阅读：62,328　　作者：解学武

顺序表（顺序存储结构）及初始化过程详解

上一节　　下一节

顺序表，全名顺序存储结构，是线性表的一种。通过《线性表》一节的学习我们知道，线性表用于存储逻辑关系为“一对一”的数据，顺序表自然也不例外。

不仅如此，顺序表对数据的物理存储结构也有要求。顺序表存储数据时，会提前申请一整块足够大小的物理空间，然后将数据依次存储起来，存储时做到数据元素之间不留一丝缝隙。

例如，使用顺序表存储集合 {1,2,3,4,5} ，数据最终的存储状态如图 1 所示：

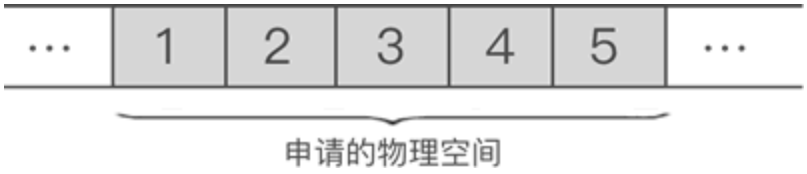

图 1 顺序存储结构示意图

由此我们可以得出，将“具有 '一对一' 逻辑关系的数据按照次序连续存储到一整块物理空间上”的存储结构就是顺序存储结构。

通过观察图 1 中数据的存储状态，我们可以发现，顺序表存储数据同数组非常接近。其实，顺序表存储数据使用的就是数组。

顺序表的初始化

使用顺序表存储数据之前，除了要申请足够大小的物理空间之外，为了方便后期使用表中的数据，顺序表还需要实时记录以下 2 项数据：

1. 顺序表申请的存储容量；
2. 顺序表的长度，也就是表中存储数据元素的个数；

提示：正常状态下，顺序表申请的存储容量要大于顺序表的长度。

因此，我们需要自定义顺序表，C 语言实现代码如下：

```
01.  typedef struct Table{
02.      int * head;//声明了一个名为head的长度不确定的数组，也叫“动态数组”
03.      int length;//记录当前顺序表的长度
04.      int size;//记录顺序表分配的存储容量
```

```
}table;
```

注意，head 是我们声明的一个未初始化的动态数组，不要只把它看做是普通的指针。

接下来开始学习顺序表的初始化，也就是初步建立一个顺序表。建立顺序表需要做如下工作：

- 给 head 动态数据申请足够大小的物理空间；
- 给 size 和 length 赋初值；

因此，C 语言实现代码如下：

```
#define Size 5 //对Size进行宏定义，表示顺序表申请空间的大小
table initTable(){
    table t;
    t.head = (int*)malloc(Size * sizeof(int));//构造一个空的顺序表，动态申请存储空间
    if (!t.head) //如果申请失败，作出提示并直接退出程序
    {
        printf("初始化失败");
        exit(0);
    }
    t.length = 0;//空表的长度初始化为0
    t.size = Size;//空表的初始存储空间为Size
    return t;
}
```

我们看到，整个顺序表初始化的过程被封装到了一个函数中，此函数返回值是一个已经初始化完成的顺序表。这样做的好处是增加了代码的可用性，也更加美观。与此同时，顺序表初始化过程中，要注意对物理空间的申请进行判断，对申请失败的情况进行处理，这里只进行了“输出提示信息和强制退出”的操作，可以根据你自己的需要对代码中的 if 语句进行改进。

通过在主函数中调用 initTable 语句，就可以成功创建一个空的顺序表，与此同时我们还可以试着向顺序表中添加一些元素，C 语言实现代码如下：

```
#include <stdio.h>
#include <stdlib.h>  //malloc()、exit()
#define Size 5
typedef struct Table {
    int * head;
    int length;
    int size;
}table;
table initTable() {
    table t;
    t.head = (int*)malloc(Size * sizeof(int));
    if (!t.head)
    {
        printf("初始化失败");
```

```
15.          exit(0);
16.      }
17.      t.length = 0;
18.      t.size = Size;
19.      return t;
20.  }
21.  //输出顺序表中元素的函数
22.  void displayTable(table t) {
23.      int i;
24.      for (i = 0; i < t.length; i++) {
25.          printf("%d ", t.head[i]);
26.      }
27.      printf("\n");
28.  }
29.  int main() {
30.      int i;
31.      table t = initTable();
32.      //向顺序表中添加元素
33.      for (i = 1; i <= Size; i++) {
34.          t.head[i - 1] = i;
35.          t.length++;
36.      }
37.      printf("顺序表中存储的元素分别是: \n");
38.      displayTable(t);
39.      return 0;
40.  }
```

程序运行结果如下：

```
顺序表中存储的元素分别是:
1 2 3 4 5
```

可以看到，顺序表初始化成功。

上一节 下一节