ใบงานที่ 7

เรื่อง Banker’s Algorithm

เสนอ

อาจารย์ปิยพล ยืนยงสถาวร

จัดทำโดย

นาย กวีวัธน์ กาญจนสุพัฒนากุล 65543206003-7

ใบงานนี้เป็นส่วนหนึ่งของรายวิชา ระบบปฏิบัติการ

หลักสูตรวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา

ประจำภาคที่ 2 ปีการศึกษา 2566

ใบงานที่ 7 : Banker's Algorithm

ให้สร้างโปรแกรมด้วยภาษาซี เพื่อจำลองการทำงานของ Banker's Algorithm ตามทฤษฎีที่เรียนมา พัฒนาบนระบบปฏิบัติการใดก็ได้

- *ส่งไฟล์รูปเล่มใบงาน*
- *บันทึกนำเสนอการใช้งานโปรแกรม และอธิบายส่วนโค้ดโปรแกรมที่เกี่ยวข้อง นำเสนอในมุมมองผู้ใช้งาน และผู้พัฒนาโปรแกรม*

ข้อกำหนดของโปรแกรมเบื้องต้น

1. ให้ออกแบบการรับข้อมูล
    a. จำนวน process ให้ได้อย่างน้อย 5 process
    b. จำนวน resource ให้ได้อย่างน้อย 3 ประเภท
    c. จำนวน resource ทั้งหมดของแต่ละประเภท
    d. จำนวน resource allocation ของแต่ละ process
    e. จำนวน resource max ของแต่ละ process
2. ประมวลผล
    a. ให้โปรแกรมคำนวณหาค่า resource available เริ่มต้น
    b. ให้โปรแกรมคำนวณหาค่า resource need
    c. หาค่า resource available หลังจากที่ process ทำงานเสร็จของแต่ละ process
    d. หา Sequences state (ลำดับการทำงานของ process)
3. แสดงผล
    a. รูปแบบตารางคล้ายตัวอย่าง แบ่งเป็นแถว และคอลัมน์
    b. ค่าต่างๆ ที่ได้จากการรับค่า และการประมวลผล
    c. ลำดับการทำงานทีละ process พร้อมค่า resource available หลังงานการทำงานเสร็จของแต่ละ process เพื่อใช้ตรวจสอบในการทำงาน process ถัดไป
    d. แสดงสถานะของ deadlock (safe หรือ unsafe)
    e. ลำดับการทำงานของ process (Sequences state)
    f. จำนวน resource ทั้งหมดที่ release หลังจากกระบวนการทำงานเสร็จสิ้นทั้งหมด เทียบกับจำนวน resource ที่รับเข้ามาทั้งหมด

ขั้นตอนการทดลอง

1. ออกแบบโปรแกรม ด้วยวิธีการที่ได้ศึกษามา เช่น การเขียนโฟลชาร์ต
2. เขียนโปรแกรม และอธิบายโค้ดฟังก์ชันการทำงานหลัก
3. ตารางการกำหนดตัวแปรหลักที่ใช้ในโค้ดโปรแกรม (ชื่อตัวแปร ชนิด เก็บข้อมูลอะไร ทำหน้าที่อะไร)
4. บันทึกขั้นตอนการทดลอง ทุกขั้นตอน (ปริ้นรูปประกอบ แล้วเขียนอธิบาย)
5. สรุปผลการทดลอง (ปริ้นรูปประกอบ แล้วเขียนอธิบาย)

โดยมีตัวอย่างการป้อนข้อมูลในรูปแบบต่างๆ ที่ใช้ในการทดสอบ 3 กรณี ดังนี้

Example 1 : Banker's Algorithm

3 resource is A(12 instances), B(16 instances), C(13 instances) drives
3 process P1, P2, P3 and P4

Process	Alloc	Max	Need	Avail	State
	A B C	A B C	A B C	A B C	
P1	1 0 1	4 0 1	3 0 0	12 16 13	4
P2	4 5 2	6 7 3	2 2 1	6 7 4	1
P3	5 3 5	8 3 5	3 0 0	11 10 9	2
P4	0 6 3	7 6 5	7 0 2	11 16 12	3

Available drives = (2,2,2)
First state allocate (2,2,1) drives to P2
P2 completes and releases (6,7,4) Available drives
Next state ...
Sequences state are P2,P3,P4,P1
**ALL process run to completion = SAFE STATE

Process	Alloc	Max	Need	Avail	State
	A B C	A B C	A B C	A B C	
P1	8 5 3	10 9 5	2 4 2		NULL
P2	2 6 5	5 7 7	3 1 2		NULL
P3	1 3 2	3 6 9	2 3 7		NULL
P4	0 1 3	6 3 5	6 2 2		NULL

Available = (1,1,0)
If grant last drive to any process may get deadlock = UNSAFE STATE
**ALL process don't run to completion

Process	Alloc	Max	Need	Avail	State
	A B C	A B C	A B C	A B C	
P1	8 5 3	10 9 5	2 4 2	10 9 5	1
P2	1 3 5	12 7 7	11 4 2		NULL
P3	1 3 0	3 6 9	2 3 9		NULL
P4	0 1 3	8 13 5	8 12 2		NULL

Available = (2,4,2)
First state allocate (2,4,2) drives to P1
P1 completes and releases (10,9,5) Available drives
Next state ... Not enough for process needs remaining (P2,P3,P4)
Process may get deadlock = UNSAFE STATE

Example 2 : Banker's Algorithm

```
->> Input state
Enter number of process : 3
Enter number of resource : 3
-----------------------------------------------------------
Enter all unit of resource :
resource 01 : 8 resource 02 : 4 resource 03 : 4
-----------------------------------------------------------
Enter allocation of process : 1
resource 01 : 0 resource 02 : 1 resource 03 : 0
Enter allocation of process : 2
resource 01 : 3 resource 02 : 0 resource 03 : 2
Enter allocation of process : 3
resource 01 : 3 resource 02 : 0 resource 03 : 2
-----------------------------------------------------------
Enter max of process : 1
resource 01 : 7 resource 02 : 4 resource 03 : 3
Enter max of process : 2
resource 01 : 3 resource 02 : 2 resource 03 : 2
Enter max of process : 3
resource 01 : 8 resource 02 : 0 resource 03 : 2
-----------------------------------------------------------
Enter available number of resource :
resource 01 : 2 resource 02 : 3 resource 03 : 0
-----------------------------------------------------------
->> Output state
All resource is 01( 8 instant ), 02( 4 instant), 03( 4 instant )
-------------------------------------------------------------------------------------------------------------
            Allocation    Max           Need          Available
Process     01 02 03      01 02 03      01 02 03      01 02 03
-------------------------------------------------------------------------------------------------------------
P1          3  0  2       3  2  2       0  2  0       2  3  0
P2          3  0  2       8  0  2       5  0  0       5  3  2
P0          0  1  0       7  4  3       7  3  3       8  3  4
                                                      8  4  4
-------------------------------------------------------------------------------------------------------------
Process Sequence : P1, P2, P0
Deadlock is SAFE
-------------------------------------------------------------------------------------------------------------
```

Code

```c
#include <stdio.h>

int np,nr;
int Allocation[10][5],Max[10][5],Need[10][5],Available[10][5];
int resource[5],state[10],AvailDrives[5],AvailSUM[5];
int returnResources = 1;
```

Np : จำนวนของกระบวนการ (Number of processes)

Nr : จำนวนของประเภททรัพยากร (Number of resource types)

Allocation, Max, Need, Available: 2D arrays : ที่ใช้แทนการจัดสรร, ความต้องการสูงสุด, ความต้องการที่เหลือ, และทรัพยากรที่พร้อมใช้งานตามลำดับ, สำหรับแต่ละกระบวนการและประเภททรัพยากร

Resource :1D array ที่แทนจำนวนรวมของแต่ละประเภททรัพยากรในระบบ

State : Array เพื่อเก็บสถานะของแต่ละกระบวนการ

AvailDrives : Array เพื่อเก็บทรัพยากรที่พร้อมใช้งานสำหรับแต่ละประเภททรัพยากรหลังจากจัดสรรทรัพยากรให้แก่กระบวนการ

AvailSUM : Array เพื่อเก็บผลรวมทรัพยากรที่พร้อมใช้งานหลังจากประมวลผลแต่ละกระบวนการ

```
void calAvailable(){
    for (int i = 0; i < nr; ++i) {
        for (int j = 0; j < np; ++j) {
            AvailDrives[i] += Allocation[j][i];
        }
        AvailDrives[i] = resource[i] - AvailDrives[i];
        AvailSUM[i] = AvailDrives[i];
    }
}
```

Function calAvailable():

คำนวณทรัพยากรที่พร้อมใช้งานสำหรับแต่ละประเภททรัพยากรหลังจากจัดสรรทรัพยากรให้แก่กระบวนการ วนลูปตามแต่ละประเภททรัพยากรและลบทรัพยากรที่จัดสรรไปจากทรัพยากรทั้งหมดเพื่อให้ได้ทรัพยากรที่พร้อมใช้งาน
ผลลัพธ์ถูกเก็บไว้ในอาร์เรย์ AvailDrives และ AvailSUM

```
void calNeed(){
    for (int i = 0; i < np; ++i) {
        for (int j = 0; j < nr; ++j) {
            Need[i][j] = Max[i][j] - Allocation[i][j];
        }
    }
}
```

Function calNeed():

คำนวณความต้องการที่เหลือสำหรับแต่ละกระบวนการและประเภททรัพยากร
ลบการจัดสรรปัจจุบันจากความต้องการสูงสุดสำหรับแต่ละกระบวนการและประเภททรัพยากร และเก็บผลลัพธ์ในอาร์เรย์ Need

```
void availSum(int i){
    for (int j = 0; j < nr; ++j) {
        Available[i][j] = Allocation[i][j] + AvailSUM[j];
        AvailSUM[j] = Available[i][j];
    }
}
```

ฟังก์ชัน availSum(int i):

คำนวณทรัพยากรที่พร้อมใช้งานหลังจากประมวลผลกระบวนการที่กำหนด (i)
บวกทรัพยากรที่จัดสรรของกระบวนการนี้ไปยังทรัพยากรที่พร้อมใช้งานสะสมสำหรับแต่ละประเภททรัพยากร
และอัปเดตอาร์เรย์ AvailSUM

```
void Banker(){
    int Level = 0;
    for (int round = 1;returnResources == 1; ++round) {
        returnResources = 0;
        for (int i = 0; i < np; ++i) {
            if(round == 1 || (round  >1 && state[i] == -1)) {

                for (int j = 0; j < nr; ++j) {
                    Available[i][j] =-1;
                    if (AvailSUM[j] >= Need[i][j]) {
                        if (j == nr - 1) {
                            availSum(i);
                            state[i] = Level+=1;
                            returnResources = 1;
                        }
                    } else {
                        state[i] = -1;
                        Available[i][j] =-1;
                        break;
                    }
                }
            }
        }
    }
}
```

Function banker();

1.ตัวแปร

Level: ระดับของกระบวนการ (process level)

round: รอบการทำงานของอัลกอริทึม

returnResources: ตัวแปรที่ใช้เพื่อกำหนดว่ามีการคืนทรัพยากรหรือไม่ (ถ้ามีการคืนจะเป็น 1, ไม่มีการคืนจะเป็น 0)

2.ลูป for (int round = 1; returnResources == 1; ++round):

ในแต่ละรอบของการทำงาน, โค้ดจะทำงานไปเรื่อยๆ จนกว่าจะไม่มีการคืนทรัพยากร (returnResources เป็น 0)

ตั้งค่า returnResources เป็น 0 แต่ถ้ามีการคืนทรัพยากร, จะเปลี่ยนเป็น 1 และทำรอบถัดไป

ลูป for (int i = 0; i < np; ++i):

3.วนลูปทุกรายการของกระบวนการ

ตรวจสอบถ้าเป็นรอบแรกหรือ (รอบมากกว่า 1 และ state[i] เป็น -1) ให้ทำงานด้านในลูป

4.ลูป for (int j = 0; j < nr; ++j):

วนลูปทุกรายการของทรัพยากร

ตั้งค่า Available[i][j] เป็น -1

ตรวจสอบว่า AvailSUM[j] มีค่ามากกว่าหรือเท่ากับ Need[i][j]

ถ้าเป็นเช่น ก็ทำงานในลูปต่อไป

ถ้า j เท่ากับ nr - 1 (เป็นทรัพยากรสุดท้าย)

เรียกใช้ฟังก์ชัน availSum(i) เพื่อคำนวณทรัพยากรที่พร้อมใช้งาน

ตั้งค่า state[i] เป็น Level+=1 (เพิ่มระดับ)

ตั้งค่า returnResources เป็น 1 (มีการคืนทรัพยากร)

5.ลูปภายนอก for (int round = 1; returnResources == 1; ++round):

จะทำงานเรื่อยๆ จนกว่า returnResources จะไม่เป็น 1

```c
void input(){
    printf("->> Input state\n");
    printf("Enter number of process :");
    scanf("%d", &np);
    printf("Enter number of resource (instances) :");
    scanf("%d", &nr);
    printf("-----------------------------------------------------\n");
    printf("Enter all unit of resource :\n");
    for (int i = 0; i < nr; ++i) {
        printf(" resource P%d (instances A,B) : ", i + 1);
        scanf("%d", &resource[i]);
    }
    printf("\n-----------------------------------------------------");
```

Function input()

1. แสดงข้อความ "Input state" บนหน้าจอ.
2. แสดงข้อความ "Enter number of process :" เพื่อรอรับค่าจำนวนของกระบวนการ (number of processes) จากผู้ใช้.
3. รับค่าจำนวนกระบวนการที่ผู้ใช้ป้อนผ่านทางคีย์บอร์ดและเก็บไว้ในตัวแปร np (number of processes).
4. แสดงข้อความ "Enter number of resource (instances) :" เพื่อรอรับค่าจำนวนของทรัพยากร (resource instances) จากผู้ใช้.
5. รับค่าจำนวนทรัพยากรที่ผู้ใช้ป้อนผ่านทางคีย์บอร์ดและเก็บไว้ในตัวแปร nr (number of resource instances).
6. แสดงข้อความ "Enter all unit of resource :" เพื่อให้ผู้ใช้ป้อนจำนวนของแต่ละทรัพยากร.
7. ใช้ลูป for เพื่อรับค่าจำนวนของแต่ละทรัพยากรที่ผู้ใช้ป้อนผ่านทางคีย์บอร์ด.
8. แสดงข้อความ "resource P1 (instances A,B) :"

```
for (int i = 0; i < np; ++i) {      You, 56 minutes ago • update banker
    printf("\nallocation process : %d\n", i + 1);
    for (int j = 0; j < nr; ++j) {
        printf("allocation %c : ", 'a' + j);
        scanf("%d", &Allocation[i][j]);
    }
}


for (int i = 0; i < np; ++i) {
    printf("\nAllocation process %d:\n", i + 1);
    for (int j = 0; j < nr; ++j) {
        printf("allocation %c : %d\n", 'a' + j, Allocation[i][j]);
    }
}
```

Allocation

1. ให้วนรอบ โดยแสดงข้อความ allocation A, allocation B

```
for (int i = 0; i < np; ++i) {
    printf("\nEnter max of process : %d\n", i + 1);
    for (int j = 0; j < nr; ++j) {
        printf("Max %c : ", 'a' + j);
        scanf("%d", &Max[i][j]);
    }
}


for (int i = 0; i < np; ++i) {
    printf("\nMax for process %d:\n", i + 1);
    for (int j = 0; j < nr; ++j) {
        printf("Max %c : %d\n", 'a' + j, Max[i][j]);
    }
}
```

Max

1. ให้วนรอบ โดยแสดงข้อความ Max A, Max B

```
void inTable(int i, int temp[][5]) {
    for (int j = 0; j < nr; ++j) {
        if(temp[i][j] == -1) {
            printf("        ");
            break;
        }
        else
            printf("%2d ", temp[i][j]);
    }
}
```

Function inTable();

1. ทำ For loop ในคอลัม temp [i][j]

```
void showTable(){
    printf("\n=================================================================\n");
    printf("Poscess      Allocation      Max      Need       Avail      State\n          ");
    for (int i = 0; i < 4; ++i){
        printf("   ");
        for (int j = 0; j < nr; ++j)
            printf("%c   ", 65 + j);
    }
    printf("\n");
    for (int i = 0; i < np; ++i){
        printf("P%d\t  ",i+1);
        inTable(i,Allocation);printf("   ");
        inTable(i, Max);printf("   ");
        inTable(i, Need);printf("   ");
        inTable(i, Available);printf("     ");
        if(state[i] == -1)
            printf("NULL");
        else
            printf("%d", state[i]);
        printf("\n");       You, 1 hour ago • update banker
    }
```

Function showTable();

1. แสดงข้อมูล Allocation โดยใช้ฟังก์ชัน inTable(i, Allocation).
2. แสดงข้อมูล Max โดยใช้ฟังก์ชัน inTable(i, Max).
3. แสดงข้อมูล Need โดยใช้ฟังก์ชัน inTable(i, Need).
4. แสดงข้อมูล Available โดยใช้ฟังก์ชัน inTable(i, Available).
5. แสดงสถานะของกระบวนการ (state[i]).

```
void Process(){
    printf("Available drives = ");
    for (int i = 0; i < nr; ++i)
        printf("%d ", AvailDrives[i]);

    int First = 0;
    for (int i = 0; i < np; ++i)
        if(state[i] == 1) {
            First = i + 1;
            break;
        }
    if(First !=0){
        printf("\nFirst state allocate(");
        for (int i = 0; i < nr; ++i)
            printf("%d,", Need[First-1][i]);
        printf(") drives to ");
        printf("P%d\n", First);
        printf("P%d completes and releases(", First);
        for(int i = 0; i<nr; i++)
            printf("%d,",Available[First-1][i]);
        printf(")Available drives\n");
        printf("Next state ...\n");
    }

    int countNULL = 0;
    for (int i = 0; i < np; ++i)
        if(state[i] == -1)
            countNULL +=1;

    if(countNULL == np)
        printf("\nIf grant last drive to any process may get deadlock = UNSAFE STATE\n**ALL process don't run to completion\n");
    else if(countNULL > 0 && countNULL < np) {
        printf("Not enough for process needs remaining(");
        for (int i = 0; i < np; ++i)
            if(state[i] == -1)
                printf("P%d,",i+1);
        printf(")\nProcess may get deadlock = UNSAFE STATE\n");
    }
    else {
        printf("Sequences state are ");      You, 1 hour ago • update banker
        for (int i = 1; i <=np ; ++i)
            for (int j = 0; j <np ; ++j)
                if(i == state[j])
                    printf("P%d ",j+1);
        printf("\n**ALL process run to completion = SAFE STATE\n");
        printf("============================================================\n");
    }
```

Function process();

1. Function จะแสดงจำนวนทรัพยากรณ์ที่ใช้งานได้ในระยย
2. ตรวจสอบสถานะของกระบวนการทำงานทั้งหมเ
3. จากนั้นตรวจสอบสถานะว่าเป็น ssafe state หรือ unsafe

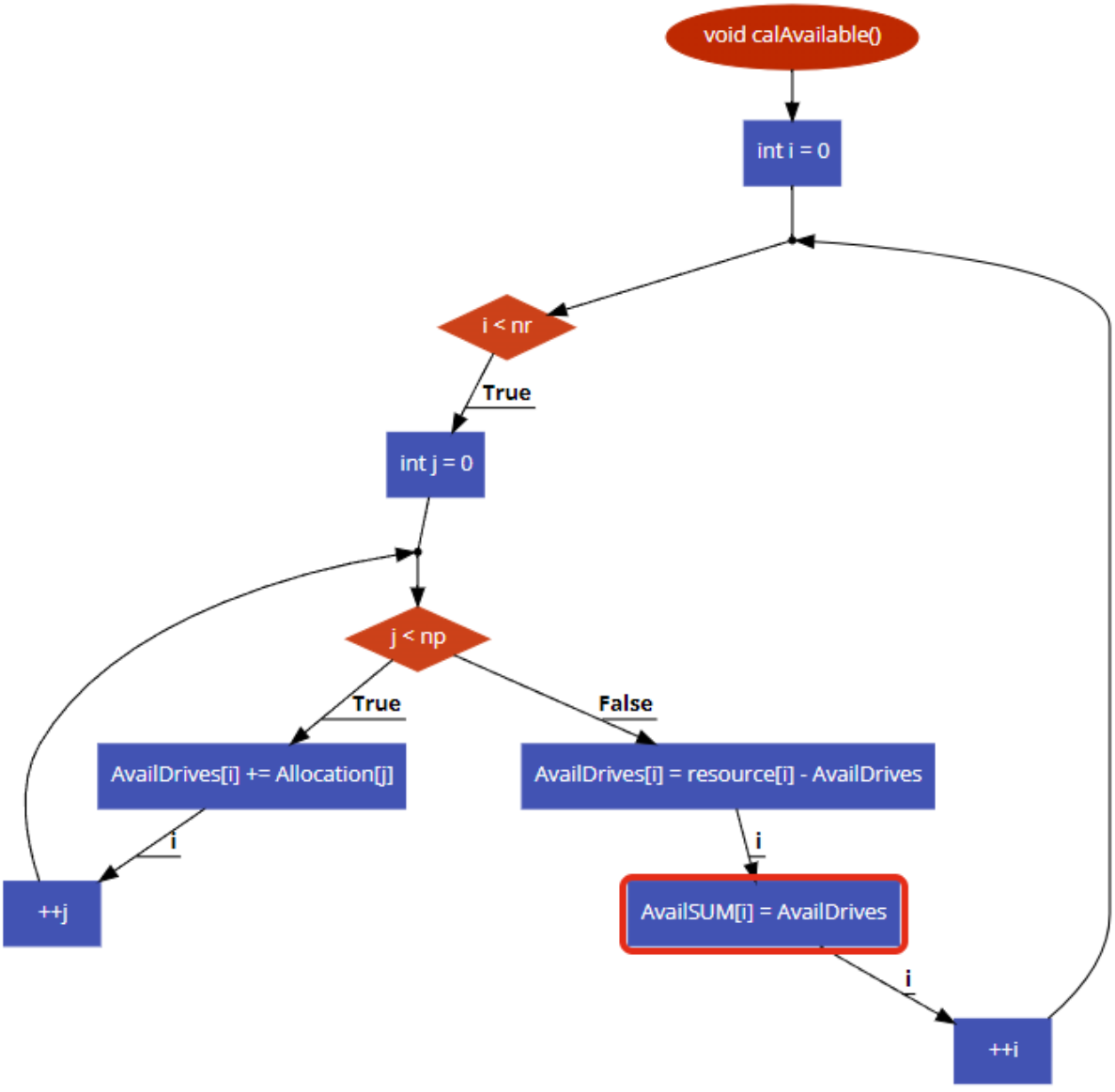
```
int main(){
    input();
    calAvailable();
    calNeed();
    Banker();

    printf("%d process\n", np);
    printf("%d resource is ", nr);
    for (int i = 0; i < nr; ++i)
        printf("%c(%d instances) ", 65 + i, resource[i]);

    showTable();
    Process();
}
```

Function main();

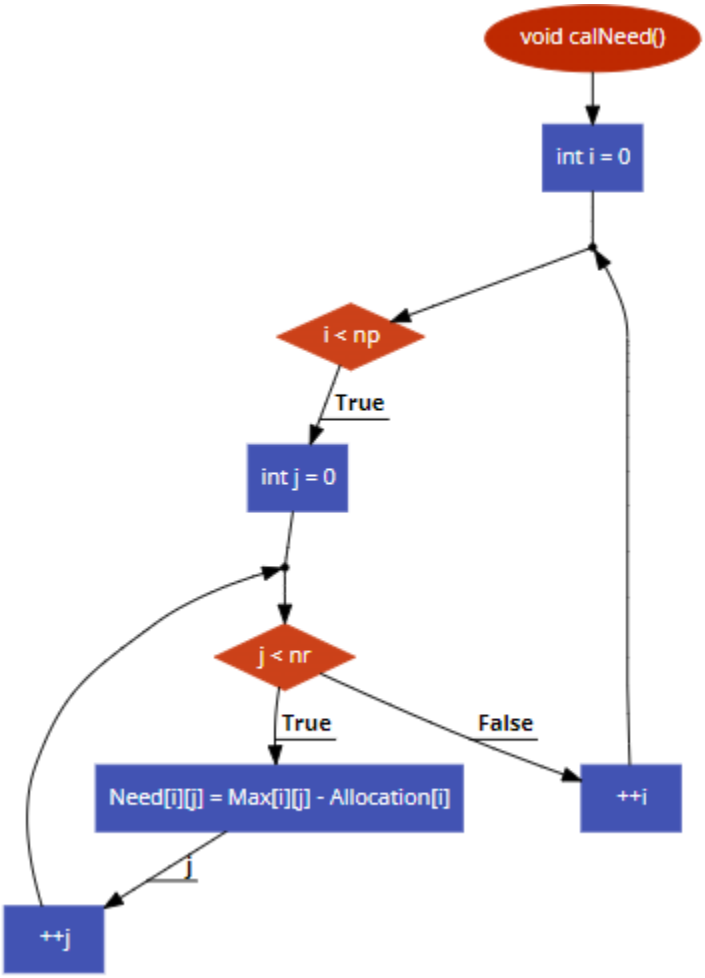
1. เรียกใช้งาน Function input
2. เรียกใช้งาน Function calAvaiable
3. เรียกใช้งาน Function calNeed
4. เรียกใช้งาน Function Banker
5. เรียกใช้งาน Function showtable
6. เรียกใช้งาน Function process

Flowchart

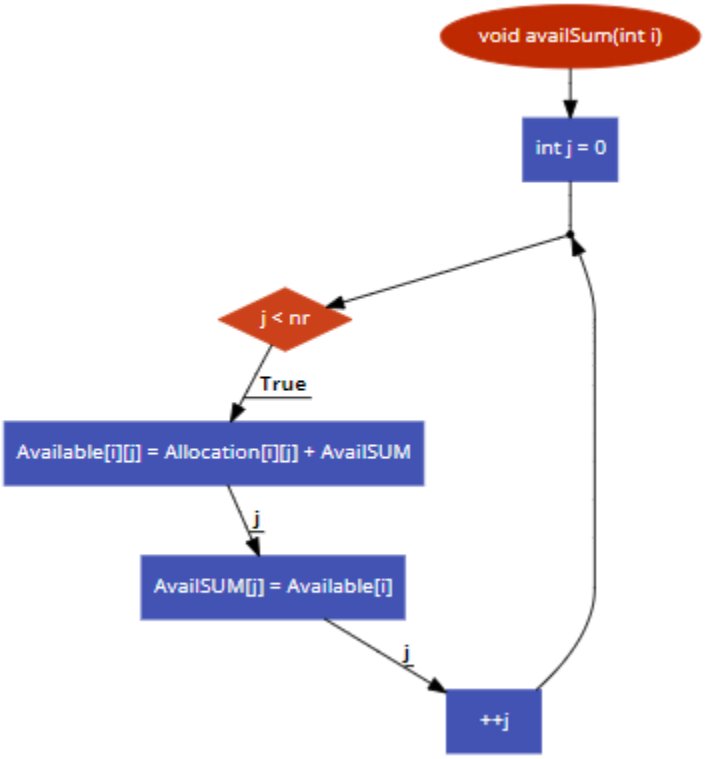
callAvailable



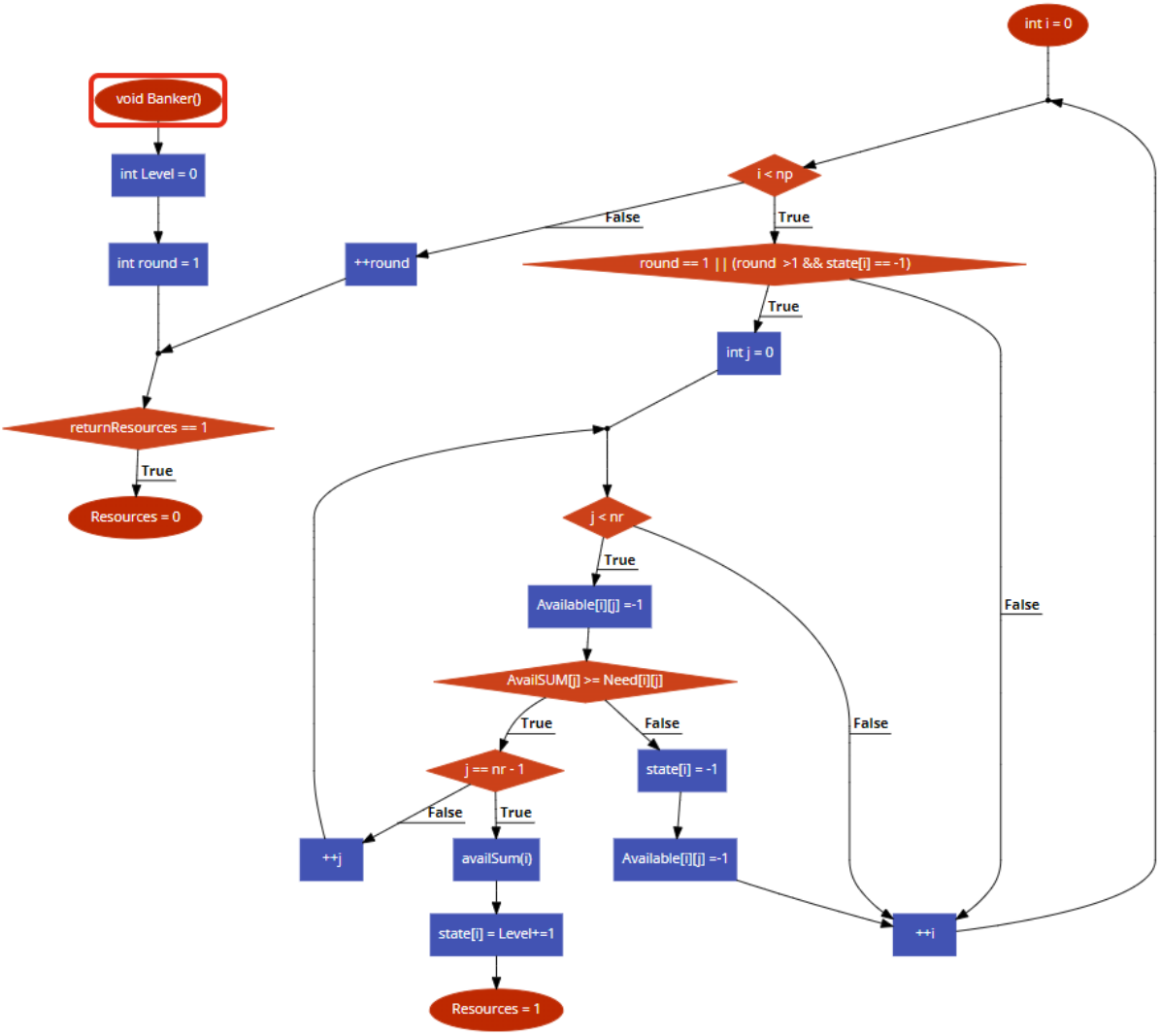
CalNeed


void calNeed()
int i = 0
i < np
True
int j = 0
j < nr
True
False
Need[i][j] = Max[i][j] - Allocation[i]
++i
j
++j

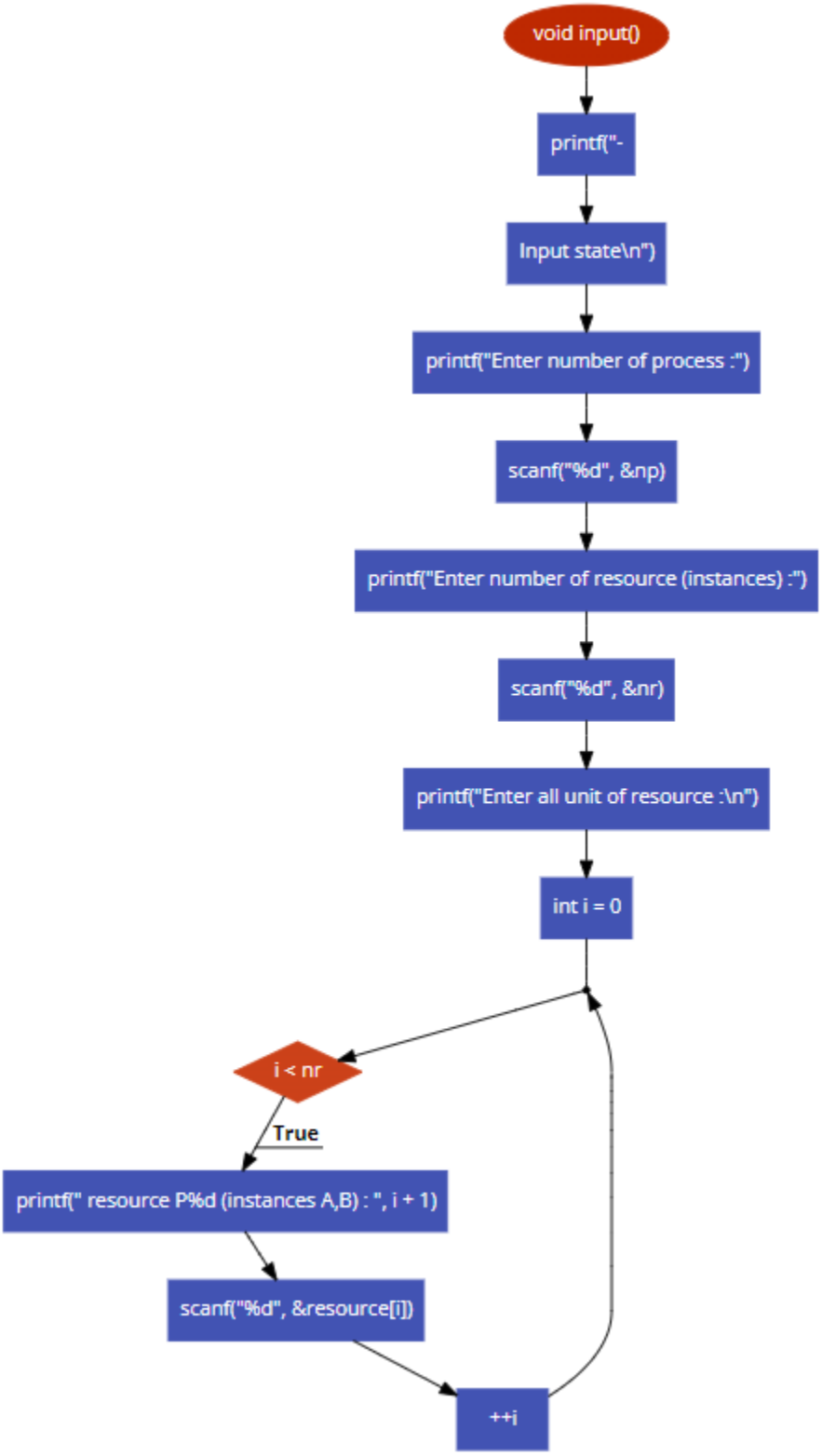
avaliSum


void availSum(int i)
int j = 0
j < nr
True
Available[i][j] = Allocation[i][j] + AvailSUM
j
AvailSUM[j] = Available[i]
j
++j

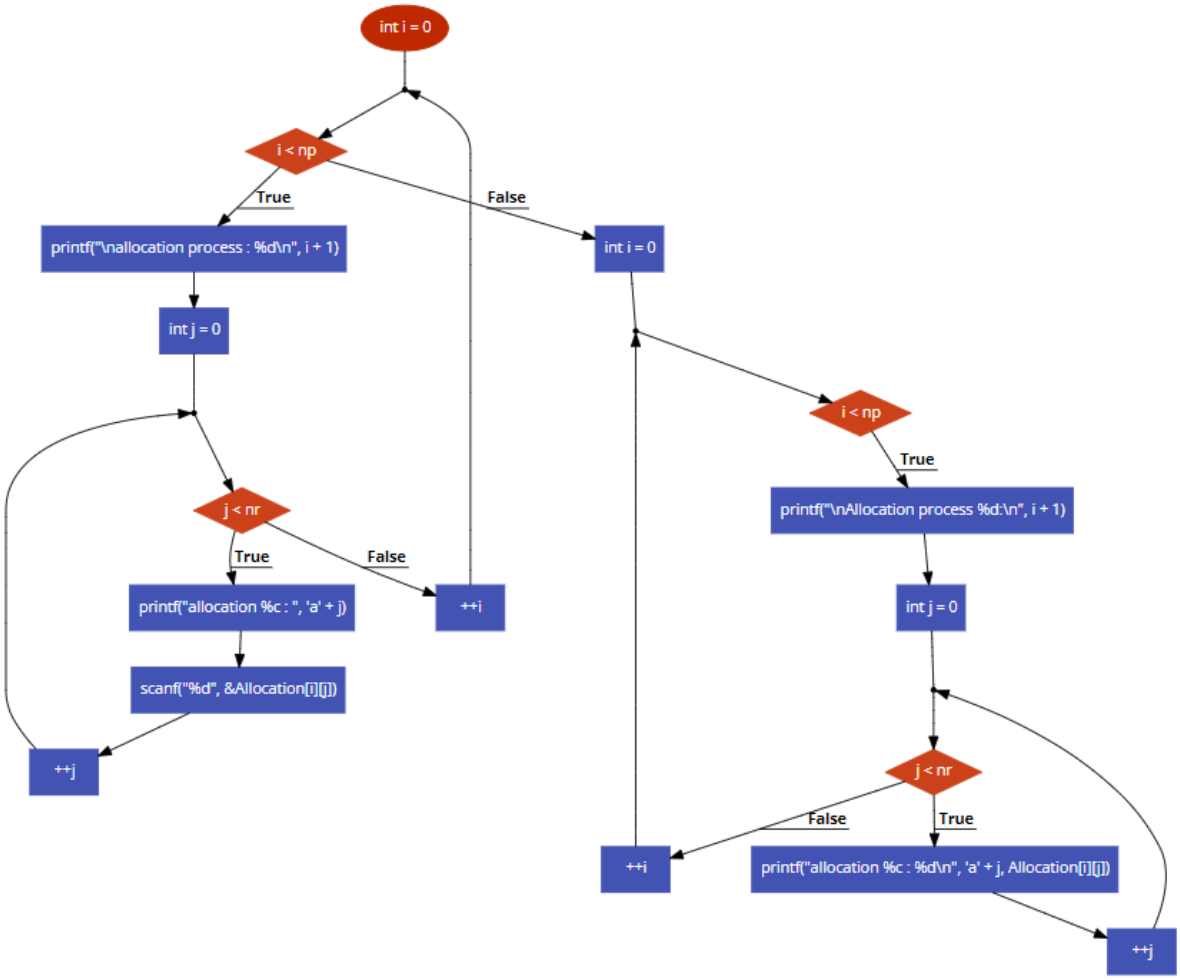
Banker


void Banker()
int Level = 0
int round = 1
returnResources == 1
True
Resources = 0
int i = 0
i < np
False
True
++round
round == 1 || (round >1 && state[i] == -1)
True
int j = 0
j < nr
True
Available[i][j] =-1
AvailSUM[j] >= Need[i][j]
True
False
j == nr - 1
False
True
++j
availSum(i)
state[i] = Level+=1
Resources = 1
state[i] = -1
Available[i][j] =-1
False
False
++i

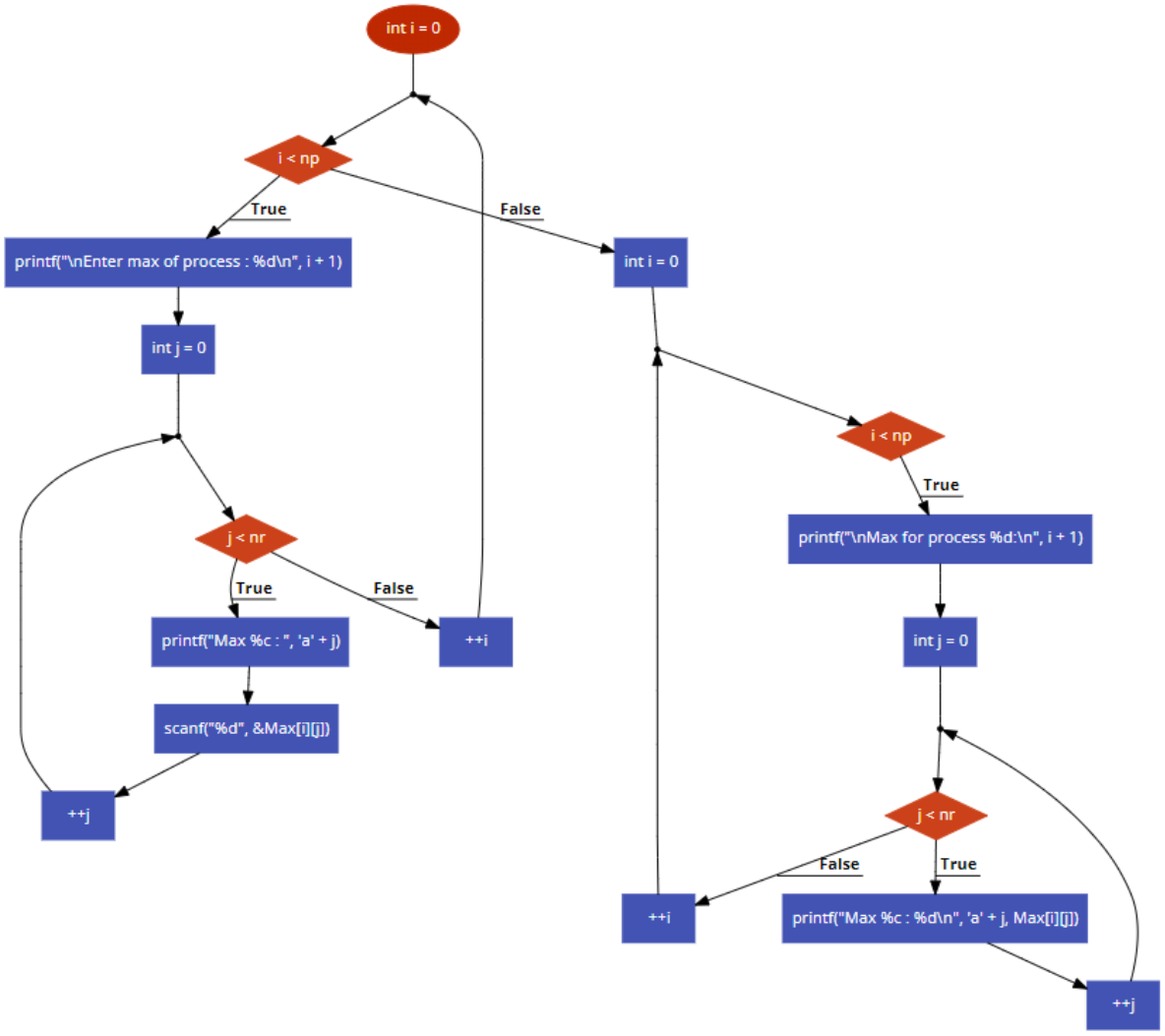
Input


void input()
printf("-
Input state\n")
printf("Enter number of process :")
scanf("%d", &np)
printf("Enter number of resource (instances) :")
scanf("%d", &nr)
printf("Enter all unit of resource :\n")
int i = 0
i < nr
True
printf(" resource P%d (instances A,B) : ", i + 1)
scanf("%d", &resource[i])
++i

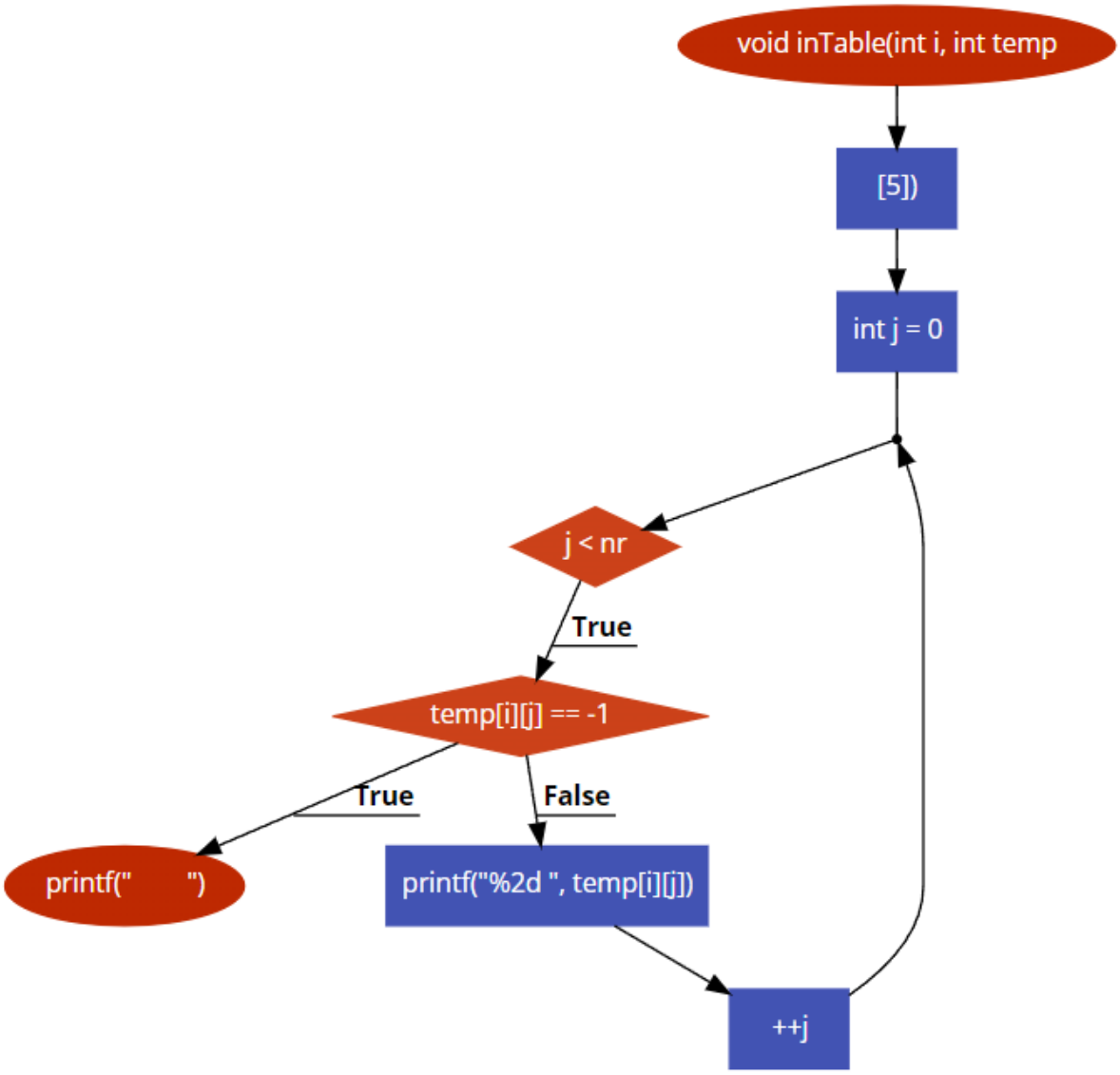
allocation


int i = 0
i < np
True
False
printf("\nallocation process : %d\n", i + 1)
int j = 0
j < nr
True
False
printf("allocation %c : ", 'a' + j)
scanf("%d", &Allocation[i][j])
++j
++i
int i = 0
i < np
True
printf("\nAllocation process %d:\n", i + 1)
int j = 0
j < nr
False
True
printf("allocation %c : %d\n", 'a' + j, Allocation[i][j])
++j
++i

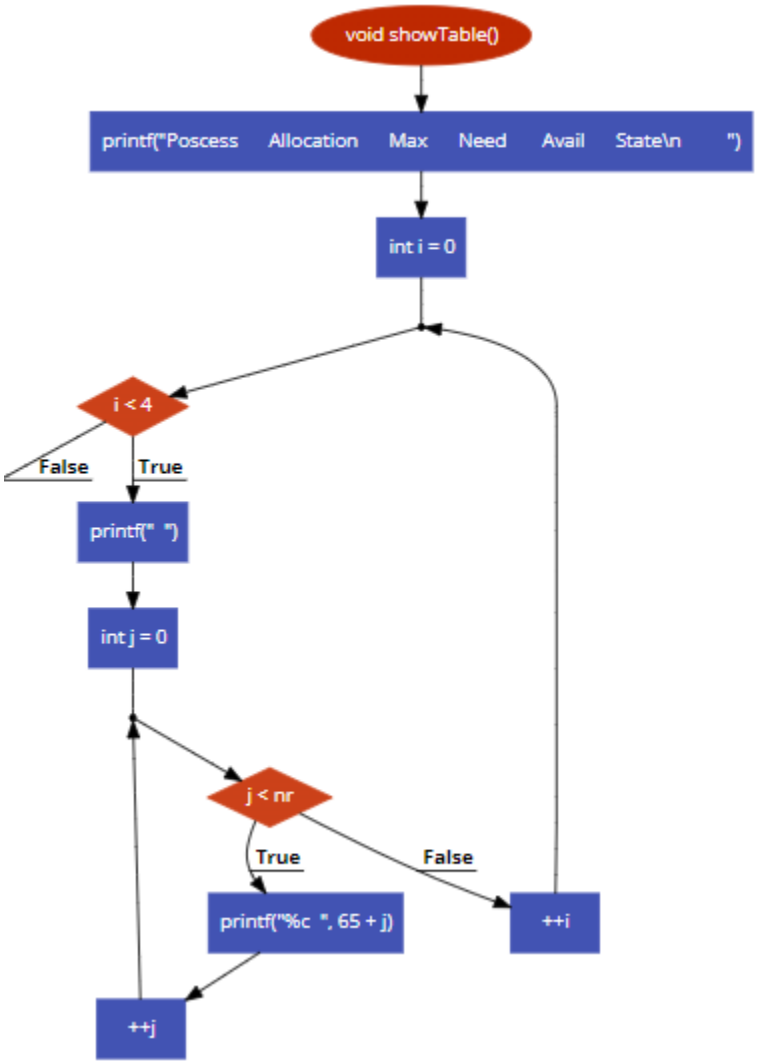
Max

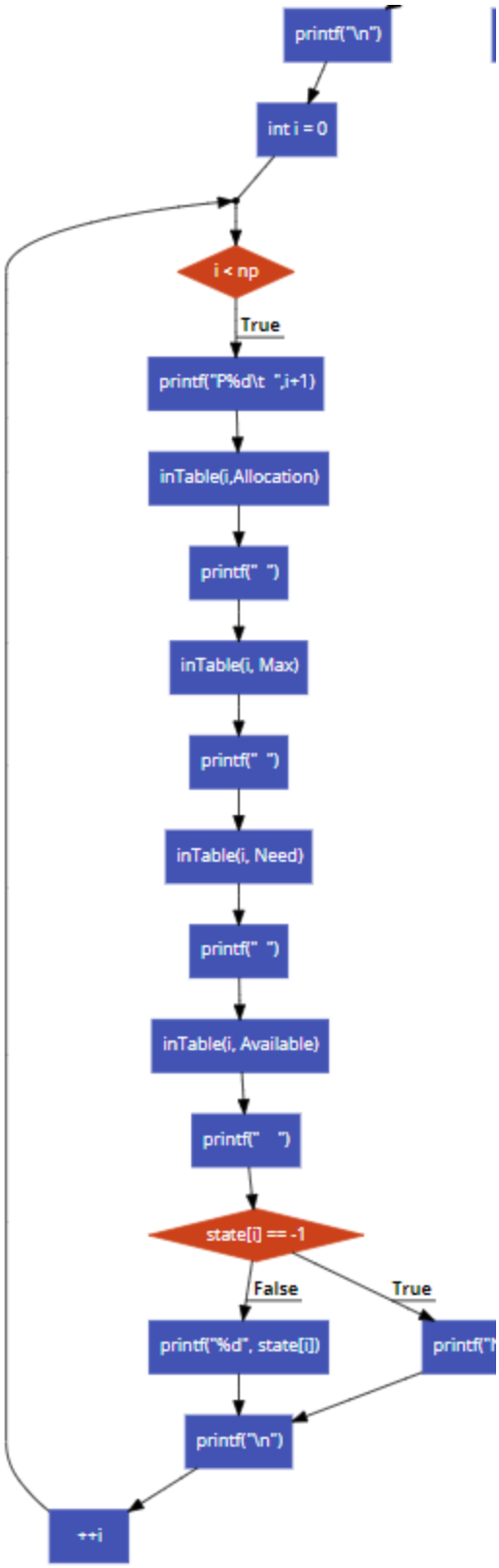

int i = 0
i < np
True
False
printf("\nEnter max of process : %d\n", i + 1)
int j = 0
j < nr
True
False
printf("Max %c : ", 'a' + j)
scanf("%d", &Max[i][j])
++j
++i
int i = 0
i < np
True
printf("\nMax for process %d:\n", i + 1)
int j = 0
j < nr
False
True
++i
printf("Max %c : %d\n", 'a' + j, Max[i][j])
++j

Intable

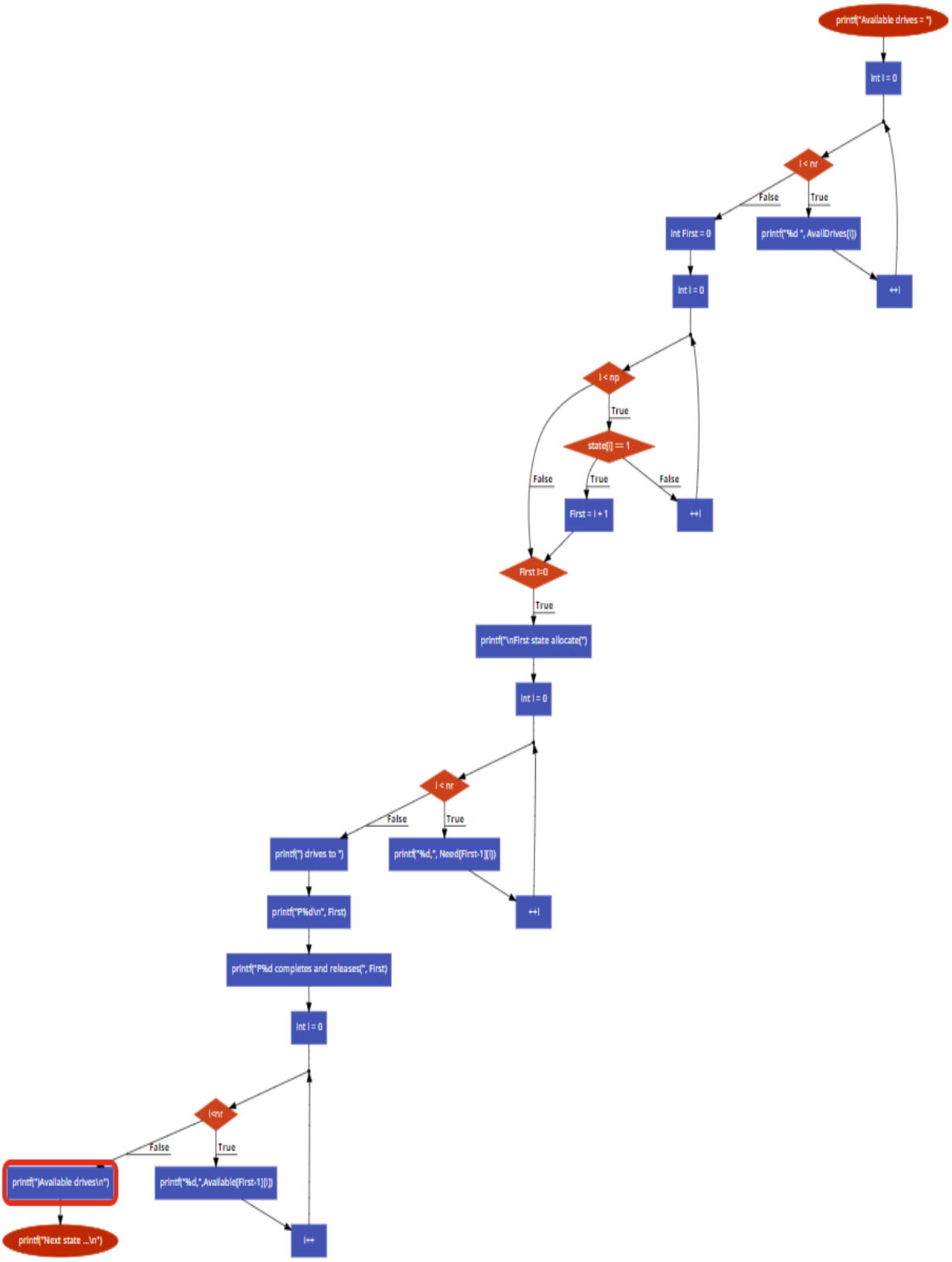

void inTable(int i, int temp
[5])
int j = 0
j < nr
True
temp[i][j] == -1
True
False
printf(" ")
printf("%2d ", temp[i][j])
++j

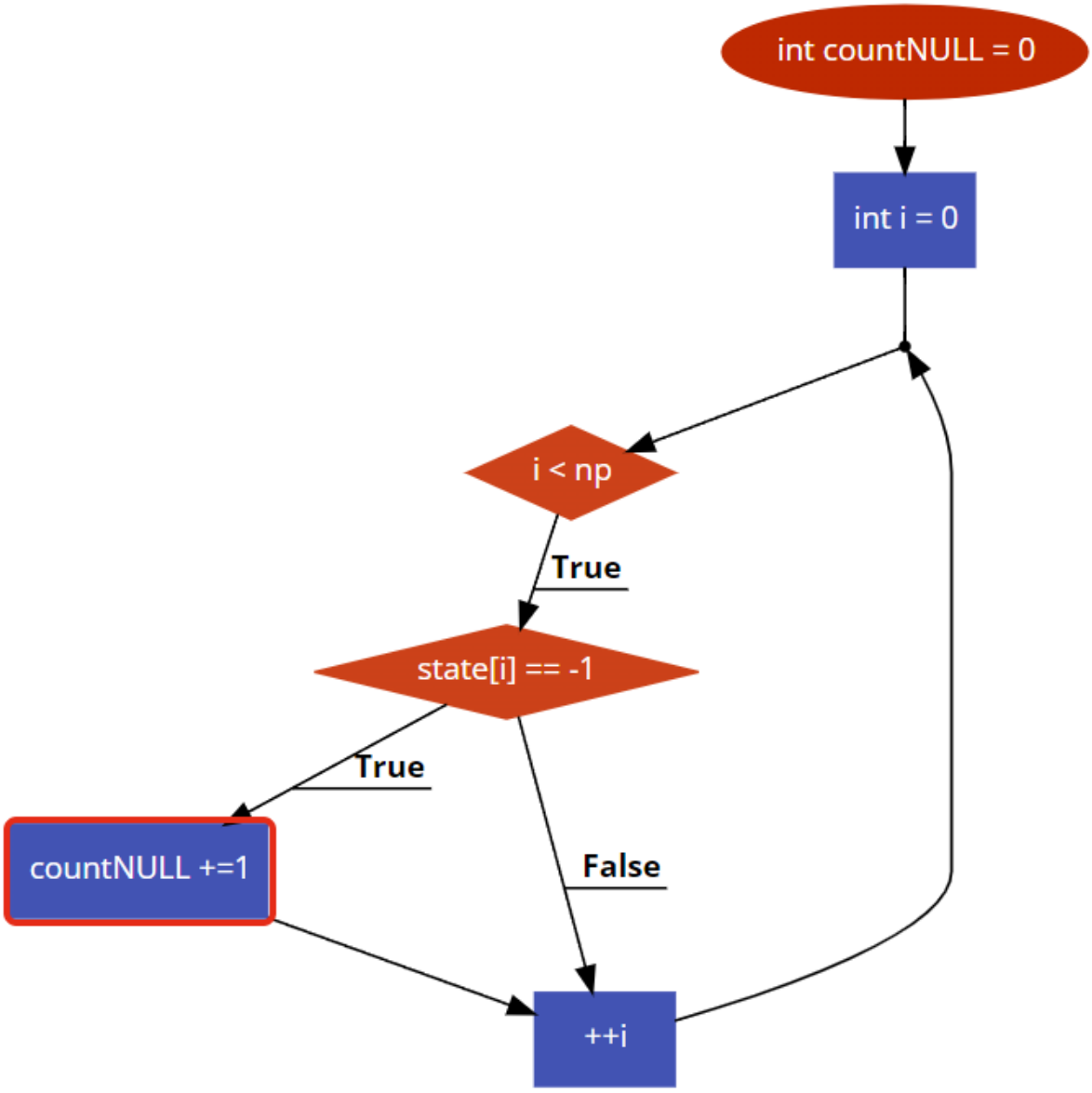
Showtable

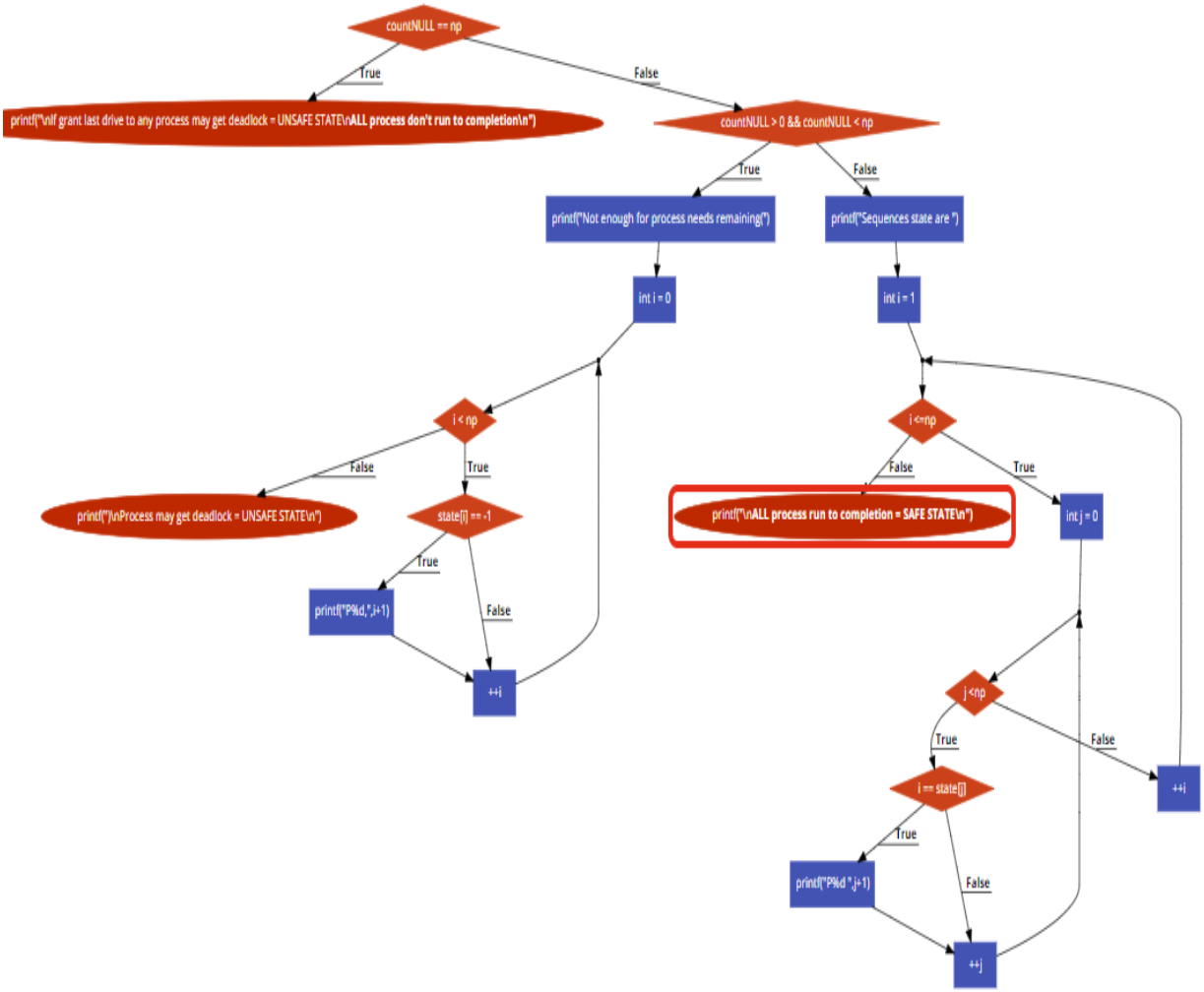

void showTable()
printf("Poscess Allocation Max Need Avail State\n ")
int i = 0
i < 4
False
True
printf(" ")
int j = 0
j < nr
True
False
printf("%c ", 65 + j)
++i
++j


printf("\n")
int i = 0
i < np
True
printf("P%d\t ",i+1)
inTable(i,Allocation)
printf(" ")
inTable(i, Max)
printf(" ")
inTable(i, Need)
printf(" ")
inTable(i, Available)
printf(" ")
state[i] == -1
False
True
printf("%d", state[i])
printf("\n")
++i

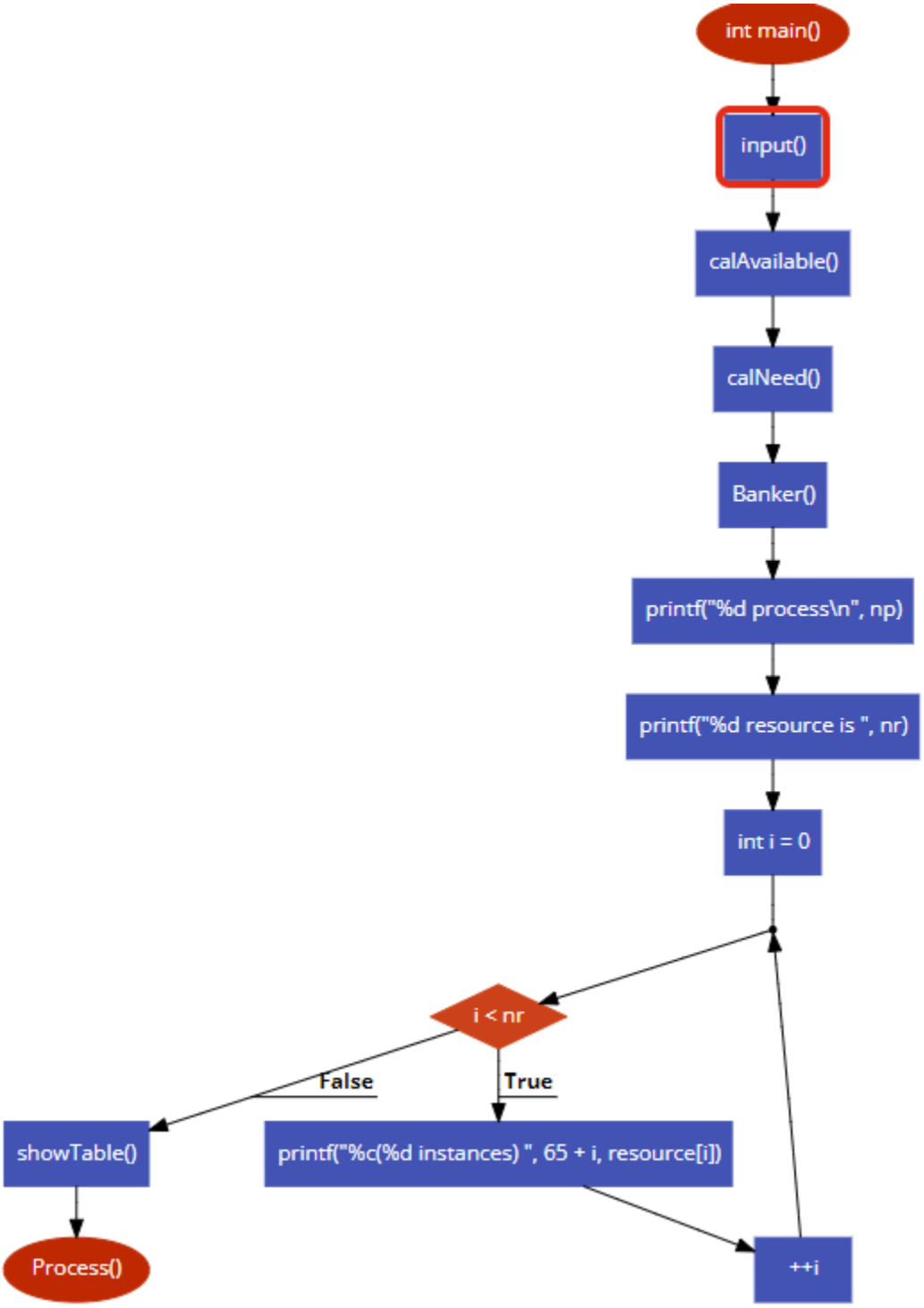
Process


printf("Available drives = ")
int i = 0
i < nr
False
True
int First = 0
printf("%d ", AvailDrives[i])
int i = 0
++i
i < np
True
state[i] == 1
False
True
False
First = i + 1
++i
First !=0
True
printf("\nFirst state allocate(")
int i = 0
i < nr
False
True
printf(") drives to ")
printf("%d,", Need[First-1][i])
printf("P%d\n", First)
++i
printf("P%d completes and releases(", First)
int i = 0
i<nr
False
True
printf("Available drives\n")
printf("%d,",Available[First-1][i])
printf("Next state ...\n")
i++


int countNULL = 0
int i = 0
i < np
True
state[i] == -1
True
countNULL +=1
False
++i


countNULL == np
True
False
printf("\nIf grant last drive to any process may get deadlock = UNSAFE STATE\n**ALL process don't run to completion\n**")
countNULL > 0 && countNULL < np
True
False
printf("Not enough for process needs remaining(")
printf("Sequences state are ")
int i = 0
int i = 1
i < np
i <=np
False
True
False
True
printf("\nProcess may get deadlock = UNSAFE STATE\n")
state[i] == -1
printf("\n**ALL process run to completion = SAFE STATE\n**")
int j = 0
True
False
printf("P%d,",i+1)
++i
j <np
True
False
++i
i == state[j]
True
False
printf("P%d ",j+1)
++j

Main


int main()
input()
calAvailable()
calNeed()
Banker()
printf("%d process\n", np)
printf("%d resource is ", nr)
int i = 0
i < nr
False
True
showTable()
printf("%c(%d instances) ", 65 + i, resource[i])
Process()
++i

ผลการทำงาน

Safe state

```
-----------------------------------------------------
5 process
3 resource is A(10 instances) B(5 instances) C(7 instances)
=====================================================
Poscess      Alloc      Max      Need       Avail      State
          A  B  C    A  B  C    A  B  C    A  B  C
P1        0  1  0    7  5  3    7  4  3    7  5  5     4
P2        2  0  0    3  2  2    1  2  2    5  3  2     1
P3        3  0  2    9  0  2    6  0  0   10  5  7     5
P4        2  1  1    2  2  2    0  1  1    7  4  3     2
P5        0  0  2    4  3  3    4  3  1    7  4  5     3
=====================================================
Available drives = 3 3 2
First state allocate(1,2,2,) drives to P2
P2 completes and releases(5,3,2,)Available drives
Next state ...
Sequences state are P2 P4 P5 P1 P3
**ALL process run to completion = SAFE STATE
```

โปรแกรมทำงานปกติตามขั้นตอน ไม่เกิดDeadlock

Unsafe state

```
5 process
3 resource is A(10 instances) B(5 instances) C(7 instances)
=============================================================
Poscess      Allocation      Max      Need       Avail      State
          A  B  C    A  B  C    A  B  C    A  B  C
P1        7  4  2    7  8  9    0  4  7              NULL
P2        5  9  6    8 12  9    3  3  3              NULL
P3        2  4  5    4  5 15    2  1 10              NULL
P4        8  5  1    6  3  5   -2 -2  4              NULL
P5        6  9 12    3  6  6   -3 -3 -6              NULL
=====================================================
Available drives = -18 -26 -19
If grant last drive to any process may get deadlock = UNSAFE STATE
**ALL process don't run to completion
```

โปรแกรมเกิดDeadlock ทันที

Unsafe state

```
5 process
3 resource is A(10 instances) B(5 instances) C(7 instances)
============================================================
Poscess      Allocation      Max      Need       Avail      State
          A  B  C    A  B  C    A  B  C    A  B  C
P1        0  4  0    7  5  3    7  1  3              NULL
P2        2  0  0    3  2  2    1  2  2              NULL
P3        3  0  2    6  0  0    3  0 -2    6  0  4     1
P4        2  1  1    2  2  2    0  1  1              NULL
P5        0  0  2    4  3  3    4  3  1              NULL
============================================================
Available drives = 3 0 2
First state allocate(3,0,-2,) drives to P3
P3 completes and releases(6,0,4,)Available drives
Next state ...
Not enough for process needs remaining(P1,P2,P4,P5,)
Process may get deadlock = UNSAFE STATE
PS D:\git\AI>
```

โปรแกรมทำงานได้แต่เกิด Deadlockระหว่างทาง

สรุปผลการทดลอง

Deadlock เกิดขึ้นได้ในระบบการประมวลผลโดยมีเงื่อนไขในการ 4 ข้อ และต้องเกิดพร้อมกันถ้ามีอันใดอันหนึ่งเกิดระบบก็จะทำงานได้ตามปกติ