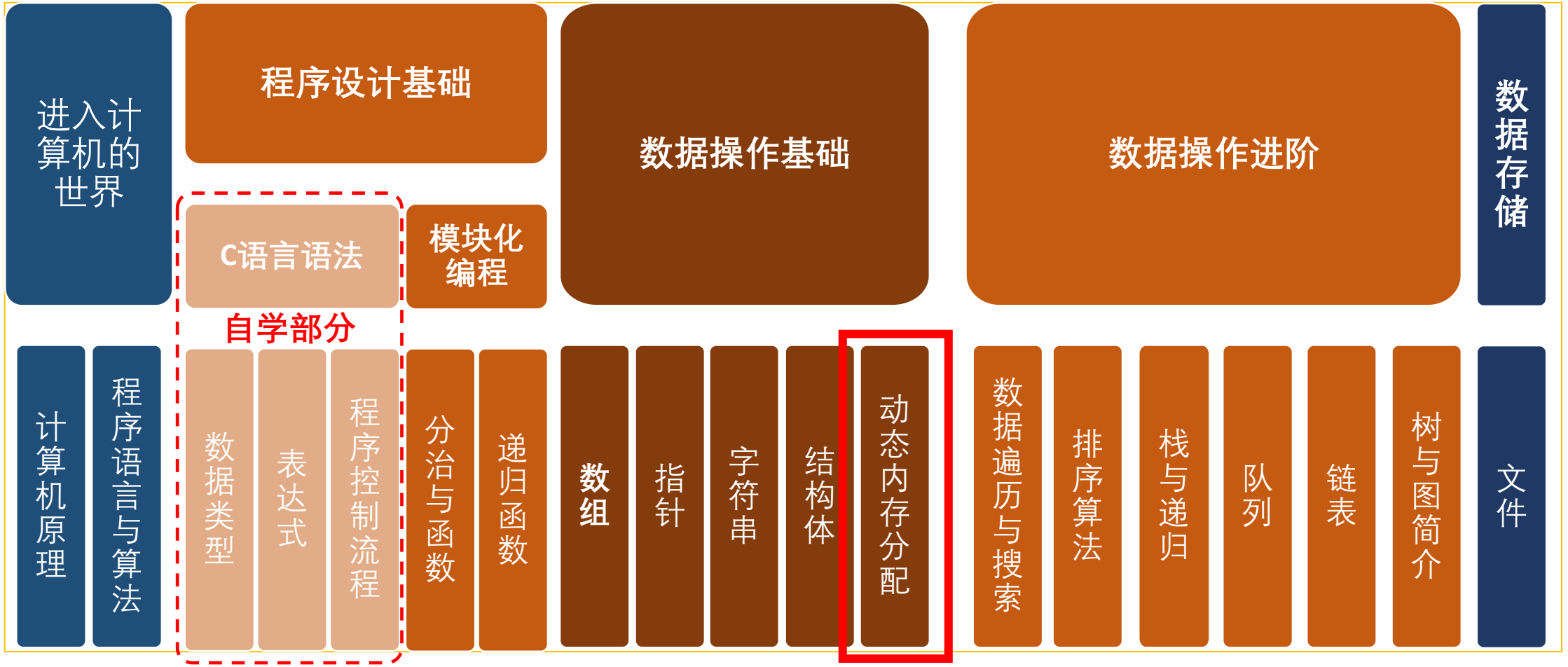
程序设计思维与实践

哈尔滨工业大学（深圳）
计算机科学与技术学院
大数据技术中心
张保权

课程内容安排



程序设计思想与数据操作方法融会贯通，内容由浅入深

第十一讲 动态内存分配——学习内容

11.1 动态内存分配

11.2 动态数组

11.3 动态栈

11.4 动态结构体数组

第十一讲 动态内存分配——学习内容

11.1 动态内存分配

11.2 动态数组

11.3 动态栈

11.4 动态结构体数组

11.1.1 动态内存分配——定长数组的缺陷

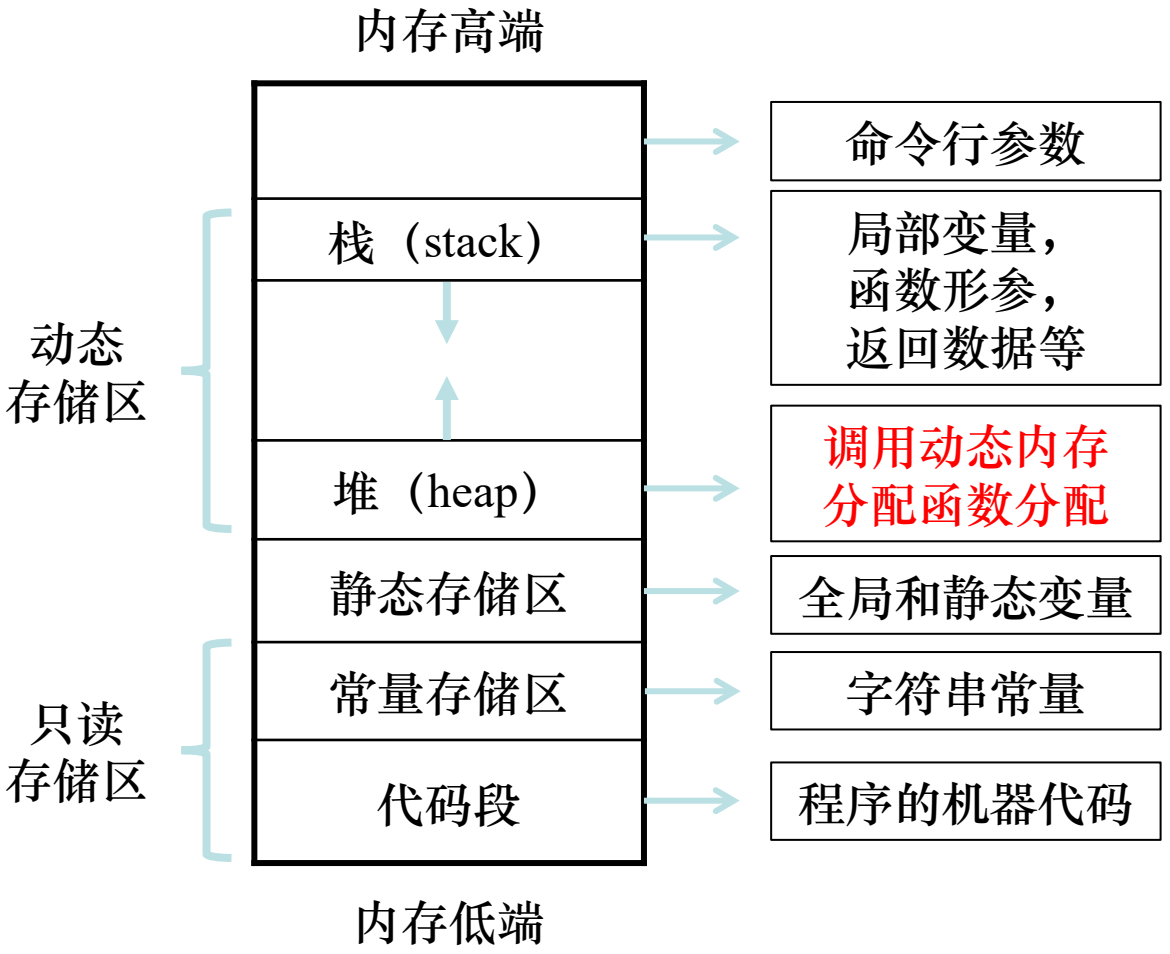
- **定长数组存在的问题：**
  - 1. 空间效率差（固定长度）
  - 2. 容易出错（溢出）

11.1.2 动态内存分配——C程序内存映像

存储区分配

编译后的C程序获得并使用4块在逻辑上不同且用于不同目的的内存储区。

- 只读存储区
  存放程序的机器代码和字符串常量等只读数据。
- 从静态（static）存储区分配
  全局变量和静态变量。



11.1.2 动态内存分配——C程序内存映像

存储区分配

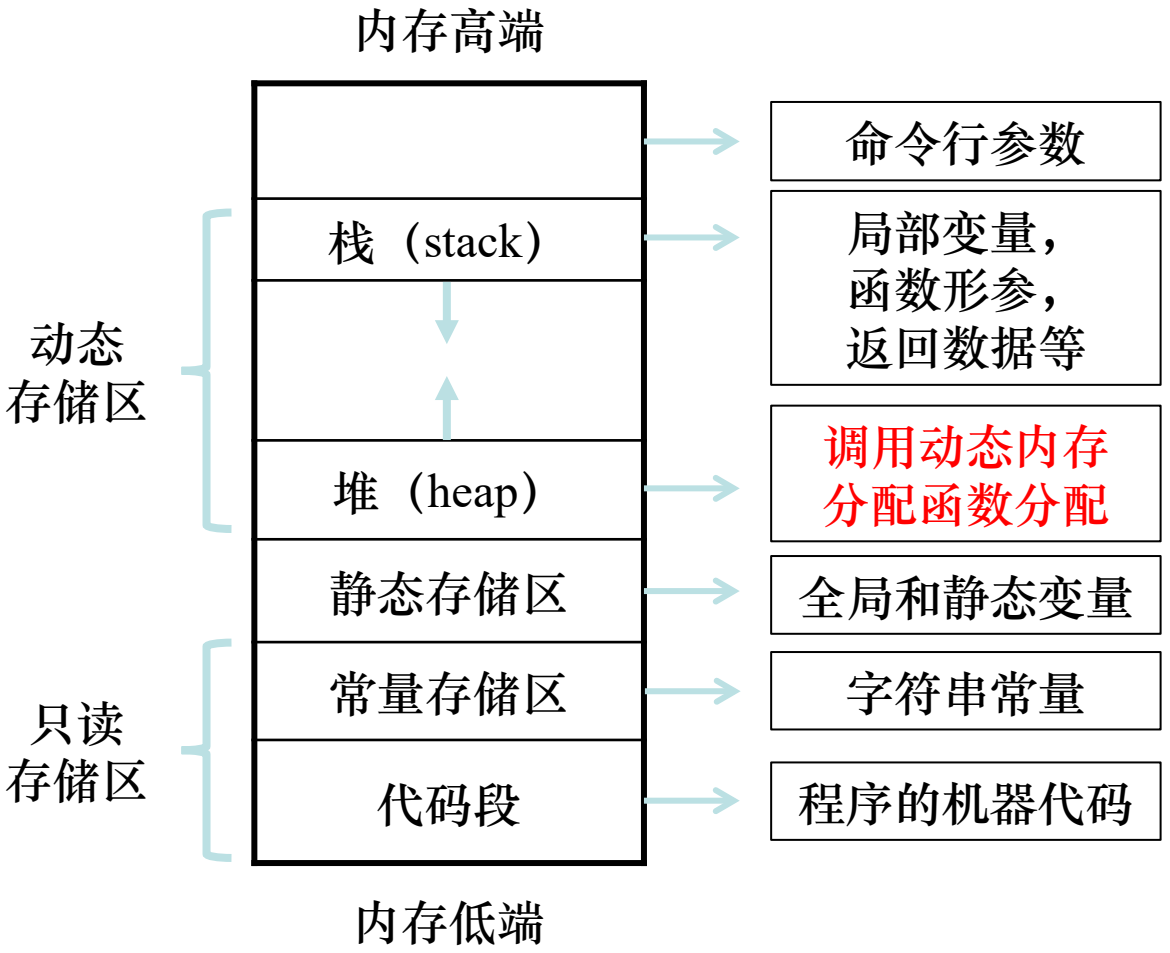
编译后的C程序获得并使用4块在逻辑上不同且用于不同目的的内存储区。

在栈（stack）上创建

- 存放函数参数值、局部变量值等
- 在执行函数调用时，系统在栈上为函数内的局部变量及形参分配内存，函数执行结束时，自动释放这些内存

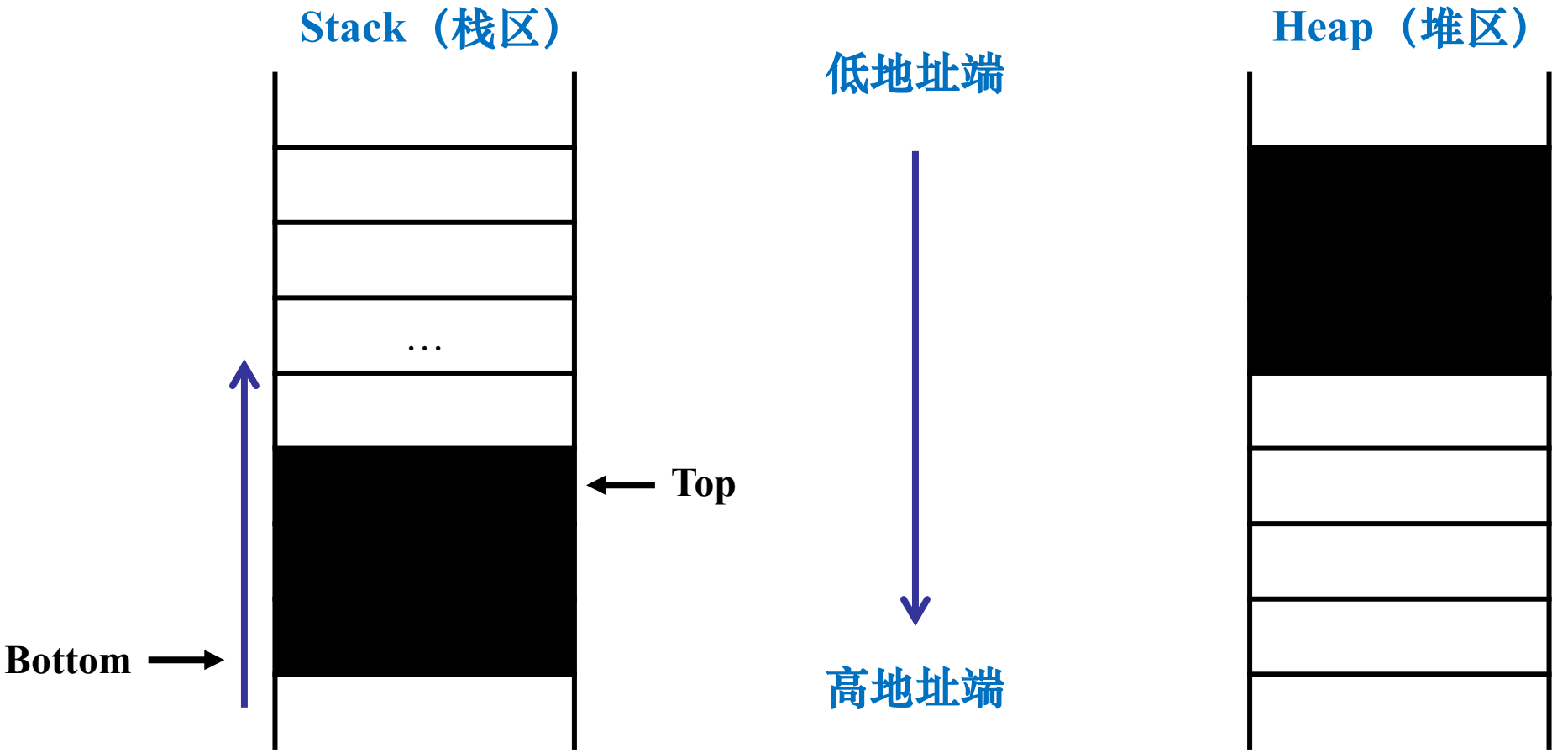
从堆（heap）上分配

- 在程序运行期间，用动态内存分配函数来申请的内存都是从堆上分配的，动态内存的生存期由程序员自己来决定



11.1.2 动态内存分配——C程序内存映像

堆 vs 栈



11.1.2 动态内存分配——C程序内存映像

堆 vs 栈

➢ **Stack-based memory**
- 生存期由函数决定

➢ **特点:**
- 由编译系统自动分配释放
  无需程序员管理
- 生长方向向下
- 分配效率高
- 无碎片问题

➢ **Heap-based memory**
- 生存期由程序员决定

➢ **特点:**
- 程序员不释放，会造成内存泄漏
  (memory leakage)
- 生长方向向上
- 分配效率低
- 频繁申请/释放易造成内存碎片
  (heap fragmentation)

11.1.3 动态内存分配——内存分配函数

两种主要的内存分配方法

```
#include <stdlib.h>
#include <alloc.h>

void* malloc(unsigned int size);
void* calloc(unsigned int num, unsigned int size);
```

返回值类型 void*

- void*类型的指针可以指向任意类型的变量，通常强转(Type*)为其他类型

11.1.3 动态内存分配——内存分配函数

两种主要的内存分配方法

```
void* malloc(unsigned int size);
```

- 向系统申请大小为**size**的内存块
- 把首地址返回，若申请不成功则返回**NULL**

```
void* calloc(unsigned int num, unsigned int size);
```

- 向系统申请**num**个大小为**size**的内存块
- 把首地址返回，若申请不成功则返回**NULL**

11.1.3 动态内存分配——内存分配函数

两种主要的内存分配方法

```
void* malloc(unsigned int size);
```

例：
```
int *pi = NULL;
pi = (int *)malloc(sizeof(int));
```

```
void* calloc(unsigned int num, unsigned int size);
```

例：
```
float *pf = NULL;
pf =(float *)calloc(10,sizeof(float));
```

11.1.3 动态内存分配——内存分配函数

内存修改方法

```
void* realloc(void* p, unsigned int size);
```

- 改变原来分配的存储空间的大小
- p是指向此块内存的指针，size是新内存块的大小
- 函数返回新分配存储空间首地址，与原来分配的首地址不一定相同

11.1.3 动态内存分配——内存分配函数

内存回收方法

```
void free(void* p);
```

- 释放由**malloc()**和**calloc()**申请的内存块
- **p**是指向此块内存的指针
- **free**时系统标记此块内存为未占用，可被重新分配

第十一讲 动态内存分配——学习内容

11.1 动态内存分配

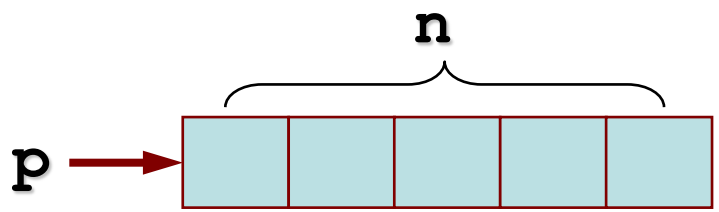
11.2 动态数组

11.3 动态栈

11.4 动态结构体数组

11.2.1 动态数组——一维

例1



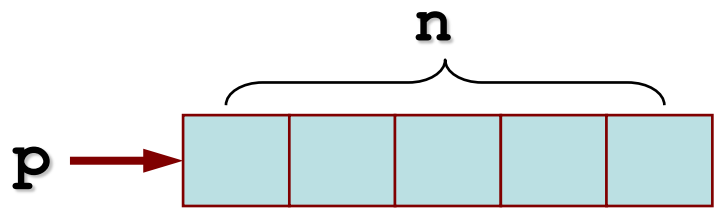
确保指针使用前是非空指针

释放向系统申请的存储空间

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   void InputArray(int *p, int n);
4   double Average(int *p, int n);
5   int  main()
6   {
7       int  *p = NULL, n;
8       double aver;
9       printf("How many students?");
10      scanf("%d", &n);
11      p = (int *) malloc(n * sizeof(int));
12      if (p == NULL)
13      {
14          printf("No enough memory!\n");
15          exit(1);
16      }
17      printf("Input %d score:", n);
18      InputArray(p, n);
19      aver = Average(p, n);
20      printf("aver = %.1f\n", aver);
21      free(p);
22      return 0;
23  }
```

11.2.1 动态数组——一维

例1



像使用一维数组一样使用动态数组

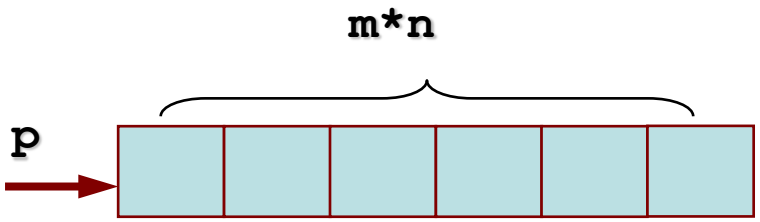
输出：

```
How many students? 5↙
Input 5 score: 90  85  70  95  80↙
aver = 84.0
```

```
24  /* 形参声明为指针变量，输入数组元素的
25  void InputArray(int *p, int n)
26  {
27      int i;
28      for (i=0; i<n; i++)
29      {
30          scanf("%d", &p[i]);
31      }
32  }
33  /* 形参声明为指针变量，计算数组元素的平均值 */
34  double Average(int *p, int n)
35  {
36      int i, sum = 0;
37      for (i=0; i<n; i++)
38      {
39          sum = sum + p[i];
40      }
41      return (double)sum / n;
42  }
```

11.2.2 动态数组——二维

例2



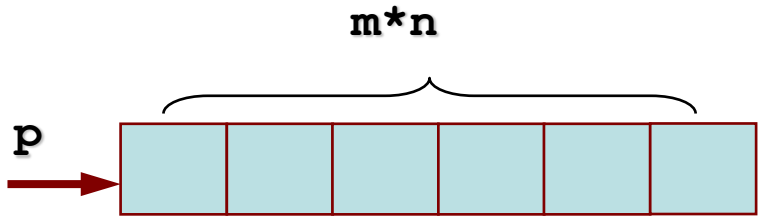
确保指针使用前是非空指针

释放向系统申请的存储空间

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   void InputArray(int *p, int m, int n);
4   double Average(int *p, int m, int n);
5   int  main()
6   {
7       int  *p = NULL, m, n;
8       double aver;
9       printf("How many classes?");
10      scanf("%d", &m);
11      printf("How many students in a class?");
12      scanf("%d", &n);
13      p = (int *)calloc(m*n, sizeof(int));
14      if (p == NULL)
15      {
16          printf("No enough memory!\n");
17          exit(1);
18      }
19      InputArray(p, m, n);
20      aver = Average(p, m, n);
21      printf("aver = %.1f\n", aver);
22      free(p);
23      return 0;
24  }
```

11.2.2 动态数组——二维

例2



仍当做一维数组来使用

输出：

```
How many classes? 3↙
How many students in a class? 4↙
Please enter scores of class 1:
81  72  73  64↙
Please enter scores of class 2:
65  86  77  88↙
Please enter scores of class 3:
91  90  85  92↙
aver = 80.3
```

```
25  /* 形参声明为指向二维数组的列指针，输入数组元素值 */
26  void InputArray(int *p, int m, int n)
27  {
28      int i, j;
29      for(i = 0; i<m; i++)        /* m个班 */
30      {
31          printf("Please enter scores of class %d:\n", i+1);
32          for(j = 0; j<n; j++)    /* 每班n个学生 */
33          {
34              scanf("%d", &p[i*n+j]);
35          }
36      }
37  }
```

```
38  /* 形参声明为指针变量，计算数组元素的平均值 */
39  double Average(int *p, int m, int n)
40  {
41      int i, j, sum = 0;
42      for(i = 0; i<m; i++)        /* m个班 */
43      {
44          for(j = 0; j<n; j++)    /* 每班n个学生 */
45          {
46              sum = sum + p[i*n+j];
47          }
48      }
49      return (double)sum / (m*n);
50  }
```

第十一讲 动态内存分配——学习内容

11.1 动态内存分配

11.2 动态数组

11.3 动态栈

11.4 动态结构体数组

11.3 动态栈

栈结构体中使用动态内存

```
//----- 栈的动态存储表示 -----
#define  STACK_INIT_SIZE  100;    //栈容量
#define  STACKINCREMENT   10;  //栈增量
typedef struct {
    DataType  *base;        //基地址
    DataType  *top;         //栈顶
    int  stacksize;         //栈容量
} SqStack;
```

```
//----- 栈的初始化-----
Status InitStack (SqStack *S) // 构造一个空栈S
{
 S.base=(ElemType*)malloc(STACK_INIT_SIZE*
                          sizeof(ElemType));
 if (!S.base) exit (OVERFLOW); //存储分配失败
 S.top = S.base;
 S.stacksize = STACK_INIT_SIZE;
 return OK;
}
```

11.3 动态栈

栈结构体中使用动态内存

```
  //----- 栈的动态扩容-----
Status Push (SqStack *S, SElemType e) {
  if (S.top - S.base >= S.stacksize)            //栈满，追加存储空间
    { S.base = (ElemType *) realloc ( S.base,
       (S.stacksize + STACKINCREMENT) *
                                        sizeof (ElemType));
      if (!S.base) exit (OVERFLOW);        //存储分配失败
      S.top = S.base + S.stacksize;
      S.stacksize += STACKINCREMENT;
    }
  S.top = e; S.top++; //先传数据再移动指针
  return OK;
}
```

第十一讲 动态内存分配——学习内容

11.1 动态内存分配

11.2 动态数组

11.3 动态栈

11.4 动态结构体数组

11.4.1 结构体数组

结构体示例

```
typedef struct student
{
    long   studentID;
    char   studentName[10];
    char   studentSex;
    DATE   birthday;
    int    score[4];
} STUDENT;
```

```
typedef struct date
{
    int   year;
    int   month;
    int   day;
}DATE;
```

学号	姓名	性别	出生日期			数学	英语	计算机原理	程序设计
			年	月	日				

11.4.1 结构体数组

```
STUDENT stu[30] = {
        {100310121, "王刚", 'M',{1991,5,19},{72,83,90,82}},
        {100310122, "李小明", 'M',{1992,8,20},{88,92,78,78}},
        {100310123, "王丽红", 'F',{1991,9,19},{98,72,89,66}},
        {100310124, "陈莉莉", 'F',{1992,3,22},{87,95,78,90}}
};
```

学号	姓名	性别	出生日期			数学	英语	计算机原理	程序设计
			年	月	日				
1	王　刚	M	1991	5	19	72	83	90	82
2	李小明	M	1992	8	20	88	92	78	78
3	王丽红	F	1991	9	19	98	72	89	66
4	陈莉莉	F	1992	3	22	87	95	78	90

建立了数据库中的多条记录，每条对应一个学生信息

11.4.2 定长的结构体数组

```
int main(){
    int i, j, sum[30];
    STUDENT stu[30] = {
            {100310121,"王刚",'M',{1991,5,19},{72,83,90,82}},
            {100310122,"李小明",'M',{1992,8,20},{88,92,78,78}},
            {100310123,"王丽红",'F',{1991,9,19},{98,72,89,66}},
            {100310124,"陈莉莉",'F',{1992,3,22},{87,95,78,90}}
    };
    for (i=0; i<4; i++) {
        sum[i] = 0;
        for (j=0; j<4; j++){
            sum[i] = sum[i] + stu[i].score[j];
        }
        printf("%10ld%8s%3c%6d/%02d/%02d%4d%4d%4d%4d%6.1f\n",
                stu[i].studentID, stu[i].studentName,
                stu[i].studentSex, stu[i].birthday.year,
                stu[i].birthday.month, stu[i].birthday.day,
                stu[i].score[0], stu[i].score[1], stu[i].score[2], stu[i].score[3],
                sum[i]/4.0);
    }
    return 0;
}
```

```
100310121    王刚  M  1991/05/19  72  83  90  82  81.8
100310122  李小明  M  1992/08/20  88  92  78  78  84.0
100310123  王丽红  F  1991/09/19  98  72  89  66  81.2
100310124  陈莉莉  F  1992/03/22  87  95  78  90  87.5

--------------------------------
Process exited after 0.07958 seconds with return value 0
请按任意键继续. . .
```

例3

利用结构体数组，计算每个学生的平均分

11.4.3 动态结构体数组

```
#include <stdio.h>
#include <stdlib.h>

typedef struct student{
    char* name;
    int age;
}Stu;

int main(){
    int num=3;
    Stu* ptr = (Stu*)malloc(num*sizeof(Stu));          // 使用malloc分配
    //Stu* ptr = calloc(num, sizeof(Stu));             // 使用calloc分配
    ptr[0].age = 18; ptr[0].name = "Alice";
    ptr[1].age = 19; ptr[1].name = "Bob";
    ptr[2].age = 20; ptr[2].name = "Charlie";
    printf("First student's name: %s\n", ptr->name);   // 访问第一个学生的名字
    printf("First student's age: %d\n", ptr->age);     // 访问第一个学生的年龄
    free(ptr);
    return 0;
}
```

```
First student's name: Alice
First student's age: 18

--------------------------------
Process exited after 0.0135 seconds with return value 0
请按任意键继续. . .
```

当将动态内存分配的数据赋给指针时，指针指向第一个元素。

在左侧的代码中，通过Stu* ptr =(Stu*)malloc(num*sizeof(Stu))将动态分配内存赋给指针ptr，此时ptr指向第一个元素。我们可以通过ptr->name和ptr->age访问第一个学生的名字和年龄。

11.4.3 动态结构体数组

```
#include <stdio.h>
#include <stdlib.h>
typedef struct student{
    char* name;
    int age;
}Stu;
int main() {
    int num=3;
    Stu* ptr = (Stu*)malloc(num*sizeof(Stu));          // 使用malloc分配
    //Stu* ptr = calloc(num, sizeof(Stu));             // 使用calloc分配
    ptr[0].age = 18; ptr[0].name = "Alice";
    ptr[1].age = 19; ptr[1].name = "Bob";
    ptr[2].age = 20; ptr[2].name = "Charlie";
    // 通过指针访问后续元素
    printf("First student's name: %s\n", (ptr + 1)->name); // 访问第二个学生的名字
    printf("First student's age: %d\n", (ptr + 1)->age);   // 访问第二个学生的年龄
    printf("First student's name: %s\n", (ptr + 2)->name); // 访问第三个学生的名字
    printf("First student's age: %d\n", (ptr + 2)->age);   // 访问第三个学生的年龄
    free(ptr);
    return 0;
}
```

```
First student's name: Bob
First student's age: 19
First student's name: Charlie
First student's age: 20

--------------------------------
Process exited after 0.01365 seconds with return value 0
请按任意键继续. . .
```

可以使用指针的偏移操作来访问下一个元素！

(ptr + 1)：向后移动一个元素，
(ptr + 2)：向后移动两个元素。
(ptr + 1)->name
(ptr + 2)->name

小结

- 动态内存分配
- 应用：动态数组
- 应用：动态栈
- 应用：动态结构体数组