Name: Aryan Shinde
Roll no: 13
Gr no: 12010823
Class: Class_D

OS COURSE PROJECT Phase 1

Code:

```
#include<iostream>
#include<fstream>
using namespace std;

class test
{
   char M[100][4], IR[4], R[4], buffer[40];
   int IC, SI;
   bool C;
   int i,j,k;

   public:
           // ifstream in;
           // ofstream out;
           fstream Input;
           fstream Output;
           void LOAD();
           void INIT();
           void STARTEXECUTION();
           void EXECUTEUSERPROGRAM();
           void MOS();
           void READ();
           void WRITE();
           void TERMINATE();
```

```
};

void test::INIT()
{
    //cout<<"INIT function when AMJ encountered"<<endl;
   for (i=0; i<100; i++)
   {
       for (j=0; j<4; j++)
       {
           M[i][j] = '\0';
       }
   }
   IR[4]={'\0'};
   R[4]={'\0'};
   C = false;
}

void test::LOAD()
{

   int y=0;
   //in.open("in.txt");
   //Input.open("in.txt", ios::in);
   //cout<<"IN LOAD"<<endl;
   cout<<"Data loaded in memory:"<<endl;
   int num=0;
   do
   {
     //  num=0;
       for (i=0; i<40; i++)
       {
           buffer[i]='\0';
       }
       Input.getline(buffer,41);
       //cout<<buffer<<endl;

       if (buffer[0]=='$' && buffer[1]=='A' && buffer[2]=='M' &&
buffer[3]=='J')
       {
           //cout<<"AMJ encountered"<<endl;
```

```
            INIT();
        }
        else if (buffer[0]=='$' && buffer[1]=='D' && buffer[2]=='T' &&
buffer[3]=='A')
        {
            //cout<<"DTA encountered"<<endl;
            STARTEXECUTION();
        }
        else if (buffer[0]=='$' && buffer[1]=='E' && buffer[2]=='N' &&
buffer[3]=='D')
        {
            //cout<<"END encountered"<<endl;
            y=0;
            num=0;
            continue;
        }
        else   //PROGRAM CARD OR DATA CARD
        {
            //cout<<"PROGRAM CARD"<<endl;
            int x=0;
            for (i=num; i<100; i++)
            {
                cout<<i;
                for (j=0; j<4; j++)
                {
                    M[i][j] = buffer[x];
                    cout<<" "<<M[i][j];
                    x++;
                }
                num++;
                cout<<"\n";
                if (x==40 || buffer[x]==' ' || buffer[x]=='\n')
                {
                    break;
                }
            }
        }

    }while(!Input.eof());
```

```
}

void test::STARTEXECUTION()
{
    //cout<<"IN START EXECUTION"<<endl;
   IC = 00;
   EXECUTEUSERPROGRAM();
}

void test::EXECUTEUSERPROGRAM()
{
    //cout<<"IN EXECUTE USER PROGRAM"<<endl;
   while(true)
   {
       for (i=0; i<4; i++)
       {
           IR[i] = M[IC][i];
       }
       IC++;
       // for (i=0 ;i<4; i++)
       // {
       //     cout<<IR[i];
       // }
       // cout<<endl;
       // cout<<IR[2]<<endl;

       //int flag = int(IR[2]);
       int flag = (IR[2] - 48);
       flag = flag*10+(IR[3] - 48);
       //cout<<flag<<endl;

       if(IR[0]=='G' && IR[1]=='D')
       {
           SI = 1;
           MOS();
       }
       else if(IR[0]=='P' && IR[1]=='D')
       {
           SI = 2;
```

```
            MOS();
        }
        else if(IR [0]=='H')
        {
            SI = 3;
            MOS();
            break;
        }
        else if(IR[0]=='L' && IR[1]=='R')
        {


            for(int j=0;j<=3;j++)
                R[j]=M[flag][j];
        }

        else if(IR[0]=='S' && IR[1]=='R')
        {


            for(int j=0;j<=3;j++)
                M[flag][j]=R[j];

        }
        else if(IR[0]=='C' && IR[1]=='R')
        {

             if(M[flag][0]==R[0] && M[flag][1]==R[1] && M[flag][2]==R[2] &&
M[flag][3]==R[3]){
                C=true;
            }
        }
        else if(IR[0] =='B' && IR[1]=='T')
        {
            if(C==true)
            {
                IC = flag;
            }
        }
        // for (i=0;i<4;i++){
```

```
        //      cout<<R[i];
        // }



    }
}


void test::MOS()
{

        switch (SI)
        {
            case 1:
                    READ();
                    break;
            case 2:
                     WRITE();
                     break;
            case 3:
                     TERMINATE();
                     break;
        }

}

void test::READ()
{
    //cout<<"IN READ"<<endl;
    for (i=0; i<40; i++)
    {
        buffer[i] = '\0';
    }
    Input.getline(buffer,41);
    i=0;
    // for (i=0 ;i<40; i++)
    // {
    //      cout<<buffer[i];
    // }
```

```
    // cout<<"\n";
    //int flag = int(IR[2]);
    int flag = (IR[2] - 48) * 10; //10

    for (int x=0; x<10; x++)
    {
        cout<<flag;
        for (k=0; k<4; k++)
        {
            M[flag][k] = buffer[i];
            cout<<" "<<M[flag][k];
            i++;
        }
        if (i == 40)
        {
            break;
        }
        flag++;
        cout<<"\n";
     }
     cout<<endl;
     SI = 0;
}


void test::WRITE()
{
    //out.open("phase1.txt");
    //Output.open("phase1.txt", ios::out);
    //cout<<"IN WRITE"<<endl;
    for (i=0; i<40; i++)
    {
        buffer[i] = '\0';
    }
    i=0;
    //int flag = int(IR[2]);
    int flag = (IR[2] - 48) * 10;

    for (int x=0; x<10; x++)
    {
```

```
        //cout<<flag;
        for (k=0; k<4; k++)
        {
            buffer[i] = M[flag][k];
            if(buffer[i] != '\0')
            Output<<buffer[i];
            //cout<<" "<<buffer[i];
            i++;
        }
        if (i == 40)
        {
            break;
        }
        flag++;
        //cout<<"\n";
    }
    Output<<"\n";
    SI = 0;

}


void test::TERMINATE()
{
        // cout<<"IN TERMINATE"<<endl;
        // out<<"\n";
        // out<<"\n";
        Output<<"\n";
        Output<<"\n";
        SI = 0;
}

int main()
{
   test obj;

   obj.Input.open("in.txt", ios::in);
   obj.Output.open("out.txt", ios::out);
   obj.LOAD();
```

```
    return 0;
}
```

Input File:

```
$AMJ020100120003
GD20LR20GD30CR33BT07GD40PD40PD20PD30GD40
PD40H
$DTA
HOPE FOR IT
THERE IS NO HOPE
BUT STILL HOPE
$END0201
$AMJ020100120003
GD20LR20GD30CR33BT07GD40PD40PD20PD30GD40
PD40H
$DTA
HOPE FOR IT
THERE IS NO HOPE
BUT STILL HOPE
$END0201
```

Output:

out.txt

```
HOPE FOR IT
THERE IS NO HOPE
BUT STILL HOPE


HOPE FOR IT
THERE IS NO HOPE
BUT STILL HOPE


```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
Data loaded in memory:
0 G D 2 0
1 L R 2 0
2 G D 3 0
3 C R 3 3
4 B T 0 7
5 G D 4 0
6 P D 4 0
7 P D 2 0
8 P D 3 0
9 G D 4 0
10 P D 4 0
11 H
```

OS COURSE PROJECT Phase 2

Code:

```
#include<bits/stdc++.h>
using namespace std;

string IPFILE = "job.txt";

char Memory[300][4];
vector<char> R(4), IR(4);
int IC=0, SI=3, job =1, PI =0, TI = 0, cnt = 0, TTL = 0, LineLImit = 0,
PTR = 0, Fill;
int input_ptr=0, output_ptr=0;
bool C=false;
string buffer="";
set<int>randnos;

struct PCB
{
   int TTL, TLL, TTC, TLC;
};
struct PCB jobs[10];


void print_mem()
{
   cout<<"Memory for job "<<job<<"\n";
   for(int i=0; i<300; i++)
   {
        for(int k=0; k<4; k++)
        {
            cout<<Memory[i][k]<<" ";
        }
        cout<<endl;
   }
}
```

```
void clearContents()
{
   // Memory Initialized by '_'
   for(int i=0;i<300;i++)
   {
       for(int j=0;j<4;j++)
       {
           Memory[i][j] = '_';
       }
   }
   buffer.clear();
   R.clear();
   randnos.clear();
   IC=0;
   C = false;
}

void lineReader(string temp)
{
   jobs[cnt].TLC++;
   fstream fout;
   fout.open("output.txt");
   fout.seekg(0, ios::end);
   fout<<temp;
   fout<<"\n";
   fout.close();

}

void READ(int block)
{
   //Obtaining data from input file
   fstream fin;
   string line;
   fin.open(IPFILE, ios::in);
   for(int i=0; i<input_ptr; i++)
   {
       getline(fin, line);
   }
```

```
    // WE SKIP LINES
    getline(fin, line);// THIS LINE IS PROGRAM CARD
    input_ptr++;
    fin.close();

    buffer = line;
    for(int i=0; i<buffer.size() && i<40; i++)
    {
        int a = i/4 + block;
        int b = i%4;
        Memory[a][b] = buffer[i];
    }
}

void WRITE(int block)
{
    string temp = "";
    int i=0;
    while(i<40)
    {
        int a = i/4 + block;
        int b = i%4;
        if(Memory[a][b] == '_')
        {
            temp.push_back(' ');
        }
        else
            temp.push_back(Memory[a][b]);
        i++;
    }

    lineReader(temp);

}

int TERMINATE(int si)
{
    if(si==3)
        return 1;
    else
```

```
        return 0;
}

int AddressMap(int VA){
   // returns REAL ADDRESS

   int RA;
   int PTE = PTR + VA/10;
   if(Memory[PTE][2]=='*' || Memory[PTE][3]=='*')
   {
        // Page fault
        PI=3;
        // Now handle it by generating a new random number to fill

        bool flag = true;

        //int Entry = 12;
        int Entry = rand()%30;
        if(randnos.find(Entry*10)==randnos.end()){
            randnos.insert(Entry*10);
        }
        else{
            // number already exists
            while(randnos.find(Entry*10)!=randnos.end()){
                Entry = rand()%30;
            }
        }

        string EntryStr = to_string(Entry);

         if(EntryStr.size() == 1){
            Memory[PTE][2] = '0';
            Memory[PTE][3] = EntryStr[0];
         }
         else{
            Memory[PTE][2] = EntryStr[0];
            Memory[PTE][3] = EntryStr[1];
         }

        RA = Entry*10 + VA%10;
```

```
    }

    else{

        // Page Exists
        string t = "";
        t.push_back(Memory[PTE][2]);
        t.push_back(Memory[PTE][3]);

        int no = stoi(t);
        RA = no*10 + VA%10;

    }

    return RA;

}



void executeUserProgram()
{
    vector<string> ins = {"GD","PD","CR","SR","LR","BT"};

    while(1)
    {

        // Now start fetching instructions from Fill Index which is First
Entry * 10

        //Bringing instr to IR
        jobs[cnt].TTC++;
        string Opc_ins = "", Opr_ins = "";
        for(int i=0; i<4; i++)
        {
            IR[i] = Memory[IC][i];
        }

        //Halt is checked first only
```

```
        if(IR[0]=='H')
        {
            TERMINATE(SI);
            break;
        }

        //check operand error  (3 digit or negative or characters in place
in numbers)
        if(IR[2] == '-' || isdigit(IR[0]) || isalpha(IR[2]) ||
isalpha(IR[3])){
            PI = 2;
            break;
        }

        Opc_ins+= Memory[IC][0];
        Opc_ins+= Memory[IC][1];

        // check opcode error
        if (find(ins.begin(), ins.end(), Opc_ins) == ins.end()){ // not in
vector (2 lettered handled)
            PI = 1;
            break;
        }


        //Address Extraction
        //cout<<"IR: "<<IR[0]<<" "<<IR[1]<<" "<<IR[2]<<" "<<IR[3]<<endl;
        int t1 = ((int)IR[2]) - ((int)'0');
        int t2 = ((int)IR[3]) - ((int)'0');
        int block = t1*10 + t2;
        //cout<<"Start Block: "<<block<<"\n\n";

        int RealAdd = AddressMap(block);



        //Checking Instruction
        if(IR[0]=='G' && IR[1]=='D')
        {
            SI=1;
```

```
            READ(RealAdd);
            jobs[cnt].TTC++;// Because in Valid Page faults one more time
quantum is required
        }

        else if(IR[0]=='P' && IR[1]=='D')
        {
            SI=2;
            WRITE(RealAdd);
        }


        else if(IR[0]=='L' && IR[1]=='R')
        {
            cout<<"\n"<<endl;

            for(int i=0; i<4; i++)
            {
                R[i] = Memory[RealAdd][i];
            }
        }

        else if(IR[0]=='C' && IR[1]=='R')
        {
            bool f = 0;

            for(int i=0; i<4; i++)
            {
                if(R[i]!=Memory[RealAdd][i])
                {
                    f=1;
                    break;
                }
            }

            if(f) C = false;
            else C = true;
        }
```

```
        else if(IR[0]=='S' && IR[1]=='R')
        {
            for(int j=0; j<4; j++)
            {
                Memory[RealAdd][j] = R[j];
            }
            jobs[cnt].TTC++;// Because in Valid Page faults one more time
quantum is required
        }

        else if(IR[0]=='B' && IR[1]=='T')
        {
            if(C==true)
            {
                IC = RealAdd-1;
            }
        }

        //Incrementing program counter
        IC++;
    }

    //cout<<"\n\n #############    Program DONE    ########### \n\n";
}

void startExecution()
{
    IC = Fill;
    executeUserProgram();

    //Leaving two lines for each jobs
    fstream fout;
    fout.open("output.txt");
    fout.seekg(0, ios::end);
    fout<<"\n\n";
    fout.close();
}

void SetPageTable(int PTR){
    for(int i=0; i<10; i++){
```

```
        Memory[PTR+i][0] = '*';
        Memory[PTR+i][1] = '*';
        Memory[PTR+i][2] = '*';
        Memory[PTR+i][3] = '*';
    }
}


void LOAD()
{
    fstream fin;
    string line ;

    while (1)
    {
        //Start file handling

        fin.open(IPFILE, ios::in);
        for(int j=0; j<input_ptr; j++)
        {
            getline(fin, line);
        }

        getline(fin, line);
        input_ptr++;
        fin.close();


        //For last job in input file
        if(line.substr(0,4)=="$END")
            break;


        //For next job in input file
        if(line.substr(0, 4) =="$AMJ")
        {
            //Clear Contents for a new JOB
            PI = 0;
            clearContents();
```

```
            // FInd TTL
            TTL = stoi(line.substr(8,4));

            // FInd Line Limit
            LineLImit = stoi(line.substr(12,4));

            // Initialize PCB
            struct PCB CJob;
            CJob.TTL = TTL;
            CJob.TLL = LineLImit;
            CJob.TTC = 0;
            CJob.TLC = 0;
            jobs[cnt]= CJob;

            // Generate PTR

            PTR = rand()%30;
            // Now to prevent random numbers to lie in page table add page
table row numbers to a set

            PTR = PTR *10;
            for(int i=PTR;i<(PTR+11);i++){
                randnos.insert(i);
            }// Page Table Avoided


            // Now initialize page table at PTR*10
            SetPageTable(PTR);

            // Generate first entry
            int First_Entry = rand()%30; // Allocate Fnction


            //FIll FIrst entry in row 1 pointed by PTR
            string FE = to_string(First_Entry);
            if(FE.size() == 1){
                Memory[PTR][2] = '0';
                Memory[PTR][3] = FE[0];
            }
            else{
```

```
            Memory[PTR][2] = FE[0];
            Memory[PTR][3] = FE[1];
        }

        // Now go to First entry block and start filling program card
        Fill = First_Entry * 10;
        randnos.insert(Fill);



        //Job started loading
        string prog;
        fin.open(IPFILE, ios::in);
        for(int j=0; j<input_ptr; j++)
        {
            getline(fin, prog);
        }
        getline(fin, prog);
        input_ptr++;
        //cout<<"Program Card: "<<prog<<endl;

        // We are now loading Program card
        int k=Fill;
        int b = 0,rw = 0;
        while(prog.substr(0,4)!="$DTA")
        {

            //Loading inst to memory!
            for(int j=0; j<prog.size(); j++)
            {
                randnos.insert(k); // this is to avoid any data to get
inserted into program card segment
                if(prog[j]=='H')
                {
                    Memory[k][0] = 'H';
                    k++;
                    continue;
                }

                else
```

```
            {
                Memory[k][b] = prog[j];
                rw++;
            }
            if(rw%4 == 0){
                k++;
                b = -1;
                rw = 0;
            }
            b++;
        }

        getline(fin, prog);

        input_ptr++;
    }
    // Program card loaded

    // here line ptr is at the starting part of Data seg
    getline(fin, prog);

    // If no data, it will directly point to $END
    if(prog.substr(0,4)=="$END")
    {
        cout<<"Out of Data Error! \n";
    }

    fin.close();

    startExecution();

    // JOB IS DONE ##########################



    //TTL errors
    //cout<<jobs[cnt].TTL<<jobs[cnt].TTC;
    if(jobs[cnt].TTL<jobs[cnt].TTC)
    {
        TI = 2;
```

```
                cout<<"Time Limit Exceeded! It took "<<jobs[cnt].TTC<<"
units of time. Expected TTL was "<<jobs[cnt].TTL<<"\n";
                //break;
            }

            //Line Limit errors
            //cout<<"Line limit"<<jobs[cnt].TLC<<"expected lines
"<<jobs[cnt].TLL;
            else if(jobs[cnt].TLL<jobs[cnt].TLC)
            {
                cout<<"Line Limit Exceeded! It printed "<<jobs[cnt].TLC<<"
lines. Expected Line Limit was "<<jobs[cnt].TLL<<"\n";
                //break;
            }

            //PI Errors
            else if(PI == 1)
            {
                cout<<"Opcode Error: \n";
                cout<<"JOb No "<<job<<" is not executed \n";
            }
            else if(PI == 2)
            {
                cout<<"Operand Error: \n";
                cout<<"JOb No "<<job<<" is not executed \n";
            }
            else
            {
                print_mem();
            }
            job++; // job variable incremented
            cnt++; // pointer for array of PCBs
        }

        //Read Input till end of program
        fin.open(IPFILE, ios::in);
        for(int j=0; j<input_ptr; j++)
        {
            getline(fin, line);
        }
```

```
        getline(fin, line);
        input_ptr++;

        while(line.substr(0, 4)!="$END")
        {
            getline(fin, line);
            input_ptr++;
        }
        getline(fin, line);

    }
    fin.close();

}




int main()
{
    LOAD();
    return 0;
}
```

Input:

$AMJ020200250004
GD20PD20LR20SR30SR31PD30SR40SR41SR42PD40
SR50SR51PD50SR60PD60H
$DTA
*
$END0202
$AMJ030200100002
GD20GD30LR31SR22LR32SR23PD20SR40PD40H
$DTA
CAT CAN
  EAT RAT

$END0302
$AMJ010200070002
GD20LR36CR20BT06GD30PD30PD20H
$DTA
RAM IS OLDER THAN SHRIRAM
NOT IN EXISTANCE
$END0102
$AMJ040100090004
GD20PD20GD30PD30GD40GD50LR20CR30BT10PD40
PD50H
$DTA
ABCD
ABCD
DO NOT
MATCH
$END0401
$AMJ150300200010
GD20GD30LR30SR7AGD40LR40SR74GD50LR50
SR75GD60GD80LR80SR71GD90LR90SR72PD70H
$DTA
SHE WENT
 TO
GET
HER
BAG
WE
WORK
$END1503
$AMJ140300500008
GD30LR33SR37GD40LR40SR38LR41SR39PA30
H
$DTA
SHE SELLS   SEA SHELLS ON
SHORE
$END1403

```
$AMJ140300500008
GD30LR33SR37GD40LR40SR38LR41SR39PA30
H
$DTA
$END1403
$AMJ040200040002
GD30PD30LR30SR40PD3FH
$DTA
SHE SELLS   SEA SHELLS ON
SHORE
$END0402
$AMJ040300040002
GD30PD30LR30SR40PS40H
$DTA
SHE SELLS   SEA SHELLS ON
SHORE
$END0403
$AMJ040500030002
GD30PD30LR30SR32PD40H
$DTA
SHE SELLS   SEA SHELLS ON
SHORE
$END0405
```

Output:

```
[Running] cd "/home/akshay/Documents/TY/Os/cp/phase2/" && g++ phase2git.cpp -o phase2git && "/home/
akshay/Documents/TY/Os/cp/phase2/"phase2git


Line Limit Exceeded! It printed 5 lines. Expected Line Limit was 4



Time Limit Exceeded! It took 15 units of time. Expected TTL was 10


Time Limit Exceeded! It took 10 units of time. Expected TTL was 7


Time Limit Exceeded! It took 14 units of time. Expected TTL was 9


Operand Error:
JOb No 5 is not executed




Opcode Error:
JOb No 6 is not executed
Out of Data Error!
```

```
*
* *
* * *
* *
*
```

```
CAT CAN EAT RAT
RAT
```

```
NOT IN EXISTANCE
RAM  IS OLDER THAN  SHRIRAM
```

```
ABCD
ABCD
```

```
SHE SELLS   SEA SHELLS ON
```

```
SHE SELLS   SEA SHELLS ON
```

```
*
* *
* * *
* *
*
```

```
CAT CAN EAT RAT
RAT
```

NOT IN EXISTANCE
RAM IS OLDER THAN SHRIRAM

ABCD
ABCD

SHE SELLS   SEA SHELLS ON

SHE SELLS   SEA SHELLS ON

*
* *
* * *
* *
*

CAT CAN EAT RAT
RAT

NOT IN EXISTANCE
RAM IS OLDER THAN SHRIRAM

ABCD
ABCD

SHE SELLS   SEA SHELLS ON

SHE SELLS   SEA SHELLS ON