# C 语言作业总结

第1-4-7次作业

# Scanf 语句

**函数原型** int scanf(const char *format, ...)

**函数说明** 从标准输入流 stdin(键盘) 读取输入。

**返回值** 成功时返回输入的总个数，失败时返回一个负数。

**参数说明**

第一个参数是格式字符串，指定了输入的格式。

第二个是可变参数列表，每一个指针要求非空，并且与字符串中的格式符一一顺次对应。

| type | Qualifying Input | Type of argument |
|---|---|---|
| c | Single character: Reads the next character. If a width different from 1 is specified, the function reads width characters and stores them in the successive locations of the array passed as argument. No null character is appended at the end. | char * |
| d | Decimal integer: Number optionally preceded with a + or - sign | int * |
| e, E, f, g, G | Floating point: Decimal number containing a decimal point, optionally preceded by a + or - sign and optionally followed by the e or E character and a decimal number. Two examples of valid entries are -732.103 and 7.12e4 | float * |
| o | Octal Integer: | int * |
| s | String of characters. This will read subsequent characters until a whitespace is found (whitespace characters are considered to be blank, newline and tab). | char * |
| u | Unsigned decimal integer. | unsigned int * |
| x, X | Hexadecimal Integer | int * |

```c
#include <stdio.h>
int main()
{

    int count;
    int a, b;
    count = scanf("%d%d", &a, &b);
    printf("count: %d\n", count);
    return 0;
}
```

在高版本的 Visual Studio 编译器中，scanf 被认为是不安全的，被弃用，应当使用 scanf_s 代替 scanf。

```c
#include <stdio.h>
int main()
{
    char str1[10];
    char str2[10];
    scanf("%s", str1);
    gets(str2);//gets(str2, 10, stdin);

    printf("str1: %s\n", str1);
    printf("str2: %s\n", str2);
    return 0;
}
```

# 浮点数表示

| 类型 | 存储大小 | 值范围 | 精度 |
| --- | --- | --- | --- |
| float<br>单精度 | 4字节(32位) | 1.2E-38 到 3.4E+38 | 6 位小数 |
| double<br>双精度 | 8字节(64位) | 2.3E-308 到 1.7E+308 | 15 位小数 |
| long double<br>长双精度 | 16字节(128位) | 3.4E-4932 到 1.1E+4932 | 19 位小数 |

```c
#include<stdio.h>
int main()
{
    float a, b, c;
    a = 0.5;
    b = 0.4;
    c = 0.3;
    printf("a = %.9f\n", a);
    printf("b = %.9f\n", b);
    printf("c = %.9f\n", c);
    return 0;
}
```

# 字符、转义字符

**字符**

字符型数据是用单引号括起来的一个字符。

**转义字符**

转义字符是一种特殊的字符。转义字符以反斜线"\"开头，后跟一个或几个字符。

转义字符具有特定的含义，不同于字符原有的意义，故称“转义”字符。

```c
#include<stdio.h>
int main()
{
    char c1 = 'c';
    char c2 = '\103';
    printf("c1: %c\n", c1);
    printf("c2: %c\n", c2);
    return 0;
}
```

| 转义字符 | 转义字符的意义 | ASCII代码 |
| --- | --- | --- |
| \n | 回车换行 | 10 |
| \t | 横向跳到下一制表位置 | 9 |
| \b | 退格 | 8 |
| \r | 回车 | 13 |
| \f | 走纸换页 | 12 |
| \\ | 反斜线符"\" | 92 |
| \' | 单引号符 | 39 |
| \" | 双引号符 | 34 |
| \a | 鸣铃 | 7 |
| \ddd | 1～3位八进制数所代表的字符 | |
| \xhh | 1～2位十六进制数所代表的字符 | |

```
#include<stdio.h>
int main()
{
    char str[] = "abh\\012\\\"";
    printf("%d\n", (int)sizeof(str));
    return 0;
}
```

上次说错了，抱歉！

# 大小写转换

| | |
|---|---|
| 大写转小写 | c = c + 32; |
| 小写转大写 | c = c - 32; |

| Dec | Hx | Oct | Char | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 000 | NUL (null) | 32 | 20 | 040 |   | Space | 64 | 40 | 100 | @ | @ | 96 | 60 | 140 | ` | ` |
| 1 | 1 | 001 | SOH (start of heading) | 33 | 21 | 041 | ! | ! | 65 | 41 | 101 | A | A | 97 | 61 | 141 | a | a |
| 2 | 2 | 002 | STX (start of text) | 34 | 22 | 042 | " | " | 66 | 42 | 102 | B | B | 98 | 62 | 142 | b | b |
| 3 | 3 | 003 | ETX (end of text) | 35 | 23 | 043 | # | # | 67 | 43 | 103 | C | C | 99 | 63 | 143 | c | c |
| 4 | 4 | 004 | EOT (end of transmission) | 36 | 24 | 044 | $ | $ | 68 | 44 | 104 | D | D | 100 | 64 | 144 | d | d |
| 5 | 5 | 005 | ENQ (enquiry) | 37 | 25 | 045 | % | % | 69 | 45 | 105 | E | E | 101 | 65 | 145 | e | e |
| 6 | 6 | 006 | ACK (acknowledge) | 38 | 26 | 046 | & | & | 70 | 46 | 106 | F | F | 102 | 66 | 146 | f | f |
| 7 | 7 | 007 | BEL (bell) | 39 | 27 | 047 | ' | ' | 71 | 47 | 107 | G | G | 103 | 67 | 147 | g | g |
| 8 | 8 | 010 | BS (backspace) | 40 | 28 | 050 | ( | ( | 72 | 48 | 110 | H | H | 104 | 68 | 150 | h | h |
| 9 | 9 | 011 | TAB (horizontal tab) | 41 | 29 | 051 | ) | ) | 73 | 49 | 111 | I | I | 105 | 69 | 151 | i | i |
| 10 | A | 012 | LF (NL line feed, new line) | 42 | 2A | 052 | * | * | 74 | 4A | 112 | J | J | 106 | 6A | 152 | j | j |
| 11 | B | 013 | VT (vertical tab) | 43 | 2B | 053 | + | + | 75 | 4B | 113 | K | K | 107 | 6B | 153 | k | k |
| 12 | C | 014 | FF (NP form feed, new page) | 44 | 2C | 054 | , | , | 76 | 4C | 114 | L | L | 108 | 6C | 154 | l | l |
| 13 | D | 015 | CR (carriage return) | 45 | 2D | 055 | - | - | 77 | 4D | 115 | M | M | 109 | 6D | 155 | m | m |
| 14 | E | 016 | SO (shift out) | 46 | 2E | 056 | . | . | 78 | 4E | 116 | N | N | 110 | 6E | 156 | n | n |
| 15 | F | 017 | SI (shift in) | 47 | 2F | 057 | / | / | 79 | 4F | 117 | O | O | 111 | 6F | 157 | o | o |
| 16 | 10 | 020 | DLE (data link escape) | 48 | 30 | 060 | 0 | 0 | 80 | 50 | 120 | P | P | 112 | 70 | 160 | p | p |
| 17 | 11 | 021 | DC1 (device control 1) | 49 | 31 | 061 | 1 | 1 | 81 | 51 | 121 | Q | Q | 113 | 71 | 161 | q | q |
| 18 | 12 | 022 | DC2 (device control 2) | 50 | 32 | 062 | 2 | 2 | 82 | 52 | 122 | R | R | 114 | 72 | 162 | r | r |
| 19 | 13 | 023 | DC3 (device control 3) | 51 | 33 | 063 | 3 | 3 | 83 | 53 | 123 | S | S | 115 | 73 | 163 | s | s |
| 20 | 14 | 024 | DC4 (device control 4) | 52 | 34 | 064 | 4 | 4 | 84 | 54 | 124 | T | T | 116 | 74 | 164 | t | t |
| 21 | 15 | 025 | NAK (negative acknowledge) | 53 | 35 | 065 | 5 | 5 | 85 | 55 | 125 | U | U | 117 | 75 | 165 | u | u |
| 22 | 16 | 026 | SYN (synchronous idle) | 54 | 36 | 066 | 6 | 6 | 86 | 56 | 126 | V | V | 118 | 76 | 166 | v | v |
| 23 | 17 | 027 | ETB (end of trans. block) | 55 | 37 | 067 | 7 | 7 | 87 | 57 | 127 | W | W | 119 | 77 | 167 | w | w |
| 24 | 18 | 030 | CAN (cancel) | 56 | 38 | 070 | 8 | 8 | 88 | 58 | 130 | X | X | 120 | 78 | 170 | x | x |
| 25 | 19 | 031 | EM (end of medium) | 57 | 39 | 071 | 9 | 9 | 89 | 59 | 131 | Y | Y | 121 | 79 | 171 | y | y |
| 26 | 1A | 032 | SUB (substitute) | 58 | 3A | 072 | : | : | 90 | 5A | 132 | Z | Z | 122 | 7A | 172 | z | z |
| 27 | 1B | 033 | ESC (escape) | 59 | 3B | 073 | ; | ; | 91 | 5B | 133 | [ | [ | 123 | 7B | 173 | { | { |
| 28 | 1C | 034 | FS (file separator) | 60 | 3C | 074 | < | < | 92 | 5C | 134 | \ | \ | 124 | 7C | 174 | | | \| |
| 29 | 1D | 035 | GS (group separator) | 61 | 3D | 075 | = | = | 93 | 5D | 135 | ] | ] | 125 | 7D | 175 | } | } |
| 30 | 1E | 036 | RS (record separator) | 62 | 3E | 076 | > | > | 94 | 5E | 136 | ^ | ^ | 126 | 7E | 176 | ~ | ~ |
| 31 | 1F | 037 | US (unit separator) | 63 | 3F | 077 | ? | ? | 95 | 5F | 137 | _ | _ | 127 | 7F | 177 | &#127; | DEL |

Source: www.LookupTables.com

# 字符数组

用来存放字符的数组称为字符数组。

字符数组实际上是一系列字符的集合，也就是字符串。在C语言中，没有专门的字符串变量，没有string类型，通常就用一个字符数组来存放一个字符串。

应该说明的是，对一个字符数组，如果不作初始化赋值，则必须说明数组长度。还应该特别注意的是，当用scanf函数输入字符串时，字符串中不能含有空格，否则将以空格作为串的结束符。

数组名就代表了该数组的首地址。整个数组是以首地址开头的一块连续的内存单元。

```c
#include<stdio.h>
int main()
{
    char str1[] = "abcd";
    char str2[] = {'a', 'b', 'c', 'd'};
    printf("size of str1: %d\n", (int)sizeof(str1));
    printf("size of str2: %d\n", (int)sizeof(str2));

    return 0;
}
```

# 指向字符串的指针

C语言中没有特定的字符串类型，我们通常是将字符串放在一个字符数组中。

除了字符数组，C语言还支持另外一种表示字符串的方法，就是直接使用一个指针指向字符串

```
char *str2 = "abcd";
```

这一切看起来和字符数组很相似，它们都可以使用%s输出整个字符串，都可以使用*或[ ]获取单个字符，这两种表示字符串的方式是不是就没有区别了呢?

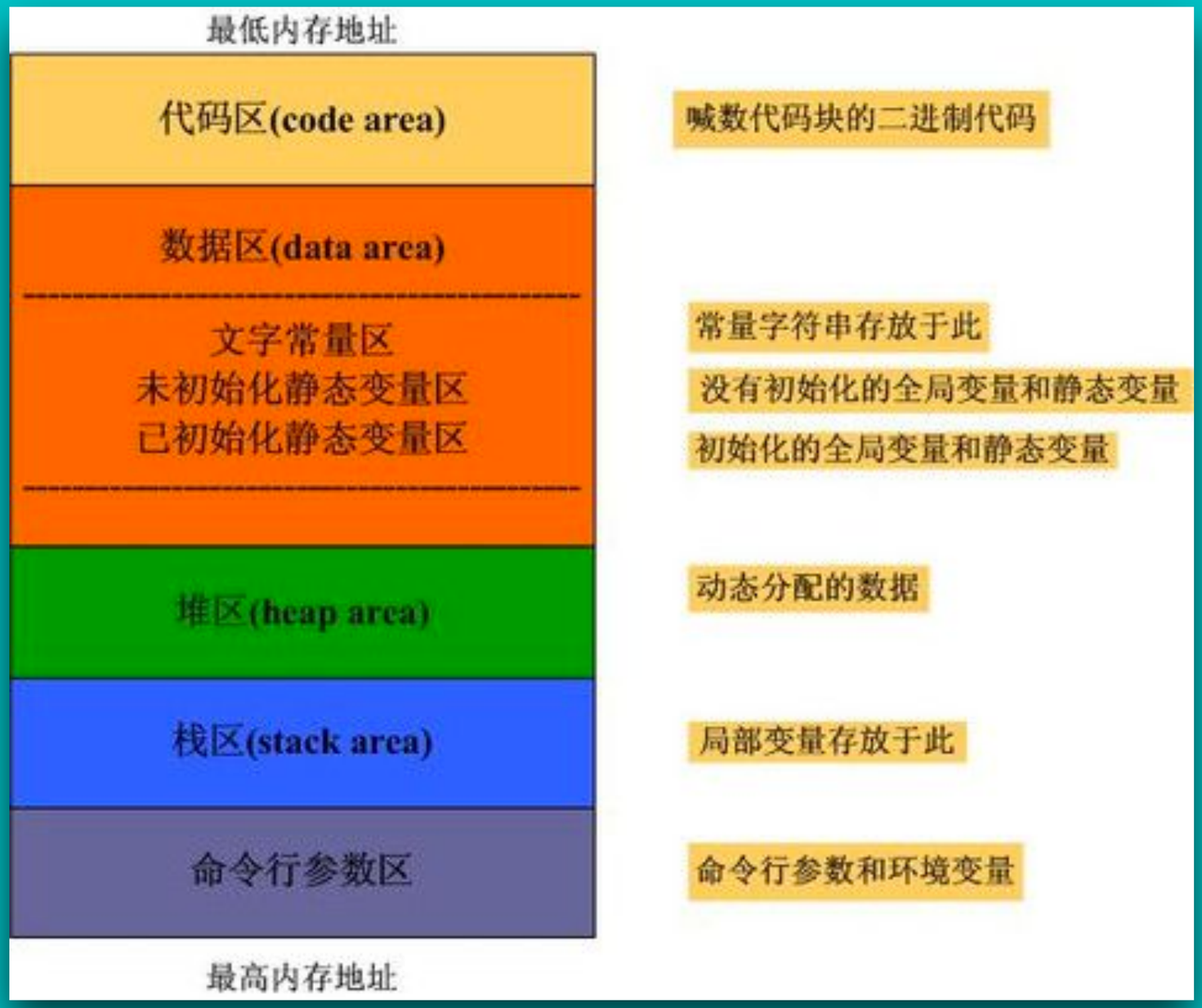
# 指向字符串的指针

它们最根本的区别是在内存中的存储区域不一样，字符数组存储在全局数据区或栈区，第二种形式的字符串存储在常量区。全局数据区和栈区的字符串（也包括其他数据）有读取和写入的权限，而常量区的字符串（也包括其他数据）只有读取权限，没有写入权限。

内存权限的不同导致的一个明显结果就是，字符数组在定义后可以读取和修改每个字符，而对于第二种形式的字符串，一旦被定义后就只能读取不能修改，任何对它的赋值都是错误的。

```c
#include <stdio.h>
int main()
{
    char *str = "Hello World!";
    str = "I love C!";
    str[3] = 'P';
    return 0;
}
```

这段代码能够正常编译和链接，但在运行时会出现段错误（Segment Fault）或者写入位置错误。




1、栈区（stack）：又编译器自动分配释放，存放函数的参数值，局部变量的值等，其操作方式类似于数据结构的栈。

2、堆区（heap）：一般是由程序员分配释放，若程序员不释放的话，程序结束时可能由OS回收，值得注意的是他与数据结构的堆是两回事，分配方式倒是类似于数据结构的链表。

3、全局区（static）：也叫静态数据内存空间，存储全局变量和静态变量，全局变量和静态变量的存储是放一块的，初始化的全局变量和静态变量放一块区域，没有初始化的在相邻的另一块区域，程序结束后由系统释放。

4、文字常量区：常量字符串就是放在这里，程序结束后由系统释放。

5、程序代码区：存放函数体的二进制代码。

# 字符串常量&字符数组

```
char str1[] = “abcd";
//或者char str1[] = {‘a’, ‘b’, ‘c’, ‘d’};

char *str2 = "abcd";
```

C语言有两种表示字符串的方法，一种是字符数组，另一种是字符串常量，它们在内存中的存储位置不同，使得字符数组可以读取和修改，而字符串常量只能读取不能修改。

# 字符串常量&字符数组

到底使用字符数组还是字符串常量?

在编程过程中如果只涉及到对字符串的读取，那么字符数组和字符串常量都能够满足要求；如果有写入（修改）操作，那么只能使用字符数组，不能使用字符串常量。

```
#include <stdio.h>
int main()
{
    char str[30];
    fgets(str, 30, stdin);
    printf("%s\n", str);
    return 0;
}
```

# 二分查找(binary search)

二分搜索，也称折半搜索（英语：half-interval search）是一种在**有序数组**中查找某一特定元素的搜索演算法。

搜索过程从数组的中间元素开始，如果中间元素正好是要查找的元素，则搜索过程结束；

如果某一特定元素大于或者小于中间元素，则在数组大于或小于中间元素的那一半中查找，而且跟开始一样从中间元素开始比较。

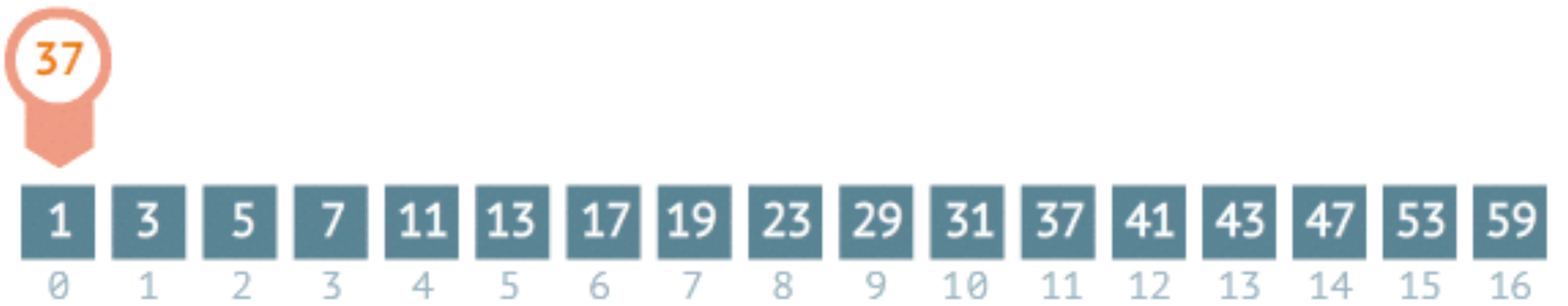
如果在某一步骤数组为空，则代表找不到。这种搜索算法每一次比较都使搜索范围缩小一半。

**Binary** search

steps: 0


**Sequential** search

steps: 0



# Demo

# 线性表

线性表（Linear List）是由n（n≥0）个数据元素（结点）a[0]，a[1]，a[2]…，a[n-1]组成的有限序列。

## 线性表的存储结构

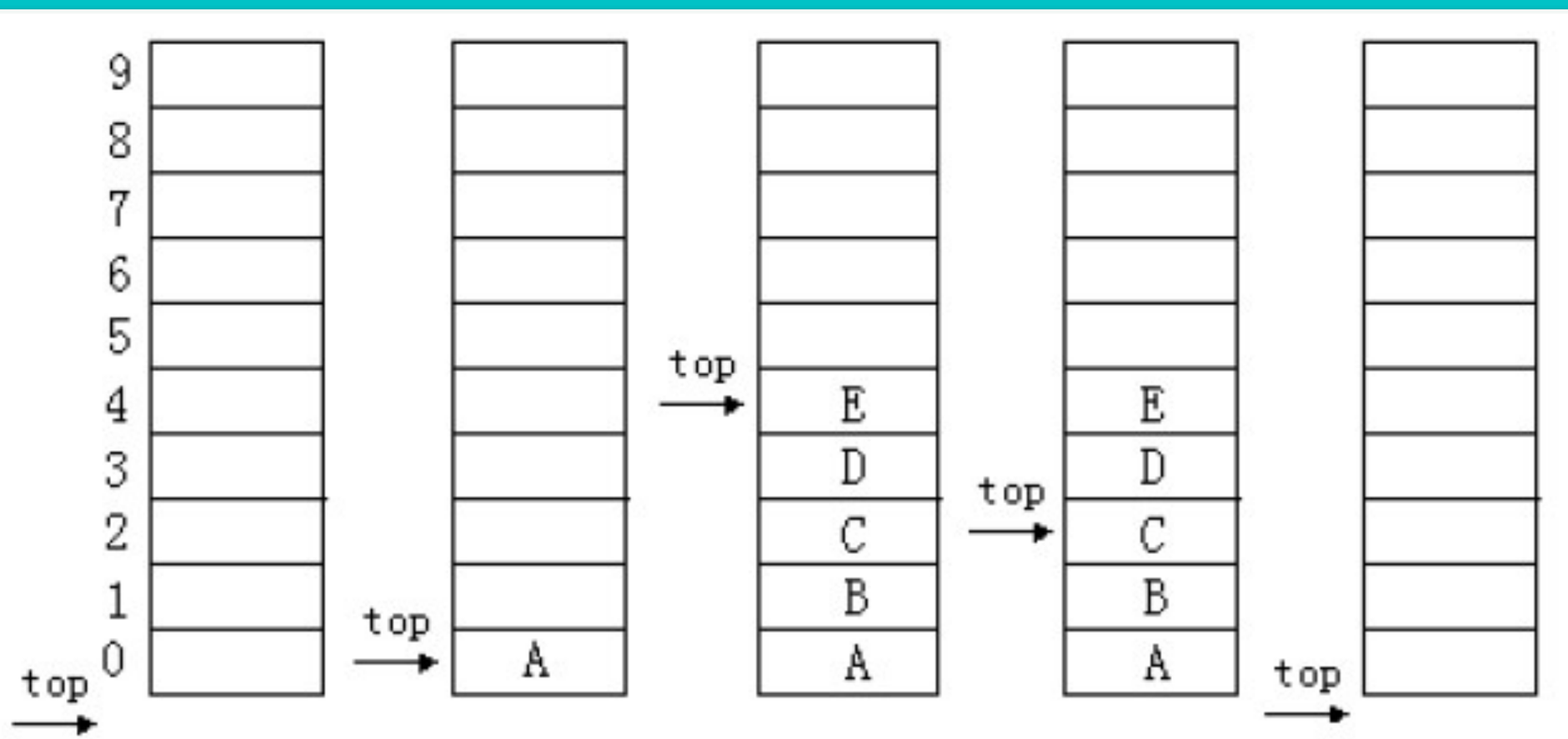
- 顺序表
- 链表

# Demo

# 栈

> 由于栈是运算受限的线性表，因此线性表的存储结构对栈也是适用的，只是操作不同而已。

```
typedef struct
{
    datatype data[MAXSIZE];
    int top;
}SeqStack;
```




top=-1　　top=0　　top=4　　top=2　　top=-1

(a)空栈　　(b)一个元素　　(c)5 个元素　　(d)3 个元素　　(e)空栈

图 3.2　栈顶指针 top 与栈中数据元素的关系

# 队列

顺序存储的队称为顺序队。因为队的队头和队尾都是活动的，因此，除了队列的数据区外还有队头、队尾两个指针。

```
typedef struct
{
    datatype data[MAXSIZE];
    int rear;
    int front;
}SeQueue;
```

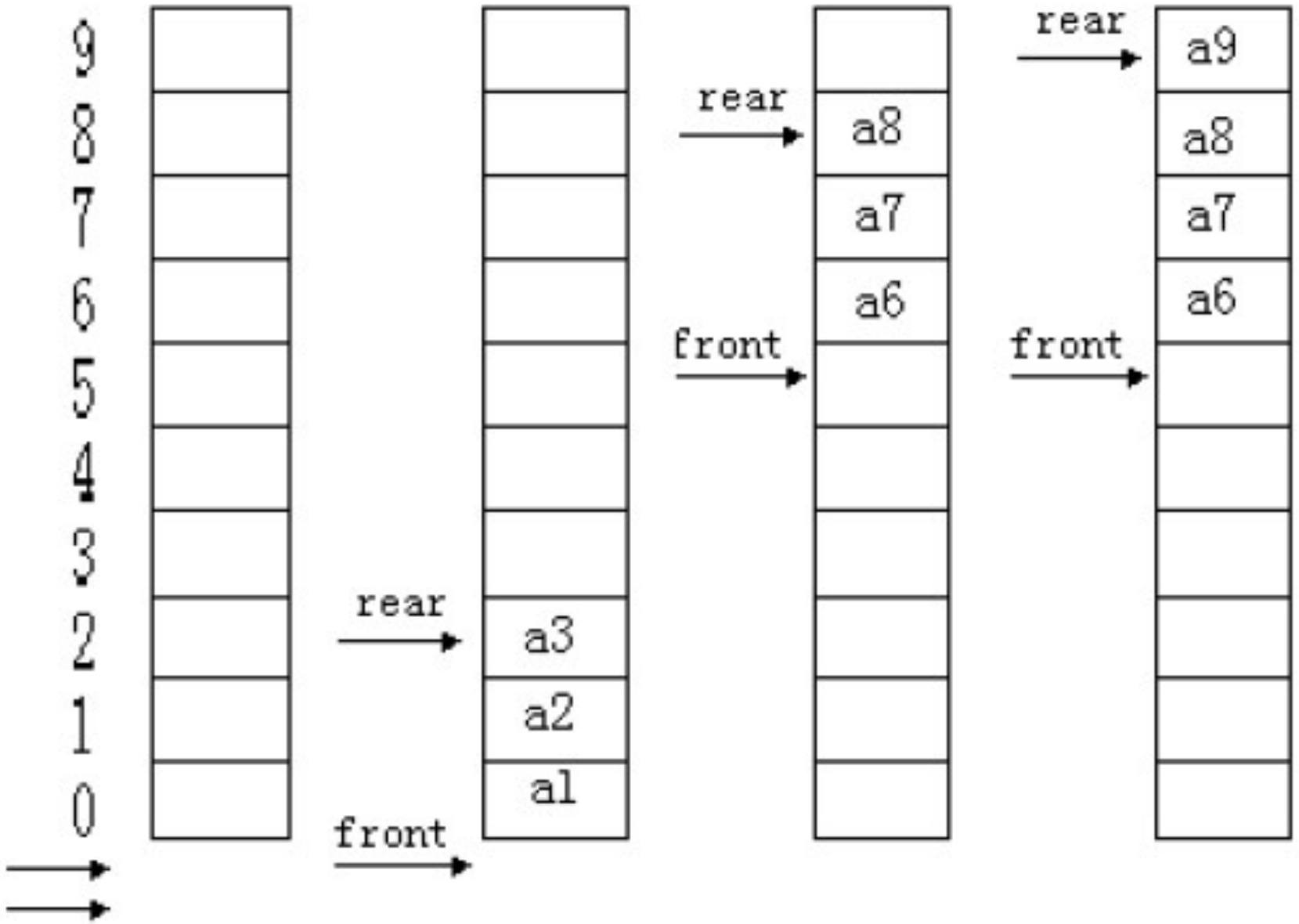



图 3.12　队列操作示意图

# 内存管理

void *calloc(int num, int size);
该函数分配一个带有 num 个元素的数组，每个元素的大小为 size 字节。

void free(void *address);
该函数分配一个带有 num 个元素的数组，每个元素的大小为 size 字节。

void *malloc(int num);
该函数分配一个 num 字节的数组，并把它们进行初始化。

void *realloc(void *address, int newsize);
该函数重新分配内存，把内存扩展到 newsize。

# Demo

祝大家考试取得好成绩！

1；浮点数表示，分解数
2；输入输出语句
3；Ascii码
2；字符、转义字符、字符串
3；字符指针
4；内存
5；排序
6；递归
7；插入
7；结构体
8；队列
9；栈
10; 线性表