```
/* 2) Consider a set of memory partitions and a set of processes. A partition can hold only
      a single process contiguously. Compare various memory allocation strategies with
      reference to external and internal fragmentation. */

// Physical Memory, Main Memory, RAM : holds OS and user programs
// Memory partitions can be fixed or variable

// Placement algorithms , Memory allocation strategies : How to assign processes to memory

// Fixed Partitioning : Physical memory is broken up into fixed partitions
// A process may be loaded into a partition of equal or greater size
// A small process may occupy a full partition : hence Internal fragmentation
// Memory in a partition not used by a process is not available to other processes

// Dynamic Partitioning : Physical memory is broken up into variable sized partitions
// Allocate just enough for process, hence external fragmentation
// Job loading and unloading produces empty holes scattered throughout memory

// Problem definition asks loading all process's pages into available frames contiguously
// Memory allocation strategies with external and internal fragmentation : are Fixed and
//   Dynamic Partitioning respectively

// Placement algorithms : How  to  assign  processes  to memory - how to plug the holes
// When it is time to load or swap a process into main memory, and if there is more than one
//   free block of memory of sufficient size, then the operating system must decide which free
//   block to allocate

// Memory allocation strategies - First , Best , Worst, Next fit
// Choosing among free blocks of main memory that are equal to or larger than the process
//   to be brought in

// First-fit begins to scan memory from the beginning and chooses the first available
//   block that is large enough - ಸಿಕ್ಕಿದ್ ಶಿವಯಾ ನಮಃ

// Best-fit chooses the block that is closest in size to the request
// Worst-fit allocate the largest  block
// Next-fit begins to scan memory from the location of the last placement, and chooses
//   the next available block that is large enough

// If M is Mega byte , F is Free and A is allocated Block respectively, then
// Consider memory configuration after a number of placement and swapping-out operations
//   F:8M , A ,  F:12M ,   A , F:22M , A , F:18M , A , A , F:8M , A , F:6M , A , F:14M , A ,
//   F:36M

// Draw block diagram to visualize above memory configuration

#include <stdio.h>
#include <string.h>
int blockSize[] = {8, 12, 22, 18, 8, 6, 14, 36}; // List of free blocks ; also try 150 , 350
int processSize[] = {16, 2, 8, 4, 1, 32};     // List of process size ; also try 300, 25, 125, 50
int numberOfBlocks = sizeof(blockSize)/sizeof(blockSize[0]);
int numberOfProcess = sizeof(processSize)/sizeof(processSize[0]);

int isBlockAllocated( int j, int allocation[]) // if jth block is allocated return 1 else 0
```

```
{
  for( int i=0 ; i<numberOfProcess; i++ )
     if( allocation[i] == j )
       return 1;

  return 0;
}

void printPlacement( const char *placementAlgorithm , int allocation[])
{ // print process and allocated blocks
  printf("\n\n Using %s Memory allocation strategy", placementAlgorithm);
  printf("\n Process No.\t Process Size\t Block No.\t Block Size");
  for (int i = 0; i < numberOfProcess; i++)
   {
    printf("\n\t %d \t\t %d",i+1, processSize[i]);
    if (allocation[i] != -1)
       printf("\t\t %d \t %d", allocation[i]+1 , blockSize[allocation[i]]);
    else
       printf("\t Not Allocated");
  }
}

void firstFit() // Function to allocate process to memory blocks as per First fit algorithm
{     // ಸಿಕ್ಕಿದ್ ಶಿವಾ
  int allocation[numberOfProcess];// Stores block id of the block allocated to a process
  memset(allocation, -1, sizeof(allocation));// Initially no block is assigned to any process
  for (int i=0; i<numberOfProcess; i++) // Pick each process and
   {  // find suitable blocks according to its size and assign to it
     for (int j=0; j<numberOfBlocks; j++)
      {
        if ( ( isBlockAllocated(j,allocation)==0 ) && (blockSize[j] >= processSize[i]) )
         {
           allocation[i] = j; // Allocate block j to p[i] process
           j=numberOfBlocks; // To break out of inner for loop as block's allocated
         }
      }
   }
  printPlacement( "First Fit" , allocation );
}

void bestFit()// Function to allocate memory to blocks as per Best fit algorithm
{
  int allocation[numberOfProcess];// Stores block id of the block allocated to a process
  memset(allocation, -1, sizeof(allocation));// Initially no block is assigned to any process
  for (int i=0; i<numberOfProcess; i++) // pick each process and find suitable blocks
   { // according to its size and assign to it
     int bestIdx = -1;
     for (int j=0; j<numberOfBlocks; j++)
      { // Find the best fit block for current process
        if ( ( isBlockAllocated(j,allocation)==0 ) && blockSize[j] >= processSize[i])
         {
          if (bestIdx == -1)
            bestIdx = j;
          else if (blockSize[bestIdx] > blockSize[j])
```

```
                bestIdx = j;
            }
        }
        if (bestIdx != -1) // If best block is found for current process
        {
          allocation[i] = bestIdx; // allocate block j to p[i] process
        }
    }
    printPlacement( "Best Fit" , allocation );
}

void worstFit()// Function to allocate memory to blocks as per Worst fit algorithm
{  // Dynamic Partitioning
    int allocation[numberOfProcess];// Stores block id of the block allocated to a process
    memset(allocation, -1, sizeof(allocation));// Initially no block is assigned to any process

   printf("\n\n Using Dynamic Partitioning, Worst Fit Memory allocation strategy");
   printf("\n Process No.\t Process Size\t Block No. \t Block Size \t Allocated Size\t Remainin
g Block Size");

    for (int i=0; i<numberOfProcess; i++) // pick each process and find suitable blocks
    { // according to its size and assign to it, then update remaining block size
      int worstIdx = -1;
      printf("\n\t %d \t\t %d",i+1, processSize[i]);
      for (int j=0; j<numberOfBlocks; j++)
      { // Find the worst fit block for current process
        if ( blockSize[j] >= processSize[i] )
        {
          if (worstIdx == -1)
            worstIdx = j;
          else if (blockSize[worstIdx] < blockSize[j])
            worstIdx = j;
        }
      }
      if (worstIdx != -1) // If worst block is found for current process
      {
        allocation[i] = worstIdx; // allocate block worstIdx to process i
        printf("\t\t %d \t\t %d \t\t %d ",allocation[i]+1,blockSize[allocation[i]],processSize[i]);
        // And because its Dynamic Partitioning - Reduce available memory in this block
        blockSize[worstIdx] -= processSize[i];
        printf("\t\t %d ", blockSize[allocation[i]]);
      }
      else
        printf("\t Not Allocated");
    }
}

int main()       // What is the (asymptotic) runtime of each algorithm implementation
{           //   and better solutions ?
   firstFit();  // Fixed Partitioning
   bestFit();   // Fixed Partitioning
   worstFit();  // Dynamic Partitioning

   return 0 ;
```

} // Advantages and Disadvantages of Static, Dynamic - First , Best , Worst, Next fit ?

/* 1. Output

Fixed Partitioning

Using First Fit Memory allocation strategy

Process No.	Process Size	Block No.	Block Size
1	16	3	22
2	2	1	8
3	8	2	12
4	4	4	18
5	1	5	8
6	32	8	36

Using Best Fit Memory allocation strategy

Process No.	Process Size	Block No.	Block Size
1	16	4	18
2	2	6	6
3	8	1	8
4	4	5	8
5	1	2	12
6	32	8	36

Using Worst Fit Memory allocation strategy

Process No.	Process Size	Block No.	Block Size
1	16	8	36
2	2	3	22
3	8	4	18
4	4	7	14
5	1	2	12
6	32	Not Allocated	

2. Output - Dynamic Partitioning - Worst Fit

Using Dynamic Partitioning, Worst Fit Memory allocation strategy

Process No.	Process Size	Block No.	Block Size	Allocated Size	Remaining Block Size
1	16	8	36	16	20
2	2	3	22	2	20
3	8	3	20	8	12
4	4	8	20	4	16
5	1	4	18	1	17
6	32	Not Allocated			

*/

/* Dynamic Partitioning, Example: Consider the requests from processes in given order 300K, 25K, 125K and 50K. Let there be two blocks of memory available of size 150K followed by a block size 350K.
Compare Static/Dynamic Best and First Fit respectively
*/

// If ER diagram , then
// Fixed Partitioning : Process in 1:1 Relationship with Blocks
// Dynamic Partitioning : Process in N:1 Relationship with Blocks
// Participation ?