IMPLEMENTATION

```
/*
Roll No         : B21CSB69
Name            : Sreelal V
Experiment No : 5.1
*/

#include<stdio.h>
#include<stdbool.h>

typedef struct memoryBlock{
   int blockId ;
   int size ;
   int allocated ;
   int fragmentedSize ;
}memoryBlock ;

typedef struct processBlock{
   int size ;
   int id ;
}processBlock ;

void displayResult(memoryBlock blocka[], processBlock blockb[], int numberOfProcesses, int
numberOfBlocks){

   int totalFrag = 0 ;
   printf("Block\t  Process ID\tBlock Size\tProcess Size\tMemory Fragmented\n") ;

   for(int i = 0 ; i < numberOfBlocks ; i++){
      if(blocka[i].allocated == -1){

         printf("Block %d\t  Process NIL\t%d\t\tNIL\t\t0\n", blocka[i].blockId, blocka[i].size) ;

      }else{
         printf("Block %d\t  Process %d\t%d\t\t%d\t\t%d\n",
blocka[i].blockId,blockb[blocka[i].allocated].id, blocka[i].size, blockb[blocka[i].allocated].size,
blocka[i].fragmentedSize ) ;
         totalFrag += blocka[i].fragmentedSize ;
      }
   }
   printf("Total Fragmentation = %d\n", totalFrag) ;
}

void firstFit(memoryBlock blocka[], processBlock blockb[], int numberOfProcesses, int
numberOfBlocks){
   for(int i = 0 ; i < numberOfProcesses ; i++){
      int flag = 0 ;

      for(int j = 0 ; j < numberOfBlocks ; j++){
         if(blocka[j].allocated == -1 && blocka[j].size >= blockb[i].size){
            flag = 1 ;
```

```
            blocka[j].allocated = i ;
            blocka[j].fragmentedSize = blocka[j].size - blockb[i].size ;
            break ;
          }
       }

       if(flag == 0){
          printf("The process %d can't be allocated \n", blockb[i].id) ;
       }
    }
    displayResult(blocka, blockb, numberOfProcesses, numberOfBlocks) ;
}

void bestFit(memoryBlock blocka[], processBlock blockb[], int numberOfProcesses, int
numberOfBlocks){
    for(int i = 0 ; i < numberOfProcesses ; i++){
       int minfrag = 100000 ;
       int flag = -1 ;

       for(int j = 0 ; j < numberOfBlocks; j++){
          if(blocka[j].allocated == -1 && blocka[j].size >= blockb[i].size && blocka[j].size -
blockb[i].size < minfrag){
            flag = j ;
            minfrag = blocka[j].size - blockb[i].size ;
          }
       }
       if(flag == -1){
          printf("The process %d can't be allocated \n", blockb[i].id) ;
       }else{
          blocka[flag].allocated = i ;
          blocka[flag].fragmentedSize = minfrag ;
       }
    }
    displayResult(blocka, blockb, numberOfProcesses, numberOfBlocks) ;
}

void worstFit(memoryBlock blocka[], processBlock blockb[], int numberOfProcesses, int
numberOfBlocks){
    for(int i = 0 ; i < numberOfProcesses ; i++){
       int maxfrag = 0 ;
       int flag = -1 ;
       for(int j = 0 ; j < numberOfBlocks; j++){
          if(blocka[j].allocated == -1 && blocka[j].size >= blockb[i].size && blocka[j].size -
blockb[i].size > maxfrag){
            flag = j ;
            maxfrag = blocka[j].size - blockb[i].size ;
          }
       }
       if(flag == -1){
          printf("The process %d can't be allocated \n", blockb[i].id) ;
       }else{
          blocka[flag].allocated = i ;
```

```
            blocka[flag].fragmentedSize = maxfrag ;

        }
    }
    displayResult(blocka, blockb, numberOfProcesses, numberOfBlocks) ;
}

void main(){
    int numberOfProcess, numberOfBlocks ;

    while(true){
        printf("Input Number of memory blocks : ") ;
        scanf("%d", &numberOfBlocks) ;
        printf("Input Number of processes : ") ;
        scanf("%d", &numberOfProcess) ;
        if(numberOfBlocks >= numberOfProcess){
            break ;
        }

        printf("Number of processes greater than available memory blocks.\nEnter again\n") ;

    }

    memoryBlock blocka1[numberOfBlocks], blocka2[numberOfBlocks], blocka3[numberOfBlocks]
;
    processBlock blockb[10] ;
    for(int i = 0 ; i < numberOfBlocks ; i++){
        printf("Enter memory size of block %d : ", i + 1) ;
        scanf("%d", &blocka1[i].size) ;
        blocka1[i].blockId = i + 1 ;
        blocka1[i].allocated = -1 ;
        blocka1[i].fragmentedSize = 0 ;
        blocka2[i] = blocka1[i] ;
        blocka3[i] = blocka1[i] ;

    }

    for (int i = 0; i < numberOfProcess; i++){
        printf("Enter size of process %d : ", i + 1) ;
        scanf("%d", &blockb[i].size) ;
        blockb[i].id = i + 1 ;
    }

    int choice ;
    do{
        printf("1) First Fit\n2) Best Fit\n3) Worst Fit\n4) Exit\n") ;
        printf("Enter Choice : ") ;
        scanf("%d", &choice) ;
        switch(choice){

            case 1:
                printf("-----------------------------------\n") ;
```

```
            printf("First Fit \n") ;
            firstFit(blocka1, blockb, numberOfProcess, numberOfBlocks) ;
            printf("-------------------------------------\n") ;
            break ;
         case 2:

            printf("-------------------------------------\n") ;
            printf("Best Fit \n") ;
            bestFit(blocka2, blockb, numberOfProcess, numberOfBlocks) ;
            printf("-------------------------------------\n") ;
            break ;
         case 3:
            printf("-------------------------------------\n") ;
            printf("Worst Fit \n") ;
            worstFit(blocka3, blockb, numberOfProcess, numberOfBlocks) ;
            printf("-------------------------------------\n") ;
            break ;
         case 4:
            printf("-------------------------------------\n") ;
            printf("Exit Point \n") ;
            printf("-------------------------------------\n") ;
            break ;
         default:

            printf("Invalid Choice\n") ;
            break ;
      }
   } while (choice != 4) ;
}
```

output:

```
Input Number of memory blocks : 5
Input Number of processes : 5
Enter memory size of block 1 : 64
Enter memory size of block 2 : 12
Enter memory size of block 3 : 75
Enter memory size of block 4 : 45
Enter memory size of block 5 : 58
Enter size of process 1 : 36
Enter size of process 2 : 97
Enter size of process 3 : 4
Enter size of process 4 : 34
Enter size of process 5 : 64
1) First Fit
2) Best Fit
3) Worst Fit
4) Exit
Enter Choice : 1
-------------------------------------
First Fit
The process 2 can't be allocated
The process 5 can't be allocated
```

```
Block     Process ID    Block Size      Process Size    Memory Fragmented
Block 1   Process 1     64              36              28
Block 2   Process 3     12              4               8
Block 3   Process 4     75              34              41
Block 4   Process NIL   45              NIL             0
Block 5   Process NIL   58              NIL             0
Total Fragmentation = 77
-------------------------------------
1) First Fit
2) Best Fit
3) Worst Fit
4) Exit
Enter Choice : 2
-------------------------------------
Best Fit
The process 2 can't be allocated
Block     Process ID    Block Size      Process Size    Memory Fragmented
Block 1   Process 5     64              64              0
Block 2   Process 3     12              4               8
Block 3   Process NIL   75              NIL             0
Block 4   Process 1     45              36              9
Block 5   Process 4     58              34              24
Total Fragmentation = 41
-------------------------------------
1) First Fit
2) Best Fit
3) Worst Fit
4) Exit
Enter Choice : 3
-------------------------------------
Worst Fit
The process 2 can't be allocated
The process 5 can't be allocated
Block     Process ID    Block Size      Process Size    Memory Fragmented
Block 1   Process 3     64              4               60
Block 2   Process NIL   12              NIL             0
Block 3   Process 1     75              36              39
Block 4   Process NIL   45              NIL             0
Block 5   Process 4     58              34              24
Total Fragmentation = 123
-------------------------------------
1) First Fit
2) Best Fit
3) Worst Fit
4) Exit
Enter Choice : 4
-------------------------------------
Exit Point
-------------------------------------
```