```
#include <stdio.h>
#include <stdlib.h>
#define max 25



 struct process
{
  int pid;
  int size;
};

struct block
{
  int size;
};

int main()
{//start main

printf("******* MEMORY MANAGER PROGRAM ********\n");
printf("******* @author: Kerri McMahon ********\n");
printf("*******    Assumptions.        ********\n");
printf("        Ready Queue Limit: 10\n");
printf("        Memory Size: Whatever the sum\n" );
printf("                      your blocks is.\n");
printf("        Job size: < 1000.\n");
printf("        Please be easy on me this was hard. :(\n");
printf("****************************************\n\n");


int running =0;
int a;

while(running !=9)

{//start while
printf("Pick an algorithm. press 1 for FF, 2 for BF, 3 for WF -- Press 9 to quit\n");
scanf("%d", &a);

int ff();
void bf();
void wf();


if (a==9)
{
exit(0);
}

if(a==1)
{
ff();

}
 if (a == 2)
{
bf();
}
if (a == 3)
{
wf();
}
} //end while


return 0;

}//end main


int ff()
{       //max processes is 10

        struct block blocks[10];
        struct process files[10];
    static int block_arr[10], file_arr[10];
      int fragments[10];// blocks[10], files[10];
      int m, n, number_of_blocks, number_of_files, temp;
      printf("\nEnter the Total Number of Blocks:\t");
      scanf("%d", &number_of_blocks);
      printf("Enter the Total Number of Files:\t");
      scanf("%d", &number_of_files);
      printf("\nEnter the Size of the Blocks:\n");
      for(m = 0; m < number_of_blocks; m++)
      {
            printf("Block No.[%d]:\t", m );
            scanf("%d", &blocks[m].size);
      }
      printf("Enter the PID for each process:\n");
      for (m =0; m <number_of_files;m++)
      {
          printf("PID: ");
          scanf("%d", &files[m].pid);
      }
      printf("Enter the Size of the Files:\n");
      for(m = 0; m < number_of_files; m++)
      {
            printf("File No.[%d]:\t", files[m].pid );
            scanf("%d", &files[m].size);
      }
      for(m = 0; m < number_of_files; m++)
      {
            for(n = 0; n < number_of_blocks; n++)
            {
                  if(block_arr[n] != 1)
                  {
                        temp = blocks[n].size - files[m].size;
                        if(temp >= 0)
                        {
                              file_arr[m] = n;
                              break;
                        }
                  }
            }
            fragments[m] = temp;
            block_arr[file_arr[m]] = 1;
      }
      printf("************** Memory ***********");
      printf("\nFile Number\tBlock Number\tFile Size\tBlock Size\tFragment");
      printf("");
      for(m = 0; m < number_of_files; m++)
      {
          if (fragments[m]<0)// || fragments[m] >1000 )
          {
              continue;
          }
            printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", files[m].pid, file_arr[m],
files[m].size, blocks[file_arr[m]].size, fragments[m]);
      }
      printf("\n\n\n");

      int b =0;
      while (b != 9)
      {
          printf("1. Remove a Process\n2.Waiting Queue\n9. Back to Main Menu\n");
          scanf("%d", &b);
            if (b==9)
            {
                memset(files,0,10);
                memset(file_arr,0,10);
                memset(blocks,0,10);
                memset(fragments,0,10);
                fflush(stdin);

                return 0;
            }
          if(b==1)
          {

              int remove;
              printf("Which process number to remove?");
              scanf("%d", &remove);
                printf("********* Memory - Removed process ********");
                printf("\nFile Number\tBlock Number\tFile Size\tBlock
Size\tFragment");
              for(m = 0; m < number_of_files; m++)
                {
                    //Establishing conditions for when a block is freed.
                    if(remove ==  files[m].pid)
                    {
                        continue;
                     //99 indicates a freed slot
                     //printf("\nin if\n");
                     //file_arr[m]=999;
                     //fragments[m] =  blocks[file_arr[m]].size
                    }
                    if (fragments[m]<0 || fragments[m] >999)
                    continue;
                printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", files[m].pid, file_arr[m],
files[m].size, blocks[file_arr[m]].size, fragments[m]);
                 }
                printf("\n\n\n");



          }//end if

        if(b==2)
        {

             //waiting queue

            printf("********** Waiting queue ***********");
             printf("\nFile Number\tBlock Number\tFile Size\tBlock Size\tFragment");
              for(m = 0; m < number_of_files; m++)
```

```
                {
                    //Establishing conditions for when a block is freed.
                    if(remove ==  files[m].pid)
                    {
                        continue;
                     //99 indicates a freed slot
                     //printf("\nin if\n");
                     //file_arr[m]=999;
                     //fragments[m] =  blocks[file_arr[m]].size
                    }
                    if(fragments[m]>0 || files[m].size < blocks[m].size)
                    continue;
                printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", files[m].pid, file_arr[m],
files[m].size, blocks[file_arr[m]].size, fragments[m]);

                 }
                printf("\n\n\n");





        }//end waiting queue


        if(b==3)
        {
                int add;
        printf("Which PID to add?");
        scanf("%d", &add);


                printf("********** Updated Memory **********");

         printf("\nFile Number\tBlock Number\tFile Size\tBlock Size\tFragment");
              for(m = 0; m < number_of_files; m++)
                {
                    //Establishing conditions for when a block is freed.
                    if(remove ==  files[m].pid)
                    {
                        continue;
                     //99 indicates a freed slot
                     //printf("\nin if\n");
                     //file_arr[m]=999;
                     //fragments[m] =  blocks[file_arr[m]].size
                    }



                    if(fragments[m]>0)
                    continue;
                    if(add==files[m].pid)
                    {
                printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", files[m].pid, file_arr[m],
files[m].size, blocks[file_arr[m]].size, 0);
                    }
                printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", files[m].pid, file_arr[m],
files[m].size, blocks[file_arr[m]].size, 0);

                 }
                printf("\n\n\n");
        }









      }//end while
      return 0;

}// end ff

void bf()
{

   int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;
 static int bf[max],ff[max],fragi = 0;

 printf("\n\tMemory Management Scheme - Best Fit");
 printf("\nEnter the number of blocks:");
 scanf("%d",&nb);
 printf("Enter the number of files:");
 scanf("%d",&nf);
 printf("\nEnter the size of the blocks:-\n");
 for(i=1;i<=nb;i++) {
   printf("Block %d:",i);
   scanf("%d",&b[i]);
   ff[i] = i;
 }
 printf("Enter the size of the Processes :-\n");

 for(i=1;i<=nf;i++) {
   printf("Process %d:",i);
   scanf("%d",&f[i]);
 }
int y,m,z,temp1,flag;
for(y=1;y<=nb;y++)
  {
    for(z=y;z<=nb;z++)
    {
      if(b[y]>b[z])
      {
        temp=b[y];
        b[y]=b[z];
        b[z]=temp;
        temp1=ff[y];
        ff[y]=ff[z];
        ff[z]=temp1;
      }
    }
  }
  int flagn[max];
  int fragx = 0;

    printf("\n\nProcess_No\tProcess_Size\tBlock_No\tBlock_Size\tFragment\n");
    for(i=1;i<=nf;i++)
    {
      flag = 1;
      for(j=1;j<=nb;j++)
      {
        if(f[i] <= b[j]){
          flagn[j] = 1;
          printf("%-15d\t%-15d\t%-15d\t%-15d\t",i, f[i],ff[j],b[j]);
          b[j] = b[j] - f[i];
          fragi = fragi + b[j];
          printf("%-15d\n",b[j]);
          break;
        }
        else
        {flagn[j] = 0;
        flag++;
        }
      }
      if(flag > nb)
      printf("%-15d\t%-15d\t%-15s\t%-15s\t%-15s\n",i,
f[i],"WAIT...","WAIT...","WAIT...");
    }
    printf("Internal Fragmentation = %d",fragi );
    for (j= 1; j <=nb ; j++) {
      if (flagn[j] != 1)
          fragx = fragx + b[j];
            /* code */
    }
    printf("\nExternal Fragmentation = %d\n",fragx);

}//end bf


void wf()
{
  int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;
        static int bf[max],ff[max];int flag,fragi = 0;

        printf("\n\tMemory Management Scheme - Worst Fit");
        printf("\nEnter the number of blocks:");
        scanf("%d",&nb);
        printf("Enter the number of Process:");
        scanf("%d",&nf);
        printf("\nEnter the size of the blocks:-\n");
        for(i=1;i<=nb;i++) {
                printf("Block %d:",i);
                scanf("%d",&b[i]);
                ff[i] = i;
        }
        printf("Enter the size of the Processes :-\n");

        for(i=1;i<=nf;i++) {
                printf("Process %d:",i);
                scanf("%d",&f[i]);
        }
        int y,z,temp1;
        /*sorting for worst and best fit only*/
        for(y=1;y<=nb;y++)
        {
                for(z=y;z<=nb;z++)
                {
                        if(b[y]<b[z])    /*change < to > for best fit*/
                        {
                                temp=b[y];
                                b[y]=b[z];
                                b[z]=temp;
                                temp1=ff[y];
                                ff[y]=ff[z];
                                ff[z]=temp1;
                        }
          }
        }
        int flagn[max];
        int fragx = 0;
        /*Following is the code of next fit*/
          printf("\n\nProcess_No\tProcess_Size\tBlock_No\tBlock_Size\tFragment\n");
          for(i=1;i<=nf;i++)
          {
            flag = 1;
            for(j=1;j<=nb;j++)
            {
              if(f[i] <= b[j]){
                                        flagn[j] = 1;
                printf("%-15d\t%-15d\t%-15d\t%-15d\t",i, f[i],ff[j],b[j]);
                b[j] = b[j] - f[i];
                                        fragi = fragi + b[j];
                printf("%-15d\n",b[j]);
                break;
              }
              else
                                {flagn[j] = 0;
              flag++;
                                }
            }
            if(flag > nb)
            printf("%-15d\t%-15d\t%-15s\t%-15s\t%-15s\n",i,
f[i],"WAIT...","WAIT...","WAIT...");
          }
                printf("Internal Fragmentation = %d",fragi );
                for (j= 1; j <=nb ; j++) {
                        if (flagn[j] != 1)
                                        fragx = fragx + b[j];
                                                /* code */
                }
                printf("\nExternal Fragmentation = %d\n",fragx);/*next fit ends*/


}
```