一、给出程序流程和源程序

```
//*************************
//姓名：田松岩
//班级：2015级4班
//学号：2015015335
//任务：动态分区式存储管理
//*************************/
#include<stdio.h>
#include<stdlib.h>
#define SIZE 100            // 内存初始大小
#define MINSIZE 5           // 碎片最小值
enum STATE { Free, Busy };

struct subAreaNode {
    int addr;               // 起始地址
    int size;               // 分区大小
    int taskId;             // 作业号
    STATE state;            // 分区状态
    subAreaNode *pre;       // 分区前向指针
    subAreaNode *nxt;       // 分区后向指针
}subHead;

// 初始化空闲分区链
void intSubArea()
{
    // 分配初始分区内存
    subAreaNode *fir = (subAreaNode *)malloc(sizeof(subAreaNode));
    // 给首个分区赋值
    fir->addr = 0;
    fir->size = SIZE;
    fir->state = Free;
    fir->taskId = -1;
    fir->pre = &subHead;
    fir->nxt = NULL;
    // 初始化分区头部信息
    subHead.pre = NULL;
    subHead.nxt = fir;
}
```

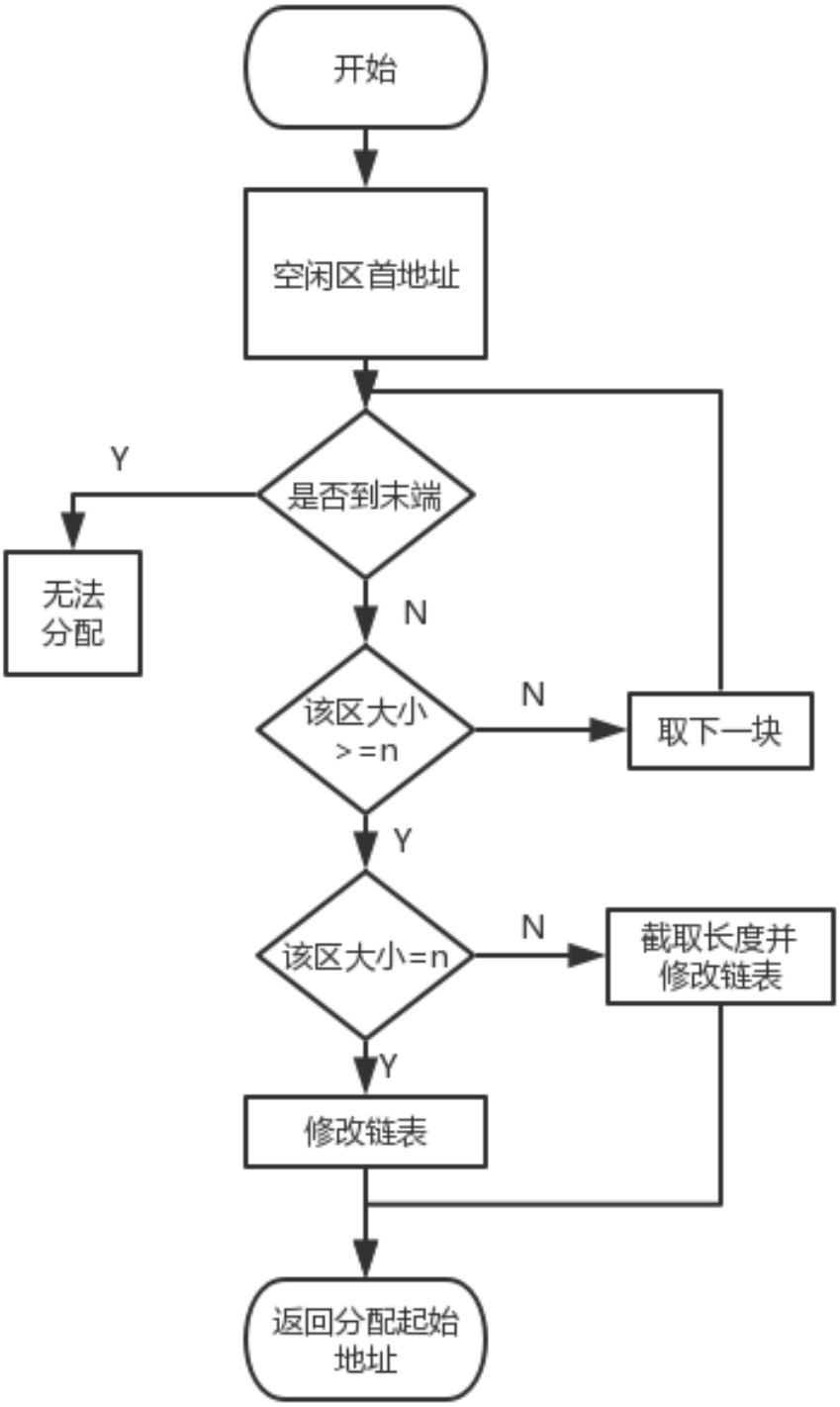
```
// 最佳适应算法
int bestFit(int taskId, int size)
{
    subAreaNode *tar = NULL;
    int tarSize = SIZE + 1;
    subAreaNode *p = subHead.nxt;
    while (p != NULL)
    {
        // 寻找最佳空闲区间
        if (p->state == Free && p->size >= size && p->size < tarSize) {
            tar = p;
            tarSize = p->size;
        }
        p = p->nxt;
    }
    if (tar != NULL) {
        // 找到要分配的空闲分区
        if (tar->size - size <= MINSIZE) {
            // 整块分配
            tar->state = Busy;
            tar->taskId = taskId;
        }
        else {
            // 分配大小为size的区间
            subAreaNode *node = (subAreaNode *)malloc(sizeof(subAreaNode));
            node->addr = tar->addr + size;
            node->size = tar->size - size;
            node->state = Free;
            node->taskId = -1;
            // 修改分区链节点指针
            node->pre = tar;
            node->nxt = tar->nxt;
            if (tar->nxt != NULL) {
                tar->nxt->pre = node;
            }
            tar->nxt = node;
            // 分配空闲区间
            tar->size = size;
            tar->state = Busy;
            tar->taskId = taskId;
        }
        printf("内存分配成功！\n");
        return 1;
    }
```

```
    else {
        // 找不到合适的空闲分区
        printf("找不到合适的内存分区，分配失败...\n");
        return 0;
    }
}

// 回收内存
int freeSubArea(int taskId)
{
    int flag = 0;
    subAreaNode *p = subHead.nxt, *pp;
    while (p != NULL)
    {
        if (p->state == Busy && p->taskId == taskId) {
            flag = 1;
            if ((p->pre != &subHead && p->pre->state == Free)
                && (p->nxt != NULL && p->nxt->state == Free)) {
                // 情况1：合并上下两个分区
                // 先合并上区间
                pp = p;
                p = p->pre;
                p->size += pp->size;
                p->nxt = pp->nxt;
                pp->nxt->pre = p;
                free(pp);
                // 后合并下区间
                pp = p->nxt;
                p->size += pp->size;
                p->nxt = pp->nxt;
                if (pp->nxt != NULL) {
                    pp->nxt->pre = p;
                }
                free(pp);
            }
            else if ((p->pre == &subHead || p->pre->state == Busy)
                && (p->nxt != NULL && p->nxt->state == Free)) {
                // 情况2：只合并下面的分区
                pp = p->nxt;
                p->size += pp->size;
                p->state = Free;
                p->taskId = -1;
                p->nxt = pp->nxt;
                if (pp->nxt != NULL) {
```

```
                    pp->nxt->pre = p;
                }
                free(pp);
            }
            else if ((p->pre != &subHead && p->pre->state == Free)
                && (p->nxt == NULL || p->nxt->state == Busy)) {
                // 情况3：只合并上面的分区
                pp = p;
                p = p->pre;
                p->size += pp->size;
                p->nxt = pp->nxt;
                if (pp->nxt != NULL) {
                    pp->nxt->pre = p;
                }
                free(pp);
            }
            else {
                // 情况4：上下分区均不用合并
                p->state = Free;
                p->taskId = -1;
            }
        }
        p = p->nxt;
    }
    if (flag == 1) {
        // 回收成功
        printf("内存分区回收成功...\n");
        return 1;
    }
    else {
        // 找不到目标作业，回收失败
        printf("找不到目标作业，内存分区回收失败...\n");
        return 0;
    }
}

// 显示空闲分区链情况
void showSubArea()
{
    printf("=============================================\n");
    printf("==          当前的内存分配情况如下：          ==\n");
    printf("=============================================\n");
    printf("== 起始地址 | 空间大小 | 工作状态 | 作业号 ==\n");
    subAreaNode *p = subHead.nxt;
```

```
    while (p != NULL)
    {
        printf("**-------------------------------------**\n");
        printf("**");
        printf("%d  k    |", p->addr);
        printf("%d  k    |", p->size);
        printf("   %s   |", p->state == Free ? "Free" : "Busy");
        if (p->taskId > 0) {
            printf("%d   ", p->taskId);
        }
        else {
            printf("        ");
        }
        printf("**\n");
        p = p->nxt;
    }
    printf("=============================================\n");
}

//主调函数进行测试
int main()
{
    int  ope, taskId, size;
    // 初始化空闲分区链
    intSubArea();

    // 模拟动态分区分配算法
    while (1)
    {
        printf("\n");
        printf("=============================================\n");
        printf("==  1: 分配内存    2: 回收内存    0: 退出  ==\n");
        printf("=============================================\n");
        scanf("%d", &ope);
        if (ope == 0) break;
        if (ope == 1) {
            // 模拟分配内存
            printf("请输入作业号： ");
            scanf("%d", &taskId);
            printf("请输入需要分配的内存大小(KB)： ");
            scanf("%d", &size);
            if (size <= 0) {
                printf("错误：分配内存大小必须为正值\n");
                continue;
```

```
            }
            // 调用分配算法

                bestFit(taskId, size);

            // 显示空闲分区链情况
            showSubArea();
        }
        else if (ope == 2) {
            // 模拟回收内存
            printf("请输入要回收的作业号：  ");
            scanf("%d", &taskId);
            freeSubArea(taskId);
            // 显示空闲分区链情况
            showSubArea();
        }
        else {
            printf("错误：请输入 0/1/2\n");
        }
    }
    printf("分配算法模拟结束\n");
    return 0;
}
```


开始
空闲区首地址
是否到末端
Y
无法
分配
N
该区大小
>=n
N
取下一块
Y
该区大小=n
N
截取长度并
修改链表
Y
修改链表
返回分配起始
地址

二、程序运行截图

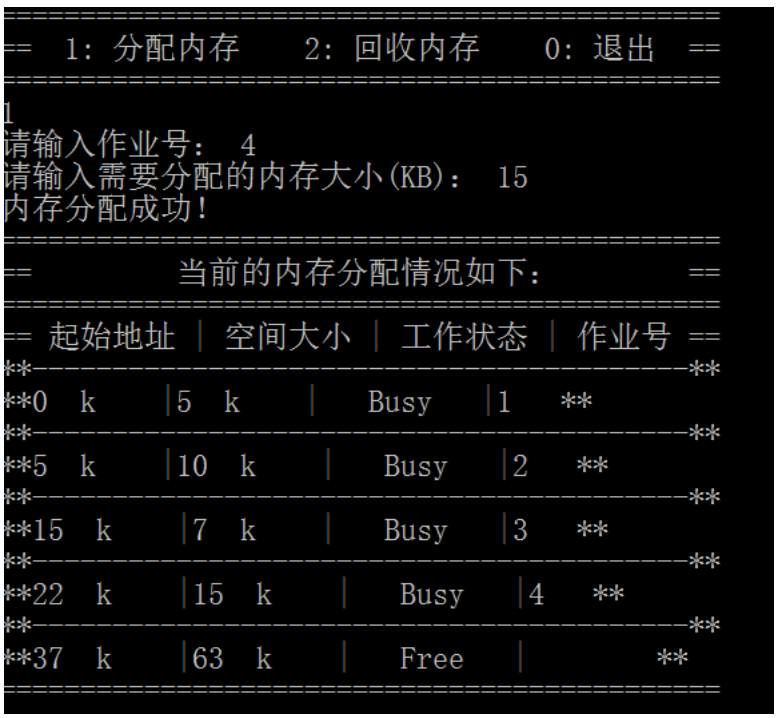
C:\WINDOWS\system32\cmd.exe

```
==========================================
==  1: 分配内存     2: 回收内存     0: 退出  ==
==========================================
1
请输入作业号:  1
请输入需要分配的内存大小(KB):  5
内存分配成功!
==========================================
==          当前的内存分配情况如下:          ==
==========================================
== 起始地址 | 空间大小 | 工作状态 | 作业号 ==
**-----------------------------------------**
**0  k     |5  k     |   Busy   |1   **
**-----------------------------------------**
**5  k     |95  k    |   Free   |        **
==========================================
```

分配内存

```
==========================================


==========================================
==  1: 分配内存     2: 回收内存     0: 退出  ==
==========================================
1
请输入作业号:  2
请输入需要分配的内存大小(KB):  10
内存分配成功!
==========================================
==          当前的内存分配情况如下:          ==
==========================================
== 起始地址 | 空间大小 | 工作状态 | 作业号 ==
**-----------------------------------------**
**0  k     |5  k     |   Busy   |1   **
**-----------------------------------------**
**5  k     |10  k    |   Busy   |2   **
**-----------------------------------------**
**15  k    |85  k    |   Free   |        **
==========================================
```


== 1: 分配内存   2: 回收内存   0: 退出 ==
1
请输入作业号： 3
请输入需要分配的内存大小(KB)： 7
内存分配成功!
==   当前的内存分配情况如下: ==
== 起始地址 | 空间大小 | 工作状态 | 作业号 ==
**0 k |5 k | Busy |1 **
**5 k |10 k | Busy |2 **
**15 k |7 k | Busy |3 **
**22 k |78 k | Free | **


== 1: 分配内存   2: 回收内存   0: 退出 ==
1
请输入作业号： 4
请输入需要分配的内存大小(KB)： 15
内存分配成功!
==   当前的内存分配情况如下: ==
== 起始地址 | 空间大小 | 工作状态 | 作业号 ==
**0 k |5 k | Busy |1 **
**5 k |10 k | Busy |2 **
**15 k |7 k | Busy |3 **
**22 k |15 k | Busy |4 **
**37 k |63 k | Free | **

C:\WINDOWS\system32\cmd.exe

```
==============================================


==============================================
==  1: 分配内存     2: 回收内存     0: 退出  ==
==============================================
1
请输入作业号:  5
请输入需要分配的内存大小(KB):  20
内存分配成功!
==============================================
==          当前的内存分配情况如下:          ==
==============================================
== 起始地址 | 空间大小 | 工作状态 | 作业号 ==
**--------------------------------------------**
**0  k     |5  k      |   Busy   |1   **
**--------------------------------------------**
**5  k     |10  k     |   Busy   |2   **
**--------------------------------------------**
**15  k    |7  k      |   Busy   |3   **
**--------------------------------------------**
**22  k    |15  k     |   Busy   |4   **
**--------------------------------------------**
**37  k    |20  k     |   Busy   |5   **
**--------------------------------------------**
**57  k    |43  k     |   Free   |       **
==============================================
```

C:\WINDOWS\system32\cmd.exe

```
==============================================
==  1: 分配内存     2: 回收内存     0: 退出  ==
==============================================
1
请输入作业号:  6
请输入需要分配的内存大小(KB):  25
内存分配成功!
==============================================
==          当前的内存分配情况如下:          ==
==============================================
== 起始地址 | 空间大小 | 工作状态 | 作业号 ==
**--------------------------------------------**
**0  k     |5  k      |   Busy   |1   **
**--------------------------------------------**
**5  k     |10  k     |   Busy   |2   **
**--------------------------------------------**
**15  k    |7  k      |   Busy   |3   **
**--------------------------------------------**
**22  k    |15  k     |   Busy   |4   **
**--------------------------------------------**
**37  k    |20  k     |   Busy   |5   **
**--------------------------------------------**
**57  k    |25  k     |   Busy   |6   **
**--------------------------------------------**
**82  k    |18  k     |   Free   |       **
==============================================
```

回收内存，上下不合并

```
==========================================
==  1: 分配内存     2: 回收内存     0: 退出  ==
==========================================
2
请输入要回收的作业号：  4
内存分区回收成功...
==========================================
==          当前的内存分配情况如下：          ==
==========================================
== 起始地址 | 空间大小 | 工作状态 | 作业号 ==
**------------------------------------------**
**0  k     |5  k      |   Busy   |1   **
**------------------------------------------**
**5  k     |10  k     |   Busy   |2   **
**------------------------------------------**
**15  k    |7  k      |   Busy   |3   **
**------------------------------------------**
**22  k    |15  k     |   Free   |        **
**------------------------------------------**
**37  k    |20  k     |   Busy   |5   **
**------------------------------------------**
**57  k    |25  k     |   Busy   |6   **
**------------------------------------------**
**82  k    |18  k     |   Free   |        **
==========================================
```

收回内存

回收 5，与上面的 15K 合并

```
==========================================
==  1: 分配内存     2: 回收内存     0: 退出  ==
==========================================
2
请输入要回收的作业号：  5
内存分区回收成功...
==========================================
==          当前的内存分配情况如下：          ==
==========================================
== 起始地址 | 空间大小 | 工作状态 | 作业号 ==
**------------------------------------------**
**0  k     |5  k      |   Busy   |1   **
**------------------------------------------**
**5  k     |10  k     |   Busy   |2   **
**------------------------------------------**
**15  k    |7  k      |   Busy   |3   **
**------------------------------------------**
**22  k    |35  k     |   Free   |        **
**------------------------------------------**
**57  k    |25  k     |   Busy   |6   **
**------------------------------------------**
**82  k    |18  k     |   Free   |        **
==========================================
```

```
==============================================
==  1: 分配内存     2: 回收内存     0: 退出  ==
==============================================
2
请输入要回收的作业号：  1
内存分区回收成功...
==============================================
==          当前的内存分配情况如下：          ==
==============================================
== 起始地址 | 空间大小 | 工作状态 | 作业号 ==
**-----------------------------------------**
**0  k     |5  k     |   Free   |        **
**-----------------------------------------**
**5  k     |10  k    |   Busy   |2   **
**-----------------------------------------**
**15  k    |7  k     |   Busy   |3   **
**-----------------------------------------**
**22  k    |35  k    |   Free   |        **
**-----------------------------------------**
**57  k    |25  k    |   Busy   |6   **
**-----------------------------------------**
**82  k    |18  k    |   Free   |        **
==============================================
```

```
==  1: 分配内存     2: 回收内存     0: 退出  ==
==============================================
1
请输入作业号：  8
请输入需要分配的内存大小(KB)：  15
内存分配成功!
==============================================
==          当前的内存分配情况如下：          ==
==============================================
== 起始地址 | 空间大小 | 工作状态 | 作业号 ==
**-----------------------------------------**
**0  k     |5  k     |   Free   |        **
**-----------------------------------------**
**5  k     |10  k    |   Busy   |2   **
**-----------------------------------------**
**15  k    |7  k     |   Busy   |3   **
**-----------------------------------------**
**22  k    |35  k    |   Free   |        **
**-----------------------------------------**
**57  k    |25  k    |   Busy   |6   **
**-----------------------------------------**
**82  k    |18  k    |   Busy   |8   **
==============================================

==============================================
==  1: 分配内存     2: 回收内存     0: 退出  ==
==============================================
```

最佳适应算法，分配15K，程序选择18k的，没选择35k 的

三、收获、体会及对该实验的改进意见和见解

收获：

我深刻的理解了动态分区存储管理的实现原理，动态分区分配是根据进程的实际需要，动态地为之分配内存空间。作业装入内存时，把可用内存分出一个连续区域给作业，且分区的大小正好适合作业大小的需要。分区的大小和个数依装入作业的需要而定。

掌握动态分区式存储管理方式的内存分配和回收的实现。学会了运用最佳适应算法，分配内存。

改进意见：应该使用多种分配算法实现内存的分配。