

待办

2021年11月22日 17:47

Acwing 1.7 59min  
大根堆 小根堆  
重载  
420 火星人 acwing

英语

视频报告

14周考翻译  
拔高题 没有原句

15周考词汇 抽40个  
12346单元3、7题  
t校园 全是原题

14周作文概要  
夫学生活 充分利用时间  
广泛的尝试 学会承担责任  
150-200 不能忽略原文

链表  
循环大题  
引用  
结构体  
P1296 sort排 双指针  
埃氏筛法

回文质数  
方块转化

听说

听力测试30分  
1-8单元 1-10  
自主学习10分 5h  
日语ppt 20分

软件部作业  
1、寒假上网课 数据结构和算法  
2、服务外包竞赛 （可以了解软件开发）  
3、c++书->slt标准库->  
4、大一大二时间多 多放精力尽情打比赛 多努力 大三四大事多  
5、22年蓝桥后 会详细地讲c++  
6、卷

政治

六选三

体育

四次跑步

# 基本

2021年9月21日 22:07

**关系运算** 如果符合预期输出整数1, 否则为整数0.

优先级优先级: 算数 > 关系 > 赋值

==或!= 比其他关系运算符优先级低

连续的关系运算从左到右依次进行

## 注释

/ 只能单行/

/\* 能多行\*/

还有%3.1f



他的意思是至少三位数 有一位小数



a+=1是a=a+1的简写

a=-a 的意思: 把a赋值负号

i++显示的是加后的

++i显示的是加前的

## Vs2019 新建文件:

### CTRL + shift+A

三角形三条边 从小到大 a,b,c

两个短边之和大于长边 就是三角形

1%3

叫取余

结果为1

```
1  #include <stdio.h>
2  int main(void)
3  {
4      int i=1;
5      int fact = 1;
6      int n;
7
8      scanf_s("%d",&n);
9      while (i <= n) {
10         fact *= i;
11         i++;
12     }
13     printf("%d!=%d", n, fact);
14     return 0;
15 }
```

# 输入输出

2022年1月3日 21:50

`scanf` 读入`double`类型的浮点数 要用`%lf`  
`printf` 输出`double`类型的浮点数 用`%f`就行

`scanf`和`printf` 要比`cin`和`cout`快 在面试和算法竞赛中使用 防止出错

## 特殊方法

在主函数中 添加这样一语句

```
ios::sync_with_stdio(false);  
//优点：提高cin cout的读写速度  
//缺点：不能再使用scanf
```

# 数据类型大小

2021年11月2日 11:06

```
D:\DevC\数据类型字节大小.exe

sizeof(int)=4
sizeof(short)=2
sizeof(long)=4
sizeof(long long)=8
sizeof(float)=4
sizeof(double)=8
sizeof(char)=1
```

指针相减

指针相减的得数=地址差/sizeof(数据类型)

也就是说有几个这样类型的元素

\*p++

意思是

先取出p所指向的数值

再把p移到下一个位置去

\*的优先级<++

常用于数组类的连续空间操作

```
数据类型字节大小.cpp 星p++.cpp
1 #include<stdio.h>
2 int main()
3 {
4     int ac[]={0,1,2,3,4,5,-1};
5     int *p=ac;
6     while(*p!=-1)
7     {
8         printf("%d ",*p++);
9     }
10    printf("\n");
11    for(int i=0;i<sizeof(ac)/sizeof(ac[0]);i++)
12    {
13        printf("%d ",ac[i]);
14    }
15    return 0;
16 }
```

```
D:\DevC\星p++.exe
0 1 2 3 4 5
0 1 2 3 4 5 -1
Process exited after 0.06434 seconds with return value 0
请按任意键继续. . .
```

# 例题

2021年11月29日 19:28

## 鸡兔同笼

```
#include<stdio.h>
int main(void)
{
    int x;
    int y;
    int rabbit = 0;
    int hen = 0;
    printf("输入鸡兔数:");
    scanf_s("%d", &x);
    printf("输入脚数:");
    scanf_s("%d", &y);
    rabbit=(y-2*x)/2;
    hen = x - rabbit;

    if (hen<=0 || rabbit<= 0) {
        printf("无解");
    }
    else {
        printf("兔子:%d\n鸡:%d\n", rabbit, hen);
    }
    return 0;
}
```

## 移花接木

```
#include <stdio.h>
int main(void)
{
    float a,b,c, d, e, f;

    scanf_s("%f %f", &a, &b);

    c = a - (int)a;
    d = b - (int)b;
    e = (int)a + d;
    f = (int)b + c;

    printf("%.2f %.2f", e, f);
    return 0;
}
```

## 阶乘

```

1      #include <stdio.h>
2      int main(void)
3      {
4          int i=1;
5          int fact = 1;
6          int n;
7
8          scanf_s("%d",&n);
9          while (i <= n) {
10             fact *= i;
11             i++;
12         }
13         printf("%d!=%d",n,fact);
14         return 0;
15     }

```

## 猜字

```

1      #include<stdio.h>
2      #include<stdlib.h>
3      #include<time.h>
4      int main()
5      {
6          srand(time(0));
7
8          int i=1;
9          int number;
10         int n=rand()%100;
11
12         scanf("%d",&number);
13         while(n!=number){
14             if(number<n){
15                 printf("您的数小了! \n");
16             }else if(number>n){
17                 printf("您的数大了! \n");
18             }
19             scanf("%d",&number);
20             i++;
21         }
22         printf("太好了用了%d次就猜到了答案~",i);
23         return 0;
24     }

```

```

srand(time(0));
int number = rand()%100+1;
int count = 0;
int a = 0;
printf("我已经想好了一个1到100之间的数。");
do {
    printf("请猜这个1到100之间数: ");
    scanf("%d", &a);
    count ++;
    if ( a > number ) {
        printf("你猜的数大了。");
    } else if ( a < number ) {
        printf("你猜的数小了。");
    }
} while (a != number);
printf("太好了, 你用了%d次就猜到了答案.\n", count);

```

## 倒序

```

1  #include<stdio.h>
2  int main()
3  {
4      int x;
5      int y;
6      int z=0;
7      scanf("%d",&x);
8      while(x>0)
9      {
10         y=x%10;
11         z=z*10+y;
12         printf("%d\n",z);
13         x/=10;
14     }
15     printf("%d\n",z);
16     return 0;
17 }

```

对于一个整数

%10 得到个位数

/10 除掉个位数

## 体积转换问题

```
case 13:
{
    double v;
    int V;
    v=4/3*3.141593*(4*4*4+1000);
    printf("v=%lf\n",v);
    V=(int)v;
    double r=1.0/3;
    double R;
    printf("r=%lf\n",r);
    R=pow(V,r);
    printf("R=%f",R);
    int x=(int)R;
    printf("%d",x);
}
```

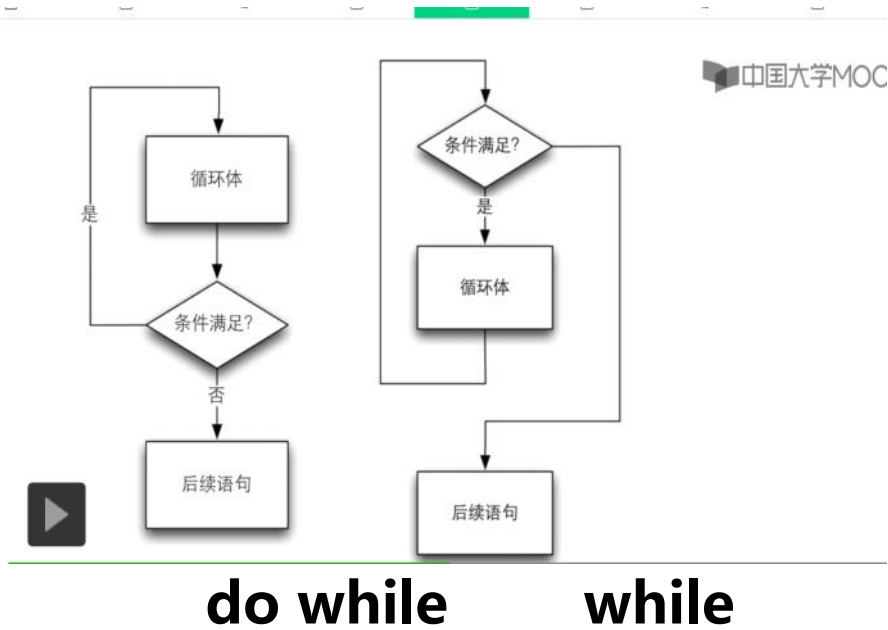
---



# 循环

2021年9月21日 22:11

## While与do while循环



```
#include<stdio.h>
int main(void)
{
    long x=0;
    long n=0;

    scanf_s("%d",&x);

    while (x > 0) {
        n++;
        x /= 10;
        printf("%d %d\n", x, n);
    }
    printf("您输入的位数是:%d\n", n);
    return 0;
}
```

```
#include <stdio.h>
int main(void)
{
    int x;
    int n=0;

    scanf_s("%d", &x);

    do {
        x /= 10;
        n++;
    }
    while (x > 0);
    printf("%d",n);
    return 0;
}
```

## for循环

For(i=1;i<=n;n++);

for循环里有三个语句:

第一个语句是 初始条件

第二个语句是 循环继续的条件

第三个语句是 循环每轮要做的动作

## 循环次数

- `for ( i=0; i<n; i++ )`
- 则循环的次数是n，而循环结束以后，i的值是n。循环的控制变量i，是选择从0开始还是从1开始，是判断*i*<n还是判断*i*≤n，对循环的次数，循环结束后变量的值都有影响



## for循环

```
for ( 初始动作; 条件; 每轮的动作 ) {  
}
```

- for中的每一个表达式都是可以省略的

```
for ( ; 条件; ) == while ( 条件 )
```

有固定次数用 for

如果必须执行一次 用 do while

其他情况: while

## 多重循环

```
#include<stdio.h>
int main()
{
    int x;
    int cnt = 0;
    x = 2;
    while (cnt < 51)
    {
        int i;
        int isprime = 1;//是素数
        for (i = 2; i < x; i++)
        {
            if (x % i == 0)
            {
                isprime = 0;
                break;
            }
        }

        if (isprime == 1)
        {
            printf("%d", x);
            cnt++;
        }
        x++;
    }
    return 0;
}
```

## 循环应用 分裂整数

---

```
int main()
{
    int x;
    scanf("%d", &x);

    x = 10;
    int mask = 1;
    int t = x;
    while ( t>9 ) {
        t /= 10;
        mask *=10;
    }
    printf("x=%d, mask=%d\n", x, mask);
    do {
        int d = x / mask;
        printf("%d", d);
        if ( mask > 9 ) {
            printf(" ");
        }
        x %= mask;
        mask /= 10;
    } while ( mask > 0 );
    printf("\n");
}
```

# 逻辑运算

2021年9月26日 15:25

中国大学MOOC

## 逻辑运算

- 逻辑运算是对逻辑量进行的运算，结果只有0或1
- 逻辑量是关系运算或逻辑运算的结果

运算符	描述	示例	结果
!	逻辑非	!a	如果a是true结果就是false，如果a是false结果就是true
&&	逻辑与	a && b	如果a和b都是true，结果就是true；否则就是false
	逻辑或	a    b	如果a和b有一个是true，结果就是true；两个都是false，结果就是false

与 一个为假即为假  
或 一个为真即为真

数学中的区间 在C语言中用 逻辑与表示

如  $x > 4 \& \& x < 6$

判断 c 是否为大写字母  
 $c > = 'A' \& \& c < = 'Z'$

$!age < 20$  无论如何 逻辑运算后都等于1

$!(age < 20)$  相当于  $age > = 20$

中国大学MOOC

## 优先级

优先级	运算符	结合性
1	()	从左到右
2	! + - ++ --	从右到左 (单目的+和-)
3	* / %	从左到右
4	+ -	从左到右
5	< <= > >=	从左到右
6	== !=	从左到右
7	&&	从左到右
8		从左到右
9	= += -= *= /= %%	从右到左

逻辑运算 自左向右进行 如果左边能够决定结果 就不会向下进行

对于&& 左边是false 就不会进行右边

对于|| 左边是true 右边不进行

# 条件和逗号运算

2021年9月26日 15:25

Count=(count>20)?count-10:count+10

第一个是条件

第二个是条件满足时的值

第三个是条件不满足的值

逗号运算作用：在for循环中使用

For(i=10,j=0;1>=j;i--,j++)

# 嵌套

2021年9月28日 11:17

else总是和最近的if匹配 除了大括号

```
1  #include<stdio.h>
2  int main(void)
3  {
4      int max = 0;
5      int a,b,c;
6      printf("请输入三个要比较大小的数 并用空格间隔\n");
7      scanf_s("%d %d %d", &a, &b, &c);
8
9      if (a < b) {
10         if (b < c) {
11             max = c;
12         } else {
13             max = b;
14         }
15     } else {
16         if (a < c) {
17             max = c;
18         } else {
19             max = a;
20         }
21     }
22     printf("%d", max);
23     return 0;
24 }
```

养成好习惯

即使一条if也要用{}

```
#include<stdio.h>
int main(void)
{
    int max = 0;
    int a,b,c;
    printf("请输入三个要比较大小的数 并用空格间隔\n");
    scanf_s("%d %d %d", &a, &b, &c);

    if (a < b) {
        if (b < c) {
            max = c;
        } else {
            max = b;
        }
    } else {
        if (a < c) {
            max = c;
        } else {
            max = a;
        }
    }
    printf("%d", max);
    return 0;
}
```

# 级联与 switch case

2021年9月30日 14:10

## Else if

```
1 #include<stdio.h>
2 int main()
3 {
4     int abs,x;
5     scanf_s("%d", &x);
6     if (x > 0) {
7         abs = 1;
8     }else if (x == 0) {
9         abs = 0;
10    }else {
11        abs = 2;
12    }
13    printf("%d", abs);
14    return 0;
15 }
```

## Switch(type)

Case 1:

语句;

Break;

Case2:

语句;

Break;

.....  
Default:

语句;

Break;

}

使用switch 必须是整数类型

## switch-case

```
switch ( 控制表达式 ) {
case 常量:
    语句
    .....
case 常量:
    语句
    .....
default:
    语句
    .....
}
```

- 控制表达式只能是整数型的结果
- 常量可以是常数，也可以是常数计算的表达式

中国大学MOOC

## break

```
switch ( type )
{
case 1:
case 2:
    printf("你好\n");
    break;
case 3:
    printf("晚上好\n");
case 4:
    printf("再见\n");
    break;
default:
    printf("啊，什么啊? \n");
    break;
}
```

- switch语句可以看作是一种基于计算的跳转，计算控制表达式的值后，程序会跳转到相匹配的case（分支标号）处。分支标号只是说明switch内部位置的路标，在执行完分支中的最后一条语句后，如果后面没有break，就会顺序执行到下面的case里去，直到遇到一个break，或者switch结束为止。

在switch中break比case更重要

case只是决定从哪里开始的位置

而break决定的是程序的终止

## Programming Ability Test

主页  
题目集

基本信息  
题目列表  
提交列表  
排名  
帮助

## 分支-06. 成绩转换(15)

本题要求编写程序将一个百分制成绩转换为五分制成绩。转换规则：

- 大于等于90分为A；
- 小于90且大于等于80为B；
- 小于80且大于等于70为C；
- 小于70且大于等于60为D；
- 小于60为E。

输入格式：

```
//printf("输入成绩 (0-100)");
int grade;
scanf("%d", &grade);
grade /=10;
switch ( grade ) {
case 10:
case 9:
    printf("A\n");
    break;
case 8:
    printf("B\n");
    break;
}
```



## Programming Ability Test

主页  
题目集  
基本信息  
题目列表  
提交列表  
排名  
帮助

### 分支-06. 成绩转换(15)

本题要求编写程序将一个百分制成绩转换为五分制成绩。转换规则：

- 大于等于90分为A；
- 小于90且大于等于80为B；
- 小于80且大于等于70为C；
- 小于70且大于等于60为D；
- 小于60为E。

输入格式：

输入在一行中给出1个整数的百分制成绩。

输出格式：

输入样例：

98

输出样例：

A

```
1 #include<stdio.h>
2 int main()
3 {
4     char name;
5     int score;
6
7     scanf_s("%d", &score);
8     score /= 10;
9
10    switch (score)
11    {
12        case 10:
13        case 9:
14            name = 'A';
15            break;
16        case 8:
17            name = 'B';
18            break;
19        default:
20            name = 'F';
21            break;
22    }
23
24    printf("%c", name);
25    return 0;
26 }
```

单一出口

```
//printf("输入成绩 (0-100)");
int grade;
scanf("%d", &grade);
grade /= 10;
switch ( grade ) {
    case 10:
    case 9:
        printf("A\n");
        break;
    case 8:
        printf("B\n");
        break;
    case 7:
        printf("C\n");
        break;
    case 6:
        printf("D\n");
        break;
    default:
        printf("F\n");
        break;
}
```

这段代码不符合“单一出口”的原则，因为我们还没学过字符或字符串数据的

%c 单一字符

%s 字符串

用switch时 如果一个case里有需要声明则需要括起来

```
switch(n)
{
    case 1:
        printf("I love Luogu!\n");
        break;
    case 2:
        printf("6 4\n");
        break;
    case 3:
    {
        int num=14,p=4;
        int x;
        x=num/p;
        printf("%d %d %d\n",x,x*p,num-x*p);
        break;
    }
    case 4:
    {
        float x=500/3;
        printf("%f\n",x);
        break;
    }
    default:
        break;
}
```

# Break和continue

2021年10月1日 21:55

break跳出循环

Continue

不管if后面的语句 重复完循环后才执行后面的语句

```
1  #include<stdio.h>
2  int main()
3  {
4      int x,i;
5      scanf("%d",&x);
6      int isprime=1;
7
8      for(i=2;i<x;i++)
9      {
10         if(x%i==0)
11         {
12             isprime=0;
13             break; //跳转 一旦能满足不是素数的条件 就直接结束 不再一直重复for循环
14         }
15     }
16
17     if(isprime==0)
18     {
19         printf("不是素数\n");
20     }
21     else
22     {
23         printf("ta是素数\n");
24     }
25
26     return 0;
27 }
```

# 数组

2021年10月12日 16:53

类型 名称 [数量]

举例

Int number [100]

Double weight [20]

数组是一种容器

其中所有的元素具有相同的数据类型

一旦创建，不能改变大小

数组中的元素在内存中是连续依次排列的

注意：数量只能是整数

C99之前元素数量必须是常量

之后版本可以拿变量

一个数组

Int a [10]

长度为0的数组

可以存在

没有用处

C99之后可以拿变量定义元素数量

可以出现在赋值的左边和右边

a[9]=a[1]+2;

赋值左边的叫左值

```
1  #include<stdio.h>
2  int main()
3  {
4      int x,i;
5      const int number=10; // 数组的大小
6      int count[number]; // 定义数组
7      for(i=0;i<number;i++) // 数组初始化
8      {
9          count[i]=0;
10     }
11     scanf("%d",&x);
12     while(x!=-1)
13     {
14         if(x>=0|x<=9) // 数组参与运算
15         {
16             count[x]++;
17         }
18         scanf("%d",&x);
19     }
20     for(i=0;i<number;i++) // 数组逐个输出
21     {
22         printf("%d:%d\n",i,count[i]);
23     }
24     return 0;
25 }
```

二维数组

int a[3][5]

意思是建立一个三行五列的矩阵

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]

## 二维数组的初始化

中国大学MOOC

```
int a[][5] = {
    {0,1,2,3,4},
    {2,3,4,5,6},
};
```

- 列数是必须给出的，行数可以由编译器来数
- 每行一个{}，逗号分隔



a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]

```
};
```

- 列数是必须给出的，行数可以由编译器来数
- 每行一个[], 逗号分隔
- 最后的逗号可以存在，有古老的传统
- 如果省略，表示补零
- 也可以用定位 (\* C99 ONLY)



```
Int a[]={5,5,5}
//数组a的0、1、2位的数字分别是5、5、5
Int a[3]={6}
//数组a的0、1、2位的数值分别是6、0、0
Int a[6]={[1]=9,[4]=7}
//数组a 0 1 2 3 4 5
          0 9 9 0 7 0
```

遍历数组常见的错误

- 1、循环结束条件i<=数组长度（注意那个0）
- 2、离开循环后继续用i做下标

数组的大小

Sizeof

=Sizeof(a)/sizeof(a[0])

数组的赋值

```
Int a[]={1,2,3,4,5,6}
```

✗ Int b[]=a

✓ for(i=0,i<sizeof a/sizeof a[0],i++)//遍历数组

```
{
    b[i]=a[i];
}
```

错题

4 若有：

```
int a[][3] = {{0},{1},{2}};
```

则

```
a[1][2]
```

的值是？

10

正确答案：0

二维数组是线性排列的可以把所有数据写在一个花括号内

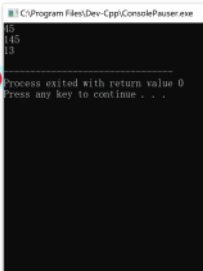
```
Int num[3]
```

```
[3]={1, 2, 3, 4, 5, 6, 7, 8, 9};
```

# 函数

2021年10月13日 19:06

```
1 #include<stdio.h>
2 void sum(int begin,int end)
3 {
4     int i;
5     int sum=0;
6     for(i=begin;i<=end;i++)
7     {
8         sum+=i;
9     }
10    printf("%d\n",sum);
11 }
12 int main()
13 {
14     sum(1,9);
15     sum(10,19);
16     sum(6,7);
17     return 0;
18 }
```



如果函数有返回值 必须使用带值的return

没有返回值的函数

函数名用void

不能使用带值的return

可以没有return

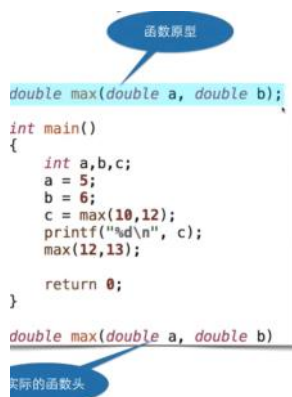
调用的时候不能做返回值的赋值

最基本的函数应用例题

把函数头 在第一行 原型声明 后 + 分号结尾;

可以把函数的主体放在后面

目的: 告诉编译器函数的 名称 参数 返回类型



编译器会自动转换类型

结果可能不是你想要的

c语言在调用函数时只能传值给函数

本地变量

本地变量定义在块内

可以定义在函数的块内

可以定义在语句的块内

可以定义在大括号内

程序运行在块之前变量不存在 块之后也不存在

定义在块外的变量在块里依然有效

块里面定义了和外面同名的 则掩盖外面的

本地变量不会被默认初始化

参数在进入函数时就被初始化了

注意局部变量（函数内）和全局变量（函数外）

不同函数的变量不可以互相访问

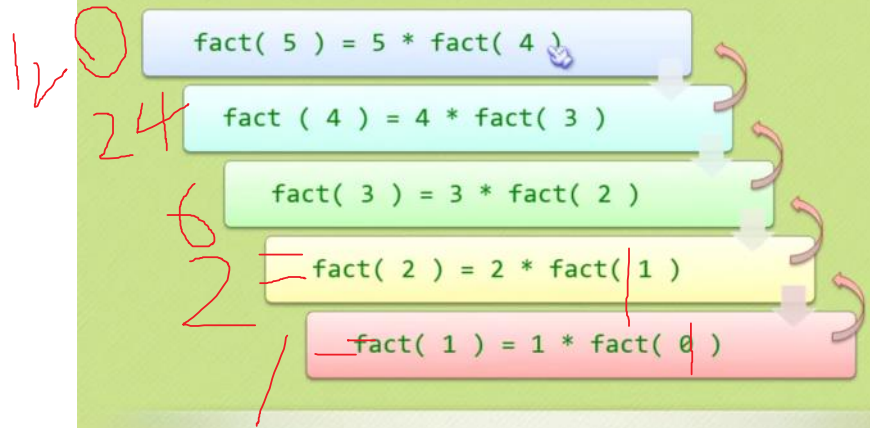
如果不对全局变量初始化 它默认为0

# 递归

2021年10月30日 18:52

两个条件

- 1、调用函数本身
  - 2、有结束条件
- 有去有回 归去来兮  
慎用!



# 学习

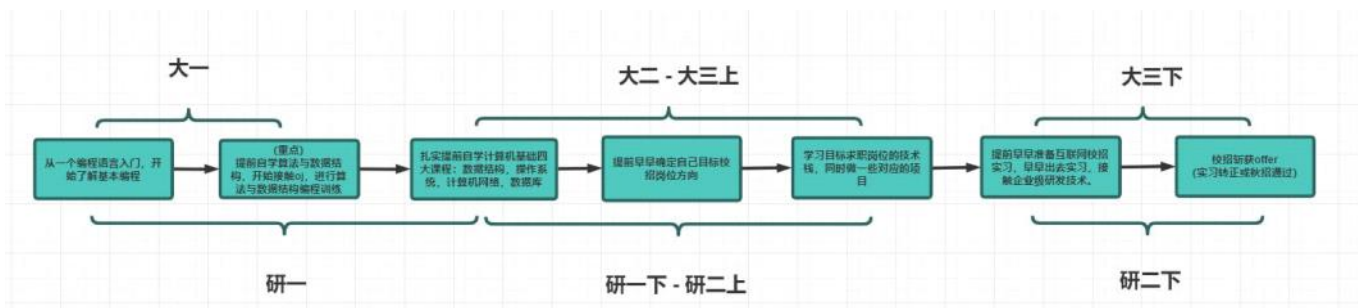
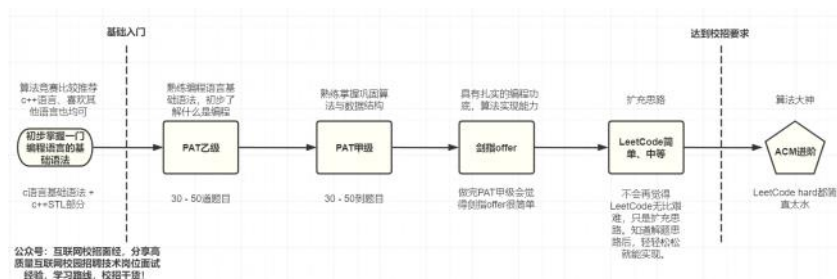
2021年10月31日 22:27

大一上 + 寒假:

c基础语法 + c++的STL => PAT乙级

啊哈算法+大话数据结构 => PAT甲级

大一下: 剑指offer+数据结构



# 浮点数的相等比较

2021年11月9日 18:55

如果要验证两个浮点数相等  
等价于  
两个数相减的得数是一个无穷小的值  
因为  
浮点数所储存的不一定与那个数相等



# 逻辑表达与循环判断

2021年11月10日 16:24

★ 只取最后一个做判断!!!

```
1 #include<iostream>
2 using namespace std;
3 int main()
4 {
5     for(int i=0;false,i<10;i++) cout<<'!'<<endl;
6     return 0;
7 }
```

D:\Code\DevC\C++\test.exe

注意坐姿  
29 : 50

Process exited after 0.1004 seconds with return value 0  
请按任意键继续. . .

# 字符、字符串

2021年10月29日 16:52

字符数组不是字符串 因为他不能用字符串的方式进行计算

字符串是以'\0'结束的 多占一位

0标志字符串结束但他不是字符串的一部分

字符串以数组的形式存在 以数组、指针访问

两个相邻的字符串之间如果没有符号 系统会把他们合并成一个

字符串是以数组形式存在的

通过数组的方式可以遍历字符串

字符串用指针还是数组??

- `char *str = "Hello";`
- `char word[] = "Hello";`
  - 数组：这个字符串在这里
    - 作为本地变量空间自动被回收
  - 指针：这个字符串不知道在哪里
    - 处理参数
    - 动态分配空间

构造字符串->用数组

处理字符串->用指针

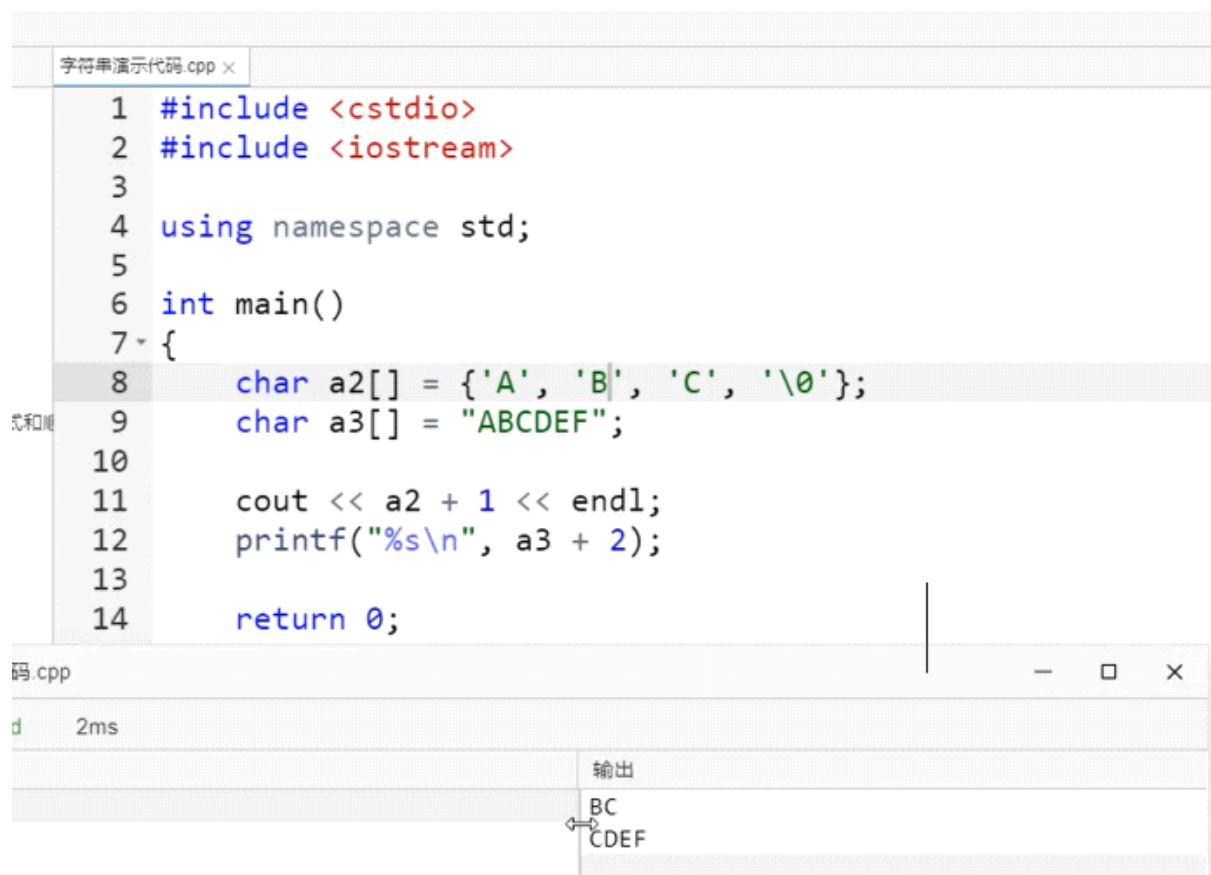
`scanf("%7s",&x);`

s是告诉scanf最多能读入7个数



## 空字符串

```
Char string a[100]="";  
a[0]='\\0'
```



The screenshot shows a C++ IDE with a file named "字符串演示代码.cpp". The code defines two character arrays: `a2` with elements {'A', 'B', 'C', '\\0'} and `a3` with the string "ABCDEF". It then prints `a2 + 1` using `cout` and `a3 + 2` using `printf`. The output window shows "BC" on the first line and "CDEF" on the second line, with a cursor pointing to the start of the second line.

```
1 #include <cstdio>  
2 #include <iostream>  
3  
4 using namespace std;  
5  
6 int main()  
7 {  
8     char a2[] = {'A', 'B', 'C', '\\0'};  
9     char a3[] = "ABCDEF";  
10  
11     cout << a2 + 1 << endl;  
12     printf("%s\\n", a3 + 2);  
13  
14     return 0;  
}
```

输出

BC

CDEF

A2+1 代表从a2[1]开始输出



# 字符串函数

2021年11月17日 18:40

Strcmp(s1,s2) 字典序比较 -1、0、1

strlen (s1) 不包含 '\0'

Strcpy(s1,s2) 把s2copy给s1

# 字符输入输出

2021年11月14日 13:03

## 输入

Cin //只适用于无空格

fgets (数组名, 数量, stdin) ; // (字符串) 能把你的回车读入进去 不安全

Getline(cin,数组名) ; // (string类)

Cin.getline (数组名, 数量) ;

scanf () 能读入字符串不能读入整数数组 string类不适用

## 输出

puts () ;

cout;

string类的printf

```
printf("%s\n", s1.c_str());
```

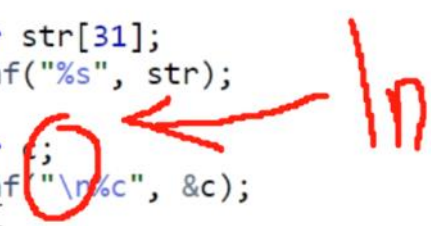
```
puts(s1.c_str());
```

Ps

当scanf读入一个char类型的数据时 不能过滤掉 上一个回车 只能把上一个回车读进来 而cin能过滤掉

挑战模式

```
1  #include <stdio>
2  #include <iostream>
3
4  using namespace std;
5
6  int main()
7  {
8      char str[31];
9      scanf("%s", str);
10
11      char c;
12      scanf("%c", &c);
13
14      for (int i = 0; str[i]; i++)
15          if (str[i] == 'c')
16      }
```



```

#include <iostream>
#include <string>

using namespace std;

int main()
{
    string s1;           // 默认初始化, s1是一个空字符串
    string s2 = s1;       // s2是s1的副本
    string s3 = "hiya";   // s3是该字符串字面值的副本
    string s4(10, 'c');   // s4的内容是 cccccccccc
    return 0;
}

```

(5) 为 string 对象赋值: ←

←

```

string s1(10, 'c'), s2;   // s1 的内容是 cccccccccc; s2 是一个空字符串←
s1 = s2;                 // 赋值: 用 s2 的副本替换 s1 的副本←
                          // 此时 s1 和 s2 都是空字符串←

```

←

(6) 两个 string 对象相加: ←

←

```

string s1 = "hello, ", s2 = "world\n";←
string s3 = s1 + s2;           // s3 的内容是 hello, world\n←
s1 += s2;                     // s1 = s1 + s2←

```

```

#include <cstdio>
#include <cstring>
#include <iostream>

using namespace std;

int main()
{
    string s = "Hello World!";
    for (char c: s) cout << c << endl;

    return 0;
}

```

遍历

这里的c是从s[i]复制过来的  
 要想改变s里面的值  
 就要在for中把c变为&c  
 这时c就是s[i];



.empty 检验是否为空字符串 如果是则返回0 不是则返回1

.size 和strlen效果一样

.back()是字符串最后一个位置

.pop\_back() 删掉最后一个字符

.substr(i,len) 返回从i开始 长度为len的字符 省略len就直接到最后停止 用于插入

string类型的两个字符串、字符 可以任意相加累加 相当于拼接  
但是+的两侧必须至少有一个string类型的字符串

```
for(int i=0;i<a.size();i++)  
{
```

```
    a=a.substr(1)+a[0];|
```

(用于把字符串移位 第一个移到最后一个)

Srting[100]字符串数组

## 错题 不会做的题目 双指针!!!

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n;
    cin>>n;
    while(n-->0)
    {
        char abs;
        int cnt=0;
        string str;
        cin>>str;
        for(int i=0;i<str.size();i++)
        {
            int j=i;
            while(j<str.size() && str[j]==str[i]) j++;
            if(j-i>cnt) cnt=j-i,abs=str[i];
            i=j-1;
        }
        cout<<abs<<" "<<cnt<<endl;
    }
    return 0;
}
```

连续最长出现的字符

```

1  #include<bits/stdc++.h>
2  #include<sstream>
3  using namespace std;
4  int main()
5  {
6      string s;
7      char bb[1000]; //XXX
8      getline(cin,s);
9      stringstream ssin(s); //把s读入 字符串流
10
11     int a,b;
12     string str;
13     double c;
14
15     ssin>>a>>b>>str>>c;
16     cout<<a<<endl<<b<<endl<<str<<endl<<c;
17
18     sscanf(bb,"%d%d%s%f",&a,&b,str,&c); //XXX
19     printf("%d\n%d\n%s\n%f",a,b,str,c); //XXX
20     return 0;
21 }

```

# 类、结构体、指针、引用

2021年11月22日 14:58

## 类的定义

```
class Person
{
    private:
        int age, height;
        double money;
        string books[100];

    public:
        string name;

        void say()
        {
            cout << "I'm " << name << endl;
        }

        int get_age()
        {
            return age;
        }

        void add_money(double x)
        {
            money += x;
        }
};
```

类中的变量和函数被统一称为类的成员变量。

private后面的内容是私有成员变量，在类的外部不能访问；

public后面的内容是公有成员变量，在类的外部可以访问。

但是private可以通过在外部访问public的关于private的函数间接访问

结构体和类的作用是一样的。不同点在于类默认是private，结构体默认是public。

## 间接访问

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  class test
4  {
5  private:
6      int age;
7  public:
8      void add_age(int a)
9      {
10         age=a;
11     }
12     int get_age()
13     {
14         return age;
15     }
16 };
17 int main()
18 {
19     test c;
20     int b;
21     cin>>b;
22     c.add_age(b);
23     cout<<c.get_age();
24     return 0;
25 }

```

**短路法**

## 84. 求 $1+2+\dots+n$

题目

提交记录

讨论

题解

视频讲解

求  $1 + 2 + \dots + n$ , 要求不能使用乘法、*for*、*while*、*if*、*else*、*switch*、*case* 等关键字及条件判断语句 ( $A ? B : C$ )。

样例

输入: 10

输出: 55

挑战模式

C++



```
1 class Solution {
2 public:
3     int getSum(int n) {
4         int res=n;
5         n>0 && ( res+=getSum(n-1));
6         return res;
7     }
8 };
```

结构体定义

数组所有元素的类型必须一致而结构体中的元素可以不一致

结构体的全局变量和局部变量

```
1 #include<stdio.h>
2 #include<string.h>
3 struct score{
4     char name[10];
5     int chinese;
6     int math;
7     int english;
8     int sum;
9 } student[1001]; //全局变量
10 int main()
11 {
12     //struct score student[1001]; //局部变量
```

结构体中构造函数

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  struct Person
4  {
5      int age,height;
6      double money;
7      //构造函数 不需要定义类型
8      Person();//不调用函数用
9      //法一
10     Person(int _age,int _height,double _money)
11     {
12         age=_age;
13         height=_height;
14         money=_money;
15     }
16     //法二
17     Person(int _age,int _height,double _money) :
18     age(_age),height(_height),money(_money) {}
19 };
20 int main()
21 {
22     //定义Person类型的p 并且调用函数对p进行赋值
23     Person p(18,180,100.0);
24     cout<<p.age<<' '<<p.height<<' '<<p.money;
25     return 0;
26 }

```

## 结构体内存对齐

[\(13条消息\) 结构体在内存中的对齐规则\\_咕唧咕唧shubo.lk的专栏 -CSDN博客](#)  
[结构体内存对齐规则](#)

原则一：结构体中元素是按照定义顺序一个一个放到内存中去的，但并不是紧密排列的。从结构体存储的首地址开始，每一个元素放置到内存中时，它都会认为内存是以它自己的大小来划分的，**因此元素放置的位置一定会在自己宽度的整数倍上开始（以结构体变量首地址为0计算）。**

**原则二：在经过第一原则分析后，检查计算出的存储单元是否为所有元素中最宽的元素的长度的整数倍，是，则结束；若不是，则补齐为它的整数倍。**

指针占用4个字节

>

# 指针

2021年10月28日 16:05

数组名只是一个地址

指针是一个左值

Int \*p[9] pointer array; it is a array every element store a pointer variate  
Int (\*p)[9] array pointer;

数组实际上就是指针  
Sizeof(a)==sizeof(int\*)

## 数组参数

函数原型四种是等价的

Int sum(int \*ar,int n);  
Int sum(int \*,int);  
Int sum(int ar[],int n);  
Int sum(int [],int);  
函数原型可以省略参数名

函数定义（不可省略参数名）

Int sum(int \*ar,int n)  
Int sum(int ar[],int n)

数组名就是数组第一个元素的指针地址

Int a[0]=1000;  
那么%d=\*a=1000

数组变量本身表达地址 不需要&

Int a[10]; int\*p=a;  
但是数组元素是变量 需要用&  
a=\*a[0];

[]and\*都可以对指针做也可以对数组

数组变量是const常量指针

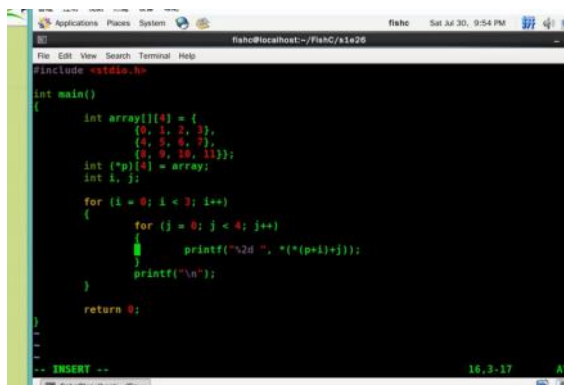
所以不能对数组赋值  
Int a[]<=>int \*const a

指向指针的指针

```
1 #include<stdio.h>
2 int main()
3 {
4     int num=5;
5     int *p=&num;
6     int **r=&p;
7
8     printf(" p=%p\n",p); // 指针p为num=5的地址
9     printf(" *p=%d\n",*p); // *p为指向num=5的指针
10    printf(" *r=%p\n",*r); // r 为指针p的地址、*r为指向指针p的内容储存的num=5的地址
11    printf("**r=%d\n",**r); // **r指向指针p p指向num=5
12    return 0;
13 }
```

两个指针相减=之间的元素个数+1;

指针数组与多维数组



## 引用



```

1  #include<bits/stdc++.h>
2  using namespace std;
3  int main()
4  {
5      int a=10;
6      int& b=a;//引用 起别名 共用地址
7      int* c=&a;
8      cout<<"a = "<<a<<endl;
9      cout<<"b = "<<b<<endl;
10     cout<<"c = "<<*c<<endl;
11     b+=5;
12     cout<<"(b+=5)a = "<<a<<endl;
13     cout<<"(b+=5)b = "<<b<<endl;
14     cout<<"(b+=5)c = "<<*c<<endl;
15     *c+=5;
16     cout<<"(*c+=5)a = "<<a<<endl;
17     cout<<"(*c+=5)ab = "<<b<<endl;
18     cout<<"(*c+=5)ac = "<<*c<<endl;
19     return 0;
20 }

```

终端 调试控制台 问题 输出

```

est )
a = 10
b = 10
c = 10
(b+=5)a = 15
(b+=5)b = 15
(b+=5)c = 15
(*c+=5)a = 20
(*c+=5)ab = 20
(*c+=5)ac = 20

```

# 指针与二维数组

2021年10月30日 20:20

```
Int array[3][5]
```

那么array代表指向第一行a[0]中五个元素的指针

```
* (array+1) ==array[1]
```

```
* (* (array+1) +3) ==a[1][3]
```

```
** (array+1)==a[1][0]
```

结论:

```
* (a+i) ==a[i];
```

```
*(*(a+i)+j)==a[i][j];
```

数组指针和二维数组

```
Int(*p)[3];
```

```
Int array[2][3]={0,1,2},{3,4,5};
```

```
** (p+1)=** (array+1)=array[1][0]=3
```

```
*(*(p+1)+2)=*(*(array+1)+2)=array[1][2]=5
```

void指针

void指针是通用指针 可以指向任意类型的数据

也就是说任意类型的指针都可以赋值给void指针

解引用void类型指针 需要强制转换数据 编译器不知道类型是什么

Null指针

```
Int *p=NULL;
```

当不明确指针指向什么的时候要初始化位NULL指针

在以后编写大型程序可以节省很多调试的时间。

不同类型的指针不能相互赋值

# 链表

2021年11月22日 20:11

```
#include <iostream>
#include <string.h>

using namespace std;

const int N = 10000;

struct Node
{
    int val;
    Node *next;
} *head;

int main()
{
    for (int i = 1; i <= 5; i ++ )
    {
        Node *p = new Node();
        p->val = i;
        p->next = head;
        head = p;
    }

    for (Node *p = head; p; p = p->next) cout << p->val << ' ';
    cout << endl;

    return 0;
}
```

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  struct Node
4  {
5      int val;
6      Node* next;
7
8      Node(int _val) : val(_val),next(NULL) {};;
9  };
10 int main()
11 {
12     Node a=Node(1);
13     cout<<a.val<<endl;//变量用.访问
14
15     Node* p=new Node(1);
16     cout<<p->val<<endl;//指针用->访问
17     return 0;
18 }
```

如果是变量 用.访问成员

如果是指针 用->访问成员

头节点：第一个节点的地址，不是值

[链表遍历](#)

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  struct Node
4  {
5      int val;
6      Node* next;
7
8      Node(int _val) : val(_val),next(NULL) {};
9  };
10 int main()
11 {
12     Node* p=new Node(1);
13     Node* q=new Node(2);
14     Node* l=new Node(3);
15
16     p->next=q;
17     q->next=l;
18     //头节点
19     Node* head=p;
20     //链表的遍历
21     for(Node*i=head;i!=NULL;i=i->next)
22         cout<<i->val<<endl;
23     return 0;
24 }

```

链表添加节点

头插法

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  struct Node
4  {
5      int val;
6      Node* next;
7
8      Node(int _val) : val(_val),next(NULL) {};
9  };
10 int main()
11 {
12     Node* p=new Node(1);
13     Node* q=new Node(2);
14     Node* l=new Node(3);
15
16     p->next=q;
17     q->next=l;
18
19     Node* head=p;
20     //头插法
21     Node* o=new Node(0);
22     o->next=head;
23     head=o;
24
25     for(Node* i=head;i;i=i->next)
26         cout<<i->val<<endl;
27     return 0;
28 }

```

### 链表节点删除

在遍历的时候把这个点跳过去

把要删除的点的上一个点的next指向要删除的下一个节点的地址

```
10  int main()
11  {
12      Node* p=new Node(1);
13      Node* q=new Node(2);
14      Node* l=new Node(3);
15
16      p->next=q;
17      q->next=l;
18
19      Node* head=p;
20      //节点删除
21      p->next=l;
22
23      for(Node* i=head;i;i=i->next)
24          cout<<i->val<<endl;
25      return 0;
26  }
```

# 位运算

2021年12月2日 15:12

& 与

有一个0即为0

| 或

有一个1即为1

~ 非

$\sim 1=0$   $\sim 0=1$

^ 异或

两个数一样为0 两个数不一样为1

>> 右移

<< 左移

原码  $x$

反码  $\sim x$

补码  $\sim x + 1$

常用操作

**求x二进制的从右数的第k位数字  $x \gg k - 1 \& 1$**

例如 我要求十进制数18的二进制数的最后一个数是多少

那么我就要写  $18 \gg 0 \& 1$ ;

**Lowbit( $x$ )= $x \& -x$ ; 返回x的最后一位1**

下面的1010 10 101000 1000均为二进制数

lowbit(1010)=10;

lowbit(101000)=1000;

$-x = \sim x + 1$

*lowbit( $x$ ): 返回x的最后一位1*

$$x \& -x = x \& (\sim x + 1)$$

$$x = \underline{1010 \dots} \underline{00 \dots 0}$$

$$\sim x = \dots$$

$$\sim x = \underline{0101\dots} \overset{\text{red}}{0} 1\dots 1$$

$$\sim x+1 = \underline{0101\dots} | 0\dots 0$$

$$x \& (\sim x+1) = 0\dots 0 | 0\dots 0$$



# 输出修饰

2021年12月5日 19:17

%02d 两个字符宽度 不足补0

# 读入技巧

2022年1月7日 16:19

出题人可能在读入处设坑 多加了空格  
在读入字符或字符串时统一用字符串  
比如 要读入一个字符表示一个命令  
我声明一个 `op[2]` 用来存储命令字符  
用的时候用`op[0]`;

# 重载运算结构体排序判断

2022年1月24日 1:22

```
struct Edge
{
    int a,b,w;
    bool operator< (const Edge &t)const
    {
        return w<t.w;//重载
    }
}edges[M];
```

```
sort(edges,edges+m);//按权值升序排序
```

平替：利用sort快排 + cmp函数

```
struct Stu
{
    int a,b,c;
}s[1000];

bool cmp(Stu x,Stu y)
{
    return x.c<y.c;
    //按照结构体中c的值进行升序排序
}
```

此方法中cmp一定要放在结构体的下面 一定注意顺序！！！！

# 取整函数

2022年2月4日 15:11

## 1、向上取整 `ceil (num)`

返回大于或等于num的最小整数

## 2、向下取整 `floor (num)`

返回小于或等于num的最大整数

## 3、`round(num)`

对num进行四舍五入

# vector

2021年11月20日 8:40

## 定义

```
#include <vector>
```

vector是变长数组，支持随机访问，不支持在任意位置O(1)插入。

为了保证效率，元素的增删一般应该在末尾进行。

## 声明

vector<int> a; 相当于一个长度动态变化的int数组

vector<int> b[233]; 相当于第一维长233，第二位长度动态变化的int数组

```
struct rec{...};
```

vector<rec> c; 自定义的结构体类型也可以保存在vector中

## 迭代器

迭代器就如同STL容器的“指针” 可以用\*解引用

一个保存int类型的迭代器的声明

```
vector<int>::iterator it;
```

vector的迭代器是“随机访问迭代器”

可以把vector的迭代器与一个整数相加减，其行为和指针的移动类似。

可以把vector的两个迭代器相减，其结果也和指针相减类似，得到两个迭代器对应下标之间的距离。

## 函数

**size()** 返回vector的实际长度（包含的元素的个数）

**empty()** 返回一个bool类型 表示是否为空

**clear()** 清空vector

**begin()** 返回指向vector中第一个元素的迭代器 \*a.begin()与a[0]作用相同

**end()** 返回vector的尾部的迭代器 即最后一个元素后边的边界

（所有容器都可以视为一个“前闭后开”的结构）

遍历vector<int>a 并输出每一个元素

```
int main()
{
    vector<int>a={1,2,3}; //vector的声明
    vector<int>::iterator it; //迭代器的声明
    //两种遍历输出方式
    for(it=a.begin();it!=a.end();it++) cout<<*it<<endl;
    for(int i=0;i<a.size();i++) cout<<a[i]<<endl;
    return 0;
}
```

**front** 返回vector的第一个元素

a.front()等价于 \*a.begin() 和 a[0]。

**back** 返回vector的最后一个元素

a.back()等价于 \*a.end() 和 a[a.size()-1]

**push\_back()**

a.push\_back(X) 把元素X插入到vector a 的尾部

**pop\_back()**

b.pop\_back() 删除vector a的最后一个元素

```

#include<bits/stdc++.h>
using namespace std;
struct Person
{
    string name;
    int age;
    Person(string mname,int mage)
    {
        name=mname;
        age=mage;
    }
};
int main()
{
    //创建一个Person类型的vector容器（指针）
    vector<Person*>p;
    //创建3个结构体数据
    Person p1("aaa",111);
    Person p2("bbb",222);
    Person p3("ccc",333);
    //添加vector数据
    p.push_back(&p1);
    p.push_back(&p2);
    p.push_back(&p3);
    //遍历输出
    for(vector<Person*>::iterator i=p.begin();i!=p.end();i++)
        cout<<(*i)->name<<' '<<(*i)->age<<endl;
    return 0;
}

```

# deque

2021年12月1日

20:44

## 定义

双端数组 可以对头段进行插入删除操作 与vector差不多

## 声明

```
#include<deque>
```

```
deque<int> d;
```

## 函数

**push\_front(elem)** 在头部插入一个数据

**pop\_front()** 在头部删除一个数据

**sort(d.begin(),d.end())** 排序

```
#include<bits/stdc++.h>
using namespace std;
void printDeque(const deque<int>& d)
{
    for (deque<int>::const_iterator it = d.begin(); it != d.end(); it++)
        cout << *it << " ";
    cout << endl;
}

int main()
{
    deque<int> d;
    d.push_back(10);
    d.push_back(20);
    d.push_front(100);
    d.push_front(200);
    printDeque(d);
    sort(d.begin(), d.end());
    printDeque(d);
    return 0;
}
```

Push\_back(elem);

Push\_front()

a.begin();

a.end();

a.front();

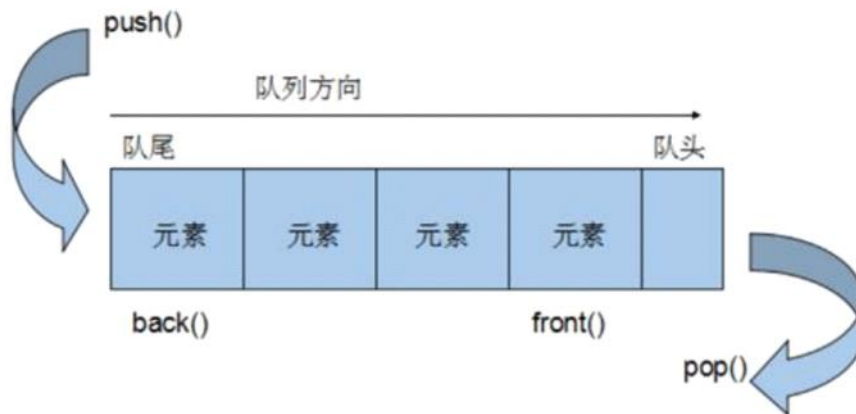
a.back();

# queue队列

2021年12月1日 22:41

## 定义

先进先出的数据结构



队列允许从一端新增元素，从另一端移除元素  
只有队头和队尾可以被外界访问 队列没有遍历

进数据-----入队push

出数据-----出队pop

## 构造函数

```
queue<int> que;
```

```
queue(const queue &que); 构造拷贝函数
```

## 函数

```
push(elem) 往队尾添加元素
```

```
pop()往队头移除第一个元素
```

```
back()返回最后一个元素
```

```
front()返回第一个元素
```

```
empty() 判断是否为空
```

```
size()返回长度
```

```
Que=queue<int> //清空队列
```

```
Priority_queue<int> a;//大根堆
```

```
a.push(1) ;//插入一个数
```

```
a.top();//取最大值
```

```
a.pop();//删除最大值
```



# 优先队列

2022年1月22日 14:28

头文件: `#include <queue>`

定义:

`priority_queue<Type, Container, Functional>`

Type: 数据类型

Container: 容器类型(用数组实现)

Functional: 比较方式: `greater`(升序)、`less`(降序)

# stack栈

2021年12月2日 9:01

## 定义

后进先出 竖置容器 队尾插入 队尾弹出 没有遍历

## 声明

```
#include<stack>
```

```
Stack<int> stk;
```

## 函数

push();//入栈

pop();//出栈

top();//返回栈顶元素

empty();//判断堆栈是否为空

Size();//返回栈的大小

# set/multiset

2021年12月2日 10:01

## 定义

所有元素插入时会自动排序

## 区别

set不允许有重复元素

multiset可以重复元素

set插入数据会同时返回插入的结果 表示是否插入成功

## 声明

```
#include<set>
```

```
Set<int> st;
```

## 函数

```
Insert(elem); //插入数据
```

```
clear();
```

```
size();
```

```
empty();
```

```
s1.swap(s2); //交换容器
```

```
erase(pos); //删除pos迭代器所指元素，返回下一个元素的迭代器
```

```
erase(begin,end); //删除区间内的所有元素，返回下一个元素的迭代器
```

```
erase(elem); //删除容器中值为elem的元素
```

```
find(key); //查找key是否存在 返回该元素的迭代器 否则返回set.end()
```

```
count(key); //统计key元素的个数
```

```
s.lower_bound(x) 查找大于等于x的元素中最小的一个，并返回指向该元素的迭代器。
```

```
s.upper_bound(x) 查找大于x的元素中最小的一个，并返回指向该元素的迭代器。
```

# pair对组

2021年12月2日 13:52

成对出现的数据 利用对组可以返回两个数据

创建方式

```
pair<int,int> p = make_pair(1,2);
```

```
cout<<p.first<<' '<<p.scond;
```

```
int main()
{
    pair<string,int> p=make_pair("zhouxinbin",18);
    cout<<p.first<<' '<<p.second;
    return 0;
}
```

pair在sort排序中 优先以first从小到大排序

# map/multimap

2021年12月2日 14:05

## 定义

map中所有的元素都是pair

pair中第一个元素为key（键值），第二个为value（实值）。

所有元素都会根据元素的key进行排序

## 本质

关联式容器 二叉树实现

## 优点

可以根据key快速找到value的值

## 区别

map不允许有重复的key

multimap可以有重复的key

## 构造

```
#include<map>
```

```
map<int,int>m;
```

```
Map<int,int>m2(m);//把m拷贝给m2
```

## 函数

```
Size();
```

```
Empty;
```

```
Swap();
```

```
Clear();
```

```
m.insert(make_pair(1,10));
```

```
Erase(pos);//删除pos迭代器所指的元素 返回下一个元素的迭代器
```

```
Erase(begin,end);//删除区间的元素， 返回下一个元素的迭代器
```

```
Erase(key);删除容器中值为key的元素
```

```
Find(key); 查找key是否存在 存在返回该元素的迭代器 否则返回set.end()
```

```
Count(key);统计key元素的个数
```

# reverse翻转

2021年12月2日 15:48

```
int main()
{
    vector<int> a={1,2,3,4,5};
    reverse(a.begin(),a.end());

    int b[]={1,2,3,4,5};
    reverse(b,b+5);

    for(int i=0;i<5;i++) cout<<a[i]<<' ';
    cout<<endl;
    for(int c:b) cout<<c<<' ';

    return 0;
}
```

# 去重

2021年12月2日 15:53

```
#include<bits/stdc++.h>
using namespace std;

int main()
{
    vector<int> a={1,2,2,4,5};
    reverse(a.begin(),a.end());

    int m=unique(a.begin(),a.end())-a.begin();
    cout<<m<<endl;
    for(int i=0;i<m;i++) cout<<a[i]<<' ';

    cout<<endl;

    a.erase(unique(a.begin(),a.end()),a.end());
    //相当于把unique后的放在前边 然后删除unique后边的
    for(int c:a) cout<<c<<' ';

    return 0;
}
```

unique 是把去重后的序列放在前边 然后把多余的放在后面 unique函数返回sh

unique函数返回重复的第一个元素的迭代器

也就是 去重后好的数列的下一个元素的迭代器

## C++ unique函数返回值

转自: [https://blog.csdn.net/aggressive\\_snail/article/details/51332659](https://blog.csdn.net/aggressive_snail/article/details/51332659)

std::unique

功能: 对有序的容器重新排列, 将第一次出现的元素从前往后排, 其他重复出现的元素依次排在后面

返回值: 返回迭代器, 迭代器指向的是重复元素的首地址

看不懂表达的看看下面这个简图:

有序的容器:

1	1	2	3	3	4	4	4	5	6
---	---	---	---	---	---	---	---	---	---

unique处理过的容器:

unique	unique	unique	unique	unique	unique	迭代器指向的地址			
1	2	3	4	5	6	1	3	4	4

(上表迭代器是指向倒数第四个数1)

# random\_shuffle 随机打乱

2021年12月2日 16:35

```
#include<bits/stdc++.h>
using namespace std;

int main()
{
    vector<int> a={1,2,3,4,5};

    srand(time(0));

    random_shuffle(a.begin(),a.end());

    for(int b:a) cout<<b<<' ';
    return 0;
}
```



# Sort

2021年12月2日 16:36

从小到大cmp排序

```
#include<bits/stdc++.h>
using namespace std;
bool cmp(int a,int b)
{
    return a<b;
}
int main()
{
    vector<int> a={1,2,3,4,5};

    srand(time(0));

    random_shuffle(a.begin(),a.end());

    for(int b:a) cout<<b<<' ';
    cout<<endl;

    sort(a.begin(),a.end(),cmp);
    for(int b:a) cout<<b<<' ';
    return 0;
}
```

# Lower\_bound/upper\_bound 二分

2021年12月2日 17:13

lower\_bound 的第三个参数传入一个元素x，  
在两个迭代器（指针）指定的部分上执行二分查找，  
返回指向第一个大于等于x的元素的位置的迭代器（指针）。

upper\_bound 的用法和lower\_bound大致相同，  
唯一的区别是查找第一个大于x的元素。  
当然，两个迭代器（指针）指定的部分应该是提前排好序的。

# permutation

2021年12月2日 20:59

```
class Solution {
public:
    vector<vector<int>> permutation(vector<int>& nums) {
        vector<vector<int>> s;
        sort(nums.begin(),nums.end());//先排序 由小到大
        do
        {
            s.push_back(nums); 把第一个由小到大的插到s的末尾
        }while(next_permutation(nums.begin(),nums.end()));
        //next_permutation函数把nums更新为nums按字典序的下一个
        //并且如果下一个有则返回true 否则返回false
        return s;
    }
};
```

# Upper\_bound

2021年12月13日 14:53

upper\_bound.cpp

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  int main()
4  {
5      int a[]={5,8,4};
6      int t = upper_bound(a,a+3,8)-a;
7      //两个指针相减 得下标
8      //括号里先是范围 再是要找的值
9      //upper_bound 返回地址
10     cout << a[t]; //输出的是8
11     return 0;
12 }
```