**Name : Yadnesh Patil**

**Div : C**

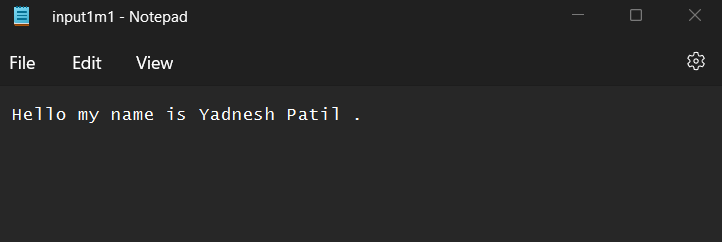
**Roll No : 23**

**PRN:12011302**

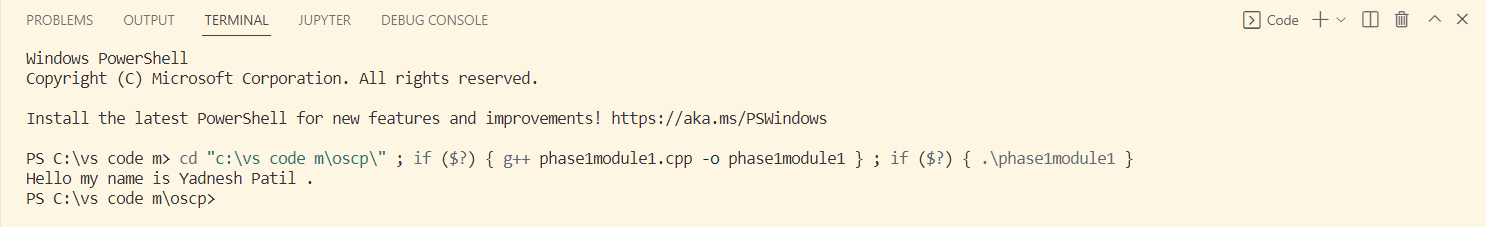
**Operating System Course Project- Phase 1**

1. Load program card in external main memory Block B0 or Block B0 B1.
2. #include <bits/stdc++.h>
3. using namespace std;
4. char M[100][4];
5. char R[4];
6. char IR[4];
7. int IC;
8. int C;
9. char Buff[40];
10. fstream input;
11. fstream output;
12. void init()
13. {
14. for (int i = 0; i < 100; i++)
15. {
16. for (int j = 0; j < 4; j++)
17. {
18. M[i][j] = ' ';
19. }
20. }
21. for (int i = 0; i < 4; i++)
22. {
23. IR[i] = ' ';
24. R[i] = ' ';
25. }
26. int IC = 0;
27. int C = 0; // toggle register
28. }
29. void load()
30. {
31. do
32. {
33. for (int i = 0; i < 40; i++)
34. Buff[i] = '\0';
35. input.getline(Buff, 40);
36. for (int i = 0; i < 40; i++)
37. cout << Buff[i];
38. if (Buff[0] == '$' && Buff[1] == 'A' && Buff[2] == 'M' && Buff['3'] == 'J')
39. {
40. init();
41. }
42. else if (Buff[0] == '$' && Buff[1] == 'D' && Buff[2] == 'T' && Buff['3'] == 'A')
43. {
44. // se();
45. }
46. else if (Buff[0] == '$' && Buff[1] == 'E' && Buff[2] == 'N' && Buff['3'] == 'D')
47. {
48. // CONTINUE;
49. }
50. else
51. {
52. int k = 0;
53. for (int i = 0; i < 100; i++)
54. for (int j = 0; j < 4; j++)
55. {
56. M[i][j] = Buff[k];
57. k++;
58. }
59. if (k == 40 || k == ' ' || k == '\n' || k == '\0')
60. {
61. break;
62. }
63. }
64. // break;
65. } while (!input.eof());
66. }
67. int main()
68. {
69. input.open("input1m1.txt", ios::in);
70. do
71. {
72. input.getline(Buff, 40);
73. for (int i = 0; i < 40; i++)
74. cout << Buff[i];
75. } while (!input.eof());
76. load();
77. output.open("output1m1n.txt", ios::out);
78. }

INPUT:



OUTPUT:



2.MOS module code which will execute GD, PD and H instructions of program.

**Code:**

#include <iostream>

#include <fstream>

using namespace std;

class OS

{

private:

    char M[100][4]; // Physical Memory

    char IR[4];     // Instruction Register (4 bytes)

    char R[4];      // General Purpose Register (4 bytes)

    int IC;         // Instruction Counter Register (2 bytes)

    int SI;         // Interrupt

    bool C;         // Toggle (1 byte)

    char buffer[40];

public:

    void init();

    void LOAD();

    void Execute();

    void MOS();

    fstream infile;

    fstream outfile;

};

// initiallise everything to 0

void OS::init()

{

    for (int i = 0; i < 100; i++)

    {

        for (int j = 0; j < 4; j++)

        {

            M[i][j] = ' ';

        }

    }

    IR[0] = {' '};

    R[0] = {' '};

    C = false;

}

// Master Mode

void OS::MOS()

{

    if (SI == 1) // Read Mode

    {

        for (int i = 0; i <= 39; i++)

            buffer[i] = '\0';

        infile.getline(buffer, 40);

        int k = 0;

        int i = IR[2] - 48;

        i = i \* 10;

        for (int l = 0; l < 10; ++l)

        {

            for (int j = 0; j < 4; ++j)

            {

                M[i][j] = buffer[k];

                k++;

            }

            if (k == 40)

            {

                break;

            }

            i++;

        }

        for (int i = 0; i < 100; i++)

        {

            cout << "M[" << i << "]\t";

            for (int j = 0; j < 4; j++)

            {

                cout << M[i][j];

            }

            cout << endl;

        }

    }

    else if (SI == 2) // Write Mode

    {

        for (int i = 0; i <= 39; i++)

            buffer[i] = '\0';

        int k = 0;

        int i = IR[2] - 48;

        i = i \* 10;

        for (int l = 0; l < 10; ++l)

        {

            for (int j = 0; j < 4; ++j)

            {

                buffer[k] = M[i][j];

                outfile << buffer[k];

                k++;

            }

            if (k == 40)

            {

                break;

            }

            i++;

        }

        for (int i = 0; i < 100; i++)

        {

            cout << "M[" << i << "]\t";

            for (int j = 0; j < 4; j++)

            {

                cout << M[i][j];

            }

            cout << endl;

        }

        outfile << "\n";

    }

    else if (SI == 3) // Terminate

    {

        outfile << "\n";

        outfile << "\n";

    }

}

// Execution

void OS::Execute()

{

    while (true)

    {

        for (int i = 0; i < 4; i++) // Load in register

        {

            IR[i] = M[IC][i];

        }

        IC++;

        if (IR[0] == 'G' && IR[1] == 'D') // GD

        {

            SI = 1;

            MOS();

        }

        else if (IR[0] == 'P' && IR[1] == 'D') // PD

        {

            SI = 2;

            MOS();

        }

        else if (IR[0] == 'H') // H

        {

            SI = 3;

            MOS();

            break;

        }

        /\*  else if(IR[0] == 'L' && IR[1] == 'R')       //LR

          {

              int i = IR[2]-48;

              i = i\*10 + ( IR[3]-48);

              for(int j=0;j<=3;j++)

                  R[j]=M[i][j];

              //for(int j=0;j<=3;j++)

                // cout<<R[j];

              cout<<endl;

          }

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

          {

              int i = IR[2]-48;

              i = i\*10 +( IR[3]-48) ;

              //cout<<i;

              for(int j=0;j<=3;j++)

                  M[i][j]=R[j];

              cout<<endl;

          }

          else if(IR[0] == 'C' && IR[1] == 'R')       //CR

          {

              int i = IR[2]-48;

              i = i\*10 + (IR[3] - 48);

              //cout<<i;

              int count=0;

              for(int j=0;j<=3;j++)

                  if(M[i][j] == R[j])

                      count++;

              if(count==4)

                  C=true;

              //cout<<C;

          }

          else if(IR[0] == 'B' && IR[1] == 'T')       //BT

          {

              if(C == true)

              {

                  int i = IR[2]-48;

                  i = i\*10 + (IR[3] - 48);

                  IC = i;

              }

          }\*/

    }

}

// Load Function

void OS::LOAD()

{

    cout << "Reading Data..." << endl;

    int x = 0;

    do

    {

        for (int i = 0; i <= 39; i++) // clear buffer

            buffer[i] = '\0';

        infile.getline(buffer, 40);

        for (int k = 0; k <= 39; k++)

            cout << buffer[k];

        if (buffer[0] == '$' && buffer[1] == 'A' && buffer[2] == 'M' && buffer[3] == 'J')

        {

            init();

        }

        else if (buffer[0] == '$' && buffer[1] == 'D' && buffer[2] == 'T' && buffer[3] == 'A')

        {

            IC = 00;

            Execute();

        }

        else if (buffer[0] == '$' && buffer[1] == 'E' && buffer[2] == 'N' && buffer[3] == 'D')

        {

            x = 0;

            continue;

        }

        else

        {

            int k = 0;

            for (; x < 100; ++x)

            {

                for (int j = 0; j < 4; ++j)

                {

                    M[x][j] = buffer[k];

                    k++;

                }

                if (k == 40 || buffer[k] == ' ' || buffer[k] == '\n')

                {

                    break;

                }

            }

        }

    } while (!infile.eof()); // continues to take input till eof

}

int main()

{

    OS os;

    os.infile.open("input1m2.txt", ios::in);

    os.outfile.open("output1m2.txt", ios::out);

    if (!os.infile)

    {

        cout << "Failure" << endl;

    }

    else

    {

        cout << "File Exist" << endl;

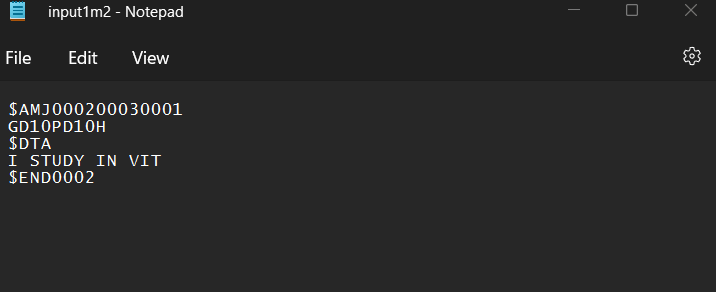
    }

    os.LOAD();

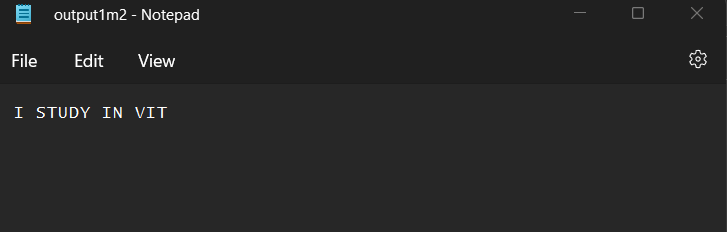
    return 0;

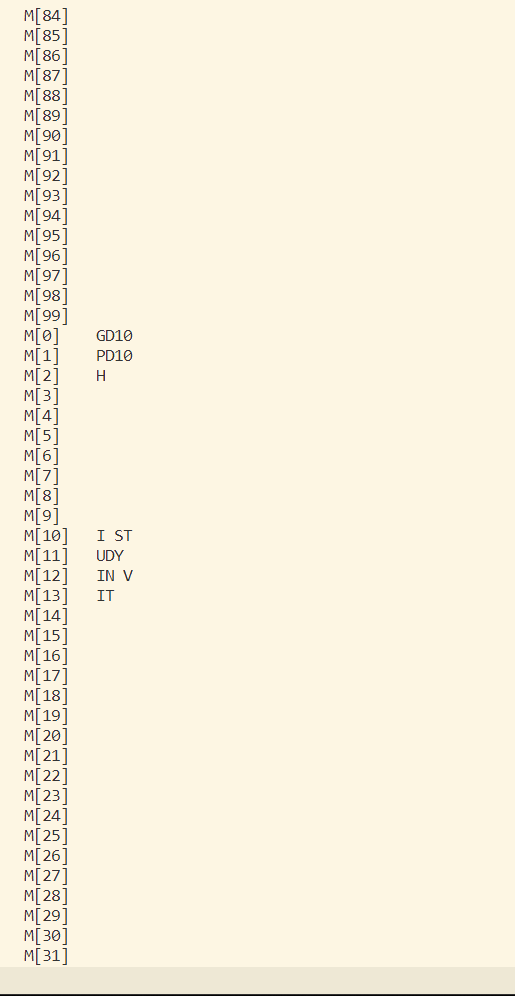
}

INPUT:



OUTPUT:





3.Slave (User) module code which will execute LR, SR, CR, BT instructions of program.

**Code:**

#include <iostream>

#include <fstream>

using namespace std;

class OS

{

private:

    char M[100][4]; // Physical Memory

    char IR[4];     // Instruction Register (4 bytes)

    char R[4];      // General Purpose Register (4 bytes)

    int IC;         // Instruction Counter Register (2 bytes)

    int SI;         // Interrupt

    bool C;         // Toggle (1 byte)

    char buffer[40];

public:

    void init();

    void LOAD();

    void Execute();

    void MOS();

    fstream infile;

    fstream outfile;

};

// initiallise everything to 0

void OS::init()

{

    for (int i = 0; i < 100; i++)

    {

        for (int j = 0; j < 4; j++)

        {

            M[i][j] = ' ';

        }

    }

    IR[0] = {' '};

    R[0] = {' '};

    C = false;

}

// Master Mode

void OS::MOS()

{

    if (SI == 1) // Read Mode

    {

        for (int i = 0; i <= 39; i++)

            buffer[i] = '\0';

        infile.getline(buffer, 40);

        int k = 0;

        int i = IR[2] - 48;

        i = i \* 10;

        for (int l = 0; l < 10; ++l)

        {

            for (int j = 0; j < 4; ++j)

            {

                M[i][j] = buffer[k];

                k++;

            }

            if (k == 40)

            {

                break;

            }

            i++;

        }

        for (int i = 0; i < 100; i++)

        {

            cout << "M[" << i << "]\t";

            for (int j = 0; j < 4; j++)

            {

                cout << M[i][j];

            }

            cout << endl;

        }

    }

    else if (SI == 2) // Write Mode

    {

        for (int i = 0; i <= 39; i++)

            buffer[i] = '\0';

        int k = 0;

        int i = IR[2] - 48;

        i = i \* 10;

        for (int l = 0; l < 10; ++l)

        {

            for (int j = 0; j < 4; ++j)

            {

                buffer[k] = M[i][j];

                outfile << buffer[k];

                k++;

            }

            if (k == 40)

            {

                break;

            }

            i++;

        }

        for (int i = 0; i < 100; i++)

        {

            cout << "M[" << i << "]\t";

            for (int j = 0; j < 4; j++)

            {

                cout << M[i][j];

            }

            cout << endl;

        }

        outfile << "\n";

    }

    else if (SI == 3) // Terminate

    {

        outfile << "\n";

        outfile << "\n";

    }

}

// Execution

void OS::Execute()

{

    while (true)

    {

        for (int i = 0; i < 4; i++) // Load in register

        {

            IR[i] = M[IC][i];

        }

        IC++;

        if (IR[0] == 'G' && IR[1] == 'D') // GD

        {

            SI = 1;

            MOS();

        }

        else if (IR[0] == 'P' && IR[1] == 'D') // PD

        {

            SI = 2;

            MOS();

        }

        else if (IR[0] == 'H') // H

        {

            SI = 3;

            MOS();

            break;

        }

        else if (IR[0] == 'L' && IR[1] == 'R') // LR

        {

            int i = IR[2] - 48;

            i = i \* 10 + (IR[3] - 48);

            for (int j = 0; j <= 3; j++)

                R[j] = M[i][j];

            // for(int j=0;j<=3;j++)

            //  cout<<R[j];

            cout << endl;

        }

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

        {

            int i = IR[2] - 48;

            i = i \* 10 + (IR[3] - 48);

            // cout<<i;

            for (int j = 0; j <= 3; j++)

                M[i][j] = R[j];

            cout << endl;

        }

        else if (IR[0] == 'C' && IR[1] == 'R') // CR

        {

            int i = IR[2] - 48;

            i = i \* 10 + (IR[3] - 48);

            // cout<<i;

            int count = 0;

            for (int j = 0; j <= 3; j++)

                if (M[i][j] == R[j])

                    count++;

            if (count == 4)

                C = true;

            // cout<<C;

        }

        else if (IR[0] == 'B' && IR[1] == 'T') // BT

        {

            if (C == true)

            {

                int i = IR[2] - 48;

                i = i \* 10 + (IR[3] - 48);

                IC = i;

            }

        }

    }

}

// Load Function

void OS::LOAD()

{

    cout << "Reading Data..." << endl;

    int x = 0;

    do

    {

        for (int i = 0; i <= 39; i++) // clear buffer

            buffer[i] = '\0';

        infile.getline(buffer, 40);

        for (int k = 0; k <= 39; k++)

            cout << buffer[k];

        if (buffer[0] == '$' && buffer[1] == 'A' && buffer[2] == 'M' && buffer[3] == 'J')

        {

            init();

        }

        else if (buffer[0] == '$' && buffer[1] == 'D' && buffer[2] == 'T' && buffer[3] == 'A')

        {

            IC = 00;

            Execute();

        }

        else if (buffer[0] == '$' && buffer[1] == 'E' && buffer[2] == 'N' && buffer[3] == 'D')

        {

            x = 0;

            continue;

        }

        else

        {

            int k = 0;

            for (; x < 100; ++x)

            {

                for (int j = 0; j < 4; ++j)

                {

                    M[x][j] = buffer[k];

                    k++;

                }

                if (k == 40 || buffer[k] == ' ' || buffer[k] == '\n')

                {

                    break;

                }

            }

        }

    } while (!infile.eof()); // continues to take input till eof

}

int main()

{

    OS os;

    os.infile.open("input3m3.txt", ios::in);

    os.outfile.open("output3m3.txt", ios::out);

    if (!os.infile)

    {

        cout << "Failure" << endl;

    }

    else

    {

        cout << "File Exist" << endl;

    }

    os.LOAD();

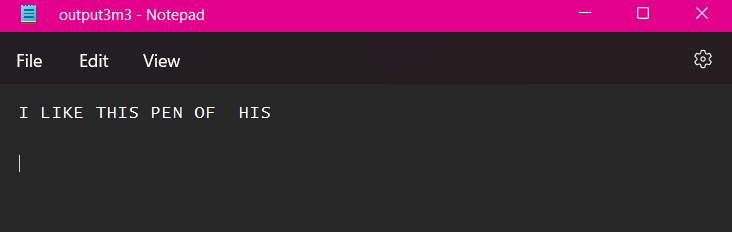
    return 0;

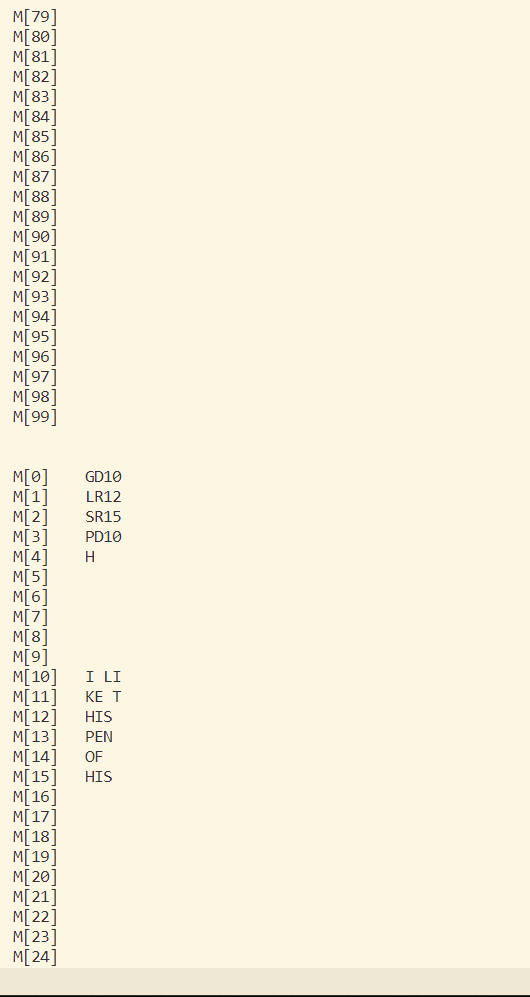
}

**INPUT:**

****

**OUTPUT:**

****

****

**PHASE 2**

#include <bits/stdc++.h>

using namespace std;

class PCB

{

public:

    int job\_id, ttl, tll, ttc, llc;

};

class os\_phase2

{

public:

    int index = 0;

    char M[300][4], IR[4], R[4], buffer[40], temp\_buff[4];

    int i, C, IC, VA, RA, PTR, SI = 0, TI = 0, PI = 0, loc, PTE, temp\_loc;

    PCB pcb;

    // int time = 0;

    vector<string> EM{"No Error", "Out of Data", "Line Limit Exceeded",

                      "Time Limit Exceeded", "Operation Code Error", "Operand Error", "Invalid Page Fault"};

    vector<int> rnumbers;

    vector<string> lines;

    ifstream ip\_file;

    ofstream op\_file;

    string line;

    os\_phase2()

    {

        for (i = 0; i < 29; i++)

        {

            rnumbers.push\_back(i);

        }

    }

    int random()

    {

        unsigned seed = chrono::system\_clock::now().time\_since\_epoch().count();

        shuffle(rnumbers.begin(), rnumbers.end(), default\_random\_engine(seed));

        int r = rnumbers[0];

        rnumbers.erase(rnumbers.begin());

        return r \* 10;

    }

    void init()

    {

        int i, j;

        // Initialize external memory

        for (i = 0; i < 300; i++)

            for (j = 0; j < 4; j++)

                M[i][j] = '\*';

        // Initialize GPR and IR

        for (i = 0; i < 4; i++)

            IR[i] = R[i] = '\*';

        // Initialize counter to 0

        C = IC = SI = TI = PI = PTR = 0;

    }

//clears the buffer

    void clear()

    {

        int i;

        for (i = 0; i < 40; i++)

        {

            buffer[i] = '\0';

        }

    }

//reads a line from input file and store in buffer

    void read()

    {

        char c;

        int i = 0;

        clear();

        do

        {

            ip\_file.get(c);

            buffer[i] = c;

            i++;

        } while (c != '\n');

    }

    void Read()

    {

        int i, j, k;

        read();

        i = loc;

        k = 0;

        while (i < loc + 10)

        {

            j = 0;

            while (j < 4 && buffer[k] != '\0')

            {

                M[i][j] = buffer[k];

                k++;

                j++;

            }

            i++;

        }

    }

    int location()

    {

        return ((int)IR[2] - 48) \* 10 + ((int)IR[3] - 48);

    }

    void Write()

    {

        pcb.llc++;

        if (pcb.llc > pcb.tll)

        {

            Terminate(2);

        }

        int i, j, k;

        IR[3] = '0';

        k = 0;

        clear();

        i = loc;

        while (i < loc + 10)

        {

            j = 0;

            while (j < 4 && M[i][j] != '\*')

            {

                if (M[i][j] == '\n')

                    buffer[k] = ' ';

                else

                    buffer[k] = M[i][j];

                k++;

                j++;

            }

            i++;

        }

        buffer[k] = '\n';

        i = 0;

        while (buffer[i] != '\n')

        {

            op\_file.put(buffer[i]);

            i++;

        }

        op\_file << '\n';

    }

    void Terminate(int n)

    {

        if (!op\_file.is\_open())

        {

            op\_file.open("output.txt", ios::app);

        }

        op\_file <<EM[n] << "\n";

        op\_file << "\n\n";

        op\_file.close();

        exit(0);

    }

    void operations()

    {

        switch (SI)

        {

        case 1:

            Read();

            break;

        case 2:

            Write();

            break;

        case 3:

            Terminate(0);

            break;

        }

    }

    void Execute()

    {

        addressMap(IC);

        do

        {

            IR[0] = M[RA][0]; // RA = 110

            IR[1] = M[RA][1];

            IR[2] = M[RA][2];

            IR[3] = M[RA][3];

            loc = location();

            IC++;

            if (IR[0] == 'L' && IR[1] == 'R')

            {

                pcb.ttc++;

                if (pcb.ttc > pcb.ttl)

                {

                    TI = 2;

                    MOS();

                }

                R[0] = M[loc][0];

                R[1] = M[loc][1];

                R[2] = M[loc][2];

                R[3] = M[loc][3];

                RA++;

            }

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

            {

                pcb.ttc += 2;

                if (pcb.ttc > pcb.ttl)

                {

                    TI = 2;

                    MOS();

                }

                M[loc][0] = R[0];

                M[loc][1] = R[1];

                M[loc][2] = R[2];

                M[loc][3] = R[3];

                RA++;

            }

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

            {

                pcb.ttc++;

                if (pcb.ttc > pcb.ttl)

                {

                    TI = 2;

                    MOS();

                }

                if (R[0] == M[loc][0] && R[1] == M[loc][1] && R[2] == M[loc][2] && R[3] == M[loc][3])

                    C = 1;

                else

                    C = 0;

                RA++;

            }

            else if (IR[0] == 'B' && IR[1] == 'T')

            {

                pcb.ttc++;

                if (pcb.ttc > pcb.ttl)

                {

                    TI = 2;

                    MOS();

                }

                if (C == 1)

                    IC = loc;

                RA++;

            }

            else if (IR[0] == 'G' && IR[1] == 'D')

            {

                SI = 1;

                VA = (IR[2] - '0') \* 10 + (IR[3] - '0');

                if (!(IR[2] <= 57 && IR[3] <= 57)) // to check oprand error

                {

                    PI = 2;

                }

                // if (PTE == VA)

                //     allocate();

                // addressMap(count);

                RA++;

                pcb.ttc += 2;

                if (pcb.ttc > pcb.ttl)

                {

                    TI = 2;

                }

                MOS();

            }

            else if (IR[0] == 'P' && IR[1] == 'D')

            {

                SI = 2;

                pcb.ttc++;

                if (IR[2] <= 57 && IR[3] <= 57) // to check oprand error

                {

                    int temp = (IR[2] - '0') \* 10 + (IR[3] - '0');

                    if (M[temp][0] == '\*')

                    {

                        PI = 3;

                    }

                    else

                    {

                        RA++;

                    }

                }

                else

                {

                    PI = 2;

                }

                if (pcb.ttc > pcb.ttl)

                {

                    TI = 2;

                }

                MOS();

            }

            else if (IR[0] == 'H')

            {

                SI = 3;

                MOS();

            }

            else

            {

                // operation code error

                PI = 1;

                MOS();

            }

        } while (IR[0] != 'H');

    }

    void startExecution()

    {

        IC = 0;

        Execute();

    }

    void MOS()

    {

        if (PI == 1 && TI == 0)

        {

            Terminate(4);

        }

        else if (PI == 2 && TI == 0)

        {

            Terminate(5);

        }

        else if (PI == 3 && TI == 0)

        {

            Terminate(6);

            // More code to be added

        }

        else if (PI == 1 && TI == 2)

        {

            Terminate(3);

            Terminate(4);

        }

        else if (PI == 2 && TI == 2)

        {

            Terminate(3);

            Terminate(5);

        }

        else if (PI == 3 && TI == 2)

        {

            Terminate(3);

        }

        else if (SI == 1 && TI == 0)

        {

            Read();

            // if (buffer[0] == '$' && buffer[1] == 'E' && buffer[2] == 'N' && buffer[3] == 'D')

            // {

            //     cout << "Out of data error" << endl;

            //     Terminate(1);

            // }

        }

        else if (SI == 2 && TI == 0)

        {

            Write();

        }

        else if (SI == 3 && TI == 0)

        {

            Terminate(0);

        }

        else if (SI == 1 && TI == 2)

        {

            Terminate(3);

        }

        else if (SI == 2 && TI == 2)

        {

            Write();

            Terminate(3);

        }

        else if (SI == 3 && TI == 2)

        {

            Terminate(0);

        }

    }

    //Get Frame for page table randomly

    void allocate()

    {

        PTR = random();

        for (int i = PTR; i < PTR + 10; i++)

            for (int j = 0; j < 4; j++)

                M[i][j] = '#';

        //Allocate frame Frame number for first line

        temp\_loc = random() / 10; // 11

        M[PTR][2] = temp\_loc / 10 + '0';

        M[PTR][3] = temp\_loc % 10 + '0';

    }

    void pcb\_init()

    {

        pcb.job\_id = (buffer[4] - '0') \* 1000 + (buffer[5] - '0') \* 100 + (buffer[6] - '0') \* 10 + (buffer[7] - '0');

        pcb.llc = 0;

        pcb.ttc = 0;

        pcb.ttl = (buffer[8] - '0') \* 1000 + (buffer[9] - '0') \* 100 + (buffer[10] - '0') \* 10 + (buffer[11] - '0');

        pcb.tll = (buffer[12] - '0') \* 1000 + (buffer[13] - '0') \* 100 + (buffer[14] - '0') \* 10 + (buffer[15] - '0');

    }

    void load()

    {

        int block = 0, a = 0, i, j;

        while (!ip\_file.eof())

        {

            read();

            index++;

            if (buffer[0] == '$' && buffer[1] == 'A' && buffer[2] == 'M' && buffer[3] == 'J')

            {

                init();

                pcb\_init();

                allocate();

            }

            else if (buffer[0] == '$' && buffer[1] == 'D' && buffer[2] == 'T' && buffer[3] == 'A')

            {

                startExecution();

            }

            else if (buffer[0] == '$' && buffer[1] == 'E' && buffer[2] == 'N' && buffer[3] == 'D')

            {

                continue;

            }

            //Program Card

            else

            {

                int a = 0;

                if (M[temp\_loc \* 10][0] != '\*')

                    temp\_loc++;

                for (int i = temp\_loc \* 10; i < (temp\_loc \* 10) + 10; i++)

                {

                    for (int j = 0; j < 4; j++)

                    {

                        M[i][j] = buffer[a++];

                    }

                }

            }

        }

    }

    void addressMap(int VA)

    {

        PTE = PTR + VA / 10;

        int n = (M[PTE][2] - '0') \* 10 + (M[PTE][3] - '0');

        RA = (n \* 10) + VA % 10; // 110

    }

};

int main()

{

    os\_phase2 obj;

    obj.ip\_file.open("input.txt");

    obj.op\_file.open("output.txt", ios::out | ios::trunc);

    obj.load();

    obj.ip\_file.close();

    obj.op\_file.close();

    return 0;

}

**INPUT**

**$AMJ040500200002**

**GD30PD30LR30SR32GD40PD40H**

**$DTA**

**SHE SELLS SEA SHELLS ON**

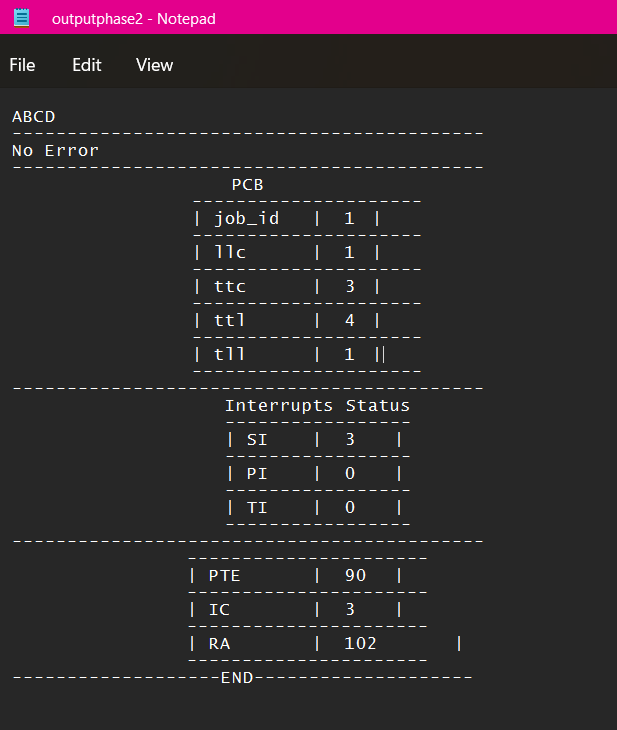
**SHORE**

**$END0405**

**Output:**







**PHASE 3**

#include <bits/stdc++.h>

#define CH1\_TS 5

#define CH2\_TS 5

#define CH3\_TS 2

using **namespace** std;

**struct** state

{

**char** IC[4], R[4], IR[4], error[100];

**bool** T;

**int** SI, TI, PI;

};

**int** Global\_Timer = 0;

**int** IOI = 1;

**class** PCB

{

**public:**

**int** Job\_id;

**int** TTL, TTC, LLC, TLL, Ptr;

**int** datacount, program\_count;

    vector<**int**> CodePtr, DataPtr, OutputPtr;

    state curr\_state;

**char** flag;

    PCB()

    {

    }

    PCB(string card)

    {

        cout << "in PCB\n"

             << card << "\n";

        Job\_id = atoi(card.substr(4, 4).c\_str());

        TTL = atoi(card.substr(8, 4).c\_str());

        TLL = atoi(card.substr(12, 4).c\_str());

        TTC = 0;

        LLC = 0;

        datacount = 0;

        program\_count = 0;

*// TLL = 0;*

        CodePtr.clear();

        DataPtr.clear();

        cout << "-------------------------------------------\n";

        cout << "                    PCB\n";

*// cout << "\n-----------------------------------------" << endl;*

*// cout << "\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;*

        cout << "\t---------------------\n";

        cout << "\t| job\_id\t|  " << Job\_id << "\t|" << endl;

        cout << "\t---------------------\n";

        cout << "\t| llc\t\t|  " << LLC << "\t|" << endl;

        cout << "\t---------------------\n";

        cout << "\t| ttc\t\t|  " << TTC << "\t|" << endl;

        cout << "\t---------------------\n";

        cout << "\t| ttl\t\t|  " << TTL << "\t|" << endl;

        cout << "\t---------------------\n";

        cout << "\t| tll\t\t|  " << TLL << "\t|" << endl;

        cout << "\t---------------------\n";

        cout << "\t| Data\_count\t|  " << datacount << "\t|" << endl;

        cout << "\t---------------------\n";

        cout << "\t| Program\_count\t|  " << program\_count << "\t|" << endl;

        cout << "\t---------------------\n";

        cout << "-------------------------------------------\n";

    }

};

PCB temp, \*ptr;

**class** SuperVisiorMem

{

**public:**

    queue<**int**> EmptyBuffers;

    queue<**int**> InputFullBuffers;

    queue<**int**> OutputFullBuffers;

**char** buffer[10][40];

    SuperVisiorMem()

    {

        buffer[0][0] = 0;

        for (**int** i = 0; i < 10; i++)

            EmptyBuffers.push(i);

    }

**int** allotEmptyBuffer()

    {

**int** t;

        if (!EmptyBuffers.empty())

        {

            t = EmptyBuffers.front();

            EmptyBuffers.pop();

            return t;

        }

        else

            return -1;

    }

**void** addInputBuffer(**int** buff\_no)

    {

        InputFullBuffers.push(buff\_no);

    }

**void** addOutputBuffer(**int** buff\_no)

    {

        OutputFullBuffers.push(buff\_no);

    }

**void** addEmptyBuffer(**int** buff\_no)

    {

        EmptyBuffers.push(buff\_no);

    }

**int** returnIFB()

    {

        if (!InputFullBuffers.empty())

        {

**int** t;

            t = InputFullBuffers.front();

            InputFullBuffers.pop();

            return t;

        }

        return -1;

    }

};

**class** DrumMem

{

**public:**

**char** drum[1000][41];

    queue<**int**> EmptyTracks;

    DrumMem()

    {

        drum[0][0] = 0;

        for (**int** i = 0; i < 100; i++)

            EmptyTracks.push(i);

    }

**int** allocateTrack()

    {

**int** t;

        t = EmptyTracks.front();

        EmptyTracks.pop();

        return t;

    }

**void** FreeTrack(**int** tno)

    {

        EmptyTracks.push(tno);

    }

};

**struct** CPU

{

**char** IR[4];

**char** R[4];

**bool** T;

**int** SI, PI, TI;

**char** IC[2];

**char** Mem[300][4];

    SuperVisiorMem sm;

    DrumMem dm;

**char** error[100];

    ifstream inputCard;

    deque<PCB \*> loadQ, readyQ, IOQ, terminateQ;

    vector<**int**> v;

    string TASK;

} c;

**int** no\_of\_pages = 0;

**void** clearbuffer(**char** buffer[][40], **int** n);

**void** loadInMain(PCB **\***pcb)

{

    cout << "\n---------------------------------\n";

    cout << "------------Load in Main----------\n";

    no\_of\_pages = 0;

**int** n;

**bool** code;

**char** temp[40], ch;

**int** b = 0;

**int** k = -1, i = 0, j, currP = 0, x = 0, l, uptr, m, currentPage;

**bool** new\_block = true, flag = false;

    cout << "Creating a new Block" << endl;

    pcb->Ptr = -1;

    code = !pcb->CodePtr.empty();

    while (!pcb->CodePtr.empty())

    {

        cout << "while (!pcb->CodePtr.empty())" << endl;

        for (**int** q = 0; q < 40; ++q)

            temp[q] = '\0';

        if (!pcb->CodePtr.empty())

        {

            cout << "CodePtr is not empty" << endl;

            cout << "Adding process to drum" << endl;

            strcpy(temp, c.dm.drum[pcb->CodePtr[0]]);

            pcb->CodePtr.erase(pcb->CodePtr.begin());

        }

        if (pcb->Ptr == -1)

        {

            cout << "If memory no allocated" << endl;

            cout << "Allocating memory" << endl;

            pcb->Ptr = rand() % 30;

            if (pcb->Ptr == -1)

            {

                exit(0);

            }

            uptr = pcb->Ptr \* 10;

            for (**int** n = 0; n < 10; n++)

                for (**int** o = 0; o < 4; o++)

                    c.Mem[uptr + n][o] = '#';

        }

        b = -1;

        k = -1;

        i = 0;

        while (b != 40)

        {

            b++;

            ch = temp[b];

            if (isprint(ch))

            {

                if (i % 4 == 0)

                {

                    k++;

                    if (k % 10 == 0)

                        flag = true;

                }

                if (flag)

                {

                    flag = false;

                    k = 0;

                    while (1)

                    {

                        currentPage = rand() % 30;

                        for (j = 0; j < c.v.size(); j++)

                        {

                            if (c.v[j] == currentPage)

                                break;

                        }

                        if (j >= c.v.size())

                            break;

                    }

                    no\_of\_pages++;

                    c.v.push\_back(currentPage);

                }

                c.Mem[pcb->Ptr \* 10 + no\_of\_pages - 1][0] = '0' + currentPage / 10;

                c.Mem[pcb->Ptr \* 10 + no\_of\_pages - 1][1] = '0' + currentPage % 10;

                c.Mem[currentPage \* 10 + k][i % 4] = ch;

                i++;

            }

            else

                break;

        }

    }

    cout << "\n---------------------------------\n";

}

**void** save\_state(PCB **\***pcb)

{

    strcpy(pcb->curr\_state.IC, c.IC);

    strcpy(pcb->curr\_state.IR, c.IR);

    strcpy(pcb->curr\_state.R, c.R);

    strcpy(pcb->curr\_state.error, c.error);

    pcb->curr\_state.T = c.T;

    pcb->curr\_state.SI = c.SI;

    pcb->curr\_state.PI = c.PI;

    pcb->curr\_state.TI = c.TI;

}

**void** recover(PCB **\***pcb)

{

    strcpy(c.IC, pcb->curr\_state.IC);

    strcpy(c.IR, pcb->curr\_state.IR);

    strcpy(c.R, pcb->curr\_state.R);

    strcpy(c.error, pcb->curr\_state.error);

    c.T = pcb->curr\_state.T;

    c.SI = pcb->curr\_state.SI;

    c.PI = pcb->curr\_state.PI;

    c.TI = pcb->curr\_state.TI;

}

**int** PD\_function(**int** block\_no, PCB **\***pcb)

{

**char** ch;

**int** t = c.dm.allocateTrack();

    pcb->OutputPtr.push\_back(t);

    pcb->LLC++;

    if (pcb->LLC > pcb->TLL)

    {

        return 2;

    }

    for (**int** i = 0; i < 10; i++)

        for (**int** j = 0; j < 4; j++)

        {

            ch = c.Mem[block\_no + i][j];

            c.dm.drum[t][i \* 10 + j] = ch;

        }

    return 0;

}

**int** GD\_function(**int** block\_no, PCB **\***pcb)

{

**int** k = 0;

**char** line[41];

    if (pcb->datacount >= pcb->DataPtr.size())

    {

        return 1;

    }

    strcpy(line, c.dm.drum[pcb->DataPtr[pcb->datacount]]);

    pcb->datacount++;

    for (**int** i = 0; i < 10; i++)

        for (**int** j = 0; j < 4; j++)

        {

            if (k >= strlen(line))

                break;

            c.Mem[block\_no + i][j] = line[k++];

        }

    return 0;

}

**void** printmsg2(string str)

{

    ofstream linePrinter;

**char** ch;

    linePrinter.open("LinePrinter.txt", ios::app);

    linePrinter.put('\n');

*// for (int i = 0; i < str.length(); i++)*

*//     linePrinter.put(str[i]);*

*// linePrinter << endl;*

    linePrinter.close();

    cout << str;

}

**void** printmsg(string str)

{

    ofstream linePrinter;

**char** ch;

    linePrinter.open("LinePrinter.txt", ios::app);

*// linePrinter << "Hello" << endl;*

    linePrinter << "NO ERROR:" << endl;

    for (**int** i = 0; i < str.length(); i++)

        linePrinter.put(str[i]);

    linePrinter << endl;

    linePrinter.close();

}

**int** AddressMap(**char** VA[], PCB **\***pcb)

{

**int** RA;

    if (isdigit(VA[0]) && isdigit(VA[1]))

    {

        if (c.Mem[pcb->Ptr \* 10 + (VA[0] - '0')][0] == '#')

        {

            c.PI = 3;

        }

        else

        {

            RA = (c.Mem[pcb->Ptr \* 10 + (VA[0] - '0')][0] - '0') \* 10 + (c.Mem[pcb->Ptr \* 10 + (VA[0] - '0')][1] - '0');

            RA = RA \* 10 + (VA[1] - '0');

        }

    }

    else

        c.PI = 2;

    return RA;

}

**void** Terminate(**int** EM)

{

    switch (EM)

    {

    case 5:

        printmsg2("Error:  NO ERROR");

        break;

    case 1:

        printmsg2("Error:  OUT OF DATA");

        break;

    case 2:

        printmsg2("Error:  LINE LIMIT EXCEEDED");

        break;

    case 3:

        printmsg2("Error:  TIME LIMIT EXCEEDED");

        break;

    case 4:

        printmsg2("Error:  OPERATION CODE ERROR");

        break;

    case 0:

        printmsg2("Error:  OPERAND ERROR");

        break;

    case 6:

        printmsg2("Error:  INVALID PAGE FAULT");

        break;

    case 7:

        printmsg2("Error:  TIME LIMIT EXCEEDED  And OPERATION CODE ERROR");

        break;

    case 8:

        printmsg2("Error:  TIME LIMIT EXCEEDED And OPERAND ERROR");

        break;

    }

*// pcb.TTL = pcb.TLL = pcb.TTC = pcb.LLC = 0;*

}

**int** master\_mode(PCB **\***cpcb)

{

**int** block\_no;

**int** EM;

**char** op[2];

    op[0] = c.IR[2];

    op[1] = '0';

    block\_no = AddressMap(op, cpcb);

    cout << "block no:" << block\_no << endl;

    if (c.TI == 0 && c.SI == 1)

    {

        EM = GD\_function(block\_no, cpcb);

        c.SI = 0;

        if (EM >= 0)

        {

            Terminate(EM);

            return -1;

        }

    }

    else if (c.TI == 0 && c.SI == 2)

    {

        EM = PD\_function(block\_no, cpcb);

        c.SI = 0;

        if (EM >= 0)

        {

            Terminate(EM);

            return -1;

        }

    }

    else if (c.TI == 0 && c.SI == 3)

    {

        c.SI = 0;

        Terminate(0);

        return -1;

    }

    else if (c.TI == 2 && c.SI == 1 && c.PI == 0)

    {

        Terminate(3);

        return -1;

    }

    else if (c.TI == 2 && c.SI == 2)

    {

        c.SI = 0;

        PD\_function(block\_no, cpcb);

        Terminate(3);

        return -1;

    }

    else if (c.TI == 2 && c.SI == 3)

    {

        Terminate(0);

        return -1;

    }

    else if (c.TI == 0 && c.PI == 1)

    {

        Terminate(4);

        return -1;

    }

    else if (c.TI == 0 && c.PI == 2)

    {

        Terminate(5);

        return -1;

    }

    else if (c.TI == 0 && c.PI == 3)

    {

**int** currentPage, t;

        if (c.IR[0] == 'G' && c.IR[1] == 'D' || (c.IR[0] == 'S' && c.IR[1] == 'R'))

        {

            while (1)

            {

                currentPage = rand() % 30;

                for (t = 0; t < c.v.size(); t++)

                {

                    if (c.v[t] == currentPage)

                        break;

                }

                if (t >= c.v.size())

                    break;

            }

            c.v.push\_back(currentPage);

            c.Mem[cpcb->Ptr \* 10 + (c.IR[2] - '0')][0] = '0' + currentPage / 10;

            c.Mem[cpcb->Ptr \* 10 + (c.IR[2] - '0')][1] = '0' + currentPage % 10;

            no\_of\_pages++;

            c.PI = 0;

            if (c.IC[1] == '0')

            {

                c.IC[1] = '9';

                c.IC[0]--;

            }

            else

                c.IC[1]--;

            cpcb->TTC--;

            cpcb->TTC++;

            if (cpcb->TTC > cpcb->TTL)

            {

                c.TI = 2;

            }

        }

        else

        {

            Terminate(6);

            return -1;

        }

    }

    return 0;

    cout << "mastermode\_here\n";

}

**void** IOExecute()

{

    cout << "\n---------------Executing I/O----------------\n";

**int** a1, a2;

**int** block\_no, EM, x;

    if (!c.IOQ.empty())

    {

        PCB \*pcb = c.IOQ.front();

        c.IOQ.pop\_front();

        cout << "Saving State of PCB" << endl;

        save\_state(pcb);

        if (pcb->curr\_state.TI == 0 && pcb->curr\_state.SI == 1)

        {

            EM = GD\_function(block\_no, pcb);

            cout << "Performing GD operation" << endl;

            pcb->curr\_state.SI = 0;

            if (EM >= 0)

            {

                Terminate(EM);

                x = -1;

            }

        }

        else if (pcb->curr\_state.TI == 0 && pcb->curr\_state.SI == 2)

        {

            EM = PD\_function(block\_no, pcb);

            cout << "Performing PD operation" << endl;

            pcb->curr\_state.SI = 0;

            if (EM >= 0)

            {

                Terminate(EM);

                x = -1;

            }

        }

        save\_state(pcb);

        if (x == -1)

        {

            cout << "Adding PCB to Terminate Queue" << endl;

            c.terminateQ.push\_back(pcb);

        }

        else

        {

            cout << "I/O operation executed Successfully\nSending PCB to Ready Queue" << endl;

            c.readyQ.push\_back(pcb);

        }

    }

}

**void** copy(**char** A[], **char** B[])

{

    for (**int** i = 0; i < 4; i++)

        A[i] = B[i];

}

**void** clearMem()

{

    for (**int** i = 0; i < 300; i++)

        for (**int** j = 0; j < 4; j++)

        {

            c.Mem[i][j] = '\0';

        }

    cout << "Clearing Memory and Queues" << endl;

    c.SI = 0;

    c.T = false;

    c.v.clear();

    c.loadQ.clear();

    c.readyQ.clear();

    c.SI = c.PI = c.TI = 0;

*// pcb.TTC = pcb.LLC = pcb.TLL = pcb.TTL = 0;*

    no\_of\_pages = 0;

}

**bool** compare(**char** A[], **char** B[])

{

**bool** result = true;

    for (**int** i = 0; i < 4; i++)

        if (A[i] != B[i])

        {

            result = false;

            break;

        }

    return result;

}

**int** execute(PCB **\***pcb)

{

    cout << "\n---------------Execution----------------\n";

**bool** endprogram = false;

**int** x = 0, a1, a2;

    c.IC[0] = c.IC[1] = '0';

    strcpy(c.error, "\0");

**char** op[2];

    cout << endl

         << "Page Table Pointer" << pcb->Ptr;

**int** RA = AddressMap(c.IC, pcb);

    cout << "mem2 " << c.Mem[RA][3];

    for (**int** i = 0; i < 4; ++i)

    {

        c.IR[i] = c.Mem[RA][i];

    }

    cout << "\nInstruction : ";

    for (**int** i = 0; i < 4; i++)

        cout << (**char**)c.IR[i];

    a1 = c.IR[2] - '0';

    a2 = c.IR[3] - '0';

    if (c.IC[1] == '9')

    {

        c.IC[0]++;

        c.IC[1] = '0';

    }

    else

        c.IC[1]++;

    cout << endl

         << "Real Address:"

         << " " << abs(RA) / 10 << endl;

*// if (c.IR[0] != 'H')*

*// {*

*//     op[0] = c.IR[2];*

*//     op[1] = c.IR[3];*

*//     x = AddressMap(op, pcb);*

*// }*

*// cout << "\nOperand:" << x;*

    if (c.PI == 0) *// if no page fault: START EXAMINE LOOP*

    {

        if (c.IR[0] == 'H')

        {

            c.SI = 3;

            c.TI = 0;

        }

        else if (c.IR[0] == 'G' && c.IR[1] == 'D')

        {

            c.SI = 1;

            c.TI = 0;

            return 1;

        }

        else if (c.IR[0] == 'P' && c.IR[1] == 'D')

        {

            c.SI = 2;

            c.TI = 0;

            return 1;

        }

        else if (c.IR[0] == 'L' && c.IR[1] == 'R')

        {

            c.SI = 0;

            copy(c.R, c.Mem[x]);

            cout << "\nRegister: ";

            for (**int** i = 0; i < 4; i++)

                cout << c.R[i];

            cout << endl;

        }

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

            copy(c.Mem[x], c.R);

        else if (c.IR[0] == 'C' && c.IR[1] == 'R')

        {

            for (**int** i = 0; i < 4; i++)

                cout << c.Mem[x][i];

            cout << endl;

            for (**int** i = 0; i < 4; i++)

                cout << c.R[i];

            cout << endl;

            c.T = compare(c.R, c.Mem[x]);

        }

        else if (c.IR[0] == 'B' && c.IR[1] == 'T')

        {

            cout << "next IC :" << x << endl;

            cout << "Toggle : " << c.T << endl;

            if (c.T)

            {

                c.IC[1] = x / 10 + '0';

                c.IC[0] = x % 10 + '0';

            }

        }

        else

            c.PI = 1;

        pcb->TTC++;

        if (pcb->TTC > pcb->TTL)

            c.TI = 2;

    }

    if (c.IR[0] != 'H')

    {

        op[0] = c.IR[2];

        op[1] = c.IR[3];

        x = AddressMap(op, pcb);

    }

    if (a1 > 10 || a2 > 10)

    {

        c.PI = 2;

    }

    if (c.SI != 0 || c.PI != 0 || c.TI != 0)

        if (master\_mode(pcb) == -1)

        {

            return -1;

        }

    if (strlen(c.error) > 1)

        return -1;

    cout << "\n---------------Execution ended------------------\n";

    cout << "\nOperand:" << x;

    return 0;

}

**void** channel1IR()

{

    cout << "\n-------------------------------------------\n";

    cout << "-----------------Channel 1-----------------" << endl;

*// for (int i = 0; i <= CH1\_TS; i+=5)*

*// {*

*// cout << "Channel 1 Timer = " <<  << endl;*

    if (c.inputCard.eof())

    {

        cout << "Reached End Of Line\nExiting Channel 1" << endl;

        return;

    }

**int** buff\_no = c.sm.allotEmptyBuffer();

    cout << "Buffer no = " << buff\_no << endl;

    clearbuffer(c.sm.buffer, buff\_no);

    if (buff\_no == -1)

    {

        cout << "Buffers ended\nExiting Channel 1" << endl;

        return;

    }

    cout << "Reading input card" << endl;

    c.inputCard.getline(c.sm.buffer[buff\_no], 41);

    cout << "Buffer value = " << c.sm.buffer[buff\_no] << endl;

    cout << "Adding Inputful Buffer" << endl;

    c.sm.addInputBuffer(buff\_no);

    IOI = 1;

    cout << "\n---------------Channel 1 ended ------------------\n";

    cout << "IOI = " << IOI << endl;

}

**void** clearbuffer(**char** buffer[][40], **int** n)

{

*// cout << "In Clear Buffer" << endl;*

    for (**int** i = 0; i < strlen(buffer[n]); ++i)

    {

        buffer[n][i] = '\0';

    }

}

**void** channel2IR()

{

    cout << "\n---------------------------------\n";

    cout << "--------------Channel 2-------------" << endl;

**int** temp;

*// if (c.sm.OutputFullBuffers.empty())*

*// {*

    cout << "OutputFullBuffer not empty" << endl;

    temp = c.sm.OutputFullBuffers.front();

    cout << "Printing Message" << endl;

    for (**int** i = 1; i < c.loadQ.front()->DataPtr.size(); i++)

    {

        printmsg(c.dm.drum[c.loadQ.front()->DataPtr[i]]);

        cout << c.dm.drum[c.loadQ.front()->DataPtr[i]] << endl;

    }

*// printmsg(string(c.sm.buffer));*

    cout << "Clearing OutputfullBuffer" << endl;

*// clearbuffer(c.sm.buffer, temp);*

*// c.sm.OutputFullBuffers.pop();*

*// c.sm.EmptyBuffers.push(temp);*

*// }*

    IOI = 2;

    cout << "\n----------------Channel 2 ended-----------------\n";

    cout << "IOI = " << IOI << endl;

}

**int** channel3IR()

{

**bool** code;

    PCB \*ptr = new PCB;

**int** n;

    IOI = 4;

**int** check = 0;

*// for (int i = 0; i < CH3\_TS; i++)*

*// {*

*// cout<<"here";*

        if (c.TASK.compare("IS") == 0)

        {

            cout << " if(c.TASK.compare(IS) == 0)" << endl;

            cout << "c.TASK = " << c.TASK << endl;

            if (!c.sm.InputFullBuffers.empty())

            {

                cout << "InputFullBuffers not empty" << endl;

                string card;

**int** k;

**char** buffer[41];

**int** bno = c.sm.returnIFB();

                cout << "Buffer no = " << bno << endl;

                for (k = 0; c.sm.buffer[bno][k] && k < 40; k++)

                {

                    buffer[k] = c.sm.buffer[bno][k];

                    cout << "buffer[k] = " << buffer[k] << endl;

                }

                buffer[k] = '\0';

                card = string(buffer);

                cout << "Card = " << card << endl;

                cout << "Adding empty buffer" << endl;

                c.sm.addEmptyBuffer(bno);

                if (card.find("$AMJ") != -1)

                {

                    cout << "Found $AMJ" << endl;

                    cout << "Initialized PCB" << endl;

                    temp = PCB(card);

                    IOI = 1;

*// Global\_Timer = 0;*

                    ptr = &temp;

                    code = true;

                    c.loadQ.push\_back(ptr);

                }

                else if (card.find("$DTA") != -1)

                {

                    c.loadQ.front()->flag = 'D';

                    cout << "Found $DTA" << endl;

                    code = false;

                }

                else if (card.find("$END") != -1)

                {

                    cout << "Found $END" << endl;

                    ptr = &temp;

*// c.TASK = string("IS");*

                    cout << "Adding Process to Load Queue" << endl;

*// IOI += 4;*

*// cout << "IOI = " << IOI << endl;*

                    cout << "Adding Process to Ready Queue" << endl;

                    loadInMain(ptr);

                    c.readyQ.push\_back(ptr);

                    cout << "Executing the process" << execute(ptr) << endl;

*// return 1;*

                    check = 1;

                }

                else

                {

                    cout << "\n--------------------------------------\n";

                    cout << "-------------- Channel 3 --------------" << endl;

                    PCB \*pcb = c.loadQ.front();

*// c.loadQ.pop\_front();*

                    if (card.find("GD") != -1)

                    {

                        pcb->flag = 'P';

                        pcb->program\_count++;

                        cout << "Program Count = " << pcb->program\_count << endl;

                        cout << "Flag = " << pcb->flag << endl;

                        code = true;

                    }

                    else

                    {

                        pcb->datacount++;

                        cout << "Data count = " << pcb->datacount << endl;

                    }

**int** t = c.dm.allocateTrack();

                    Global\_Timer++;

                    cout << "Global Timer = " << Global\_Timer << endl;

                    cout << "Channel 3 Timer = " << 1 << endl;

                    strcpy(c.dm.drum[t], card.c\_str());

                    if (code)

                        pcb->CodePtr.push\_back(t);

                    else

                        pcb->DataPtr.push\_back(t);

*// c.loadQ.push\_front(pcb);*

                    Global\_Timer++;

                    cout << "Global Timer = " << Global\_Timer << endl;

                    cout << "Channel 3 Timer = " << 2 << endl;

*// IOI -= 4;*

                    cout << "\n-----------------Channel 3 ended---------------------\n";

                    cout << "IOI = " << IOI << endl;

                }

            }

        }

        if (!c.IOQ.empty())

        {

            IOExecute();

        }

        if (!c.readyQ.empty())

        {

            ptr = c.readyQ.front();

            n = execute(ptr);

            save\_state(ptr);

            c.readyQ.pop\_front();

            if (n == 0)

                c.readyQ.push\_back(ptr);

            if (n == 1)

                c.IOQ.push\_back(ptr);

            if (n == -1)

                c.terminateQ.push\_back(ptr);

        }

    return check;

}

**void** simulate()

{

**int** ch1\_t, ch2\_t, ch3\_t;

**int** flag = 1;

    ofstream o;

    o.open("Mem.txt", ios::out);

    for (**int** i = 0; i < 10; ++i) *// Because we have 10 buffers in supervisory storage*

    {

        clearbuffer(c.sm.buffer, i);

    }

    c.inputCard.open("ip.txt");

    while (flag)

    {

        ch1\_t = ch2\_t = ch3\_t = 0;

        for (**int** j = 0; j < 5; j++)

        {

            Global\_Timer++;

            cout << "Global Timer = " << Global\_Timer << endl;

            ch1\_t++;

            cout << "Channel 1 Timer = " << ch1\_t << endl;

            if (Global\_Timer % 5 == 0)

            {

*// cout << "IOI = " << IOI << endl;*

                IOI++;

                channel1IR();

            }

        }

*// Global\_Timer++;*

*// cout << "Global Timer = " << Global\_Timer << endl;*

        c.TASK = string("IS");

        IOI += 4;

*// cout << "IOI = " << IOI << endl;*

**int** check = channel3IR();

*// Global\_Timer++;*

*// cout << "Global Timer = " << Global\_Timer << endl;*

*// channel2IR();*

        if (c.inputCard.eof())

        {

            cout << "Reached End Of Line" << endl;

            flag = 0;

        }

        if (check)

        {

            cout << "size:" << c.loadQ.front()->DataPtr.size() << endl;

            for (**int** j = 0; j < 5; j++)

            {

                Global\_Timer++;

                cout << "Global Timer = " << Global\_Timer << endl;

                ch2\_t++;

                cout << "Channel 2 Timer = " << ch2\_t << endl;

            }

            IOI += 2;

*// cout << "IOI = " << IOI << endl;*

            channel2IR();

*// cout << "For loop ended" << endl;*

        }

    }

    for (**int** i = 0; i < 300; i++)

    {

        for (**int** j = 0; j < 4; j++)

            o << c.Mem[i][j] << "";

        o << endl;

    }

}

**int** main()

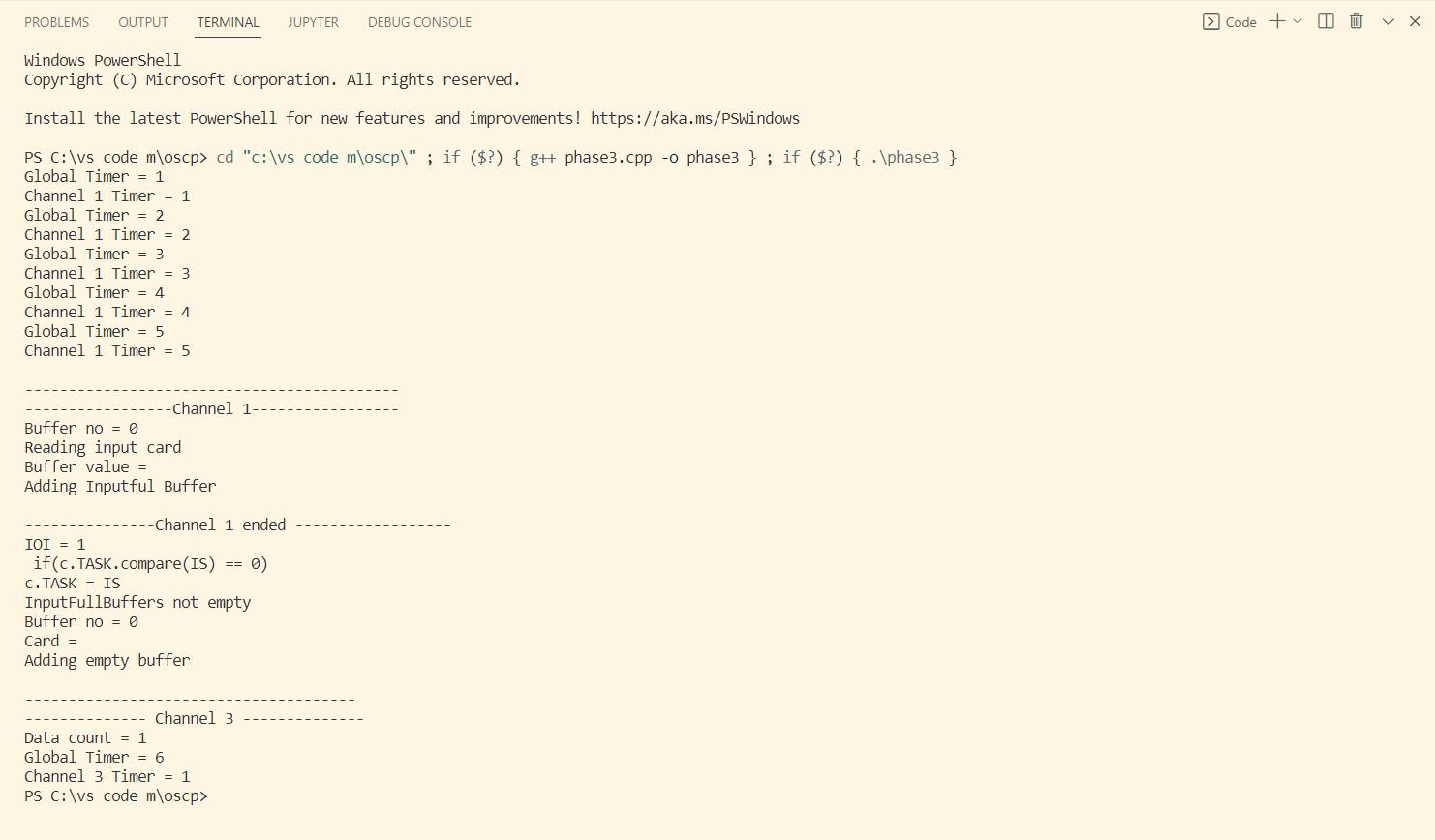
{

    simulate();

    return 0;

}

**OUTPUT:**

****