ใบงานที่ 6

เรื่อง CPU Scheduling

Priority

เสนอ

อาจารย์ปิยพล ยืนยงสถาวร

จัดทำโดย

นาย กวีวัธน์ กาญจน์สุพัฒนากุล 65543206003-7

ใบงานนี้เป็นส่วนหนึ่งของรายวิชา ระบบปฏิบัติการ

หลักสูตรวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา

ประจำภาคที่ 2 ปีการศึกษา 2566

ขั้นตอนการทดลอง

1. *ออกแบบโปรแกรมด้วยผังงาน (Flowchart)*
2. *เขียนโปรแกรมตามที่ออกแบบไว้ ด้วยภาษาซี บนระบบปฏิบัติการ CentOS*
3. *เขียนอธิบายโค้ดโปรแกรมอย่างละเอียด*
4. *บันทึกผลการทดลอง และสรุปผล*
5. *ส่งไฟล์รูปเล่มใบงาน พร้อมอัดคลิปแสดงผลการรันโปรแกรมที่เขียนขึ้นมาใน MS Team*

โจทย์ให้เขียนโปรแกรมเพื่อจำลองการทำงานของ CPU Scheduling ตามอัลกอริทึมที่กำหนดให้ต่อไปนี้

Non preemptive SJF scheduling.	Preemptive SJF scheduling.
Round Robin scheduling. (Time quantum = 4)	Priority scheduling.

โดยใช้ข้อมูลจากตารางนี้ เป็นข้อมูลเริ่มต้นในการสร้างโปรแกรม

Process (ใช้ทุกอัลกอริทึม)	**Burst Time** (ใช้ทุกอัลกอริทึม)	**Arrival Time** (ใช้ทุกอัลกอริทึม)	**Priority** (ใช้เฉพาะ **Priority scheduling**)
P1	9	1	3
P2	3	1	5
P3	5	3	1
P4	4	4	4
P5	2	7	2

กำหนดให้แสดงผลลัพธ์ของโปรแกรมในการรัน 1 ครั้ง โดยแยกการทำงานของแต่ละอัลกอริทึม ดังนี้

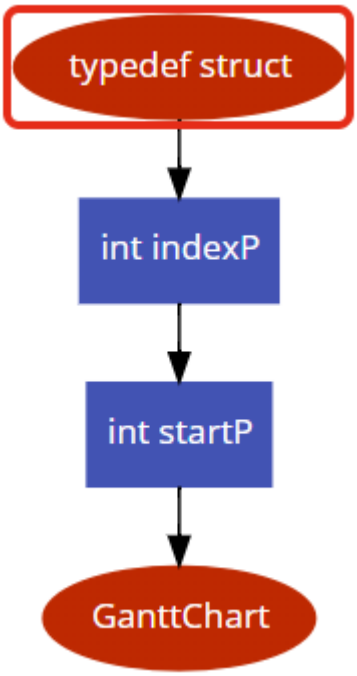
1. ชื่ออัลกอริทึมที่ดำเนินการ
2. ลำดับการทำงานของ Process
3. เวลารอคอยของแต่ละ Process
4. เวลารอคอยเฉลี่ยของอัลกอริทึมที่ดำเนินการ
5. เวลาครบวงงาน (Turnaround time) ของแต่ละ Process

ตัวอย่างผลลัพธ์

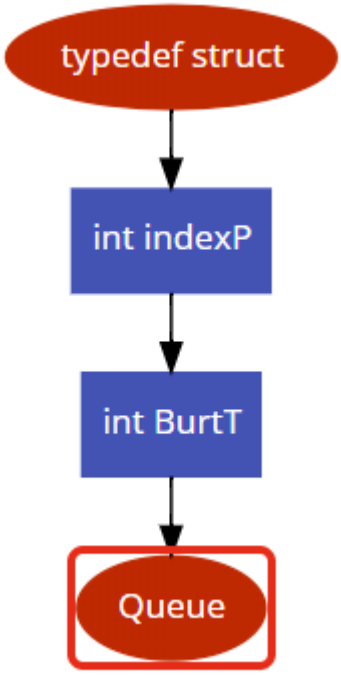
```
# Mr.Firstname Surname ID:60570909099-9 Sec.01    #
# OUTPUT LAB 6 CPU Scheduling                     #
## 1.FCFS Scheduling ##
---------------------------------------------------
Sequence process is :P1->P2->P3->P4->P5
Wait time of process (millisecond)
| P1          | P2          | P3          | P4          | P5
| 00.00       | 09.00       | 10.00       | 14.00       | 15.00
Average time is :09.60 ms
Turnaround time
P1=9.00 ms | P2=12.00 ms | P3=15.00 ms | P4=18.00 ms | P5=17.00 ms
---------------------------------------------------
```

Priority Flow chart

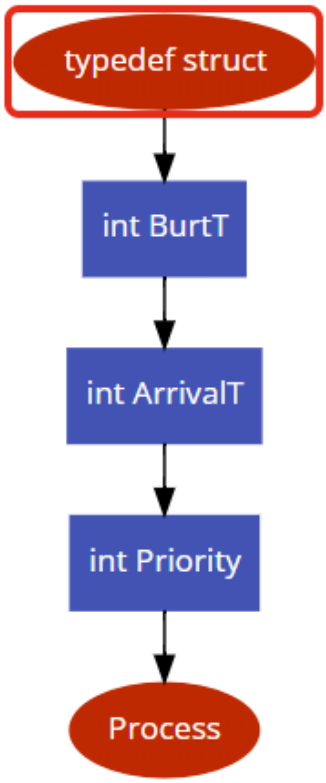
GanttChart



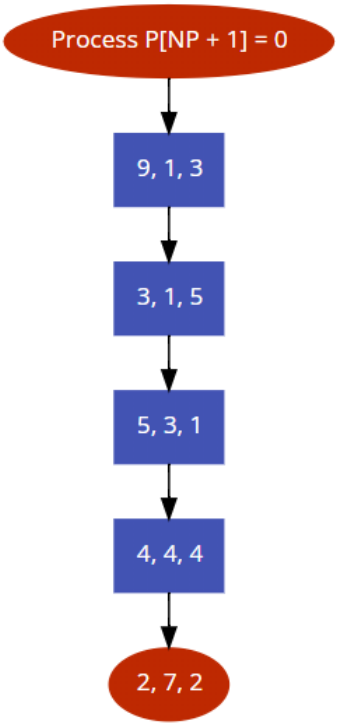
Queue



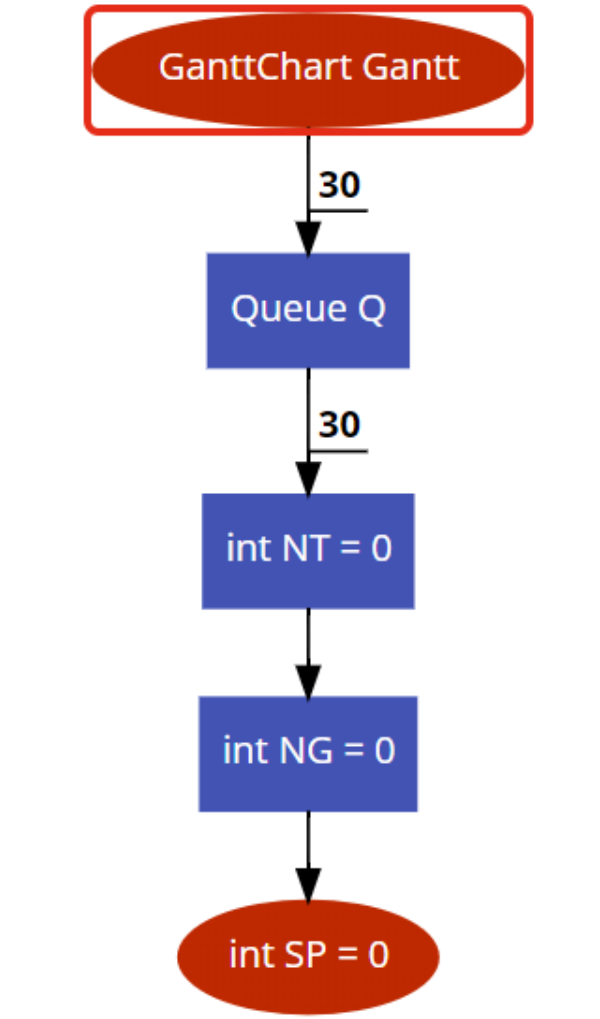
Process



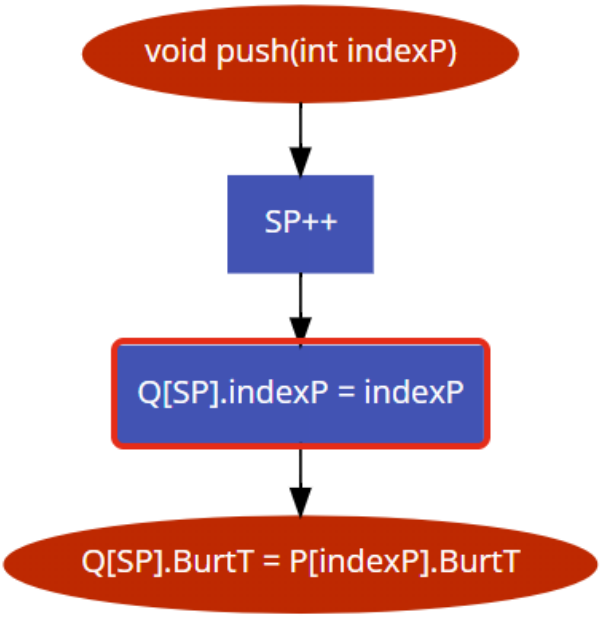
Process of p



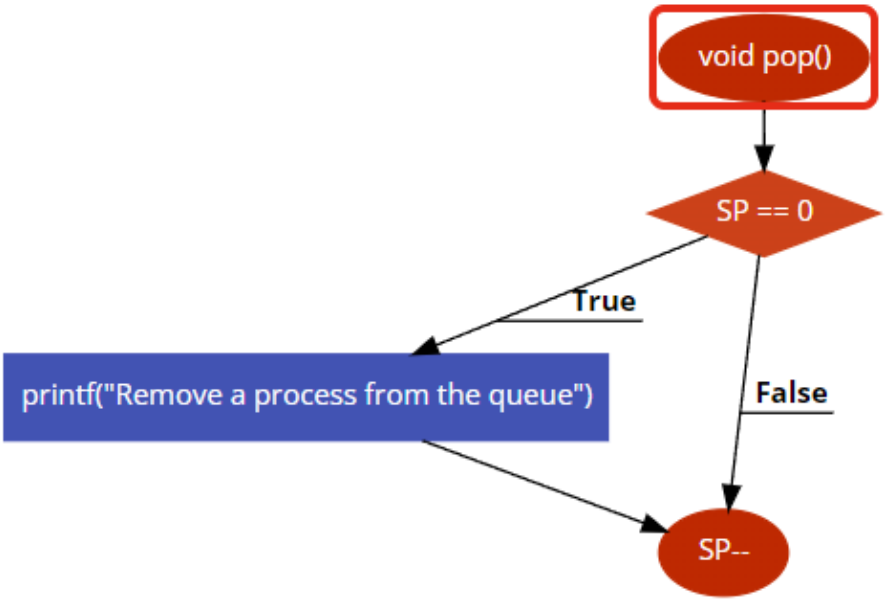
GanttChart n Queue



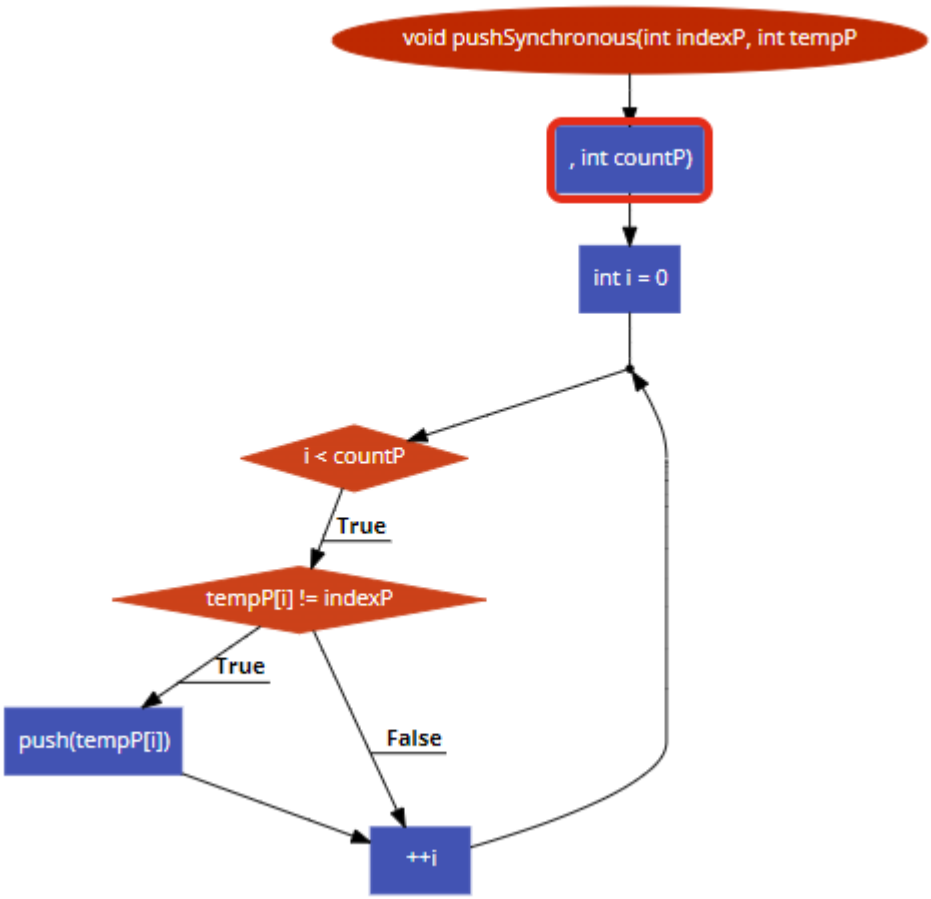
Function push



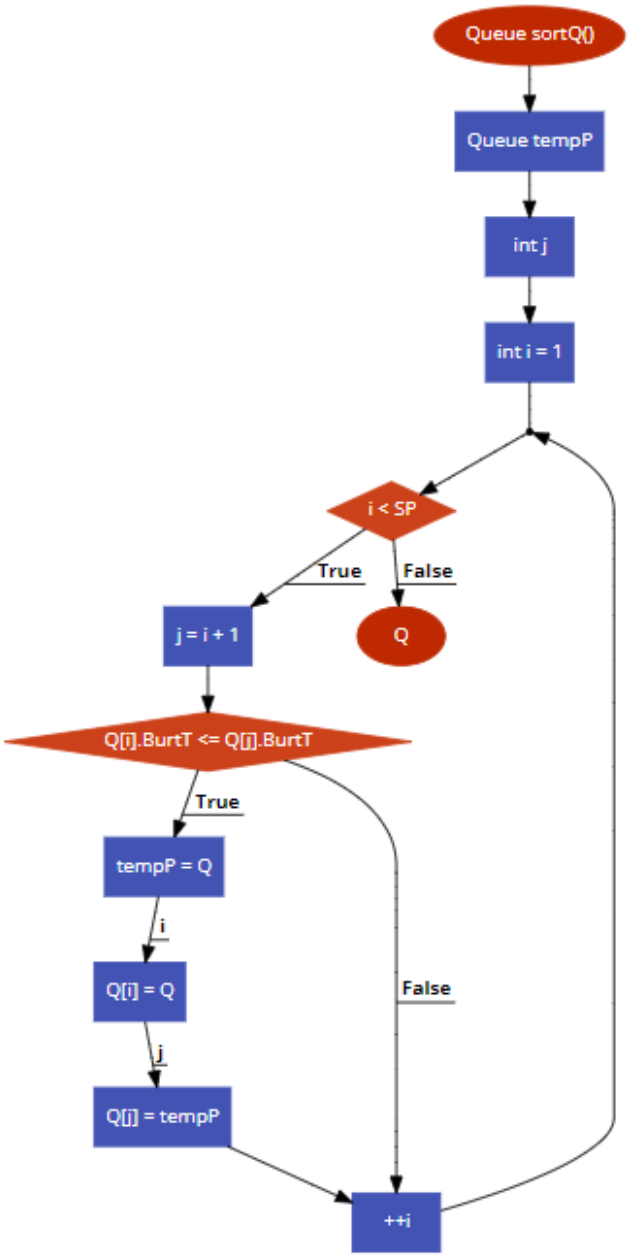
Function pop



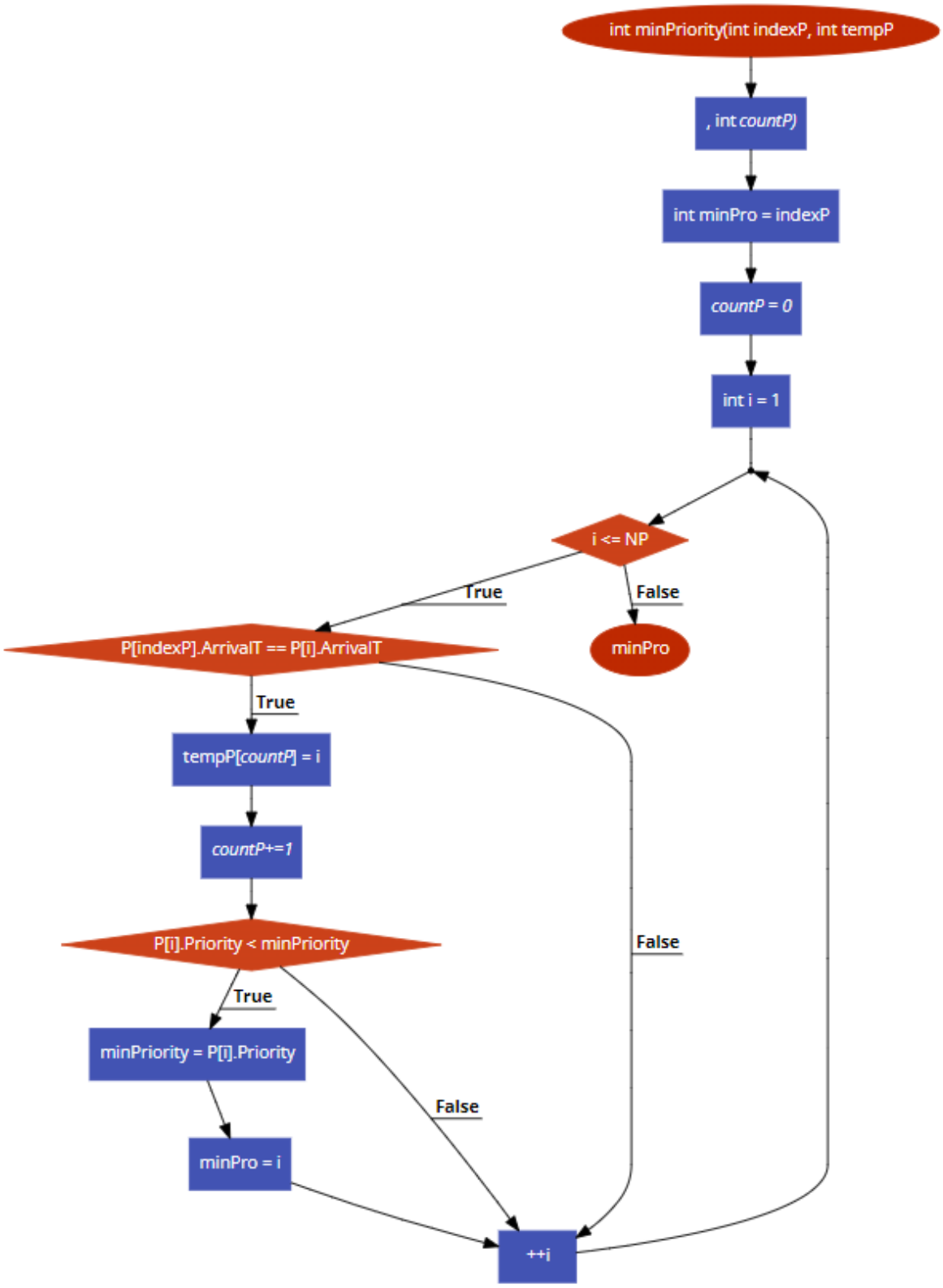
Function pushSynchronous



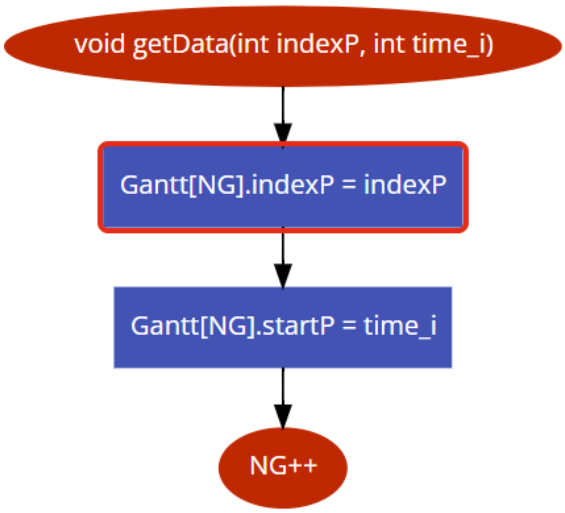
Function Queue sortq



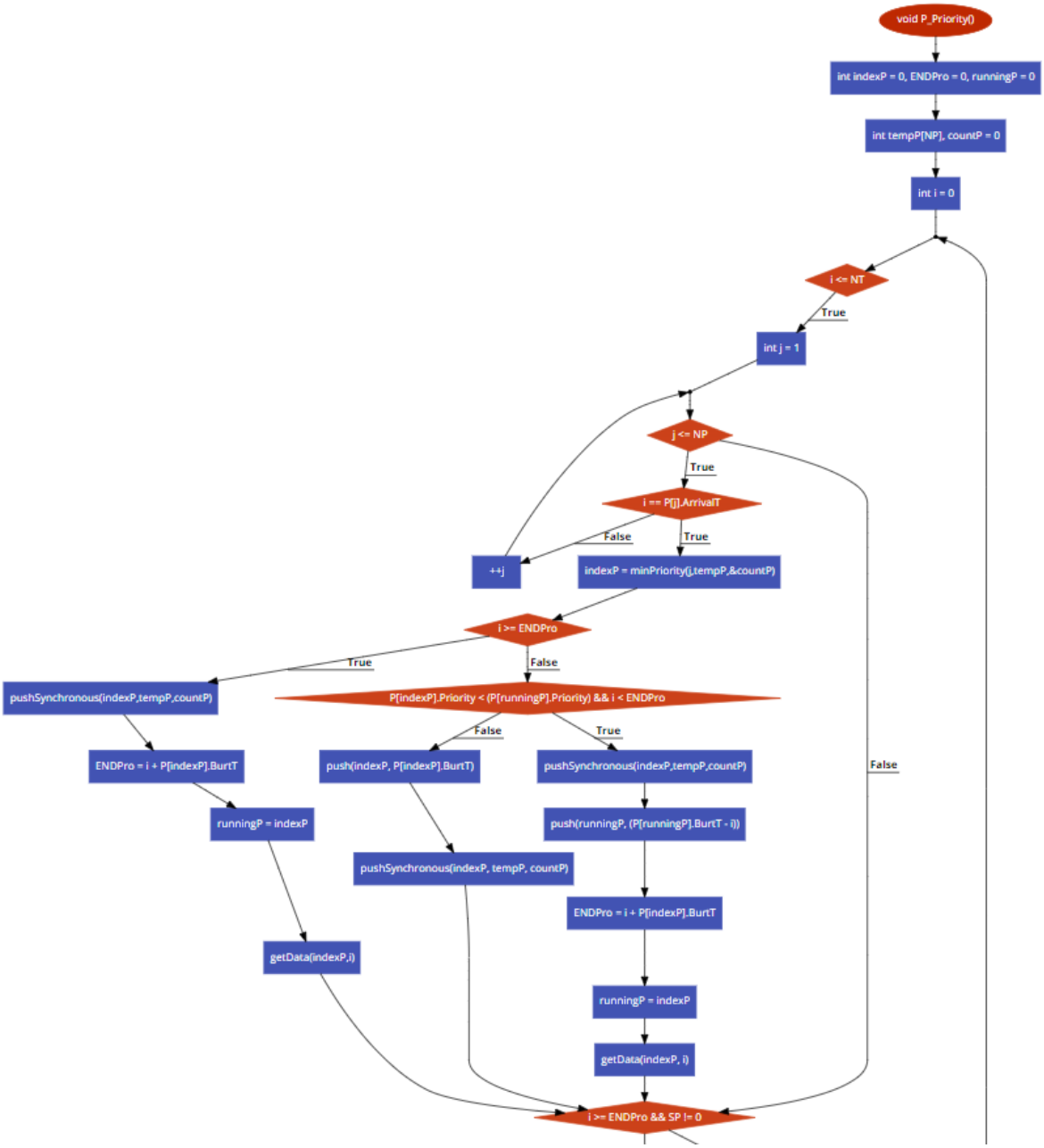
Function Priority

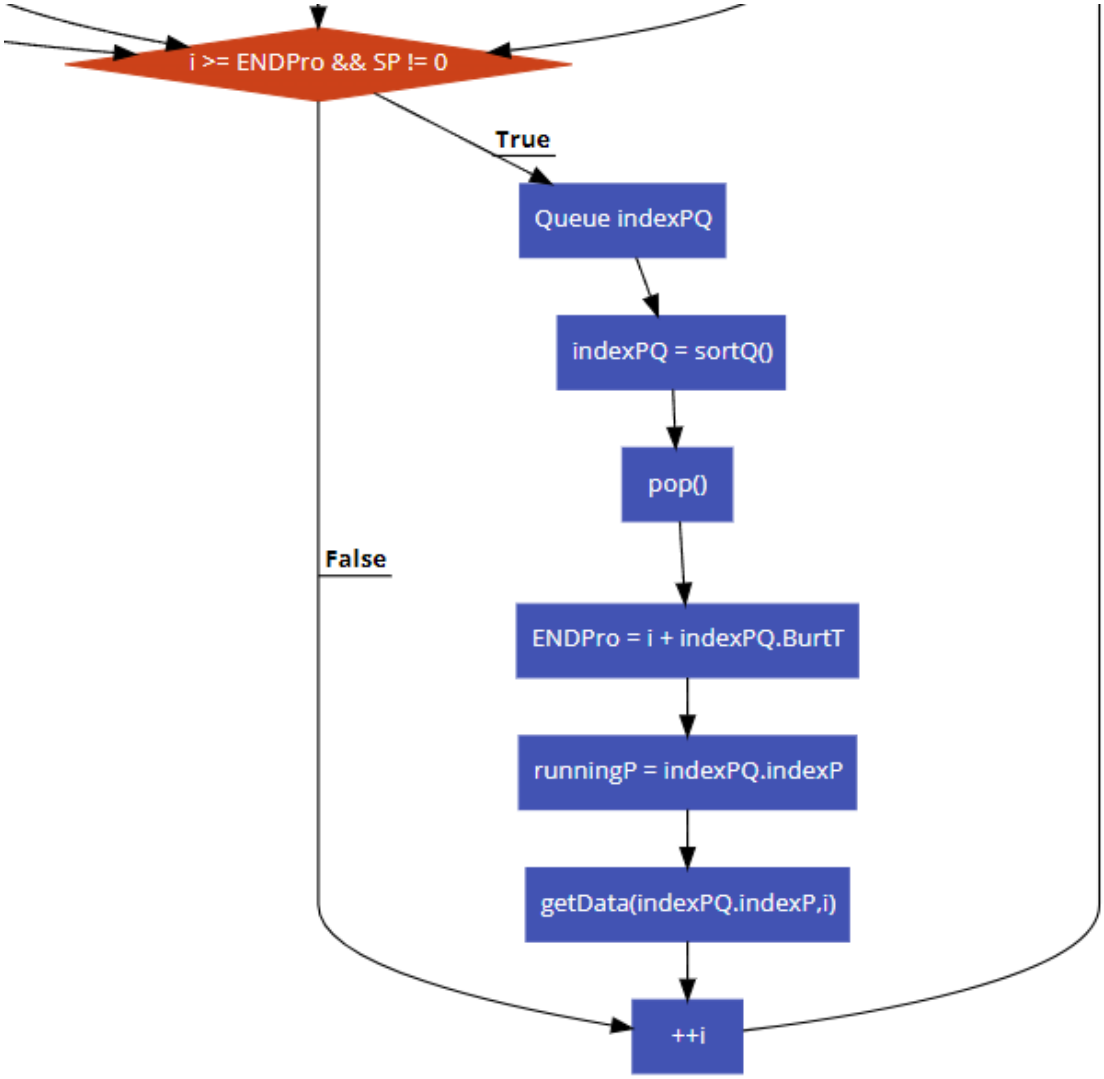


Function Getdata

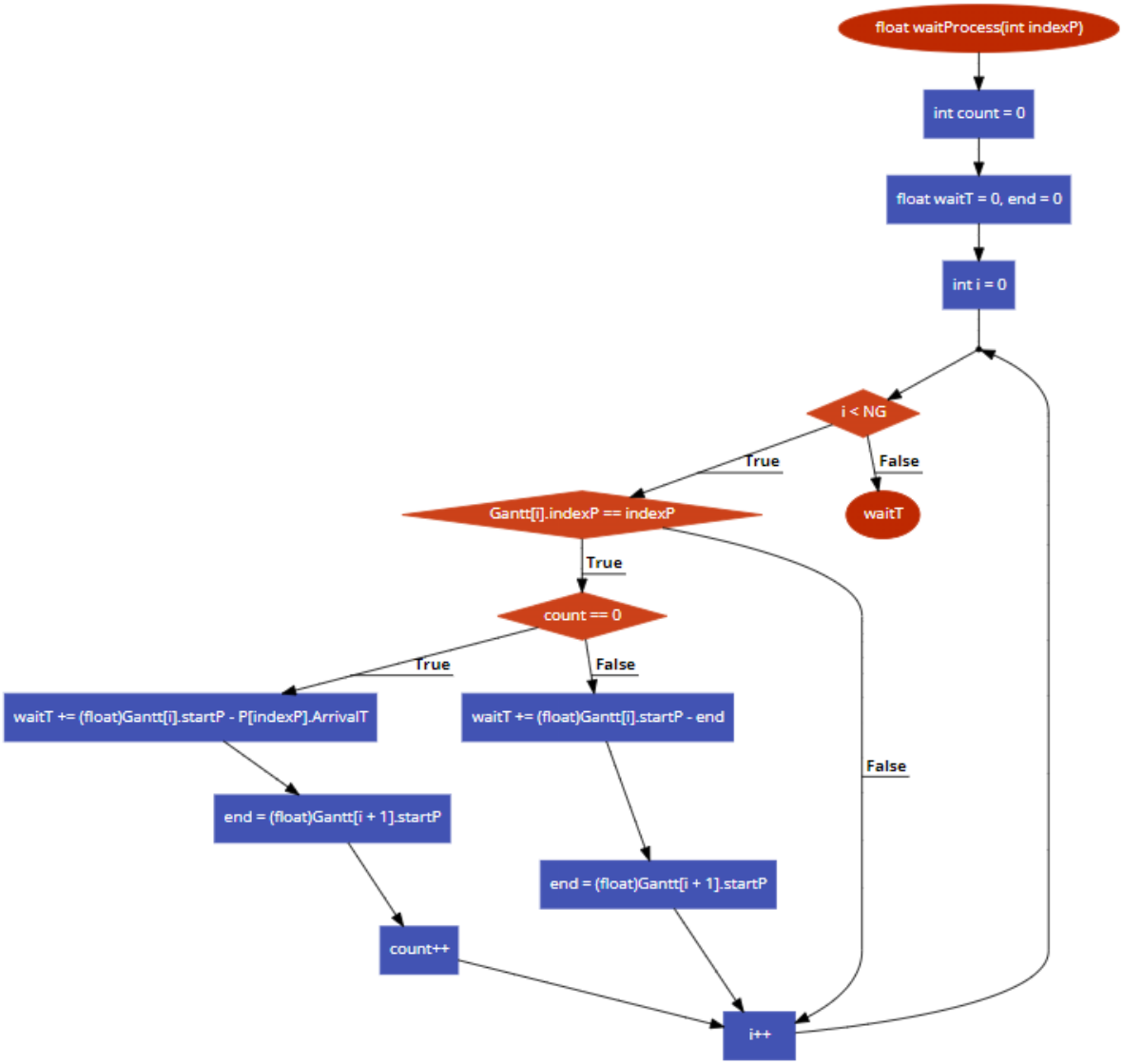


Function Priority

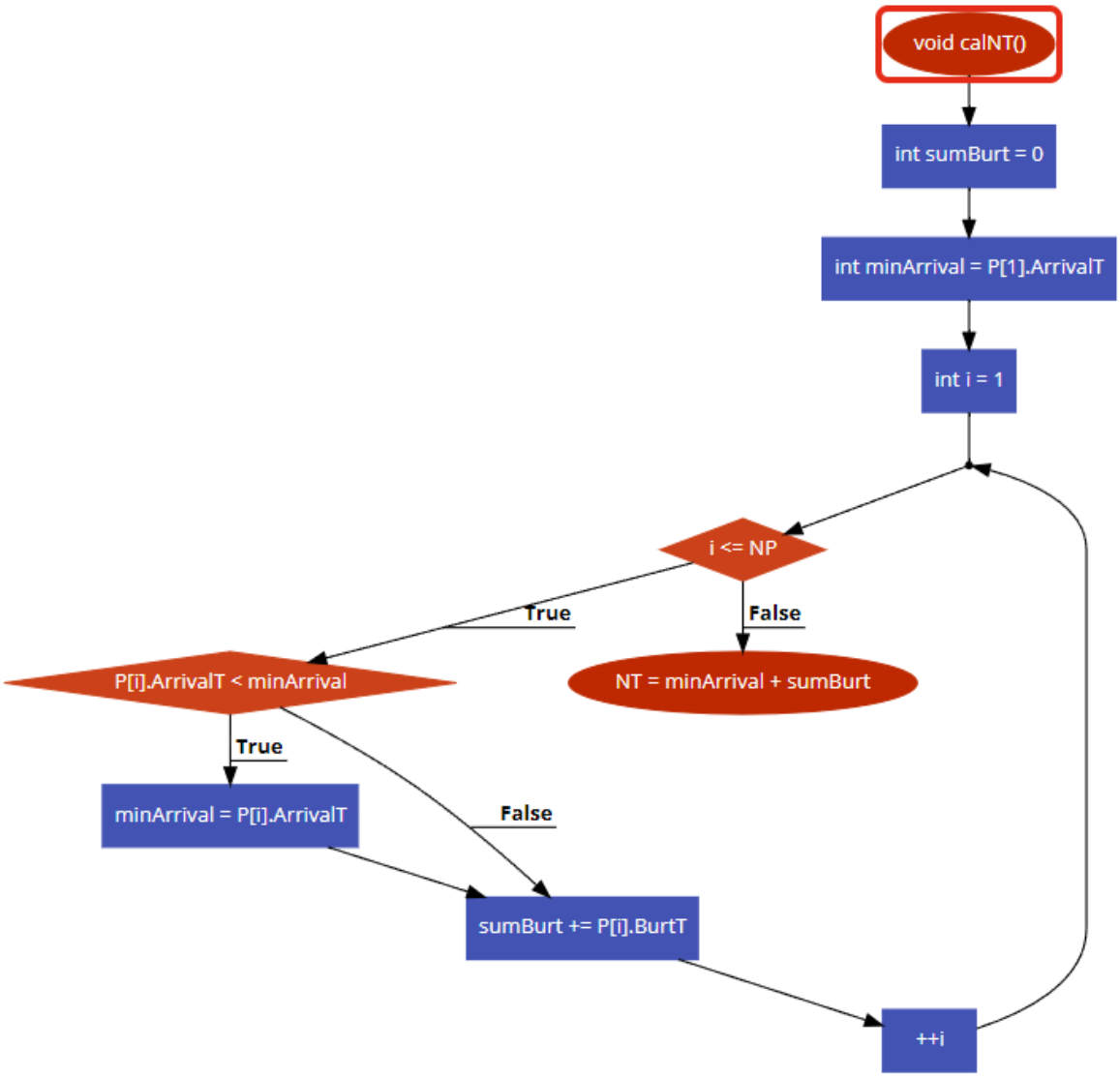



i >= ENDPro && SP != 0
True
Queue indexPQ
indexPQ = sortQ()
pop()
False
ENDPro = i + indexPQ.BurtT
runningP = indexPQ.indexP
getData(indexPQ.indexP,i)
++i

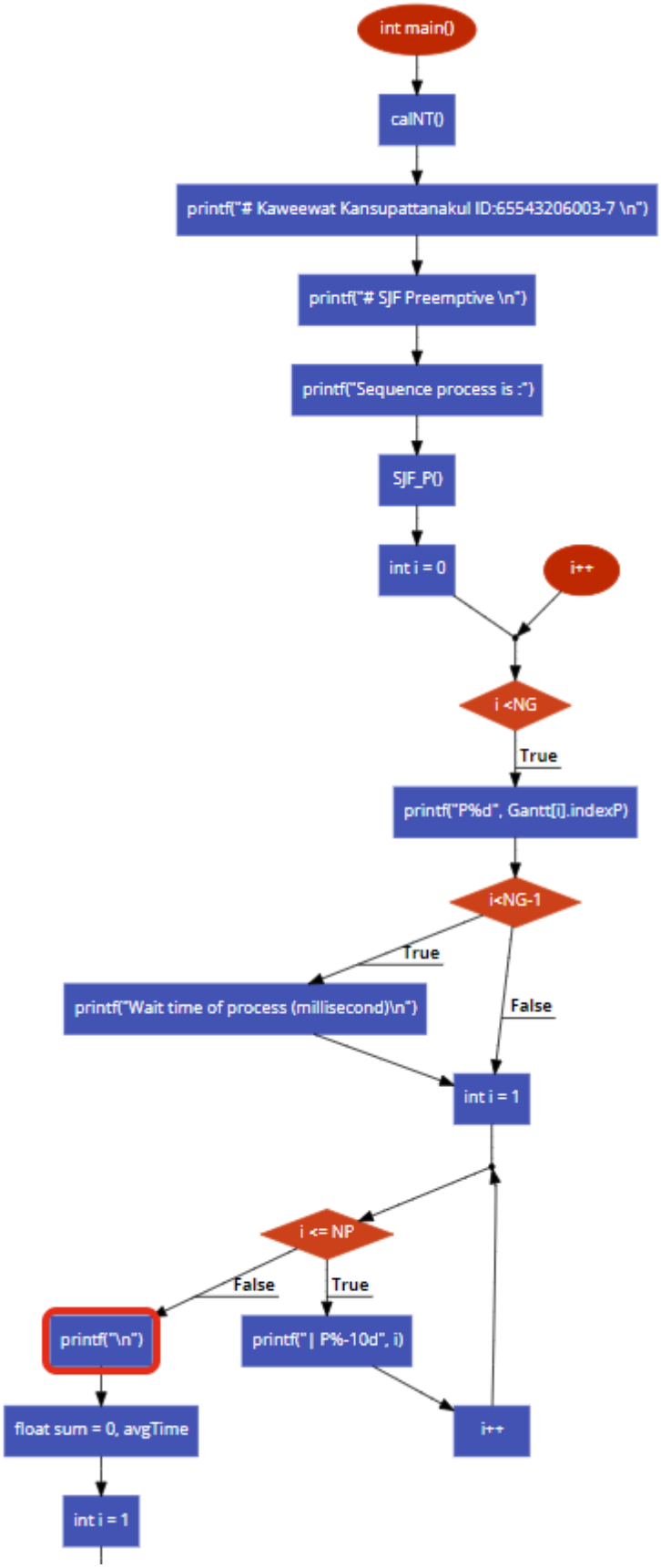
Function WaitProcess

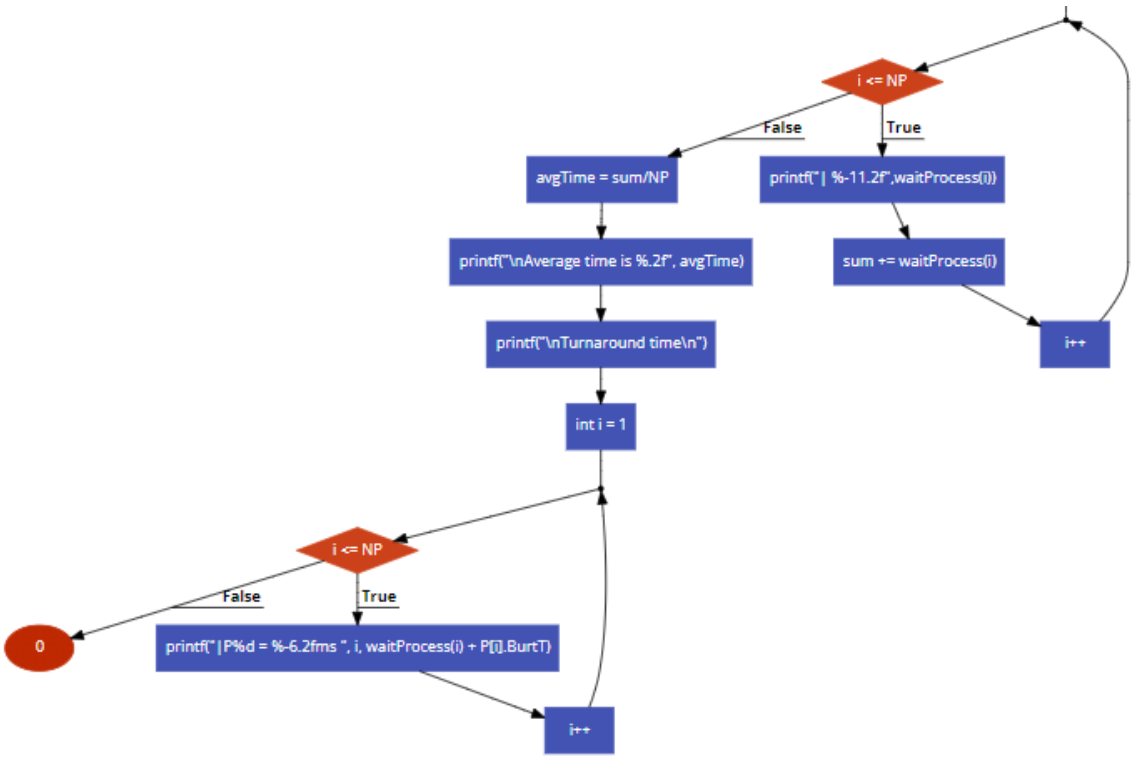


Function CalNT



Main function




i <= NP
False
True
avgTime = sum/NP
printf(" | %-11.2f",waitProcess(i))
printf("\nAverage time is %.2f", avgTime)
sum += waitProcess(i)
printf("\nTurnaround time\n")
i++
int i = 1
i <= NP
False
True
0
printf(" |P%d = %-6.2fms ", i, waitProcess(i) + P[i].BurtT)
i++

Code

```c
#include <stdio.h>
#define NP 5

You, 1 hour ago | 1 author (You)
typedef struct {
    int indexP;
    int startP;
} GanttChart;

You, 1 hour ago | 1 author (You)
typedef struct {
    int indexP;
    int BurtT;
} Queue;

You, 1 hour ago | 1 author (You)
typedef struct {
    int BurtT;
    int ArrivalT;
    int Priority;
} Process;

Process P[NP + 1] = {{0},
                     /*P1*/ {9, 1, 3},
                     /*P2*/ {3, 1, 5},
                     /*P3*/ {5, 3, 1},
                     /*P4*/ {4, 4, 4},
                     /*P5*/ {2, 7, 2}};

GanttChart Gantt[30];
Queue Q[30];
int NT = 0;  // Time
int NG = 0;  // Number GanttChart
int SP = 0;  // Pointer in Queue
```

1. GanttChart (โครงสร้างข้อมูลของแผนภูมิ GanttChart):
   - มี indexP เพื่อระบุลำดับของกระบวนการ
   - มี startP เพื่อระบุเวลาที่กระบวนการเริ่มทำงาน
2. Queue (โครงสร้างข้อมูลของคิวกระบวนการ):
   - มี indexP เพื่อระบุลำดับของกระบวนการ
   - มี BurtT เพื่อระบุเวลาที่กระบวนการต้องการในการประมวลผล
3. Process (โครงสร้างข้อมูลของกระบวนการ):
   - มี BurtT เพื่อระบุเวลาที่กระบวนการต้องการในการประมวลผล
   - มี ArrivalT เพื่อระบุเวลาที่กระบวนการมาถึง
   - มี Priority เพื่อระบุลำดับความสำคัญ

```
void push(int indexP) {
    SP++;
    Q[SP].indexP = indexP;
    Q[SP].BurtT = P[indexP].BurtT;
}

void pop() {
    if (SP == 0)
        printf("UNDER FLOW!!!\n");
    SP--;
}

Queue sortQ() {
    Queue tempP;
    int j;
    for (int i = 1; i < SP; ++i) {
        j = i + 1;
        if (Q[i].BurtT <= Q[j].BurtT) {
            tempP = Q[i];
            Q[i] = Q[j];
            Q[j] = tempP;
        }
    }
    return Q[SP];
}
```

1. push
   ใช้เพื่อเพิ่มกระบวนการลงในคิว
   รับ indexP เพื่อระบุลำดับของกระบวนการ
   เพิ่มข้อมูลลงในคิวที่ตำแหน่ง SP (Pointer in Queue)
2. pop
   ใช้เพื่อนำกระบวนการออกจากคิว
   ตรวจสอบว่าคิวว่างหรือไม่
3. Queue sortQ
   ใช้เพื่อเรียงลำดับคิวกระบวนการตามเวลาที่ต้องการในการประมวลผล
   ส่งคืนข้อมูลของกระบวนการที่มีเวลาน้อยที่สุดในคิว

```
int minPriority(int indexP, int tempP[], int *countP) {
    int minPriority = P[indexP].Priority;
    int minPro = indexP;
    *countP = 0;
    for (int i = 1; i <= NP; ++i)
        if (P[indexP].ArrivalT == P[i].ArrivalT) {
            tempP[*countP] = i;
            *countP+=1;
            if(P[i].Priority < minPriority ){
                minPriority = P[i].Priority;
                minPro = i;
            }
        }
    return minPro;
}

void pushSynchronous(int indexP, int tempP[], int countP) {
    for (int i = 0; i < countP ; ++i)
        if(tempP[i] != indexP )
            push(tempP[i],P[tempP[i]].BurtT);
}

void getData(int indexP, int time_i){
    Gantt[NG].indexP = indexP;
    Gantt[NG].startP = time_i;          You, 58 second
    NG++;
}
```

4. MinPriority

กำหนดค่า minPriority เป็นค่าความสำคัญของโปรเซสที่ถูกส่งเข้ามาทาง indexP.

กำหนด minPro เป็นเลขลำดับของโปรเซสที่ถูกส่งเข้ามาทาง indexP.

กำหนด countP เป็น 0 เพื่อเตรียมให้นับจำนวนโปรเซสที่มีเวลาเข้าถึงเดียวกัน.

5. pushSynchronous

ใช้เพื่อเพิ่มกระบวนการที่มี ArrivalT เท่ากันในคิว

รับ indexP เพื่อระบุลำดับของกระบวนการ

รับ tempP เพื่อระบุลำดับของกระบวนการที่มี ArrivalT เท่ากัน

รับ countP เพื่อระบุจำนวนของกระบวนการที่มี ArrivalT เท่ากัน

6. getData

ใช้เพื่อเก็บข้อมูลของแผนภูมิ GanttChart

รับ indexP เพื่อระบุลำดับของกระบวนการ

รับ time_i เพื่อระบุเวลาที่กระบวนการเริ่มทำงาน

```
void P_Priority(){
    int indexP = 0, ENDPro = 0, runningP = 0;
    int tempP[NP], countP = 0;
    for (int i = 0; i <= NT; ++i) {
        for (int j = 1; j <= NP; ++j) {
            if(i == P[j].ArrivalT){
                indexP = minPriority(j,tempP,&countP);
                if(i >= ENDPro){
                    pushSynchronous(indexP,tempP,countP);
                    ENDPro = i + P[indexP].BurtT;
                    runningP = indexP;
                    getData(indexP,i);
                }else{
                    if(P[indexP].Priority < (P[runningP].Priority) && i < ENDPro){
                        pushSynchronous(indexP,tempP,countP);
                        push(runningP, (P[runningP].BurtT - i));
                        ENDPro = i + P[indexP].BurtT;
                        runningP = indexP;
                        getData(indexP, i);
                    }else {
                        push(indexP, P[indexP].BurtT);
                        pushSynchronous(indexP, tempP, countP);
                    }
                }
                break;
            }
        }
```

```
void SJF_NP() { //
    int indexP = 0, ENDPro = 0;
    int tempP[NP], countP = 0;
    for (int i = 0; i <= NT; ++i) {
        for (int j = 1; j <= NP; ++j) {
            if (i == P[j].ArrivalT) {
                indexP = minBurt(j, tempP, &countP);
                if (i >= ENDPro && SP == 0) {
                    pushSynchronous(indexP, tempP, countP);
                    ENDPro = i + P[indexP].BurtT;
                    getData(indexP, i);
                } else {
                    push(indexP);
                    pushSynchronous(indexP, tempP, countP);
                }
                break;
            }
        }
        if (i >= ENDPro && SP != 0) {
            Queue indexPQ;
            indexPQ = sortQ();
            pop();
            ENDPro = i + indexPQ.BurtT;
            getData(indexPQ.indexP, i);
        }
    }
}       You, last week • updata l6 …
        if (i >= ENDPro && SP != 0) {
            Queue indexPQ;
            indexPQ = sortQ();
            pop();
            ENDPro = i + indexPQ.BurtT;
            runningP = indexPQ.indexP;
            getData(indexPQ.indexP,i);
        }
    }
}
```

7. Priority
    1. กำหนดตัวแปร indexP เพื่อเก็บลำดับของโปรเซสที่มีเวลาเข้าถึง (Arrival Time) ตรงกับ i.
    2. สร้างอาเรย์ tempP เพื่อใช้เก็บลำดับของโปรเซสที่มีเวลาเข้าถึงเดียวกัน.
    3. ตั้งค่า countP เป็น 0 เพื่อให้ฟังก์ชัน minPriority สามารถใช้เก็บลำดับโปรเซสที่มีเวลาเข้าถึงเดียวกันใน tempP.
    4. ทำลูปภายใน for เพื่อตรวจสอบโปรเซสทั้งหมด ในกรณีที่เวลาปัจจุบัน i เท่ากับเวลาเข้าถึงของโปรเซส P[j].ArrivalT ในลูปนอก.
    5. หา indexP โดยใช้ฟังก์ชัน minPriority ที่กำหนดไว้ก่อนหน้า.
    6. ตรวจสอบว่าเวลาปัจจุบัน i มีค่าไม่น้อยกว่าเวลาสิ้นสุดของโปรเซสที่กำลังทำงาน (ENDPro) หรือไม่.

```
float waitProcess(int indexP) {
    int count = 0;
    float waitT = 0, end = 0;
    for (int i = 0; i < NG; i++) {
        if (Gantt[i].indexP == indexP) {
            if (count == 0) {
                waitT += (float)Gantt[i].startP - P[indexP].ArrivalT;
                end = (float)Gantt[i + 1].startP;
                count++;
            } else {
                waitT += (float)Gantt[i].startP - end;
                end = (float)Gantt[i + 1].startP;
            }
        }
    }
    return waitT;
}       You, last week • updata 16 …

void calNT() {
    int sumBurt = 0;
    int minArrival = P[1].ArrivalT;
    for (int i = 1; i <= NP; ++i) {
        if (P[i].ArrivalT < minArrival) {
            minArrival = P[i].ArrivalT;
        }
        sumBurt += P[i].BurtT;
    }
    NT = minArrival + sumBurt;
}
```

8. waitProcess

   ใช้เพื่อคำนวณเวลารอของกระบวนการ

   รับ indexP เพื่อระบุลำดับของกระบวนการ

   ส่งคืนเวลารอของกระบวนการ

9. calNT

   ประกาศตัวแปล minArrival

   หาค่าเวลาเข้าที่น้อยที่สุด

   เก็บค่าไว้ในตัวแปล NT

```c
int main() {
    calNT();
    printf("# Kaweewat Kansupattanakuk ID:65543206003-7\n");
    printf("# OUTPUT LAB6 CPU Scheduling\n");
    printf("# SJF Non Preemptive \n");
    printf("Sequence process is :");
    SJF_NP();
    for (int i = 0; i < NG; i++) {
        printf("P%d", Gantt[i].indexP);
        if (i < NG - 1)
            printf("->");
    }
    printf("\n-----------------------------------------------------------------\n");
    printf("Wait time of process (millisecond)\n");
    for (int i = 1; i <= NP; i++) {
        printf("| P%-10d", i);
    }
    printf("\n");
    float sum = 0, avgTime;
    for (int i = 1; i <= NP; i++) {
        printf("| %-11.2f", waitProcess(i));
        sum += waitProcess(i);
    }
    avgTime = sum / NP;
    printf("\nAverage time is %.2f", avgTime);
    printf("\nTurnaround time\n");
    for (int i = 1; i <= NP; i++) {
        printf("|P%d = %-6.2fms ", i, waitProcess(i) + P[i].BurtT);
    }
    printf("\n-----------------------------------------------------------------\n");
    return 0;
}
```

10. Main

    แสดงค่าของ Process P1,P2

    แสดงค่าของ wait time

    แสดง avg Time

    แสดง turn aroundtime

ผลลัพธ์

```
# Kaweewat Kansupattanakul 65543206003-7
# OUTPUT LAB6 CPU Scheduling
# Priority (SJF Preemptive)
Sequence process is :P1->P3->P5->P1->P4->P2
------------------------------------------------------------
Wait time of process (millisecond)
| P1         | P2         | P3         | P4         | P5
| 7.00       | 19.00      | 0.00       | 12.00      | 1.00
Average time is 7.80
Turnaround time
|P1 = 16.00 ms |P2 = 22.00 ms |P3 = 5.00  ms |P4 = 16.00 ms |P5 = 3.00  ms
------------------------------------------------------------
```