K.GOWTHAM-192372078

13.Construct a C program for implementation of the various memory allocation strategies.

Aim

To implement various memory allocation strategies in C, including **First Fit**, **Best Fit**, and **Worst Fit**, for allocating memory to processes.

Algorithm

1. **Input**:
   - Memory block sizes.
   - Process sizes.
2. For each process:
   - Apply the selected allocation strategy:
     - **First Fit**: Allocate the first block that fits the process.
     - **Best Fit**: Allocate the smallest block that fits the process.
     - **Worst Fit**: Allocate the largest block that fits the process.
3. Print the allocation results for each process.
4. **Output**: Process allocation details, indicating block numbers or unallocated processes.

Procedure

1. Define arrays for memory blocks and process sizes.
2. Use loops to simulate the allocation based on the chosen strategy.
3. Check block size availability and assign processes to blocks.
4. Print the results showing the process-to-block mapping or "Not Allocated" for unfit processes.

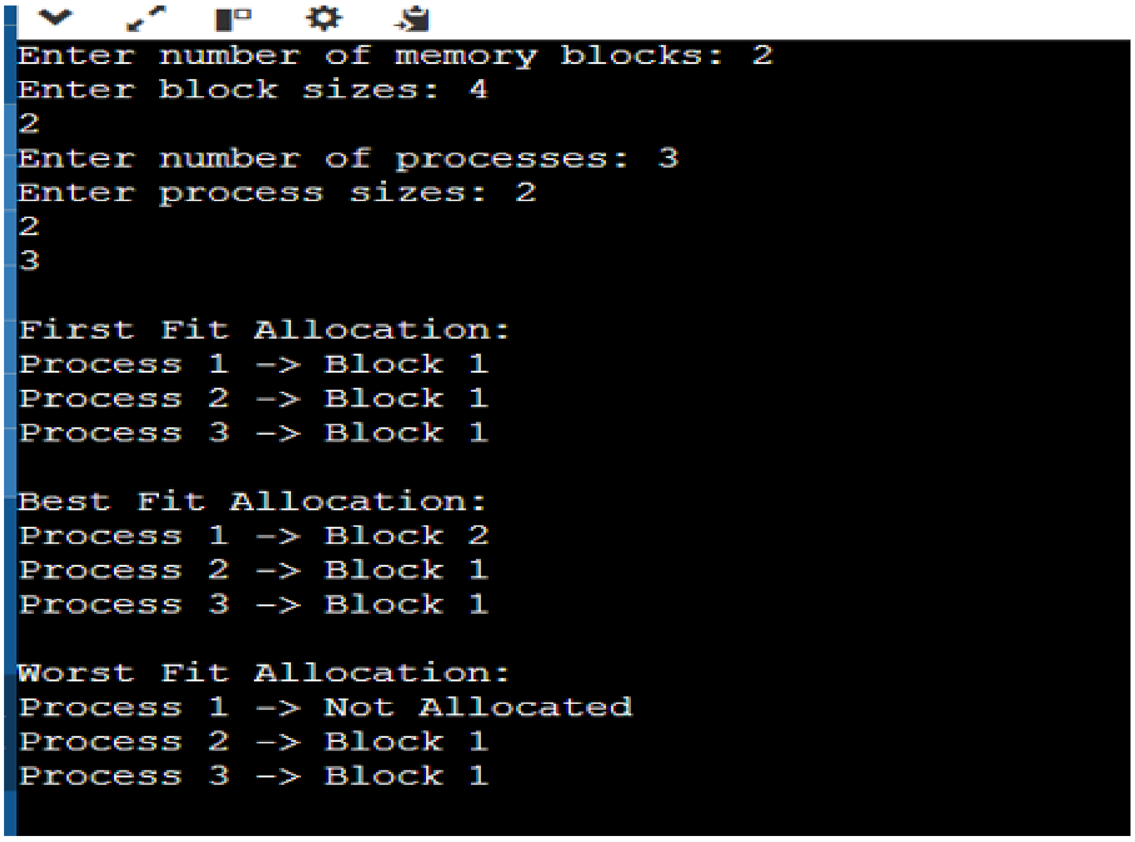
Code:

```
#include <stdio.h>

#define MAX 100



void firstFit(int blockSize[], int blocks, int processSize[], int processes) {

    int allocation[MAX] = {-1};

    for (int i = 0; i < processes; i++) {

        for (int j = 0; j < blocks; j++) {
            if (blockSize[j] >= processSize[i]) {
                allocation[i] = j;
                blockSize[j] -= processSize[i];
                break;
            }
        }
    }
    for (int i = 0; i < processes; i++) {
        if (allocation[i] != -1)
            printf("Process %d -> Block %d\n", i + 1, allocation[i] + 1);
        else
            printf("Process %d -> Not Allocated\n", i + 1);
    }
}

void bestFit(int blockSize[], int blocks, int processSize[], int processes) {
    int allocation[MAX] = {-1};
    for (int i = 0; i < processes; i++) {
        int bestIdx = -1;
        for (int j = 0; j < blocks; j++) {
            if (blockSize[j] >= processSize[i]) {
                if (bestIdx == -1 || blockSize[j] < blockSize[bestIdx])
                    bestIdx = j;
            }
        }
        if (bestIdx != -1) {
            allocation[i] = bestIdx;
            blockSize[bestIdx] -= processSize[i];
        }
    }
    for (int i = 0; i < processes; i++) {
        if (allocation[i] != -1)
            printf("Process %d -> Block %d\n", i + 1, allocation[i] + 1);
        else
            printf("Process %d -> Not Allocated\n", i + 1);
    }
}

void worstFit(int blockSize[], int blocks, int processSize[], int processes) {
    int allocation[MAX] = {-1};
    for (int i = 0; i < processes; i++) {
        int worstIdx = -1;
        for (int j = 0; j < blocks; j++) {
            if (blockSize[j] >= processSize[i]) {
                if (worstIdx == -1 || blockSize[j] > blockSize[worstIdx])
                    worstIdx = j;
            }
        }
        if (worstIdx != -1) {
            allocation[i] = worstIdx;
            blockSize[worstIdx] -= processSize[i];
        }
    }
    for (int i = 0; i < processes; i++) {
        if (allocation[i] != -1)
            printf("Process %d -> Block %d\n", i + 1, allocation[i] + 1);
        else
            printf("Process %d -> Not Allocated\n", i + 1);
    }
}

int main() {
    int blocks, processes;
    int blockSize[MAX], processSize[MAX];
    printf("Enter number of memory blocks: ");
    scanf("%d", &blocks);
    printf("Enter block sizes: ");
    for (int i = 0; i < blocks; i++) scanf("%d", &blockSize[i]);
    printf("Enter number of processes: ");
    scanf("%d", &processes);

    printf("Enter process sizes: ");

    for (int i = 0; i < processes; i++) scanf("%d", &processSize[i]);


    printf("\nFirst Fit Allocation:\n");

    firstFit(blockSize, blocks, processSize, processes);


    printf("\nBest Fit Allocation:\n");

    bestFit(blockSize, blocks, processSize, processes);


    printf("\nWorst Fit Allocation:\n");

    worstFit(blockSize, blocks, processSize, processes);


    return 0;

}
```

Result

1. The program prompts for memory block sizes and process sizes.
2. It outputs the allocation of processes using **First Fit**, **Best Fit**, and **Worst Fit** strategies.
3. Unallocated processes are displayed as "Not Allocated."

Output:



```
Enter number of memory blocks: 2
Enter block sizes: 4
2
Enter number of processes: 3
Enter process sizes: 2
2
3

First Fit Allocation:
Process 1 -> Block 1
Process 2 -> Block 1
Process 3 -> Block 1

Best Fit Allocation:
Process 1 -> Block 2
Process 2 -> Block 1
Process 3 -> Block 1

Worst Fit Allocation:
Process 1 -> Not Allocated
Process 2 -> Block 1
Process 3 -> Block 1
```