第四章　实验二

模拟最佳适应（Best Fit）算法

- **最佳适应算法 BF(Best Fit)**
- **基本思想：**BF 的空闲分区表（或空闲分区链）**按空闲区大小的升序方式组织**。分配时，按空闲分区表（或空闲分区链）的先后次序，从头查找，找到符合要求的第一个分区。就说明它是**最适合的（即最佳的）**。大的空闲区可以被保留下来。

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。现有以下作业序列：96K（作业名：A），20K（作业名：B），200K（作业名：C）。请用最佳适应 BF 算法来处理这些作业序列，并打印出该算法分配空间后的空闲分区表和分配分区表。(若出现内存不足的情况，请提示作业等待内存资源)

分配分区表：

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

空闲分区表：

大小	起始地址
32K	100K
10K	150K
5K	200K
218K	220K
96K	530K

按照 BF 算法分配内存：

- 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
132 40 1
        input job_name:Task2
626 174 1
        input job_name:Task6
438 92 1
        input job_name:Task5
205 15 1
        input job_name:Task4
160 40 1
        input job_name:Task3
0 0 0
```

- 动态地任意顺序输入“空闲分区表项”，按照**按空闲区大小的升序**方式建立“空闲分区表”

  输入：

```
The free table is:
address length flag(0 or 1)
100 32 0
150 10 0
200 5 0
530 96 0
220 218 0
0 0 0
```

- 按照 BF 算法完成内存分配，模拟第一次内存申请：96K（作业名：A）。

  输入：

```
The length of worked job is:96
The name of worked job is:A
```

  分配成功，输出：

```
distributing is successful!

The free table is !
200,5,0,nil
150,10,0,nil
100,32,0,nil
220,218,0,nil

The distributed table is !
0,60,1,OS
60,40,1,Task1
132,40,1,Task2
160,40,1,Task3
205,15,1,Task4
438,92,1,Task5
530,96,1,A
626,174,1,Task6
```

- 按照 BF 算法完成内存分配，模拟第二次内存申请：20K（作业名：B）。

  输入：

```
The length of worked job is:20
The name of worked job is:B
```

  分配成功，输出：

```
distributing is successful!

The free table is !
200,5,0,nil
150,10,0,nil
120,12,0,nil
220,218,0,nil

The distributed table is !
0,60,1,OS
60,40,1,Task1
100,20,1,B
132,40,1,Task2
160,40,1,Task3
205,15,1,Task4
438,92,1,Task5
530,96,1,A
626,174,1,Task6
```

- 按照 BF 算法完成内存分配，模拟第三次内存申请：200K（作业名：C）。

  输入：

```
The length of worked job is:200
The name of worked job is:C
```

  分配成功，输出：

```
distributing is successful!

The free table is !
200,5,0,nil
150,10,0,nil
120,12,0,nil
420,18,0,nil

The distributed table is !
0,60,1,OS
60,40,1,Task1
100,20,1,B
132,40,1,Task2
160,40,1,Task3
205,15,1,Task4
220,200,1,C
438,92,1,Task5
530,96,1,A
626,174,1,Task6
```

- 使用 Micrsoft Visual Studio C++ 6.0 或 CodeBlocks 编程：程序 4_2_bestfit.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;
bool success;       /*分配成功与否的标志*/

node *insertaddress(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 *insertlength(node *head, node *p)
                  /*按照“空闲区大小递增方式”将 p 结点插入链表相应位置*/
  { 填补程序 }

node *creat()     /*建立分配分区表(flag==1)或空闲分区表(flag==0)*/
  { 填补程序 }
node *distribute(node *freehead, node *distributedhead, node *work)
/*在空闲分区表中找出首次合适 work 的分区，同时修改空闲分区表和分配分区表*/
  { 填补程序 }
void print (node *head)   /*输出链表*/
  { 填补程序 }
void main()
 {   int a;
    struct table *dtable,*ftable,*work;
    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*/
     /*以下模拟逐个内存申请过程*/
     {   填补程序 }
 }
```