maker:cy

# 顺序表实现 /C/C++语言



->自己总结!

->第一小节：关于C语言实现线性结构之一顺序表

## 一、关于C语言的内存是如何分配的

前期准备：搞懂动态内存分配的原理，内存是怎么通过动态分配内存空间的？数据又是怎样进行存储的？在内存里面！怎么通过动态来分配内存空间？

关于内存：之前我们学习如何去定义变量、定义一个确定值大小的数组

1、变量：开辟一个空间 `int a = 10;`

2、数组：开辟一个连续的空间 `int arr[10] = {0};`

思考？怎么能开辟一段连续的空间，又能对它进行一系列操作呢？例如：增删改查

单纯的数组肯定是不行的，因为初始化的时候就给他了一个确定的定长，要想改变必须通过人来确定或者修改，而且容量不可扩，也不可删，可不可插..等一些列的缺陷。

3.引用可变长数组，但是一般的语言都不会支持，C99可实现

以上方法都不可行，就引用了动态内存函数!

## 二、三个动态内存函数

### malloc：

> 用法 malloc(开辟个数*所占字节) malloc(10 * sizeof(int)) 开辟整形的空间需要转换成整形指针
>
> 1.开辟成功，则返回一个指向开辟好空间的指针
>
> 2.如果开辟失败，则返回一个NULL指针，因此Malloc的返回值一定要做检查
>
> 3.返回值的类型是void* 所以malloc函数并不知道开辟空间的类型，具体在使用的时候使用者自己来决定
>
> 4.如果参数size为0，malloc的行为是标准未定义的，取决与编译器
>
> 5.与函数free连用，专门用来动态内存的释放和回收的。
>
> 6.头文件引用：stdlib.h

代码演示：

```c
int main()
{
    //向内存申请10个整形的空间
    int* p =(ElemEype*)malloc(50 * sizeof(ElemEype)); //指向的是50个整形空间
    if (p==NULL)
    {
        printf("%s\n", strerror(errno));//打印错误
    }
    else
    {
        int i = 0;
        for (i = 0;i<50;i++)
        {
            *(p + i) = i;//简引用
        }
        for (i = 0; i < 50; i++)
        {
            printf("%d ", *(p + i));
        }
        printf("\n");
    }
    //free(p);//把空间释放 就不要占用了    动态内存的释放和回收 操作系统收回
    //p = NULL;
}
```

### calloc：

calloc(个数，字节大小)

1.为num个大小为size的元素开辟一块空间，并且把空间的每个字节都初始化为0

2.与malloc区别在于calloc会在返回地址之前把申请的空间的每个字节初始化为0

代码演示：

```
int* m = (int*)calloc(50, sizeof(int));
   if (m == NULL)
   {
       printf("%s\n", strerror(errno));//打印错误
   }
   else
   {
       int i = 0;
       for (i = 0; i < 50; i++)
       {
           *(m + i) = i;
       }
       for (i = 0; i < 50; i++)
       {
           printf("%d ", *(m + i));
       }
   }
   free(m);//把空间释放 就不要占用了    动态内存的释放和回收 操作系统收回
   m= NULL;
```

realloc:

特性：  更具灵活性  功能：调整动态内存开辟空间的大小

```
void* realloc(void *p,size);
//上面空间已经不满足了！  用realloc来调整动态内存的空间
int p2 = (int*)realloc(p, 100 * sizeof(int));
```

1.如果p指向空间 之后有足够的内存空间可以追加，则可以直接返回 后返回p

2.如果没有足够的内存空间，则realloc函数会找一块新的内存区域开辟满足需求的空间，并且把原来内存中的数据拷贝回来，释放旧的内存空间，最后返回新开辟的内存空间地址

3.原有空间没有足够大的空间 属于异地扩 再其他的地方进行扩充 再拷贝数据到新空间 再释放新空间 对内存要求比较高4.原有空间有足够大的空间 原地扩

> 5.也能对calloc和malloc的一个空间的追加
> 6.重新开辟：realloc(NULL,40);

代码演示：

```
    //上面空间已经不满足了！  用realloc来调整动态内存的空间
    int* ptr = (int*)realloc(p, 100 * sizeof(int));
    if (ptr == NULL)
    {
        printf("%s\n", strerror(errno));//打印错误
    }
    else
    {
        p = ptr;
        for (int i = 50; i < 100; i++)
        {
            *(p + i) = i;
        }
        for (int i = 0; i < 100; i++)
        {
            printf("%d ", *(p + i));
        }

    }
    printf("\n");
    free(p);//把空间释放 就不要占用了    动态内存的释放和回收 操作系统收回
    p = NULL;
```

上述头文件：

```
#pragma once
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
typedef int ElemEype;
```

## 三、常见的动态内存错误

1.没有进行合理性判断 有可能分配失败

首先进行返回值的判断 对空指针进行简引用操作

```
void fun2()
{
    int* m = (int*)calloc(50, sizeof(int));
```

```
4
5           int i = 0;
6           for (i = 0; i < 50; i++)
7           {
8               *(m + i) = i;
9           }
10          for (i = 0; i < 50; i++)
11          {
12              printf("%d ", *(m + i));
13          }
14
15      free(m);//把空间释放 就不要占用了    动态内存的释放和回收 操作系统收回
16      m = NULL;
17  }
```

2.对动态开辟的内存越界访问

实际开辟内存小于分配的内存--属于越界访问

```
1   void fun3()
2   {
3       int* m = (int*)calloc(50, sizeof(int));
4       if (m == NULL)
5       {
6           printf("%s\n", strerror(errno));//打印错误
7       }
8       else
9       {
10          int i = 0;
11          for (i = 0; i < 50; i++)
12          {
13              *(m + i) = i;
14          }
15          for (i = 0; i < 150; i++)
16          {
17              printf("%d ", *(m + i));
18          }
19      }
20      free(m);//把空间释放 就不要占用了    动态内存的释放和回收 操作系统收回
21      m = NULL;
22  }
```

3.对非动态内存分配的释放 errno

```c
void fun()
{
    int a = 10;
    int* p = &a;
    free(p);//错误
    return;
}

```

4.使用free释放动态开辟内存的一部分　free只能从起始位置开始释放

```c
void fun4()
{
    int* m = (int*)calloc(50, sizeof(int));
    if (m == NULL)
    {
        printf("%s\n", strerror(errno));//打印错误
    }
    else
    {
        int i = 0;
        for (i = 0; i < 10; i++)
        {
            *m = i;
        }
        for (i = 0; i < 50; i++)
        {
            printf("%d ", *(m + i));
        }
    }
    free(m);//把空间释放 就不要占用了　动态内存的释放和回收 操作系统收回
    m = NULL;
}
```

5.对同一块动态内存多次释放

6.对动态开辟的内存忘记释放 导致内存泄露

## 四、顺序表的代码实现

用法1：malloc函数开辟空间

但是方法都是一样的

头文件

```c
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <malloc.h>
typedef int ElemType;

#define MaxSize 100 //空间的实际容量大小
```

成员构建

```c
typedef struct
{
    ElemType* a;//一个指针用来开辟动态内存空间
    int size;//空间中所含的数据个数
    //int MaxSize;//空间的实际容量大小
    //其他变量自己定 目前只用三个解决
}L;
```

开辟顺序表 初始化

简单方法开辟

```c
void InitList(L* ps)
{

    ps->a = (ElemType*)malloc(sizeof(ElemType) * MaxSize);
    IsEmptyList(ps);//判断一下是否开辟成功
    ps->size = 0;
}
```

判断一下是否开辟成功

```c
void IsEmptyList(L* ps)
{
    if (ps->a == NULL)
    {
        printf("%s\n", strerror(errno));
    }
}
```

向顺序表存入数据

```
void InsertList(L* ps,int x)
{

    ps->a[ps->size] = x;
    ps->size++;
}
```

打印数据

```
void DisList(L* ps)
{
    for (int i = 0;i<ps->size;i++)
    {
        printf("%d ", ps->a[i]);
    }
}
```

释放数据就是将这块开辟的空间销毁

防止占内存

```
void DestroyList(L* ps)
{
    free(ps->a);
    ps->a = NULL;
    ps->size = 0;
}
```

向顺序表某个位置插入数据

```

void AddElemList(L*ps,int x,int number)//某个位置插入
{
    //首先判断是不是空表
    IsEmptyList(ps);
    int temp = ps->a[x - 1];//先把这个x位置空出来
    for (int i = ps->size-1;i>=x-1;i--)
    {
        ps->a[i + 1] = ps->a[i];
    }
    ps->a[x-1] = number;
    ps->size++;
}
```

在顺序表某个位置删除数据

```
void DeleteElemList(L* ps, int x)//某个位置删除
{
    //首先判断是不是空表
    IsEmptyList(ps);
    int temp2 = ps->a[x - 1];
    for (int i =x; i<=ps->size; i++)
    {
        ps->a[i -1] = ps->a[i];
    }
    ps->size--;

}
```

给定一个值查看是否存在

查找

```
ElemType FindListElem(L* ps, int number)//查找相同元素 返回对应下标
{
    for (int i = 0;i<ps->size;i++)
    {
        if (ps->a[i] == number)
        {
            return i;
        }
    }
    return -1;
}
```

五、顺序表的缺陷

> 顺序表缺陷：
>
> 1.空间不够了需要扩容，增容是要付出代价
>
> 2.避免频繁扩容，空间满了基本上就是扩充两倍，可能就会导致一定的空间的浪费
>
> 3.顺序表要求数据从开始位置连续存储，那么就只能在头部或者中间位置插入删除数据，就需要进行挪动数据，效率低
>
> 针对顺序表的缺陷 就设计出了链表
>
> 连续的空间 ：我们就只记录第一位置的地址就可以了 就可以通过简引用来访问其他的位置地址

---

顺序表realloc实现：

1.接口函数：

通用

```c
#include "SeqList.h"
void SeqListInit(SL* ps)
{
    ps->a = NULL;
    ps->size = ps->capacity = 0;
}
/*
 * 条件:
 * 1.空间足够 直接插入
 * 2.空间不够 扩容
 */
void SeqListCheckCapatiy(SL* ps)
{

    //如果没有空间 或者 空间不足，就可以扩容
    if (ps->size == ps->capacity)
    {
        int newcapacity = ps->capacity == 0 ? 5 : ps->capacity * 2;
        SLdataType* tmp = (SLdataType*)realloc(ps->a, newcapacity *
sizeof(SLdataType));
        if (tmp == NULL)
        {
            printf("%s\n",strerror(errno));

        }
        ps->a = tmp;
        ps->capacity = newcapacity;
    }
}
void SeqListPushBack(SL* ps, SLdataType x) //后面插入数据
{
    SeqListCheckCapatiy(ps);
    ps->a[ps->size] = x;//将值赋值给数组指针里
    ps->size++;
}
void disdata(SL* ps)
{
    for (int i = 0;i<ps->size;i++)
    {
        printf("%d ", ps->a[i]);
    }
    printf("\n");
}
void SeqListDestory(SL* ps)
{
    free(ps->a);
```

```
    ps->a = NULL;
    //ps->size = ps->capacity = 0;
}
void SeqListPopBack(SL* ps)
{
    //ps->a[ps->size - 1] = 0;
    if (ps->size > 0)
    {
        ps->size--;
    }

}
void SeqListPushFront(SL* ps, SLdataType x)
{
    SeqListCheckCapatiy(ps);
    //挪动数据
    int end = ps->size - 1;
    while (end >= 0)
    {
        ps->a[end + 1] = ps->a[end];
        --end;
    }
    ps->a[0] = x;
    ps->size++;
}
void SeqListPopFront(SL* ps)
{
    //挪动数据
    assert(ps->size > 0);
    int front = 1;
    while (front <  ps->size)
    {
        ps->a[front - 1] = ps->a[front];
        ++front;
    }
    ps->size--;
}

void SeqListSort(SL* ps)
{
    for (int i = 0;i< ps->size-1;i++)
    {
        for (int j = 0;j<ps->size-1 -i;j++)
        {
            int temp;
            if (ps->a[j]>ps->a[j+1])
            {
                temp = ps->a[j];
                ps->a[j] = ps->a[j + 1];
                ps->a[j + 1] = temp;
            }
```

```
        }
    }
}
SLdataType SeqListMax(SL* ps)
{
    int temp = ps->a[0];
    for (int i =0;i<ps->size;i++)
    {
        if (ps->a[i]>temp)
        {
            temp = ps->a[i];
        }
    }
    return temp;
}
SLdataType SeqListFind(SL* ps, SLdataType x)
{
    for (int i = 0;i<ps->size;i++)
    {
        if (ps->a[i]==x)
        {
            return i;
        }
    }
    return -1;
}
void SeqListInsert(SL* ps, int p, SLdataType x)
{
    for (int i = ps->size-1;i>=p-1;i--)
    {
        ps->a[i + 1] = ps->a[i];
    }
    ps->a[p - 1] = x;
    ps->size++;

}
void SeqListErase(SL* ps, int p)
{
    int temp = ps->a[p - 1];
    for (int i = p;i<=ps->size;i++)
    {
        ps->a[i - 1] = ps->a[i];
    }
    ps->size--;
}
```

2.测试

```
#include "SeqList.h"
void TestSeqList1()

```

```c
{
    SL s1;
    SeqListInit(&s1);
    SeqListPushBack(&s1, 1);
    SeqListPushBack(&s1, 2);
    SeqListPushBack(&s1, 3);
    SeqListPushBack(&s1, 4);
    SeqListPushBack(&s1, 5);
    disdata(&s1);
    SeqListPopBack(&s1);
    SeqListPopBack(&s1);
    SeqListPopBack(&s1);
    SeqListPopBack(&s1);
    SeqListPopBack(&s1);
    SeqListPopBack(&s1);
    SeqListPopBack(&s1);
    disdata(&s1);
    SeqListPushBack(&s1, 225);
    SeqListPushBack(&s1, 335);
    disdata(&s1);
    SeqListDestory(&s1);
}
void TestSeqList2()
{
    SL s1;
    SeqListInit(&s1);
    SeqListPushBack(&s1, 1);
    SeqListPushBack(&s1, 2);
    SeqListPushBack(&s1, 3);
    SeqListPushBack(&s1, 4);
    SeqListPushBack(&s1, 5);
    disdata(&s1);
    SeqListPushFront(&s1,200);
    SeqListPushFront(&s1, 100);
    SeqListPushFront(&s1, 300);
    disdata(&s1);
    SeqListPopFront(&s1);
    SeqListSort(&s1);
    disdata(&s1);
    SeqListDestory(&s1);
}
void menu()
{
    //写菜单
}

int main()
{

    //TestSeqList1();
    /*TestSeqList2();*/
```

```
    SL s2;
    SeqListInit(&s2);
    SeqListPushBack(&s2, 1);
    SeqListPushBack(&s2, 2);
    SeqListPushBack(&s2, 3);
    SeqListPushBack(&s2, 4);
    SeqListPushBack(&s2, 5);
    int SeqMax = SeqListMax(&s2);
    printf("%d\n", SeqMax);
    int SeqListIndex = SeqListFind(&s2,4);
    if (SeqListIndex != -1)
    {
        printf("Find!\n");
    }else
    {
        printf("No Find!");
    }
    SeqListInsert(&s2, 2, 100);
    disdata(&s2);
    SeqListErase(&s2, 2);
    disdata(&s2);
    return 0;
}
```

3.头文件.h

```
#pragma once
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <stdbool.h>
#include <errno.h>
#include <string.h>
//#define N 1000
typedef int SLdataType;
//静态顺序表
typedef  struct SeqList
{
    //SLdataType a[N];//静态的数组
    //变成动态顺序表
    SLdataType* a;
    int size;//表示数组中存储了多少个数据
    int capacity; //表示数组的实际的空间容量多大
}SL;
//接口函数 命名根据STL库
//初始化
void SeqListInit(SL* ps);
//打印数据
void disdata(SL* ps);
//释放空间
```

```
void SeqListDestory(SL* ps);
//realloc函数 开辟空间还不是很懂
void SeqListCheckCapatiy(SL* ps);
//静态顺序表 ：如果满了就不让插入  缺点：无法确定多大的空间 就是静态的缺陷
//尾部插入
void SeqListPushBack(SL* ps, SLdataType x);
//尾部删除
void SeqListPopBack(SL* ps);
//头部插入
void SeqListPushFront(SL* ps, SLdataType x);
//头部删除
void SeqListPopFront(SL* ps);
//排序和查找
void SeqListSort(SL* ps);
SLdataType SeqListMax(SL* ps);
//查找
SLdataType SeqListFind(SL* ps,SLdataType x);
//插入
void SeqListInsert(SL* ps, int p, SLdataType x);
//删除
void SeqListErase(SL* ps, int p);
```

---

## 六、C++顺序表实现/运用malloc

C++ /感兴趣的看

```
/*
 *  ---顺序表---
 */

#define _CRT_SECURE_NO_WARNINGS 1
#include<iostream>
using namespace std;
#include<stdlib.h>//malloc函数
#include<malloc.h>
#include<string.h>
#define ture 1
#define false 0
#define ok 1
#define error 0
#define infersible -1
#define overflow -2
#define maxsize 1000
typedef int ElemType;
typedef struct {
    ElemType* data;// 数组的数据类型 elemtype  也可以改成数组动态分配
    int length;// 链表的长度
}sqlist;//功能包括：创建，初始化，插入，删除，访问，合并，输出

```

```cpp
int initlsit(sqlist& L) {
    L.data = new ElemType[maxsize];//c++new分配空间
    //L.data = (ElemType*)malloc(sizeof(ElemType) * maxsize);// 动态分配内存C
    //异常处理
    /*if(!L.length){
        printf("存储空间失败! ");
    }*/
    if (!L.data)
    {
        exit(overflow);
    }
    L.length = 0;
    return ok;
}
//创建一个有n个元素的顺序表
void CreList(sqlist& L, int n) {
    int i = 0;
    for (i = 0; i < n; i++)
    {
        cin >> L.data[i];
        L.length = n - 1;
    }

}
//判断为空?
bool isempty(sqlist& L) {
    if (L.length == 0)
    {
        return 1;
    }
    return 0;
}
//销毁顺序表
void destroylist(sqlist& L) {
    if (L.data)delete  L.data; //释放所指的空间  C++操作
}

//清空线性表
void ClearList(sqlist& L) {
    L.length = 0;  //将线性表的长度置为0
}

//获取线性表的长度
int getlen(sqlist L) {
    return L.length;
}

//查找与指定值e相同数据元素的位置
int Locatedata(sqlist L, ElemType e) {
    if (L.length == -1)
    {
```

```
        puts("此顺序表为空，无法执行“查找”操作");
        return 0;
    }

    for (int i = 0; i < L.length; i++)
    {
        if (L.data[i] == e)
        {
            return i + 1;//返回它的位置
        }
    }
    return 0;
}

//在顺序表中第i个位置插入元素e
int Inslist(sqlist& L, int i, ElemType e) {
    if (i<1 || i>L.length)
    {
        return error;
    }
    if (L.length == maxsize)
    {
        return error;
    }
    for (int j = L.length - 1; j >= i - 1; j--)
    {
        L.data[j + 1] = L.data[j];//前面一个元素向后移动
    }
    L.data[i - 1] = e;//将该元素给当前空着的这个元素 下标为i-1
    ++L.length;//插入一个数 总长度+1
    return ok;


}
//在顺序表中第i个位置删除元素e
int deldata(sqlist& L, int i) {
    ElemType temp;
    if (i<1 || i>L.length)
    {
        return error;
    }
    if (isempty(L)) {
        cout << "该顺序表是空表" << endl;
    }
    temp = L.data[i - 1];
    for (int j = i; j <= L.length - 1; j++)
    {
        L.data[j - 1] = L.data[j];
    }
    --L.length;
    return temp;
```

```
}


//查询通过位置来访问元素
int getdata(sqlist& L, int i) {
    if (i<1 || i>L.length)
    {
        return error;
    }
    cout << L.data[i - 1] << endl;
    return ok;
}

//排序该顺序表 冒泡
void sort(sqlist& L) {
    for (int i = 0; i <= L.length; i++)
    {
        for (int j = 0; j <= L.length - 1; j++)
        {
            if (L.data[j] > L.data[j + 1])
            {
                int t = L.data[j];
                L.data[j] = L.data[j + 1];
                L.data[j + 1] = t;
            }
        }
    }
}

//遍历
void disdata(sqlist& L) {
    if (isempty(L))
    {
        cout << "该顺序表是空表" << endl;
    }
    else {
        for (int i = 0; i <= L.length; i++)
        {
            printf("%d ", L.data[i]);
        }
    }
}
int main() {
    sqlist L;
    int size, cho, temp1, number, temp2;
    do
    {
        printf("请您创建一个顺序列表\n");
        printf("请输入列表的长度(长度需要小于最大长度(MaxSize):");
        cin >> size;
        printf("输入列表的元素:");
```

```
177         initlsit(L);              //进行初始化
178         CreList(L, size);//创建
179     } while (size <= 0 || size > maxsize);            //检测长度定义的列表是否超过最大
    值
180
181     do
182     {
183         printf("以下为可以进行的操作~\n");
184         printf("1_访问列表中某一个指定位置的元素\n");
185         printf("2_查找列表中某一个元素的所在位置\n");
186         printf("3_在列表中插入一个元素\n");
187         printf("4_从列表中删除一个元素\n");
188         printf("5_初始化列表\n");
189         printf("6_合并第二序列列表\n");
190         printf("7_排序\n");
191         printf("请进行选择:");
192         cin >> cho;
193     } while (cho > 7 || cho < 1); //对非法输入进行对策
194
195     //进行查找 删除 插入的操作想
196     switch (cho) {
197     case 1: printf("请输入位置:");
198         cin >> temp1;
199         //scanf("%d", &temp1);
200         getdata(L, temp1);
201         //cout << "你是否想继续操作? " << endl;
202         break;
203
204     case 2: printf("请输入元素:");
205         cin >> temp1;
206         temp1 = Locatedata(L, temp1);
207         if (temp1 == 0)
208         {
209             cout << "该元素不存在" << endl;
210         }
211         else {
212             cout << "该位置的元素是:" << temp1 << endl;
213             //printf("该位置的元素是 %d\n", temp1);
214         }
215         cout << "该线性表的元素有: " << endl;
216         disdata(L);
217         break;
218
219     case 3: printf("请输入要插入的位置:");
220         cin >> temp1;
221         cout << "请输入要插入的元素: " << endl;
222         cin >> number;
223         Inslist(L, temp1, number);
224         disdata(L);
225         break;
226
```

```
227     case 4: printf("请输入要删除元素所在的位置:");
228         cin >> temp2;
229         temp1 = deldata(L, temp2);
230         printf("您删除的元素是%d\n", temp1);
231         printf("目前列表是:");
232         disdata(L);
233         puts("");
234         break;
235     case 5:
236         initlsit(L);
237         break;
238     case 6:
239     case 7:
240         sort(L);
241         disdata(L);
242         break;
243 
244     default:puts("您输入的有错误!");
245         return 0;
246 
247 
248     }
249 
250     return 0;
251 }
252 
```

第一小节结束

制作：CY