Ex.No:11 Implementation of the following Memory Allocation Methods for fixed partition

Date :

Aim:

To write a program to implement memory allocation method for fixed partition using first fit, worst fit,best fit algorithms.

Algorithm:

1. Start the process.
2. Declare the size.
3. Get the number of processes to be inserted.
4. For first fit
   a. Allocate the first hole that is big enough for searching.
   b. Start from the beginning set of holes.
   c. If not start at the hole, which is sharing the previous first fit search end.
   d. Compare the hole.
   e. If large enough, then stop searching in the procedure.
5. For Worst Fit

a. Allocate the **largest free hole** available in the memory that is sufficient enough to hold the process within the system.

b. Search the complete memory for available free partitions

c. Allocate the process to the memory partition which is the largest out of all.

6. For best fit

   a. Allocate the best hole that is small enough for searching.

   b. Start at the best of the set of holes.

   c. If not start at the hole, which is sharing the previous best fit search end.
   d. Compare the hole.
   e. If small enough, then stop searching in the procedure.
   f. Display the values.
7. Terminate the process.

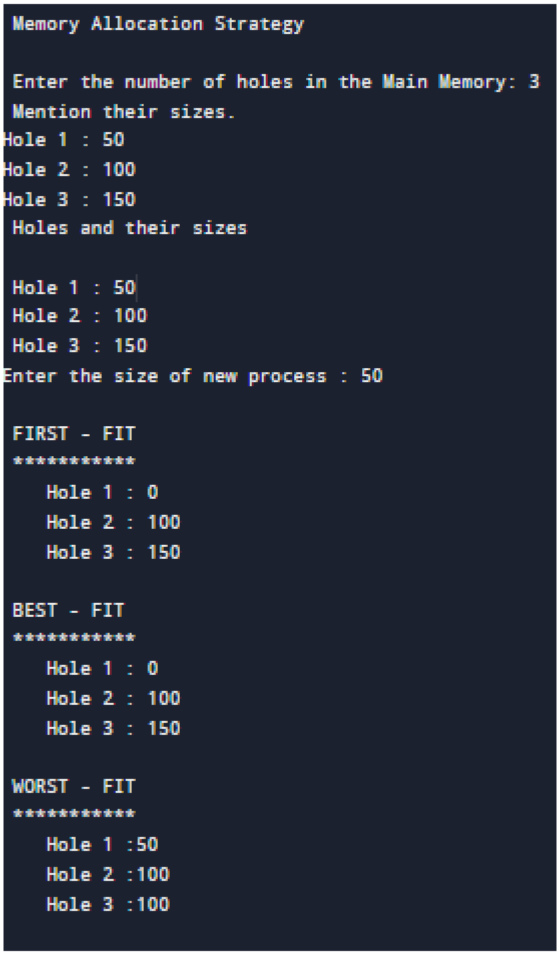
Program:

```
#include<stdio.h>
int main()
{
int n,p,i,j,tmp,t;
int size[10],first[10],best[10],worst[10];
```

```
printf(" Memory Allocation Strategy \n\n Enter the number of holes in the Main Memory: ");
scanf("%d",&n);
printf(" Mention their sizes.\n");
for (i=0;i<n;i++)
{
printf("Hole %d : ",i+1);
scanf("%d",&size[i]);
}
printf(" Holes and their sizes \n\n");
for (i=0;i<n;i++)
{
printf(" Hole %d : %d\n",i+1,size[i]);
first[i]=size[i];
best[i]=size[i];
worst[i]=size[i];
}
printf("Enter the size of new process : ");
scanf("%d",&p);
printf("\n FIRST - FIT \n *********** \n");
for (i=0;i<n;i++)
{
if (size[i]>=p)
{
first[i]=size[i]-p;
break;
}
}
if
(n==i+1)
{
printf("New process of size %d cannot be stored in any holes",p);
goto l;
}
for (i=0;i<n;i++)
{
printf("\tHole %d : %d\n",i+1,first[i]);
}
l:printf("\n BEST - FIT \n *********** \n"); t=0;
for (i=0;i<n;i++)
best[i]=size[i]-p; tmp=best[0];
for (i=1;i<n;i++)
{
if (best[i]>0)
{
if (best[i]<tmp)
{
tmp=best[i]; t=i;     }
}
}
for (i=0;i<n;i++) best[i]=size[i];
if (best[t]>=p) best[t]=best[t]-p;
else
{
```

```
printf("New process of size %d cannot be stored in any holes.",p);
goto l1;
}
for (i=0;i<n;i++)
printf("\tHole %d : %d\n",i+1,best[i]);
l1: printf("\n WORST - FIT \n *********** \n"); t=0;
for (i=0;i<n;i++)     best[i]=size[i]-p; tmp=best[0];
for (i=1;i<n;i++)
{
if (best[i]>0)
{
if (best[i]>tmp)
{
tmp=best[i]; t=i;     }
}
}
for (i=0;i<n;i++) worst[i]=size[i];
if (worst[t]>=p) worst[t]=worst[t]-p;
else
{
printf(".      New process of size %d cannot be stored in any holes.",p);
goto l2;
}
for (i=0;i<n;i++)
printf("\tHole %d :%d\n",i+1,worst[i]);
l2: printf("\nProgram Ended");
}
```

Output:



```
 Memory Allocation Strategy

 Enter the number of holes in the Main Memory: 3
 Mention their sizes.
Hole 1 : 50
Hole 2 : 100
Hole 3 : 150
 Holes and their sizes

 Hole 1 : 50
 Hole 2 : 100
 Hole 3 : 150
Enter the size of new process : 50

 FIRST - FIT
 ***********
    Hole 1 : 0
    Hole 2 : 100
    Hole 3 : 150

 BEST - FIT
 ***********
    Hole 1 : 0
    Hole 2 : 100
    Hole 3 : 150

 WORST - FIT
 ***********
    Hole 1 :50
    Hole 2 :100
    Hole 3 :100
```

Result: