ใบงานที่ 6

เรื่อง CPU Scheduling

Round Robin

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

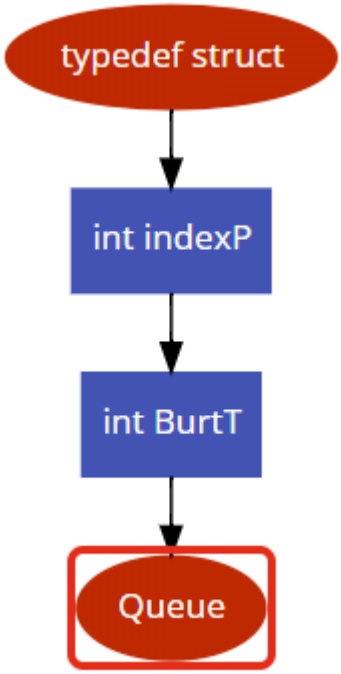
1. ชื่ออัลกอริทึมที่ดำเนินการ
2. ลำดับการทำงานของ Process
3. เวลารอคอยของแต่ละ Process
4. เวลารอคอยเฉลี่ยของอัลกอริทึมที่ดำเนินการ
5. เวลาครบวงงาน (Turnaround time) ของแต่ละ Process

ตัวอย่างผลลัพธ์

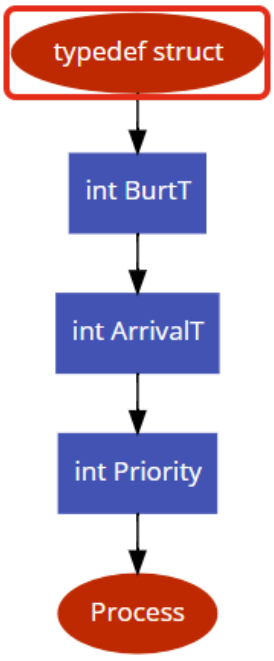
```
# Mr.Firstname Surname ID:60570909099-9 Sec.01    #
# OUTPUT LAB 6 CPU Scheduling                     #
## 1.FCFS Scheduling ##
---------------------------------------------------
Sequence process is :P1->P2->P3->P4->P5
Wait time of process (millisecond)
| P1          | P2          | P3          | 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
---------------------------------------------------
```

RoundRobin Flow chart

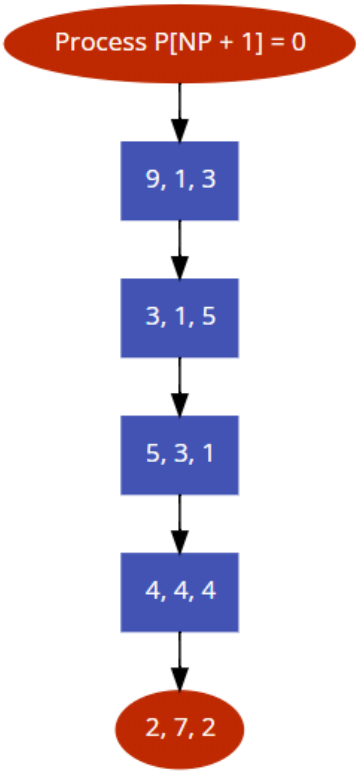
Queue



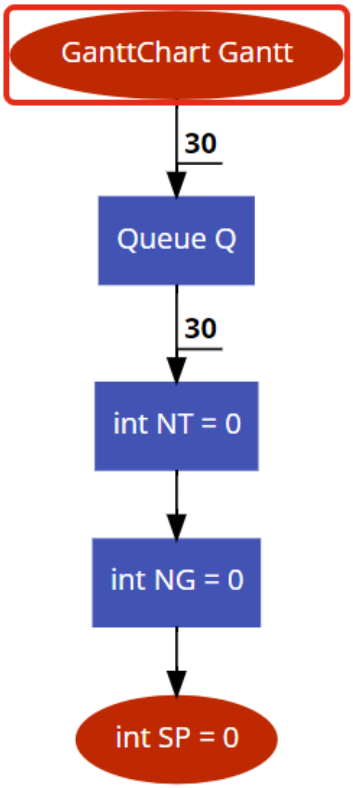
Process



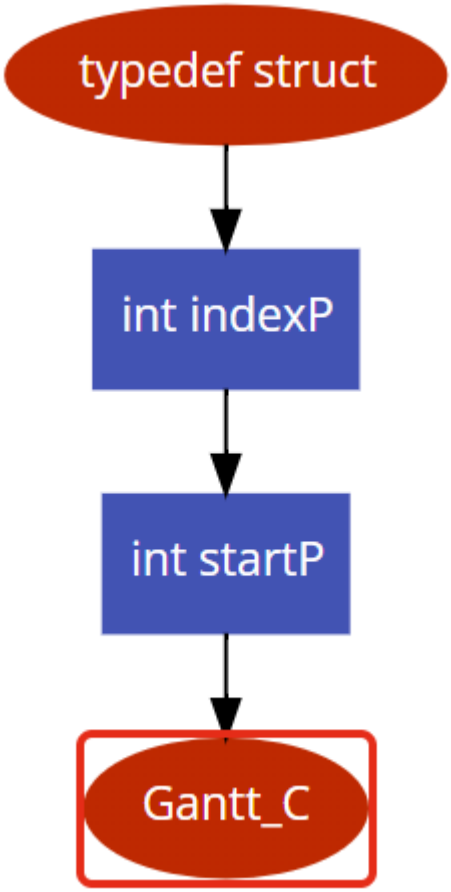
Process of p



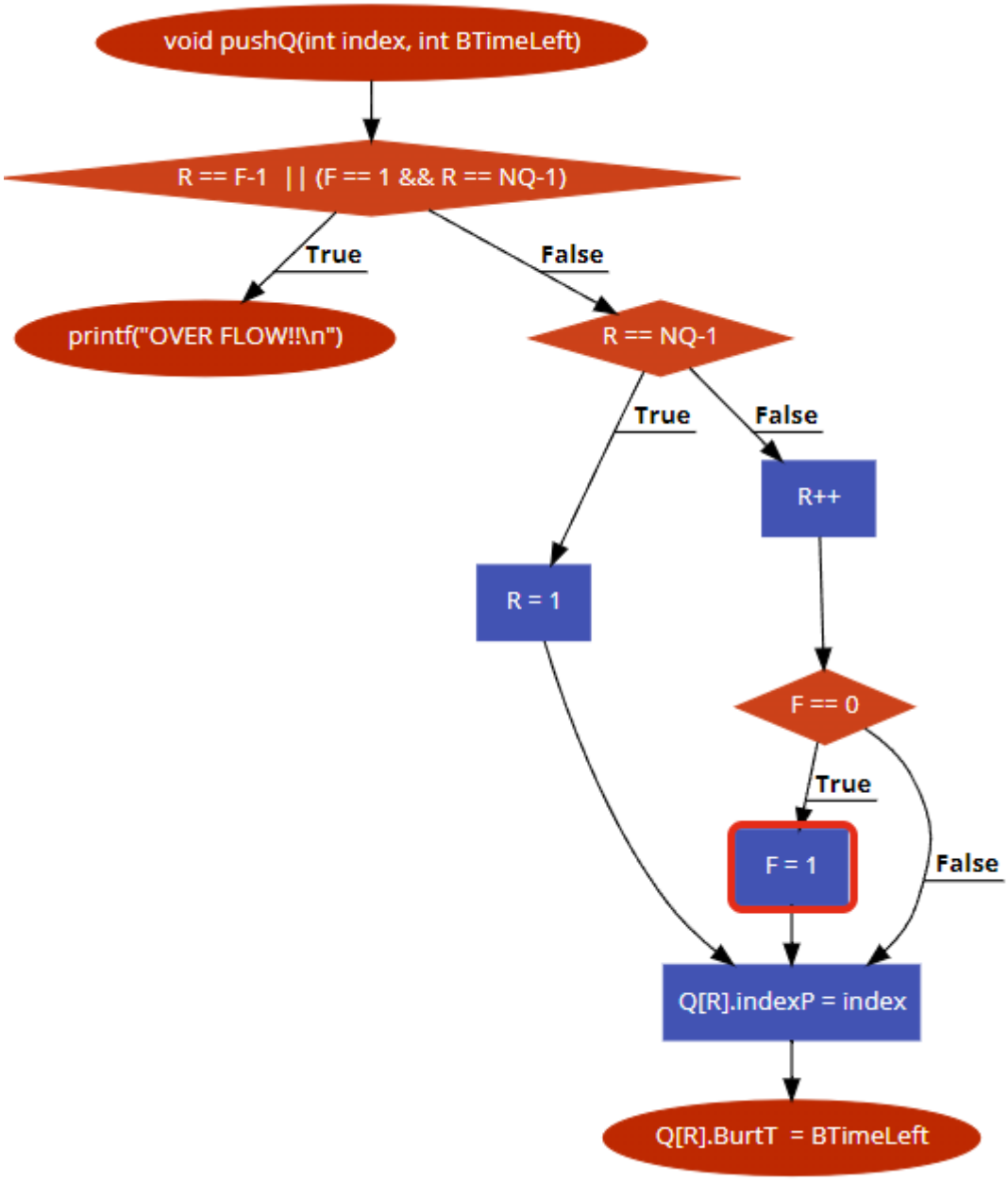
GanttChart n Queue



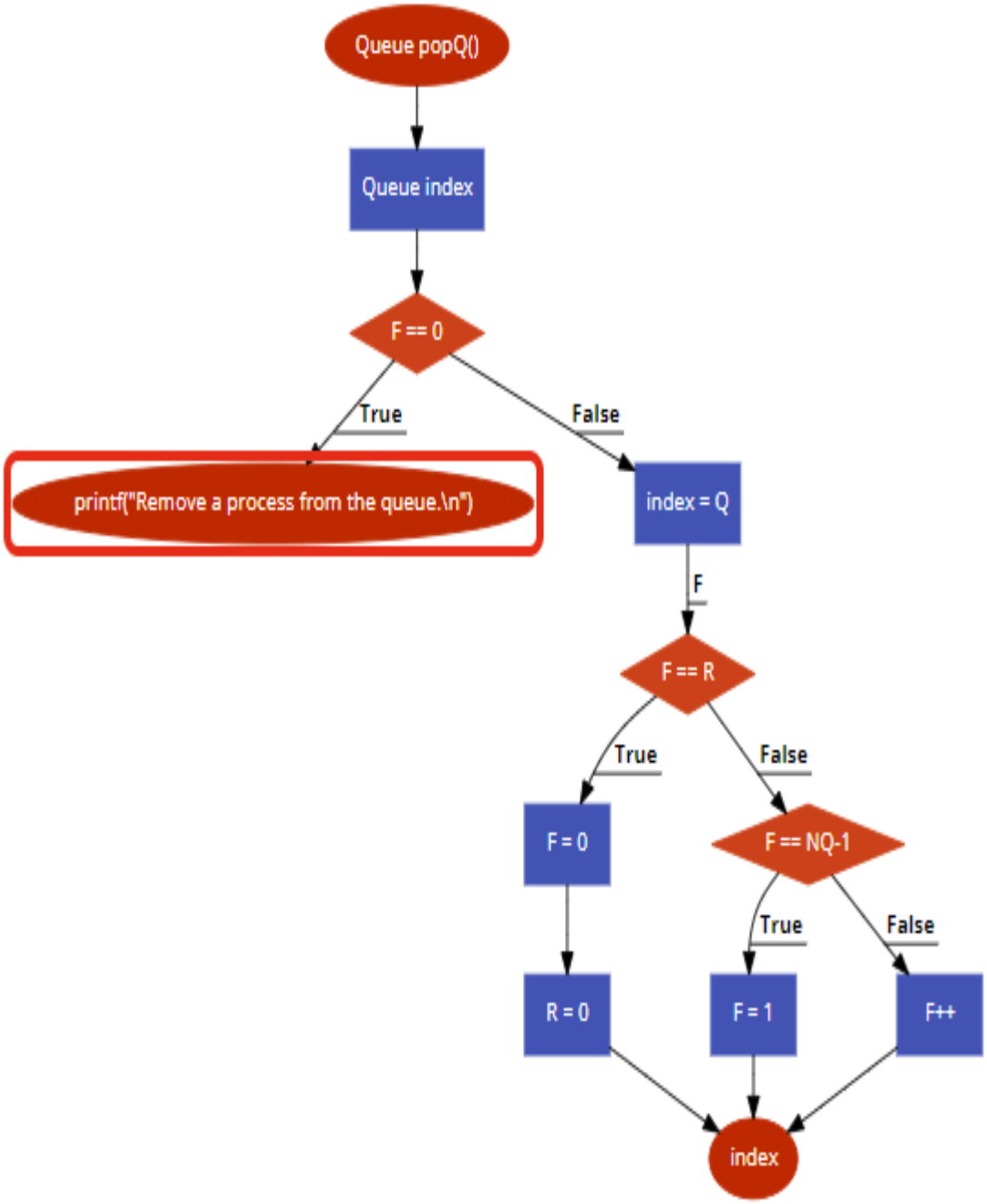
Gantt_c



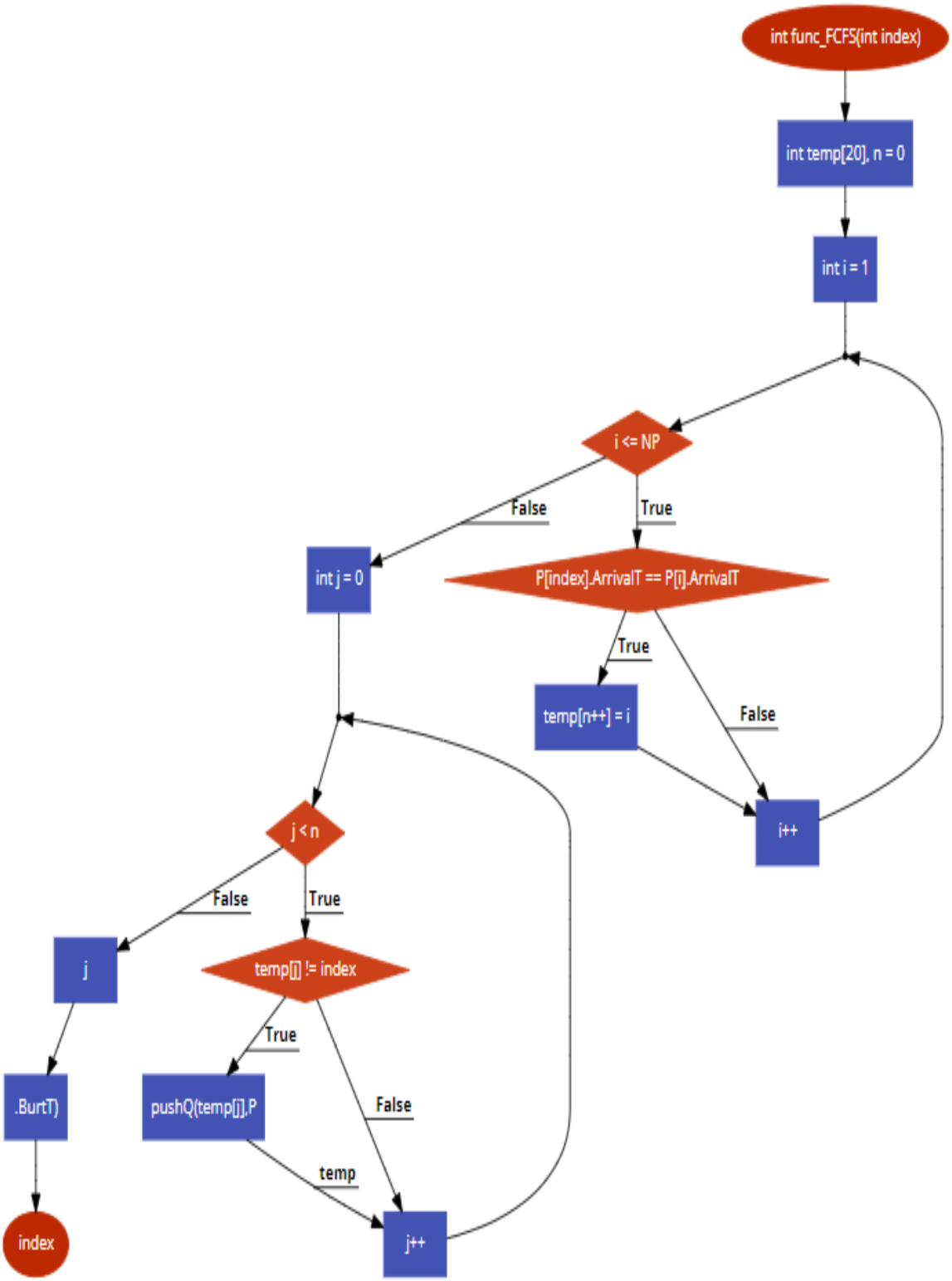
Function pushQ



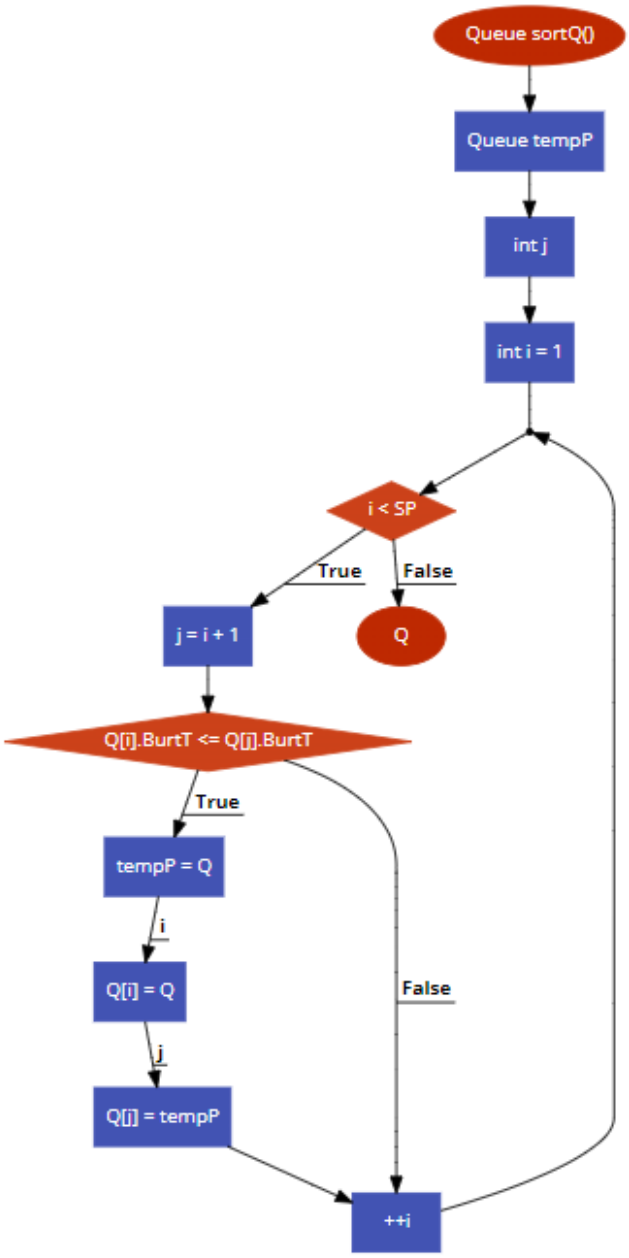
Function pop



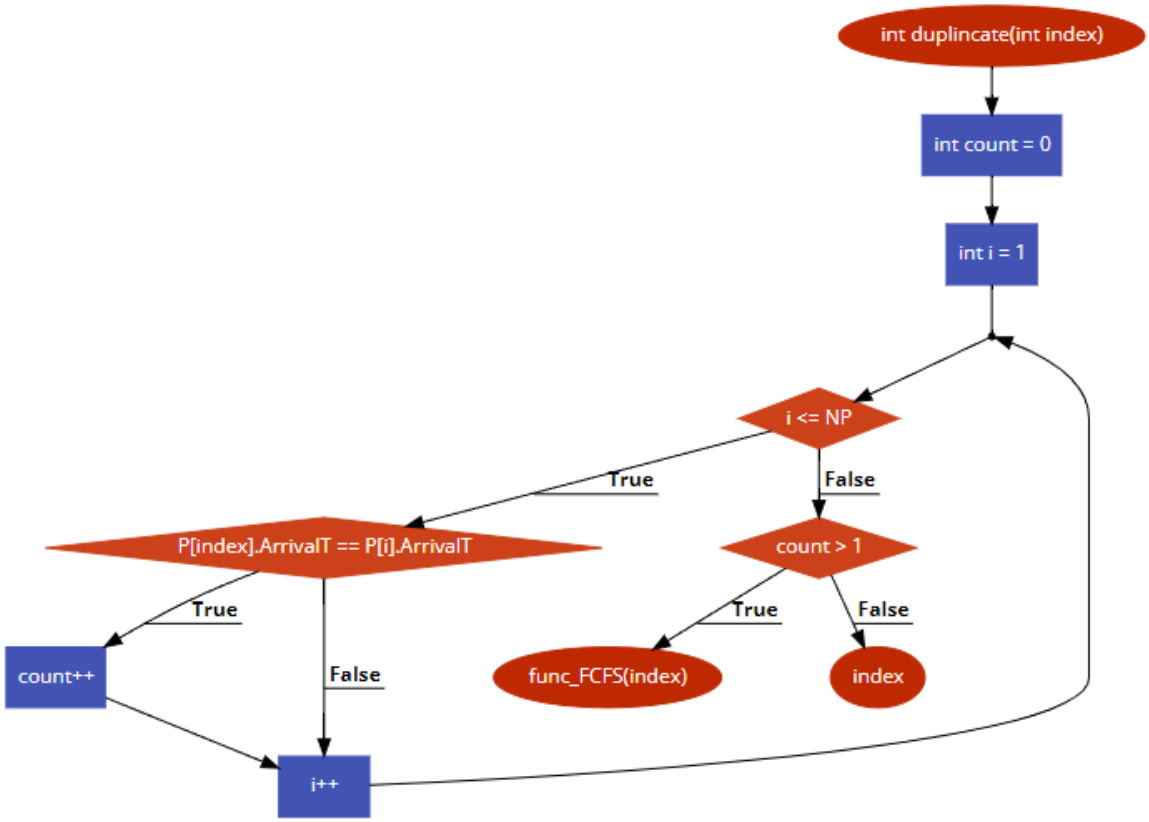
Function FCFS



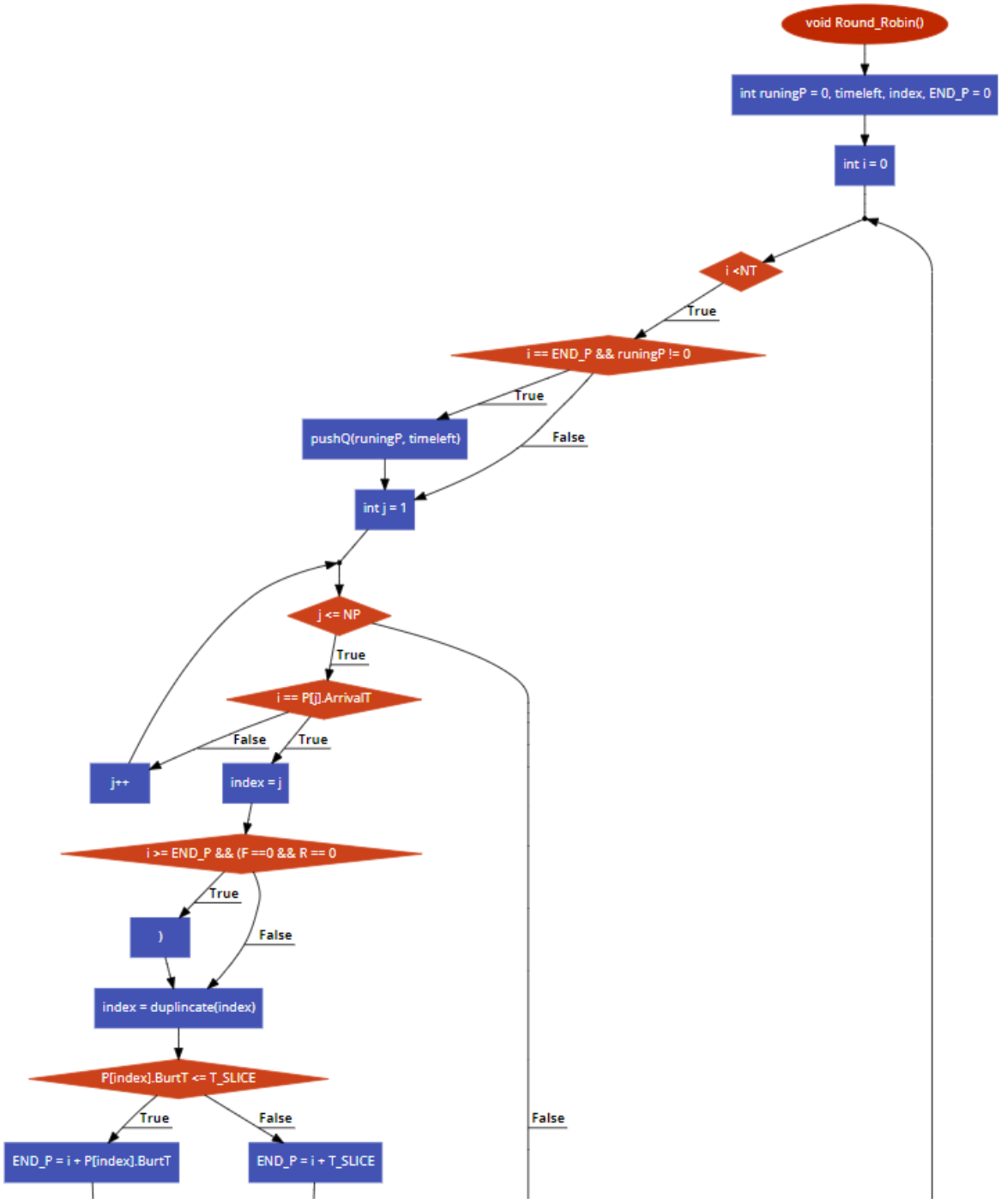
Function Queue sortq

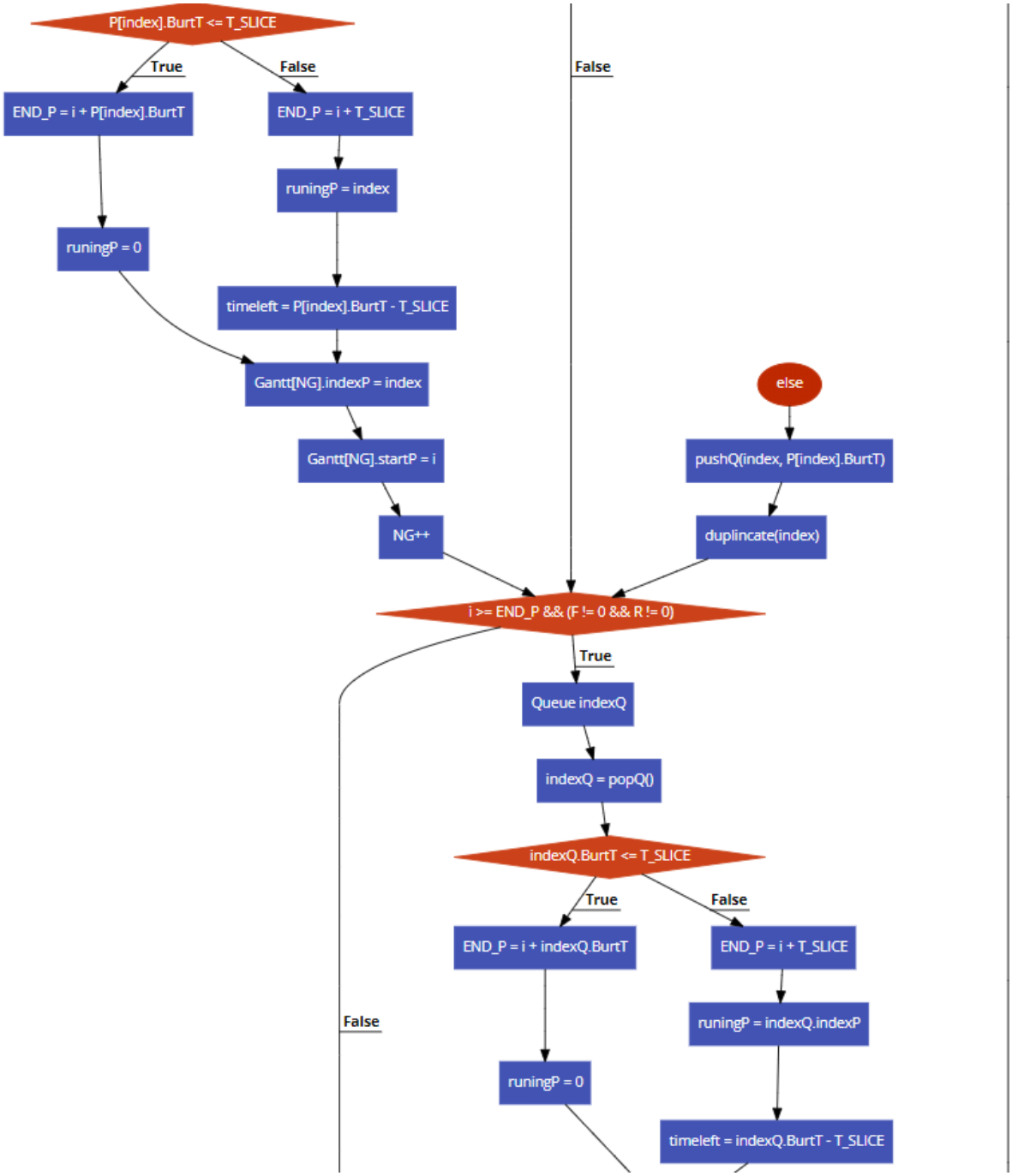


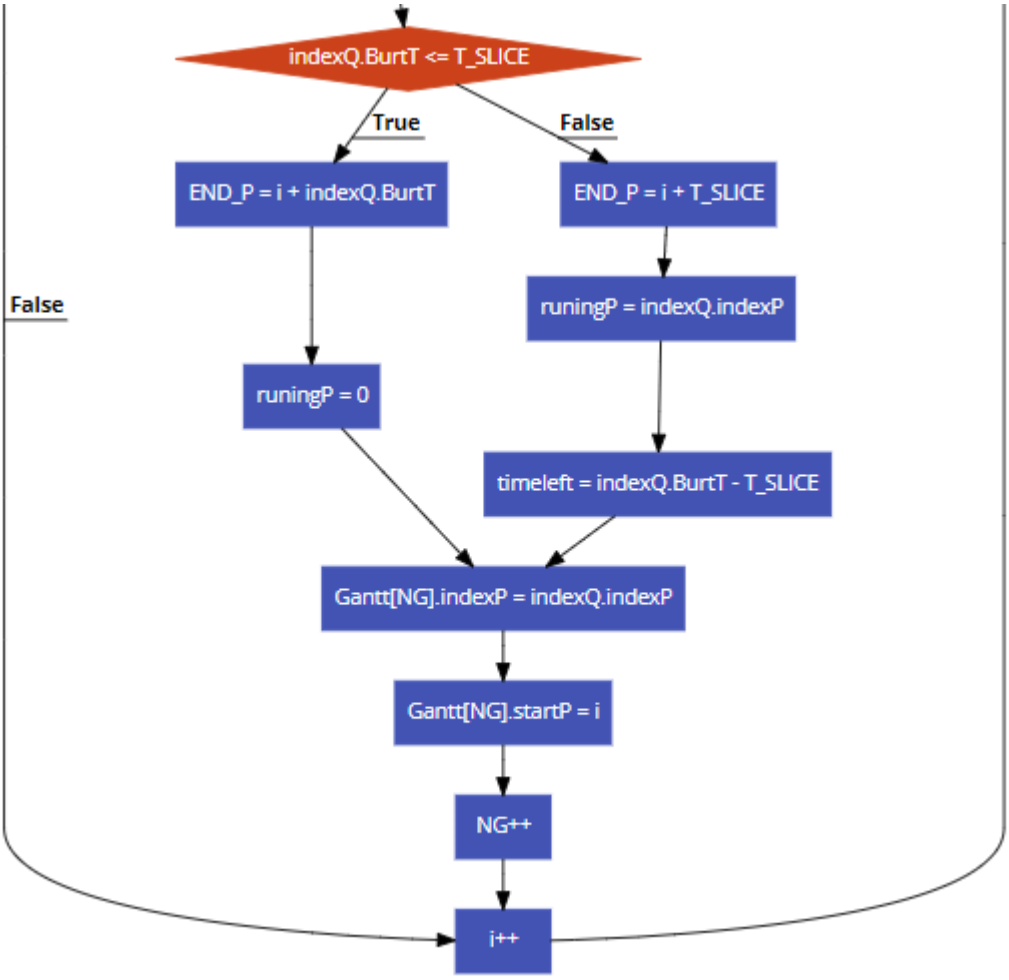
Function duplication



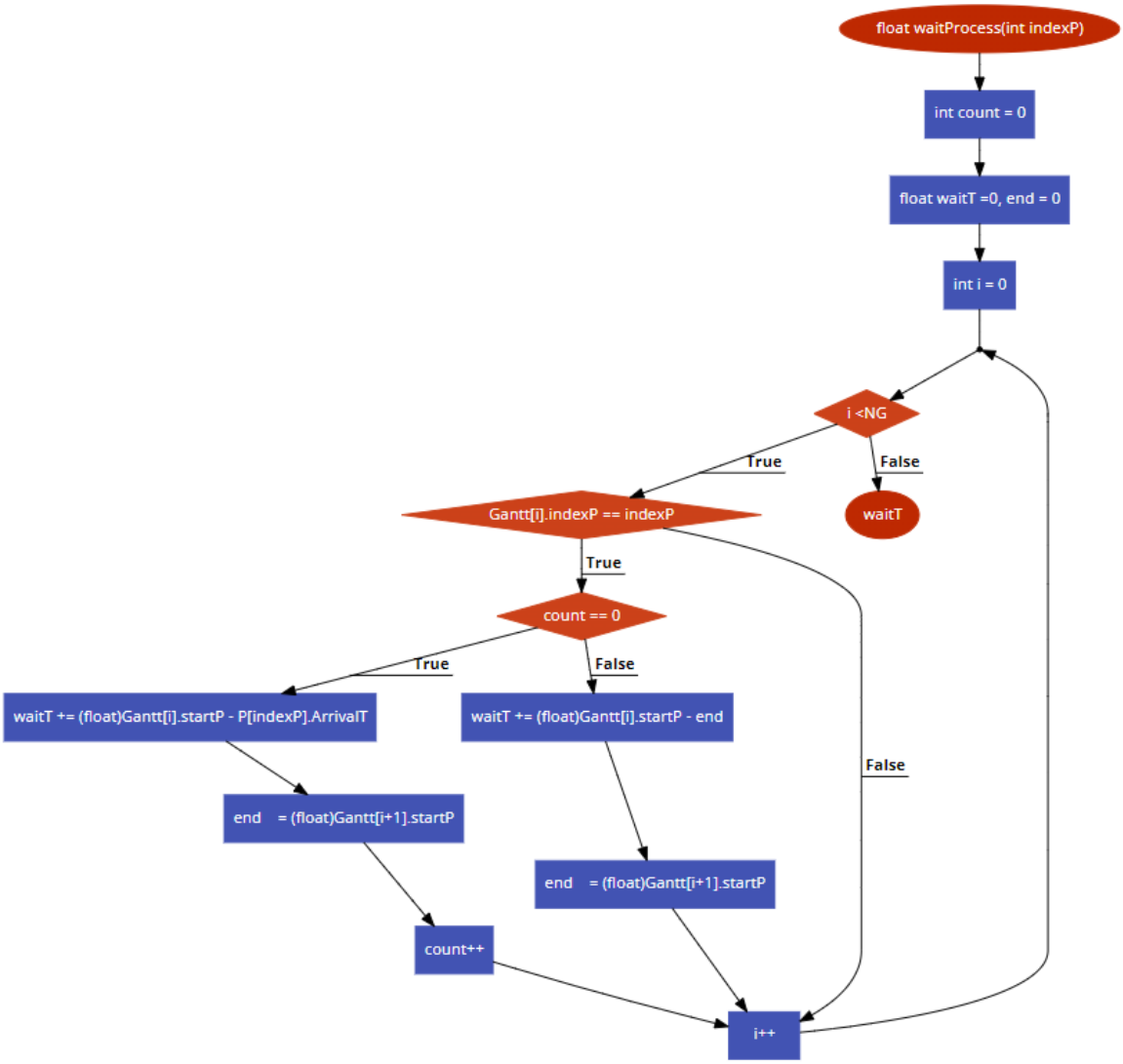
Function Round robin



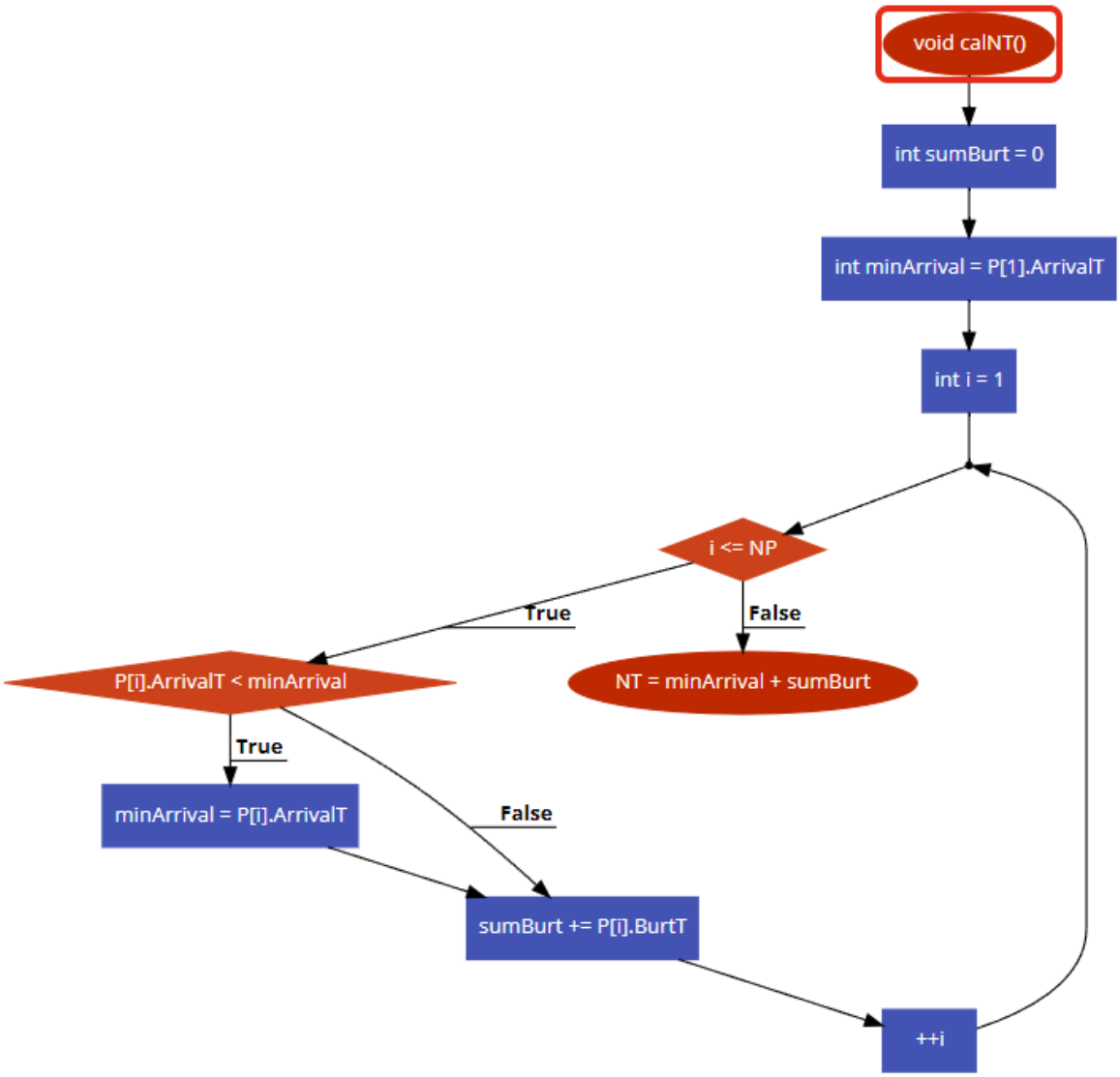

P[index].BurtT <= T_SLICE
True
False
END_P = i + P[index].BurtT
END_P = i + T_SLICE
runingP = index
runingP = 0
timeleft = P[index].BurtT - T_SLICE
Gantt[NG].indexP = index
Gantt[NG].startP = i
NG++
False
else
pushQ(index, P[index].BurtT)
duplincate(index)
i >= END_P && (F != 0 && R != 0)
True
Queue indexQ
indexQ = popQ()
indexQ.BurtT <= T_SLICE
True
False
END_P = i + indexQ.BurtT
END_P = i + T_SLICE
runingP = indexQ.indexP
runingP = 0
timeleft = indexQ.BurtT - T_SLICE
False


indexQ.BurtT <= T_SLICE
True
False
END_P = i + indexQ.BurtT
END_P = i + T_SLICE
False
runingP = indexQ.indexP
runingP = 0
timeleft = indexQ.BurtT - T_SLICE
Gantt[NG].indexP = indexQ.indexP
Gantt[NG].startP = i
NG++
i++

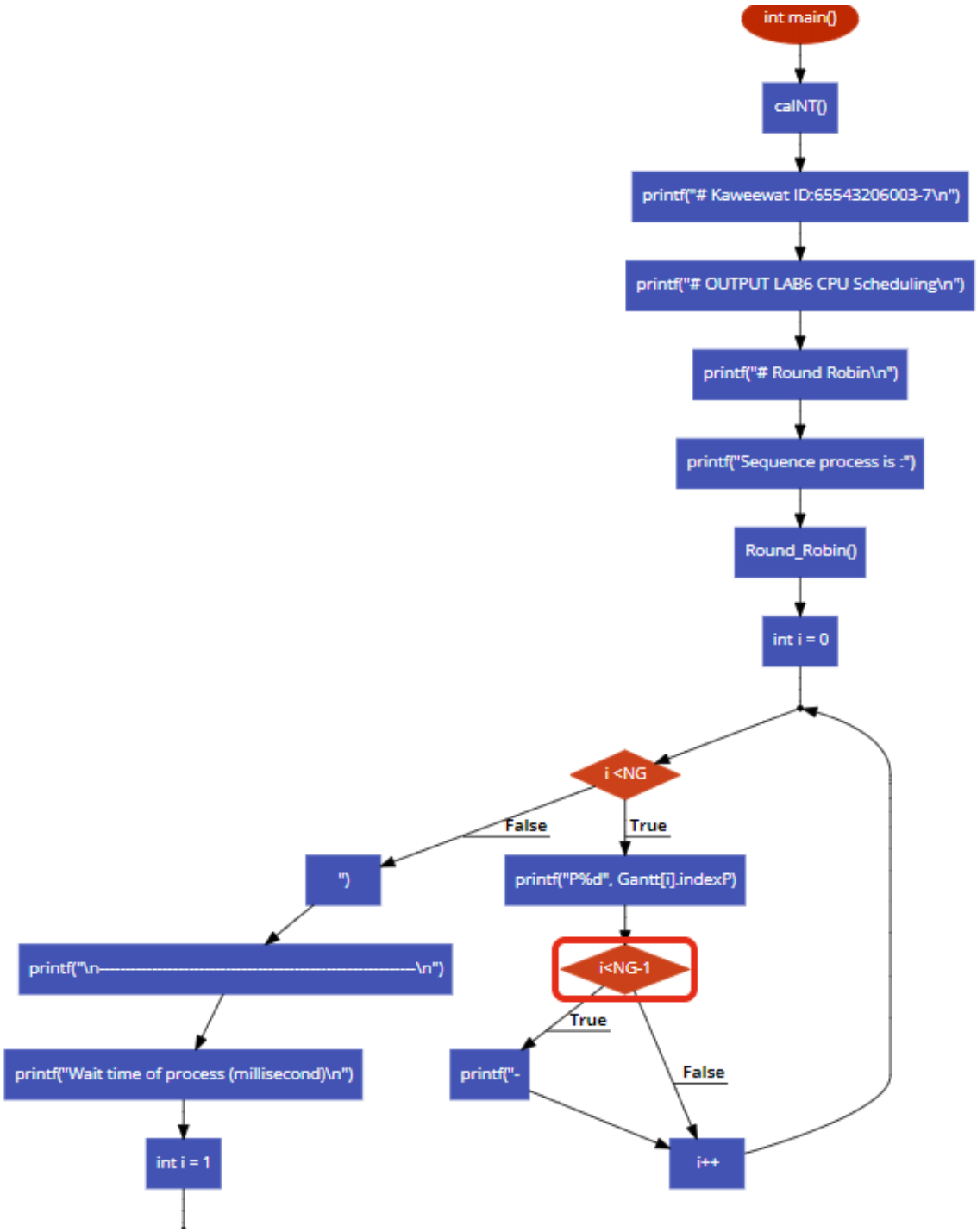
Function WaitProcess

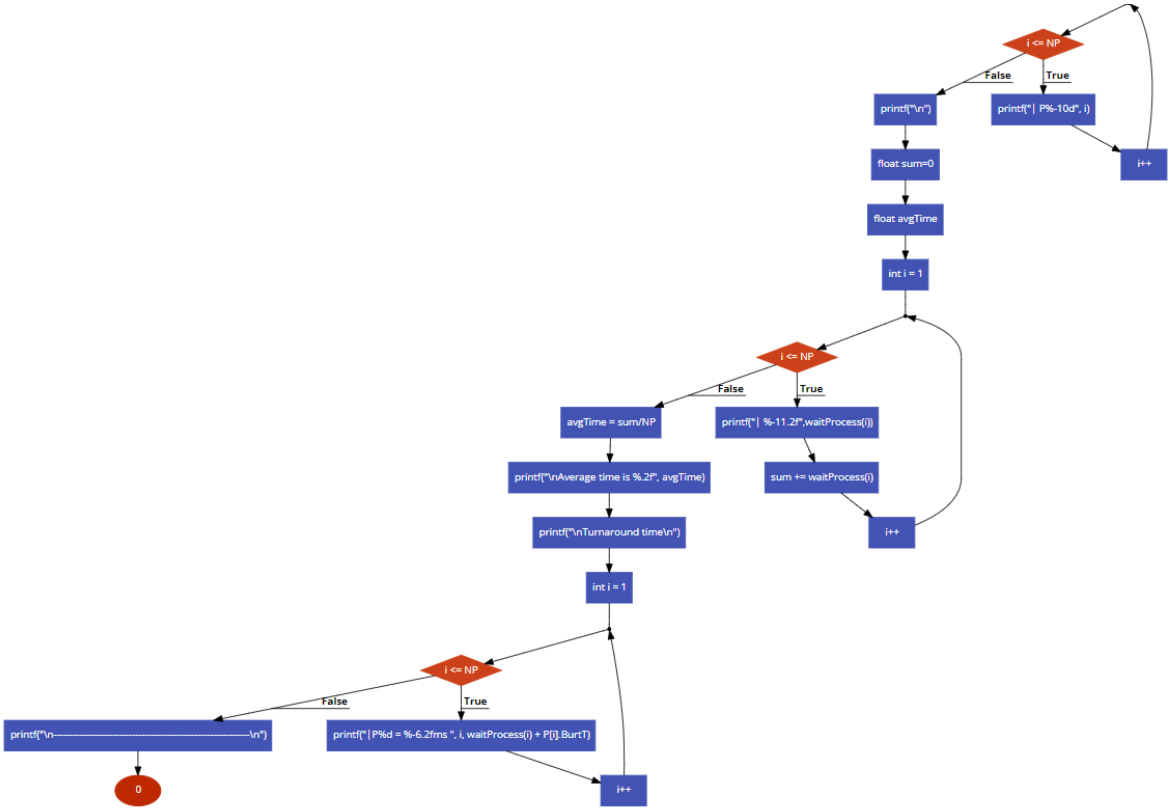


Function CalNT



Main function




i <= NP
False
True
printf("\n")
printf("| P%-10d", i)
i++
float sum=0
float avgTime
int i = 1
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
printf("\n----------\n")
printf("|P%d = %-6.2fms ", i, waitProcess(i) + P[i].BurtT)
0
i++

Code

```c
#include <stdio.h>
#define N 5
#define T_SLICE 4
#define NQ 20

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

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

You, last week | 1 author (You)
typedef struct{
    int indexP;
    int startP;
}Gantt_C;

//Process  burt time , Arrival time , Priority
Process P[N+1]  =  {{0},
        /*P1*/ {    9     ,       1      ,      3},     // P1 = P[1]
        /*P2*/ {    3     ,       1      ,      5},     // P2 = P[2]
        /*P3*/ {    5     ,       3      ,      1},     // P3 = P[3]
        /*P4*/ {    4     ,       4      ,      4},     // P4 = P[4]
        /*P5*/ {    2     ,       7      ,      2}};    // P5 = P[5]
```

1. Gantt_c (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

```
Gantt_C Gantt[20];
Queue   Q[NQ];
int F = 0, R = 0;
int NG = 0;
int NT = 0;
int NP = N;

void pushQ(int index, int BTimeLeft){
    if (R == F-1  || (F == 1 && R == NQ-1)) {
        printf("OVER FLOW!!\n");
    }else{
        if(R == NQ-1){
            R = 1;
        }else{
            R++;
            if(F == 0)
                F = 1;
        }
        Q[R].indexP = index;
        Q[R].BurtT  = BTimeLeft;
    }
}
```

4. PushQ

การตรวจสอบค่า quese

การเพิ่มข้อมูล data

กำหนดค่า index

```
Queue popQ() {
    Queue index;
    if (F == 0) {
        printf("Remove a process from the queue.\n");
    } else {
        index = Q[F];
        if (F == R) {
            F = 0; R = 0;
        } else {
            if (F == NQ-1)
                F = 1;
            else
                F++;
        }
        return index;
    }
}

int func_FCFS(int index){
    int temp[20], n = 0;
    for (int i = 1; i <= NP; i++) {
        if(P[index].ArrivalT == P[i].ArrivalT){
            temp[n++] = i;
        }
    }
    for (int j = 0; j < n; j++) {
        if(temp[j] != index){
            pushQ(temp[j],P[temp[j]].BurtT);
        }       You, last week • updata 16 …
    }
    return index;
}
```

5. popQ

   ตรวจสอบว่า F มีค่าเป็น0 หรือไม่

   ถ้า f ไม่เป็น 0 กำหนดให้เป็น index

   ตรวจสอบว่าค่า F และ R มีค่าเท่ากันหรือไม่

   คืนค่า index

6. Func_FCFS

   หาค่าArrival ที่มีค่าเท่ากัน มาเก็บไว้ใน temp

   ใช้ for loop เก็บค่าที่วนมา ใน array

   คืนค่าสู่ index

```
int duplincate(int index){
    int count = 0;
    for (int i = 1; i <= NP; i++)
        if(P[index].ArrivalT == P[i].ArrivalT)
            count++;
    if (count > 1)
        return func_FCFS(index);
    else
        return index;
}
```

7. Duplicate

นับจำนวนที่วนรอบ loop และเก็บไว้ในตัวแปร count

ถ้า count มีมากกว่า 1 จะเรียนใช้ function fcfs

ถ้า count น้อยกว่า 1 จะคืนค่า index

```
void Round_Robin(){
    int runingP = 0, timeleft, index, END_P = 0;
    for (int i = 0; i <NT ; i++) {
        if(i == END_P && runingP != 0){
            pushQ(runingP, timeleft);
        }
        for (int j = 1; j <= NP; j++) {
            if (i == P[j].ArrivalT) {
                index = j;
                if (i >= END_P && (F ==0 && R == 0 )) {
                    index = duplincate(index);
                    if(P[index].BurtT <= T_SLICE) {
                        END_P = i + P[index].BurtT;
                        runingP = 0;
                    }
                    else {
                        END_P = i + T_SLICE;
                        runingP = index;
                        timeleft = P[index].BurtT - T_SLICE;
                    }
                    Gantt[NG].indexP = index;
                    Gantt[NG].startP = i;
                    NG++;
                    break;
                } else{
                    pushQ(index, P[index].BurtT);
                    duplincate(index);
                    break;
                }
            }
        }
        if (i >= END_P && (F != 0 && R != 0)) {
            Queue indexQ;
            indexQ = popQ();

            if (indexQ.BurtT <= T_SLICE) {
                END_P = i + indexQ.BurtT;
                runingP = 0;
            } else {
                END_P = i + T_SLICE;
                runingP = indexQ.indexP;;
                timeleft = indexQ.BurtT - T_SLICE;
            }
            Gantt[NG].indexP = indexQ.indexP;
            Gantt[NG].startP = i;
            NG++;
        }
    }
```

8. Round Robin

A. ในแต่ละรอบ, ตรวจสอบ Arrival Time ของกระบวนการและเพิ่มลงในคิว.

B. ถ้ามีกระบวนการที่กำลังทำงานและถึงเวลาที่ต้องหยุด, นำกระบวนการนั้นลงในคิว.

C. ตรวจสอบ Arrival Time และเลือกกระบวนการที่มี Arrival Time เท่ากับเวลาปัจจุบัน.

D. ตาม Arrival Time ทำการตรวจสอบเงื่อนไขเพื่อกำหนดเวลาสิ้นสุด (END_P), กระบวนการที่ทำงาน (runingP), และเวลาที่เหลือ (timeleft).

E. เพิ่มข้อมูลลงใน Gantt chart ถ้ากระบวนการทำงานมีการเปลี่ยนแปลง.
F. ถ้าเวลาปัจจุบันถึง END_P และคิวไม่ว่าง, นำกระบวนการจากคิวลงในตัวแปร indexQ.
G. ตรวจสอบเงื่อนไขเพื่อกำหนด END_P, runingP, และ timeleft จากข้อมูลใน indexQ.
H. เพิ่มข้อมูลลงใน Gantt chart ถ้ากระบวนการทำงานมีการเปลี่ยนแปลง.
I. ทำขั้นตอนที่ 1-8 ในทุกรอบวนไปเรื่อยๆ

```
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;
}

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;
}
```

9. wait process

   กำหนดตัวแปลและค่าเริ่มต้น

   ทำวนลูปตาม gantt chart

   คืนค่า wait time

10. CalNT

    กำหนดค่าตัวแปล

    for loop ตรวจสอบหาค่าของ arrival time เก็บค่าไว้ใน minArrival

    ยวก burt Time ของแต่ละกระบวนการเข้าคิง sumBurt

    คำนวนเวลาทั้งหมอของ NT

```c
int main(){
    calNT();
    printf("# Kaweewat ID:65543206003-7\n");
    printf("# OUTPUT LAB6 CPU Scheduling\n");
    printf("# Round Robin\n");
    printf("Sequence process is :");
    Round_Robin();
    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;
    float 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;
}
```

11. Main

ใช้งาน function Round robin และเริ่มการทำงานของ for loop

แสดงค่าของ Sequence process

แสดงหาค่าของ wait time และ avg time และ turnaround time

ผลลัพธ์

```
# Kaweewat ID:65543206003-7
# OUTPUT LAB6 CPU Scheduling
# Round Robin
Sequence process is :P1->P2->P3->P4->P1->P5->P3->P1
------------------------------------------------------------
Wait time of process (millisecond)
| P1          | P2          | P3          | P4          | P5
| 14.00       | 4.00        | 15.00       | 8.00        | 13.00
Average time is 10.80
Turnaround time
|P1 = 23.00 ms |P2 = 7.00  ms |P3 = 20.00 ms |P4 = 12.00 ms |P5 = 15.00 ms
------------------------------------------------------------
```