SSN COLLEGE OF ENGINEERING, KALAVAKKAM
(An Autonomous Institution, Affiliated to Anna University, Chennai)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UCS1411 - OPERATING SYSTEMS LAB

LAB EXERCISE 8

Implementation of Memory Management Algorithms

Submission Date:23-05-2022

Name: Jayannthan P T *Dept: CSE 'A'* *Roll No.: 205001049*

1. Free space is maintained as a linked list of nodes with each node having the starting byte address and the ending byte address of a free block. Each memory request consists of the process-id and the amount of storage space required in bytes. Allocated memory space is again maintained as a linked list of nodes with each node having the process id, starting byte address and the ending byte address of the allocated space. When a process finishes (taken as input), the appropriate node from the allocated list should be deleted and this free disk space should be added to the free space list. [Care should be taken to merge contiguous free blocks into one single block. This results in deleting more than one node from the free space list and changing the start and end address in the appropriate node]. For allocation use first fit, worst fit and best fit algorithms..

Algorithm:

1. We create a header file which defines the structure with components as start, end, sixe, status, id, next pointer to the node.
2. It also has a functions insertlast(), insertmiddle(), create new node() and sorted merging()
3. We create an instance of the structure in main function
4. Read the number of partitions of the memory from the usetr
5. Read the starting and ending points of each partitions
6. For each entry create new node and insertlast
7. Display the status after memory partitioning
8. Each hole is allotted a unique id by itself and is not changed until the program is ended
9. Inside the do while loop:
   - Ask the user to choose the algorithm
   - Have a 2D array where we store the choice of algorithm
   - Inside it we have another do while loop where we ask for entry/allocate, exit/deallocate, display, coalesce and exit
   - ALLOCATION:

Ask the user for the size of process

i. First fit:

        a) Check which is the first hole that satisfies the given process's size and return the position of that node
    ii. Best fit:
        a) Assumes min as a constant
        b) Iterates through the list to find the min difference between process size and node size
        c) If min value has changed the node pointed by the function is allocated for the process
    iii. Worst fit:
        a) Assumes 0 as max value
        b) Iterates through the list to find the max difference
        c) If max value is changed allocate the process to the node pointed by the function

allocate()

If ptrsize is equal to the node size we allocate the nide as it is

If ptrsize is greater than the process size we insert 2 node one with size equal to process size and other with size as difference between the pointed node

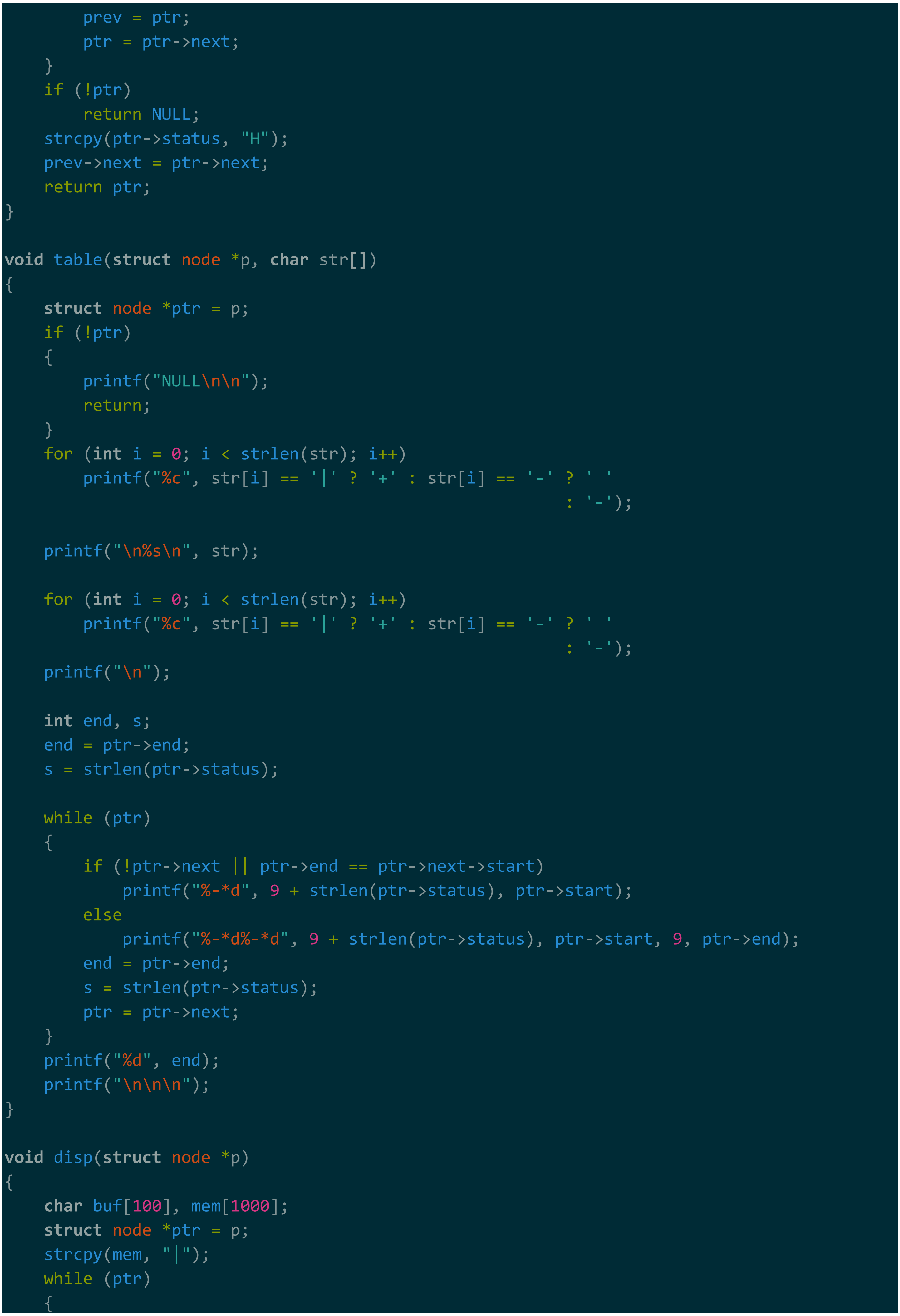
- DEALLOCATION:
    i. We search the pid in the list and find the node in which the process is allocated.
    ii. Change the status of the node to 'H' and delete the process
    iii. It also the combine function which combines the holes with same id to bring back the initial state
- DISPLAY:
    i. Displays the list with free spaces and the other list with accommodation of processes
- COALESCING:
    i. It combines all the adjacent partitions with status 'H' irrespective of the hole id

Code:

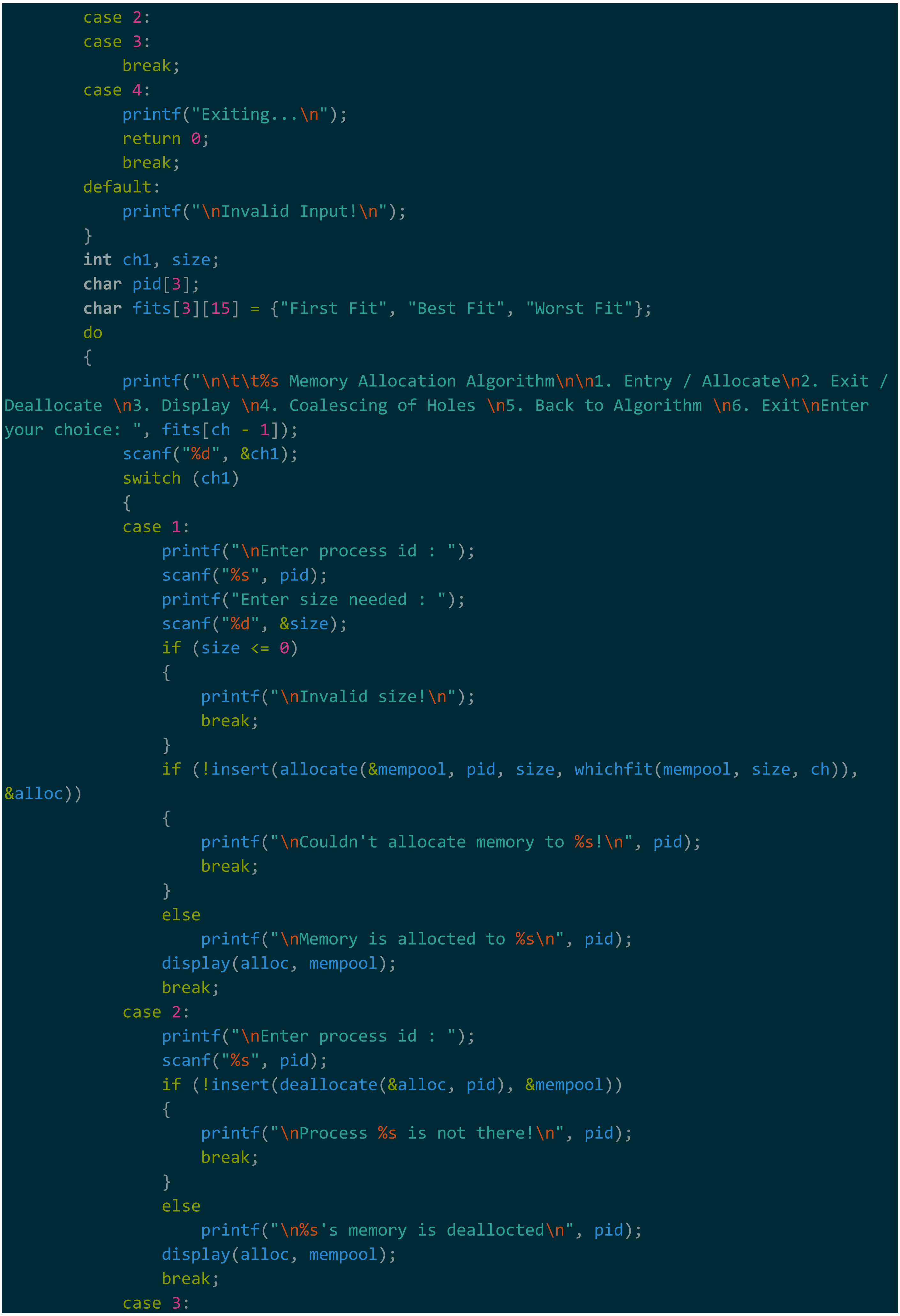
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>

struct node
{
    int start;
    int end;
    int size;
    char status[3];
    struct node *next;
};

struct node *newNode(int start, int end);
int insert(struct node *temp, struct node **head);
struct node *insertEnd(struct node *, struct node *);
struct node *clone(struct node *list);
struct node *merge(struct node *h1, struct node *h2);

struct node *newNode(int start, int end)
```

```
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    strcpy(temp->status, "H");
    temp->end = end;
    temp->start = start;
    temp->size = temp->end - temp->start;
    temp->next = NULL;
    return temp;
}

struct node *insertEnd(struct node *p, struct node *temp)
{
    struct node *ptr = p;
    if (!ptr)
    {
        p = temp;
        p->next = NULL;
    }
    else
    {
        while (ptr->next)
            ptr = ptr->next;
        ptr->next = temp;
        temp->next = NULL;
    }
    return p;
}

int insert(struct node *temp, struct node **head)
{
    if (!temp)
        return 0;
    struct node *ptr = *head;
    if (!*head)
    {
        *head = insertEnd(*head, temp);
        return 1;
    }
    if (temp->start < ptr->start)
    {
        temp->next = *head;
        *head = temp;
        return 1;
    }
    while (ptr->next && temp->start > ptr->next->start)
        ptr = ptr->next;
    temp->next = ptr->next;
    ptr->next = temp;
    return 1;
}

struct node *clone(struct node *list)
{
    if (!list)
```

```
        return NULL;

    struct node *result = (struct node *)malloc(sizeof(struct node));
    result->start = list->start;
    result->end = list->end;
    result->size = list->size;
    strcpy(result->status, list->status);
    result->next = clone(list->next);
    return result;
}

struct node *merge(struct node *h1, struct node *h2)
{
    if (!h1)
        return h2;
    if (!h2)
        return h1;
    if (h1->start < h2->start)
    {
        h1->next = merge(h1->next, h2);
        return h1;
    }
    else
    {
        h2->next = merge(h1, h2->next);
        return h2;
    }
}

void combine(struct node **p)
{
    struct node *ptr = *p, *temp;
    while (ptr)
    {
        temp = ptr;
        while (temp->next && temp->end == temp->next->start)
            temp = temp->next;
        ptr->next = temp->next;
        ptr->end = temp->end;
        ptr = ptr->next;
    }
}

struct node *deallocate(struct node **p, char *pid)
{
    struct node *ptr = *p, *prev;
    if (ptr && strcmp(ptr->status, pid) == 0)
    {
        *p = ptr->next;
        strcpy(ptr->status, "H");
        return ptr;
    }
    while (ptr && strcmp(ptr->status, pid))
    {
```



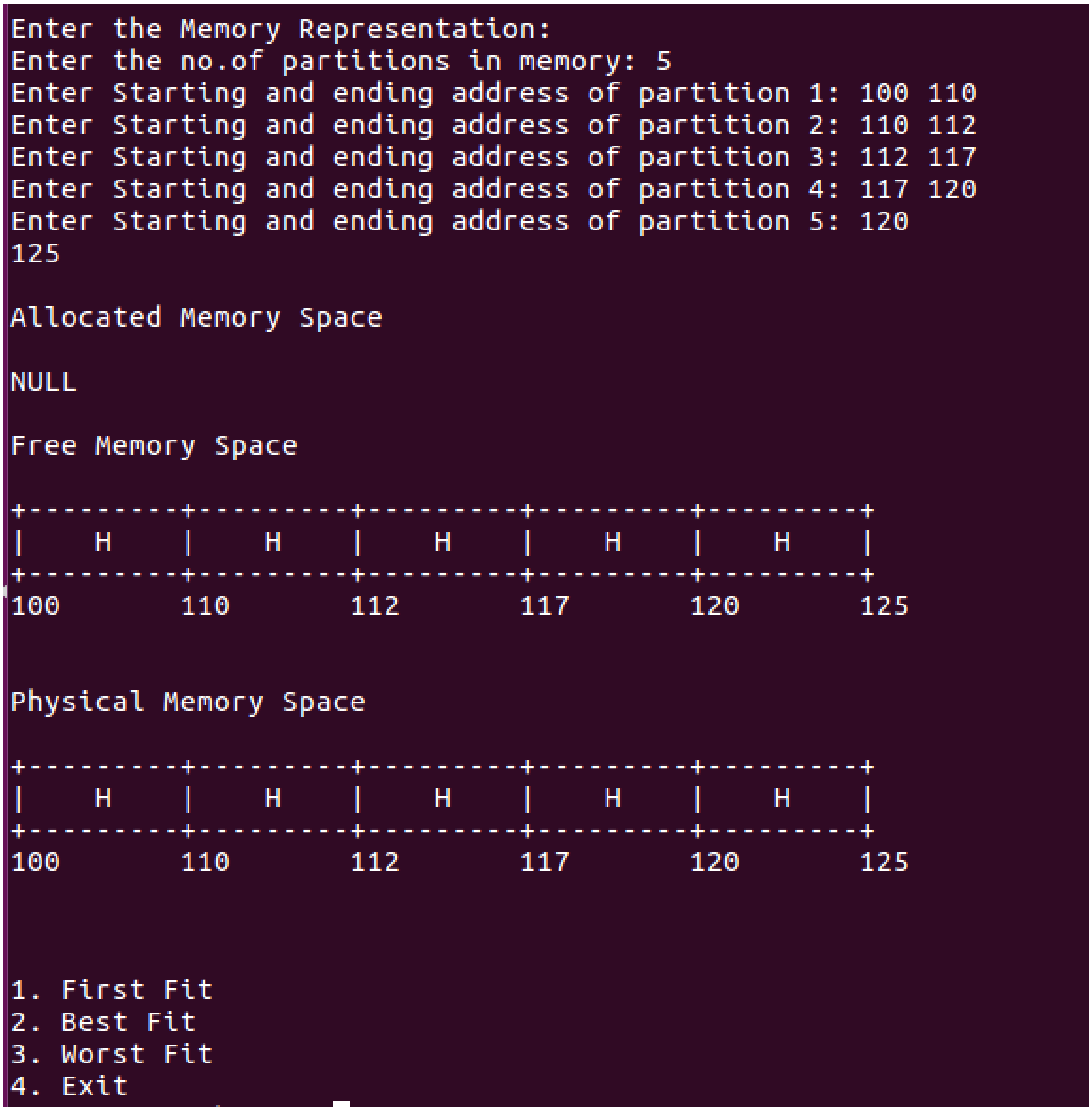
```
        prev = ptr;
        ptr = ptr->next;
    }
    if (!ptr)
        return NULL;
    strcpy(ptr->status, "H");
    prev->next = ptr->next;
    return ptr;
}

void table(struct node *p, char str[])
{
    struct node *ptr = p;
    if (!ptr)
    {
        printf("NULL\n\n");
        return;
    }
    for (int i = 0; i < strlen(str); i++)
        printf("%c", str[i] == '|' ? '+' : str[i] == '-' ? ' '
                                                         : '-');

    printf("\n%s\n", str);

    for (int i = 0; i < strlen(str); i++)
        printf("%c", str[i] == '|' ? '+' : str[i] == '-' ? ' '
                                                         : '-');
    printf("\n");

    int end, s;
    end = ptr->end;
    s = strlen(ptr->status);

    while (ptr)
    {
        if (!ptr->next || ptr->end == ptr->next->start)
            printf("%-*d", 9 + strlen(ptr->status), ptr->start);
        else
            printf("%-*d%-*d", 9 + strlen(ptr->status), ptr->start, 9, ptr->end);
        end = ptr->end;
        s = strlen(ptr->status);
        ptr = ptr->next;
    }
    printf("%d", end);
    printf("\n\n\n");
}

void disp(struct node *p)
{
    char buf[100], mem[1000];
    struct node *ptr = p;
    strcpy(mem, "|");
    while (ptr)
    {
        if (!ptr->next || ptr->end == ptr->next->start)
            sprintf(buf, "    %s    |", ptr->status);
        else if (ptr->end != ptr->next->start)
            sprintf(buf, "    %s    |---------|", ptr->status);
        strcat(mem, buf);
        ptr = ptr->next;
    }
    table(p, mem);
}

void display(struct node *p, struct node *q)
{
    printf("\nAllocated Memory Space\n\n");
    disp(p);
    printf("Free Memory Space\n\n");
    disp(q);
    struct node *r = merge(clone(p), clone(q));
    printf("Physical Memory Space\n\n");
    disp(r);
    free(r);
}

int first(struct node *f, int size)
{
    struct node *ptr = f;
    while (ptr && !(ptr->size >= size))
    {
        ptr = ptr->next;
    }
    if (!ptr)
        return -1;
    return ptr->size;
}

int best(struct node *f, int size)
{
    struct node *ptr = f;
    int min = INT_MAX;
    while (ptr)
    {
        if (ptr->size - size > 0 && min > ptr->size - size)
            min = ptr->size - size;
        ptr = ptr->next;
    }
    if (min == INT_MAX)
        return -1;
    return min + size;
}

int worst(struct node *f, int size)
{
    struct node *ptr = f;
    int max = INT_MIN;
    while (ptr)
    {
        if (ptr->size - size > 0 && max < ptr->size - size)
            max = ptr->size - size;
        ptr = ptr->next;
    }
    if (max == INT_MIN)
        return -1;
    return max + size;
}

int whichfit(struct node *f, int size, int ch)
{
    if (ch == 1)
        return first(f, size);
    if (ch == 2)
        return best(f, size);
    if (ch == 3)
        return worst(f, size);
}

struct node *allocate(struct node **f, char *pid, int size, int ptrsize)
{
    if (ptrsize - size < 0)
        return NULL;
    struct node *ptr = *f, *prev;

    if (ptr->size == ptrsize)
    {
        if (ptr && ptr->size >= size)
        {
            if (ptr->size == size)
            {
                *f = ptr->next;

                strcpy(ptr->status, pid);
                return ptr;
            }
            else
            {
                struct node *temp1 = newNode(ptr->start, ptr->start + size);
                struct node *temp2 = newNode(ptr->start + size, ptr->end);
                *f = temp2;
                temp2->next = ptr->next;
                strcpy(temp1->status, pid);
                free(ptr);
                return temp1;
            }
        }
    }

    while (ptr && !(ptr->size == ptrsize))
    {
        prev = ptr;
        ptr = ptr->next;
    }
    if (!ptr)
        return NULL;
    if (ptr->size == size)
    {
        prev->next = ptr->next;
        strcpy(ptr->status, pid);
        return ptr;
    }
    else
    {
        struct node *temp1 = newNode(ptr->start, ptr->start + size);
        struct node *temp2 = newNode(ptr->start + size, ptr->end);
        prev->next = temp2;
        temp2->next = ptr->next;
        strcpy(temp1->status, pid);
        free(ptr);
        return temp1;
    }
}

int main()
{
    int ch, n, start, end;
    printf("\nEnter the Memory Representation:");
    printf("\nEnter the no.of partitions in memory: ");
    scanf("%d", &n);
    struct node *mempool = NULL, *alloc = NULL, *temp = NULL;
    for (int i = 0; i < n; i++)
    {
        printf("Enter Starting and ending address of partition %d: ", i + 1);
        scanf("%d%d", &start, &end);
        if (start >= end || i && temp->end != start)
        {
            i--;
            printf("Invalid entry,enter again\n");
        }
        else
        {
            temp = newNode(start, end);
            mempool = insertEnd(mempool, temp);
        }
    }

    display(alloc, mempool);

    do
    {
        printf("\n1. First Fit\n2. Best Fit \n3. Worst Fit \n4. Exit \nEnter your choice:
");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
```



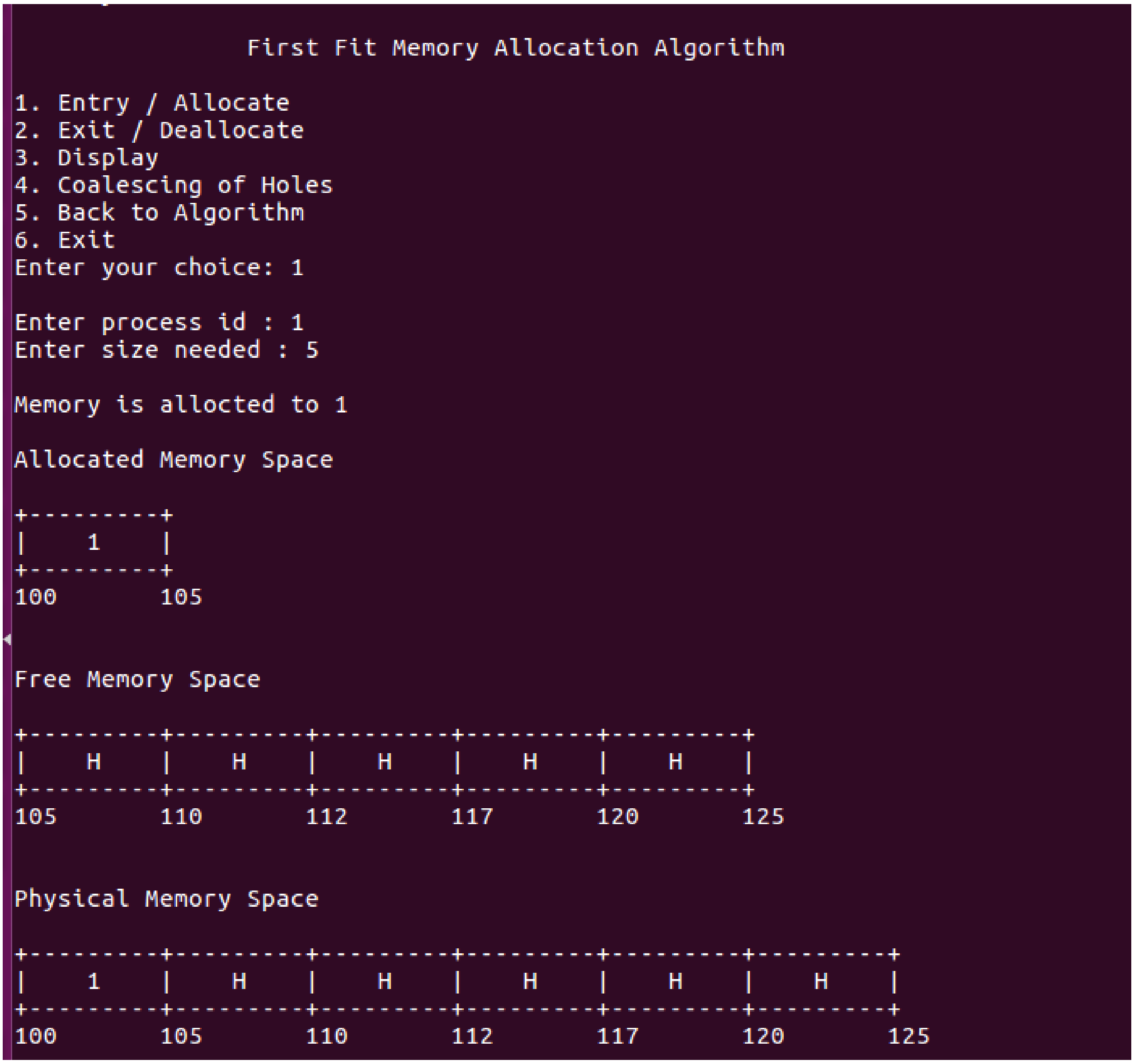
```
        case 2:
        case 3:
            break;
        case 4:
            printf("Exiting...\n");
            return 0;
            break;
        default:
            printf("\nInvalid Input!\n");
        }
        int ch1, size;
        char pid[3];
        char fits[3][15] = {"First Fit", "Best Fit", "Worst Fit"};
        do
        {
            printf("\n\t\t%s Memory Allocation Algorithm\n\n1. Entry / Allocate\n2. Exit / Deallocate \n3. Display \n4. Coalescing of Holes \n5. Back to Algorithm \n6. Exit\nEnter your choice: ", fits[ch - 1]);
            scanf("%d", &ch1);
            switch (ch1)
            {
            case 1:
                printf("\nEnter process id : ");
                scanf("%s", pid);
                printf("Enter size needed : ");
                scanf("%d", &size);
                if (size <= 0)
                {
                    printf("\nInvalid size!\n");
                    break;
                }
                if (!insert(allocate(&mempool, pid, size, whichfit(mempool, size, ch)), &alloc))
                {
                    printf("\nCouldn't allocate memory to %s!\n", pid);
                    break;
                }
                else
                    printf("\nMemory is allocted to %s\n", pid);
                display(alloc, mempool);
                break;
            case 2:
                printf("\nEnter process id : ");
                scanf("%s", pid);
                if (!insert(deallocate(&alloc, pid), &mempool))
                {
                    printf("\nProcess %s is not there!\n", pid);
                    break;
                }
                else
                    printf("\n%s's memory is deallocted\n", pid);
                display(alloc, mempool);
                break;
            case 3:
                display(alloc, mempool);
                break;
            case 4:
                combine(&mempool);
                display(alloc, mempool);
                break;
            case 5:
                break;
            case 6:
                printf("Exiting...\n");
                return 0;
                break;
            default:
                printf("\nInavlid Input!\n");
            }
        } while (!(ch1 == 5 || ch1 == 6));
    } while (ch != 4);
    return 0;
}
```

Output:

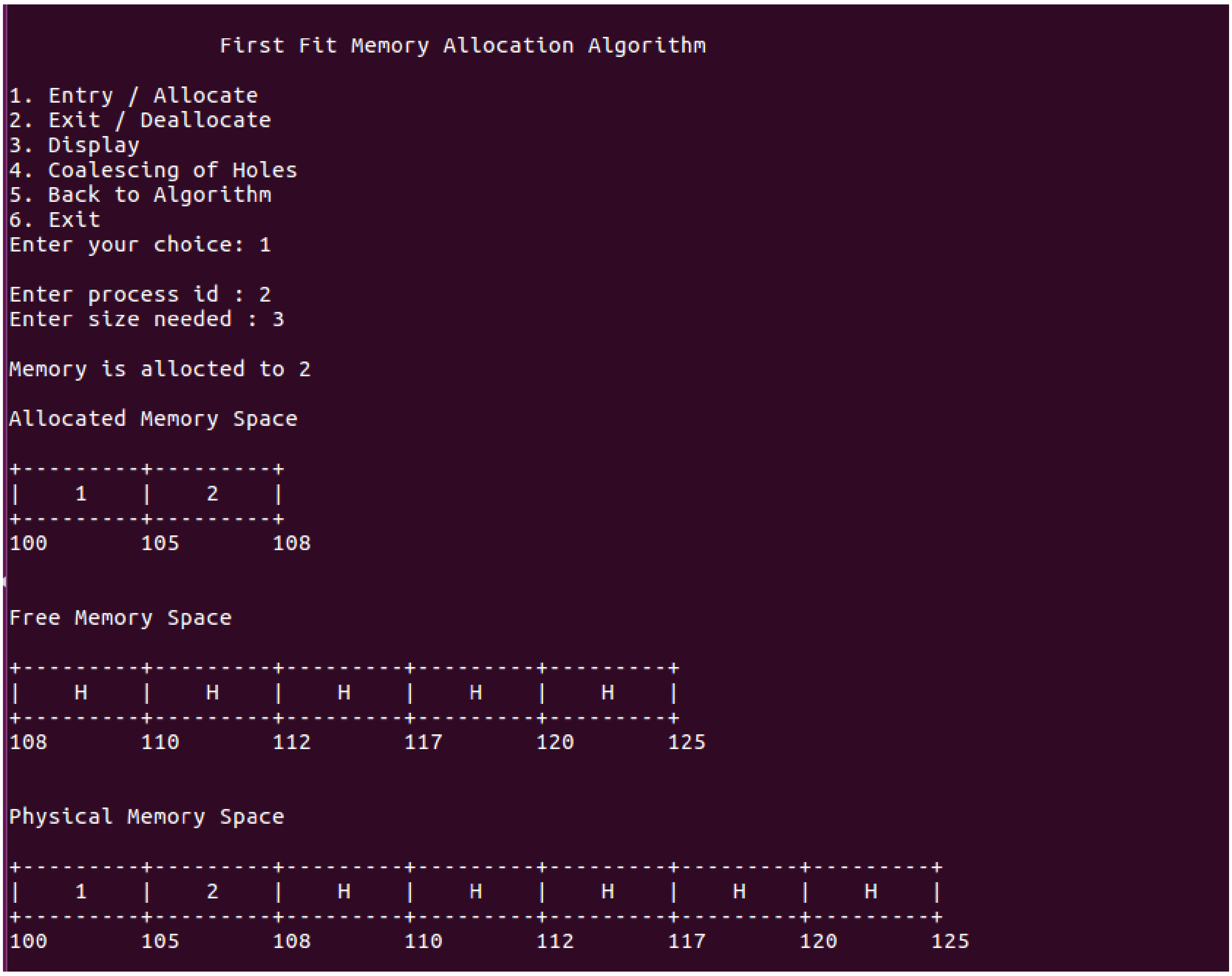
1.



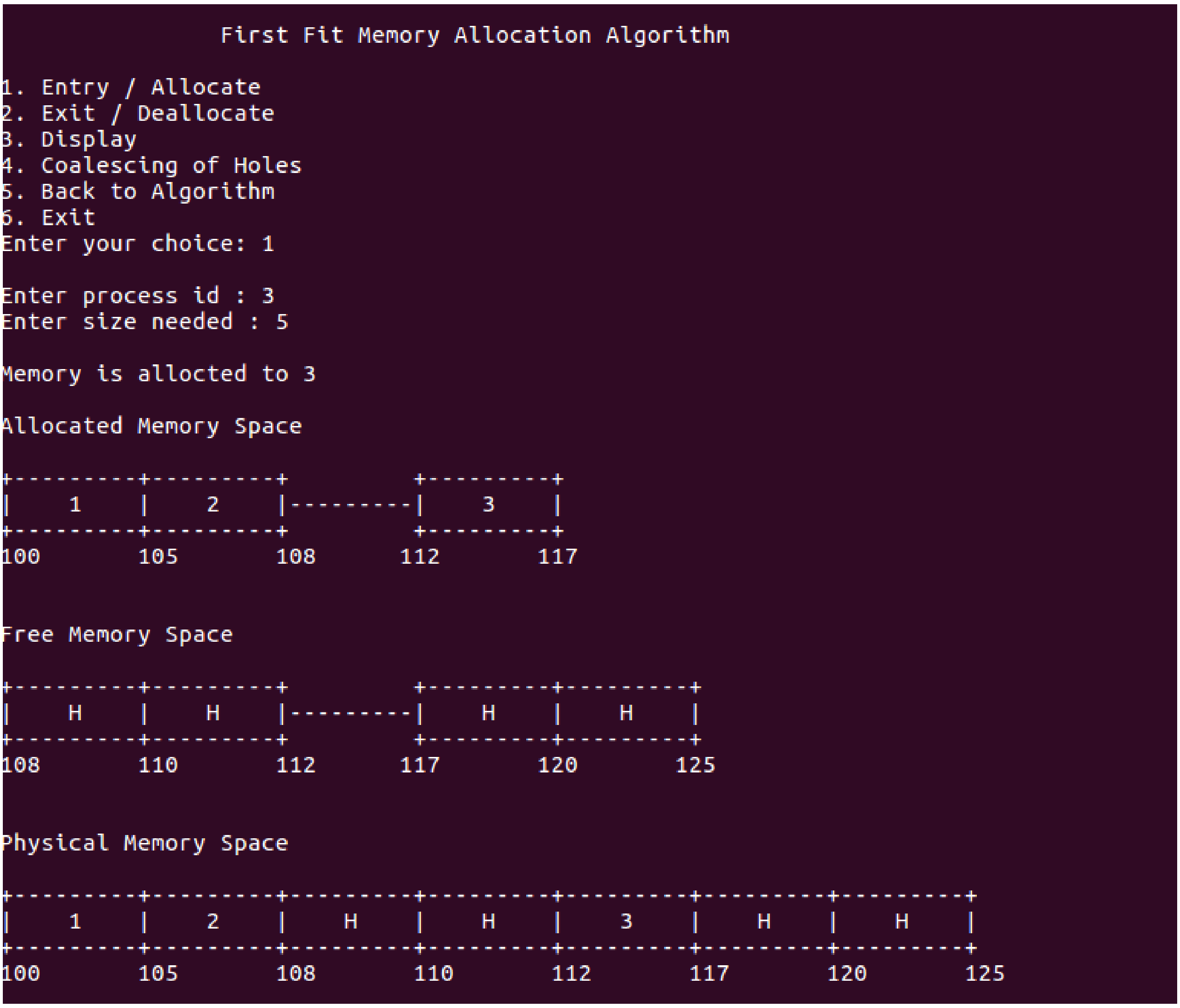
```
Enter the Memory Representation:
Enter the no.of partitions in memory: 5
Enter Starting and ending address of partition 1: 100 110
Enter Starting and ending address of partition 2: 110 112
Enter Starting and ending address of partition 3: 112 117
Enter Starting and ending address of partition 4: 117 120
Enter Starting and ending address of partition 5: 120
125

Allocated Memory Space

NULL

Free Memory Space

+---------+---------+---------+---------+---------+
|    H    |    H    |    H    |    H    |    H    |
+---------+---------+---------+---------+---------+
100       110       112       117       120       125


Physical Memory Space

+---------+---------+---------+---------+---------+
|    H    |    H    |    H    |    H    |    H    |
+---------+---------+---------+---------+---------+
100       110       112       117       120       125



1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
```



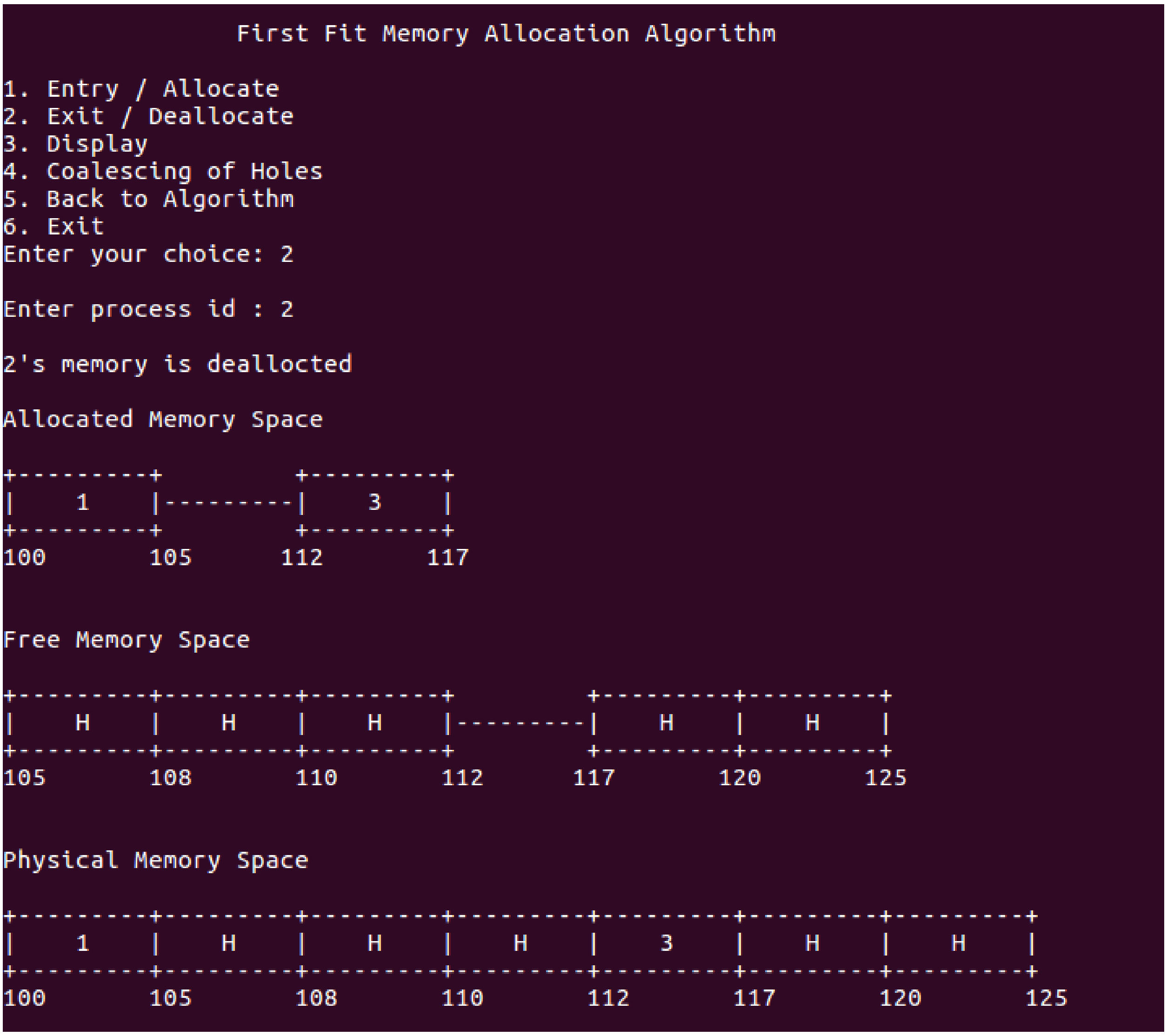
```
                First Fit Memory Allocation Algorithm

1. Entry / Allocate
2. Exit / Deallocate
3. Display
4. Coalescing of Holes
5. Back to Algorithm
6. Exit
Enter your choice: 1

Enter process id : 1
Enter size needed : 5

Memory is allocted to 1

Allocated Memory Space

+---------+
|    1    |
+---------+
100       105


Free Memory Space

+---------+---------+---------+---------+---------+
|    H    |    H    |    H    |    H    |    H    |
+---------+---------+---------+---------+---------+
105       110       112       117       120       125


Physical Memory Space

+---------+---------+---------+---------+---------+---------+
|    1    |    H    |    H    |    H    |    H    |    H    |
+---------+---------+---------+---------+---------+---------+
100       105       110       112       117       120       125
```



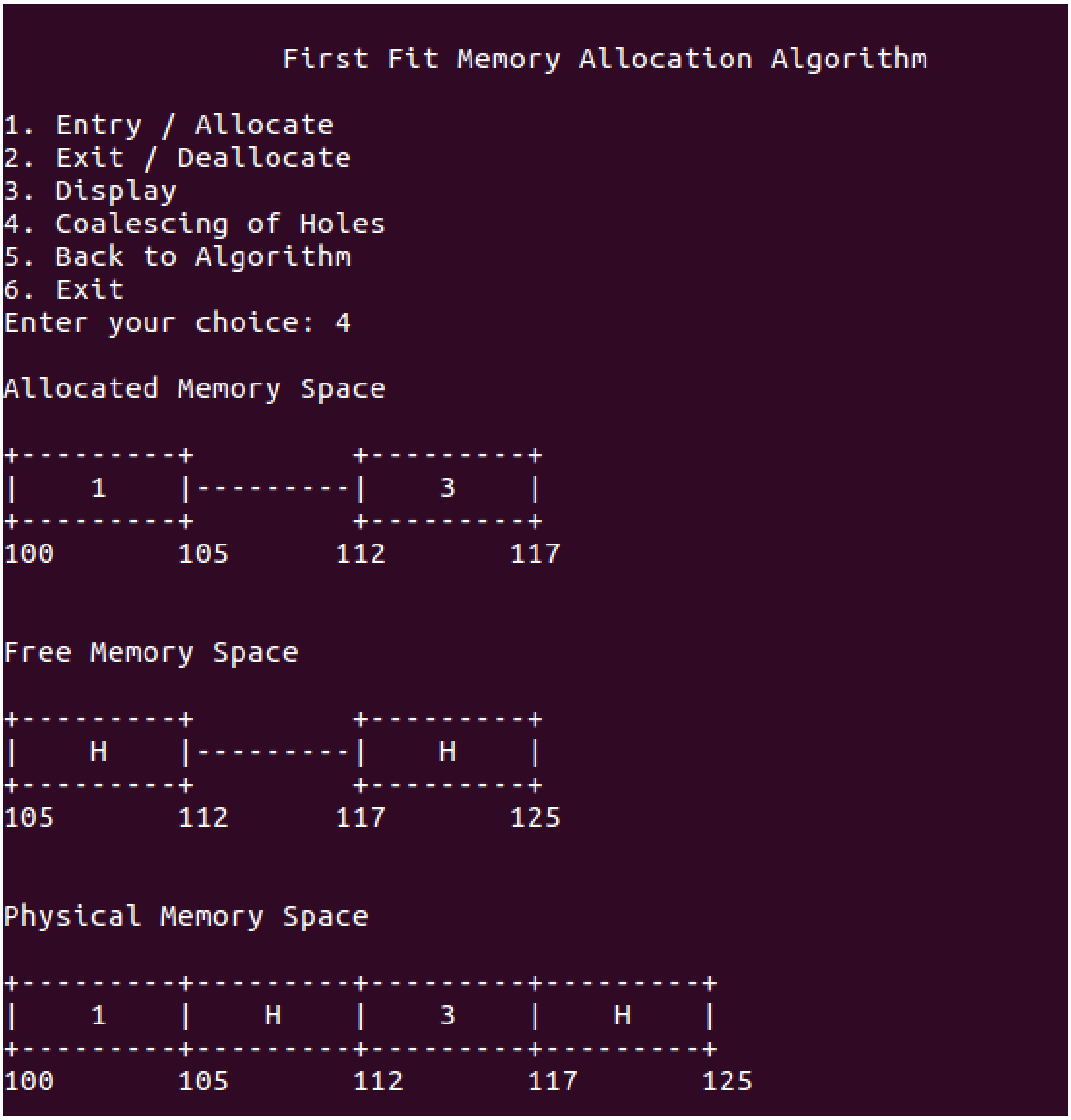
```
                First Fit Memory Allocation Algorithm

1. Entry / Allocate
2. Exit / Deallocate
3. Display
4. Coalescing of Holes
5. Back to Algorithm
6. Exit
Enter your choice: 1

Enter process id : 2
Enter size needed : 3

Memory is allocted to 2

Allocated Memory Space

+---------+---------+
|    1    |    2    |
+---------+---------+
100       105       108


Free Memory Space

+---------+---------+---------+---------+---------+
|    H    |    H    |    H    |    H    |    H    |
+---------+---------+---------+---------+---------+
108       110       112       117       120       125


Physical Memory Space

+---------+---------+---------+---------+---------+---------+---------+
|    1    |    2    |    H    |    H    |    H    |    H    |    H    |
+---------+---------+---------+---------+---------+---------+---------+
100       105       108       110       112       117       120       125
```



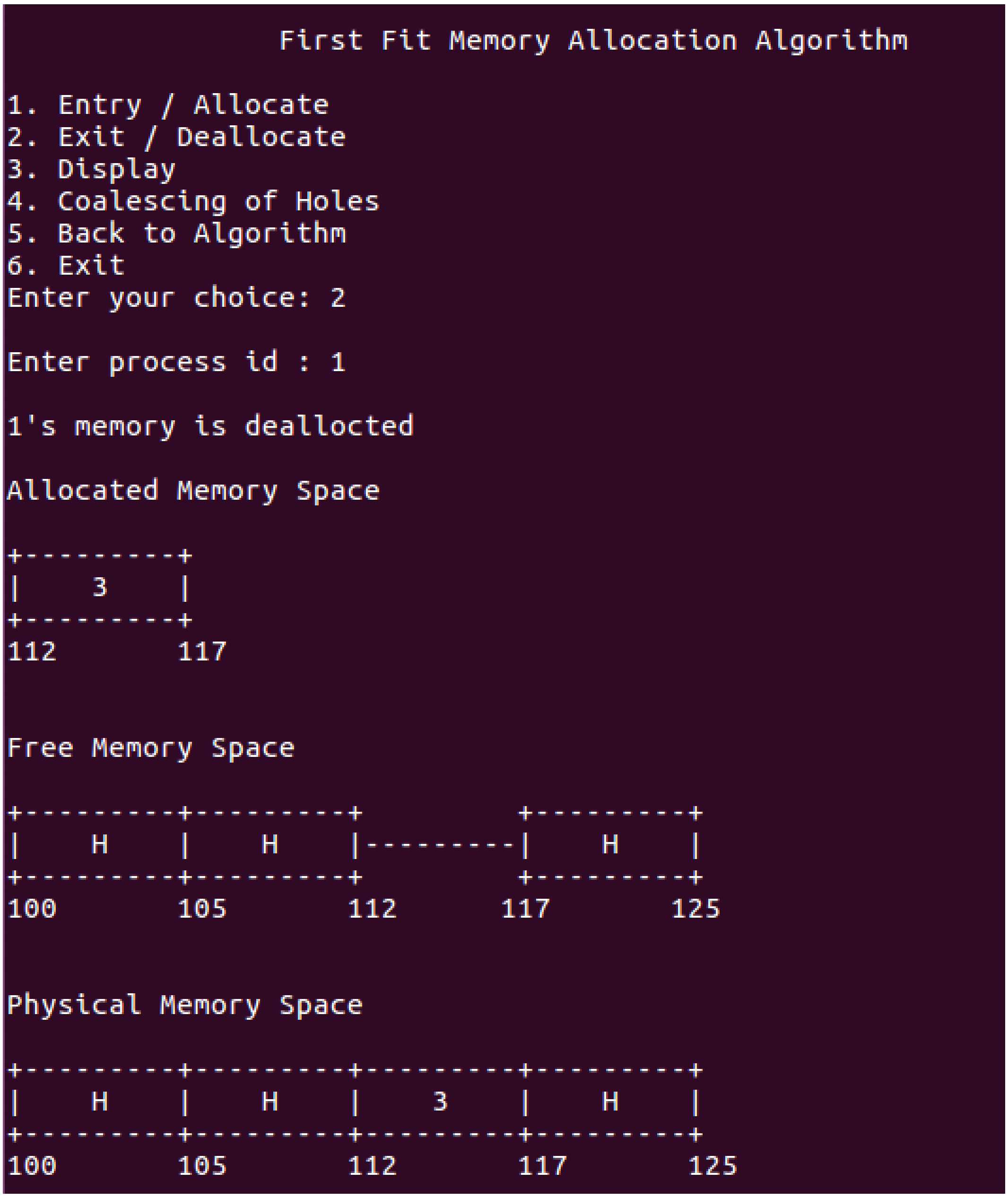
```
                First Fit Memory Allocation Algorithm

1. Entry / Allocate
2. Exit / Deallocate
3. Display
4. Coalescing of Holes
5. Back to Algorithm
6. Exit
Enter your choice: 1

Enter process id : 3
Enter size needed : 5

Memory is allocted to 3

Allocated Memory Space

+---------+---------+         +---------+
|    1    |    2    |---------|    3    |
+---------+---------+         +---------+
100       105       108      112       117


Free Memory Space

+---------+---------+         +---------+---------+
|    H    |    H    |---------|    H    |    H    |
+---------+---------+         +---------+---------+
108       110       112      117       120       125


Physical Memory Space

+---------+---------+---------+---------+---------+---------+---------+
|    1    |    2    |    H    |    H    |    3    |    H    |    H    |
+---------+---------+---------+---------+---------+---------+---------+
100       105       108       110       112       117       120       125
```



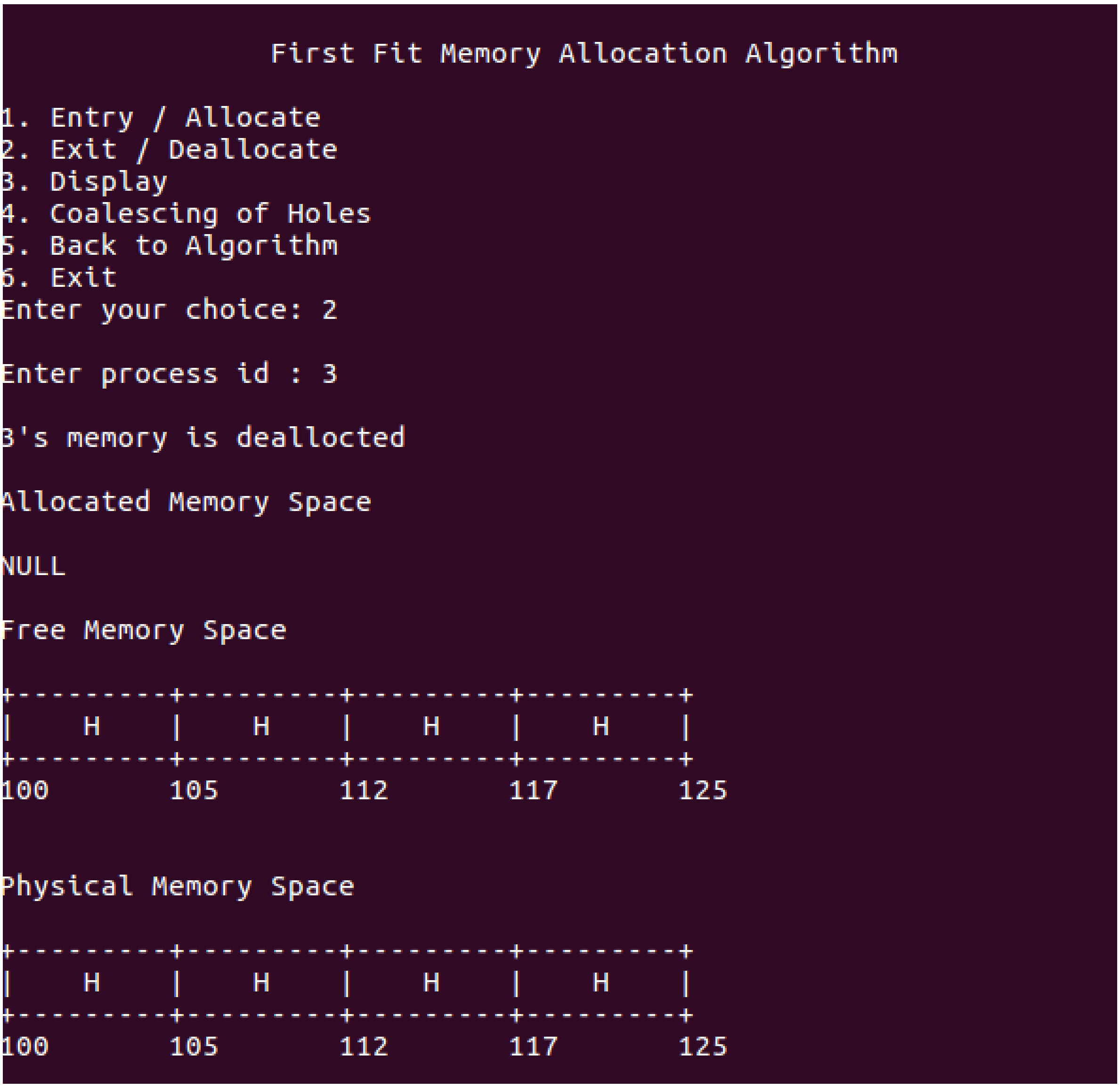
```
                First Fit Memory Allocation Algorithm

1. Entry / Allocate
2. Exit / Deallocate
3. Display
4. Coalescing of Holes
5. Back to Algorithm
6. Exit
Enter your choice: 2

Enter process id : 2

2's memory is deallocted

Allocated Memory Space

+---------+         +---------+
|    1    |---------|    3    |
+---------+         +---------+
100       105      112       117


Free Memory Space

+---------+---------+---------+         +---------+---------+
|    H    |    H    |    H    |---------|    H    |    H    |
+---------+---------+---------+         +---------+---------+
105       108       110       112      117       120       125


Physical Memory Space

+---------+---------+---------+---------+---------+---------+---------+
|    1    |    H    |    H    |    H    |    3    |    H    |    H    |
+---------+---------+---------+---------+---------+---------+---------+
100       105       108       110       112       117       120       125
```



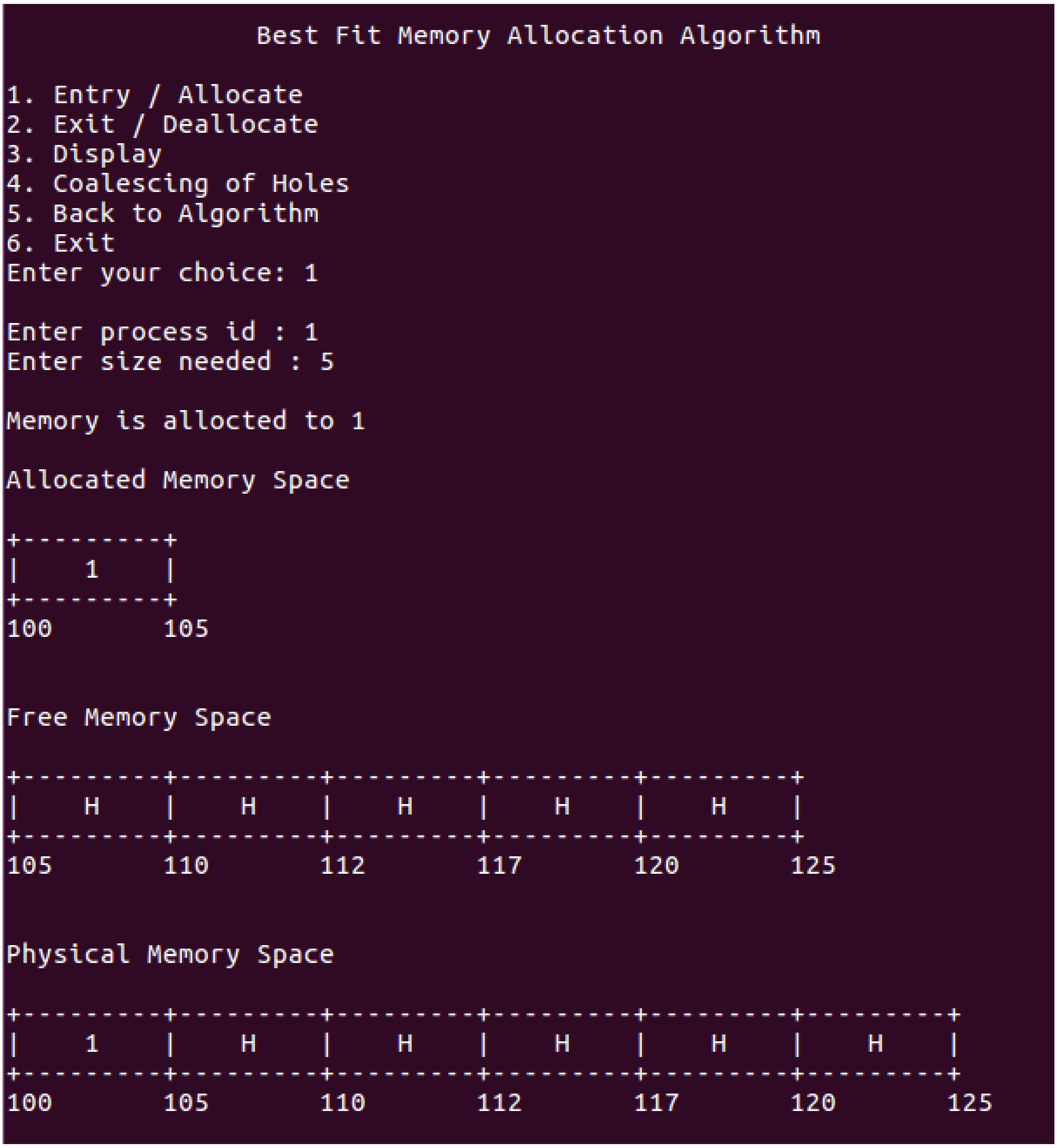
```
                First Fit Memory Allocation Algorithm

1. Entry / Allocate
2. Exit / Deallocate
3. Display
4. Coalescing of Holes
5. Back to Algorithm
6. Exit
Enter your choice: 4

Allocated Memory Space

+---------+         +---------+
|    1    |---------|    3    |
+---------+         +---------+
100       105      112       117


Free Memory Space

+---------+         +---------+
|    H    |---------|    H    |
+---------+         +---------+
105       112      117       125


Physical Memory Space

+---------+---------+---------+---------+
|    1    |    H    |    3    |    H    |
+---------+---------+---------+---------+
100       105       112       117       125
```



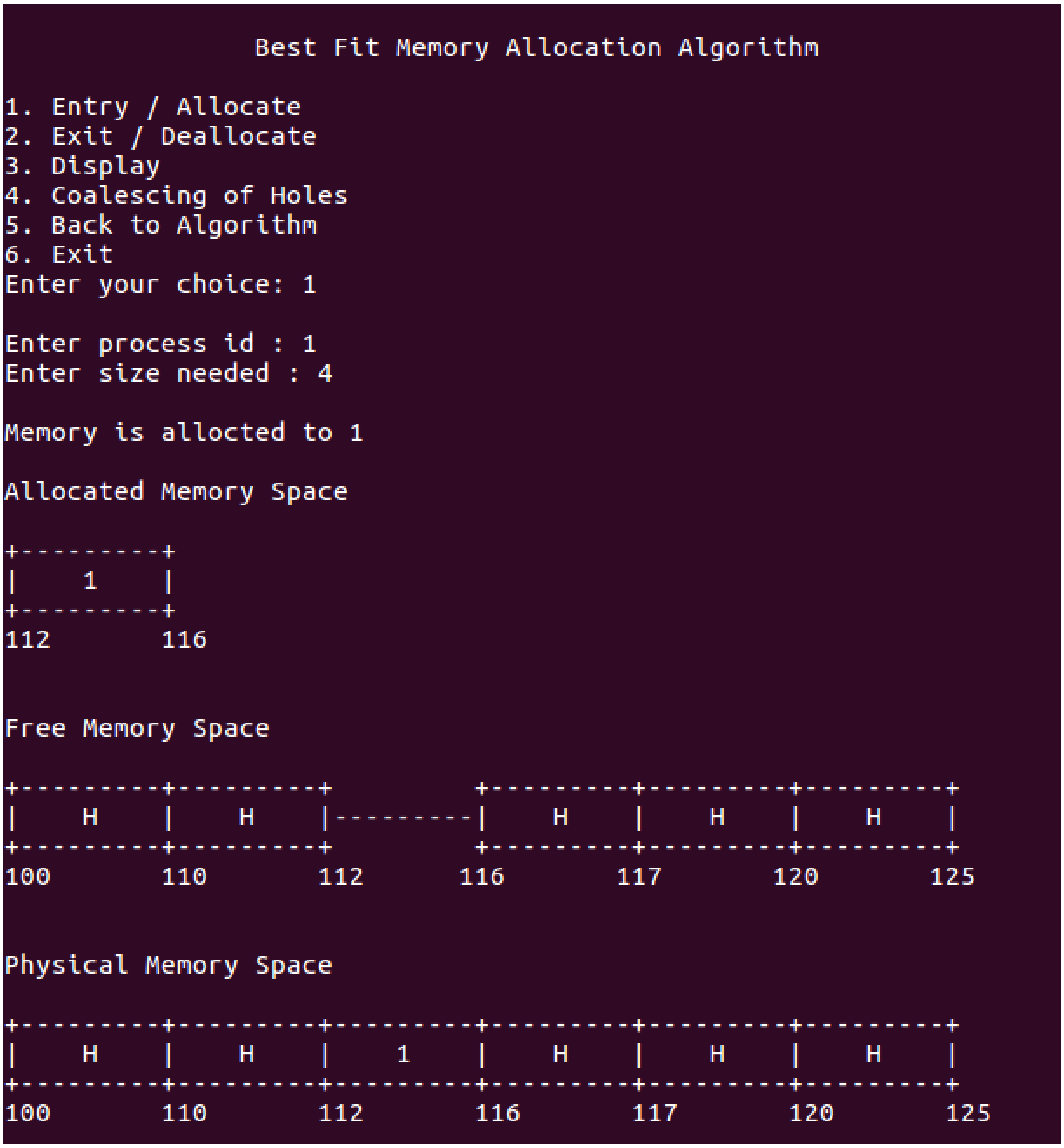
```
                First Fit Memory Allocation Algorithm

1. Entry / Allocate
2. Exit / Deallocate
3. Display
4. Coalescing of Holes
5. Back to Algorithm
6. Exit
Enter your choice: 2

Enter process id : 1

1's memory is deallocted

Allocated Memory Space

+---------+
|    3    |
+---------+
112       117


Free Memory Space

+---------+---------+         +---------+
|    H    |    H    |---------|    H    |
+---------+---------+         +---------+
100       105       112      117       125


Physical Memory Space

+---------+---------+---------+---------+
|    H    |    H    |    3    |    H    |
+---------+---------+---------+---------+
100       105       112       117       125
```



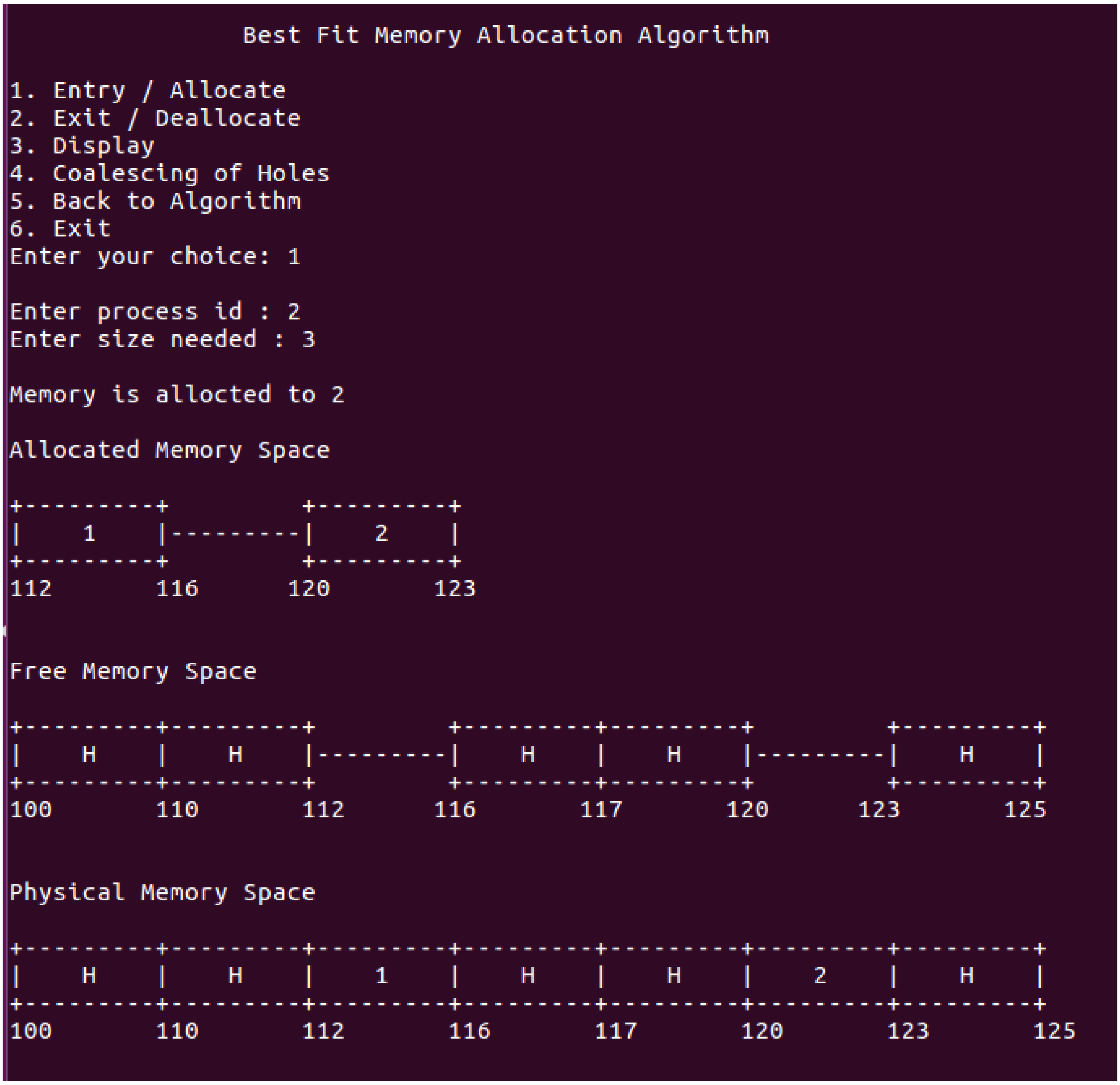
```
                First Fit Memory Allocation Algorithm

1. Entry / Allocate
2. Exit / Deallocate
3. Display
4. Coalescing of Holes
5. Back to Algorithm
6. Exit
Enter your choice: 2

Enter process id : 3

3's memory is deallocted

Allocated Memory Space

NULL

Free Memory Space

+---------+---------+---------+---------+
|    H    |    H    |    H    |    H    |
+---------+---------+---------+---------+
100       105       112       117       125


Physical Memory Space

+---------+---------+---------+---------+
|    H    |    H    |    H    |    H    |
+---------+---------+---------+---------+
100       105       112       117       125
```



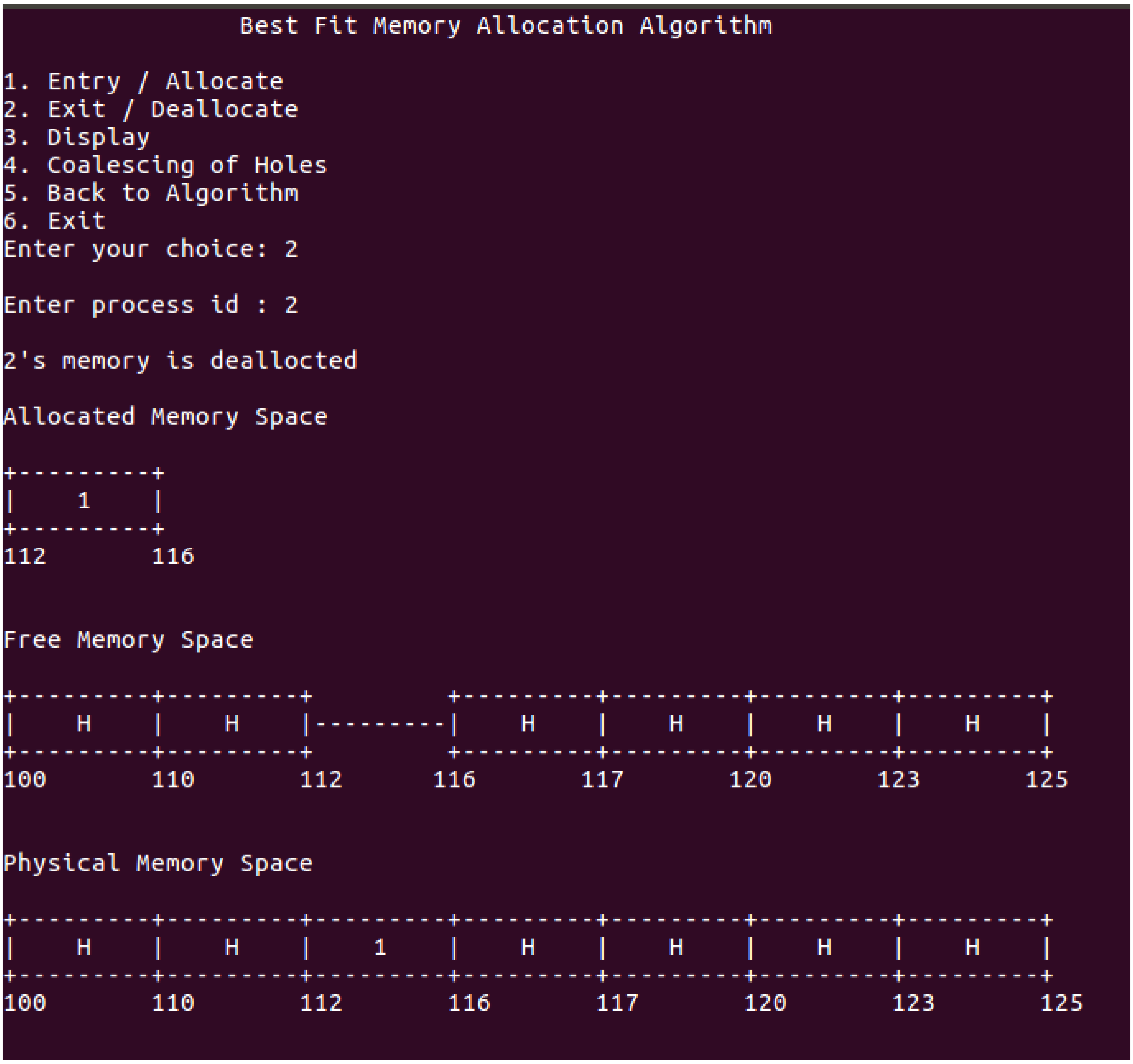
```
                Best Fit Memory Allocation Algorithm

1. Entry / Allocate
2. Exit / Deallocate
3. Display
4. Coalescing of Holes
5. Back to Algorithm
6. Exit
Enter your choice: 1

Enter process id : 1
Enter size needed : 5

Memory is allocted to 1

Allocated Memory Space

+---------+
|    1    |
+---------+
100       105


Free Memory Space

+---------+---------+---------+---------+---------+
|    H    |    H    |    H    |    H    |    H    |
+---------+---------+---------+---------+---------+
105       110       112       117       120       125


Physical Memory Space

+---------+---------+---------+---------+---------+---------+
|    1    |    H    |    H    |    H    |    H    |    H    |
+---------+---------+---------+---------+---------+---------+
100       105       110       112       117       120       125
```



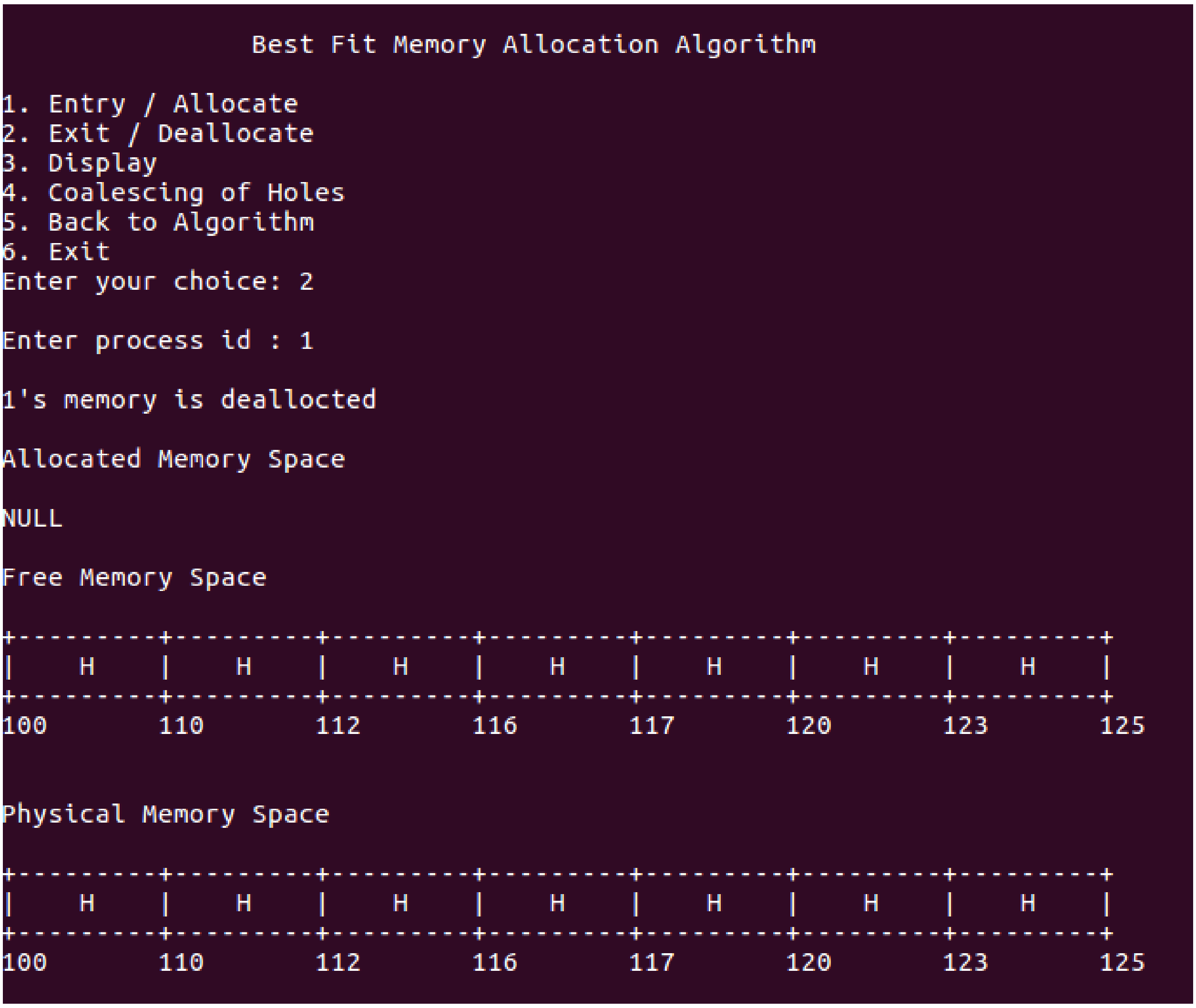
```
                Best Fit Memory Allocation Algorithm

1. Entry / Allocate
2. Exit / Deallocate
3. Display
4. Coalescing of Holes
5. Back to Algorithm
6. Exit
Enter your choice: 1

Enter process id : 1
Enter size needed : 4

Memory is allocted to 1

Allocated Memory Space

+---------+
|    1    |
+---------+
112       116


Free Memory Space

+---------+---------+         +---------+---------+---------+
|    H    |    H    |---------|    H    |    H    |    H    |
+---------+---------+         +---------+---------+---------+
100       110       112      116       117       120       125


Physical Memory Space

+---------+---------+---------+---------+---------+---------+
|    H    |    H    |    1    |    H    |    H    |    H    |
+---------+---------+---------+---------+---------+---------+
100       110       112       116       117       120       125
```



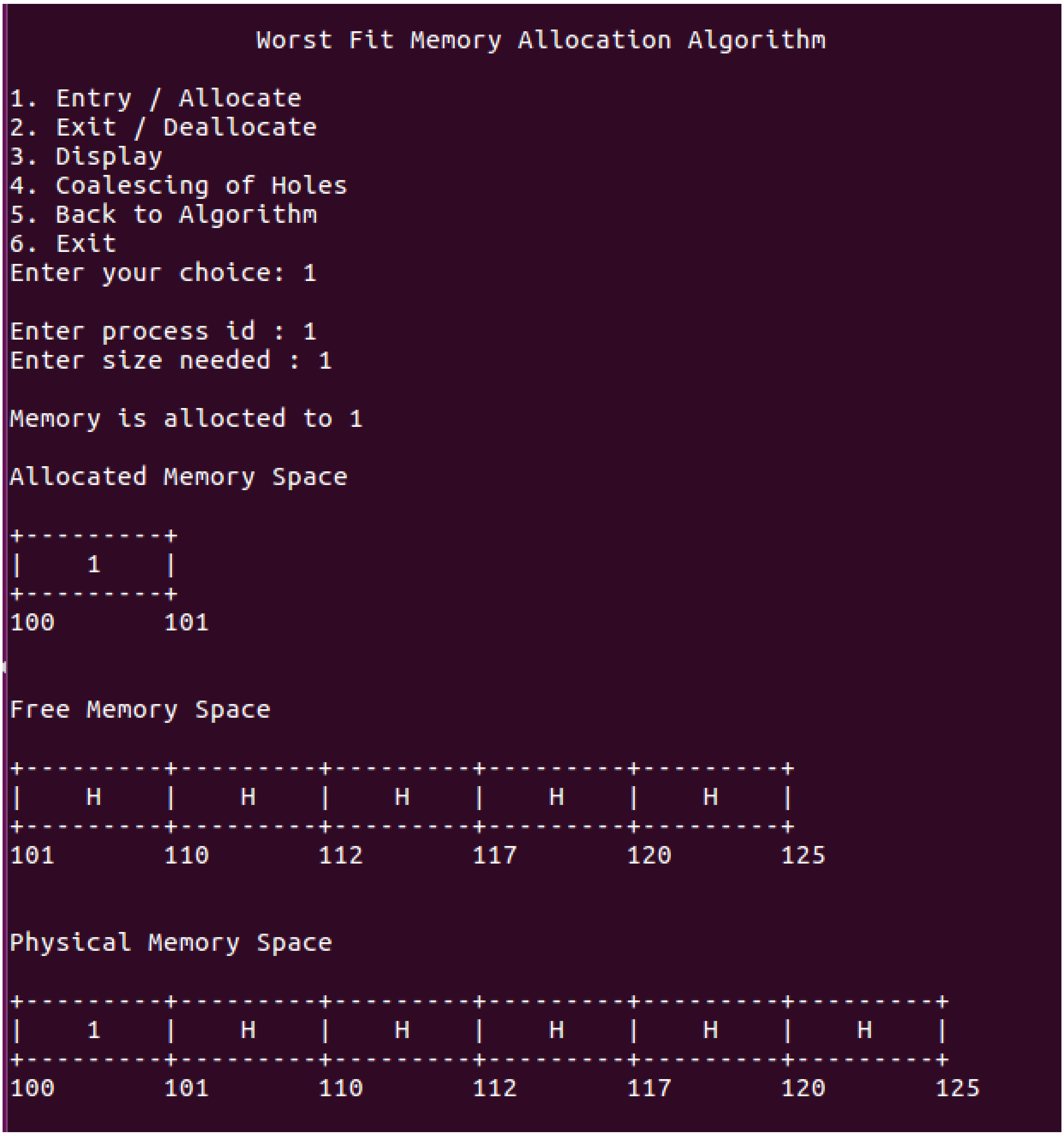
```
                Best Fit Memory Allocation Algorithm

1. Entry / Allocate
2. Exit / Deallocate
3. Display
4. Coalescing of Holes
5. Back to Algorithm
6. Exit
Enter your choice: 1

Enter process id : 2
Enter size needed : 3

Memory is allocted to 2

Allocated Memory Space

+---------+         +---------+
|    1    |---------|    2    |
+---------+         +---------+
112       116      120       123


Free Memory Space

+---------+---------+         +---------+---------+         +---------+
|    H    |    H    |---------|    H    |    H    |---------|    H    |
+---------+---------+         +---------+---------+         +---------+
100       110       112      116       117       120      123       125


Physical Memory Space

+---------+---------+---------+---------+---------+---------+---------+
|    H    |    H    |    1    |    H    |    H    |    2    |    H    |
+---------+---------+---------+---------+---------+---------+---------+
100       110       112       116       117       120       123       125
```



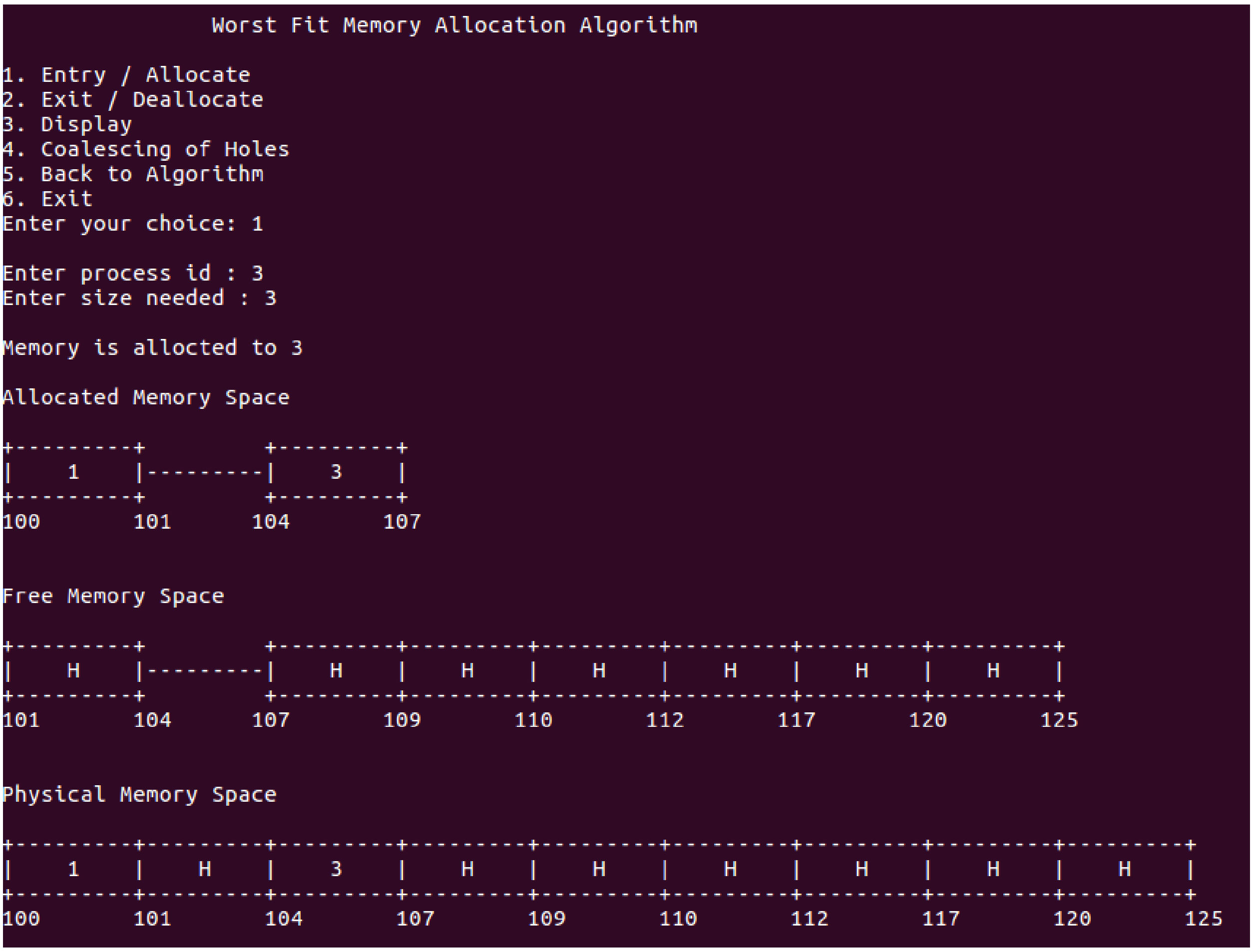
```
                Best Fit Memory Allocation Algorithm

1. Entry / Allocate
2. Exit / Deallocate
3. Display
4. Coalescing of Holes
5. Back to Algorithm
6. Exit
Enter your choice: 2

Enter process id : 2

2's memory is deallocted

Allocated Memory Space

+---------+
|    1    |
+---------+
112       116


Free Memory Space

+---------+---------+         +---------+---------+---------+---------+
|    H    |    H    |---------|    H    |    H    |    H    |    H    |
+---------+---------+         +---------+---------+---------+---------+
100       110       112      116       117       120       123       125


Physical Memory Space

+---------+---------+---------+---------+---------+---------+---------+
|    H    |    H    |    1    |    H    |    H    |    H    |    H    |
+---------+---------+---------+---------+---------+---------+---------+
100       110       112       116       117       120       123       125
```



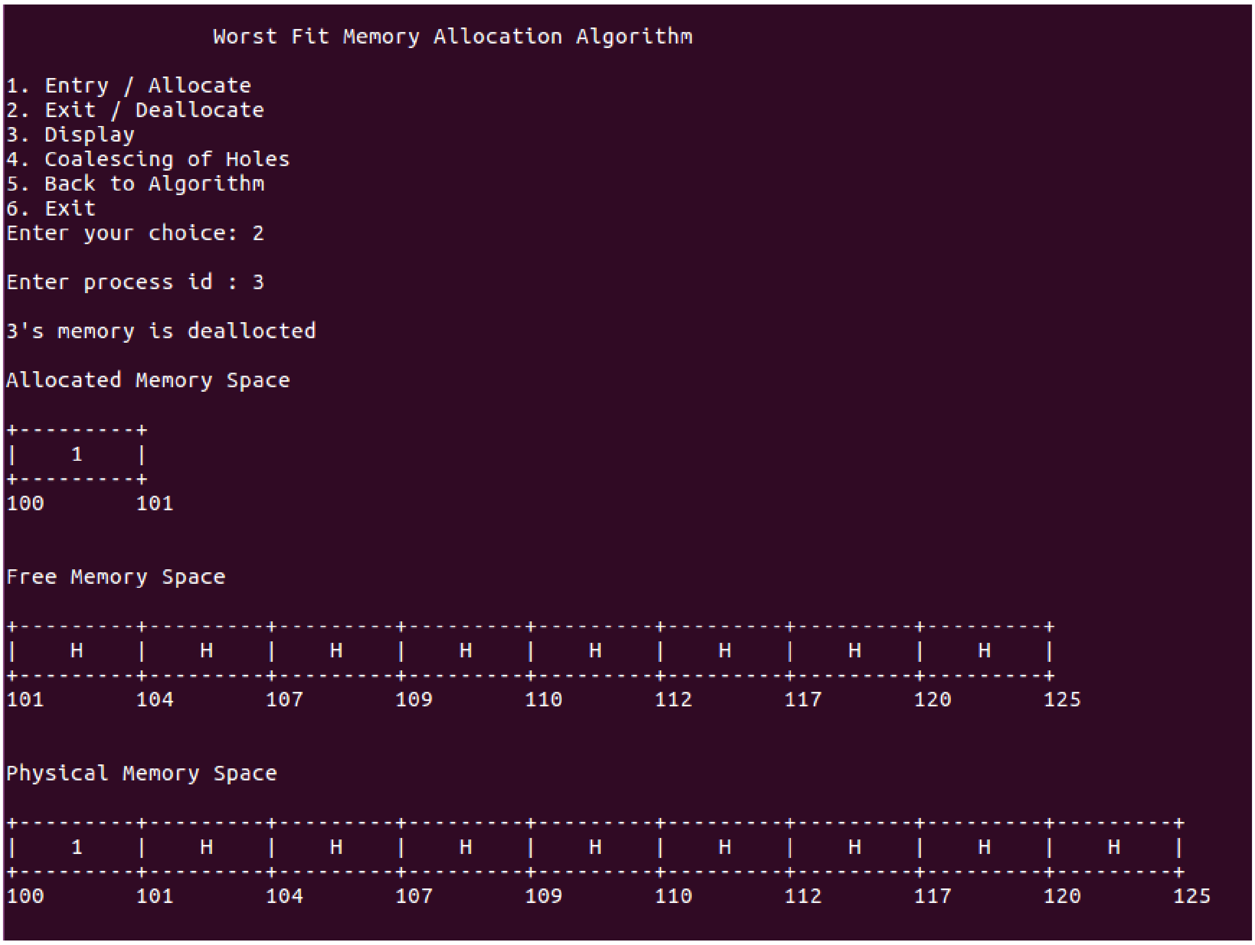
```
                Best Fit Memory Allocation Algorithm

1. Entry / Allocate
2. Exit / Deallocate
3. Display
4. Coalescing of Holes
5. Back to Algorithm
6. Exit
Enter your choice: 2

Enter process id : 1

1's memory is deallocted

Allocated Memory Space

NULL

Free Memory Space

+---------+---------+---------+---------+---------+---------+---------+
|    H    |    H    |    H    |    H    |    H    |    H    |    H    |
+---------+---------+---------+---------+---------+---------+---------+
100       110       112       116       117       120       123       125


Physical Memory Space

+---------+---------+---------+---------+---------+---------+---------+
|    H    |    H    |    H    |    H    |    H    |    H    |    H    |
+---------+---------+---------+---------+---------+---------+---------+
100       110       112       116       117       120       123       125
```



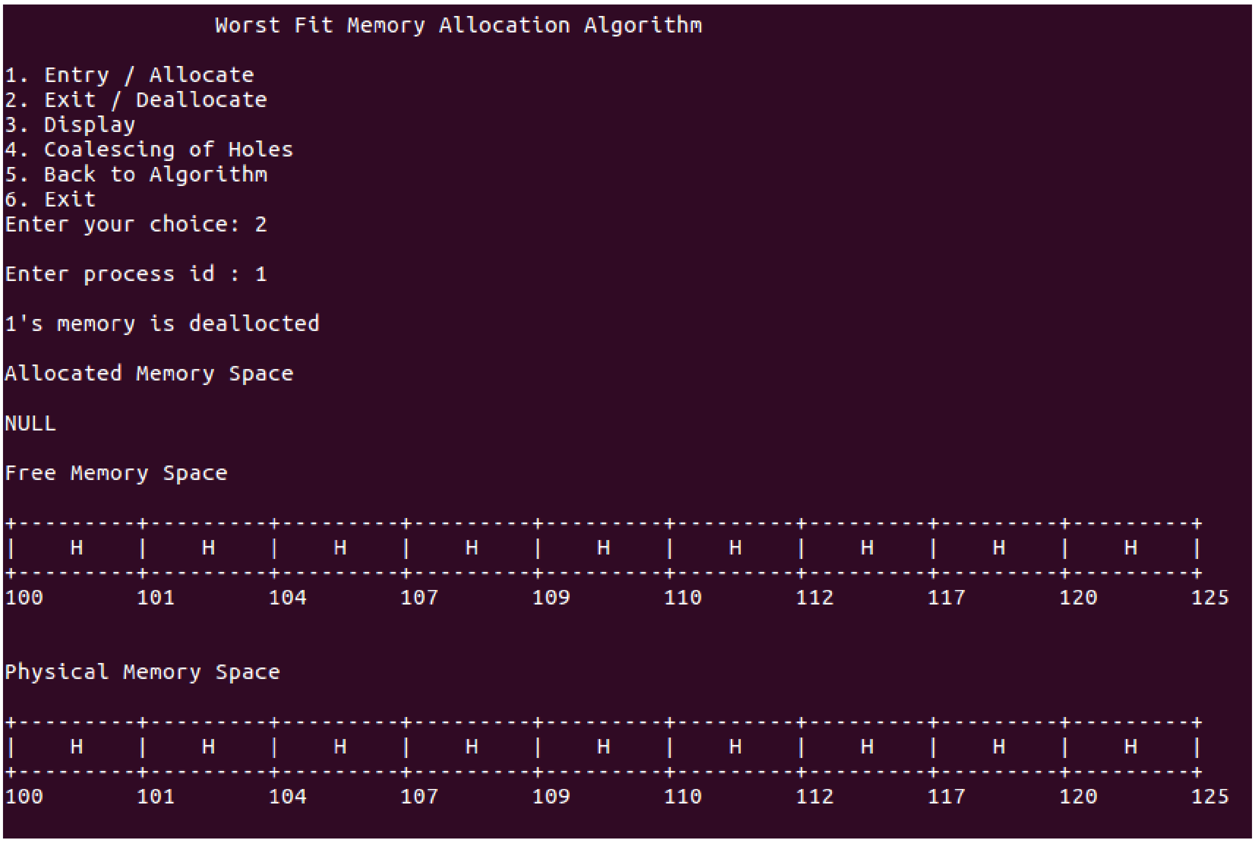
```
                Worst Fit Memory Allocation Algorithm

1. Entry / Allocate
2. Exit / Deallocate
3. Display
4. Coalescing of Holes
5. Back to Algorithm
6. Exit
Enter your choice: 1

Enter process id : 1
Enter size needed : 1

Memory is allocted to 1

Allocated Memory Space

+---------+
|    1    |
+---------+
100       101


Free Memory Space

+---------+---------+---------+---------+---------+
|    H    |    H    |    H    |    H    |    H    |
+---------+---------+---------+---------+---------+
101       110       112       117       120       125


Physical Memory Space

+---------+---------+---------+---------+---------+---------+
|    1    |    H    |    H    |    H    |    H    |    H    |
+---------+---------+---------+---------+---------+---------+
100       101       110       112       117       120       125
```



```
                Worst Fit Memory Allocation Algorithm

1. Entry / Allocate
2. Exit / Deallocate
3. Display
4. Coalescing of Holes
5. Back to Algorithm
6. Exit
Enter your choice: 1

Enter process id : 3
Enter size needed : 3

Memory is allocted to 3

Allocated Memory Space

+---------+         +---------+
|    1    |---------|    3    |
+---------+         +---------+
100       101      104       107


Free Memory Space

+---------+         +---------+---------+---------+---------+---------+---------+
|    H    |---------|    H    |    H    |    H    |    H    |    H    |    H    |
+---------+         +---------+---------+---------+---------+---------+---------+
101       104      107       109       110       112       117       120       125


Physical Memory Space

+---------+---------+---------+---------+---------+---------+---------+---------+---------+
|    1    |    H    |    3    |    H    |    H    |    H    |    H    |    H    |    H    |
+---------+---------+---------+---------+---------+---------+---------+---------+---------+
100       101       104       107       109       110       112       117       120       125
```



```
                Worst Fit Memory Allocation Algorithm

1. Entry / Allocate
2. Exit / Deallocate
3. Display
4. Coalescing of Holes
5. Back to Algorithm
6. Exit
Enter your choice: 2

Enter process id : 3

3's memory is deallocted

Allocated Memory Space

+---------+
|    1    |
+---------+
100       101


Free Memory Space

+---------+---------+---------+---------+---------+---------+---------+---------+
|    H    |    H    |    H    |    H    |    H    |    H    |    H    |    H    |
+---------+---------+---------+---------+---------+---------+---------+---------+
101       104       107       109       110       112       117       120       125


Physical Memory Space

+---------+---------+---------+---------+---------+---------+---------+---------+---------+
|    1    |    H    |    H    |    H    |    H    |    H    |    H    |    H    |    H    |
+---------+---------+---------+---------+---------+---------+---------+---------+---------+
100       101       104       107       109       110       112       117       120       125
```



```
                Worst Fit Memory Allocation Algorithm

1. Entry / Allocate
2. Exit / Deallocate
3. Display
4. Coalescing of Holes
5. Back to Algorithm
6. Exit
Enter your choice: 2

Enter process id : 1

1's memory is deallocted

Allocated Memory Space

NULL

Free Memory Space

+---------+---------+---------+---------+---------+---------+---------+---------+---------+
|    H    |    H    |    H    |    H    |    H    |    H    |    H    |    H    |    H    |
+---------+---------+---------+---------+---------+---------+---------+---------+---------+
100       101       104       107       109       110       112       117       120       125


Physical Memory Space

+---------+---------+---------+---------+---------+---------+---------+---------+---------+
|    H    |    H    |    H    |    H    |    H    |    H    |    H    |    H    |    H    |
+---------+---------+---------+---------+---------+---------+---------+---------+---------+
100       101       104       107       109       110       112       117       120       125
```

Learning Outcome:

- Learnt how to allocate memory for processes
- Learnt to manipulate memory and linked lists