Computer Science and Engineering & Information Technology (2^{nd} Year B.Tech.)
IIIT Kalyani, West Bengal

Operating System Lab (CS 411): (Spring: 2019-2020)

Assignment - 12 *Marks: 10*
Assignment Out: *10^{th} April, 2020*

This one is going to be the last assignment of the OS laboratory. You will implement your own *malloc()* (`myMalloc()`) and *free()* (`myFree()` functions on a block of acquired memory. The main use of it will be for the shared memory. The starting address of the acquired shared memory is copied to different processes and will remain unchanged.

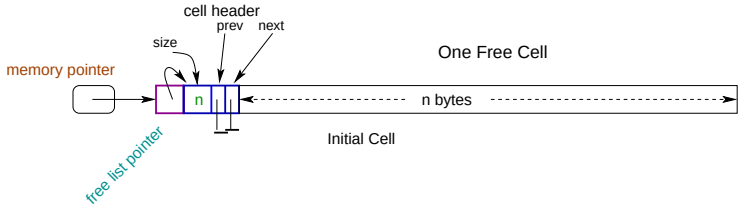
A program calls `void *myMalloc(int n);` to get `n` Bytes of memory. It calls `void myFree(void *p);` to free the block of memory acquired and pointed by `p` (do not bother about the situation where the parameter `p` is arbitrary).

The acquired memory at any point is divided into *free memory cells* and *used areas*. The list of free memory cells is maintained by an *in memory* doubly linked list. A pointer to the *free memory cell list* is also maintained in the shared memory.

The type of header of a free memory cell and function interfaces are as follows:

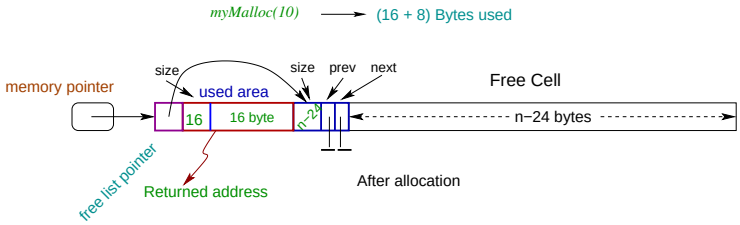
```
typedef struct myHeap{
        int size;
        struct myHeap *next;
        struct myHeap *prev;
} myHeap_t;

extern myHeap_t **memoryP;

void initMyHeap();
void *myMalloc(int);
void myFree(void *);
void mergeFL();
```

- `size`: size of the free space available in a cell in bytes.
- `next`: pointer to the next free memory cell. It is `NULL` if there is no next cell (the last cell).
- `prev`: pointer to the previous free memory cell. It is `NULL` if there is no previous cell (the first cell).
- `memoryP`: global pointer to the acquired memory block. This will be shared by different processes and will not change. The first 8 bytes of the block is used as the free memory cell list *header*. So its type is `myHeap_t **`, but that is not a necessity.
- `initMyHeap()`: initializes the memory to a big free cell. It looks like the following.



- The function `myMalloc()` allocates space in multiple of 8 bytes. It also maintains a field to store the *size* of the *useful space* allocated. It returns the starting address of the *useful space*. If the requested space cannot be allocated, it returns `NULL`.

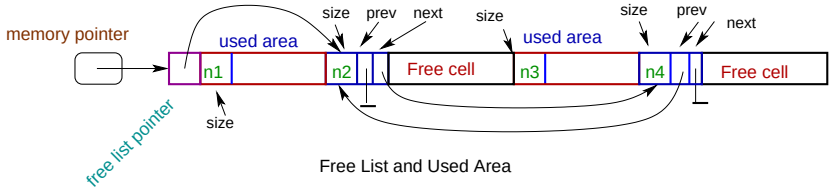
  So after a call `myMalloc(10)`, (16+8) bytes are taken out of the initial free cell space, and it looks as follows.



In fact `myMalloc(n)` traverses the free cell list to find a free cell of suitable size ($> n$). You may use *first fit* or any other algorithm. After allocation it is necessary to update the the free cell list.

- The function `myFree(ptr)` will free the memory block pointed by `ptr` and inserts it in the free cell list in proper place. Note that `ptr` is not the actual starting address of the block. It is 8 byte larger than that (due to the `size` field).

  The memory block looks as follows at some intermediate point.



- The function `mergeFL()` is used to merge contiguous free cells to get a larger free cell.

You need to write the above mentioned four functions. The user program is supplied. Do not modify it, also do not change any function interface. Prepare a makefile and return the assignment.

A user program to test your code is as follows:

```
// userProg.c++
#include <iostream>
using namespace std;
#include "myHeap.h"
#include <cstdlib>
#include <sys/ipc.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/shm.h>
#include <sys/wait.h>

myHeap_t **memoryP;

typedef struct node{
        int data;
        struct node *next;
} node_t;

void printFreeList(myHeap_t *);

int main(){
    int shmID, n, pid, status;
    struct shmid_ds buff;
    node_t **lP;

    shmID = shmget(IPC_PRIVATE, SIZE+sizeof(node_t *), IPC_CREAT | 0777);
    lP = (node_t **)shmat(shmID, 0, 0777);
    memoryP = (myHeap **)(lP+1);
    initMyHeap();
```

```
    cout << "Enter a positive integer: ";
    cin >> n;

    if((pid = fork()) == -1){
      cerr << "fork() fails\n";
      exit(1);
    }
    if(pid == 0){  // child
       node_t *l = NULL, *nP;

       cout << "Enter " << n << " data\n";
       for(int i=0; i<n; ++i){
          nP = (node_t *)myMalloc(sizeof(node_t));
          cin >> nP->data;
          nP->next = l;
          l = nP;
        }
        *lP = l;
    }
    else{   // parent
         node_t *nP;

         waitpid(pid, &status, 0);

         nP=*lP;
         while(nP) {
              cout << nP->data << " ";
              nP = nP->next;
         }
         cout << endl;

//  Testing free ....

        cout << "memoryP: " << hex << memoryP << endl;
        initMyHeap();
        printFreeList(*memoryP);
        void *rp=myMalloc(100);
        cout << "rp: " << rp << endl;
        printFreeList(*memoryP);
        myFree(rp);
        printFreeList(*memoryP);
        rp=myMalloc(150);
        cout << "rp: " << rp << endl;
        printFreeList(*memoryP);
        myFree(rp);
        printFreeList(*memoryP);
        mergeFL();
        printFreeList(*memoryP);


         shmdt(memoryP);
         shmctl(shmID, IPC_RMID, &buff);
    }
    return 0;
}

void printFreeList(myHeap_t *l){
     cout << "Free list: ";
     while(l){
          cout << hex << l << ", ";
          cout << " size: " << l->size << ", ";
          l = l->next;
     }
```

```
    cout << endl;
}
```

A sample run:

```
$ ./a.out
Enter a positive integer: 5
Enter 5 data
-2 -1 0 1 2
2 1 0 -1 -2
memoryP: 0x7fc74c7c2008
Free list: 0x7fc74c7c2010,  size: 3fe0,
rp: 0x7fc74c7c2018
Free list: 0x7fc74c7c2080,  size: 3f70,
Free list: 0x7fc74c7c2010,  size: 58, 0x7fc74c7c2080,  size: 3f70,
rp: 0x7fc74c7c2088
Free list: 0x7fc74c7c2010,  size: 58, 0x7fc74c7c2120,  size: 3ed0,
Free list: 0x7fc74c7c2010,  size: 58, 0x7fc74c7c2080,  size: 88, 0x7fc74c7c2120,  size: 3e
Free list: 0x7fc74c7c2010,  size: 3fe0,
```