TSN2101 OPERATING SYSTEMS

1. Lecturer: Dr Ng Hu & Chan Wai Kok
2. Class Section: TT1V
3. Topic : CODING ASSIGNMENT REPORT
4. Group members :

No	Student Name	Student ID
1	Muhammad Nabil Anwar Bin Md Zaid	1181201475
2	Muhamad Danial bin Arrifin	1181201041
3	Affiq Bin Mohamed Zulkifli	1181201507

Table of Content

Coding Segment

Process Class

```
#include <iostream>


class Process
{
    private:
        int name;
        int burst;
        int ori_burst;
        int arrival;
        int priority = 0;
        int completion = 0;
        int waiting;
        int turnaround;
    public:
        int getBurst()
        {
            return burst;
        }
        int getOriginalBurst()
        {
            return ori_burst;
        }
        int getArrival()
        {
```

```
            return arrival;
        }
        int getPriority()
        {
            return priority;
        }
        int getName()
        {
            return name;
        }
        int getCompletion()
        {
            return completion;
        }
        int getTurnaround()
        {
            turnaround = completion - arrival;
            return turnaround;
        }
        int getWaiting()
        {
            waiting = turnaround - burst;
            if(waiting <= 0)
                waiting = 0;
            return waiting;
        }
        void setArrival(int arrival)
        {
            this->arrival = arrival;
```

```
        }
        void setBurst(int burst)
        {
            this->burst = burst;
        }
        void setOriginalBurst(int burst)
        {
            this->ori_burst = burst;
        }
        void setPriority(int priority)
        {
            this->priority = priority;
        }
        void setCompletion(int completion)
        {
            this->completion = completion;
        }
        void setWaiting(int waiting)
        {
            this->waiting = waiting;
        }
        void setName(int name)
        {
            this->name = name;
        }
};
```

Printing Table and Gantt Chart Function

```
#include <iostream>
#include <iomanip>
#include <cmath>
#include <windows.h>

using namespace std;

const char HORZ = 196;
const char VERT = 179;
const char TL   = 218;
const char TM   = 194;
const char TR   = 191;
const char BL   = 192;
const char BM   = 193;
const char BR   = 217;
const char VH   = 197;
const char VR   = 195;
const char VL   = 180;

void printGanttChart(int count, int order[], int clock[])
{
    cout<<endl<<endl<<"Gantt Chart: "<<endl;
    cout<<TL;
    for(int i = 0; i < count; i++)
    {
        for(int x = 0; x < 6; x++)
            cout<<HORZ;
        if(i+1 != count)
```

```
            cout<<TM;
    }
    cout<<TR<<endl;
    for(int i = 0; i < count; i++)
    {
        cout<<VERT;
        if(order[i] != -1)
            cout<<"  P"<<order[i]<<"  ";
        else
            cout<<"  XX  ";
    }
    cout<<VERT<<endl<<BL;
    for(int i = 0; i < count; i++)
    {
        for(int x = 0; x < 6; x++)
            cout<<HORZ;
        if(i+1 != count)
            cout<<BM;
    }
    cout<<BR;
    cout<<endl<<"0";
    for(int i = 0; i < count; i++)
            cout<<setw(7)<<clock[i];
}

void printTableHorz(char P1, char P2, char P3)
{
    int length[8] = {9, 14, 12, 10, 16, 17, 14};
```

```
    cout<<endl<<P1;
    for(int i = 0; i < 7; i++)
    {
        for(int x = 0; x < length[i]; x++)
            cout<<HORZ;
        if(i != 6)
            cout<<P2;
        else
            cout<<P3;
    }
}

void printTable( float avg[], float sum[], int size, Process P[])
{
    cout<<endl<<"Table: ";
    printTableHorz(TL, TM, TR);
    cout<<endl<<VERT<<"         "<<VERT<<" Arrival Time "<<VERT<<" Burst
Time "<<VERT<<" Priority "<<VERT<<" Completion Time"<<VERT<<" Turnaround
Time "<<VERT<<" Waiting Time "<<VERT;
    for(int i = 0; i < size; i++)
    {
        printTableHorz(VR, VH, VL);
        cout<<left;
        cout<<endl<<VERT<<" P"<<setw(7)<<P[i].getName()<<VERT<<"
"<<setw(13)<<P[i].getArrival()<<VERT<<"
"<<setw(11)<<P[i].getBurst()<<VERT<<"
"<<setw(9)<<P[i].getPriority()<<VERT<<"
"<<setw(15)<<P[i].getCompletion()<<VERT<<"
```

```
"<<setw(16)<<P[i].getTurnaround()<<VERT<<"
"<<setw(13)<<P[i].getWaiting()<<VERT;
    }
    printTableHorz(VR, VH, VL);
    cout<<endl<<VERT<<" Total
"<<VERT<<setw(15)<<right<<VERT<<setw(13)<<VERT<<setw(11)<<VERT<<setw(17)<<
VERT<<" "<<setw(16)<<left<<sum[0]<<VERT<<" "<<setw(13)<<sum[1]<<VERT;
    printTableHorz(VR, VH, VL);
    cout<<endl<<VERT<<" Average
"<<VERT<<setw(15)<<right<<VERT<<setw(13)<<VERT<<setw(11)<<VERT<<setw(17)<<
VERT<<" "<<setw(16)<<left<<avg[0]<<VERT<<"
"<<setw(13)<<avg[1]<<right<<VERT;
    printTableHorz(BL, BM, BR);
}
```

Main Driver Function

```cpp
#include    <iostream>
#include    <algorithm>
#include    <iterator>
#include    <iomanip>
#include    <queue>
#include    "Process.cpp"
#include    "table.cpp"

using namespace std;

int getNoOfProcesses()
{
    int size;
    int completion = 0;

    cout << "How many processes would you like: \n";
    do
    {
        cin >> size;
        if (size < 3 || size > 10)
        {
            cout << "Invalid size.\n";
            cout << "Please enter a new value.\n";
        }
    } while (size < 3 || size > 10);
    return size;
}
```

```
void swap(Process *xp, Process *yp)
{
    Process temp = *xp;
    *xp = *yp;
    *yp = temp;
}



void bubbleSortArrival(Process arr[], int n,int ea)
{
    int i, j;
    for (i = ea; i < n - 1; i++)

        for (j = ea; j < n - i - 1; j++)
            if (arr[j].getArrival() > arr[j + 1].getArrival())
                swap(&arr[j], &arr[j + 1]);
}

void bubbleSortPriority(Process arr[], int n, int completion, int ea)
{
    int i, j;
    for (i = ea; i < n - 1; i++)

        for (j = ea; j < n - i - 1; j++)
        {
            if((arr[j].getArrival() <= completion) && (arr[j +
1].getArrival() <= completion))
                if (arr[j].getPriority() > arr[j + 1].getPriority())
                    swap(&arr[j], &arr[j + 1]);
```

```
        }
}

void bubbleSortBurst(Process arr[], int n, int completion, int ea)
{
    int i, j;
    for (i = ea; i < n - 1; i++)

        for (j = ea; j < n - i - 1; j++)
        {
            if((arr[j].getArrival() <= completion) && (arr[j +
1].getArrival() <= completion))
                if (arr[j].getBurst() > arr[j + 1].getBurst())
                    swap(&arr[j], &arr[j + 1]);
        }
}

int main()
{
    int size;
    int arrival;
    int priority;
    int burst;
    int completion = 0;
    int algoType;
    int quantum;
    int ganttSize;
    float totalTurnaround = 0;
    float totalWaiting = 0;
```

```
    float avgTurnaround = 0;
    float avgWaiting = 0;

    cout<<"What type of algorithm would you like to use?"<<endl<<endl;
    cout<<"1.Non-Preemptive SJF"<<endl;
    cout<<"2.Non-Preemptive Priority"<<endl;
    cout<<"3.Round Robin"<<endl;
    cin>>algoType;
    size = getNoOfProcesses();
    ganttSize = size;
    Process P[size];
    queue<Process> PQ;

        cout << "Define the details of each processes" << endl;
        for (int i = 0; i < size; i++)
        {
            cout << "Arrival Time of Process " << i + 1 << " :";
            cin >> arrival;
            P[i].setArrival(arrival);
            cout << "Burst Time of Process " << i + 1 << " :";
            cin >> burst;
            if(algoType == 3)
                P[i].setOriginalBurst(burst);
            P[i].setBurst(burst);
            if(algoType == 2)
            {
                cout << "Priority of Process " << i + 1 << " :";
                cin >> priority;
                P[i].setPriority(priority);
```

```
            }
            P[i].setName(i);
        }
        if(algoType == 3)
        {
            cout<< "Enter Quantum"<<endl;
            cin>>quantum;
        }
        completion = 0;
        if(algoType != 3)
        {
            for(size_t i = 0; i < size; i++)
            {
                bubbleSortArrival(P, size,i);
                if (algoType == 1)
                    bubbleSortBurst(P,size,completion,i);
                if(algoType == 2)
                    bubbleSortPriority(P,size,completion,i);
                completion += P[i].getBurst();
                P[i].setCompletion(completion);
                if(!(P[i + 1].getArrival() <= completion))
                    break;
            }
        }
        queue<Process> PV;
        if(algoType == 3)
        {
            Process temp_array[size];
            bubbleSortArrival(P,size,0);
```

```
            completion = 0;
            int j = 0;

            for(int i = j; i < size ;i++)
            {
                if(P[i].getArrival() <= completion)
                {
                    PQ.push(P[i]);
                    j++;
                }
            }
            do
            {
                Process front = PQ.front();
                if(front.getBurst() > quantum)
                {
                    front.setBurst(front.getBurst() - quantum);
                    completion += quantum;

                    front.setCompletion(completion);
                }
                else
                {
                    completion += front.getBurst();
                    front.setBurst(front.getBurst() - front.getBurst());
                    front.setCompletion(completion);
                    cout<<front.getName()<<": "<<front.getCompletion()<< "
> "
                        <<completion<<endl;
```

```
                    temp_array[front.getName()] = front;
                }
                for(int i = j; i < size ;i++)
                {
                    if(P[i].getArrival() <= completion)
                    {
                        PQ.push(P[i]);
                        j++;
                    }
                }
                if(front.getBurst() != 0)
                    PQ.push(front);

                PV.push(front);
                PQ.pop();
            }while(!PQ.empty());
            for(int i = 0; i < size;i++)
            {
                P[i] = temp_array[i];
                P[i].setBurst(P[i].getOriginalBurst());
            }
            ganttSize = PV.size();
        }
    for(int i = 0;i < size;i++)
    {
        totalTurnaround += P[i].getTurnaround();
        totalWaiting += P[i].getWaiting();
    }
    avgTurnaround = totalTurnaround / size;
```

```
    avgWaiting = totalWaiting / size;
    cout << endl;
    cout << "No. of Process :" << size << endl;
    cout << "Total Turnaround Time :" << totalTurnaround << endl;
    cout << "Average Turnaround Time :" << avgTurnaround << endl;
    cout << "Total Waiting Time :" << totalWaiting << endl;
    cout << "Average Waiting Time :" << avgWaiting << endl;
    int name[ganttSize];
    int time[ganttSize];
    if(algoType != 3)
    {
        for (size_t i = 0; i < size; i++)
        {
            name[i] = P[i].getName();
            time[i] = P[i].getCompletion();
        }
    }
    else
    {
        int i = 0;
        while(!(PV.empty()))
        {
            Process front = PV.front();
            name[i] = front.getName();
            time[i] = front.getCompletion();
            PV.pop();

            i++;
        }
```

```
    }
    float avg[2] = {avgTurnaround,avgWaiting};
    float total[2] = {totalTurnaround,totalWaiting};
        printTable(avg,total,size,P);
        printGanttChart(ganttSize,name,time);

}
```