OS Lab1：可变分区存储管理

1 实验题目

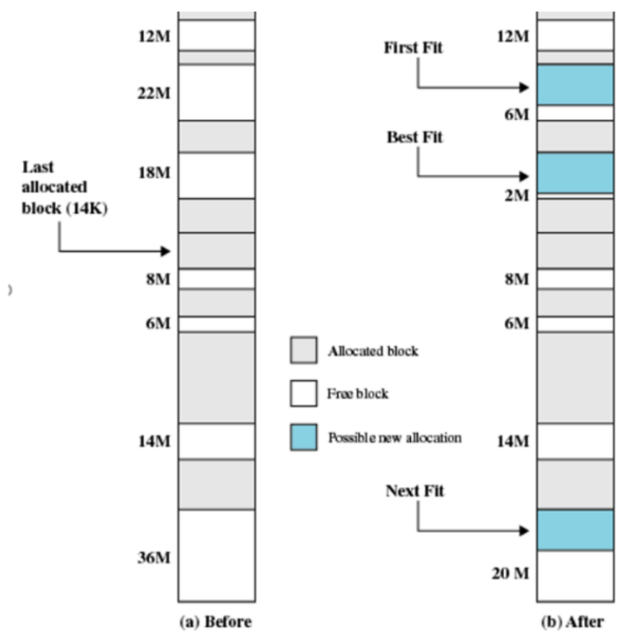
编写一个C语言程序，模拟UNIX的可变分区内存管理，使用**循环首次适应法**实现对一块内存区域的分配和释放管理。

2 算法思想及概要设计

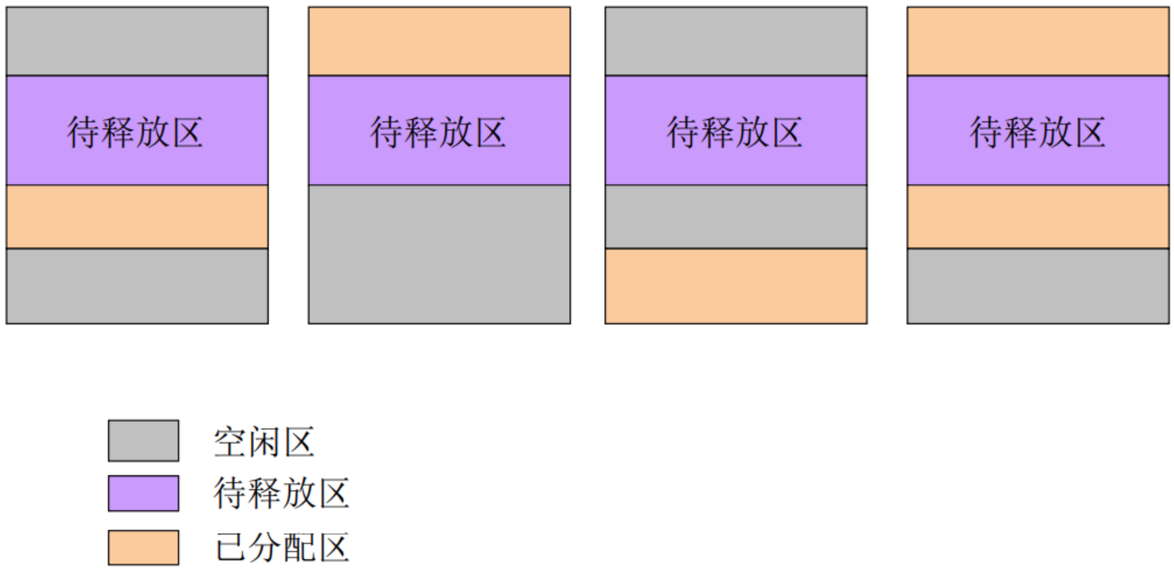
2.1 算法思想

2.1.1 分配算法

在分配内存空间时，不再每次从表头（链首）开始查找，而是从上次找到空闲区的下一个空闲开始查找，直到找到第一个能满足要求的空闲区为止，并从中划出一块与请求大小相等的内存空间分配给作业。该算法能使内存中的空闲区分布得较均匀。如下图所示。



2.1.2 回收算法



如上图所示，内存回收分为四种情况：

- 待释放区与上一空闲区不邻接、与下一空闲区邻接：下一空闲区向前扩展，扩展包含待释放区
- 待释放区与上一空闲区邻接、与下一空闲区不邻接：上一空闲区向后扩展，包含待释放区
- 待释放区与上一空闲区不邻接、与下一空闲区不邻接：待释放区形成新的空闲区
- 待释放区与上一空闲区邻接、与下一空闲区邻接：待释放区与上下两个分区拼接成一个更大分区

2.2 概要设计

```
.
├── main.c
├── Makefile
├── NextFit.c
├── NextFit.h
├── out
│   ├── out1
│   ├── out2
│   ├── out3
│   ├── out4
│   └── out5
├── test.sh
└── traces
    ├── trace1
    ├── trace2
    ├── trace3
    ├── trace4
    └── trace5
```

文件目录树如上。main.c包含初始化和输入输出控制功能。NextFit.c和NextFit.h包含内存操作函数。

test.sh为测试脚本，将traces中的5个文件通过程序后输出在out文件夹的5个文件中。Makefile为编译文件。

基本思想为利用双向链表对内存空闲区进行操作管理，具体逻辑、函数及接口设计如下所述。

3 重要模块的功能、详细设计以及接口说明

```
// 1. 主程序输入控制部分
char str[10]; // 申请一个字符串数组，方便后续做处理
while(scanf("%s", str) != EOF) { // 若还有输入，继续进行操作
    if(!strcmp(str, "m") || !strcmp(str, "malloc")) { // 输入m或malloc开头都可以做分配标识
        getchar(); // 空格
        int space1; // 命令参数，保存申请内存大小
        scanf("%d", &space1); // 获取参数
        char *addr1 = lmalloc(space1, &coremap, &current_loc); // 分配内存
        // 若结果非空表示操作成功，否则失败，输出对应反馈
        if(addr1) printf("Memory was successfully allocated to address %lu.\n",
            (unsigned long)(addr1));
        else printf("The operation failed.\n");
        display(coremap, start_addr);
    } else if (!strcmp(str, "f") || !strcmp(str, "free")) { // f或free都可以进行释放操作
        getchar(); // 空格
        int space2;
        int addr2; // logical address
        scanf("%d %d", &space2, &addr2); // 获取参数
        if(lfree(space2, start_addr + addr2, &coremap)) // 如果释放成功
            printf("Memory freed successfully.\n");
        else printf("Error!\n"); // 释放失败
        display(coremap, start_addr); // 显示整个列表的结果
    } else { // 表示命令不匹配
        fgets(str,100,stdin); // 将该行输入读完
        printf("Error!\n");
    }


// 2. 分配内存函数
// 输入：size-需要内存的大小，coremap-存放空闲分区表的地址，方便修改，
//        current_loc-当前位置地址，方便修改
// 输出：char*-分配的首地址，如果分配失败则返回NULL
char *lmalloc(unsigned size, struct maplist *coremap, struct map* *current_loc) {
    char *a;
    struct map *bp = *current_loc;

    while(1) {
        if (bp->m_size >= size) { // 找到空闲分区大小大于需要内存大小的
            a = bp->m_addr; // 地址记录
            bp->m_addr += size; // 空闲分区新地址为原来地址加上size
            bp->m_size -= size; // 空闲分区大小缩减

            if (bp->m_size == 0) { // 如果大小为0，该空闲分区应当删除
                if (coremap->len == 1) { // 如果只有一个空闲分区
                    coremap->addr = NULL;
                    struct map *dp = bp;
                    free(dp);
                } else { // 如果有多个空闲分区
                    bp->prior->next = bp->next;
                    bp->next->prior = bp->prior;
```

```
                    struct map *dp = bp;
                    free(dp);
                }
                coremap->len--; // 空闲分区元素个数减少
            }
            *current_loc = bp; // 更新当前位置
            return a;
        }
        bp = bp->next; // 找下一个分区
        if(bp == *current_loc) // 循环过一轮，不再进行查找
            break;
    }

    return NULL;
}


// 3. 释放内存函数
// 输入：size-释放内存的大小，coremap-存放空闲分区表的地址，方便修改，
//       addr-释放的起始地址
// 输出：int-操作成功为1，失败为0
int lfree(unsigned size, char* addr, struct maplist *coremap) {
    if(size <= 0) return false; // size小于零检查

    struct map *bp = coremap->addr; // 起始位置

    // 找到该内存分区下一个空闲分区地址为bp，上一个为bp->prior
    while (bp->m_addr <= addr) {
        bp = bp->next;
        if(bp == coremap->addr)
            break;
    }

    if (bp->prior->m_addr + bp->prior->m_size == addr && addr + size != bp->m_addr) {
        // case 1 前邻接后不邻接
        bp->prior->m_size += size;
        return true;
    } else if (bp->prior->m_addr + bp->prior->m_size == addr && addr + size == bp->m_addr) {
        // case 2 前后皆邻接
        bp->prior->m_size += size + bp->m_size;
        bp->prior->next = bp->next;
        bp->next->prior = bp->prior;
        if(coremap->addr == bp) coremap->addr =  bp->prior;
        coremap->len--;
        // 释放下一个空闲分区
        struct map *dp = bp;
        free(dp);
        return true;
    } else if (bp->prior->m_addr + bp->prior->m_size != addr && addr + size == bp->m_addr) {
        // case 3 前不邻接，后邻接
        bp->m_size += size;
        bp->m_addr -= size;
        return true;
    } else if (bp->prior->m_addr + bp->prior->m_size != addr && addr + size != bp->m_addr) {
        // case 4 前后皆不邻接
        struct map *p = (struct map*)malloc(sizeof(struct map));
        p->m_size = size;
        p->m_addr = addr;
        p->next = bp;
        p->prior = bp->prior;
        bp->prior->next = p;
        bp->prior = p;
        coremap->len++;
```

```
        if(addr < coremap->addr->m_addr) coremap->addr = p;
        return true;
    }
    return false;
}


// 4. 打印空闲分区表函数
// 输入：m-空闲分区表，start_addr-起始地址，用于计算逻辑地址
void display(struct maplist m, char *start_addr) {
    printf("--------------------------------------------------------------------------------\n");
    printf("\t\tCurrent Status(Start Address: %lu)\n", (unsigned long)start_addr);
    printf("--------------------------------------------------------------------------------\n");
    struct map *bp = m.addr;
    for (int i = 0;; i++) {
        // 输出物理地址，逻辑地址与分区大小
        printf("    Item %d: [Physical Addr: %lu; Logical Addr: %lu; Size: %u]\n",
               i, (unsigned long)bp->m_addr, (unsigned long)(bp->m_addr - start_addr), bp->m_size);
        bp = bp->next;
        if(bp == m.addr)
            break;
    }
    printf("--------------------------------------------------------------------------------\n\n");
}
```

```
# Makefile
# 编译链接各个文件，形成可执行文件

cc = gcc
prom = NextFit
deps = $(shell find ./ -name "*.h")
src = $(shell find ./ -name "*.c")
obj = $(src:%.c=%.o)

$(prom): $(obj)
  $(cc) -o $(prom) $(obj)

%.o: %.c $(deps)
  $(cc) -c $< -o $@

clean:
  rm -rf $(obj) $(prom)
```

4 重要数据结构及变量的说明

```
// map保存一个空闲分区的有关信息
struct map {
    unsigned m_size; // 空闲分区的大小
    char *m_addr; // 空闲分区起始地址
    struct map *next, *prior; // 前向后向指针，指向相邻空闲分区
};

// 整个空闲分区列表的管理
struct maplist {
```

```
    struct map *addr; // 指向第一个空闲分区的指针
    int len; // 空闲分区的个数
};

// 初始化一个空闲分区表
static struct maplist coremap;
coremap.addr = (struct map*)malloc(sizeof(struct map)); // 申请一个空闲分区
coremap.addr->m_size = 1000; // 第一个空闲分区的大小，初始为1000
coremap.addr->m_addr = (char*) malloc(1000); // 申请1000字节的数据
coremap.addr->next = coremap.addr->prior = coremap.addr; // 前后指针指向自己

// 保存一个表明当前位置的指针
static struct map *current_loc;
current_loc = coremap.addr; // 初始化为起始地址

// 保存起始地址
char *start_addr = coremap.addr->m_addr;
```

5 测试方法及结果

在traces文件夹中保存五个输入文件，前4个文件分别对应内存释放的四种情况，用来检查逻辑正确性，第五个用例为综合用例，较为复杂，来检查其他逻辑错误。

```
// trace1
m 300
m 300
f 300 0
f 300 100

// trace2
m 300
m 300
f 300 0
f 300 300

// trace3
m 300
m 300
f 300 0
f 500 100

// trace4
m 300
m 300
f 300 0
f 400 100

// trace5
m 100
m 200
m 200
f 200 100
m 400
m 50
f 400 500
m 175
```

```
f 275 0
f 200 300
f 50 900
```

测试脚本如下：

```
#!/bin/bash

# 测试对5个输入文件的输出是否正确，结果保存在out中

for i in 1 2 3 4 5
do
    ./NextFit < "traces/trace$i" > "out/out$i"
done
```

输出结果将保存在out文件夹的五个文件中。

结果分别为：

```
// out1
Memory was successfully allocated to address 94295005819600.
--------------------------------------------------------------------------------
    Current Status(Start Address: 94295005819600)
--------------------------------------------------------------------------------
    Item 0: [Physical Addr: 94295005819900; Logical Addr: 300; Size: 700]
--------------------------------------------------------------------------------

Memory was successfully allocated to address 94295005819900.
--------------------------------------------------------------------------------
    Current Status(Start Address: 94295005819600)
--------------------------------------------------------------------------------
    Item 0: [Physical Addr: 94295005820200; Logical Addr: 600; Size: 400]
--------------------------------------------------------------------------------

Memory freed successfully.
--------------------------------------------------------------------------------
    Current Status(Start Address: 94295005819600)
--------------------------------------------------------------------------------
    Item 0: [Physical Addr: 94295005819600; Logical Addr: 0; Size: 300]
    Item 1: [Physical Addr: 94295005820200; Logical Addr: 600; Size: 400]
--------------------------------------------------------------------------------

Memory freed successfully.
--------------------------------------------------------------------------------
    Current Status(Start Address: 94295005819600)
--------------------------------------------------------------------------------
    Item 0: [Physical Addr: 94295005819600; Logical Addr: 0; Size: 300]
    Item 1: [Physical Addr: 94295005819700; Logical Addr: 100; Size: 300]
    Item 2: [Physical Addr: 94295005820200; Logical Addr: 600; Size: 400]
--------------------------------------------------------------------------------

//out2
Memory was successfully allocated to address 94588975616720.
--------------------------------------------------------------------------------
    Current Status(Start Address: 94588975616720)
--------------------------------------------------------------------------------
    Item 0: [Physical Addr: 94588975617020; Logical Addr: 300; Size: 700]
```

```
--------------------------------------------------------------------------------

Memory was successfully allocated to address 94588975617020.
--------------------------------------------------------------------------------
    Current Status(Start Address: 94588975616720)
--------------------------------------------------------------------------------
    Item 0: [Physical Addr: 94588975617320; Logical Addr: 600; Size: 400]
--------------------------------------------------------------------------------

Memory freed successfully.
--------------------------------------------------------------------------------
    Current Status(Start Address: 94588975616720)
--------------------------------------------------------------------------------
    Item 0: [Physical Addr: 94588975616720; Logical Addr: 0; Size: 300]
    Item 1: [Physical Addr: 94588975617320; Logical Addr: 600; Size: 400]
--------------------------------------------------------------------------------

Memory freed successfully.
--------------------------------------------------------------------------------
    Current Status(Start Address: 94588975616720)
--------------------------------------------------------------------------------
    Item 0: [Physical Addr: 94588975616720; Logical Addr: 0; Size: 1000]
--------------------------------------------------------------------------------

// out3
Memory was successfully allocated to address 94416195089104.
--------------------------------------------------------------------------------
    Current Status(Start Address: 94416195089104)
--------------------------------------------------------------------------------
    Item 0: [Physical Addr: 94416195089404; Logical Addr: 300; Size: 700]
--------------------------------------------------------------------------------

Memory was successfully allocated to address 94416195089404.
--------------------------------------------------------------------------------
    Current Status(Start Address: 94416195089104)
--------------------------------------------------------------------------------
    Item 0: [Physical Addr: 94416195089704; Logical Addr: 600; Size: 400]
--------------------------------------------------------------------------------

Memory freed successfully.
--------------------------------------------------------------------------------
    Current Status(Start Address: 94416195089104)
--------------------------------------------------------------------------------
    Item 0: [Physical Addr: 94416195089104; Logical Addr: 0; Size: 300]
    Item 1: [Physical Addr: 94416195089704; Logical Addr: 600; Size: 400]
--------------------------------------------------------------------------------

Memory freed successfully.
--------------------------------------------------------------------------------
    Current Status(Start Address: 94416195089104)
--------------------------------------------------------------------------------
    Item 0: [Physical Addr: 94416195089104; Logical Addr: 0; Size: 300]
    Item 1: [Physical Addr: 94416195089204; Logical Addr: 100; Size: 900]
--------------------------------------------------------------------------------

// out4
Memory was successfully allocated to address 94317657981648.
--------------------------------------------------------------------------------
    Current Status(Start Address: 94317657981648)
--------------------------------------------------------------------------------
    Item 0: [Physical Addr: 94317657981948; Logical Addr: 300; Size: 700]
--------------------------------------------------------------------------------
```

```
Memory was successfully allocated to address 94317657981948.
--------------------------------------------------------------------------------
    Current Status(Start Address: 94317657981648)
--------------------------------------------------------------------------------
    Item 0: [Physical Addr: 94317657982248; Logical Addr: 600; Size: 400]
--------------------------------------------------------------------------------

Memory freed successfully.
--------------------------------------------------------------------------------
    Current Status(Start Address: 94317657981648)
--------------------------------------------------------------------------------
    Item 0: [Physical Addr: 94317657981648; Logical Addr: 0; Size: 300]
    Item 1: [Physical Addr: 94317657982248; Logical Addr: 600; Size: 400]
--------------------------------------------------------------------------------

Memory freed successfully.
--------------------------------------------------------------------------------
    Current Status(Start Address: 94317657981648)
--------------------------------------------------------------------------------
    Item 0: [Physical Addr: 94317657981648; Logical Addr: 0; Size: 300]
    Item 1: [Physical Addr: 94317657981748; Logical Addr: 100; Size: 400]
    Item 2: [Physical Addr: 94317657982248; Logical Addr: 600; Size: 400]
--------------------------------------------------------------------------------

// out5
Memory was successfully allocated to address 93881157915344.
--------------------------------------------------------------------------------
    Current Status(Start Address: 93881157915344)
--------------------------------------------------------------------------------
    Item 0: [Physical Addr: 93881157915444; Logical Addr: 100; Size: 900]
--------------------------------------------------------------------------------

Memory was successfully allocated to address 93881157915444.
--------------------------------------------------------------------------------
    Current Status(Start Address: 93881157915344)
--------------------------------------------------------------------------------
    Item 0: [Physical Addr: 93881157915644; Logical Addr: 300; Size: 700]
--------------------------------------------------------------------------------

Memory was successfully allocated to address 93881157915644.
--------------------------------------------------------------------------------
    Current Status(Start Address: 93881157915344)
--------------------------------------------------------------------------------
    Item 0: [Physical Addr: 93881157915844; Logical Addr: 500; Size: 500]
--------------------------------------------------------------------------------

Memory freed successfully.
--------------------------------------------------------------------------------
    Current Status(Start Address: 93881157915344)
--------------------------------------------------------------------------------
    Item 0: [Physical Addr: 93881157915444; Logical Addr: 100; Size: 200]
    Item 1: [Physical Addr: 93881157915844; Logical Addr: 500; Size: 500]
--------------------------------------------------------------------------------

Memory was successfully allocated to address 93881157915844.
--------------------------------------------------------------------------------
    Current Status(Start Address: 93881157915344)
--------------------------------------------------------------------------------
    Item 0: [Physical Addr: 93881157915444; Logical Addr: 100; Size: 200]
    Item 1: [Physical Addr: 93881157916244; Logical Addr: 900; Size: 100]
--------------------------------------------------------------------------------

Memory was successfully allocated to address 93881157916244.
```

```
--------------------------------------------------------------------------------
    Current Status(Start Address: 93881157915344)
--------------------------------------------------------------------------------
    Item 0: [Physical Addr: 93881157915444; Logical Addr: 100; Size: 200]
    Item 1: [Physical Addr: 93881157916294; Logical Addr: 950; Size: 50]
--------------------------------------------------------------------------------

Memory freed successfully.
--------------------------------------------------------------------------------
    Current Status(Start Address: 93881157915344)
--------------------------------------------------------------------------------
    Item 0: [Physical Addr: 93881157915444; Logical Addr: 100; Size: 200]
    Item 1: [Physical Addr: 93881157915844; Logical Addr: 500; Size: 400]
    Item 2: [Physical Addr: 93881157916294; Logical Addr: 950; Size: 50]
--------------------------------------------------------------------------------

Memory was successfully allocated to address 93881157915444.
--------------------------------------------------------------------------------
    Current Status(Start Address: 93881157915344)
--------------------------------------------------------------------------------
    Item 0: [Physical Addr: 93881157915619; Logical Addr: 275; Size: 25]
    Item 1: [Physical Addr: 93881157915844; Logical Addr: 500; Size: 400]
    Item 2: [Physical Addr: 93881157916294; Logical Addr: 950; Size: 50]
--------------------------------------------------------------------------------

Memory freed successfully.
--------------------------------------------------------------------------------
    Current Status(Start Address: 93881157915344)
--------------------------------------------------------------------------------
    Item 0: [Physical Addr: 93881157915344; Logical Addr: 0; Size: 300]
    Item 1: [Physical Addr: 93881157915844; Logical Addr: 500; Size: 400]
    Item 2: [Physical Addr: 93881157916294; Logical Addr: 950; Size: 50]
--------------------------------------------------------------------------------

Memory freed successfully.
--------------------------------------------------------------------------------
    Current Status(Start Address: 93881157915344)
--------------------------------------------------------------------------------
    Item 0: [Physical Addr: 93881157915344; Logical Addr: 0; Size: 900]
    Item 1: [Physical Addr: 93881157916294; Logical Addr: 950; Size: 50]
--------------------------------------------------------------------------------

Memory freed successfully.
--------------------------------------------------------------------------------
    Current Status(Start Address: 93881157915344)
--------------------------------------------------------------------------------
    Item 0: [Physical Addr: 93881157915344; Logical Addr: 0; Size: 1000]
--------------------------------------------------------------------------------
```

根据演算判断，输出过程结果正确无误。

6 代码

```
// main.c

#include "NextFit.h"
```

```c
// initialize a list
static struct maplist coremap;
// record loop position
static struct map *current_loc;

int main() {

    // initialization
    coremap.addr = (struct map*)malloc(sizeof(struct map));
    coremap.addr->m_size = 1000;
    coremap.addr->m_addr = (char*) malloc(1000); // malloc 1000 bytes storage
    coremap.addr->next = coremap.addr->prior = coremap.addr;

    current_loc = coremap.addr;
    char *start_addr = coremap.addr->m_addr; // record start address

    char str[10];
    while(scanf("%s", str) != EOF) {
        if(!strcmp(str, "m") || !strcmp(str, "malloc")) {
            getchar();
            int space1;
            scanf("%d", &space1);
            char *addr1 = lmalloc(space1, &coremap, &current_loc);
            if(addr1) printf("Memory was successfully allocated to address %lu.\n", (unsigned long)(addr1));
            else printf("The operation failed.\n");
            display(coremap, start_addr);
        } else if (!strcmp(str, "f") || !strcmp(str, "free")) {
            getchar();
            int space2;
            int addr2; // logical address
            scanf("%d %d", &space2, &addr2);
            if(lfree(space2, start_addr + addr2, &coremap))
                printf("Memory freed successfully.\n");
            else printf("Error!\n");
            display(coremap, start_addr);
        } else {
            fgets(str,100,stdin);  // set a limit in case of unlimited input
            printf("Error!\n");
        }

    }
    return 0;
}
```

```c
// NextFit.c

#include "NextFit.h"

// allocate memory
char *lmalloc(unsigned size, struct maplist *coremap, struct map* *current_loc) {
    char *a;
    struct map *bp = *current_loc;

    while(1) {
        if (bp->m_size >= size) {
            a = bp->m_addr;
            bp->m_addr += size;
            bp->m_size -= size;

            if (bp->m_size == 0) {
```

```
                // the partition is full
                if (coremap->len == 1) {
                    // only one item in the list
                    coremap->addr = NULL;
                    struct map *dp = bp;
                    free(dp);
                } else {
                    bp->prior->next = bp->next;
                    bp->next->prior = bp->prior;
                    struct map *dp = bp;
                    free(dp);
                }
                coremap->len--;
            }
            *current_loc = bp;
            return a;
        }
        bp = bp->next;
        if(bp == *current_loc)
            break;
    }

    return NULL;
}

// free memory
int lfree(unsigned size, char* addr, struct maplist *coremap) {
    if(size <= 0) return false;

    struct map *bp = coremap->addr;

    // find the next free part
    while (bp->m_addr <= addr) {
        bp = bp->next;
        if(bp == coremap->addr)
            break;
    }

    if (bp->prior->m_addr + bp->prior->m_size == addr && addr + size != bp->m_addr) {
        // case 1
        // printf("1\n");
        bp->prior->m_size += size;
        return true;
    } else if (bp->prior->m_addr + bp->prior->m_size == addr && addr + size == bp->m_addr) {
        // case 2
        // printf("2\n");
        bp->prior->m_size += size + bp->m_size;
        bp->prior->next = bp->next;
        bp->next->prior = bp->prior;
        if(coremap->addr == bp) coremap->addr =  bp->prior;
        coremap->len--;
        struct map *dp = bp;
        free(dp);
        return true;
    } else if (bp->prior->m_addr + bp->prior->m_size != addr && addr + size == bp->m_addr) {
        // case 3
        // printf("3\n");
        bp->m_size += size;
        bp->m_addr -= size;
        return true;
    } else if (bp->prior->m_addr + bp->prior->m_size != addr && addr + size != bp->m_addr) {
        // case 4
        // printf("4\n");
```

```
        struct map *p = (struct map*)malloc(sizeof(struct map));
        p->m_size = size;
        p->m_addr = addr;
        p->next = bp;
        p->prior = bp->prior;
        bp->prior->next = p;
        bp->prior = p;
        coremap->len++;
        if(addr < coremap->addr->m_addr) coremap->addr = p;
        return true;
    }
    return false;
}

// display the result
void display(struct maplist m, char *start_addr) {
    printf("--------------------------------------------------------------------------------\n");
    printf("\t\tCurrent Status(Start Address: %lu)\n", (unsigned long)start_addr);
    printf("--------------------------------------------------------------------------------\n");
    struct map *bp = m.addr;
    for (int i = 0;; i++) {
            printf("    Item %d: [Physical Addr: %lu; Logical Addr: %lu; Size: %u]\n",
               i, (unsigned long)bp->m_addr, (unsigned long)(bp->m_addr - start_addr), bp->m_size);
        bp = bp->next;
        if(bp == m.addr)
            break;
    }
    printf("--------------------------------------------------------------------------------\n\n");
}
```

```
// NextFit.h

#ifndef _HEAD_H
#define _HEAD_H

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <string.h>

#define true 1
#define false 0

// item to record the info of one free memory partition
struct map {
    unsigned m_size;
    char *m_addr;
    struct map *next, *prior;
};

// free memory partition list
struct maplist {
    struct map *addr;
    int len;
};

// allocate memory
char *lmalloc(unsigned size, struct maplist *coremap, struct map* *current_loc);

// free memory
```

```
int lfree(unsigned size, char *addr, struct maplist *coremap);

// display the result
void display(struct maplist m, char *start_addr);

#endif
```

```
# Makefile

cc = gcc
prom = NextFit
deps = $(shell find ./ -name "*.h")
src = $(shell find ./ -name "*.c")
obj = $(src:%.c=%.o)

$(prom): $(obj)
  $(cc) -o $(prom) $(obj)

%.o: %.c $(deps)
  $(cc) -c $< -o $@

clean:
  rm -rf $(obj) $(prom)
```

```
#!/bin/bash

# test.sh

for i in 1 2 3 4 5
do
    ./NextFit < "traces/trace$i" > "out/out$i"
done
```