#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

struct process

{

int A\_allot, A\_max, A\_need;

int B\_allot, B\_max, B\_need;

int C\_allot, C\_max, C\_need;

bool completed;

};

void display(struct process arr[], int p\_num, int a, int b, int c)

{

printf("\nAllot\t\t\t\tMax\t\t\t\tNeed\n");

printf("A\tB\tC\t\tA\tB\tC\t\tA\tB\tC\n");

for (int i = 0; i < p\_num; i++)

{

printf("%d\t%d\t%d\t\t%d\t%d\t%d\t\t%d\t%d\t%d\n", arr[i].A\_allot, arr[i].B\_allot, arr[i].C\_allot, arr[i].A\_max, arr[i].B\_max, arr[i].C\_max, arr[i].A\_need, arr[i].B\_need, arr[i].C\_need);

}

printf("\nAvailable\nA\tB\tC\n%d\t%d\t%d\n", a, b, c);

}

bool isSafe(struct process arr[], int p\_num, int A\_avail, int B\_avail, int C\_avail)

{

int workA = A\_avail, workB = B\_avail, workC = C\_avail;

bool finish[p\_num];

// Initialize finish array

for (int i = 0; i < p\_num; i++)

{

finish[i] = false;

}

int count = 0;

while (count < p\_num)

{

bool found = false;

for (int i = 0; i < p\_num; i++)

{

if (!finish[i] &&

arr[i].A\_need <= workA &&

arr[i].B\_need <= workB &&

arr[i].C\_need <= workC)

{

// Allocate resources to process i

workA += arr[i].A\_allot;

workB += arr[i].B\_allot;

workC += arr[i].C\_allot;

// Update needs

arr[i].A\_need = arr[i].A\_max - arr[i].A\_allot;

arr[i].B\_need = arr[i].B\_max - arr[i].B\_allot;

arr[i].C\_need = arr[i].C\_max - arr[i].C\_allot;

finish[i] = true;

found = true;

count++;

}

}

if (!found)

{

// No process found, system is not in a safe state

return false;

}

}

// If all processes are finished, the system is in a safe state

return true;

}

void bankers(struct process arr[], int p\_num, int A\_avail, int B\_avail, int C\_avail)

{

while (true)

{

bool allocationDone = false;

for (int i = 0; i < p\_num; i++)

{

if (arr[i].A\_need <= A\_avail &&

arr[i].B\_need <= B\_avail &&

arr[i].C\_need <= C\_avail)

{

// Try allocating resources to process i

A\_avail += arr[i].A\_allot;

B\_avail += arr[i].B\_allot;

C\_avail += arr[i].C\_allot;

// Update needs

arr[i].A\_need = arr[i].A\_max - arr[i].A\_allot;

arr[i].B\_need = arr[i].B\_max - arr[i].B\_allot;

arr[i].C\_need = arr[i].C\_max - arr[i].C\_allot;

allocationDone = true;

}

}

if (!allocationDone)

{

// No more allocations possible

break;

}

}

if (isSafe(arr, p\_num, A\_avail, B\_avail, C\_avail))

{

printf("Allocation successful. System is in a safe state.\n");

}

else

{

printf("Allocation failed. System is not in a safe state.\n");

}

}

int main()

{

printf("Enter the number of processes you want: ");

int p\_num;

scanf("%d", &p\_num);

struct process p\_arr[10];

for (int i = 0; i < p\_num; i++)

{

// Input allocation and maximum resources

printf("Enter the A\_allot for process %d: ", i);

scanf("%d", &p\_arr[i].A\_allot);

printf("Enter the B\_allot for process %d: ", i);

scanf("%d", &p\_arr[i].B\_allot);

printf("Enter the C\_allot for process %d: ", i);

scanf("%d", &p\_arr[i].C\_allot);

printf("Enter the A\_max for process %d: ", i);

scanf("%d", &p\_arr[i].A\_max);

printf("Enter the B\_max for process %d: ", i);

scanf("%d", &p\_arr[i].B\_max);

printf("Enter the C\_max for process %d: ", i);

scanf("%d", &p\_arr[i].C\_max);

// Initialize need and completion status

p\_arr[i].A\_need = abs(p\_arr[i].A\_max - p\_arr[i].A\_allot);

p\_arr[i].B\_need = abs(p\_arr[i].B\_max - p\_arr[i].B\_allot);

p\_arr[i].C\_need = abs(p\_arr[i].C\_max - p\_arr[i].C\_allot);

p\_arr[i].completed = false;

}

int A\_avail, B\_avail, C\_avail;

printf("Enter available instances of A: ");

scanf("%d", &A\_avail);

printf("Enter available instances of B: ");

scanf("%d", &B\_avail);

printf("Enter available instances of C: ");

scanf("%d", &C\_avail);

display(p\_arr, p\_num, A\_avail, B\_avail, C\_avail);

// Call bankers function to allocate resources

bankers(p\_arr, p\_num, A\_avail, B\_avail, C\_avail);

display(p\_arr, p\_num, A\_avail, B\_avail, C\_avail);

return 0;

}

OUTPUT

