Name: K.SaiKrishna

Reg-No: 192311106

22.Construct a C program to implement the best fit algorithm of memory management.

Aim

The aim of the program is to implement the **Best Fit Algorithm** for memory management, which allocates memory blocks to processes such that the process gets the smallest block that fits its requirements.

Algorithm

1. Start by initializing memory blocks and processes with their respective sizes.
2. For each process, find the smallest memory block that can accommodate the process (best fit).
3. Allocate the memory block to the process, and reduce the block size accordingly.
4. If no block can accommodate a process, mark it as unallocated.
5. Display the allocation results.

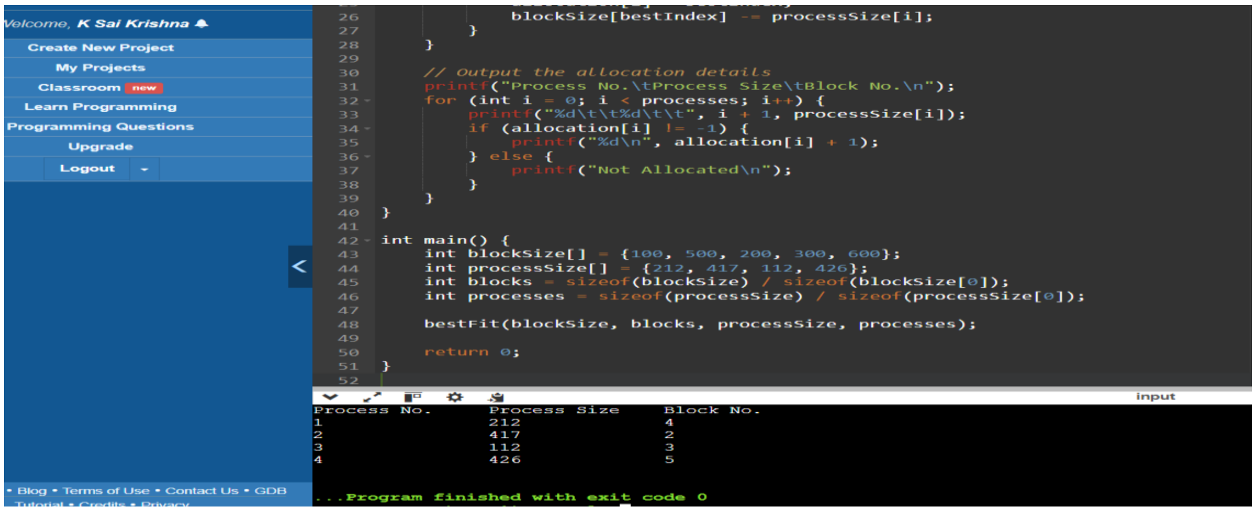
Procedure

1. Input the sizes of memory blocks and processes.
2. Iterate through each process and find the best-fit block (minimum size sufficient for the process).
3. Update the block size after allocation or leave the process unallocated if no block fits.
4. Output the allocation table showing the process, allocated block, and remaining block size.

Code:

```
#include <stdio.h>



int main() {

    int blocks[10], processes[10], allocation[10], n, m, i, j, minBlockIndex, minSize;



    printf("Enter the number of memory blocks: ");

    scanf("%d", &n);
    printf("Enter the size of each memory block:\n");
    for (i = 0; i < n; i++) {
        scanf("%d", &blocks[i]);
    }
printf("Enter the number of processes: ");
    scanf("%d", &m);
    printf("Enter the size of each process:\n");
    for (i = 0; i < m; i++) {
        scanf("%d", &processes[i]);
        allocation[i] = -1;
    }
 for (i = 0; i < m; i++) {
        minBlockIndex = -1;
        minSize = 1e9;
        for (j = 0; j < n; j++) {
            if (blocks[j] >= processes[i] && blocks[j] < minSize) {
                minSize = blocks[j];
                minBlockIndex = j;
            }
        }
        if (minBlockIndex != -1) {
            allocation[i] = minBlockIndex + 1;
            blocks[minBlockIndex] -= processes[i];
        }
    }
 printf("\nProcess\tSize\tBlock Allocated\n");
    for (i = 0; i < m; i++) {
        if (allocation[i] != -1) {
            printf("%d\t%d\t%d\n", i + 1, processes[i], allocation[i]);
        } else {
            printf("%d\t%d\tNot Allocated\n", i + 1, processes[i]);
        }
    }
    return 0;
}
```

Output:



```
                blockSize[bestIndex] -= processSize[i];
            }
        }

        // Output the allocation details
        printf("Process No.\tProcess Size\tBlock No.\n");
        for (int i = 0; i < processes; i++) {
            printf("%d\t\t%d\t\t", i + 1, processSize[i]);
            if (allocation[i] != -1) {
                printf("%d\n", allocation[i] + 1);
            } else {
                printf("Not Allocated\n");
            }
        }
}

int main() {
    int blockSize[] = {100, 500, 200, 300, 600};
    int processSize[] = {212, 417, 112, 426};
    int blocks = sizeof(blockSize) / sizeof(blockSize[0]);
    int processes = sizeof(processSize) / sizeof(processSize[0]);

    bestFit(blockSize, blocks, processSize, processes);

    return 0;
}
```

```
Process No.     Process Size    Block No.
1               212             4
2               417             2
3               112             3
4               426             5

...Program finished with exit code 0
```

Result

- Input:
  - Memory blocks: `100, 500, 200, 300, 600`
  - Processes: `212, 417, 112, 426`