**Manual**

**Heap management simulator using C++ for First-Fit and Best-Fit strategies.**

**Splitting policy used -** split if required space is less than the space of allocated block.

**Coalescing policy used** - Coalesce each time free is called.

**Data structure used** : Doubly linked list

**Implicit List structure :**

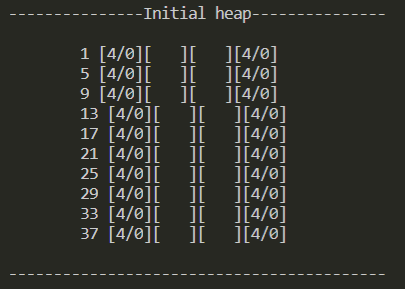
* Header at the start and footer at the end is associated with each block for the purpose of **bidirectional coalescing**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 4/0 |  |  | 4/0 | 4/0 |  |  | 4/0 | 4/0 |  |  | 4/0 |

**. . . . . . . . . . .**

Footerr

Header

****

* A address/ID is also associated with each block which is the starting address of the block.

**Assumptions:**

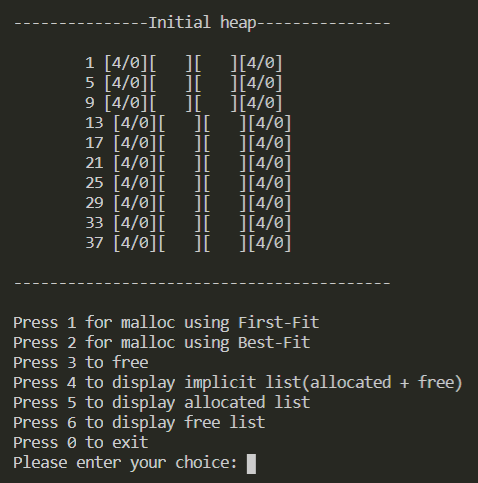
1. The size of heap is fixed as 40 bytes.
2. The header and footer of each block stores the size of block and allocation status (i.e. allocated /free).
3. A address/ID is assumed for each block to demonstrate **free**  action over a block.
4. Our simulation can take only 4 types of values i.e. **short (2 bytes)**, **integer (4bytes)**, **double (8bytes)** and **float (16 bytes)**.

**Precaution:**

* Program has a sub-menu which asks user to enter value of certain data-type but the program does not checks for the data-type of the entered value. **So user should note that, enter only that type of value which the user has opt for from the sub-menu.**

**Program features:**

When we runs the program, we will get the initial status of heap along with menu of different functions that can be performed over the implicit list.

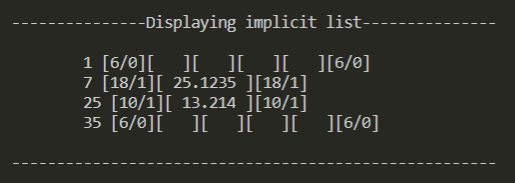


Our program have 6 main functions to perform:

1. Allocation using **first fit** strategy.
2. Allocation using **best fit** strategy.
3. **Freeing** block in the heap.
4. Displaying **implicit list (Allocated + Free)**.
5. Displaying only **Allocated list**.
6. Displaying only **free list**.

**Demonstration of First-Fit:**

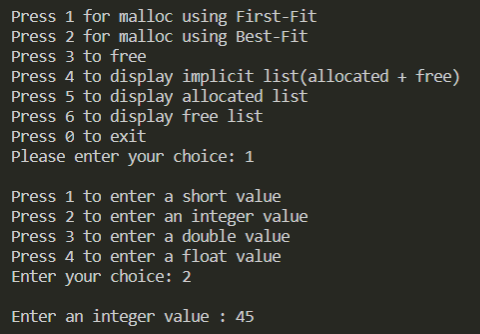
Let after performing **malloc( )** and f**ree( )** operations a certain number of times, the status of heap be as shown:

****

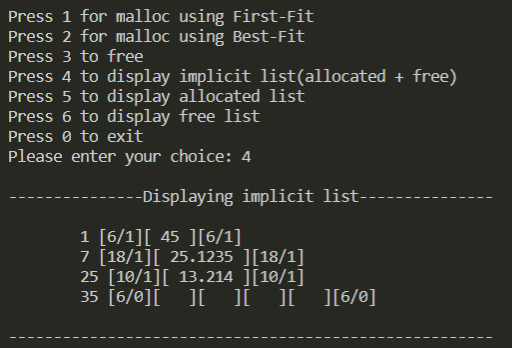
4 Bytes are free in block whose starting address is 1 and another 4 Bytes are free in block whose starting address is 35.

Now let us try to allocate space for an integer (required space = 4 bytes) using First-Fit.

1. Press “1” to use First-Fit, then press “2” to allocate space for an integer and then insert an integer value.



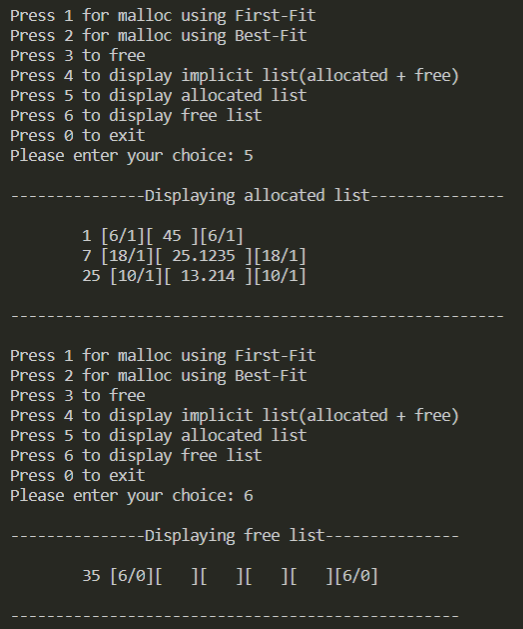
1. Now press “4” to see the changes in **implicit list (allocated + free)**

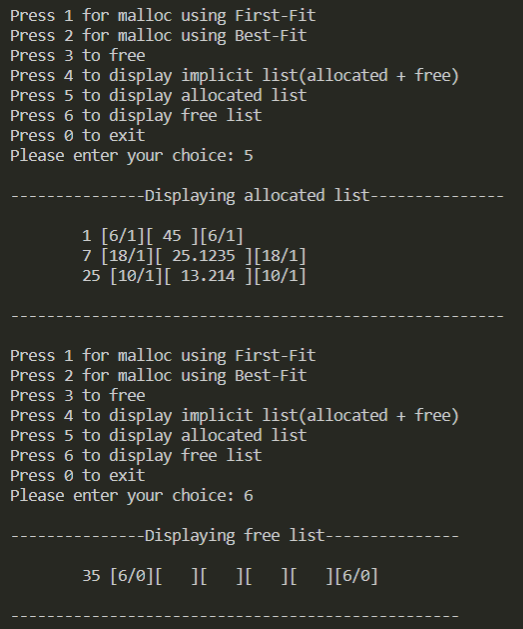
****

Integer value 45 got space in heap according to **first-fit strategy**.

So basically, if we want to allocate the memory for some integer using **First-Fit strategy**,then the program will traverse the list and allocate the space as soon as it finds free space which is greater than or equal to the space that is needed by the integer i.e.(4 bytes for integer + 2 bytes for header and footer information).

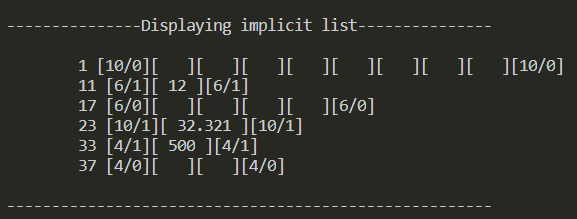
1. To see the change in allocated and free list, press “5” and “6”.





**Demonstration of Best-Fit:**

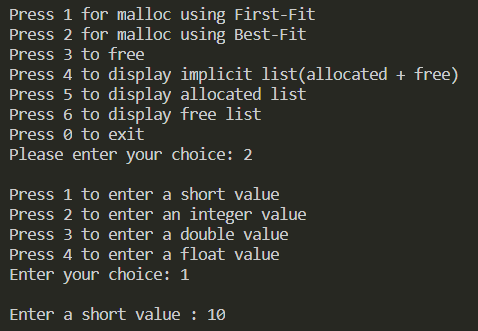
Let after performing **malloc( )** and **free( )** operations a certain number of times, the status of heap be as shown**:**

****

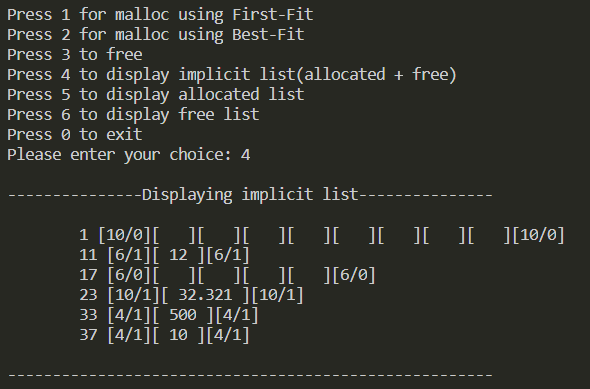
8 Bytes are free in block with starting address 1, 4 Bytes are free in block with starting address 17 and 2 Bytes are free in block with starting address 37.

Now let us try to allocate space for a short value (required space = 2 Bytes) using **Best-Fit**:

1. Press “2” to use **Best-Fit**, then press “1” to allocate space for a short and then insert a short value.

****

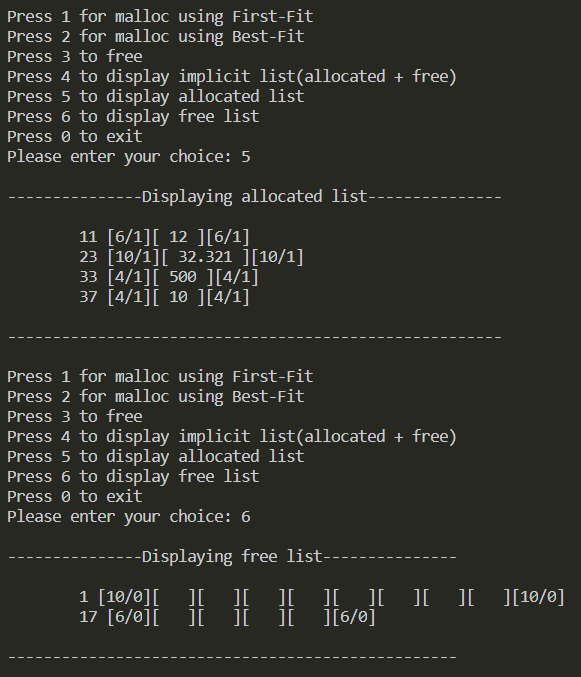
1. Now press “4” to see the change in **implicit list (allocated + free)**

****

Short value 10 got space in heap according to Best-Fit strategy.

So this is how memory is allocated to a short value using **Best-Fit** strategy which searches the complete list and finds the free block having space closest to the required amount of space so as to use the memory efficiently.

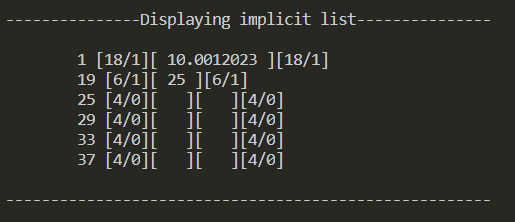
1. To see the change in **allocated** and **free list**, press “5” and “6”.



**Coalesce free blocks during allocation:**

If the size of all free blocks is less than the required space for the allocation of a data value, then free blocks will be coalesced according to the allocation strategy used.

Ex. Let the status of heap be as shown :



Here we have 4 free blocks each having two bytes of free space. Now if we wish to allocate

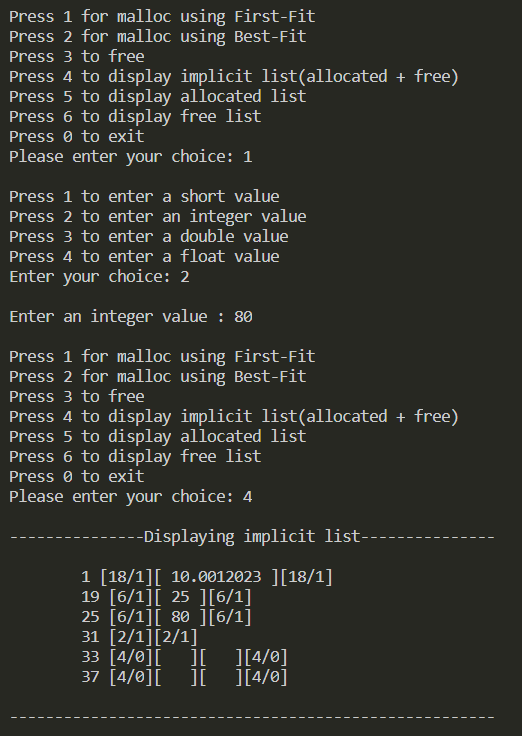
space for an integer using **First-Fit**, then we will find that none of the free block can hold an

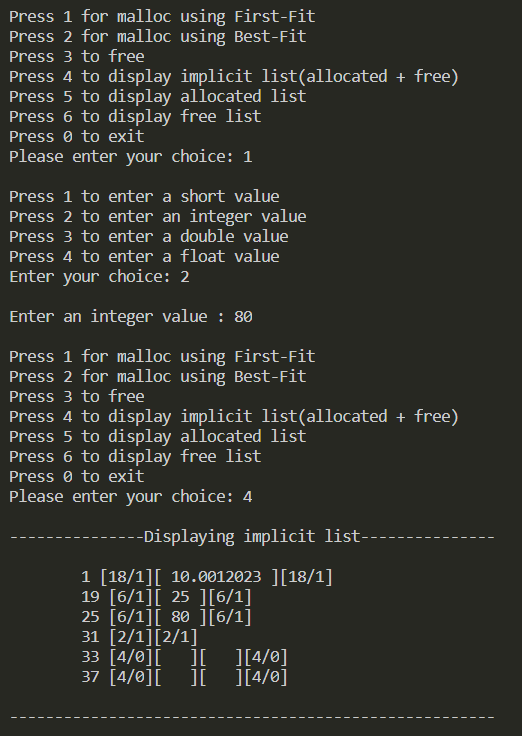
integer value which requires **4 Bytes**.

So to fulfill the requirement of space our program coalesce adjacent free blocks according to

the allocation strategy used.

So now lets allocate space for an integer value.





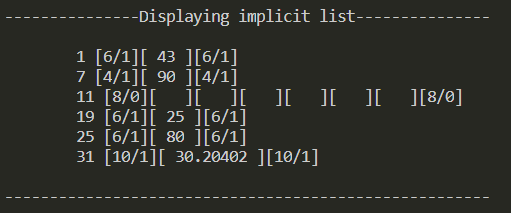
Blocks with starting address 25 & 29 were coalesced to provide 4 Bytes of continuous space to store an integer value.

**Freeing an allocated block :**

While freeing an allocated block our program checks the allocation status of the adjacent

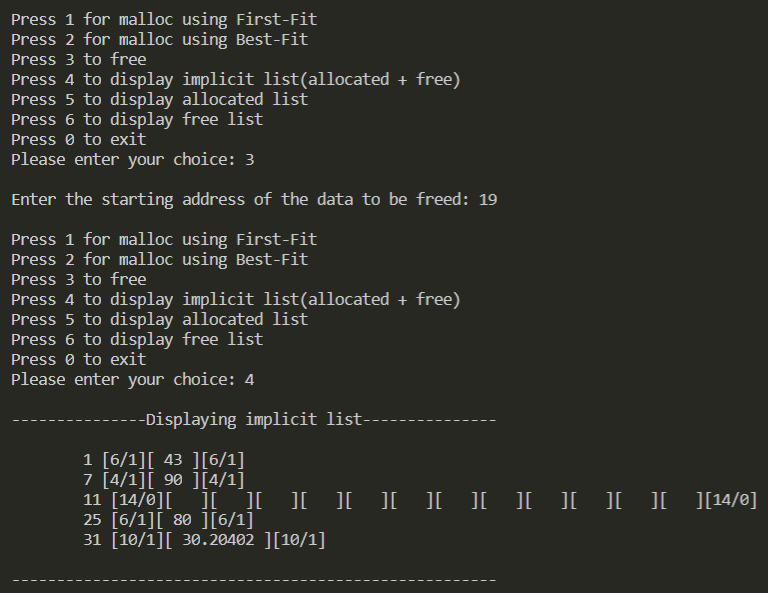
blocks so as to coalesce the block which is being freed and already free adjacent blocks.

Let the status of heap be as shown:



1. Now press “3” to free an allocated block
2. Then enter the starting address of block to be freed
3. Then press “4”, “5” & “6” to see then change in the **implicit list**, **allocated list** and

**free list** respectively.



Here, as soon as the block with starting address “19” got freed, it got coalesced with adjacent

free block with starting address “11”.

