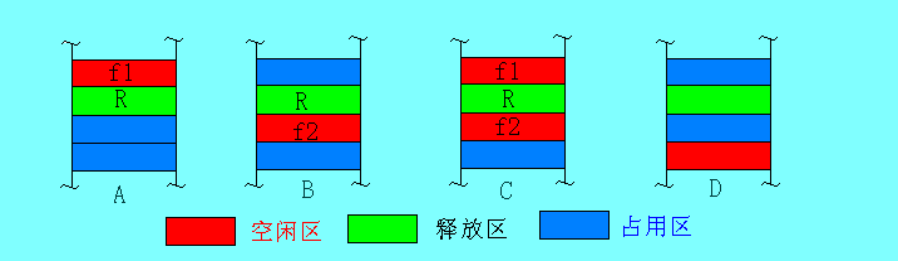
第四章　实验三

模拟内存回收算法

- 内存回收四种情况：



实例：某系统内存容量为 800K，下面分别给出中的空闲分区表和分配分区表，系统采用动态分区存储管理策略。现有按照如下顺序释放作业空间： (1)释放作业：Task2 **【情况 D】**; (2)释放作业：Task4 **【情况 A】**; (3)释放作业：Task6 **【情况 B】**; (4)释放作业：Task7 **【情况 C】**; (5)释放作业：Task9 **【不存在】**并打印每次回收分配空间后的空闲分区表和分配分区表。

分配分区表：

作业名	大小	起始地址
OS	60K	0 K
Task1	40K	60K
Task2	32K	100K
Task3	18K	132K
Task4	40K	160K
Task5	5K	200K
Task6	15K	205K
Task7	92K	438K
Task8	174K	626K

空闲分区表：

大小	起始地址
10K	150K
218K	220K
96K	530K

- 动态地任意顺序输入“分配分区表项”，按照**地址递增方式**建立“分配分区表”

  输入：

```
The distributed table is:
address length flag(0 or 1)
0 60 1
        input job_name:OS
60 40 1
        input job_name:Task1
100 32 1
        input job_name:Task2
132 18 1
        input job_name:Task3
160 40 1
        input job_name:Task4
200 5 1
        input job_name:Task5
205 15 1
        input job_name:Task6
438 92 1
        input job_name:Task7
626 174 1
        input job_name:Task8
0 0 0
```

- 动态地任意顺序输入“空闲分区表项”，按**照地址递增方式**建立“空闲分区表”

  输入：

```
address length flag(0 or 1)
150 10 0
220 218 0
530 96 0
0 0 0
```

- 共内存回收 5 次：

  输入：

```
Input the released work segment sum:5
```

- 回收内存分配区，模拟第一次内存回收：Task2 **【情况 D】**。

  输入：

```
1: input the released work segment name:Task2
```

  分配成功，输出：

```
要回收的分区存在!

The type of release is D!

distribute table is !
0,60,1,OS
60,40,1,Task1
132,18,1,Task3
160,40,1,Task4
200,5,1,Task5
205,15,1,Task6
438,92,1,Task7
626,174,1,Task8

free table is !
100,32,0,nil
150,10,0,nil
220,218,0,nil
530,96,0,nil
```

- 回收内存分配区，模拟第二次内存回收：Task4 【情况 A】。

  输入：

```
2: input the released work segment name:Task4
```

  分配成功，输出：

```
要回收的分区存在!

The type of release is A!

distribute table is !
0,60,1,OS
60,40,1,Task1
132,18,1,Task3
200,5,1,Task5
205,15,1,Task6
438,92,1,Task7
626,174,1,Task8

free table is !
100,32,0,nil
150,50,0,nil
220,218,0,nil
530,96,0,nil
```

- 回收内存分配区，模拟第三次内存回收：Task6 【情况 B】。

  输入：

```
3: input the released work segment name:Task6
```

  分配失败，输出：

```
要回收的分区存在!

The type of release is B!

distribute table is !
0,60,1,OS
60,40,1,Task1
132,18,1,Task3
200,5,1,Task5
438,92,1,Task7
626,174,1,Task8

free table is !
100,32,0,nil
150,50,0,nil
205,233,0,nil
530,96,0,nil
```

- 回收内存分配区，模拟第四次内存回收：Task7 【情况 C】。

  输入：

```
4: input the released work segment name:Task7
```

  分配成功，输出：

```
要回收的分区存在!

The type of release is C!

distribute table is !
0,60,1,OS
60,40,1,Task1
132,18,1,Task3
200,5,1,Task5
626,174,1,Task8

free table is !
100,32,0,nil
150,50,0,nil
205,421,0,nil
```

- 回收内存分配区，模拟第五次内存回收：Task9 【不存在】。

  输入：

  ```
  5: input the released work segment name:Task9
  ```

  分配成功，输出：

  ```
  要回收的分区不存在!
  ```

- 使用 Micrsoft Visual Studio C++ 6.0 或 CodeBlocks 编程：程序 4_3_release.cpp。完善如下程序代码：

```
#include <malloc.h>
#include <stdio.h>
#include <string.h>
#define NULL 0
typedef struct table
   {int address;            /*存储分区起始地址*/
    int length;             /*存储分区长度*/
    int flag;               /*存储分区标志，0 为空闲，1 为被作业占据*/
    char name[10];          /*当 flag==1 时存储分区占用标志作业名，否则存储空 nil*/
    struct table *next;
 }node;
node *work;       /*设置一个全局变量 work：定位需要释放的结点*/
char type;        /*设置一个全局变量 type：标注回收类型*/
bool success;     /*设置一个全局变量 success：标注回收结点是否在分配分区表中*/
node *insert(node *head, node *p)   /*按照“地址递增方式”将 p 结点插入链表相应位置*/
  { 填补程序 }
node *creat()     /*根据地址递增方式建立分配分区表(flag==1)或空闲分区表(flag==0)*/
  { 填补程序 }
node *found(node *distributedhead,char workn[10])   /*查找已分配表中要回收的分区位置*/
  { 填补程序 }
node *release(node *freehead,node *work)     /*分四种情况完成空闲分区回收过程*/
  { 填补程序 }
void print (node *head)   /*输出链表*/
  { 填补程序 }
void main()
 {   int i,sum;
     struct table *dtable,*ftable;
     char workn[10];      printf("The distributed table is:\n");
     dtable=creat();                    /*dtable 输入已分配情况表*/
     printf("The free table is:\n");
     ftable=creat();                    /*ftable 输入未分配情况表*/
      /*以下模拟逐个内存回收过程*/
      {   填补程序 }
 }
```