#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <vector>
#include <fstream.h>
#include "pmpclass.h"
using namespace std;

/vector<process*> Delete(vector<process*> DeleteQ, int ID);
/vector<process*> DeleteW(vector<process*> DeleteWQ, int IDW);

int main(int argc, char* argv[])
{
    if (argc != 3)
    {
        cout << "You schmuck! This program was hard enough to write, you can at least call it correctly!\n";
        exit(1);
    }

    vector<process*> ReadyQ;
    vector<process*> WaitQ;

vector<process*> DeleteQ;
ifstream in;
char option;
int n,b,p,temp,waitID;
char *q = argv[2];
int quantum = atoi(q);
process * tempProcess = NULL;
process * tempP = NULL;
in.open(argv[1]);

while(1)
{
    in >> option;
    if(in.eof())
        exit(1);
    switch(option)
    {
    case 'C' : case 'c' :
    {
        in >> n;
        in >> b;

        if(ReadyQ.empty())
            p = 0;
        else
            p = ReadyQ[0]->id;

        cout << option << " " << n << " " << b << " " << p << endl;
        ReadyQ.push_back(new process(n,b,p,quantum));

        tempProcess = ReadyQ.back();
        cout << "PID " << tempProcess->id << " " << tempProcess->burst << " put on Ready Queue" << endl;

        if(ReadyQ.size() == 0)
            cout << "Process 0 running" << endl;
        else
            
            cout << "PID " << ReadyQ[0]->id << " " << ReadyQ[0]->burst << " running with " << ReadyQ[0]->activeTime << " left" << endl;
            temp = ReadyQ[0]->burst;
            temp--;
            ReadyQ[0]->burst = temp;
            temp = ReadyQ[0]->activeTime;
            temp--;
case 'D' : case 'd' :
{
    in >> n;
    cout << option << " " << n << endl;

    if(ReadyQ.size() != 0)
    {
        ReadyQ = Delete(ReadyQ,n);
        WaitQ = DeleteW(WaitQ,n);
    }

    if(ReadyQ.size() == 0)
    cout << "Process 0 running" << endl;
    else
    {
        cout << "PID " << ReadyQ[0]->id << " " << ReadyQ[0]->burst << " running with " << ReadyQ[0]->activeTime << " left" << endl;
        temp = ReadyQ[0]->burst;
        temp--;
        ReadyQ[0]->burst = temp;
        temp = ReadyQ[0]->activeTime;
        temp--;
        ReadyQ[0]->activeTime = temp;
    }
}
break;
}

case 'I' : case 'i' :
{
    cout << option << endl;

    if(ReadyQ.size() == 0)
    cout << "Process 0 running" << endl;
    else
    {
        cout << "PID " << ReadyQ[0]->id << " " << ReadyQ[0]->burst << " running with " << ReadyQ[0]->activeTime << " left" << endl;
        temp = ReadyQ[0]->burst;
        temp--;
        ReadyQ[0]->burst = temp;
        temp = ReadyQ[0]->activeTime;
        temp--;
        ReadyQ[0]->activeTime = temp;
    }
}
break;
}
```
    ReadyQ[0]->burst = temp;
    temp = ReadyQ[0]->activeTime;
    temp--;
    ReadyQ[0]->activeTime = temp;
}
break;
}

    case 'W' : case 'w' :
    {
    in >> n;
    cout << option << " " << n << endl;

    cout << "PID " << ReadyQ[0]->id << " " << ReadyQ[0]->burst << " placed on wait queue" << endl;
    ReadyQ[0]->waitNumber = n;
    ReadyQ[0]->activeTime = quantum;

    tempP = ReadyQ[0];
    ReadyQ.erase(ReadyQ.begin());
    WaitQ.push_back(tempP);

    if(ReadyQ.size() == 0)
        cout << "Process 0 running" << endl;
    else
    {
        cout << "PID " << ReadyQ[0]->id << " " << ReadyQ[0]->burst << " running with " << ReadyQ[0]->activeTime << " left" << endl;
        temp = ReadyQ[0]->burst;
        temp--;
        ReadyQ[0]->burst = temp;
        temp = ReadyQ[0]->activeTime;
        temp--;
        ReadyQ[0]->activeTime = temp;
    }
break;
}

    case 'E' : case 'e' :
    {
    in >> n;
    waitID = n;
    cout << option << " " << n << endl;
```
if(ReadyQ.size() == 0)
    cout << "Process 0 running" << endl;
else
{
    cout << "PID " << ReadyQ[0]->id << " " << ReadyQ[0]->burst << " running with " <<
    ReadyQ[0]->activeTime << " left" << endl;
    temp = ReadyQ[0]->burst;
    temp--;
    ReadyQ[0]->burst = temp;
    temp = ReadyQ[0]->activeTime;
    temp--;
    ReadyQ[0]->activeTime = temp;
}
break;
}

case 'X' : case 'x' :
{
    cout << option << endl << endl;
    cout << "Current state of simulation: " << endl;
    if(ReadyQ.size() == 0)
        cout << "No processes currently running" << endl;
    else
        cout << "Current Process: PID " << ReadyQ[0]->id << " " << ReadyQ[0]->burst << endl;

    if(ReadyQ.size() == 1)
        cout << "Ready Queue: empty" << endl;
    else
    {
        cout << "Contents of Ready Queue: " << endl;
        for(int m = 1; m < ReadyQ.size(); m++)
        {
            cout << "   PID " << ReadyQ[m]->id << " " << ReadyQ[m]->burst << endl;
        }
    }

    if(WaitQ.size() == 0)
        cout << "Wait Queue: empty" << endl;
    else
    {
        cout << "Contents of Wait Queue: " << endl;
        for(int n = 0; n < WaitQ.size(); n++)
        {
        }
cout << "PID " << WaitQ[n]->id << " " << WaitQ[n]->burst << " waiting on EID " << WaitQ[n]->waitNumber << endl;
}
}

exit(1);
}

if(WaitQ.size() != 0)
{
    for(int i = 0; i < WaitQ.size(); i++)
    {
        if(WaitQ[i]->waitNumber == waitID)
        {
            tempP = WaitQ[i];
            WaitQ.erase(WaitQ.begin() + i);
            cout << "PID " << tempP->id << " " << tempP->burst << " put on Ready Queue" << endl;
            ReadyQ.push_back(tempP);
        }
    }
}

if(ReadyQ[0]->activeTime == 0)
{
    ReadyQ[0]->activeTime = quantum;
    cout << "PID " << ReadyQ[0]->id << " " << ReadyQ[0]->burst << " put on Ready Queue" << endl;
    tempP = ReadyQ[0];
    ReadyQ.erase(ReadyQ.begin());
    ReadyQ.push_back(tempP);
}

if (ReadyQ.size() != 0)
{
    if(ReadyQ[0]->burst == 0)
    {
        n = ReadyQ[0]->id;
        ReadyQ = Delete(ReadyQ,n);
        ReadyQ = DeleteW(ReadyQ,n);
    }
}

cout << endl;
in.close();
return 0;
}

/********************
** Functions **
*******************/

vector<process*> Delete(vector<process*> DeleteQ, int ID)
{
    int k;
    if(DeleteQ.size() != 0)
    {
        for(int j = 0; j < DeleteQ.size(); j++)
        {
            k = DeleteQ[j]->parentNumber;
            if(DeleteQ[j]->parentNumber == ID)
            {
                cout << "PID " << DeleteQ[j]->id << " deleted!" << endl;
                DeleteQ.erase(DeleteQ.begin() + j);
                DeleteQ = Delete(DeleteQ, k);
                return DeleteQ;
            }
            if(DeleteQ[j]->id == ID)
            {
                cout << "PID " << DeleteQ[j]->id << " deleted!" << endl;
                DeleteQ.erase(DeleteQ.begin() + j);
                return DeleteQ;
            }
        }
    }
    return DeleteQ;
}

vector<process*> DeleteW(vector<process*> DeleteWQ, int IDW)
{
    int k;
    if(DeleteWQ.size() != 0)
    {
        for(int j = 0; j < DeleteWQ.size(); j++)
        {
            k = DeleteWQ[j]->parentNumber;
            if(DeleteWQ[j]->parentNumber == IDW)
            {
cout << "PID " << DeleteWQ[j]->id << " deleted!" << endl;
DeleteWQ.erase(DeleteWQ.begin() + j);
DeleteWQ = DeleteWQ, k);
return DeleteWQ;
}
if(DeleteWQ[j]->id == IDW)
{
    cout << "PID " << DeleteWQ[j]->id << " deleted!" << endl;
    DeleteWQ.erase(DeleteWQ.begin() + j);
    return DeleteWQ;
}
return DeleteWQ;
}