```
Script started on 2020-03-30 20:07:08+0530
 ]0;GAYU@GAYU: ~/Desktop/fit  [01;32mGAYU@GAYU [00m: [01;34m~/Desktop/fit [00m$ gcc fit.c -o f
 ]0;GAYU@GAYU: ~/Desktop/fit  [01;32mGAYU@GAYU [00m: [01;34m~/Desktop/fit [00m$ s  [Kcat fo  [Kit.c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
typedef struct node
{
 int start_address;
 int end_address;
 int size;
 int state;
 struct node* next;
}node;
 node* createNode(int start,int end,int state)
{
 node* newNode = (node*)malloc(sizeof(node));
 newNode->start_address = start;
 newNode->end_address = end;
 newNode->size = end - start;
 newNode->state = state;
 newNode->next = NULL;
 return newNode;
}
void insertLast( node* head,  node* newNode)
{
 node* temp = head;
 while(temp->next!=NULL)
  temp = temp->next;
 newNode->next = temp->next;
 temp->next = newNode;
}
node* create()
{
  node* head = (node*)malloc(sizeof(node));
  head->next = NULL;
  return head;
}
void allocFF(int state,int size,node* free_pool, node* allocated)
{
 node* temp = free_pool->next;
 while(temp!=NULL)
 {
  if(size <= temp->size)
   break;
  temp = temp->next;
 }
 if(temp == NULL)
  printf("Memory cannot be allocated \n");
 else
 {
  int start = temp->start_address;
  int end = start + size;
  temp->start_address = end;
  temp->size -= size;
  node* newNode = createNode(start, end, state);
  insertLast(allocated, newNode);
  printf("Memory Allocation Success \n");
 }
 temp = allocated->next;
 printf("allocated list \n");
 while(temp!=NULL)
 {
  printf("Start address : %d \n",temp->start_address);
  printf("End Address : %d \n",temp->end_address);
  printf("State : %d \n", temp->state);
  temp = temp->next;
 }
 temp = free_pool->next;
 printf("free pool  list \n");
 while(temp!=NULL)
 {
  printf("Start address : %d \n",temp->start_address);
  printf("End Address : %d \n",temp->end_address);
  printf("State : %d \n", temp->state);
  temp = temp->next;
 }
}
void allocBF(int state,int size,node* free_pool, node* allocated)
{
 node* temp = free_pool->next;
 int count = 0;
 int iter;
 int min_hole = 2000;
 while(temp!=NULL)
 {
  if((temp->size - size < min_hole )&& (size <= temp->size))
   {
    iter = count;
    min_hole = temp->size - size;
   }
  count++;
  temp = temp->next;
 }
 if(min_hole == 2000)
  {
   printf("Memory cannot be allocated \n");
  return;
  }
 node * t = free_pool->next;
 int i = 0;
 while(i < iter)
 {
  t = t->next;
  i++;
 }
 if(t == NULL)
  {
```

```
  printf("Memory cannot be allocated \n");
  return;
  }
 else
 {
  int start = t->start_address;
  int end = start + size;
  t->start_address = end;
  t->size -= size;
  node* newNode = createNode(start, end, state);
  insertLast(allocated, newNode);
  printf("Memory Allocation Success \n");
 }
 temp = allocated->next;
 printf("allocated list \n");
 while(temp!=NULL)
 {
  printf("Start address : %d \n",temp->start_address);
  printf("End Address : %d \n",temp->end_address);
  printf("State : %d \n", temp->state);
  temp = temp->next;
 }
 temp = free_pool->next;
 printf("free pool  list \n");
 while(temp!=NULL)
 {
  printf("Start address : %d \n",temp->start_address);
  printf("End Address : %d \n",temp->end_address);
  printf("State : %d \n", temp->state);
  temp = temp->next;
 }
}
void allocWF(int state,int size,node* free_pool, node* allocated)
{
 node* temp = free_pool->next;
 int count = 0;
 int iter;
 int max_hole = -1;
 while(temp!=NULL)
 {
  if((temp->size - size > max_hole )&& (size <= temp->size))
    {
    iter = count;
    max_hole = temp->size - size;
    }
  count++;
  temp = temp->next;
 }
 if(max_hole == -1)
  {
   printf("Memory cannot be allocated \n");
  return;
  }
 node * t = free_pool->next;
 int i = 0;
 while(i < iter)
 {
  t = t->next;
  i++;
 }
 if(t == NULL)
  {
  printf("Memory cannot be allocated \n");
  return;
  }
 else
 {
  int start = t->start_address;
  int end = start + size;
  t->start_address = end;
  t->size -= size;
  node* newNode = createNode(start, end, state);
  insertLast(allocated, newNode);
  printf("Memory Allocation Success \n");
 }
 temp = allocated->next;
 printf("allocated list \n");
 while(temp!=NULL)
 {
  printf("Start address : %d \n",temp->start_address);
  printf("End Address : %d \n",temp->end_address);
  printf("State : %d \n", temp->state);
  temp = temp->next;
 }
 temp = free_pool->next;
 printf("free pool  list \n");
 while(temp!=NULL)
 {
  printf("Start address : %d \n",temp->start_address);
  printf("End Address : %d \n",temp->end_address);
  printf("State : %d \n", temp->state);
  temp = temp->next;
 }
}
void delete(node* allocated, int state)
{
 node* temp = allocated;
 if (temp == NULL)
  return;
 while(temp->next != NULL)
 {
  if(temp->next->state == state)
   {
   printf("\t\t\tFOUND - Deallocation Success \n");
   break;
   }
  temp = temp->next;
 }
 if(temp->next !=NULL )
  temp->next = temp->next->next;
 else
  return;
}
void insertMerge(node* free_pool,node* newNode)
{
 node* temp = free_pool->next;
 while(temp!=NULL)
 {
  if(temp->end_address == newNode->start_address)
  {
   temp->state = -1;
   temp->end_address = newNode->end_address;
   temp->size = temp->end_address - temp->start_address;
   break;
  }
  temp = temp->next;
 }
 printf("\t\t\tMemory added to free pool \n");
 temp = free_pool->next;
 printf("free pool  list \n");
 while(temp!=NULL)
 {
  printf("Start address : %d \n",temp->start_address);
  printf("End Address : %d \n",temp->end_address);
  printf("State : %d \n", temp->state);
  temp = temp->next;
 }
}


void dloc(int state, node* free_pool, node* allocated)
{
 node* temp = allocated->next;
 while(temp!=NULL)
 {
  if(temp->state == state)
   break;
  temp = temp->next;
 }
 if(temp == NULL)
  {
   printf("Memory not found \n");
   return;
  }
 int start = temp->start_address;
 int end = temp->end_address;
 delete(allocated, state);
 node* newNode = createNode(start,end,-1);
 insertMerge(free_pool,newNode);

}
void co_hole(node* free_pool)
```

```
{
 node* temp = free_pool->next;
 while(temp!=NULL)
 {
  if(temp->next == NULL)
   break;
  if((temp->end_address == temp->next->start_address))
  {
   node* delNode = temp->next;
   temp->end_address = temp->next->end_address;
   temp->size = temp->end_address - temp->start_address;
   if(temp->next->next != NULL)
    temp->next = temp->next->next;
   free(delNode);
  }
  else
   temp = temp->next;
 }
 temp = free_pool->next;
 printf("free pool  list \n");
 while(temp!=NULL)
 {
  printf("Start address : %d \n",temp->start_address);
  printf("End Address : %d \n",temp->end_address);
  printf("State : %d \n", temp->state);
  temp = temp->next;
 }

}
void sortLL(node *h)   // too long
{

    int st,en,sta,si;

    struct node *temp1;
    struct node *temp2;

    for(temp1=h->next;temp1!=NULL;temp1=temp1->next)
      {
        for(temp2=temp1->next;temp2!=NULL;temp2=temp2->next)
          {
            if(temp2->start_address < temp1->start_address)
              {
                st = temp1->start_address;
                en = temp1->end_address;
                st = temp1->state;
                si = temp1->size;

                temp1->start_address = temp2->start_address;
                temp1->end_address = temp2->end_address;
                temp1->state = temp2->state;
                temp1->size = temp2->size;

                temp2->start_address = st;
```

```
               temp2->end_address = en;
               temp2->state = sta;
               temp2->size = si;
              }
           }
       }
}
void display(node* free_pool, node* allocated)
{
 printf("\n\n\t\t\tFree Pool allocation \n");
 node* temp = free_pool->next;
 while(temp!=NULL)
 {
  printf("  |");
  for(int i=0;i<5;i++)
   printf(" ");
  printf("%d",temp->state);
  for(int i=0;i<5;i++)
   printf(" ");
  temp = temp->next;
 }
 printf("|\n");
 temp = free_pool->next;
 while(temp!=NULL)
 {
  printf("%d",temp->start_address);
  for(int i=0;i<10;i++)
   printf(" ");
  printf("%d",temp->end_address);
  temp = temp->next;
 }
 printf("\n");

 printf("\n\n\t\t\t Allocated Memory \n");
 temp = allocated->next;
 while(temp!=NULL)
 {
  printf("  |");
  for(int i=0;i<5;i++)
   printf(" ");
  printf("%d",temp->state);
  for(int i=0;i<5;i++)
   printf(" ");
  temp = temp->next;
 }
 printf("|\n");
 temp = allocated->next;
 while(temp!=NULL)
 {
  printf("%d",temp->start_address);
  for(int i=0;i<10;i++)
   printf(" ");
  printf("%d",temp->end_address);
  temp = temp->next;
 }


 printf("\n");
 node* physical = create();
 node* temp1 = free_pool;
 while(temp1!=NULL)
 {
  insertLast(physical,temp1);
  temp1 = temp1->next;
 }
 node* temp2 = allocated->next;
 while(temp2!=NULL)
 {
  insertLast(physical,temp2);
  temp2 = temp2->next;
 }
 //sortLL(physical);

 printf("\n\n\t\tPhysical Memory \n");

 temp = physical->next;
 while(temp!=NULL)
 {
  printf("  |");
  for(int i=0;i<5;i++)
   printf(" ");
  printf("%d",temp->state);
  for(int i=0;i<5;i++)
   printf(" ");
  temp = temp->next;
 }
 printf("|\n");
 temp = physical->next;
 while(temp!=NULL)
 {
  printf("%d",temp->start_address);
  for(int i=0;i<10;i++)
   printf(" ");
  printf("%d",temp->end_address);
  temp = temp->next;
 }
 printf("\n");

}
void main()
{
 int no_of_partitions;
 int start_address;
 int end_address;
 int size;
 int state; // state = -1 implies hole
 int choice;
 int a_choice;
```

```
node* allocated = create();
node* free_pool = create();
printf("\t\t\tEnter number of memory partitions \n");
scanf("%d", &no_of_partitions);
printf("\t\t\tEnter partition details \n");
for(int i = 0;i < no_of_partitions;i++)
{
 printf("Enter starting address \n");
 scanf("%d", &start_address);
 printf("Enter ending address \n");
 scanf("%d", &end_address);
 printf("Enter state \n");
 scanf("%d", &state);
 node* temp = createNode(start_address, end_address, state);
 insertLast(free_pool,temp);
}
do
{
 printf("\t\t\t\tMemory Allocation Algorithm \n");
 printf("1.First Fit \n");
 printf("2.Best Fit \n");
 printf("3.Worst Fit \n");
 printf("4.Exit \n");
 printf("Enter choice \n");
 scanf("%d", &a_choice);
 switch(a_choice)
 {
  case 1:    do
          {
          printf("\t\t\t\t\tFIRST FIT ALGORITHM \n");
          printf("1.Entry / Allocate \n");
          printf("2.Exit / deallocate \n");
          printf("3.Display \n");
          printf("4.Coalescing of Holes \n");
          printf("5.Back to algorithm \n");
          printf("Enter Choice \n");
          scanf("%d", &choice);
          switch(choice)
          {
          case 1:
           printf("Enter process id \n");
           scanf("%d", &state);
           printf("Enter size required \n");
           scanf("%d", &size);
           allocFF(state,size,free_pool,allocated);
    break;
    case 2:
           printf("Enter process id \n");
           scanf("%d", &state);
    dloc(state,free_pool,allocated);
    break;
    case 3:
           display(free_pool,allocated);
           break;
    case 4:
           co_hole(free_pool);
           break;
          }
          }
          while(choice!=5);
          break;
  case 2:

         do
          {
          printf("\t\t\t\t\tBEST FIT ALGORITHM \n");
          printf("1.Entry / Allocate \n");
          printf("2.Exit / deallocate \n");
          printf("3.Display \n");
          printf("4.Coalescing of Holes \n");
          printf("5.Back to algorithm \n");
          printf("Enter Choice \n");
          scanf("%d", &choice);
          switch(choice)
          {
          case 1:
           printf("Enter process id \n");
           scanf("%d", &state);
           printf("Enter size required \n");
           scanf("%d", &size);
           allocBF(state,size,free_pool,allocated);
    break;
    case 2:
           printf("Enter process id \n");
           scanf("%d", &state);
           dloc(state,free_pool,allocated);
    break;
    case 3:
           display(free_pool,allocated);
           break;
    case 4:
           co_hole(free_pool);
           break;
          }
          }
          while(choice!=5);
          break;
  case 3:
          do
          {
          printf("\t\t\t\tWORST FIT ALGORITHM \n");
          printf("1.Entry / Allocate \n");
          printf("2.Exit / deallocate \n");
          printf("3.Display \n");
          printf("4.Coalescing of Holes \n");
          printf("5.Back to algorithm \n");
          printf("Enter Choice \n");
          scanf("%d", &choice);
          switch(choice)
          {
          case 1:
           printf("Enter process id \n");
           scanf("%d", &state);
           printf("Enter size required \n");
           scanf("%d", &size);
            allocWF(state,size,free_pool,allocated);
    break;
    case 2:
           printf("Enter process id \n");
           scanf("%d", &state);
           dloc(state,free_pool,allocated);
    break;
    case 3:
           display(free_pool,allocated);
           break;
    case 4:
           co_hole(free_pool);
           break;
          }
          }
          while(choice!=5);
          break;
   }
   }while(a_choice!=4);
  }


   /*temp = allocated->next;
   printf("allocated list \n");
   while(temp!=NULL)
   {
    printf("Start address : %d \n",temp->start_address);
    printf("End Address : %d \n",temp->end_address);
    printf("State : %d \n", temp->state);
    temp = temp->next;
   }
   temp = free_pool->next;
   printf("free pool  list \n");
   while(temp!=NULL)
   {
    printf("Start address : %d \n",temp->start_address);
    printf("End Address : %d \n",temp->end_address);
    printf("State : %d \n", temp->state);
    temp = temp->next;
   }*/
```

```
 ]0;GAYU@GAYU: ~/Desktop/fit  [01;32mGAYU@GAYU [00m: [01;34m~/Desktop/fit [00m$ ./f
   Enter number of memory partitions
3
   Enter partition details
Enter starting address
100
```

```
Enter ending address
110
Enter state
-1
Enter starting address
110
Enter ending address
150
Enter state
-1
Enter starting address
150
Enter ending address
210
Enter state
-1
   Memory Allocation Algorithm
1.First Fit
2.Best Fit
3.Worst Fit
4.Exit
Enter choice
1
    FIRST FIT ALGORITHM
1.Entry / Allocate
2.Exit / deallocate
3.Display
4.Coalescing of Holes
5.Back to algorithm
Enter Choice
1
Enter process id
13
Enter size required
15
Memory Allocation Success
allocated list
Start address : 110
End Address : 125
State : 13
free pool  list
Start address : 100
End Address : 110
State : -1
Start address : 125
End Address : 150
State : -1
Start address : 150
End Address : 210
State : -1
    FIRST FIT ALGORITHM
1.Entry / Allocate
2.Exit / deallocate
3.Display
```

```
4.Coalescing of Holes
5.Back to algorithm
Enter Choice
5
   Memory Allocation Algorithm
1.First Fit
2.Best Fit
3.Worst Fit
4.Exit
Enter choice
2
    BEST FIT ALGORITHM
1.Entry / Allocate
2.Exit / deallocate
3.Display
4.Coalescing of Holes
5.Back to algorithm
Enter Choice
1
Enter process id
87
Enter size required
4   30
Memory Allocation Success
allocated list
Start address : 110
End Address : 125
State : 13
Start address : 150
End Address : 180
State : 87
free pool  list
Start address : 100
End Address : 110
State : -1
Start address : 125
End Address : 150
State : -1
Start address : 180
End Address : 210
State : -1
    BEST FIT ALGORITHM
1.Entry / Allocate
2.Exit / deallocate
3.Display
4.Coalescing of Holes
5.Back to algorithm
Enter Choice
5
   Memory Allocation Algorithm
1.First Fit
2.Best Fit
3.Worst Fit
4.Exit
```

```
Enter choice
3
   WORST FIT ALGORITHM
1.Entry / Allocate
2.Exit / deallocate
3.Display
4.Coalescing of Holes
5.Back to algorithm
Enter Choice
4   3   5   1
Enter process id
5   768
Enter size required
5
Memory Allocation Success
allocated list
Start address : 110
End Address : 125
State : 13
Start address : 150
End Address : 180
State : 87
Start address : 180
End Address : 185
State : 768
free pool  list
Start address : 100
End Address : 110
State : -1
Start address : 125
End Address : 150
State : -1
Start address : 185
End Address : 210
State : -1
   WORST FIT ALGORITHM
1.Entry / Allocate
2.Exit / deallocate
3.Display
4.Coalescing of Holes
5.Back to algorithm
Enter Choice
2
Enter process id
13
  FOUND - Deallocation Success
  Memory added to free pool
free pool  list
Start address : 100
End Address : 125
State : -1
Start address : 125
End Address : 150
State : -1
```

```
Start address : 185
End Address : 210
State : -1
   WORST FIT ALGORITHM
1.Entry / Allocate
2.Exit / deallocate
3.Display
4.Coalescing of Holes
5.Back to algorithm
Enter Choice
4
free pool  list
Start address : 100
End Address : 150
State : -1
Start address : 185
End Address : 210
State : -1
   WORST FIT ALGORITHM
1.Entry / Allocate
2.Exit / deallocate
3.Display
4.Coalescing of Holes
5.Back to algorithm
Enter Choice
3


  Free Pool allocation
 |    -1      |    -1    |
100        150185        210


   Allocated Memory
 |    87      |    768    |
150        180180        185


  Physical Memory
 |    0      |    87    |
0        0150        180
   WORST FIT ALGORITHM
1.Entry / Allocate
2.Exit / deallocate
3.Display
4.Coalescing of Holes
5.Back to algorithm
Enter Choice
5
   Memory Allocation Algorithm
1.First Fit
2.Best Fit
3.Worst Fit
4.Exit
```

```
Enter choice
4
 ]0;GAYU@GAYU: ~/Desktop/fit  [01;32mGAYU@GAYU [00m: [01;34m~/Desktop/fit [00m$ exit
exit

Script done on 2020-03-30 20:09:32+0530
```