

**SYSTEM SOFTWARE LAB RECORD**

(CS331)



hARIkrishnan v

cs5a

Roll No:27

**INDEX**

|  |  |  |
| --- | --- | --- |
| **Sl. No.** | **Title** | **Page No.** |
| 1 | CPU Scheduling | 3 |
| 2 | File Organization | 13 |
| 3 | Banker’s Algorithm | 23 |
| 4 | Disk Scheduling | 27 |
| 5 | Producer-Consumer Problem | 33 |
| 6 | Dining Philosopher’s Problem | 36 |
| 7 | Pass 1 of Two Pass Assembler | 40 |
| 8 | Pass 2 of Two Pass Assembler | 44 |
| 9 | Single Pass Assembler | 48 |
| 10 | Two Pass Macro Processor | 54 |
| 11 | Symbol Table with Hashing | 58 |
| 12 | Absolute Loader | 64 |

**PART A**

**CPU SCHEDULING**

**Program Code:-**

1. **FCFS**

#include<stdio.h>

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

{

int service\_time[n];

service\_time[0]=0;

wt[0]=0;

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

{

service\_time[i]=service\_time[i-1]+bt[i-1];

wt[i]=service\_time[i]-at[i];

if (wt[i]<0)

{

wt[i]=0;

}

}

}

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 at[])

{

int wt[n],tat[n];

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

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

printf("\nProcesses Burst Time Arrival Time Waiting Time Turn-Around Time Completion Time");

int total\_wt=0,total\_tat=0;

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

{

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

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

int compl\_time=tat[i]+at[i];

printf("\n %d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d",i+1,bt[i],at[i],wt[i],tat[i],compl\_time);

}

printf("\nAverage waiting time=%f",(float)total\_wt/(float)n);

printf("\nAverage turn around time=%f",(float)total\_tat/(float)n);

}

void main()

{

int processes[20],n,burst\_time[20],arrival\_time[20],i;

printf("Enter the number of processes:");

scanf("%d",&n);

printf("\nEnter Burst Times(in ms) of Processes:");

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

{

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

scanf("%d",&burst\_time[i]);

}

printf("\nEnter Arrival Times(in ms) of Processes:");

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

{

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

scanf("%d",&arrival\_time[i]);

}

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

}

1. **SJF**

#include<stdio.h>

void main()

{

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

float avg\_wt,avg\_tat;

printf("Enter the number of process:");

scanf("%d",&n);

printf("\nEnter Burst Times(in ms) for Processes:");

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

{

printf("\nProcess %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+=wt[i];

}

avg\_wt=(float)total\_wt/n;

printf("\nProcess Burst Time Waiting Time Turn Around Time");

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

{

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

total\_tat+=tat[i];

printf("\n%d\t\t%d\t\t%d\t\t%d",p[i],bt[i],wt[i],tat[i]);

}

avg\_tat=(float)total\_tat/n;

printf("\n\nAverage Waiting Time=%f",avg\_wt);

printf("\nAverage Turnaround Time=%f\n",avg\_tat);

}

1. **Round Robin (pre-emptive)**

#include<stdio.h>

void main()

{

int count,j,n,time,remain,flag=0,time\_quantum,wait\_time=0,turnaround\_time=0,at[20],bt[20],rt[20];

printf("Enter the number of Processes:");

scanf("%d",&n);

remain=n;

printf("\nEnter Arrival Times(in ms) for Processes:");

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

{

printf("\nProcess %d:",count+1);

scanf("%d",&at[count]);

}

printf("\nEnter Burst Times(in ms) for Processes:");

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

{

printf("\nProcess %d:",count+1);

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

rt[count]=bt[count];

}

printf("\nEnter the Time Quantum:");

scanf("%d",&time\_quantum);

printf("\nProcesses Turn Around Time Waiting Time");

for(time=0,count=0;remain!=0;)

{

if ((rt[count]<=time\_quantum)&&(rt[count]>0))

{

time+=rt[count];

rt[count]=0;

flag=1;

}

else if (rt[count]>0)

{

rt[count]-=time\_quantum;

time+=time\_quantum;

}

if ((rt[count]==0)&&(flag==1))

{

remain--;

printf("\n%d\t\t%d\t\t%d",count+1,time-at[count],time-at[count]-bt[count]);

wait\_time+=time-at[count]-bt[count];

turnaround\_time+=time-at[count];

flag=0;

}

if (count==n-1)

{

count=0;

}

else if(at[count+1]<=time)

{

count++;

}

else

{

count=0;

}

}

printf("\nAverage Waiting Time=%f",(float)(wait\_time\*1.0/n));

printf("\nAverage Turn Around Time:%f",(float)(turnaround\_time\*1.0/n));

}

1. **Priority**

#include<stdio.h>

void main()

{

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

float avg\_wt,avg\_tat;

printf("Enter the number of Processes:");

scanf("%d",&n);

printf("\nEnter the Burst Times(in ms) for Processes:");

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

{

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

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

p[i]=i+1;

}

printf("\nEnter the Priority values for Processes:");

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

{

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

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

}

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+=wt[i];

}

avg\_wt=total\_wt/n;

printf("\nProcesses Burst Time Waiting Time Turn Around Time");

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

{

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

total\_tat+=tat[i];

printf("\n%d\t\t%d\t\t%d\t\t%d",p[i],bt[i],wt[i],tat[i]);

}

avg\_tat=total\_tat/n;

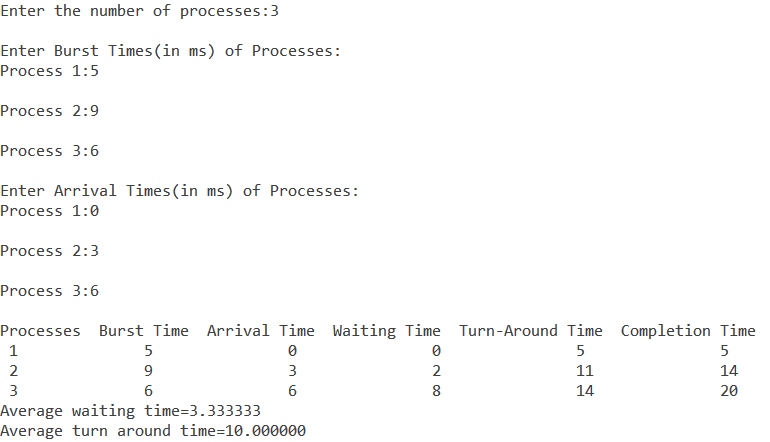
printf("\nAverage Waiting Time=%f",avg\_wt);

printf("\nAverage Turn Around Time=%f",avg\_tat);

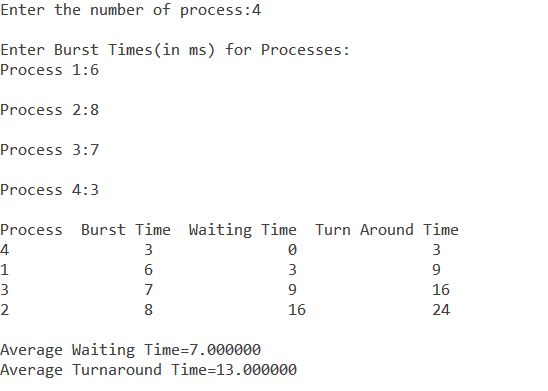
}

**Output:-**

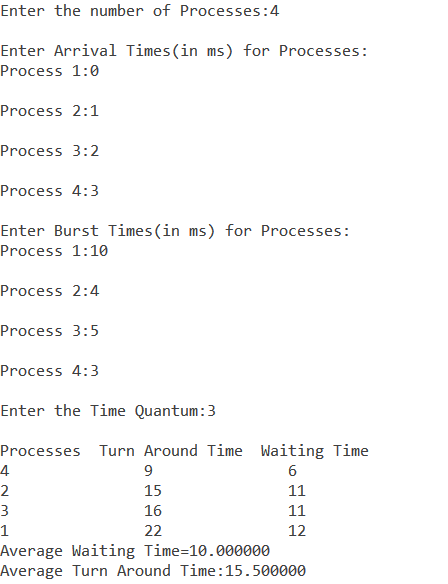
1. **FCFS**



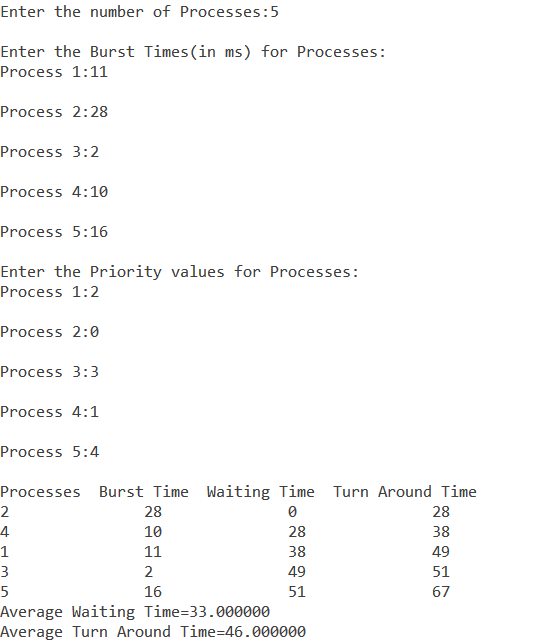
1. **SJF**



1. **Round Robin (pre-emptive)**



1. **Priority**



**File Organization**

**Program Code:-**

1. **Single Level**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

struct

{

char dname[10],fname[10][10];

int fcnt;

}dir;

void main()

{

int i,ch;

char f[30];

dir.fcnt = 0;

printf("Enter Name of Directory:");

scanf("%s",dir.dname);

while(1)

{

printf("\n1.Create File\n2.Delete File\n3.Search File\n4.Display Files\n5.Exit\nEnter Your Choice:");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("\nEnter the Name of the File to Create:");

scanf("%s",dir.fname[dir.fcnt]);

dir.fcnt++;

break;

case 2: printf("\nEnter the Name of the File to Delete:");

scanf("%s",f);

for(i=0;i<dir.fcnt;i++)

{

if(strcmp(f, dir.fname[i])==0)

{

printf("\nDeleted File %s",f);

strcpy(dir.fname[i],dir.fname[dir.fcnt-1]);

break;

}

}

if(i==dir.fcnt)

printf("\nFile %s Not Found!",f);

else

dir.fcnt--;

break;

case 3: printf("\nEnter the Name of the File to Search:");

scanf("%s",f);

for(i=0;i<dir.fcnt;i++)

{

if(strcmp(f, dir.fname[i])==0)

{

printf("\nFile %s Found ", f);

break;

}

}

if(i==dir.fcnt)

printf("\nFile %s Not Found!",f);

break;

case 4: if(dir.fcnt==0)

printf("\nDirectory is Empty!");

else

{

printf("\nThe Files are:");

for(i=0;i<dir.fcnt;i++)

printf("\n%s",dir.fname[i]);

}

break;

default: exit(0);

}

}

}

1. **Two Level**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

struct

{

char dname[10], fname[10][10];

int fcnt;

} dir[10];

void main()

{

int i, ch, dcnt, k;

char f[30], d[30];

dcnt = 0;

while (1)

{

printf("\n1.Create Directory\n2.Create File\n3.Delete File\n4.Search File\n5.Display\n6.Exit\nEnter Your Choice:");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("\nEnter Name of Directory:");

scanf("%s", dir[dcnt].dname);

dir[dcnt].fcnt = 0;

dcnt++;

printf("\nDirectory created");

break;

case 2:

printf("\nEnter Name of the Directory:");

scanf("%s", d);

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

if (strcmp(d, dir[i].dname) == 0)

{

printf("\nEnter Name of the File to Create:");

scanf("%s", dir[i].fname[dir[i].fcnt]);

dir[i].fcnt++;

printf("\nFile created");

break;

}

if (i == dcnt)

printf("\nDirectory %s Not Found!", d);

break;

case 3:

printf("\nEnter Name of the Directory:");

scanf("%s", d);

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

{

if (strcmp(d, dir[i].dname) == 0)

{

printf("\nEnter Name of the File to Delete:");

scanf("%s", f);

for (k = 0; k < dir[i].fcnt; k++)

{

if (strcmp(f, dir[i].fname[k]) == 0)

{

printf("\nFile %s Deleted", f);

dir[i].fcnt--;

strcpy(dir[i].fname[k], dir[i].fname[dir[i].fcnt]);

goto jmp;

}

}

printf("\nFile %s Not Found!", f);

goto jmp;

}

}

printf("\nDirectory %s Not Found!", d);

jmp:

break;

case 4:

printf("\nEnter Name of the Directory:");

scanf("%s", d);

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

{

if (strcmp(d, dir[i].dname) == 0)

{

printf("\nEnter the Name of the File to Search:");

scanf("%s", f);

for (k = 0; k < dir[i].fcnt; k++)

{

if (strcmp(f, dir[i].fname[k]) == 0)

{

printf("\nFile %s Found", f);

goto jmp1;

}

}

printf("\nFile %s Not Found!", f);

goto jmp1;

}

}

printf("\nDirectory %s Not Found!", d);

jmp1:

break;

case 5:

if (dcnt == 0)

printf("\nNo Directories!");

else

{

printf("\nDirectory\tFiles");

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

{

printf("\n%s\t\t", dir[i].dname);

for (k = 0; k < dir[i].fcnt; k++)

printf("\t%s", dir[i].fname[k]);

}

}

break;

default:

exit(0);

}

}

}

1. **Hierarchical**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <string.h>

struct node

{

char name[128];

bool isDir;

struct node \*p;

struct node \*c[100];

int i;

} \* head, \*curr;

void ls()

{

if (curr->i == 0)

{

printf("\nEmpty directory!");

return;

}

for (int i = 0; i < curr->i; i++)

{

if (curr->c[i]->isDir)

printf("\n\*%s\* ", curr->c[i]->name);

else

printf("\n%s ", curr->c[i]->name);

}

}

void touch(bool d)

{

printf("\nEnter Name:");

char fname[128];

scanf("%s", fname);

struct node \*temp = (struct node \*)malloc(sizeof(struct node));

strcpy(temp->name, fname);

temp->isDir = d;

temp->p = curr;

curr->c[curr->i] = temp;

curr->i = (curr->i) + 1;

}

void cd()

{

printf("\nEnter Directory Name:");

char dname[128];

scanf("%s", dname);

for (int i = 0; i < curr->i; i++)

{

if (!strcmp(curr->c[i]->name, dname) && curr->c[i]->isDir == true)

{

curr = curr->c[i];

return;

}

}

printf("\nDirectory Not Found!");

}

void cdup()

{

if (curr->p == NULL)

{

printf("\nYou are at the Root Directory");

return;

}

curr = curr->p;

}

void rm(bool d)

{

printf("\nEnter Name of File or Directory to Delete:");

char name[128];

scanf("%s", name);

for (int i = 0; i < curr->i; i++)

{

if (!strcmp(curr->c[i]->name, name) && ((d && curr->c[i]->isDir == true) || (!d && curr->c[i]->isDir == false)))

{

int t = i;

while (t < (curr->i) - 1)

{

curr->c[t] = curr->c[t + 1];

t++;

}

curr->i = (curr->i) - 1;

printf("\nSuccessfully Deleted.");

return;

}

}

printf("\nNot found");

}

void main()

{

int in;

head = (struct node \*)malloc(sizeof(struct node));

strcpy(head->name, "root");

head->isDir = true;

head->p = NULL;

head->i = 0;

curr = head;

while (true)

{

printf("\nYou are in %s directory.\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n1.Show everything in this Directory\n2.Change Directory\n3.Go to Parent Directory\n4.Create New File\n5.Delete File\n6.Create New Directory\n7.Delete Directory\n8.Exit\nEnter your choice:", curr->name);

scanf("%d", &in);

switch (in)

{

case 1:

ls();

break;

case 2:

cd();

break;

case 3:

cdup();

break;

case 4:

touch(false);

break;

case 5:

rm(false);

break;

case 6:

touch(true);

break;

case 7:

rm(true);

break;

default:

exit(0);

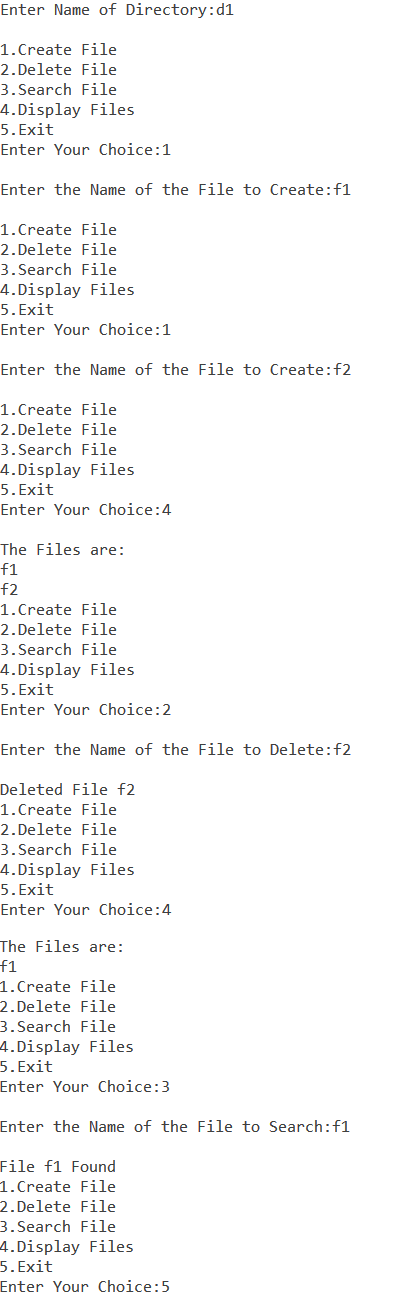
}

}

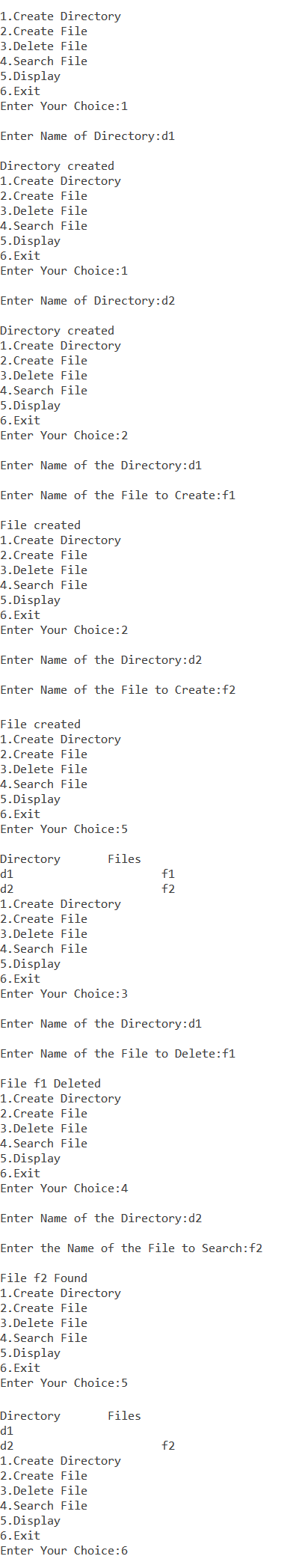
}

**Output:-**

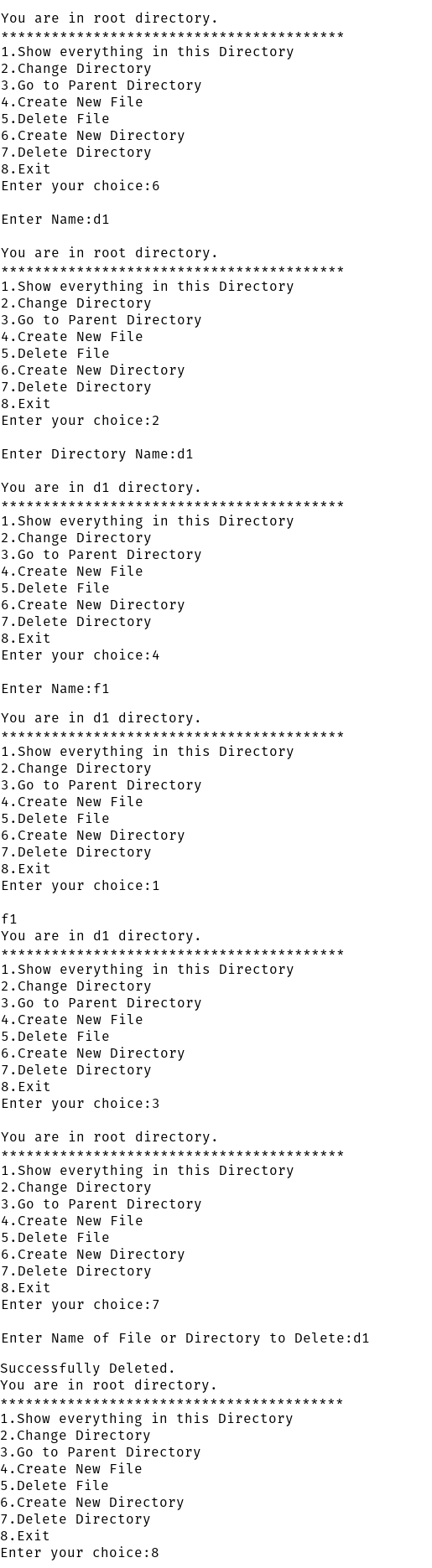
1. **Single Level**



1. **Two Level**



1. **Hierarchical**



**Banker’s Algorithm**

**Program Code:-**

#include <stdio.h>

void main()

{

int n, m, available[10], need[10][10], allocation[10][10], max[10][10], i, j, k, p[10], work[10], finish[10], index = 0, flag, y;

printf("Enter number of Resource Types:");

scanf("%d", &m);

printf("\nEnter number of Processes:");

scanf("%d", &n);

printf("\nEnter number of instances of each Resource Type:");

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

{

printf("\nEnter number of instances of Resource Type %d:", i + 1);

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

work[i] = available[i];

}

printf("\nEnter number of Allocated instances for each Process:");

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

{

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

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

{

printf("\nResource Type %d:", j + 1);

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

}

}

printf("\nEnter number of Maximum instances for each Process:");

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

{

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

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

{

printf("\nResource Type %d:", j + 1);

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

}

}

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

{

finish[i] = 0;

}

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

{

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

{

need[i][j] = max[i][j] - allocation[i][j];

}

}

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

{

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

{

if (finish[j] == 0)

{

flag = 0;

for (k = 0; k < m; k++)

{

if (need[j][k] > available[k])

{

flag = 1;

break;

}

}

if (flag == 0)

{

p[index++] = j;

for (y = 0; y < m; y++)

{

available[y] += allocation[j][y];

}

finish[j] = 1;

}

}

}

}

printf("\nSafe Sequence:");

for (i = 0; i < n - 1; i++)

{

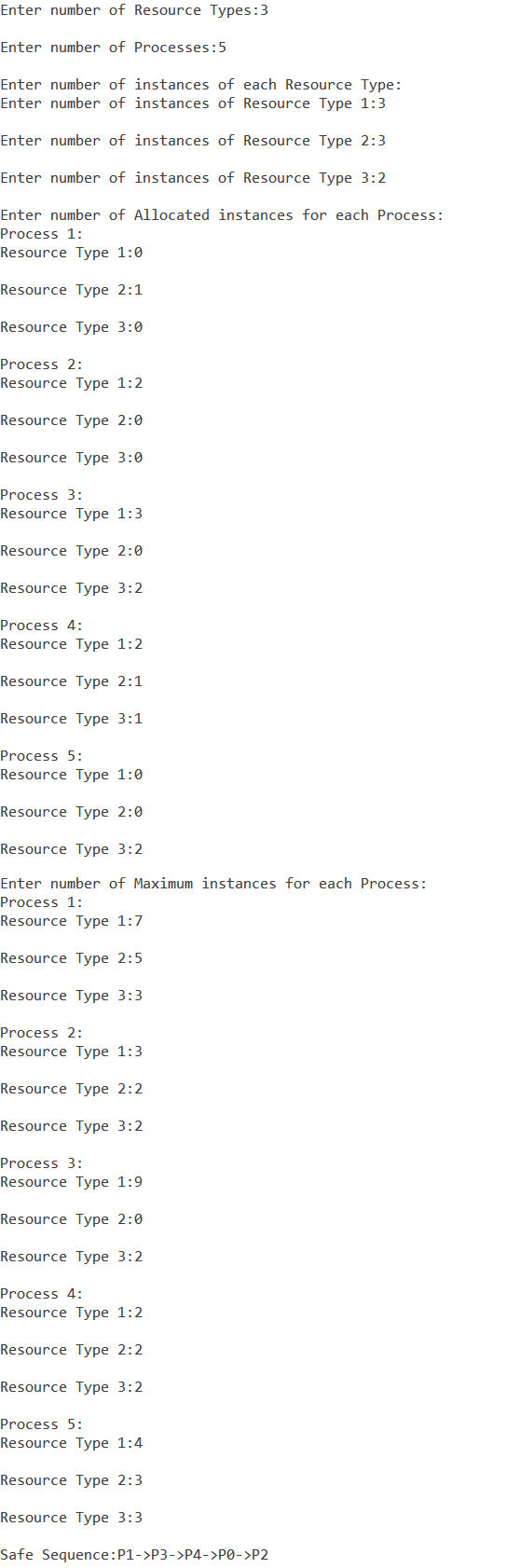
printf("P%d->", p[i]);

}

printf("P%d", p[n - 1]);

}

**Output:-**



**Disk Scheduling**

**Program Code:-**

1. **FCFS**

#include <stdio.h>

#include <stdlib.h>

void main()

{

int queue[100], q\_size, head, seek = 0, diff;

printf("Enter the Size of the Queue:");

scanf("%d", &q\_size);

printf("\nEnter Queue Elements:");

for (int i = 1; i <= q\_size; i++)

{

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

}

printf("\nEnter Current Head Position:");

scanf("%d", &head);

queue[0] = head;

for (int j = 0; j <= q\_size - 1; j++)

{

diff = abs(queue[j] - queue[j + 1]);

seek += diff;

printf("Move from %d to %d with Seek %d\n", queue[j], queue[j + 1], diff);

}

printf("\nTotal number of Cylinder Access=%d", seek);

}

1. **SCAN**

#include <stdio.h>

#include <stdlib.h>

#define LOW 0

#define HIGH 199

void main()

{

int queue[20], head, max, q\_size, temp, sum, dloc;

printf("Enter the Number of Disk Locations:");

scanf("%d", &q\_size);

printf("\nEnter the Current Head Position:");

scanf("%d", &head);

printf("\nEnter the Elements into Disk Queue:");

for (int i = 0; i < q\_size; i++)

{

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

}

queue[q\_size] = head;

q\_size++;

for (int i = 0; i < q\_size; i++)

{

for (int j = i; j < q\_size; j++)

{

if (queue[i] > queue[j])

{

temp = queue[i];

queue[i] = queue[j];

queue[j] = temp;

}

}

}

max = queue[q\_size - 1];

for (int i = 0; i < q\_size; i++)

{

if (head == queue[i])

{

dloc = i;

break;

}

}

if (abs(head - LOW) <= abs(head - HIGH))

{

for (int j = dloc; j >= 0; j--)

{

printf("%d->", queue[j]);

}

for (int j = dloc + 1; j < q\_size; j++)

{

printf("%d->", queue[j]);

}

}

else

{

for (int j = dloc + 1; j < q\_size; j++)

{

printf("%d->", queue[j]);

}

for (int j = dloc; j >= 0; j--)

{

printf("%d->", queue[j]);

}

}

sum = head + max;

printf("\nTotal number of Cylinder Access=%d", sum);

}

1. **C-SCAN**

#include <stdlib.h>

#include <stdio.h>

#define HIGH 199

#define LOW 0

void main()

{

int queue[20], q\_size, head, i, j, seek = 0, diff, max, temp, queue1[20], queue2[20], temp1 = 0, temp2 = 0;

printf("Enter the Number of Disk Locations:");

scanf("%d", &q\_size);

printf("\nEnter the Current Head Position:");

scanf("%d", &head);

printf("\nEnter Elements into Disk Queue:");

for (i = 0; i < q\_size; i++)

{

scanf("%d", &temp);

if (temp >= head)

{

queue1[temp1] = temp;

temp1++;

}

else

{

queue2[temp2] = temp;

temp2++;

}

}

for (i = 0; i < temp1 - 1; i++)

{

for (j = i + 1; j < temp1; j++)

{

if (queue1[i] > queue1[j])

{

temp = queue1[i];

queue1[i] = queue1[j];

queue1[j] = temp;

}

}

}

for (i = 0; i < temp2 - 1; i++)

{

for (j = i + 1; j < temp2; j++)

{

if (queue2[i] > queue2[j])

{

temp = queue2[i];

queue2[i] = queue2[j];

queue2[j] = temp;

}

}

}

if (abs(head - LOW) >= abs(head - HIGH))

{

for (i = 1, j = 0; j < temp1; i++, j++)

{

queue[i] = queue1[j];

}

queue[i] = HIGH;

queue[i + 1] = 0;

for (i = temp1 + 3, j = 0; j < temp2; i++, j++)

{

queue[i] = queue2[j];

}

}

else

{

for (i = 1, j = temp2 - 1; j >= 0; i++, j--)

{

queue[i] = queue2[j];

}

queue[i] = LOW;

queue[i + 1] = HIGH;

for (i = temp2 + 3, j = temp1 - 1; j >= 0; i++, j--)

{

queue[i] = queue1[j];

}

}

queue[0] = head;

for (j = 0; j <= q\_size + 1; j++)

{

diff = abs(queue[j + 1] - queue[j]);

seek += diff;

printf("\nDisk head moves from %d to %d with seek %d\n", queue[j], queue[j + 1], diff);

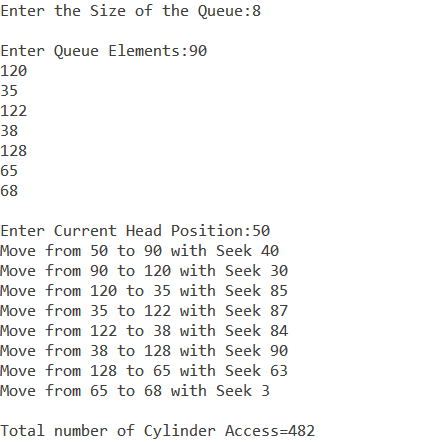
}

printf("\nTotal number of Cylinder Access=%d", seek);

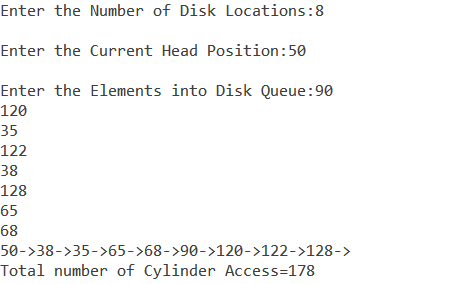
}

**Output:-**

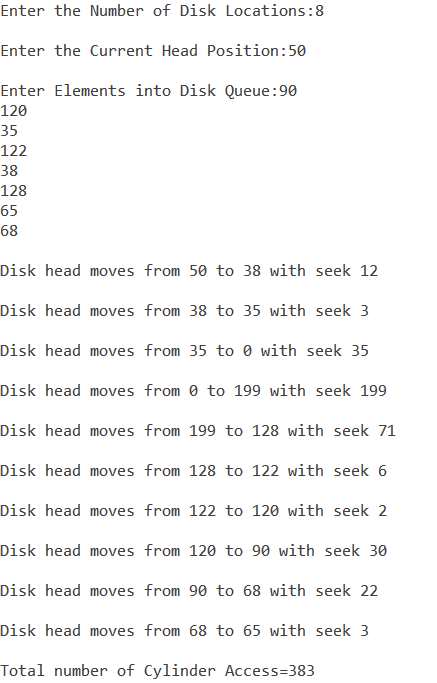
1. **FCFS**



1. **SCAN**



1. **C-SCAN**



**Producer Consumer Problem**

**Program Code:-**

#include <stdio.h>

#include <stdlib.h>

int mutex = 1, full = 0, empty = 3, item = 0;

int wait(int s)

{

return (--s);

}

int signal(int s)

{

return (++s);

}

void Producer()

{

mutex = wait(mutex);

full = signal(full);

empty = wait(empty);

item++;

printf("\nProducer produced item %d.", item);

mutex = signal(mutex);

}

void Consumer()

{

mutex = wait(mutex);

full = wait(full);

empty = signal(empty);

printf("\nConsumer consumed item %d.", item);

item--;

mutex = signal(mutex);

}

void main()

{

int ch;

while (1)

{

printf("\nMenu\n1.Produce Item\n2.Consume Item\n3.Exit\nEnter Your Choice:");

scanf("%d", &ch);

switch (ch)

{

case 1:

if ((mutex == 1) && (empty != 0))

Producer();

else

printf("\nBuffer is Full!!");

break;

case 2:

if ((mutex == 1) && (full != 0))

Consumer();

else

printf("\nBuffer is Empty!!");

break;

case 3:

exit(0);

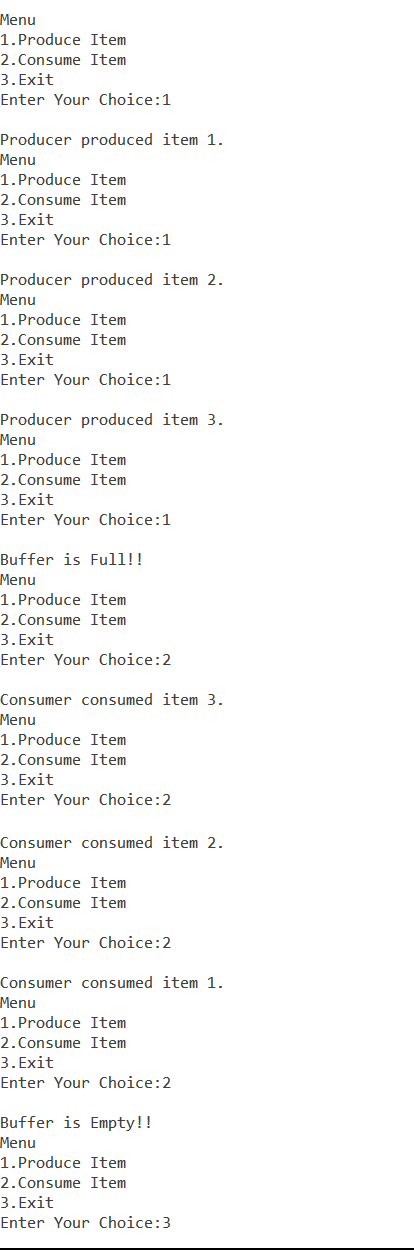
break;

}

}

}

**Output:-**



**Dining Philosophers Problem**

**Program Code:-**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

bool all\_philosophers\_finished\_eating(bool phils[], int n)

{

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

if (!phils[i])

return false;

return true;

}

void clear\_chopsticks(bool chops[], int n)

{

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

chops[i] = true;

}

int main()

{

int n;

bool chops[n], philosophers\_finished\_eating[n], flag = true;

printf("Enter the Number of Philosophers:");

scanf("%d", &n);

clear\_chopsticks(chops, n);

while (flag)

{

printf("\nNew Loop:");

clear\_chopsticks(chops, n);

flag = false;

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

{

if (!philosophers\_finished\_eating[i])

{

if (chops[i] && chops[(i + 1) % 5])

{

chops[i] = false;

chops[(i + 1) % 5] = false;

printf("\nPhilosopher %d is Eating.", i);

philosophers\_finished\_eating[i] = true;

flag = true;

}

else

printf("\nPhilosopher %d is Thinking.", i);

}

else

printf("\nPhilosopher %d has Finished Eating.", i);

}

if (all\_philosophers\_finished\_eating(philosophers\_finished\_eating, n))

{

printf("\nProgram Completed Successfully.");

exit(0);

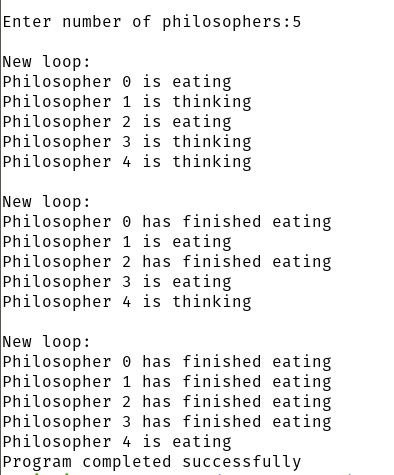
}

}

printf("\nDeadlock is Present!!");

}

**Output:-**



**PART B**

**Pass 1 of Two Pass Assembler**

**Program Code:-**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

void main()

{

FILE \*f1, \*f2, \*f3;

int lc, sa, l, op1, o, len;

char m1[20], la[20], op[20], otp[20];

f1 = fopen("input.txt", "r");

f3 = fopen("symtab.txt", "w");

fscanf(f1, "%s %s %d", la, m1, &op1);

if (strcmp(m1, "START") == 0)

{

sa = op1;

lc = sa;

printf("\t%s\t%s\t%d\n", la, m1, op1);

}

else

lc = 0;

fscanf(f1, "%s %s", la, m1);

while (!feof(f1))

{

fscanf(f1, "%s", op);

printf("\n%d\t%s\t%s\t%s\n", lc, la, m1, op);

if (strcmp(la, "-") != 0)

{

fprintf(f3, "\n%d\t%s\t%s\t%s\n", lc, la, m1, op);

}

f2 = fopen("optab.txt", "r");

fscanf(f2, "%s %d", otp, &o);

while (!feof(f2))

{

if (strcmp(m1, otp) == 0)

{

lc = lc + 3;

break;

}

fscanf(f2, "%s %d", otp, &o);

}

fclose(f2);

if (strcmp(m1, "WORD") == 0)

{

lc = lc + 3;

}

else if (strcmp(m1, "RESW") == 0)

{

op1 = atoi(op);

lc = lc + (3 \* op1);

}

else if (strcmp(m1, "BYTE") == 0)

{

if (op[0] == 'X')

lc = lc + 1;

else

{

len = strlen(op) - 3;

lc = lc + len;

}

}

else if (strcmp(m1, "RESB") == 0)

{

op1 = atoi(op);

lc = lc + op1;

}

fscanf(f1, "%s%s", la, m1);

}

if (strcmp(m1, "END") == 0)

{

printf("\nProgram Length = %d", lc - sa);

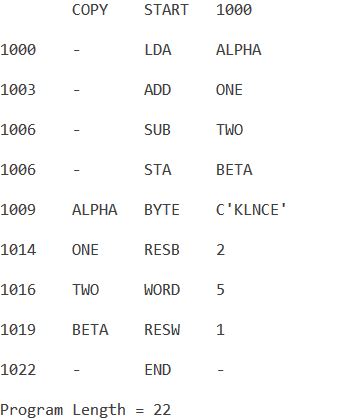
}

fclose(f1);

fclose(f3);

}

**Output:-**



**Pass 2 of Two Pass Assembler**

**Program Code:-**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void main()

{

FILE \*fint, \*ftab, \*flen, \*fsym;

int op1[10], textlen, textlen1, i, j = 0, len;

char add[5], symadd[5], op[5], start[10], temp[30], line[20], label[20], mne[10], operand[10], symtab[10], opmne[10];

fint = fopen("input.txt", "r");

flen = fopen("length.txt", "r");

ftab = fopen("optab.txt", "r");

fsym = fopen("symtab.txt", "r");

fscanf(fint, "%s%s%s%s", add, label, mne, operand);

if (strcmp(mne, "START") == 0)

{

strcpy(start, operand);

fscanf(flen, "%d", &len);

}

printf("H^%s^%s^%d\nT^00%s^", label, start, len, start);

fscanf(fint, "%s%s%s%s", add, label, mne, operand);

while (strcmp(mne, "END") != 0)

{

fscanf(ftab, "%s%s", opmne, op);

while (!feof(ftab))

{

if (strcmp(mne, opmne) == 0)

{

fclose(ftab);

fscanf(fsym, "%s%s", symadd, symtab);

while (!feof(fsym))

{

if (strcmp(operand, symtab) == 0)

{

printf("%s%s^", op, symadd);

break;

}

else

{

fscanf(fsym, "%s%s", symadd, symtab);

}

}

break;

}

else

{

fscanf(ftab, "%s%s", opmne, op);

}

}

if ((strcmp(mne, "BYTE") == 0) || (strcmp(mne, "WORD") == 0))

{

if (strcmp(mne, "WORD") == 0)

{

printf("0000%s^", operand);

}

else

{

len = strlen(operand);

for (i = 2; i < len; i++)

{

printf("%d", operand[i]);

}

printf("^");

}

}

fscanf(fint, "%s%s%s%s", add, label, mne, operand);

ftab = fopen("optab.txt", "r");

fseek(ftab, SEEK\_SET, 0);

}

printf("\nE^00%s", start);

fclose(fint);

fclose(ftab);

fclose(fsym);

fclose(flen);

}

**Output:-**



**Single Pass Assembler**

**Program Code:-**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

char \*my\_itoa(int num, char \*str)

{

if (str == NULL)

{

return NULL;

}

sprintf(str, "%d", num);

return str;

}

void main()

{

FILE \*f1, \*f2, \*f3, \*f4, \*f5;

int lc, sa, i = 0, j = 0, m[10], pgmlen, len, k, len1, l = 0;

char name[10], opnd[10], la[10], mne[10], s1[10], mne1[10], opnd1[10];

char lcs[10], ms[10];

char sym[10], symaddr[10], obj1[10], obj2[10], s2[10], q[10], s3[10];

f1 = fopen("input.txt", "r");

f2 = fopen("optab.txt", "r");

f3 = fopen("symtab.txt", "w+");

f4 = fopen("symtab1.txt", "w+");

f5 = fopen("output.txt", "w+");

fscanf(f1, "%s%s%s", la, mne, opnd);

if (strcmp(mne, "START") == 0)

{

sa = atoi(opnd);

strcpy(name, la);

lc = sa;

}

strcpy(s1, "\*");

fscanf(f1, "%s%s%s", la, mne, opnd);

while (strcmp(mne, "END") != 0)

{

if (strcmp(la, "-") == 0)

{

fscanf(f2, "%s%s", mne1, opnd1);

while (!feof(f2))

{

if (strcmp(mne1, mne) == 0)

{

m[i] = lc + 1;

fprintf(f3, "%s\t%s\n", opnd, s1);

fprintf(f5, "%s\t0000\n", opnd1);

lc = lc + 3;

i = i + 1;

break;

}

else

fscanf(f2, "%s%s", mne1, opnd1);

}

}

else

{

fseek(f3, SEEK\_SET, 0);

fscanf(f3, "%s%s", sym, symaddr);

while (!feof(f3))

{

if (strcmp(sym, la) == 0)

{

my\_itoa(lc, lcs);

fprintf(f4, "%s\t%s\n", la, lcs);

my\_itoa(m[j], ms);

j = j + 1;

fprintf(f5, "%s\t%s\n", ms, lcs);

i = i + 1;

break;

}

else

fscanf(f3, "%s%s", sym, symaddr);

}

if (strcmp(mne, "RESW") == 0)

lc = lc + 3 \* atoi(opnd);

else if (strcmp(mne, "BYTE") == 0)

{

strcpy(s2, "-");

len = strlen(opnd);

lc = lc + len - 2;

for (k = 2; k < len; k++)

{

q[l] = opnd[k];

l = l + 1;

}

fprintf(f5, "%s\t%s\n", q, s2);

break;

}

else if (strcmp(mne, "RESB") == 0)

lc = lc + atoi(opnd);

else if (strcmp(mne, "WORD") == 0)

{

strcpy(s3, "#");

lc = lc + 3;

fprintf(f5, "%s\t%s\n", opnd, s3);

break;

}

}

fseek(f2, SEEK\_SET, 0);

fscanf(f1, "%s%s%s", la, mne, opnd);

}

fseek(f5, SEEK\_SET, 0);

pgmlen = lc - sa;

printf("H^%s^%d^0%x\n", name, sa, pgmlen);

printf("T^");

printf("00%d^0%x", sa, pgmlen);

fscanf(f5, "%s%s", obj1, obj2);

while (!feof(f5))

{

if (strcmp(obj2, "0000") == 0)

printf("^%s%s", obj1, obj2);

else if (strcmp(obj2, "-") == 0)

{

printf("^");

len1 = strlen(obj1);

for (k = 0; k < len1; k++)

printf("%d", obj1[k]);

}

else if (strcmp(obj2, "#") == 0)

{

printf("^");

printf("%s", obj1);

}

fscanf(f5, "%s%s", obj1, obj2);

}

fseek(f5, SEEK\_SET, 0);

fscanf(f5, "%s%s", obj1, obj2);

while (!feof(f5))

{

if (strcmp(obj2, "0000") != 0)

{

if (strcmp(obj2, "-") != 0)

{

if (strcmp(obj2, "#") != 0)

{

printf("\n");

printf("T^%s^02^%s", obj1, obj2);

}

}

}

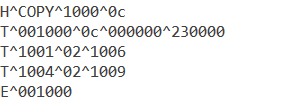
fscanf(f5, "%s%s", obj1, obj2);

}

printf("\nE^00%d", sa);

}

**Output:-**



**Two Pass Macro Processor**

**Program Code:-**

1. **Pass 1**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void main()

{

FILE \*f1, \*f2, \*f3;

char mne[20], opnd[20], la[20];

f1 = fopen("input.txt", "r");

f2 = fopen("namtab.txt", "w+");

f3 = fopen("deftab.txt", "w+");

fscanf(f1, "%s%s%s", la, mne, opnd);

while (strcmp(mne, "MEND") != 0)

{

if (strcmp(mne, "MACRO") == 0)

{

fprintf(f2, "%s\n", la);

fprintf(f3, "%s\t%s\n", la, opnd);

}

else

fprintf(f3, "%s\t%s\n", mne, opnd);

fscanf(f1, "%s%s%s", la, mne, opnd);

}

fprintf(f3, "%s", mne);

fclose(f1);

fclose(f2);

fclose(f3);

printf("\nPass 1 of 2 pass macroprocessor is successful.");

}

1. **Pass 2**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

void main()

{

FILE \*f1, \*f2, \*f3, \*f4, \*f5;

int i, len;

char mne[20], opnd[20], la[20], name[20], mne1[20], opnd1[20], arg[20];

f1 = fopen("input.txt", "r");

f2 = fopen("namtab.txt", "r");

f3 = fopen("deftab.txt", "r");

f4 = fopen("argtab.txt", "w+");

f5 = fopen("output.txt", "w");

fscanf(f1, "%s%s%s", la, mne, opnd);

while (strcmp(mne, "END") != 0)

{

if (strcmp(mne, "MACRO") == 0)

{

fscanf(f1, "%s%s%s", la, mne, opnd);

while (strcmp(mne, "MEND") != 0)

fscanf(f1, "%s%s%s", la, mne, opnd);

}

else

{

fscanf(f2, "%s", name);

if (strcmp(mne, name) == 0)

{

len = strlen(opnd);

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

{

if (opnd[i] != ',')

fprintf(f4, "%c", opnd[i]);

else

fprintf(f4, "\n");

}

fseek(f2, SEEK\_SET, 0);

fseek(f4, SEEK\_SET, 0);

fscanf(f3, "%s%s", mne1, opnd1);

fprintf(f5, ".\t%s\t%s\n", mne1, opnd);

fscanf(f3, "%s%s", mne1, opnd1);

while (strcmp(mne1, "MEND") != 0)

{

if ((opnd1[0] == '&'))

{

fscanf(f4, "%s", arg);

fprintf(f5, "-\t%s\t%s\n", mne1, arg);

}

else

fprintf(f5, "-\t%s\t%s\n", mne1, opnd1);

fscanf(f3, "%s%s", mne1, opnd1);

}

}

else

fprintf(f5, "%s\t%s\t%s\n", la, mne, opnd);

}

fscanf(f1, "%s%s%s", la, mne, opnd);

}

fprintf(f5, "%s\t%s\t%s\n", la, mne, opnd);

fclose(f1);

fclose(f2);

fclose(f3);

fclose(f4);

fclose(f5);

printf("\nPass 2 of 2 Pass Macroprocessor is Successful.");

}

**Output:-**

1. **Pass 1**



1. **Pass 2**



**Symbol Table with Hashing**

**Program Code:-**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX 11

char l[10];

struct symb

{

int add;

char label[10];

} sy[11];

void search()

{

FILE \*fp1;

char la[10];

int set = 0, s;

int j, i;

printf("\nEnter the Label:");

scanf("%s", la);

fp1 = fopen("symbol.txt", "r");

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

{

fscanf(fp1, "%d%d", &j, &sy[i].add);

if (sy[i].add != 0)

fscanf(fp1, "%s", sy[i].label);

}

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

{

if (sy[i].add != 0)

{

if (strcmp(sy[i].label, la) == 0)

{

set = 1;

s = sy[i].add;

}

}

}

if (set == 1)

printf("\nThe Label --%s-- is present in the Symbol Table at Address:%d", la, s);

else

printf("\nThe Label is Not Present in the Symbol Table!!");

}

void display(int a[MAX])

{

FILE \*fp;

int i;

fp = fopen("symbol.txt", "w");

printf("\nThe Symbol Table");

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

printf("\nHash Values\tAddress\tLabel");

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

{

printf("\n%d\t %d\t %s", i, sy[i].add, sy[i].label);

fprintf(fp, "\n%d %d %s", i, sy[i].add, sy[i].label);

}

fclose(fp);

}

int create(int num)

{

int key;

key = num % 11;

return key;

}

void lprob(int a[MAX], int key, int num)

{

int flag, i, count = 0;

flag = 0;

if (a[key] == 0)

{

a[key] = num;

sy[key].add = num;

strcpy(sy[key].label, l);

}

else

{

i = 0;

while (i < MAX)

{

if (a[i] != 0)

count++;

i++;

}

if (count == MAX)

{

printf("\nHash table is Full!!");

display(a);

exit(1);

}

for (i = key + 1; i < MAX; i++)

if (a[i] == 0)

{

a[i] = num;

flag = 1;

sy[key].add = num;

strcpy(sy[key].label, l);

break;

}

for (i = 0; i < key && flag == 0; i++)

if (a[i] == 0)

{

a[i] = num;

flag = 1;

sy[key].add = num;

strcpy(sy[key].label, l);

break;

}

}

}

void main()

{

int a[MAX], num, key, i, ch;

char ans = 'y';

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

a[i] = 0;

do

{

printf("\nSymbol Table Menu\n1.Create a Symbol Table\n2.Search in the Symbol Table\n3.Exit\nEnter your choice:");

scanf("%d", &ch);

switch (ch)

{

case 1:

while (ans == 'y')

{

printf("\nEnter the Address:");

scanf("%d", &num);

key = create(num);

printf("\nEnter The Label:");

scanf("%s", l);

lprob(a, key, num);

printf("\nDo you want to Continue(y/n)?");

scanf(" %c", &ans);

}

display(a);

break;

case 2:

search();

break;

case 3:

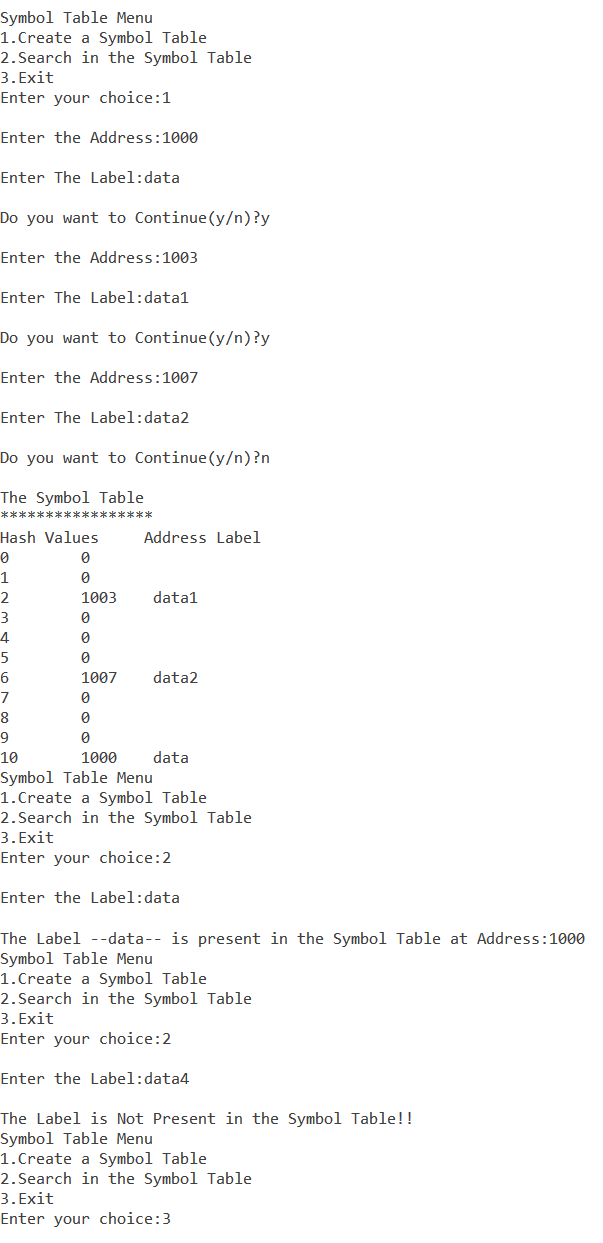
exit(0);

}

} while (ch <= 3);

}

**Output:-**



**Absolute Loader**

**Program Code:-**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

void main()

{

FILE \*fp;

int i, addr1, l, j, staddr1;

char name[10], line[50], name1[10], addr[10], rec[10], ch, staddr[10];

printf("Enter Program Name:");

scanf("%s", name);

fp = fopen("ablinput.txt", "r");

fscanf(fp, "%s", line);

for (i = 2, j = 0; i < 8, j < 6; i++, j++)

name1[j] = line[i];

name1[j] = '\0';

printf("\nName from Obj. %s\n", name1);

if (strcmp(name, name1) == 0)

{

do

{

fscanf(fp, "%s", line);

if (line[0] == 'T')

{

for (i = 2, j = 0; i < 8, j < 6; i++, j++)

staddr[j] = line[i];

staddr[j] = '\0';

staddr1 = atoi(staddr);

i = 12;

while (line[i] != '$')

{

if (line[i] != '^')

{

printf("00%d \t %c%c\n", staddr1, line[i], line[i + 1]);

staddr1++;

i = i + 2;

}

else

i++;

}

}

else if (line[0] = 'E')

;

} while (!feof(fp));

}

fclose(fp);

}

**Output:-**

