```
1. #include<stdio.h>
void main()
{
int bsize[10], psize[10], bno, pno, flags[10], allocation[10], i, j;

for(i = 0; i < 10; i++)
{
flags[i] = 0;
allocation[i] = -1;
}
printf("Enter no. of blocks: ");
scanf("%d", &bno);
printf("\nEnter size of each block: ");
for(i = 0; i < bno; i++)
scanf("%d", &bsize[i]);

printf("\nEnter no. of processes: ");
scanf("%d", &pno);
printf("\nEnter size of each process: ")
Counting basic algorithm
;
for(i = 0; i < pno; i++)
scanf("%d", &psize[i]);
for(i = 0; i < pno; i++)        //allocation as per first fit
for(j = 0; j < bno; j++)
if(flags[j] == 0 && bsize[j] >= psize[i])
{
allocation[j] = i;
flags[j] = 1;
break;
}
//display allocation details
printf("\nBlock no.\tsize\t\tprocess no.\t\tsize");
for(i = 0; i < bno; i++)
{
printf("\n%d\t\t%d\t\t", i+1, bsize[i]);
if(flags[i] == 1)
printf("%d\t\t\t%d",allocation[i]+1,psize[allocation[i]]);
else
printf("Not allocated");
}
}
```

---

2. // C Program for Worst Fit

```
#include <stdio.h>

void implimentWorstFit(int blockSize[], int blocks, int processSize[], int processes)
{
  // This will store the block id of the allocated block to a process
  int allocation[processes];

  // initially assigning -1 to all allocation indexes
  // means nothing is allocated currently
  for(int i = 0; i < processes; i++){
    allocation[i] = -1;
  }

  // pick each process and find suitable blocks
  // according to its size ad assign to it
  for (int i=0; i<processes; i++)
  {

    int indexPlaced = -1;
    for (int j=0; j<blocks; j++)
    {
      if (blockSize[j] >= processSize[i])
      {
        // place it at the first block fit to accomodate process
        if (indexPlaced == -1)
          indexPlaced = j;

        // if any future block is larger than the current block where
        // process is placed, change the block and thus indexPlaced
        else if (blockSize[indexPlaced] < blockSize[j])
          indexPlaced = j;
      }
    }

    // If we were successfully able to find block for the process
    if (indexPlaced != -1)
    {
      // allocate this block j to process p[i]
      allocation[i] = indexPlaced;

      // Reduce available memory for the block
      blockSize[indexPlaced] -= processSize[i];
    }
  }

  printf("\nProcess No.\tProcess Size\tBlock no.\n");
```

```
    for (int i = 0; i < processes; i++)
    {
        printf("%d \t\t\t %d \t\t\t", i+1, processSize[i]);
        if (allocation[i] != -1)
            printf("%d\n",allocation[i] + 1);
        else
            printf("Not Allocated\n");
    }
}

// Driver code
int main()
{
    int blockSize[] = {5, 4, 3, 6, 7};
    int processSize[] = {1, 3, 5, 3};
    int blocks = sizeof(blockSize)/sizeof(blockSize[0]);
    int processes = sizeof(processSize)/sizeof(processSize[0]);

    implimentWorstFit(blockSize, blocks, processSize, processes);

    return 0 ;
}
```

---

3. #include<stdio.h>

```
void main()
{
int fragment[20],b[20],p[20],i,j,nb,np,temp,lowest=9999;
static int barray[20],parray[20];
printf("\n\t\t\tMemory Management Scheme - Best Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of processes:");
scanf("%d",&np);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
    {
printf("Block no.%d:",i);
        scanf("%d",&b[i]);
    }
printf("\nEnter the size of the processes :-\n");
for(i=1;i<=np;i
```

Maximizing System Performance with Wise Memory Optimizer

```
++)
```

```
   {
      printf("Process no.%d:",i);
      scanf("%d",&p[i]);
   }
for(i=1;i<=np;i++)
{
for(j=1;j<=nb;j++)
{
if(barray[j]!=1)
{
temp=b[j]-p[i];
if(temp>=0)
if(lowest>temp)
{
parray[i]=j;
lowest=temp;
}
}
}
fragment[i]=lowest;
barray[parray[i]]=1;
lowest=10000;
}
printf("\nProcess_no\tProcess_size\tBlock_no\tBlock_size\tFragment");
for(i=1;i<=np && parray[i]!=0;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,p[i],parray[i],b[parray[i]],fragment[i]);
}
```

```
4. #include<stdio.h>
main()
{
          int n,a[10],b[10],t[10],w[10],g[10],i,m;
          float att=0,awt=0;
           for(i=0;i<10;i++)
           {
                      a[i]=0; b[i]=0; w[i]=0; g[i]=0;
           }
          printf("enter the number of process");
           scanf("%d",&n);
          printf("enter the burst times");
           for(i=0;i<n;i++)
```

```
              scanf("%d",&b[i]);
   printf("\nenter the arrival times");
          for(i=0;i<n;i++)
              scanf("%d",&a[i]);
             g[0]=0;
           for(i=0;i<10;i++)
                g[i+1]=g[i]+b[i];
           for(i=0;i<n;i++)
          {
         w[i]=g[i]-a[i];
                    t[i]=g[i+1]-a[i];
                    awt=awt+w[i];
                    att=att+t[i];
          }
    awt =awt/n;
          att=att/n;
          printf("\n\tprocess\twaiting time\tturn arround time\n");
          for(i=0;i<n;i++)
          {
                    printf("\tp%d\t\t%d\t\t%d\n",i,w[i],t[i]);
          }
         printf("the average waiting time is %f\n",awt);
         printf("the average turn around time is %f\n",att);
}
```

OUTPUT:

```
enter the number of process 4
enter the burst times
4  9  8  3
enter the arrival times
0  2  4  3
       process        waiting time     turn arround time
          p0                 0                  4
          p1                 2                 11
          p2                 9                 17
          p3                18                 21
the average waiting time is 7.250000
the average turn around time is 13.250000
```

```
5. #include <stdio.h>
#include <conio.h>
#include <stdlib.h>

int files[50], indexBlock[50], indBlock, n;
void recurse1();
void recurse2();

void recurse1(){
    printf("Enter the index block: ");
    scanf("%d", &indBlock);
    if (files[indBlock] != 1){
        printf("Enter the number of blocks and the number of
files needed for the index %d on the disk: ", indBlock);
        scanf("%d", &n);
    }
    else{
        printf("%d is already allocated\n", indBlock);
        recurse1();
    }
    recurse2();
}

void recurse2(){
    int ch;
    int flag = 0;
    for (int i=0; i<n; i++){
        scanf("%d", &indexBlock[i]);
        if (files[indexBlock[i]] == 0)
            flag++;
    }
    if (flag == n){
        for (int j=0; j<n; j++){
            files[indexBlock[j]] = 1;
        }
        printf("Allocated\n");
        printf("File Indexed\n");
        for (int k=0; k<n; k++){
            printf("%d ------> %d : %d\n", indBlock,
indexBlock[k], files[indexBlock[k]]);
        }
    }
    else{
        printf("File in the index is already allocated\n");
        printf("Enter another indexed file\n");
        recurse2();
    }
    printf("Do you want to enter more files?\n");
    printf("Enter 1 for Yes, Enter 0 for No: ");
    scanf("%d", &ch);
    if (ch == 1)
        recurse1();
    else
        exit(0);
    return;
}

int main()
{
    for(int i=0;i<50;i++)
        files[i]=0;

    recurse1();
    return 0;
}
```

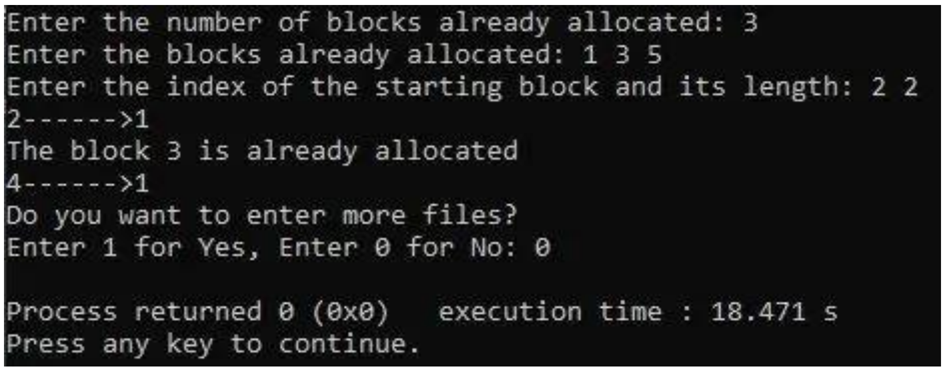
```
7. #include<stdio.h>
#include<unistd.h>

int main() {
  int pipefds[2];
  int returnstatus;
  char writemessages[2][20]={"Hi", "Hello"};
  char readmessage[20];
  returnstatus = pipe(pipefds);

  if (returnstatus == -1) {
    printf("Unable to create pipe\n");
    return 1;
  }
  printf("Writing to pipe - Message 1 is %s\n", writemessages[0]);
  write(pipefds[1], writemessages[0], sizeof(writemessages[0]));
  read(pipefds[0], readmessage, sizeof(readmessage));
  printf("Reading from pipe – Message 1 is %s\n", readmessage);
  printf("Writing to pipe - Message 2 is %s\n", writemessages[0]);
  write(pipefds[1], writemessages[1], sizeof(writemessages[0]));
  read(pipefds[0], readmessage, sizeof(readmessage));
  printf("Reading from pipe – Message 2 is %s\n", readmessage);
  return 0;
}
```

```
Writing to pipe - Message 1 is Hi
Reading from pipe – Message 1 is Hi
Writing to pipe - Message 2 is Hi
Reading from pipe – Message 2 is Hell
```

8.
```
void Producer(){
   while(true){
      // producer produces an item/data
      wait(Empty);
      wait(mutex);
      add();
      signal(mutex);
      signal(Full);
   }
}
void Producer(){
   while(true){
      // producer produces an item/data
      wait(Empty);
      wait(mutex);
      add();
      signal(mutex);
      signal(Full);
   }
}
```

9.
```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

void recursivePart(int pages[]){
   int st, len, k, c, j;
```

```
    printf("Enter the index of the starting block and its length: ");
    scanf("%d%d", &st, &len);
    k = len;
    if (pages[st] == 0){
        for (j = st; j < (st + k); j++){
            if (pages[j] == 0){
                pages[j] = 1;
                printf("%d------>%d\n", j, pages[j]);
            }
            else {
                printf("The block %d is already allocated \n", j);
                k++;
            }
        }
    }
    else
        printf("The block %d is already allocated \n", st);
    printf("Do you want to enter more files? \n");
    printf("Enter 1 for Yes, Enter 0 for No: ");
    scanf("%d", &c);
    if (c==1)
        recursivePart(pages);
    else
        exit(0);
    return;
}

int main(){
    int pages[50], p, a;

    for (int i = 0; i < 50; i++)
        pages[i] = 0;
    printf("Enter the number of blocks already allocated: ");
    scanf("%d", &p);
    printf("Enter the blocks already allocated: ");
    for (int i = 0; i < p; i++){
        scanf("%d", &a);
        pages[a] = 1;
    }

    recursivePart(pages);
    getch();
    return 0;
}
```



```
Enter the number of blocks already allocated: 3
Enter the blocks already allocated: 1 3 5
Enter the index of the starting block and its length: 2 2
2------>1
The block 3 is already allocated
4------>1
Do you want to enter more files?
Enter 1 for Yes, Enter 0 for No: 0

Process returned 0 (0x0)   execution time : 18.471 s
Press any key to continue.
```