location |
| 4 | **A**  Inserts text at the end of the current line |
| 5 | **o**  Creates a new line for text entry below the cursor location |
| 6 | **O**  Creates a new line for text entry above the cursor location |

**Deleting Characters**

|  |  |
| --- | --- |
| 1 | **x**  Deletes the character under the cursor location |
| 2 | **X**  Deletes the character before the cursor location |
| 3 | **dw**  Deletes from the current cursor location to the next word |
| 4 | **d^**  Deletes from the current cursor position to the beginning of the line |
| 5 | **d$**  Deletes from the current cursor position to the end of the line |
| 6 | **D**  Deletes from the cursor position to the end of the current line |
| 7 | **dd**  Deletes the line the cursor is on |

**Change Commands**

|  |  |
| --- | --- |
| 1 | **cc**  Removes the contents of the line, leaving you in insert mode. |
| 2 | **cw**  Changes the word the cursor is on from the cursor to the lowercase **w** end of the word. |
| 3 | **r**  Replaces the character under the cursor. vi returns to the command mode after the replacement is entered. |
| 4 | **R**  Overwrites multiple characters beginning with the character currently under the cursor. You must use **Esc** to stop the overwriting. |
| 5 | **s**  Replaces the current character with the character you type. Afterward, you are left in the insert mode. |
| 6 | **S**  Deletes the line the cursor is on and replaces it with the new text. After the new text is entered, vi remains in the insert mode. |

**Copy and Paste Commands**

|  |  |
| --- | --- |
| 1 | **yy**  Copies the current line. |
| 2 | **yw**  Copies the current word from the character the lowercase w cursor is on, until the end of the word. |
| 3 | **p**  Puts the copied text after the cursor. |
| 4 | **P**  Puts the yanked text before the cursor. |