ใบงานที่ 7

เรื่อง Banker’s Algorithm

จัดทำโดย

นางสาวรัชนีกร เชื้อดี 65543206077-1

เสนอ

อาจารย์ปิยพล ยืนยงสถาวร

ใบงานนี้เป็นส่วนหนึ่งของรายวิชา ระบบปฏิบัติการ

หลักสูตรวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา

ประจำภาคที่ 2 ปีการศึกษา 2566

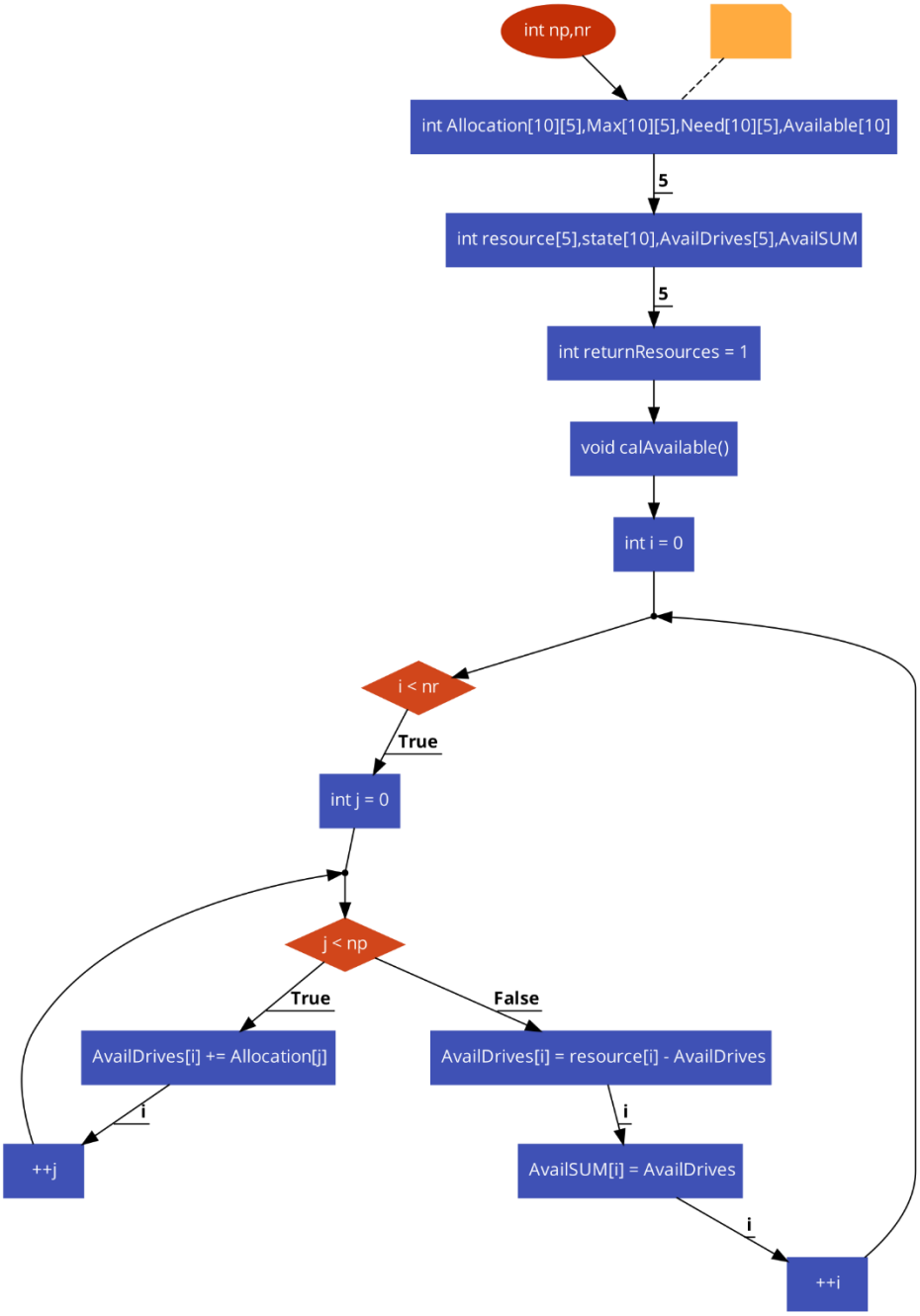
ใบงานที่ 7

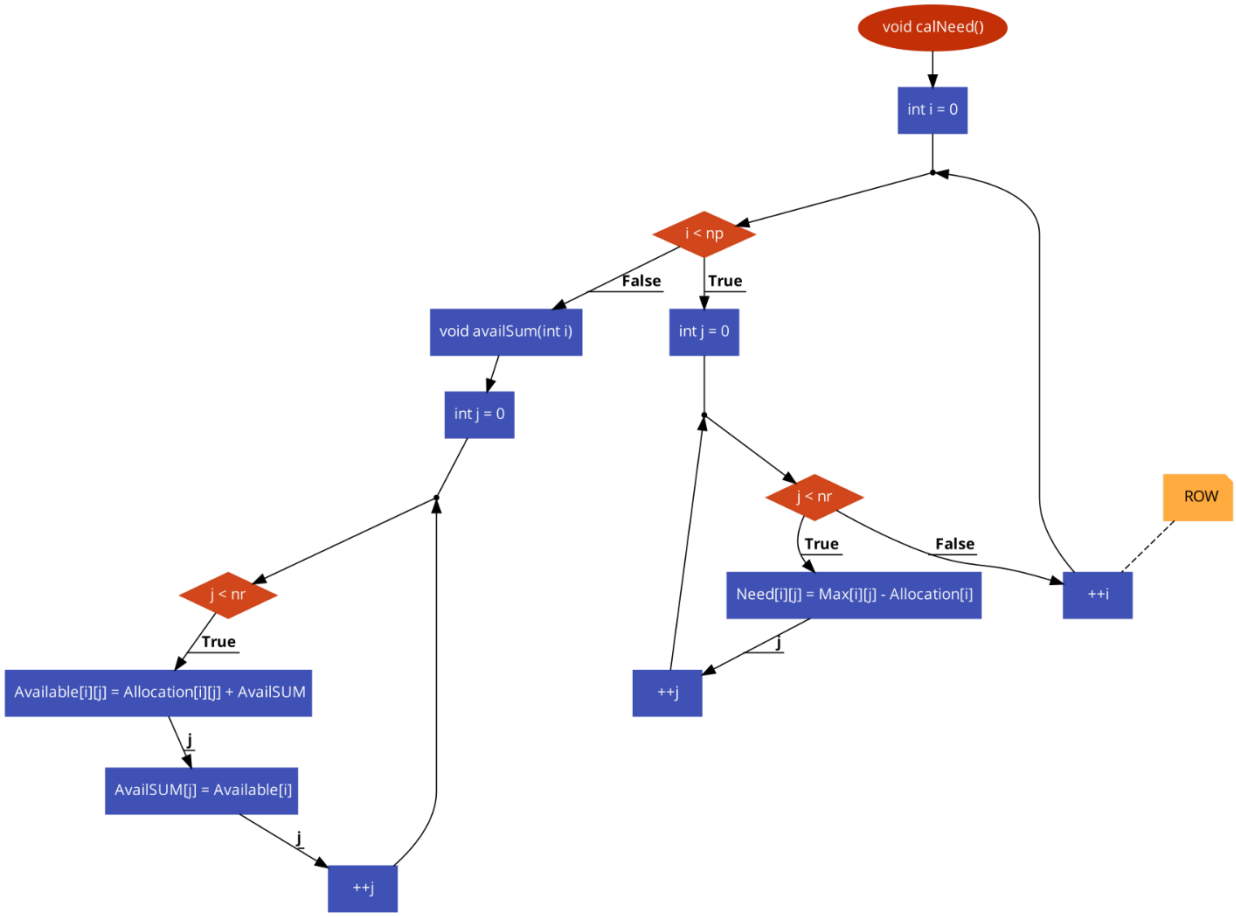
Banker's Algorithm

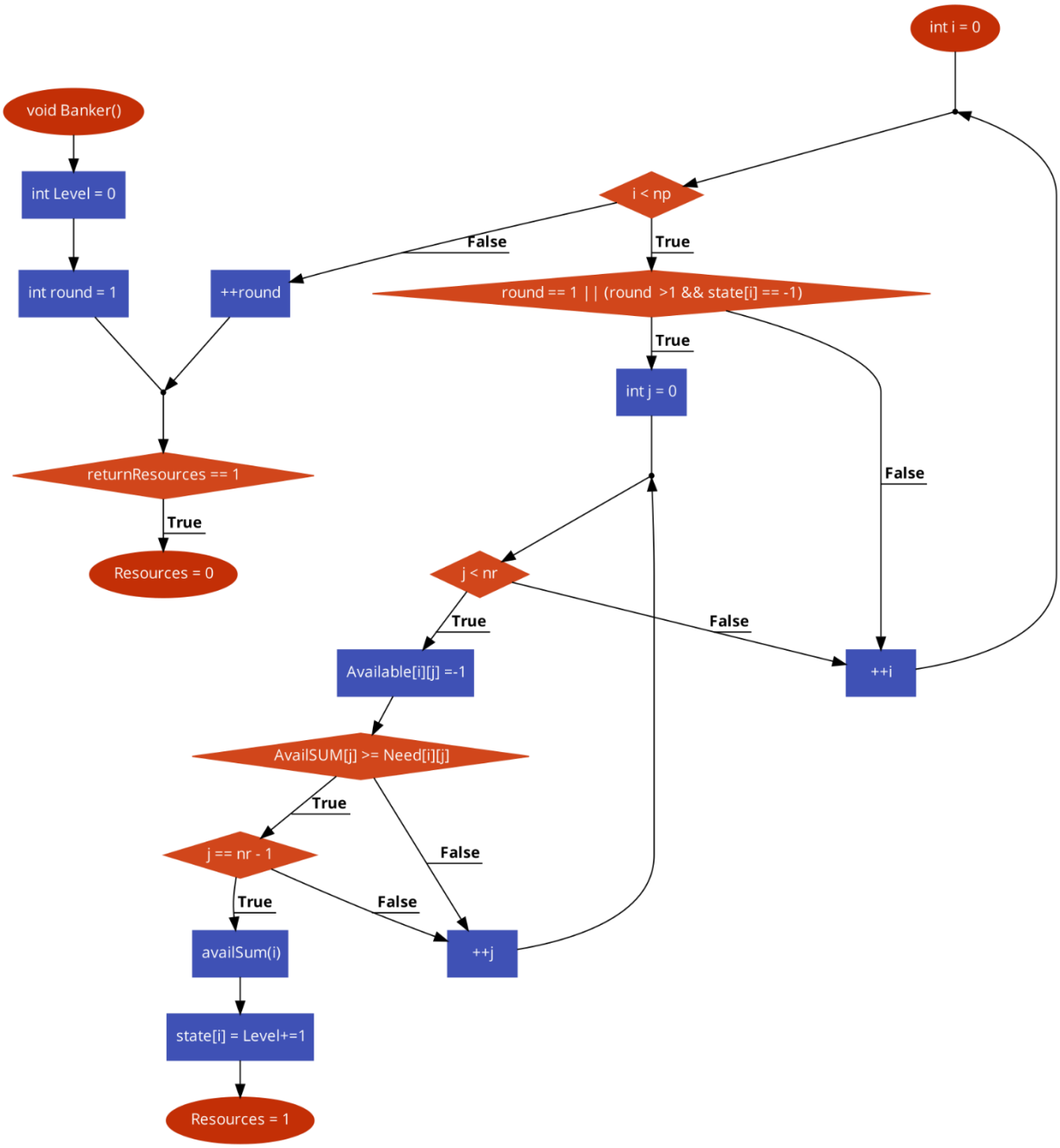
ขั้นตอนการทดลอง

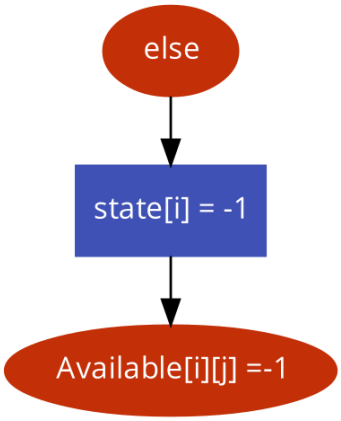
1. ออกแบบโปรแกรม ด้วยวิธีที่ได้ศึกษามา เช่น การเขียนโฟลชาร์ต
2. เขียนโปรแกรม และอธิบายโค้ดฟังก์ชันการทำงานหลัก
3. ตารางกำหนดตัวแปรหลักที่ใช้ในโค้ดโปรแกรม (ชื่อตัวแปร ชนิด เก็บข้อมูลอะไรบ้าง ทำหน้าที่อะไร)
4. บันทึกขั้นตอนการทดลอง ทุกขั้นตอน (ปริ้นรูปประกอบ แล้วเขียนอธิบาย)
5. สรุปผลการทดลอง (ปริ้นรูปประกอบ แล้วเขียนอธิบาย)

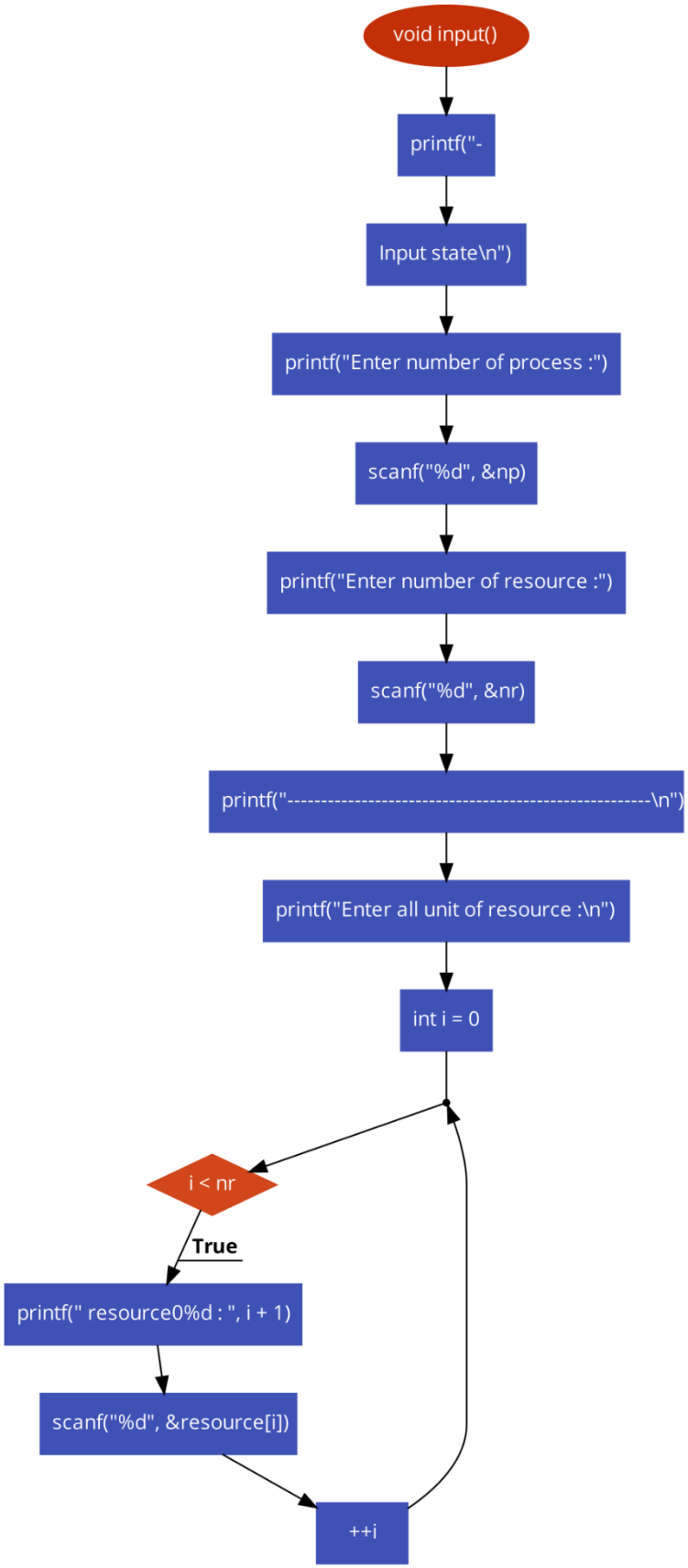
Flowchart

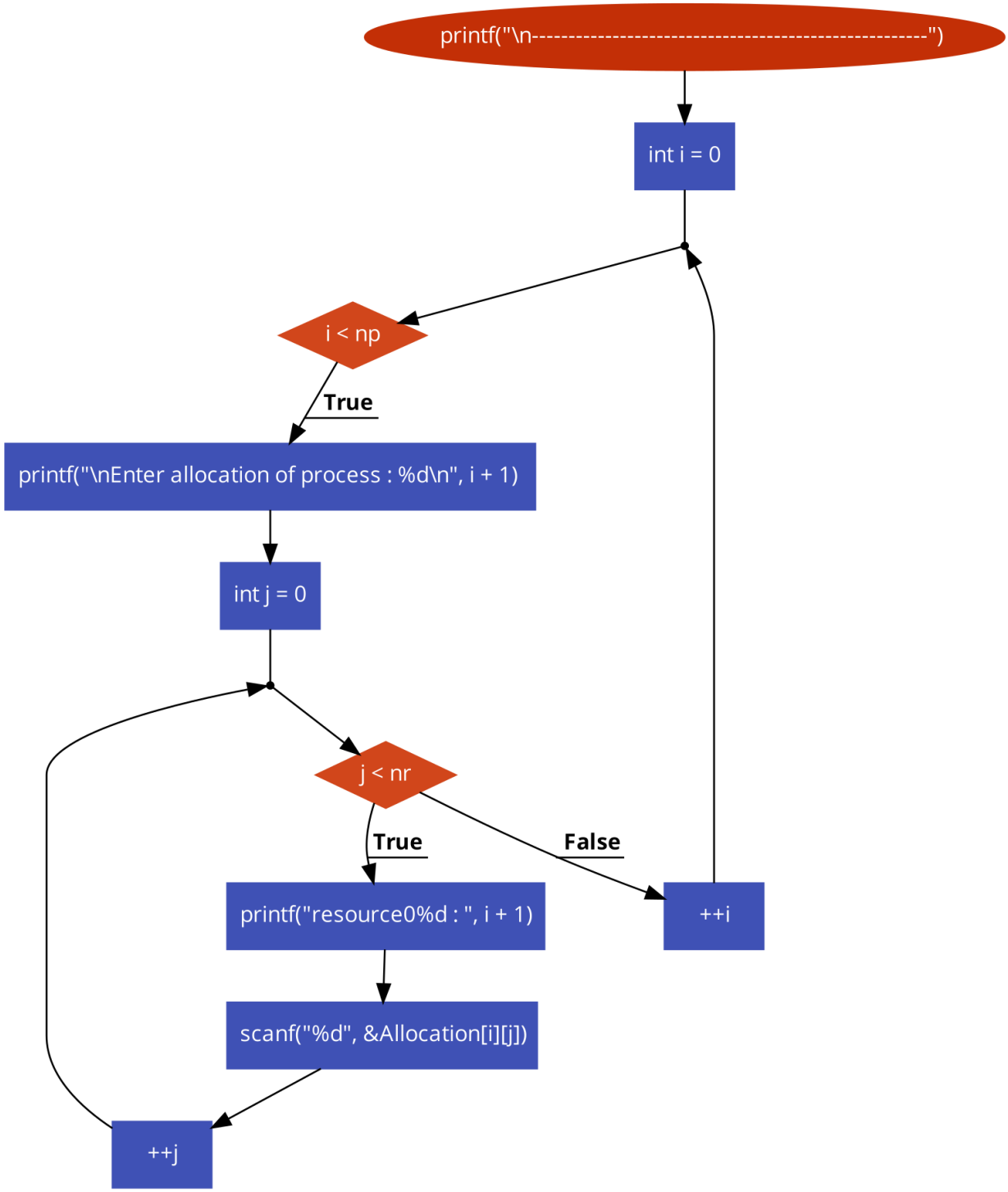

int np,nr
int Allocation[10][5],Max[10][5],Need[10][5],Available[10]
5
int resource[5],state[10],AvailDrives[5],AvailSUM
5
int returnResources = 1
void calAvailable()
int i = 0
i < nr
True
int j = 0
j < np
True
False
AvailDrives[i] += Allocation[j]
AvailDrives[i] = resource[i] - AvailDrives
i
i
++j
AvailSUM[i] = AvailDrives
i
++i

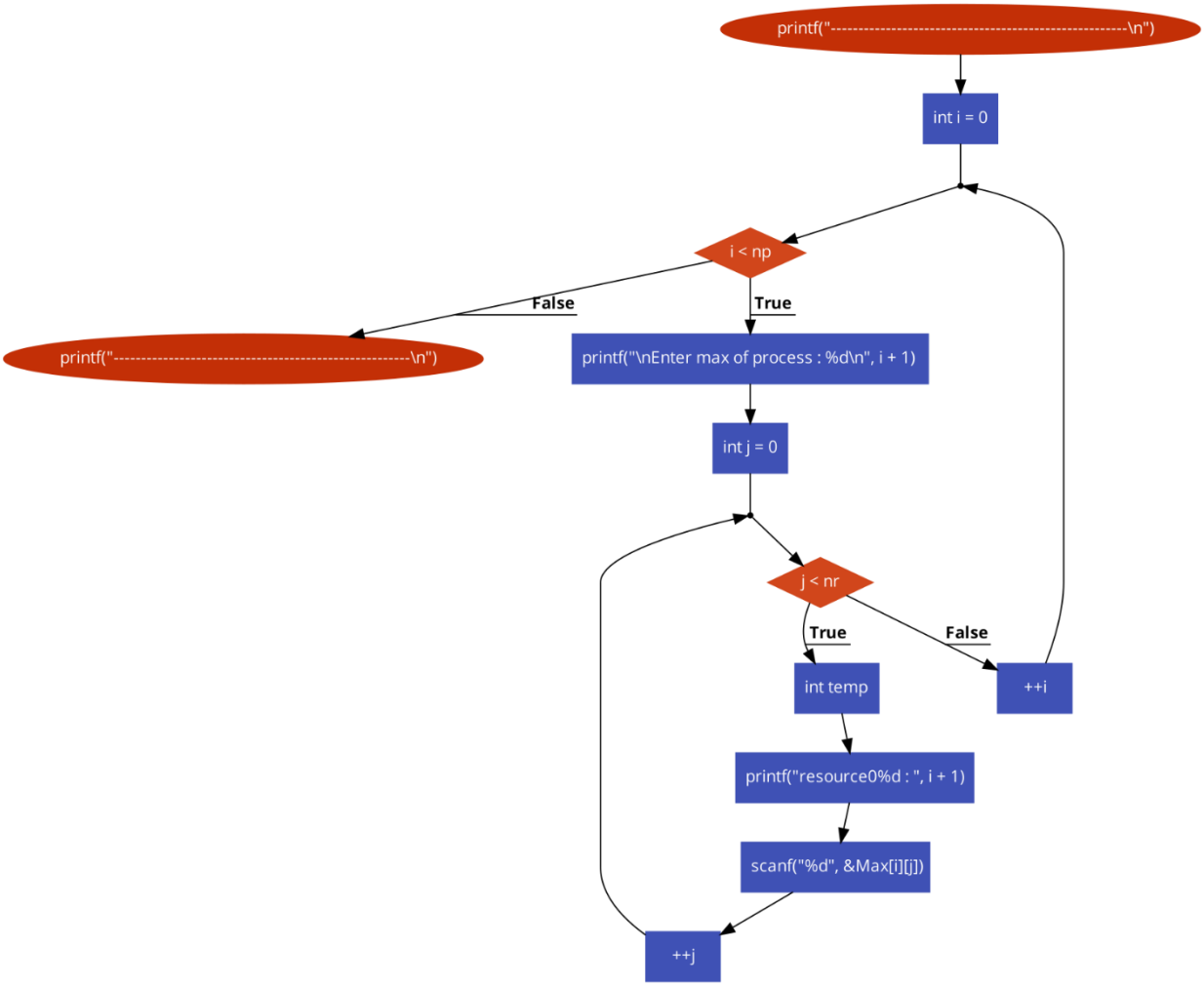

void calNeed()
int i = 0
i < np
False
True
void availSum(int i)
int j = 0
int j = 0
j < nr
True
False
ROW
Need[i][j] = Max[i][j] - Allocation[i]
++i
j
++j
j < nr
True
Available[i][j] = Allocation[i][j] + AvailSUM
j
AvailSUM[j] = Available[i]
j
++j

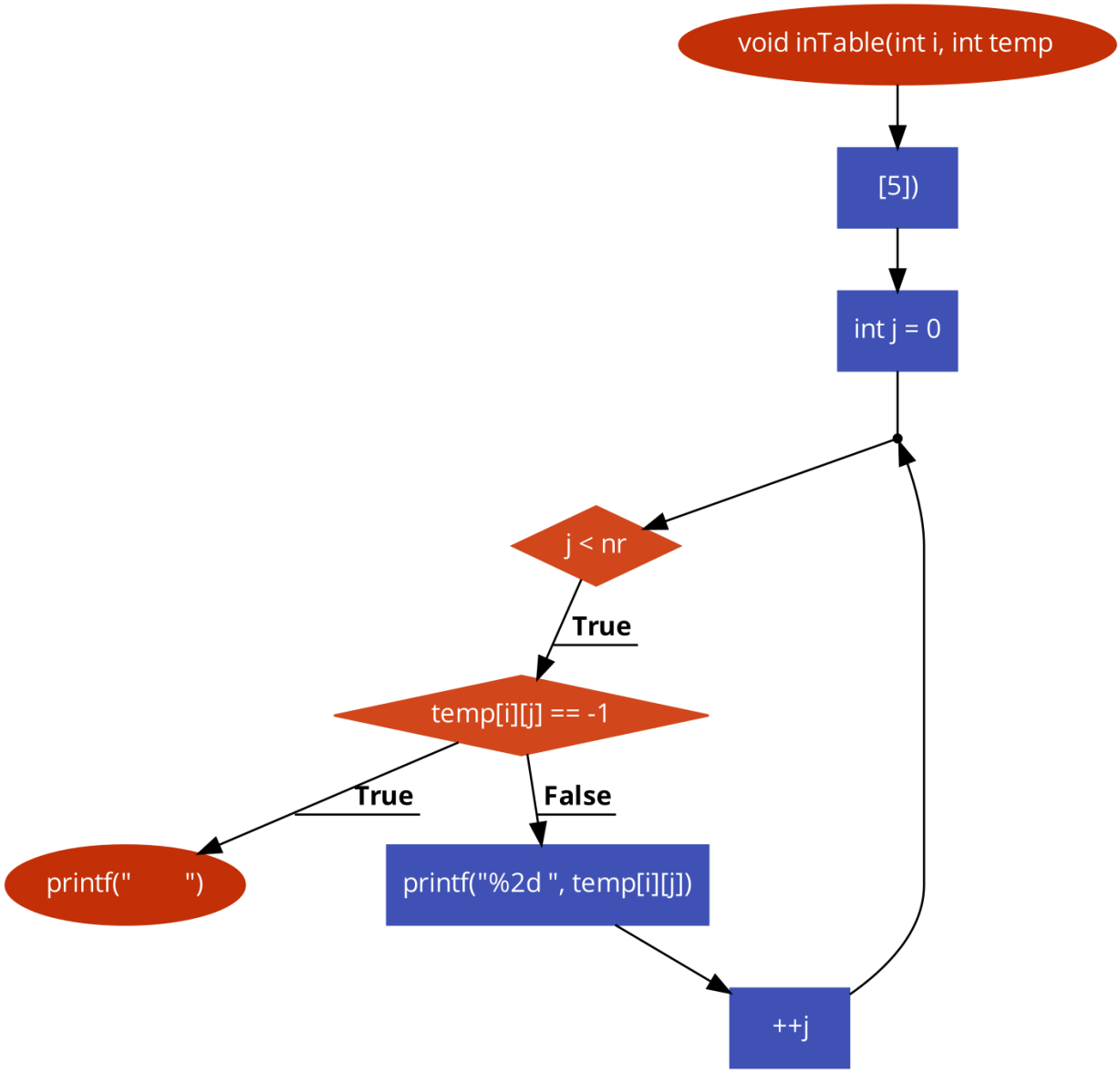

void Banker()
int Level = 0
int round = 1
returnResources == 1
True
Resources = 0
++round
int i = 0
i < np
False
True
round == 1 | | (round >1 && state[i] == -1)
True
int j = 0
False
j < nr
True
False
Available[i][j] =-1
++i
AvailSUM[j] >= Need[i][j]
True
False
j == nr - 1
True
False
availSum(i)
++j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

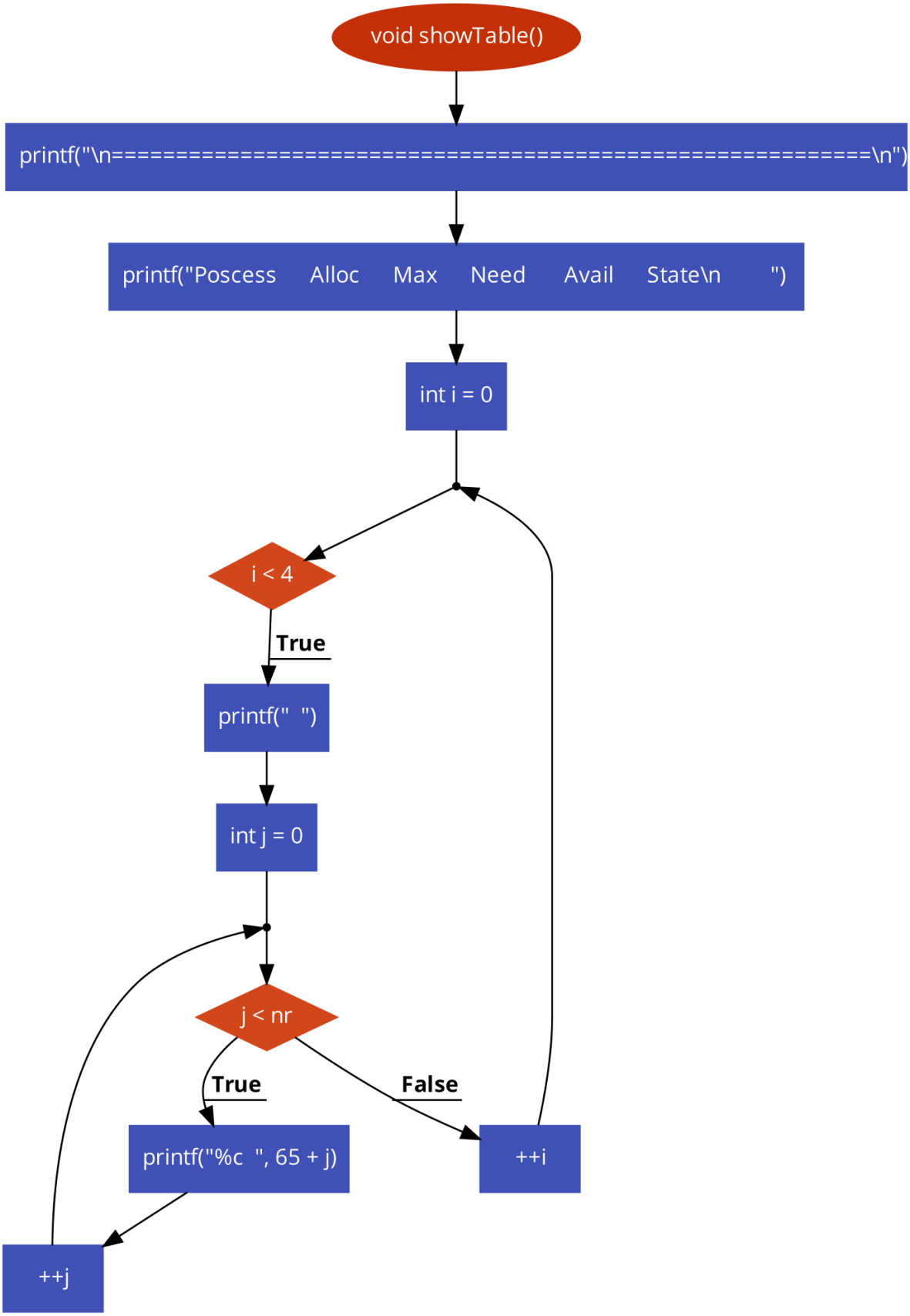

else
state[i] = -1
Available[i][j] =-1

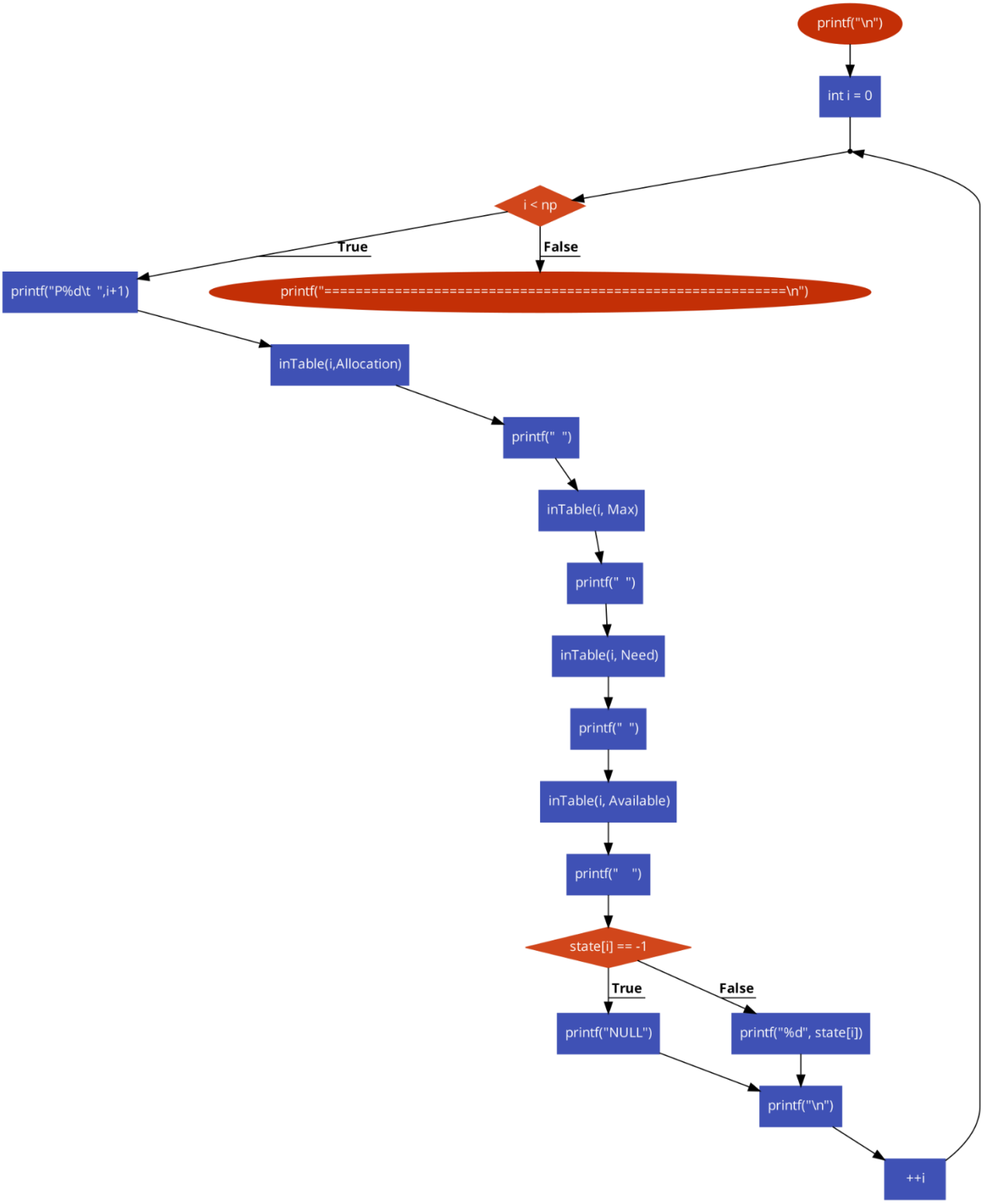

void input()
printf("-
Input state\n")
printf("Enter number of process :")
scanf("%d", &np)
printf("Enter number of resource :")
scanf("%d", &nr)
printf("----------\n")
printf("Enter all unit of resource :\n")
int i = 0
i < nr
True
printf(" resource0%d : ", i + 1)
scanf("%d", &resource[i])
++i

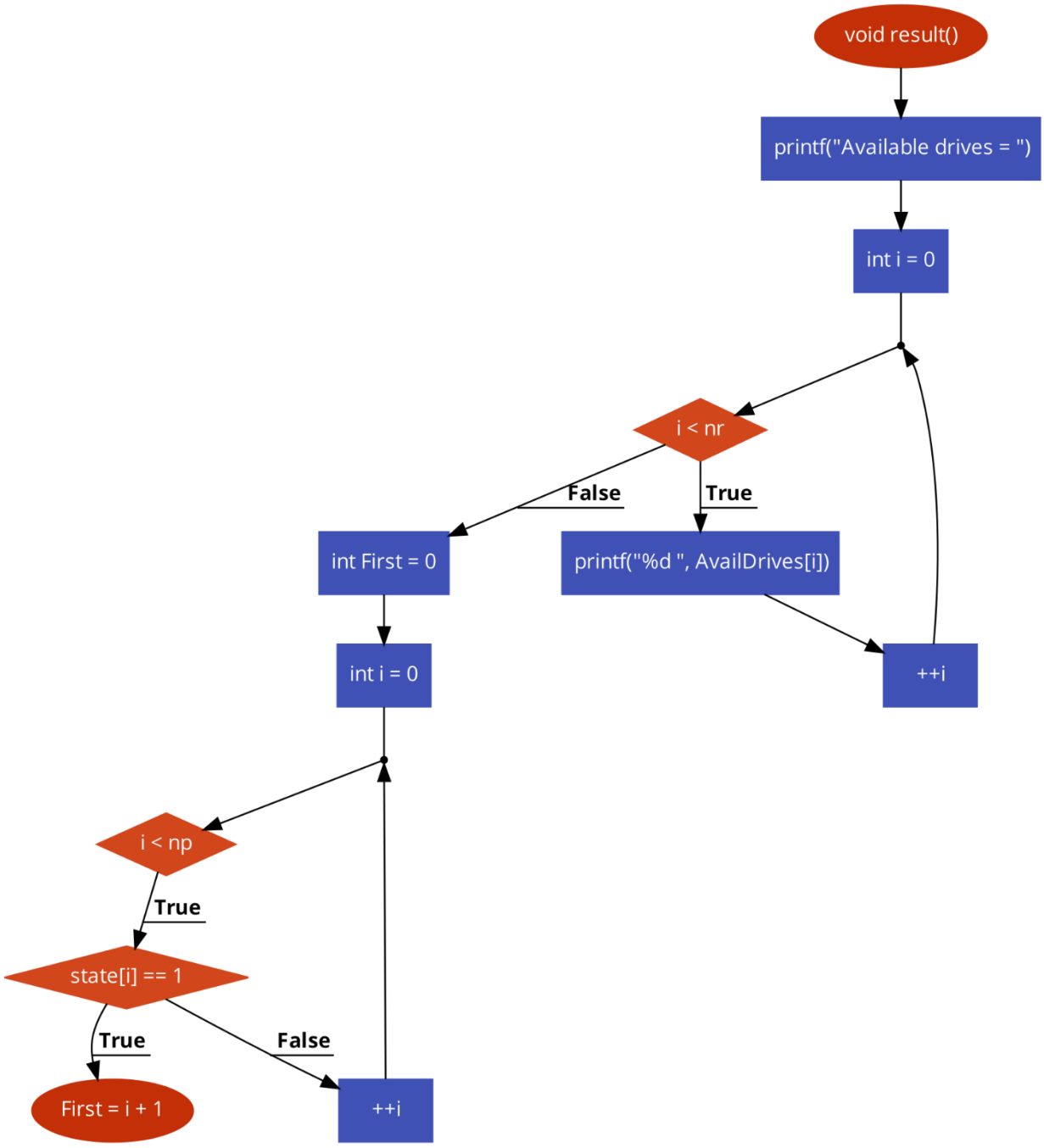

printf("\n----------")
int i = 0
i < np
True
printf("\nEnter allocation of process : %d\n", i + 1)
int j = 0
j < nr
True
False
printf("resource0%d : ", i + 1)
++i
scanf("%d", &Allocation[i][j])
++j

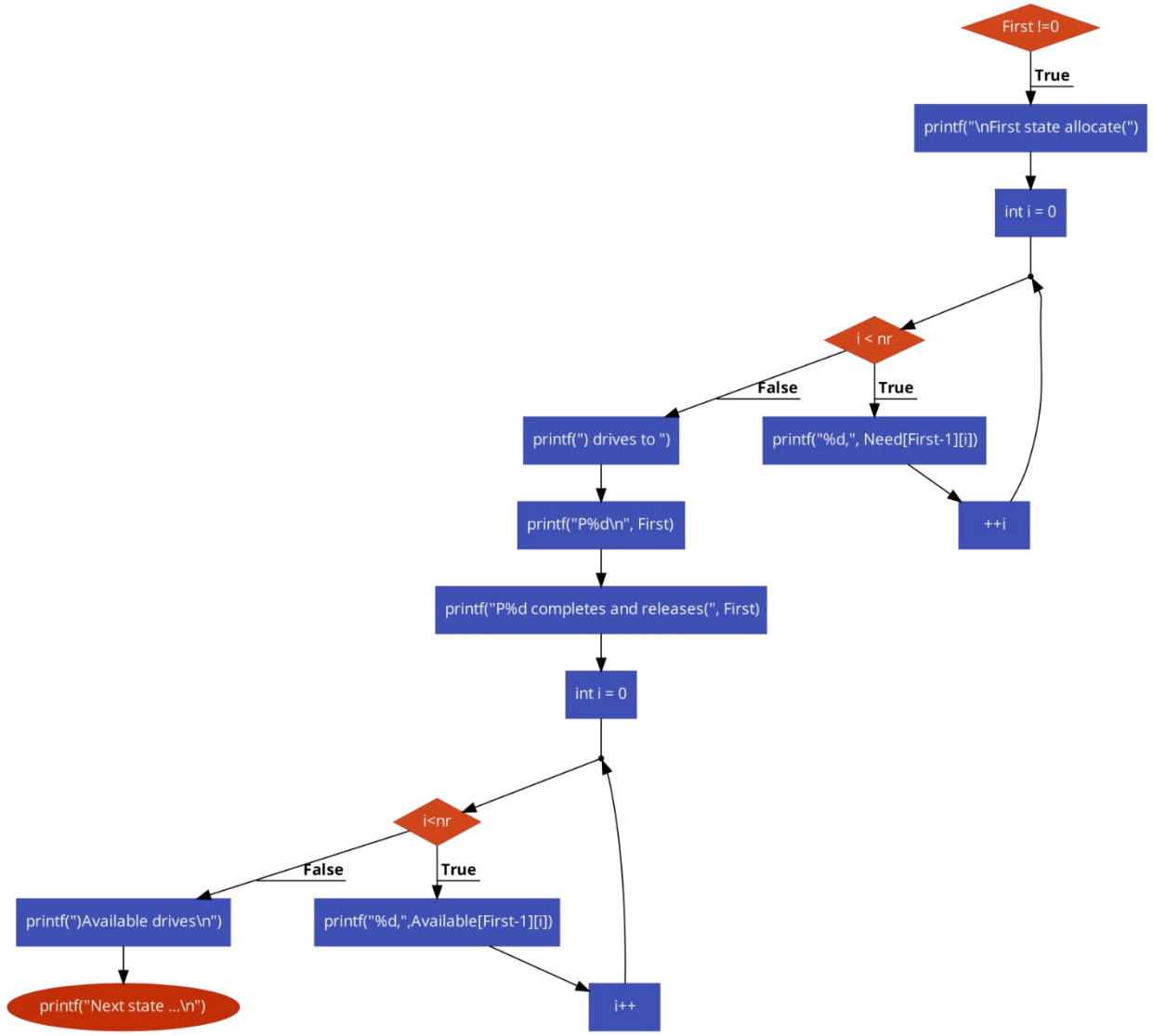

printf("----------\n")
int i = 0
i < np
False
True
printf("----------\n")
printf("\nEnter max of process : %d\n", i + 1)
int j = 0
j < nr
True
False
int temp
++i
printf("resource0%d : ", i + 1)
scanf("%d", &Max[i][j])
++j

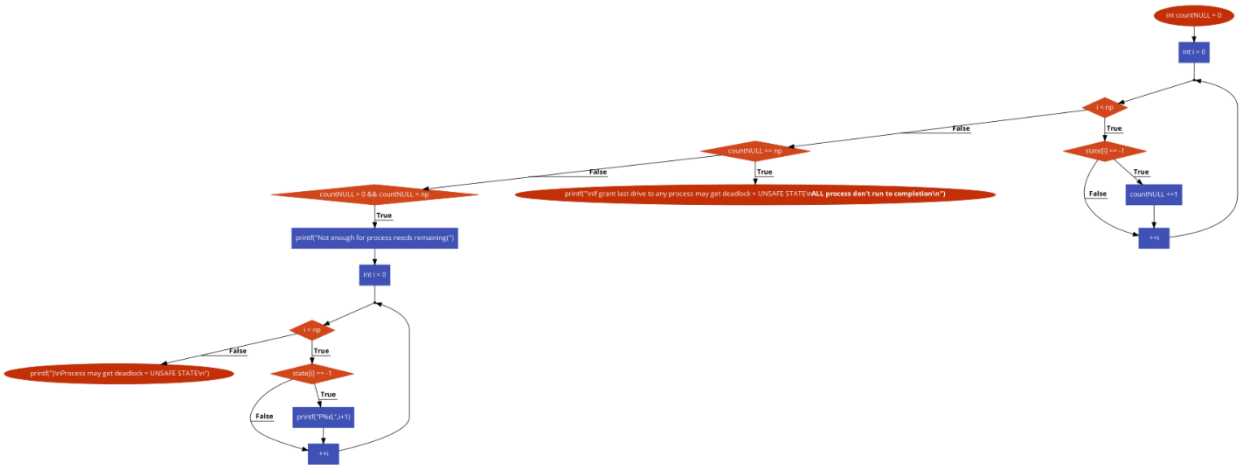

void inTable(int i, int temp
[5])
int j = 0
j < nr
True
temp[i][j] == -1
True
False
printf(" ")
printf("%2d ", temp[i][j])
++j

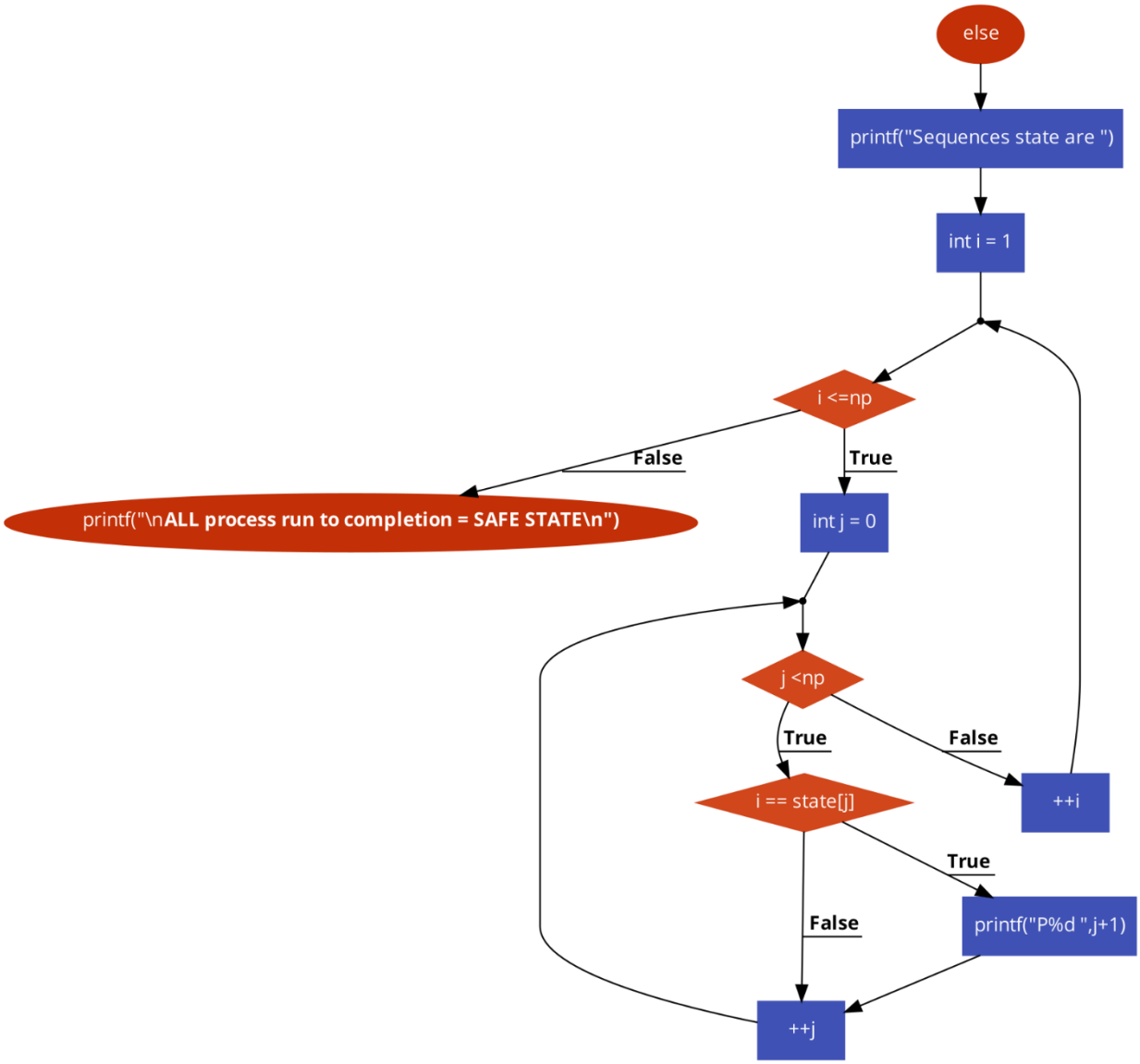

void showTable()
printf("\n==\n")
printf("Poscess Alloc Max Need Avail State\n ")
int i = 0
i < 4
True
printf(" ")
int j = 0
j < nr
True
False
printf("%c ", 65 + j)
++i
++j

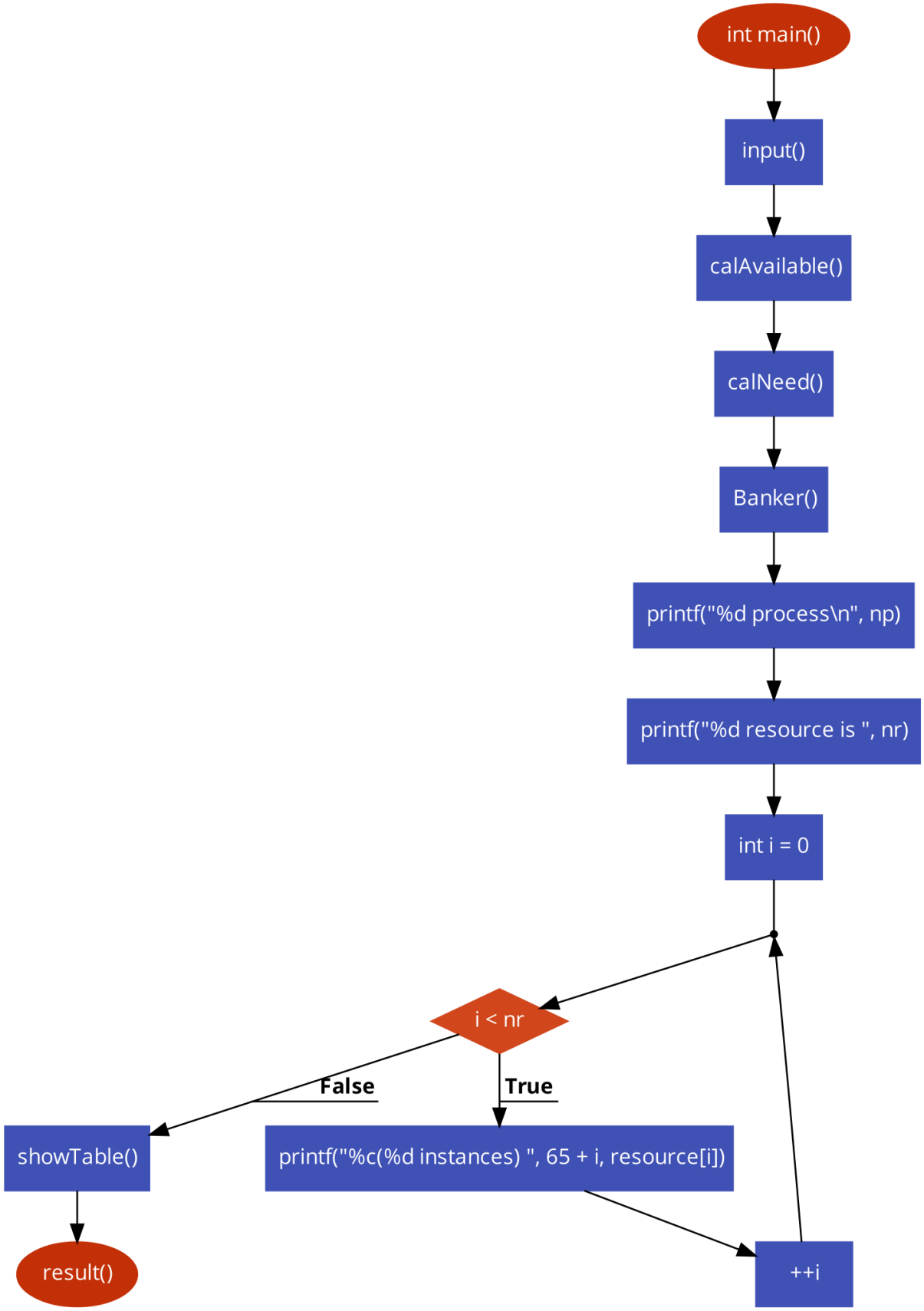

printf("\n")
int i = 0
i < np
True
False
printf("P%d\t ",i+1)
printf("===\n")
inTable(i,Allocation)
printf(" ")
inTable(i, Max)
printf(" ")
inTable(i, Need)
printf(" ")
inTable(i, Available)
printf("   ")
state[i] == -1
True
False
printf("NULL")
printf("%d", state[i])
printf("\n")
++i


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


else
printf("Sequences state are ")
int i = 1
i <=np
False
True
printf("\nALL process run to completion = SAFE STATE\n")
int j = 0
j <np
True
False
i == state[j]
++i
True
False
printf("P%d ",j+1)
++j


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])
result()
++i

อธิบาย Code

```c
#include <stdio.h>

int np,nr;      // number process and number resource
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;

```

1. **int np, nr;:** เป็นตัวแปรที่ใช้เก็บจำนวนของกระบวนการ (Process) และจำนวนของทรัพยากร (Resource) ตามลำดับ
2. **int Allocation[10][5], Max[10][5], Need[10][5], Available[10][5];:** เป็นอาร์เรย์ที่ใช้เก็บข้อมูลที่เกี่ยวข้องกับการจัดสรรทรัพยากร ซึ่งแต่ละอาร์เรย์มีขนาด 10x5 ซึ่งมีความหมายดังนี้
    - **Allocation:** เก็บจำนวนของทรัพยากรที่ถูกจัดสรรให้กับแต่ละกระบวนการ
    - **Max:** เก็บจำนวนสูงสุดของทรัพยากรที่กระบวนการนั้นๆ ต้องการ
    - **Need:** เก็บจำนวนทรัพยากรที่เหลือที่กระบวนการนั้นๆ ต้องการ
    - **Available:** เก็บจำนวนของทรัพยากรที่พร้อมใช้งานในปัจจุบัน
3. **int resource[5], state[10], AvailDrives[5], AvailSUM[5];:** เป็นอาร์เรย์และตัวแปรที่ใช้เก็บข้อมูลเพิ่มเติม เช่น สถานะของแต่ละกระบวนการ หรือการใช้งานของทรัพยากร
4. **int returnResources = 1;:** เป็นตัวแปรที่ใช้เก็บค่าเพื่อกำหนดว่าจะคืนทรัพยากรหรือไม่ โดยในที่นี้ถ้าเป็น 1 หมายถึงจะคืนทรัพยากร และถ้าเป็น 0 หมายถึงไม่คืนทรัพยากร

```
8   void calAvailable(){ //คำนวณหา Available drives
9       for (int i = 0; i < nr; ++i) {
10          for (int j = 0; j < np; ++j) {
11              AvailDrives[i] += Allocation[j][i];
12          }
13          AvailDrives[i] = resource[i] - AvailDrives[i];
14          AvailSUM[i] = AvailDrives[i];
15      }
16  }
```

ฟังก์ชัน calAvailable() ทำหน้าที่คำนวณหาจำนวนของทรัพยากรที่พร้อมใช้งานในปัจจุบัน (Available drives) โดยใช้ข้อมูลจากอาร์เรย์ Allocation และ resource ซึ่งเป็นอาร์เรย์ที่เก็บข้อมูลเกี่ยวกับการจัดสรรทรัพยากรและข้อมูลเกี่ยวกับทรัพยากรทั้งหมดตามลำดับ

การทำงานของฟังก์ชันนี้อธิบายได้ดังนี้:

1. ใช้ลูป for เพื่อทำการวนลูปผ่านทุกหลักของทรัพยากร (column) ในอาร์เรย์ Allocation โดยใช้ตัวแปร i เป็นตัวชี้ลำดับหลัก
2. ภายในลูป for นี้มีลูปซ้อนอีกทีที่ใช้เพื่อวนลูปผ่านทุกแถว (row) ของกระบวนการ (Process) ในอาร์เรย์ Allocation โดยใช้ตัวแปร j เป็นตัวชี้ลำดับแถว
3. ภายในลูปซ้อนนี้ เมื่อวนลูปผ่านทุกแถวแล้ว จะทำการบวกค่าทรัพยากรที่ถูกจัดสรรให้กับแต่ละกระบวนการ ในหลักที่กำหนดโดยตัวแปร i เข้ากับอาร์เรย์ AvailDrives ซึ่งจะเป็นจำนวนของทรัพยากรที่ถูกใช้งานอยู่
4. หลังจากนั้นจะทำการคำนวณหาจำนวนทรัพยากรที่พร้อมใช้งาน โดยลบจำนวนทรัพยากรที่ถูกใช้งานอยู่ (AvailDrives) ออกจากจำนวนทรัพยากรทั้งหมด (resource) ซึ่งผลลัพธ์จะเก็บไว้ในตัวแปร AvailDrives[i]
5. นอกจากนี้ยังมีการกำหนดค่า AvailSUM[i] เท่ากับค่า AvailDrives[i] เพื่อใช้งานในโค้ดอื่นๆ ต่อไป

```
17  void calNeed(){  //คำนวณหาทรัพยากรที่โปรเซสต้องการ
18      for (int i = 0; i < np; ++i) { //ROW
19          for (int j = 0; j < nr; ++j) {//COMLUMN
20              Need[i][j] = Max[i][j] - Allocation[i][j];
21          }
22      }
23  }
```

ฟังก์ชัน calNeed() ทำหน้าที่คำนวณหาจำนวนของทรัพยากรที่โปรเซสต้องการเพื่อให้สามารถทำงานได้ (Need) โดยใช้ข้อมูลจากอาร์เรย์ Max และ Allocation ซึ่งเป็นข้อมูลเกี่ยวกับทรัพยากรสูงสุดที่โปรเซสต้องการและจำนวนทรัพยากรที่ถูกจัดสรรให้แก่โปรเซสตามลำดับ

การทำงานของฟังก์ชันนี้อธิบายได้ดังนี้:

1. ใช้ลูป for เพื่อทำการวนลูปผ่านทุกแถว (row) ของกระบวนการ (Process) ในอาร์เรย์ Max โดยใช้ตัวแปร i เป็นตัวชี้ลำดับแถว
2. ภายในลูป for นี้มีลูปซ้อนอีกทีที่ใช้เพื่อวนลูปผ่านทุกหลัก (column) ของทรัพยากร (Resource) ในอาร์เรย์ Max โดยใช้ตัวแปร j เป็นตัวชี้ลำดับหลัก
3. ภายในลูปซ้อนนี้ เมื่อวนลูปผ่านทุกหลักแล้ว จะทำการคำนวณหาจำนวนทรัพยากรที่โปรเซสต้องการ (Need) โดยลบจำนวนทรัพยากรที่ถูกจัดสรรให้กับโปรเซสตามลำดับ (Allocation) ออกจากจำนวนทรัพยากรที่โปรเซสต้องการสูงสุด (Max) ซึ่งผลลัพธ์จะเก็บไว้ในอาร์เรย์ Need
4. การวนลูปนี้จะทำให้ค่าของ Need[i][j] ที่เก็บไว้ในแต่ละตำแหน่งของอาร์เรย์ Need เป็นจำนวนของทรัพยากรที่โปรเซสต้องการเพื่อทำงานในแต่ละช่วงเวลา

```
24  void availSum(int i){  // คืนทรัพยากรณ์ที่ใช้แล้ว
25      for (int j = 0; j < nr; ++j) {
26          Available[i][j] = Allocation[i][j] + AvailSUM[j];
27          AvailSUM[j] = Available[i][j];
28      }
29  }
```

ฟังก์ชัน availSum() ทำหน้าที่คืนทรัพยากรที่ใช้ไปแล้วโดยปรับปรุงค่าของ Available drives และ AvailSUM เมื่อมีการคืนทรัพยากรจากกระบวนการ (Process) ใดๆ ในระบบ

การทำงานของฟังก์ชันนี้อธิบายได้ดังนี้:

1. ใช้ลูป for เพื่อทำการวนลูปผ่านทุกหลัก (column) ของทรัพยากร (Resource) ในอาร์เรย์ Available โดยใช้ตัวแปร j เป็นตัวชี้ลำดับหลัก
2. ในลูป for นี้ จะทำการคำนวณหาจำนวนทรัพยากรที่เป็นไปได้ที่สามารถใช้งานได้ใหม่ โดยการบวกจำนวนทรัพยากรที่ถูกจัดสรรให้กับกระบวนการที่ต้องการคืนทรัพยากร (Allocation[i][j]) กับจำนวนทรัพยากรที่มีอยู่ใน AvailSUM (AvailSUM[j])
3. หลังจากนั้น จะนำค่าที่ได้จากขั้นตอนที่ 2 มาเก็บไว้ใน Available drives และ AvailSUM เพื่อใช้ในการคำนวณในการคืนทรัพยากรของกระบวนการถัดไป
4. การวนลูปนี้จะทำให้ค่าของ Available[i][j] และ AvailSUM[j] ที่เก็บไว้ในแต่ละตำแหน่งของอาร์เรย์ Available และ AvailSUM เป็นจำนวนของทรัพยากรที่สามารถใช้งานได้ใหม่หลังจากที่มีการคืนทรัพยากร โดยจะสามารถนำไปใช้ในการวางแผนการจัดสรรทรัพยากรต่อไปในระบบได้

```
void Banker(){
    int Level = 0;
    for (int round = 1;returnResources == 1; ++round) {//จะวนรอบเรื่อยๆ ถ้ายังมีการคืนทรัยากรอยู่
        returnResources = 0;  //set ค่าเริ่มต้น
        for (int i = 0; i < np; ++i) {          // วนลูปตามจำนวนโปนเซส กำหนดให้เป็น ROW
            if(round == 1 || (round  >1 && state[i] == -1)) {
                //รอบที่ 1 จะตรวจสอบทุกโปรเซส แต่ถ้าเป็นรอบต่อไป จะตรวจสอบแค่โปรเซสที่มี state เป็น NULL
                for (int j = 0; j < nr; ++j) {  // วนลูปตามจำนวน resourceกำหนดให้เป็น COLUMN
                    Available[i][j] =-1;        // set ค่าเริ่มต้นให้ Available เป็น NULL
                    if (AvailSUM[j] >= Need[i][j]) {  //ถ้า Avali มีทรัพยากรเพียงพอต่อ ที่โปรเซสต้องการ (Need)
                        if (j == nr - 1) {      // Avali มีทรัพยากรจนครบเพียงพอจนครบ ตามจำนวน resource
                            availSum(i);        // คืนทรัพยากรณ์ที่ใช้แล้ว
                            state[i] = Level+=1; // ลำดับที่โปรเซสได้ใช้ทรัพยากร
                            returnResources = 1; // มีการคืนทรัพยากร
                        }
                    } else {                     //ถ้า Avali มีทรัพยากรไม่เพียงพอต่อ ที่โปรเซสต้องการ (Need)
                        state[i] = -1;           // state เป็น NULL
                        Available[i][j] =-1;     // Available เป็น NULL
                        break;
                    }
                }
            }
        }
    }
}
```

ฟังก์ชัน Banker() นั้นเป็นฟังก์ชันหลักที่ใช้ในการจำลองขั้นตอนการทำงานของอัลกอริทึม "Banker's Algorithm" ซึ่งเป็นอัลกอริทึมที่ใช้ในการจัดการการใช้ทรัพยากรในระบบที่มีหลายโปรเซสพร้อมกัน โดยเฉพาะเมื่อทรัพยากรมีจำนวนจำกัด

การทำงานของฟังก์ชันนี้อธิบายได้ดังนี้:

1. กำหนดตัวแปร round เพื่อใช้ในการนับรอบของการทำงาน และ Level เพื่อเก็บข้อมูลเกี่ยวกับลำดับของการใช้ทรัพยากร
2. เริ่มต้นการทำงานของลูป for โดยจะทำการวนลูปไปเรื่อยๆ จนกว่าค่า returnResources จะเป็น 0 ซึ่งหมายถึงไม่มีการคืนทรัพยากรเกิดขึ้น
3. กำหนดค่า returnResources เป็น 0 เพื่อเตรียมตัวสำหรับการเช็คการคืนทรัพยากรในรอบใหม่
4. วนลูปตามจำนวนโปรเซส (Process) ที่อยู่ในระบบ โดยในรอบแรกจะตรวจสอบทุกโปรเซส แต่ในรอบถัดไปจะตรวจสอบเฉพาะโปรเซสที่มี state เป็น -1 (ไม่มีการกำหนด state)
5. วนลูปตามจำนวนทรัพยากร โดยตรวจสอบว่าทรัพยากรที่โปรเซสต้องการ (Need) มีอยู่ในทรัพยากรที่พร้อมใช้งาน (Available) หรือไม่
6. ถ้าทรัพยากรที่โปรเซสต้องการมีในทรัพยากรที่พร้อมใช้งาน ให้ทำการคืนทรัพยากรและกำหนด state และ Available ให้โปรเซสนั้นๆ
7. หากทรัพยากรที่โปรเซสต้องการไม่มีในทรัพยากรที่พร้อมใช้งาน ให้ทำการกำหนด state และ Available เป็น -1 (ไม่มีการกำหนด)

8. เมื่อทำการตรวจสอบทุกโปรเซสแล้ว ถ้ามีการคืนทรัพยากรเกิดขึ้น (returnResources == 1) ให้เพิ่ม Level ขึ้นไป และทำการวนลูปต่อไป
9. ทำซ้ำขั้นตอนทั้งหมดไปเรื่อยๆ จนกว่าจะไม่มีการคืนทรัพยากรเกิดขึ้น

```
56  void input(){
57      printf("->> Input state\n");
58      printf("Enter number of process :");
59      scanf("%d", &np);
60      printf("Enter number of resource :");
61      scanf("%d", &nr);
62      printf("-----------------------------------------------------\n");
63      printf("Enter all unit of resource :\n");
64      for (int i = 0; i < nr; ++i) {
65          printf(" resource0%d : ", i + 1);
66          scanf("%d", &resource[i]);
67      }
68      printf("\n-----------------------------------------------------");
69      for (int i = 0; i < np; ++i) {
70          printf("\nEnter allocation of process : %d\n", i + 1);
71          for (int j = 0; j < nr; ++j) {
72              printf("resource0%d : ", i + 1);
73              scanf("%d", &Allocation[i][j]);
74          }
75      }
76      printf("-----------------------------------------------------\n");
77      for (int i = 0; i < np; ++i) {
78          printf("\nEnter max of process : %d\n", i + 1);
79          for (int j = 0; j < nr; ++j) {
80              int temp;
81              printf("resource0%d : ", i + 1);
82              scanf("%d", &Max[i][j]);
83          }
84      }
85      printf("-----------------------------------------------------\n");
86  }
```

ฟังก์ชัน input() ใช้ในการรับข้อมูลเริ่มต้นเกี่ยวกับสถานะของระบบ ซึ่งประกอบด้วยจำนวนของกระบวนการ (Process) และจำนวนของทรัพยากร (Resource) ที่ใช้ในระบบ รวมถึงการกำหนดจำนวนของแต่ละทรัพยากรที่พร้อมใช้งานและการจัดสรรทรัพยากรให้แก่แต่ละกระบวนการ

การทำงานของฟังก์ชันนี้อธิบายได้ดังนี้:

1. แสดงข้อความที่บอกให้ผู้ใช้ป้อนข้อมูลเริ่มต้นโดยใช้ฟังก์ชัน printf() เพื่อแสดงข้อความที่บอกให้ผู้ใช้ป้อนข้อมูล
2. รับค่าจำนวนของกระบวนการและจำนวนของทรัพยากร โดยใช้ฟังก์ชัน scanf()
3. แสดงข้อความที่บอกให้ผู้ใช้ป้อนจำนวนของแต่ละทรัพยากรที่พร้อมใช้งาน
4. ใช้ลูป for เพื่อให้ผู้ใช้ป้อนจำนวนของแต่ละทรัพยากรที่พร้อมใช้งาน
5. แสดงข้อความที่บอกให้ผู้ใช้ป้อนจำนวนของทรัพยากรที่ถูกจัดสรรให้กับแต่ละกระบวนการ
6. ใช้ลูป for เพื่อให้ผู้ใช้ป้อนจำนวนของทรัพยากรที่ถูกจัดสรรให้กับแต่ละกระบวนการ

7. สร้างตัวแปรชั่วคราว temp เพื่อใช้ในการรับค่าจำนวนทรัพยากร
8. ใช้ลูป for เพื่อให้ผู้ใช้ป้อนจำนวนของทรัพยากรที่ถูกจัดสรรให้กับแต่ละกระบวนการ
9. แสดงข้อความที่บอกให้ผู้ใช้ป้อนจำนวนของทรัพยากรสูงสุดที่โปรเซสต้องการ
10. ใช้ลูป for เพื่อให้ผู้ใช้ป้อนจำนวนของทรัพยากรสูงสุดที่โปรเซสต้องการ
11. จบการทำงานของฟังก์ชัน

```
88  void inTable(int i, int temp[][5]) {
89      for (int j = 0; j < nr; ++j) {
90          if(temp[i][j] == -1) {
91              printf("        ");
92              break;
93          }
94          else
95              printf("%2d ", temp[i][j]);
96      }
97  }
```

ฟังก์ชัน inTable() ใช้สำหรับแสดงข้อมูลในรูปแบบของตาราง โดยรับค่า index i และอาร์เรย์ temp ขนาด nr x 5 เพื่อแสดงข้อมูลที่อยู่ในแถวที่ i ของตารางนี้

การทำงานของฟังก์ชันนี้อธิบายได้ดังนี้:

1. ใช้ลูป for เพื่อวนลูปผ่านทุกหลัก (column) ของแถวที่ i ในตาราง
2. ตรวจสอบว่าค่าในแถว i และหลัก j ของอาร์เรย์ temp เป็น -1 หรือไม่ ถ้าใช่แสดงว่าไม่มีข้อมูลที่ต้องแสดง ให้พิมพ์ช่องว่างเพื่อแสดงช่องว่างในตาราง และจบลูปด้วย break
3. ถ้าไม่ใช่ ให้แสดงค่าที่อยู่ในแถว i และหลัก j ของอาร์เรย์ temp ด้วย printf() และจัดรูปแบบการแสดงค่าด้วย %2d เพื่อให้ค่ามีขนาดคงที่ 2 หลัก
4. ทำซ้ำขั้นตอนที่ 2-3 ไปเรื่อยๆ จนกว่าจะครบทุกหลักในแถวที่ i ของตาราง

```
98  void showTable(){ // ตาราง
99      printf("\n============================================================\n");
100     printf("Poscess      Alloc      Max      Need      Avail      State\n         ");
101     for (int i = 0; i < 4; ++i){
102         printf("  ");
103         for (int j = 0; j < nr; ++j)
104             printf("%c  ", 65 + j);
105     }
106     printf("\n");
107     for (int i = 0; i < np; ++i){
108         printf("P%d\t  ",i+1);
109         inTable(i,Allocation);printf("  ");
110         inTable(i, Max);printf("  ");
111         inTable(i, Need);printf("  ");
112         inTable(i, Available);printf("    ");
113         if(state[i] == -1)
114             printf("NULL");
115         else
116             printf("%d", state[i]);
117         printf("\n");
118     }
119     printf("============================================================\n");
120 }
```

ฟังก์ชัน showTable() ใช้สำหรับแสดงข้อมูลในรูปแบบของตารางที่มีรายละเอียดของ Process Allocation, Max, Need, Available และ State ของแต่ละ Process ซึ่งจะแสดงในรูปแบบตาราง

การทำงานของฟังก์ชันนี้อธิบายได้ดังนี้:

1. ใช้ printf() เพื่อแสดงเส้นขอบของตารางด้านบน
2. ใช้ printf() เพื่อแสดงหัวข้อของแต่ละคอลัมน์ของตาราง
3. ใช้ลูป for เพื่อแสดงหัวข้อของแต่ละคอลัมน์ของตาราง โดยใช้ตัวแปร nr แทนจำนวนคอลัมน์ทั้งหมด
4. ใช้ลูป for เพื่อวนลูปทุกโปรเซสและแสดงข้อมูลของแต่ละโปรเซสในแต่ละแถวของตาราง
5. ใช้ printf() เพื่อแสดงตัวแปร P และตัวเลขของโปรเซส
6. เรียกใช้ฟังก์ชัน inTable() เพื่อแสดงข้อมูล Allocation, Max, Need, Available และ State ของแต่ละโปรเซส โดยใช้ลูป for เพื่อวนลูปทุกคอลัมน์ของตาราง
7. ใช้ printf() เพื่อแสดงข้อมูลของ State ของแต่ละโปรเซส
8. ใช้ printf() เพื่อแสดงเส้นขอบของตารางด้านล่าง

```
121  void result(){
122      printf("Available drives = ");
123      for (int i = 0; i < nr; ++i)
124          printf("%d ", AvailDrives[i]);
125
126      int First = 0;              // ใช้เก็บโปรเก็บโปรเซสที่ได้ใช้ทรัยพยากรอันดับแรก
127      for (int i = 0; i < np; ++i)  // วนลูปหาโปรเซสแรกที่ได้เข้าทำงาน
128          if(state[i] == 1) {
129              First = i + 1;
130              break;
131          }
132      if(First !=0){        //ถ้ามีโปรเซสได้ใช้ทรัพกรอันดับแรก ให้แสดง.....
133          printf("\nFirst state allocate(");
134          for (int i = 0; i < nr; ++i)
135              printf("%d,", Need[First-1][i]);
136          printf(") drives to ");
137          printf("P%d\n", First);
138          printf("P%d completes and releases(", First);
139          for(int i = 0; i<nr; i++)
140              printf("%d,",Available[First-1][i]);
141          printf(")Available drives\n");
142          printf("Next state ...\n");
143      }
144
```

```
145      int countNULL = 0;
146      for (int i = 0; i < np; ++i)
147          if(state[i] == -1)  // ถ้า state = NULL
148              countNULL +=1;  // ให้นับจำนวน  NULL
149
150      if(countNULL == np)    // ถ้า state เป็น NULL ทั้งหมด
151          printf("\nIf grant last drive to any process may get deadlock = UNSAFE STATE\n**ALL process don't run to comp
152      else if(countNULL > 0 && countNULL < np) {  // ถ้า stateเป็น NULL แค่บางอัน
153          printf("Not enough for process needs remaining(");
154          for (int i = 0; i < np; ++i)
155              if(state[i] == -1)
156                  printf("P%d,",i+1);
157          printf(")\nProcess may get deadlock = UNSAFE STATE\n");
158      }
159      else {    // ถ้า stat ไม่มี NULL เลย
160          printf("Sequences state are ");
161          for (int i = 1; i <=np ; ++i)
162              for (int j = 0; j <np ; ++j)
163                  if(i == state[j])
164                      printf("P%d ",j+1);
165          printf("\n**ALL process run to completion = SAFE STATE\n");
166      }
167
168  }
```

ฟังก์ชัน result() นั้นมีหน้าที่แสดงผลลัพธ์หลังจากการทำงานของอัลกอริทึม Banker's Algorithm เพื่อตรวจสอบว่าสถานะของระบบเป็น "SAFE STATE" หรือ "UNSAFE STATE" โดยคำนวณจากข้อมูลที่ได้จากการทำงานของอัลกอริทึม

การทำงานของฟังก์ชันนี้อธิบายได้ดังนี้:

1. ใช้ printf() เพื่อแสดงข้อความ "Available drives =" ตามด้วยข้อมูลในตัวแปร AvailDrives[] โดยใช้ลูป for เพื่อแสดงข้อมูลทุกตัวแปรในอาร์เรย์ AvailDrives[]
2. กำหนดตัวแปร First เพื่อเก็บค่าของโปรเซสที่ได้รับการจัดสรรทรัพยากรอันดับแรก และทำการวนลูปเพื่อหาโปรเซสแรกที่ได้รับการจัดสรรทรัพยากร

3. ถ้ามีโปรเซสที่ได้รับการจัดสรรทรัพยากรอันดับแรก ให้แสดงข้อความ "First state allocate()" และแสดงจำนวนทรัพยากรที่โปรเซสที่ได้รับการจัดสรรอันดับแรกต้องการ และแสดงโปรเซสที่ได้รับการจัดสรรอันดับแรก และแสดงข้อความ "P completes and releases()" และแสดงจำนวนทรัพยากรที่เหลือหลังจากโปรเซสที่ได้รับการจัดสรรอันดับแรกสิ้นสุด
4. หากมีโปรเซสที่ state เป็น NULL ให้นับจำนวนของโปรเซสที่ state เป็น NULL และถ้าจำนวนนั้นเท่ากับจำนวนโปรเซสทั้งหมด ให้แสดงข้อความ "UNSAFE STATE"
5. ถ้ามีบางโปรเซสที่ state เป็น NULL ให้แสดงข้อความ "Not enough for process needs remaining()" และแสดงรายการของโปรเซสที่เหลือที่มี state เป็น NULL
6. ถ้า state ไม่มี NULL เลย ให้แสดงข้อความ "Sequences state are" และแสดงลำดับของโปรเซสที่ทำงานสำเร็จ
7. สรุปผลลัพธ์ว่าระบบเป็น "SAFE STATE" หรือ "UNSAFE STATE" โดยใช้ printf() ในการแสดงข้อความให้ผู้ใช้ทราบถึงผลลัพธ์และสถานะของระบบ

```
170  int main(){
171      input();   // input value
172      calAvailable();
173      calNeed();
174      Banker();  // banker algorithm
175
176      printf("%d process\n", np);
177      printf("%d resource is ", nr);
178      for (int i = 0; i < nr; ++i)
179          printf("%c(%d instances) ", 65 + i, resource[i]);
180
181      showTable();  //แสดงตาราง
182      result();     //สรุปผลลัพธ์
183  }
184
```

ฟังก์ชัน main() นี้เป็นฟังก์ชันหลักที่ทำหน้าที่เรียกใช้งานฟังก์ชันต่างๆ เพื่อดำเนินการตามลำดับขั้นตอนในการใช้งานของอัลกอริทึม Banker's Algorithm และแสดงผลลัพธ์ออกทางหน้าจอ

การทำงานของฟังก์ชันนี้อธิบายได้ดังนี้:

1. เรียกใช้ฟังก์ชัน input() เพื่อรับข้อมูลเริ่มต้นจำนวนโปรเซสและทรัพยากร และรับข้อมูลการจัดสรรทรัพยากรให้แต่ละโปรเซส
2. เรียกใช้ฟังก์ชัน calAvailable() เพื่อคำนวณหาทรัพยากรที่ใช้งานได้ (Available resources)
3. เรียกใช้ฟังก์ชัน calNeed() เพื่อคำนวณหาทรัพยากรที่โปรเซสต้องการ (Need resources)
4. เรียกใช้ฟังก์ชัน Banker() เพื่อใช้งานอัลกอริทึม Banker's Algorithm เพื่อตรวจสอบว่าระบบอยู่ในสถานะ "SAFE STATE" หรือ "UNSAFE STATE"
5. แสดงข้อมูลพื้นฐานเกี่ยวกับจำนวนโปรเซสและทรัพยากรที่ได้รับเข้ามาในระบบ
6. เรียกใช้ฟังก์ชัน showTable() เพื่อแสดงตารางที่แสดงรายละเอียดของทรัพยากรและการจัดสรรทรัพยากรในแต่ละโปรเซส
7. เรียกใช้ฟังก์ชัน result() เพื่อสรุปผลลัพธ์ว่าระบบอยู่ในสถานะ "SAFE STATE" หรือ "UNSAFE STATE" และแสดงลำดับของโปรเซสที่ทำงานสำเร็จถ้าเป็น "SAFE STATE"

ตารางกำหนดตัวแปรหลักที่ใช้ในโค้ดโปรแกรม

ชื่อตัวแปร	ชนิดข้อมูล	เก็บข้อมูล	หน้าที่
np	int	จำนวนโปรเซส	เก็บจำนวนโปรเซสที่ระบบต้องการจัดการ
nr	int	จำนวนทรัพยากร	เก็บจำนวนทรัพยากรทั้งหมดในระบบ
Allocation	int[10][5]	ข้อมูลการจัดสรรทรัพยากรให้กับโปรเซส	เก็บข้อมูลการจัดสรรทรัพยากรให้กับแต่ละโปรเซส
Max	int[10][5]	ข้อมูลความต้องการทรัพยากรสูงสุดของแต่ละโปรเซส	เก็บข้อมูลความต้องการทรัพยากรสูงสุดของแต่ละโปรเซส
Need	int[10][5]	ข้อมูลความต้องการทรัพยากรที่เหลือให้กับแต่ละโปรเซส	เก็บข้อมูลความต้องการทรัพยากรที่เหลือให้กับแต่ละโปรเซส
Available	int[10][5]	ข้อมูลทรัพยากรที่ใช้งานได้ (Available resources)	เก็บข้อมูลทรัพยากรที่ใช้งานได้ (Available resources) ที่ใช้ในการตรวจสอบ SAFE STATE หรือ UNSAFE STATE
resource	int[5]	ข้อมูลจำนวนทรัพยากรที่มีในระบบ	เก็บข้อมูลจำนวนทรัพยากรที่มีในระบบ
state	int[10]	ข้อมูลสถานะของแต่ละโปรเซส	เก็บข้อมูลสถานะของแต่ละโปรเซสในระบบ
AvailDrives	int[5]	จำนวนทรัพยากรที่ใช้งานได้ (Available drives)	เก็บจำนวนทรัพยากรที่ใช้งานได้ (Available drives)
AvailSUM	int[5]	ผลรวมของทรัพยากรที่ใช้งานได้ (Available sum)	เก็บผลรวมของทรัพยากรที่ใช้งานได้ (Available sum)

ผลลัพธ์ 1

```
->> Input state
Enter number of process :5
Enter number of resource :3
-----------------------------------------------------
Enter all unit of resource :
 resource01 : 10
 resource02 : 5
 resource03 : 7

-----------------------------------------------------
Enter allocation of process : 1
resource01 : 0
resource01 : 1
resource01 : 0

Enter allocation of process : 2
resource02 : 2
resource02 : 0
resource02 : 0

Enter allocation of process : 3
resource03 : 3
resource03 : 0
resource03 : 2

Enter allocation of process : 4
resource04 : 2
resource04 : 1
resource04 : 1

Enter allocation of process : 5
resource05 : 0
resource05 : 0
resource05 : 2
```

```
 ----------------------------------------------------

 Enter max of process : 1
 resource01 : 7
 resource01 : 5
 resource01 : 3

 Enter max of process : 2
 resource02 : 3
 resource02 : 2
 resource02 : 2

 Enter max of process : 3
 resource03 : 9
 resource03 : 0
 resource03 : 2

 Enter max of process : 4
 resource04 : 2
 resource04 : 2
 resource04 : 2

 Enter max of process : 5
 resource05 : 4
 resource05 : 3
 resource05 : 3
 ----------------------------------------------------
```

```
----------------------------------------------------
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
PS D:\ [illegible] 2 66\OS_Lab\lab7>
```

ผลลัพธ์จากการทำงานของโปรแกรม Banker's Algorithm มีการแสดงตารางที่แสดงสถานะของแต่ละโปรเซสตามดัชนีของการประมวลผลที่กำหนดในโปรแกรม แสดงให้เห็นว่าทรัพยากรที่มีอยู่ในแต่ละโปรเซส (Allocation), ทรัพยากรสูงสุดที่โปรเซสต้องการ (Max), จำนวนทรัพยากรที่ต้องการเพิ่มเติม (Need), และจำนวนทรัพยากรที่สามารถใช้งานได้ในขณะนั้น (Available) โดยที่สถานะของแต่ละโปรเซสถูกแสดงด้วยตัวเลขของลำดับที่โปรเซสได้รับทรัพยากรไปแล้ว หรือถูกตั้งให้เป็น NULL หากโปรเซสยังไม่ได้รับทรัพยากรใดๆ ในการลำดับรอบต่างๆ

ในกรณีนี้ โปรเซส P2 ได้รับทรัพยากรไปแล้วเป็นอันดับแรก และจำนวนทรัพยากรที่ต้องการอยู่ใน Need ของ P2 คือ (1, 2, 2) จำนวนทรัพยากรที่มีอยู่หลังจาก P2 ทำงานเสร็จคือ (5, 3, 2) และตามด้วยการทำงานของ P4, P5, P1, และ P3 ตามลำดับ

ผลลัพธ์ 2

```
->> Input state
Enter number of process :5
Enter number of resource :3
-----------------------------------------------------
Enter all unit of resource :
 resource01 : 10
 resource02 : 5
 resource03 : 7

-----------------------------------------------------
Enter allocation of process : 1
resource01 : 0
resource01 : 4
resource01 : 0

Enter allocation of process : 2
resource02 : 2
resource02 : 0
resource02 : 0

Enter allocation of process : 3
resource03 : 3
resource03 : 0
resource03 : 2

Enter allocation of process : 4
resource04 : 2
resource04 : 1
resource04 : 1

Enter allocation of process : 5
resource05 : 0
resource05 : 0
resource05 : 2
```

```
-----------------------------------------------------

Enter max of process : 1
resource01 : 7
resource01 : 5
resource01 : 3

Enter max of process : 2
resource02 : 3
resource02 : 2
resource02 : 2

Enter max of process : 3
resource03 : 9
resource03 : 0
resource03 : 2

Enter max of process : 4
resource04 : 2
resource04 : 2
resource04 : 2

Enter max of process : 5
resource05 : 4
resource05 : 3
resource05 : 3
-----------------------------------------------------
```

```
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  4  0    7  5  3    7  1  3              NULL
P2        2  0  0    3  2  2    1  2  2              NULL
P3        3  0  2    9  0  2    6  0  0              NULL
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
If grant last drive to any process may get deadlock = UNSAFE STATE
**ALL process don't run to completion
PS D:\ภาค 2 66\OS_Lab\lab7>
```

ผลลัพธ์แสดงว่าระบบไม่ปลอดภัย (UNSAFE STATE) เนื่องจากทรัพยากรไม่เพียงพอสำหรับโปรเซสทุกตัว ทำให้ไม่มีโปรเซสใดสามารถทำงานไปจนจบได้โดยไม่ติดล็อค ดังนั้นจึงต้องปรับเปลี่ยนการจัดสรรทรัพยากรใหม่เพื่อป้องกันสถานการณ์นี้ในอนาคต การจัดการทรัพยากรอย่างเหมาะสมและมีการวางแผนรอบคอบเป็นสิ่งสำคัญในการป้องกันปัญหา Deadlock และให้ระบบทำงานได้อย่างมีประสิทธิภาพและปลอดภัย

ผลลัพธ์ที่ 3

```
->> Input state
Enter number of process :5
Enter number of resource :3
-----------------------------------------------------
Enter all unit of resource :
 resource01 : 10
 resource02 : 5
 resource03 : 7

-----------------------------------------------------
Enter allocation of process : 1
resource01 : 0
resource01 : 4
resource01 : 0

Enter allocation of process : 2
resource02 : 2
resource02 : 0
resource02 : 0

Enter allocation of process : 3
resource03 : 3
resource03 : 0
resource03 : 2

Enter allocation of process : 4
resource04 : 2
resource04 : 1
resource04 : 1

Enter allocation of process : 5
resource05 : 0
resource05 : 0
resource05 : 2
-----------------------------------------------------
```

```
-----------------------------------------------------

Enter max of process : 1
resource01 : 7
resource01 : 5
resource01 : 3

Enter max of process : 2
resource02 : 3
resource02 : 2
resource02 : 2

Enter max of process : 3
resource03 : 6
resource03 : 0
resource03 : 0

Enter max of process : 4
resource04 : 2
resource04 : 2
resource04 : 2

Enter max of process : 5
resource05 : 4
resource05 : 3
resource05 : 3
-----------------------------------------------------
```

```
-----------------------------------------------------
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  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:\ৰাম 2 66\OS_Lab\lab7>
```

สถานการณ์:

- มีกระบวนการ (Process) 5 ตัว (P1, P2, P3, P4, P5)
- มีทรัพยากร (Resource) 3 ประเภท (A, B, C) ซึ่งมี A 10 หน่วย, B 5 หน่วย, C 7 หน่วย
- มีการจัดสรรทรัพยากรให้แต่ละกระบวนการไปแล้วบางส่วน
- ทราบความต้องการทรัพยากรสูงสุด (Max) ของแต่ละกระบวนการ

ผลลัพธ์:

- ตารางสรุปสถานการณ์: แสดงการจัดสรรทรัพยากรปัจจุบัน (Alloc), ความต้องการสูงสุด (Max), ความต้องการที่เหลือ (Need), ทรัพยากรว่าง (Avail), และสถานะ (State) ของแต่ละกระบวนการ
- คำอธิบาย: ระบุว่าระบบอยู่ในสถานะ UNSAFE STATE หมายความว่า มีความเป็นไปได้ที่กระบวนการจะเกิด Deadlock (ติดตาย) และไม่สามารถดำเนินการต่อได้

อธิบายเพิ่มเติม:

- ระบบคำนวณความต้องการที่เหลือของแต่ละกระบวนการ (Need = Max - Alloc) และทรัพยากรว่าง (Avail)
- จากนั้น พบว่า ทรัพยากรว่าง ไม่เพียงพอที่จะตอบสนองความต้องการที่เหลือของกระบวนการใดๆ ที่ยังไม่ได้รับการจัดสรรทรัพยากรเต็มที่ (P1, P2, P4, P5)
- แม้ว่า P3 จะสามารถดำเนินการต่อและปล่อยทรัพยากรคืนมาได้ แต่ทรัพยากรที่ปล่อยคืนก็ไม่เพียงพอต่อการปลดล็อกกระบวนการอื่นๆ
- ดังนั้น ระบบจึงอยู่ในสถานะ UNSAFE STATE

สรุป: ในสถานการณ์นี้ มีความเป็นไปได้สูงที่กระบวนการจะเกิด Deadlock จำเป็นต้องมีการจัดการทรัพยากรอย่างระมัดระวังเพื่อป้องกันสถานการณ์ดังกล่าว

สรุปผลการทดลอง

การทดลอง Banker's Algorithm เป็นการจำลองการจัดสรรทรัพยากรในระบบหลายๆ โปรเซสโดยใช้ขั้นตอนและเงื่อนไขตามอัลกอริทึมของ Banker's Algorithm เพื่อให้มั่นใจว่าระบบจะไม่เข้าสู่สถานะที่ไม่ปลอดภัย (UNSAFE STATE) และโปรเซสจะสามารถทำงานได้โดยไม่เกิดปัญหา deadlock หรือสิ่งที่เรียกว่าสถานะปลอดภัย (SAFE STATE)

การทดลองได้แสดงให้เห็นว่า:

1. กรณีที่ทรัพยากรที่มีไม่เพียงพอต่อความต้องการของโปรเซส: แสดงว่าระบบไม่สามารถจัดสรรทรัพยากรให้กับโปรเซสทั้งหมดได้ ทำให้เกิดสถานะที่ไม่ปลอดภัย (UNSAFE STATE) และโปรเซสบางตัวอาจไม่สามารถทำงานได้ตามปกติ
2. กรณีที่ทรัพยากรเพียงพอต่อความต้องการของโปรเซส: แสดงว่าระบบสามารถจัดสรรทรัพยากรให้กับโปรเซสทั้งหมดได้โดยไม่เกิดสถานะที่ไม่ปลอดภัย (SAFE STATE) และโปรเซสทุกตัวสามารถทำงานได้สำเร็จ

สื่อ / เอกสารอ้างอิง

อาจารย์ปิยพล ยืนยงสถาวร: เอกสารประกอบการสอน 7 วงจรอับ (Deadlock)