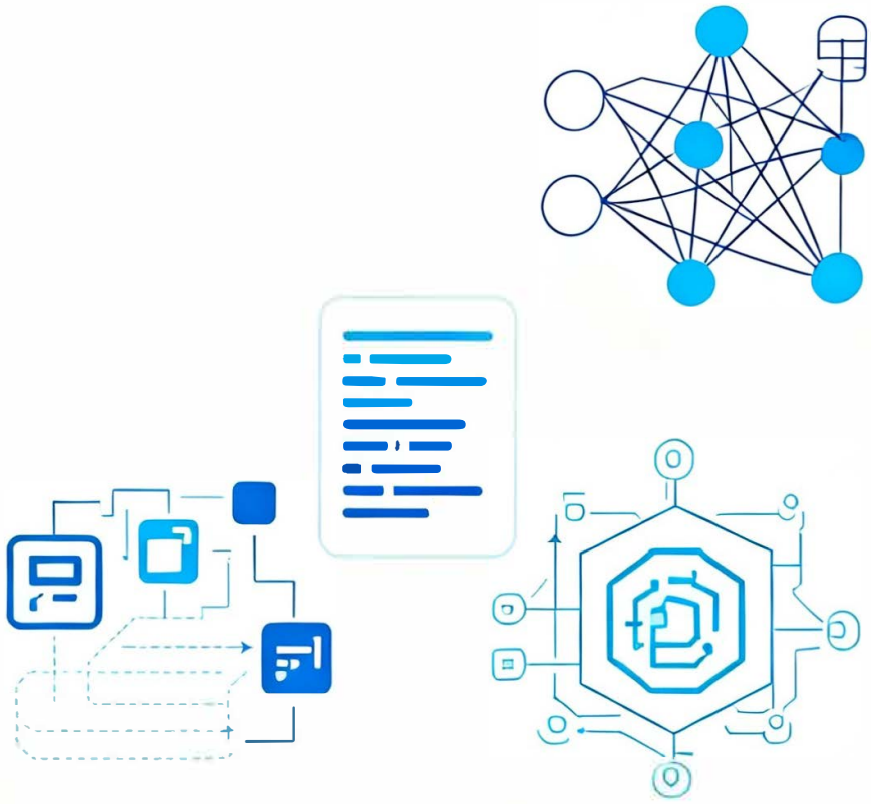
2025 级

程序设计辅导资料

AI学组 编著



及时更新知识，掌握编程动态

总结算法思路，理解代码逻辑

案例标准规范，提升实践能力

程序设计辅导资料目录

I 绪 论

1.1 计算机语言

低级语言	机器语言	一组二进制数组成的指令
	汇编语言	用缩写和助记符替代 0 和 1 的比特串
高级语言	过程化语言	BASIC、FORTRAN、PASCAL、C、C++、JAVA 等
	非过程化语言	面向问题

1.2 程序的执行过程

1. 编译：将源文件（.cpp）的程序翻译成目标程序（.obj）

2. 链接（link)：将目标程序与已有的其它目标程序连接起来，产生一个可执行的程序(.exe)

3. 加载（load)：为程序在内存中定位

1.3 基础知识

函数：表示一个或多个输入值对应唯一输出值的一种对应关系，可以包含多个语句变量：用来存储计算过程使用的值

```
#include <iostream> // 预编译指令，包含标准头文件
using namespace std; // 使用标准命名空间，不要忘了写
int main() // 主函数，程序入口
    {  //函数体，每一个语句结束用“;”
    int a = 5;//变量的定义和赋值
    int b = 3;
    int sum = a + b;
    cout << a << "+" << b << "=" << sum << endl;// 输出信息到屏幕
    return 0;// 返回值
    }
```

II 数 据

2.1 进制

C 语言中，八进制用“0”开头的整数表示

十六进制用“0x”或“0X”表示，“A~F”和“a~f”表示十进制的 10~15

2.2 数据的单位:一个 Byte 保存一个字符（ASCII 码），两个 Byte 保存一个汉字

说明	换算关系	读法
bit	最小单位{0,1}	比特、bit
Byte	1 Byte = 8 bit	字节、bit
KB	1 KB = 1024 B	千、kilobit
MB	1 MB = 1024 KB	兆、megabit
GB	1 GB = 1024 MB	吉、gigabit
TB	1 TB = 1024 GB	太、terabit
PB	1 PB = 1024 TB	拍、petabit
EB	1 EB = 1024 PB	艾、eksabit

2.3 数据类型

数据类型	基本类型	布尔值 bool
		短整型 short
		整型 int
		长整型 long (long)
		浮点型 float
		双精度浮点型 double
		long double
		字符型 char
		浮点型
	复合类型	数组
		结构体 struct
		枚举型 enum
		共用体 union
	指针类型	
	空值型	void

Ⅲ 表达式与分支控制结构

3.1 运算符和表达式

3.1.1 算术运算符和算数表达式

+，-，*，/，%（求余）

注意：

- 两个整数相除结果为整数 1/2=0，5/2=2
- 整数才可求余，余数的符号与左边数的符号相同。
- 括弧最先，先乘除，后加减，从左到右

3.1.2 赋值运算

名称	操作符	示例	等价于
简单赋值	=	y = x	
加赋值	+=	y += x	y = y+x
减赋值	-=	y -= x	y = y-x
乘赋值	*=	y *= x	y = y*x
除赋值	/=	y /= x	y = y/x
模赋值	%=	y %= x	y = y%x

注意：“=”的结合性为自右向左

3.1.3 关系运算符与关系表达式

（假设y=10, x=5）

名称	操作符	示例	结果
大于	>	y > x	1
等于	==	y == x	0
小于	<	y < x	0
大于等于	>=	y >= x	1
小于等于	<=	y <= x	0
不等于	!=	y != x	1

关系表达式结果返回1：表明该关系成立

关系表达式结果返回0：表明该关系不成立

注意：区分 == 和 =

3.1.4 逻辑运算符与逻辑表达式

与 &&，或 ||，非!

逻辑表达式：作为条件，所有非 0 值均为真；作为结果，只有 0 或 1 两种。

3.1.5 自增自减运算符

- i++：在使用 i 之后，使 i 的值加 1。即先使用，后加。（等价于 i=i+1）i--：在使用 i 之后，使 i 的值减 1。即先使用，后减。
- ++i：在使用 i 之前，先使 i 的值加 1。即先加，后使用。--i：在使用 i 之前，先使 i 的值减 1。即先减，后使用。

3.1.6 逗号运算符与逗号表达式

- 表达式 1，表达式 2，表达式 3，…，表达式 n
- 顺序求解，结果为最后一个表达式的值，并且优先级最低。

3.1.7 运算符的优先级

- ( ) [ ] -> .
- ! ~ ++ -- + - * & sizeof （类型）
- * / %
- << >>
- < <= > >=
- == !=
- &
- ^
- |
- &&
- ||
- ?:
- = += -= *= /= %= &= ^= |= <<= >>=
- ,

表：几种运算符的优先级

！（逻辑非）	（高）
算术运算符	
关系运算符	
&& 和||	
赋值运算符	（低）

3.2 程序控制结构

3.2.1 顺序结构

- 由一组顺序执行的处理块组成，每个程序块可能包含一条或一组语句，完成一项任务
- 程序块：一对花括号括住的若干条语句构成一个程序块，语法上等于单条语句，其后不需要加分号

3.2.2 选择结构

1. if 语句

```
if(表达式)
      程序块1
else
      程序块2
```

```
if(表达式)
      程序块1
```

```
if(表达式1)
      程序块1
else if(表达式2)
      程序块2
else
      程序块3
```

(1) if

```
if(x>y)
     cout<<x;
```

(2)if-else

```
if(x>y)
 {
    z=x;
    cout<<z;
 }
 elSe
 {
    z=y;
    cout<<z;
 }
```

（3）if-else if-else

```
if(score>=90)
     level='A';
else if(score>=80)
     level='B';
else if(score>=60)
     level='C';
else
     level='D;
```

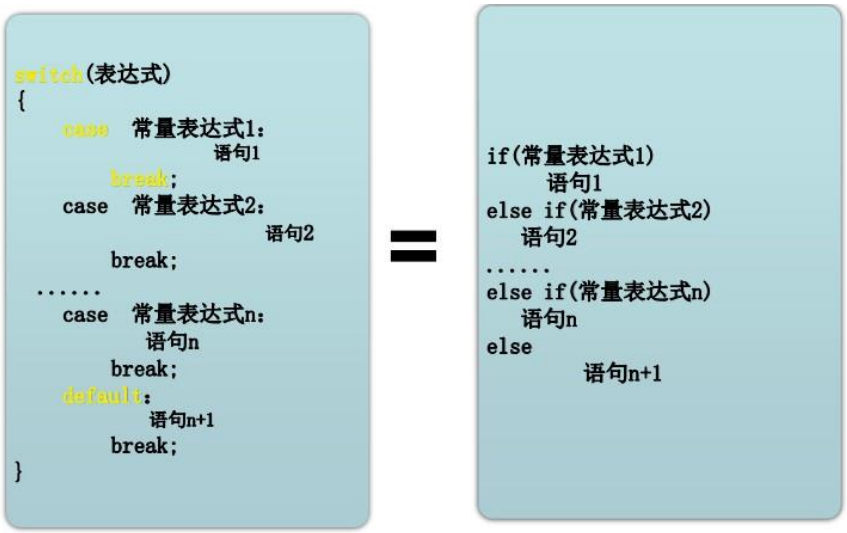
注意：1.else 总是寻找在它前面最近的 if 配对，建议用 {} 确定配对关系

2. 条件运算符（类分支运算符）

表达式 1? 表达式 2：表达式 3


```
max=a>b?a:b;    //求 a，b 中的大者
```

2.switch 语句



当表达式值为常量表达式 1 时，执行语句 1;

当表达式值为常量表达式 2 时，执行语句 2;

.....

当表达式值为常量表达式 n 时，执行语句 n;

否则，执行语句 n+1

```
#include<iostream>
using namespace std;

int main()
{
    int score,level;
    cin>>score;
    level=score/10;
    switch(level)
    {
        case 10:
        case9:
            cout<<"A\n";
            break;
        case 8:
            cout<<"B\n";
            break;
        case 7:
            cout<<"C\n";
            break;
        case 6:
            cout<<"D\n";
            break;
        default:
            cout<<"E\n";
            break;
    }
}
```

注意：1.case 后面跟常量表达式。

2. 若没有 break，则后面的 case 会继续执行。

IV 循环控制结构

4.1 循环语句简介

计算机的强项是不厌其烦地做同样的操作，这是通过循环语句实现的。例如，计算 100! = 100 × 99 × ⋯ × 1，这是通过循环语句实现的。

循环结构就是根据某一条件的判断结果，反复执行某一程序块的过程。实现过程为：

1. 进入循环结构，判断循环条件；
2. 如果循环条件的结果为真，则执行 A 程序块的操作，即循环一次；
3. 再次判断循环条件；
4. 当循环条件为假时，循环结束。

在程序中，循环语句通常包括三个主要部分：

- 表达式 1：循环变量赋初值；
- 表达式 2：循环条件；
- 表达式 3：循环变量增值。

在循环体中，如果有一个以上的语句，须用括起。

循环体内或表达式中必须有使循环结束的条件，即一定要有一个循环变量。

实现循环结构的语句主要包括 for 语句、while 语句和 do-while 语句。

4.2 for 循环

for 循环的代码结构与 1～100 求和程序如下所示：

```
for (表达式 1; 表达式 2, 表达式 3)
{
    程序块
}
```

代码格式

```
int sum = 0;
for (int i = 1; i <= 100; i++) {
sum += i;
}
```

1 - 100 求和

4.2.1 循环语句的流程

- 首先执行表达式 1，初始化循环变量；
- 然后判断循环条件（表达式 2），如果条件为真，则执行循环体；
- 执行完循环体后，执行表达式 3，更新循环变量，返回判断条件；
- 如果条件为假，则退出循环。

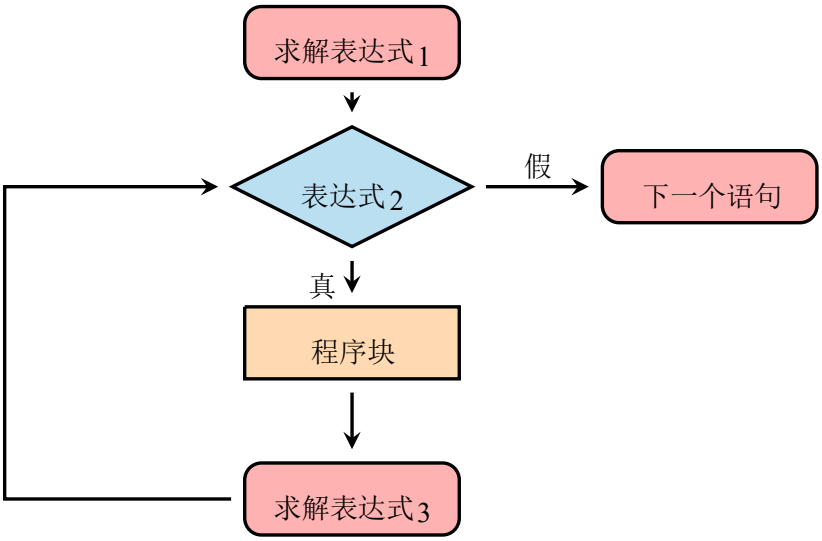


图 1: for 循环流程图

4.2.2 for 语句的注意事项

- 如果省略表达式 1，循环变量必须在循环体之前初始化；
- 如果省略表达式 2，循环会变为无限循环，必须在循环体内设置条件来结束；
- 如果省略表达式 3，循环变量的更新必须在循环体内完成。

```
for(;;)
{
    程序块
}
```

(1) for 的所有表达式均可省略，但要防止进入死循环

```
i=1;
for( ; i<=10; i++)
{
    sum += i;
}
```

(2) 省略表达式 1

```
for(i=1;;i++)
{
    sum += i;
    if(...) 程序块
}
```

(3) 省略表达式 2，循环无法停止，必须在循环内设置停止条件

```
for(i=1; i<=100;)
{
    sum+=i;
    i++;
}
```

(4) 省略表达式 3，应确保循环能正常结束

```
for(sum=0, i=1; i<=100; i++)
{
    sum += i;
}
```

(5) 表达式 1 和 3 可以是逗号表达式，包含一个以上的简单的表达式，之间用逗号隔开，建议少用这种方式

```
for(i=1; i<=100; i++)
{
    sum += i;
}
```

(6) 表达式 2 一般是关系表达式 (i<=100) 或者逻辑表达式（a<b && x<y）

4.

4.2.3 循环嵌套

一个循环内又包含另一个完整的循环结构，称为循环嵌套。内循环必须完全嵌套在外循环内，内外循环不能交叉，且内外循环的循环变量不能重名。

```
for （表达式 1; 表达式 2; 表达式 3）
{
    语句块 1;
    for （表达式 4; 表达式 5; 表达式 6）
    {
        语句块 2;
    }
    语句块 3;
}
```

下面是用循环嵌套实现打印九九乘法表的程序示例：

```
#include <iostream>
int main (int argc, const char * argv[])
{
    using namespace std;
    int i, j;
    for(i=1; i<10; i++)
     {  for(j=1;j<10;j++)
         {   cout<<i<<"*"<<j<<"="<<i*j<<"\t";
          }
         cout<<endl;
     }
     return 0;
}
```

输出示例：

```
1*1=1 1*2=2  1*3=3  1*4=4  1*5=5  1*6=6   1*7=7  1*8=8  1*9=9

2*1=2 2*2=4  2*3=6  2*4=8  2*5=10 2*6=12 2*7=14 2*8=16 2*9=18

3*1=3 3*2=6  3*3=9  3*4=12 3*5=15 3*6=18 3*7=21 3*8=24 3*9=27

4*1=4 4*2=8  4*3=12 4*4=16 4*5=20 4*6=24 4*7=28 4*8=32 4*9=36

5*1=5 5*2=10 5*3=15 5*4=20 5*5=25 5*6=30 5*7=35 5*8=40 5*9=45

6*1=6 6*2=12 6*3=18 6*4=24 6*5=30 6*6=36 6*7=42 6*8=48 6*9=54

7*1=7 7*2=14 7*3=21 7*4=28 7*5=35 7*6=42 7*7=49 7*8=56 7*9=63

8*1=8 8*2=16 8*3=24 8*4=32 8*5=40 8*6=48 8*7=56 8*8=64 8*9=72

9*1=9 9*2=18 9*3=27 9*4=36 9*5=45 9*6=54 9*7=63 9*8=72 9*9=81
```

4.2.4 for 循环程序设计事例

某日，某人骑车出交大，在南门口被一小车撞倒，小车撞人后逃逸。现场有 A、B、C 三同学目击此事，他们向警察描述了车号的一些特征 A 说：牌照的前两位数字是相同的；

B 说：牌照的后两位数字是相同的，但与前两位不同；

C 说：四位的车号刚好是一个整数的平方；请根据以上线索编程找出车号。

问题分析

这个题目就是需要找出一个四位数，满足以下要求：

A：前两位数相同；

B：后两位数同, 且与前两位不同；

C：该整数是另一个整数的平方；

1. 第一种方法：（枚举法）

```c
#include <stdio.h>
#include <math.h>
int main()
{
    int i, k;
    int number=0; // 结果
    int n1, n2, n3, n4; // 千位，百位，十位，个位
    int flag=0; // 条件成立标志
    int count=0; // 记录循环次数
    for(i=1; i<=9999; i++)
    {
        number = i;
        n1 = (number/1000);
```

```
        n2 = (number/100)%10;
        n3 = (number/10)%10;
        n4 = number%10;
        flag = (n1 == n2) && (n3 == n4) && (n1 != n3);
        if(flag)
        {
           for(k=1; k*k<=number; k++)
           {   if(k*k == number)
               {
                  printf("车牌号是 %d. \n", number);
               }
               count++;
            }
        }
    }
    printf("循环次数: %d. \n", count); // 打印循环次数 return 0;
}
```

2.第二种方法:

```
#include
stdio.h>
#include
<math.h>      int
main(){
     int i, j, k;
    int number=0;
    int count=0;
    for(i=1; i<=9; i++) { // i: 车牌前两位取值
         for(j=1; j<=9; j++) { // j: 车牌后两位取值
            if(i != j) { // 判断两位数字是否不同
                number = i*1000 + i*100 + j*10 + j; // 计算出可能的整数
               for(k=1; k*k<=number; k++) { // 判断该数是否是另一个整数的平方
                     if(k*k == number) { printf("车牌号是 %d. \n", number);
                    }
                    count++
                    ;
                }
            }
        }
    }
    printf("循环次数: %d. \n", count); // 打印循环次数
    return 0;
}
```

3.第三种方法：（循环 **69** 次）

```
#include<stdio.h>
int main() {
int num;
    int a, b, c, d; // 千位，百位，十位，个位
   int count=0; // 循环次数
   int i;
   for(i=32; i<100; i++) { // 四位数
        num = i*i;
        a = num/1000;
        b = (num - a*1000)/100;
        c = (num - a*1000 - b*100)/10;
        d = num%10;
        if(a == b && c == d && a != c)
            printf("车牌号是 %d. \n", num);
        count++;
    }
    printf("循环次数 = %d\n", count);
    return 0;
}
```

4.第四种方法：（循环 **9** 次）

```
#include <stdio.h>
/* 交通违章的最快方法 */
int main()
{
    int i, k;
    int number;
    int n1, n2, n3, n4;
    int flag=0; // 记录循环次数
    int count=0;
    for(i=1; i<=9; i++)
    {
        k=i*11; // 车牌号的平方根为 11 的倍数
        number=k*k; // 计算车牌号
        n1 = (number/1000); // 千位
        n2 = (number/100)%10; // 百位
        n3 = (number/10)%10; // 十位
        n4 = number%10; // 个位
        flag = (n1 == n2) && (n3 == n4) && (n1 !=n3);
        if(flag)
        {
            printf("车牌号是 %d. \n", number); // 打印结果
        }
        count++;
    }
    printf("循环次数:%d\n", count); // 打印循环次数
    return 0;
}
```

4.3 while 语句

while 语句适用于需要重复执行某个操作，直到满足某个条件为止的情况。
一般情况下，选 for 循环还是 while 循环都是可以的，取决于个人喜好。

4.3.1 while 循环流程

- 首先判断循环条件，如果条件为真，则执行循环体；
- 如果条件为假，则跳出循环。

4.3.2 while 循环格式

```
while(表达式)
{
    程序块1;
}
```

(1) while 循环示例

```
int i=0;
int sum=0;
while(i<=100)   // while循环示例
{
    sum += i;   // 累加
    i++;   // 递增
}
```

(2) while 循环实现求和

在 sum 求和中，循环次数、sum 值和 i 值之间的关系如下：

循环条件	初值	真	真	真	真	真	真	真	真	假
循环次数		1	2	3	4	. . .	99	100	101	
sum	0	1	3	6	10	. . .	4950	5050		
i	1	2	3	4	5	. . .	100	101		

表 1: while 循环执行过程示意表

4.3.3 while 循环程序设计示例

1. 从键盘连续输入字符，直到输入“回车”符为止，统计输入的字符个数

```
#include <iostream>
using namespace std;
int main ()
{
    char c;
    int cnt;
    cout<<"Please enter your sentence:\n";
    cin.get(c); // 取第一个字符
    cnt=1;          // 计数初始值
     while (c!='\n')  // 循环结束条件
     {
          cnt++;
          cin.get(c); // 取下一个字符，把'\n'统计在内
     }
     cout<<"There are "<<cnt<<" characters in this sentence.\n";
     return 0;
}
```

输入输出示例：

```
Please enter your sentence:
we are in class
There are 15 characters in this sentence.
```

2. 输入 n 个数，输出这 n 个数的和

```
#include <iostream>
using namespace std;
int main ()
{
    int n, sum = 0, i = 1, no;
    cout<<"Please input n:\n";
    cin>>n;
    cout<<"Please input the numbers:\n";
    while (i<=n)
    {
        cin>>no;
        sum+=no;
        i++;
      }
      cout<<"The sumis”<<sum<<endl;
      return 0;}
```

输入输出示例：

```
Please input n: 5
Please input the numbers:
33
11
2
9
11
The sum is 66
```

4.3.4 while 循环找不同

以下语句，循环退出时 x 为多少？

(1) x = 10; while (x != 0) x- -;

(2) x = 10; while (x) x- -;

(3) x = 10; while (x- -) ;

(4) x = 10; while (- -x);

解析：x 的初始值都为 10。

(1). 当 x 不等于 0 时，进入 while 循环。每次循环时，x 会减 1。当 x 等于 0 时，不满足循环条件，退出循环。最终 x = 0。

(2). while (x) 会在 x 为真，即 x 不等于 0 时执行。每次循环时，x 会减 1。直到 x 等于 0。当 x 等于 0 时，不满足循环条件，退出循环。最终 x = 0。

(3). 这个语句在 while (x–) 中使用了后缀递减运算符（x–）。它表示先使用 x 的值，然后再减少 x。循环会在 x 不为零时执行，每次循环结束后 x 会减 1。在最后一次循环时 x 减到 0，循环结束，此时 x =

0。

(4). 这个语句在 while (–x) 中使用了前缀递减运算符（–x）。它表示先减少 x 的值，再执行循环。在每次进入 while (–x) 循环前，x 会先减 1。当 x 减到 0 后，循环会结束，但因为是前缀递减，所以最后 x 的值为 -1。

答案：(1) x = 0 ; (2) x = 0; (3) x = 0; (4) x = -1

4.4 do-while 语句

同样的，在一般情况下，选 for 循环还是 while 循环，亦或是 do - while 循环都是可以的，取决于个人喜好。

4.4.1 do-while 循环流程

- 首先执行循环体一次；

- 然后判断循环条件，如果条件为真，则继续执行循环体；
- 如果条件为假，则跳出循环。

4.4.2 do-while 循环格式

```
do
{
    程序块 1;
}while(表达式);
```

(1)do-while 循环示例

```
int i=0;
int sum=0;
do{
    sum += i; // 累加 i++;
    // 递增
}while(i<=100); // do-while 循环示例
```

（2）do-while 循环实现求和

4.4.3 for 循环和 do -while 循环的区别

- do-while 语句是在判断条件是否成立之前, 先执行循环体语句一次；
- while 语句则是先判断条件是否成立, 如果条件成立才执行循环体;

因此, while 语句的循环体可能一次都不执行；而 do-while 语句的循环体至少被执行一次, 这是 while 语句和 do-while 语句的根本区别。

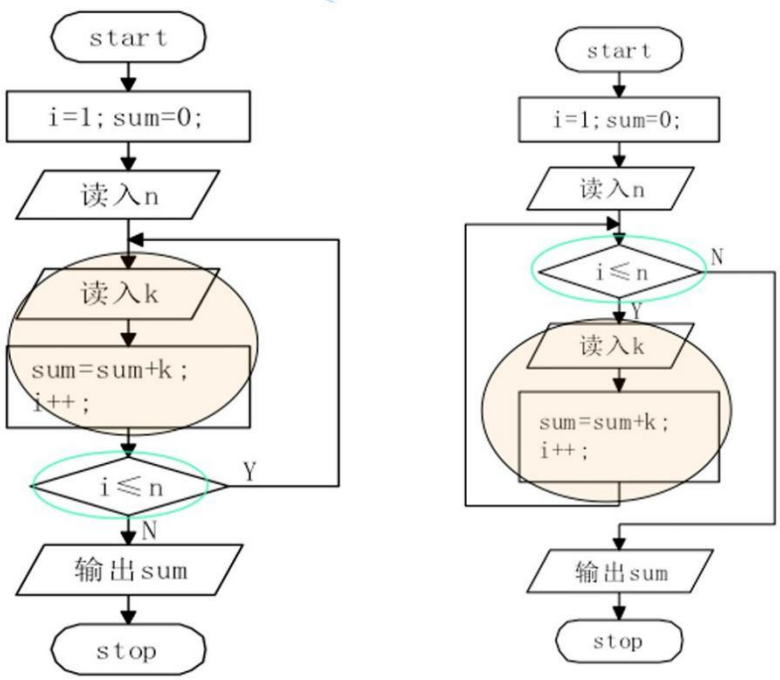


图 2: while、do-while 流程图

4.4.4 do-while 循环程序设计示例

1. 从屏幕输入 n 个数，输出这 n 个数的和

```
#include <iostream>
using namespace std;
int main ()
{
    int n, sum = 0, i = 1, no; cout<<"Please input n:\n";
    cin>>n;
    cout<<"Please input the umbers:\n";
    do{
         cin >> no; sum
         += no;
         i++;
      }while(i <= n);
    cout<<"The sum is"<<sum<<endl;
    return 0;
}
```

输入示例：

```
Please input n: 5
Please input the numbers:
33
11
2 9
11
The sum is 66
```

2.从键盘连续输入字符，直到输入“回车”符为止，统计输入的字符个数；使用 do — while 完成。

4.5 break 和 continue 语句

‘break‘语句用于立即退出当前循环。

‘continue‘语句用于跳过当前循环体的剩余部分，直接进入下一次循环。

break 与 continue 的区别演示程序：

```
#include <iostream>

using namespace std;

int main ()

{

    for (int i = 0; i < 3; i++)

     {  for (int j = 0; j < 3; j++)

          {

              cout<<"i = "<<i<<"\tj ="<<j<<endl;

              continue;

              //break;

          }

     }

     return0;

}
```

输出示例（continue）：

```
i = 0    j = 0

i = 0    j = 1

i = 0    j = 2

i = 1    j = 0

i = 1    j = 1

i = 1    j = 2

i = 2    j = 0

i = 2    j = 1

i = 2    j = 2
```

输出示例（break）：

```
i = 0    j = 0

i = 1    j = 0

i = 2    j = 0
```

由上面的输出结果，我们可以看出 break 和 continue 的区别：

- continue 语句只结束本次循环，而不是中止整个循环的执行。
- break 语句则是结束循环，不再进行条件判断。

4.6 goto 语句

使用 goto 可以让程序清晰，但是一定要谨慎使用！

配合 goto 语句使用的标号为 error，用于配合 goto 语句跳转至特定位置的程序。同变量、函数的命名规则一样，error 后面需要加上一个冒号，一般顶格书写.

使用 goto 语句时，应该遵循以下原则：

- 使用之后，程序仍然是单入口、单出口。
- 不要使用一个以上的标号。
- 不要用 goto 往回走，要往下走。
- 不要让 goto 制造出永远不会被执行的代码。

```
for (...) for
    (...)
    { ...

        if （出现错误条件）
            goto error;
    }
error:
    错误处理语句
```

(1) 使用 goto 跳转实现退出嵌套循环

```
int flag =0;
for (...)
     for (...)
    { ...
        if （出现错误条件）
        {
        flag = 1;
        break;
        }
}
if (flag)
    break;
    错误处理语句
```

（2）使用 flag 和 break 来实现嵌套循环退出

4.7 课后练习

1.编程实现以下函数的求解，要求从屏幕输入 x，然后输出 y：

- $y = 3x + 1$, 当 $x > 0$
- $y = 0$, 当 $x = 0$
- $y = x^2 - 1$, 当 $x < 0$

2.打印 10! 的值。

3.打印 $1! + 2! + \cdots + 10!$ 的值。

V 函数与数组

5.1 函数

5.1.1 函数概念引入

某同学去某公司工作：第一个月和第二个月老板各给 100，第三个月开始，每月的工资是前两个月工资的和。

该同学工作一年后的当月工资是多少？一年来总共收入多少？（如果二十年呢？）这个问题，实质上是在求解斐波那契数列。斐波那契数列指的是这样一个数列：

0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,1597,2584,4181,6765,10946,17711,28657,46368

我们用 $F(N)$ 表示斐波那契数列的第 N 项。容易发现，斐波那契数列满足这样一个递推式：

$$F(N) = F(N-1) + F(N-2)$$

其中，$F(1) = F(2) = 100, 3 <= N$

这个问题如果单纯用之前的知识很难解决，但用函数方法是很好解决的。

5.1.2 函数的作用

1. 分解问题，使程序结构更加清晰。

在解决复杂问题时，先将问题划分或化解成一些子问题分别加以解决，从而降低问题的复杂度。在程序实现时可针对这些子问题分别编程实现和测试。（分而治之）。

2. 避免重复劳动。

在解决不同的实际问题时，常常需要用到许多相同的子方法或子技术。它们对于不同问题可能只是选择参数的不同，而内部功能完全相同。如果能将这些子方法或子技术形成一个经过测试的优化的固定程序模块，让用户在解决问题时直接调用，就会比较方便。

5.1.3 函数三要素

学习知识要会“抓纲”，学习函数就要抓住函数的三要素——函数的声明、定义与调用。

5.1.3.1 函数的声明

```
返回值类型 函数名(类型 参数 1, 类型 参数 2, ……);
```

函数声明格式

```
Int CalRabbitsNum(int vMonth);
int RabbitsAmount(int month);
int max(int x, int y);
void printf_message();
```

函数声明示例

函数声明时，需要注意，在使用某个函数之前，必须知道函数在哪里，即编译器需要看到该函数的声明或定义。有时，一些函数和主函数在一个文件里，有时在单独的文件里。

5.1.3.2 函数的定义

以下是一个返回最大值的函数的例子以及表示形式：

```
类型 函数名 (类型 参数1, 类型 参数2,
    ……)
{
    ......
    ......
    ......
    return 表达式;
}
```

函数定义格式

```
int max(int x, int y)
{
    int max;
    if(x > y)
        max = x;
    else
        max = y;
    return max;
}
```

函数定义示例

函数一般有三种定义形式：有参函数、无参函数和空函数。

1.有参函数

```
类型 函数名(类型 参数1, 类型 参数2, ……)
{
    ......
    return 表达式;
}
```

2.无参函数

```
类型 函数名()
{
    ......
    return 表达式;
}
```

3.空函数

```
void 函数名()
{
    ......
}
```

以下是 max 函数的使用示例：

```
#include <iostream>
using namespace std;
int max(int x, int y){
    int result;  // 变量名改为 result，避免与函数名冲突
    if(x > y)
        result = x;
    else
        result = y;
    return result;
}
int main(){
    int a, b;
    cin >> a >> b;
    cout << max(a, b) << endl;
    return 0;
}
```

return 语句将函数计算结果返回。当函数执行到 return 语句或}时，函数的运算停止。程序从当此调用函数的地方继续执行。

实际上，函数可以有多个 return 语句，如下所示，

两种方式都是正确的：

```
int max(int x, int y){
    if(x>y) returnx;
    else return y;
}
```

函数定义格式

```
int max(int x, int y)
{
    int max;
    if(x > y) max =x;
    else max = y; return
    max;
}
```

函数定义示例

有时某些函数没有返回值，调用过程的输出已经在被调函数体内完成，这时要用 void 定义这些“无返回值”类型的函数。此时 return 语句可以省略。

void 函数示例：

```
void print_star() {
   cout<<"*******\n";
}
void print_message(){
   cout<<"how are u?\n";}
```

在下面的代码中，调用函数时，将 a 的值赋给 x，将 b 的值赋给 y，最后在函数中用 x 和 y 计算出 z，最后函数将 z 的值传回给 c。

```
/* int 是函数类型, max 是函数名*/
int max(int x, int y) {
    int z;
    z = (x > y) ? x : y;
    return z; /* 函数值 */
}/* 函数体 */

int main(void) { /* 主调函数 */
    int a, b, c;
    cin >> a >> b;
    c = max(a, b); /* 调用函数 */
    cout << "The max is: " << c << endl;
}
```

5.1.3.3 函数的调用

函数调用要满足以下形式：

函数名(参数 1,参数 2,.....)

在调用一个函数之前，先要对其返回值类型、函数名和参数进行声明或定义。

调用函数时，提供的参数（实际参数）和该函数定义中的参数（形式参数）必须在下列方面匹配：

- 数目一致
- 类型一一对应（会发生自动类型转换）
- 把实际参数的值复制给形式参数（指针不同）

返回值可以按需处理。

可以通过以下方式调用函数：

- 作为语句：

$$printstar();$$

- 作为表达式：

$$c = max(a,b);$$

- 作为另一个函数的参数：

$$cout << max(a,b);$$

在一个函数中调用另一函数（即被调用函数）需要具备以下条件：

- 被调用的函数必须是已存在的函数
- 如使用库函数，必须预先声明，如

#include < cmath >

- 函数调用遵循先定义、后调用的原则，即被调函数应出现在主调函数之前。下面是一个使用函数的例子：

```
float max(float x, floaty)
{
    float z;
    z = (x>y) ? x : y ; return
    z;
}

void main (void)
{
    float   a,b, c;
    cin>>a>>b;
    c=max (a+b , a*b) ;
    cout<<"The max is"<<c<<endl;
}
```

5.1.4 函数参数传递

函数参数就是一个变量，它随着使用者的不同而发生变化。函数参数包括形参和实参。

形参：函数定义中的形式参数。

实参：在函数调用时，传递给函数的参数，也就是参数的实际值。

在定义函数中指定的形参变量，在未出现函数调用时，它们并不占内存单元。只有在发生函数调用时函数中的参数才被分配内存单元。在调用结束后，形参所占的内存单元才被释放。

在被定义的函数中，必须指定形参的类型。

实参和形参类型应一致，实参可以是整型、实型、字符型或表达式，但要求它们有确定的值。

这个函数是不能实现交换功能的：

```
#include <iostream>
using namespace std;
void Swap(int x,inty);
int main()
{
   int a=123;
   int b=456;
    cout<<"a = "<<a<<"; b="<<b<<endl;

    Swap(a,b);  //交换 a,b

    cout<<"a = "<<a<<"; b="<<b<<endl;
    return 0;

}
void Swap(int x,int y)
{
    int tmp;
    tmp = x;
    x = y;
    y = tmp;
}
```

这个函数之所以实现不了交换 *a* 和 *b* 的功能，是因为在 *Swap* 函数中，*x* 和 *y* 是 *a* 和 *b* 的副本。任何对 *x* 和 *y* 的修改都不会反映到 *main* 函数中的 *a* 和 *b* 上。也就是说，在函数调用时，给函数传递的是变量的值，而非变量本身。下面的程序就证明了这一点：

```
#include <iostream>
using namespace std;
void Swap(int x, int y);
int main()
{
   int a=123;
   int b=456;
   Swap(a,b); cout<<"Var\tAddress\t\tValue\n";
   cout<<"--------before swap--\n";
   cout<<"a\t"<<&a<<"\t"<<a<<endl;
   cout<<"b\t"<<&b<<"\t"<<b<<endl;
   cout<<"---------after swap---\n";
   cout<<"a\t"<<&a<<"\t"<<a<<endl;
   cout<<"b\t"<<&b<<"\t"<<b<<endl;
   return 0;
}
void Swap(int x, int y)
{
    int tmp;
    cout<<"---------before swap-------------------\n";
    cout<<"x\t"<<&x<<"\t"<<x<<endl;
    cout<<"y\t"<<&y<<"\t"<<y<<endl;
    tmp = x;
    x = y;
    y= tmp;


    cout<<"---------after swap-------------------\n";
    cout<<"x\t"<<&x<<"\t"<<x<<endl;
    cout<<"y\t"<<&y<<"\t"<<y<<endl;


}
```

输出结果：

```
Var      Address         Value

---------before swap----------------------

a         0x7fff5fbff7d8 123

b         0x7fff5fbff7d4 456

---------before swap----------------------

x         0x7fff5fbff73c 123

y         0x7fff5fbff738 456

---------after swap-----------------------

x         0x7fff5fbff73c 456
y         0x7fff5fbff738 123

---------after swap-----------------------

a         0x7fff5fbff7d8 123
b         0x7fff5fbff7d4 456
```

这个结果说明了函数 *Swap* 内的形参 *x* 和 *y* 只是 *main* 函数中变量 *a* 和 *b* 的副本（值传递），因为 *x* 和 *a* ，*y* 和 *b* 的地址不同。因此，在 *Swap* 函数内部，*x* 和 *y* 的值可以交换，但这并不会影响 *main* 函数中的 *a* 和 *b* 的值。

实际上，实参变量对形参变量的数据传递是“值传递”，即单向传递，只能由实参传给形参，而不能由形参传给实参。在内存中实参单元与形参单元是不同的单元。

下面的两个例子进一步说明了：函数的形参和实参存储在不同的存储单元，不能混为一谈，即使对应的形参和实参名称是一样的：

```
void fun(int a, int b)

{

    a=a*10;

    b=b+a; cout<<a<<'\t'

    <<b<<endl;

}

void main(void)

{

   int a=2, b=3;

   fun(a,b); cout<<a<<'\t'

   <<b<<endl;

}
```

输出结果：

```
20 23

2 3
```

需要注意，形参是被调函数中的变量；实参是主调函数赋给被调函数的特定值。形参必须要定义类型，因为在定义被调函数时，不知道具体要操作什么数，而定义的是要操作什么类型的数。而在函数调用语句中，实参不必定义数据类型，因为实参传递的是一个具体的值 (常量)，程序可依据这个数值的表面形式来判断其类型，并将其赋值到对应的形参变量中。

5.2 数组

程序实现中常常需要批量组织数据。例如当记录每位学生的成绩的时候，这时想要使用多个变量十分复杂且很难操作，此时就需要一种新的数据结构 —— 数组。

数组一般有一维数组，二维及多维数组，以及字符数组、字符串等形式。在这里主要介绍一维数组。

一维数组在声明时满足以下结构：

类型数组名 [表达式]

要特别注意的是，表达式表示数组元素的个数，下标从 0 开始！

例如，*int*a[4]; 就是定义了由 4 个 int 型元素组成的 a 数组，由于下标从 0 开始，其元素分别为： *a*[0],*a*[1],*a*[2],*a*[3]。

此外，表达式在如 VC6, VS2010 中不能为变量，但是在 gcc4, dev-cpp 允许表达式为常量或变量，即支持动态数组。但从严格意义上讲，C++ 不允许对数组的大小作动态的定义，即数组的大小不能是变量，必须是常量。

5.2.1 数组的本质

定义数组就是定义了一块连续的空间，数组元素按序存放在这块空间中。数组名字就是数组首元素的内存地址。数组名是一个常量，不能被赋值。

空间的大小等于元素数×每个元素所占的空间大小, 如定义 int a[10]，将占用 40 个字节空间，因为每个整型数占 4 个字节。我们可以通过下面的代码来了解如何给数组分配存储空间：

```
#include
<iostream>
#define NUM 10

using namespace
std; int main()

{

    int a[NUM];

    int i;

     for(i=0; i<NUM; i++)
     {
      a[i]=i;
     }


     cout<<"Variable\tAddress\t\tContent\n";
     cout<<"-----------------------------\n";
     for(i=0; i<NUM; i++)
      {
         cout<<"a["<<i<<"]\t"<<&a[i]<<"\t"<<a[i]<<
     "\n;
      }


     cout<<"-----------------------------\n";
     cout<<"a\t"<<&a<<"\t"<<*a<<"\n";

     return 0;
}
```

输出示例：

Variable	Address	Content
a[0]	0x7fff5fbff7c0	0
a[1]	0x7fff5fbff7c4	1
a[2]	0x7fff5fbff7c8	2
a[3]	0x7fff5fbff7cc	3
a[4]	0x7fff5fbff7d0	4
a[5]	0x7fff5fbff7d4	5
a[6]	0x7fff5fbff7d8	6
a[7]	0x7fff5fbff7dc	7
a[8]	0x7fff5fbff7e0	8

5.2.2 一维数组的初始化

一维数组可以直接在定义时被初始化，例如：

$$int\ a[10] = \{0,1,2,3,4,5,6,7,8,9\};$$

初始化的长度短于要被初始化的数组元素数目，那么剩余元素被初始化为 0，例如。

$$int\ a[10] = \{0,1,2,3,4\};$$

就相当于：

$$int\ a[10] = \{0,1,2,3,4,0,0,0,0,0\};$$

利用这个方式，我们可以很快初始化数组元素全为 0：

$$int\ a[10] = \{0\};$$

(就相当于 $int\ a[10] = \{0,0,0,0,0,0,0,0,0,0\};$)

带有初始化的数组可以不定义长度，例如下面的写法是可行的：

$$int\ a[\ ] = \{1,2,3,4,5\};$$

(就相当于 $int\ a[5] = \{1,2,3,4,5\}$);

下面的代码展示了几种初始化的方法：

```
#include <iostream>
#define NUM 10
using namespace std;
int main()
{
    int x[]={2, 3, 5, 1, 4};
    int y[5] = {2, 3, 5};
    int z[5] = {};


    cout<<"array x:\t";
    for(int i=0; i<5; i++)
    {
    cout<<x[i]<<"\t";
    }


    cout<<"\n array y:\t";
    for(int i=0; i<5; i++)
    {

        cout<<y[i]<<"\t";
    }


    cout<<"\n array z:\t";
    for(int i=0; i<5; i++)
    {
        cout<<z[i]<<"\t";
    }


    return 0;
}
```

输出示例：

```
array x:    2    3    5    1    4
array y:    2    3    5    0    0
array z:    0    0    0    0    0
```

5.2.3 数组使用问题

5.2.3.1 数组赋值问题

数组的运算都是通过其元素实现的，不允许对数组进行整体赋值操作，只能使用循环逐一复制元素。

例如，将整数类型数组 a 和数组 b 的差送入数组 c 中，实现方法如下：

```
for(i=0; i<8; i++)
    c[i] = a[i] - b[i];
```

正确示范

```
c = a - b;
```

错误示范

5.2.3.2 数组下标问题

数组元素编号从 0 开始计数，元素访问格式为 x[0]、x[1]、……

其中，元素的序号称为下标。程序中，下标可为整数、整型变量或结果为整型的任意表达式。

语言不检查数组下标的越界。如果定义数组 a[10]，合法的下标范围是 0 - 9，但如果你引用 a[10]，系统不会报错，但是程序非常容易出现异常。

5.2.3.3 输入输出问题

在输入及输出数组元素时，务必注意要对单个元素进行操作，切忌直接对数组名称进行操作！

```
int array[3];
for(int i = 0; i < 3; i++){
    cin >> array[i];
}
```

输入正确示范

```
int array[3];
cin >> array;
```

输入错误示范

```
int array[3];
for(int i = 0; i < 3; i++){
    cout << array[i];
}
```

输出正确示范

```
int array[3];
cout << array;
```

输出错误示范

5.2.4 一维数组应用示例

为了分析学生对程序设计课程的掌握程度，需要对该课程成绩进行统计。请编程序，输入一个

班 32 名学生的程序设计课程成绩，然后按 10 分为一段，统计各段成绩的学生人数，并输出。

这个问题主要需要四个步骤：初始化、输入、计算和输出。

```
#include <iostream>
#define NUM 5
#define SCOPE 11
using namespace std;
int main()
{
    float score[NUM];
    int level[SCOPE]={};
    int i;
    for(i = 0; i < NUM; i++)
    {
         cin>>score[i]; int k
         = score[i]/10;
         level[k]++;
     }
     for(i = 0; i < SCOPE; i++){
         cout<<i*10<<" -- "<<(i+1)*10-1<<": "<<level[i]<<"\n";
     }
     Return 0;
}
```

输入示例：

```
67 88 83 55 99
```

输出示例：

```
  0 -- 9:       0

 10 -- 19:      0

 20 -- 29:      0

 30 -- 39:      0

 40 -- 49:      0

 50 -- 59:      1

 60 -- 69:      1

 70 -- 79:      0

 80 -- 89:      2

 90 -- 99:      1

100 -- 109:     0
```

5.2.5 小练习

1. 逆序输出。输入 8 个数，存入数组后逆序打印输出

输入样例：4 7 3 13 9 6 21 1

输出样例：1 21 6 9 13 3 7 4

2. 计算向量内积。

输入向量包含的元素个数 n，输入第一个向量的元素；输入第二个向量的元素，最后输出向量的内积。

输入样例：

```
3
1  0  1
2  5  6
```

输出样例：

```
8
```

5.2.6 数组作为函数参数

数组作为函数参数分以下两种情形：

- 数组元素作为函数参数
- 数组名作为函数参数

5.2.6.1 数组元素作为函数参数

数组元素作为函数参数跟普通变量或表达式作为函数的参数完全一样，采用值传递将数据拷贝给函数。

5.2.6.2 数组名作为函数参数

当把数组名传递给一个函数时，其实质是把该数组的首地址传递过去，即把实际参数中的数组首地址作为形式参数中的数组的首地址。

用数组名作函数参数，实参与形参都应用数组名。这时，函数传递的是数组在内存中的地址。

将数组名作为函数参数时，实参中的数组地址传到形参中，此时实参形参共用同一段内存。

这是一个反转数组的程序：

```
#include <iostream>
#include <cassert>
#define NUM 9

using namespace std;
void swap(int array[], int m, int n);
void print_array(int array[]);
int main(){
    int score[NUM] = {10,20,30,40,50,60,70,80,90};
    int i = 0;
    for(i=0; i < (NUM/2); i++){
```

```
        swap(score, i, NUM-i-1);
    }
    print_array(score);
    return 0;
}

void swap(int array[], int m, int n){
    cout << "array[" << m << "]:\t" << &array[m] << "\t" << array[m] << endl;
    cout<< "array["<<n<<"]:\t" << &array[n] << "\t"<<array[n] << endl;
    int tmp;
    tmp = array[m];
    array[m] = array[n];
    array[n] = tmp;
}

void print_array(int array[]){
    for(int i=0; i<NUM; i++){
        cout<<array[i]<<"\t";
    }
    cout<<endl;
}
```

下面是一个求课程平均成绩的程序：

```
#include <iostream>
#define NUM 10
using namespace std;
float average(float array[], int length);
int main()
{
    float score[NUM] = {78,82,85,60,90,100,98,94,99,100};
    float avg= average(score, NUM);
    cout<<"Average score = "<<avg<<endl;
    return 0;
}
float average(float array[], int length)
{
    int i;
    float avg=0;
    float sum=0;
    for(i = 0; i < length; i++)
    {
    sum = sum + array[i];
     }
     avg = sum/length;
     return avg;
}
```

从这个例子中，我们可以看到，数组 b 和数组 a 占据同一段内存。

```
Void fun(int a[2])
{
    for(int i=0;i<2;i++)
       a[i]=a[i]*a[i];
}
void main(void)
{
   int b[2]={2,4}; cout<<b[0]<<'\t'
   <<b[1]<<endl; fun(b);
   cout<<b[0]<<'\t'<<b[1]<<endl;
}
```

注意：1.用数组名作函数参数，应在主调函数和被调函数中分别定义数组，且类型一致。

2.需指定实参数组大小，形参数组的大小可不指定。数组名作实参实际上是传递数组的首地址。

3、数组名代表数组在内存中存储的首地址，这样，数组名作函数实参，实际上传递的是数组在内存中的首地址。实参和形参共占一段内存单元，形参数组中的值发生变化，也相当于实参数组中的值发生变化。

VI 指 针

6.1指针核心概念

1. 本质：指针变量是存放内存地址的特殊变量，地址与指针可视为同义词，变量的指针即其内存地址。

2. 双重特性：是 C/C++高效强大的核心（如实现复杂数据传递），但也是程序 bug（非法操作、内存错误）的主要来源。

例题：指针存储变量地址

```
#include <iostream>
using namespace std;
int main() {
    int a = 10;
    int *p = &a; // p 存储 a 的地址
    cout << "a 的地址：" << &a << endl;
    cout << "指针 p 的值：" << p << endl;
    cout << "指针 p 指向的值：" << *p << endl;
    return 0;
}
```

说明：通过输出验证指针存储变量地址，且 *p 可访问变量值。

6.2指针定义与初始化

6.2.1 定义

指针既然是变量，所以也具有变量的三个要素：

- 变量的名称。命名规则与一般变量一样，是由英文字符或下划线开始的。
- 变量的类型。“X 型指针”、“指向 X 类型的指针”。
- 变量的值。有特殊含义---是某个内存地址。

格式：类型 *指针名 ，如 int *pi （指向 int 型的指针）、 char *pc （指向 char 型的指针）。

6.2.2. 安全初始化

- 赋已初始化变量的地址： int i=0; int *pi=&i;
- 赋值为 NULL（空指针，不指向任何内存）： int *p=NULL;
- 动态分配内存（new）： int *p=new int; ，需用 delete 释放。

例题：三种安全初始化方式

```cpp
#include <iostream>
using namespace std;
int main() {
    // 1. 赋值已初始化变量地址
    int x = 20;
    int *p1 = &x;
    // 2. 赋值为 NULL
    int *p2 = NULL;
    // 3. 动态分配内存
    int *p3 = new int;
    *p3 = 30;

    cout << "*p1 = " << *p1 << endl;
    cout << "p2 = " << p2 << endl;
    cout << "*p3 = " << *p3 << endl;

    delete p3; // 释放动态内存
    return 0;
}
```

说明：展示指针的三种安全初始化方式，动态内存需手动释放。

6.3指针的赋值

6.3.1关键运算符

1.取地址符&：&运算符用来获取变量地址，如 &i 表示变量 i 的内存地址。

2.取值运算符 *：*运算访问指针指向的变量，如 *p=5 表示将 5 存入 p 指向的内存单元。

3.& 与 * 的关系：&和*互为逆运算， &(*p)=p ， *(&x)=x 。

例题：&与*的用法

```cpp
#include <iostream>
using namespace std;
int main() {
    int num = 5;
    int *ptr = &num; // &取地址，赋值给指针

    cout << "num 的值：" << num << endl;
    cout << "num 的地址：" << &num << endl;
    cout << "ptr 存储的地址：" << ptr << endl;
    cout << "ptr 指向的值：" << *ptr << endl; // *取值

    *ptr = 10; // 通过*修改原变量值
    cout << "修改后 num 的值：" << num << endl;
    return 0;
}
```

说明：体现&取地址、*取值及修改原变量的功能。

6.3.2指针new和delete

1. 指针变量没有赋上全值之前，不要用*改变取值， *其实在改变另一个实体数据类型的值

2. 此语句会在堆上为指定类型的变量分配足够的内存，并返回指向该内存的指针。如果内存分配成功，这个指针可以用于访问新分配的内存。如果内存分配失败（例如，因为堆空间不足）

3. 三种安全赋值方法

- NULL
- 赋初始化过的变量地址
- new-delete

6.4指针运算规则

6.4.1加减运算： 以指向类型的字长为单位，仅适用于数组指针，如 p++ 指向数组下一个元素。

1. 指针的加法（+）运算：

- p++表示 p=p+1
- 如果指针指向数组变量，允许对指针值加上一个整数表达式。
- 例如

```
p = &a[5],则 q=p+3 将使 q 指向 a[8]
```

2. 指针的减法（-）运算

- p--表示 p=p-1
- 如果指针指向数组变量，允许对指针值减去一个整数表达式。
- 允许两个指针值相减，得到的结果是两个指针之间的距离
- 指针的加减运算是以其指向的类型的字长为单位的

6.4.2 关系运算： 支持 ==、!=、>、< 等，用于比较指针指向的地址大小，如 pi==NULL 判断空指针。

- == 和 !=

  判断两个指针的值是否相等和不相等

  例如：

```
pi == NULL; pi != NULL

px == py;
```

- > , >= 和<=

  判断两个指针值的大小关系

  例如：

px < py 判断 px 所指向的地址是否小于 py 所指向的地址

例题：指针加减运算（数组场景）

```
#include <iostream>
using namespace std;
int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int *p = arr; // p 指向数组首元素

    cout << "p 指向的值：" << *p << endl; // 1
    p++; // 指针后移，指向 arr[1]
    cout << "p++后指向的值：" << *p << endl; // 2
    p += 2; // 指针后移 2 位，指向 arr[3]
    cout << "p+=2 后指向的值：" << *p << endl; // 4

    int *q = &arr[4];
    cout << "q - p = " << q - p << endl; // 计算指针间距，结果为 1
    return 0;
}
```

说明：指针加减以元素类型为单位，支持计算指针间距。

6.5指针与数组

1. 等价性：数组名是常量指针， a[i]、p[i]、*(a+i)、*(p+i) 完全等价。

   例题：四种数组元素访问方式

```
#include <iostream>
using namespace std;
int main() {
    int a[5] = {0, 1, 2, 3, 4};
    int *p = a;

    for (int i = 0; i < 5; i++) {
        cout << "a[" << i << "] = " << a[i]
             << ", p[" << i << "] = " << p[i]
             << ", *(a+" << i << ") = " << *(a + i)
             << ", *(p+" << i << ") = " << *(p + i) << endl;
    }
    return 0;
}
```

2. 差异性：指针是变量（可自增/赋值），数组名是常量（不可修改）。

3. 访问效率：指针直接操作地址（如 p++ ）比下标法（ a[i] ）效率更高，但下标法更直观。 （ 三种方法：引用数组中各元素的值有 3 种方法）

- (1) 下标法；

```
#include <stdio.h>

int main()
{
    int a[10];
    int i;
    printf("enter 10 integer numbers:\n");
    for(i=0; i<10; i++)
        scanf("%d", &a[i]);
    for(i=0; i<10; i++)
        printf("%d ", a[i]);
    printf("\n");
    return 0;
}
```

```
enter 10 integer numbers:
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
```

- (2) 通过数组名计算数组元素地址，找出元素的值；

```
#include <stdio.h>
int main()
{
    int a[10]; int i;
    printf("enter 10 integer numbers:\n");
    for(i=0;i<10;i++) scanf("%d",&a[i]);
    for(i=0;i<10;i++)
        printf("%d ",*(a+i));
    printf("\n");
    return 0;
}
```

scanf("%d",a+i)

• (3) 用指针变量指向数组元素

```
#include <stdio.h>
int main()
{
    int a[10]; int *p,i;
    printf("enter 10 integer numbers:\n");
    for(i=0;i<10;i++) scanf("%d",&a[i]);
    for(p=a;p<(a+10);p++)
        printf("%d ",*p);
    printf("\n");
    return 0;
}
```

6.6动态内存管理

1. 动态分配： new 类型 （单个变量）、 new 类型[长度] （动态数组）。

2. 释放： delete 指针 （释放单个变量）、 delete[] 指针 （释放动态数组），避免内存泄漏。

例题：动态数组的创建与释放

```
#include <iostream>
using namespace std;
int main() {
    int n;
    cout << "输入数组长度：";
    cin >> n;

    int *arr = new int[n]; // 动态创建数组
    for (int i = 0; i < n; i++) {
        arr[i] = i + 1;
    }

    cout << "动态数组元素：";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }

    delete[] arr; // 释放动态数组
    return 0;
}
```

说明：动态数组长度可自定义，使用后必须用 delete[] 释放。

6.7指针的应用场景

6.7.1 函数参数

- 指针既然是数据类型，自然可以做函数的参数和返回值的类型值的类型
- 指针作为参数的经典例子：
- 这里的函数调用过程还是“实参”的内容复制到“形参”

例题：指针作为函数参数（交换函数）

```
#include <iostream>
using namespace std;
void Swap(int *x, int *y) {
    int tmp = *x;
    *x = *y;
    *y = tmp;
}
int main() {
    int a = 10, b = 20;
    cout << "交换前：a=" << a << ", b=" << b << endl;
    Swap(&a, &b); // 传递地址
    cout << "交换后：a=" << a << ", b=" << b << endl;
    return 0;
}
```

说明：通过指针修改实参值，实现两数交换。

- 小结

1. 实参数组名是指针常量，但形参数组名是按指针变量处理
2. 在函数调用进行虚实结合后，它的值就是实参数组首元素的地址
3. 函数执行期间，形参数组可以再被赋值

```
void fun (int arr[ ],int n)
{
cout << *arr <<endl;
arr = arr + 3;
cout << *arr <<endl;
}
```

6.7.2. 函数指针

1. 定义

- 如果在程序中定义了一个函数，在编译时，编译系统为函数代码分配一段存储空间，这段存储空间的起始地址，称为这个函数的指针。
- 可以定义一个指向函数的指针变量，用来存放某一函数的起始地址，这就意味着此指针变量

指向该函数。例如： int (*p)(int,int);

- 定义 p 是指向函数的指针变量，它可以指向类型为整型且有两个整型参数的函数。p 的类型用 int (*)(int,int)表示

2. 用函数指针变量调用函数

- 用函数求整数 a 和 b 中的大者。

- 解题思路：定义一个函数 max，实现求两个整数中的大者。在主函数调用 max 函数，除了可以通过函数名调用外，还可以通过指向函数的指针变量来实现。分别编程并作比较。

举例：通过函数名调用函数

```
int max(int, int);
int main()
{
    int a, b, c;
    printf("please enter a and b:");
    scanf("%d,%d", &a, &b);
    c = max(a, b);
    printf("%d,%d,max=%d\n", a, b, c);
    return 0;
}
int max(int x, int y)
{    int z;
    if (x > y)
        z = x;
    else
        z = y;
    return (z);
}
```

3. 用指向函数的指针作函数参数

- 指向函数的指针变量的一个重要用途是把函数的地址作为参数传递到其他函数

- 指向函数的指针可以作为函数参数，把函数的入口地址传递给形参，这样就能够在被调用的函数中使用实参函数

例题：有两个整数 a 和 b，由用户输入 1,2 或 3。如输入 1，程序就给出 a 和 b 中大者，输入 2，就给出 a 和 b 中小者，输入 3，则求 a 与 b 之和。

```
Void fun(int x, int y, int (*p)(int,int));
int max(int,int); int min(int,int); int
add(int,int);
int main()
{
int a=34,b=-21,n;
printf("please choose 1,2 or 3:");
scanf(“%d”,&n);
if (n==1) fun(a,b,max);
else if (n==2) fun(a,b,min);
else if (n==3) fun(a,b,add);
return 0;
}
```

例题：函数指针（动态调用函数）

```
#include <iostream>
using namespace std;
int max(int x, int y) { return x > y ? x : y; }
int min(int x, int y) { return x < y ? x : y; }
int main() {
    int (*p)(int, int); // 定义函数指针
    int a = 5, b = 3;

    p = max;
    cout << "最大值: " << (*p)(a, b) << endl; // 调用 max

    p = min;
    cout << "最小值: " << (*p)(a, b) << endl; // 调用 min
    return 0;
}
```

说明：函数指针可切换指向不同函数，动态调用。

6.7.3 返回指针值的函数

- 一个函数可以返回一个整型值、字符值、实型值等，也可以返回指针型的数据，即地址。其概念与以前类似，只是返回的值的类型是指针类型而已
- 定义返回指针值的函数的一般形式为
- 类型名 *函数名(参数表列);

例题：有 a 个学生，每个学生有 b 门课程的成绩。要求在用户输入学生序号以后，能输出该学生的全部成绩。用指针函数实现（参数含指针参数、返回值含指针参数）。

```
float *search(float (*pointer)[4], int n);

int main()
{
    float score[3][4] = {{60, 70, 80, 90}, {56, 89, 67, 88}, {34, 78, 90, 66}};
    float *p;
    int k;
    scanf("%d", &k);
    printf("The scores of No.%d are:\n", k);
    p = search(score, k);  // 返回 k 号学生课程首地址
    for (int i = 0; i < 4; i++)
    {
        printf("%5.2f\t", *(p + i));
    }
    printf("\n");
    return 0;
}
float *search(float (*pointer)[4], int n)
{
    float *pt;
    pt = *(pointer + n);  // 指针寻址，类别下标寻址
    return (pt);
}
```

例题：指针函数（返回数组地址）

```
#include <iostream>
using namespace std;
float *search(float (*pointer)[4], int n) {
    float *pt = *(pointer + n); // 返回第 n 个学生成绩首地址
    return pt;
}
int main() {
    float score[3][4] = {{60,70,80,90}, {56,89,67,88}, {34,78,90,66}};
    int k;
    cout << "输入学生序号（0-2）：";
    cin >> k;

    float *p = search(score, k);
    cout << "该学生成绩：";
    for (int i = 0; i < 4; i++) {
        cout << *(p + i) << " ";
    }
    return 0;
}
```

说明：指针函数返回数组地址，实现查询指定学生成绩。

6.7.4 指针数组作 main 函数的形参

main 函数形参：int main(int argc, char *argv[]) ，argv 为指针数组，存储命令行参数。
例题：main 函数形参（命令行参数）

```
#include <iostream>
using namespace std;
int main(int argc, char *argv[]) {
    cout << "命令行参数个数：" << argc << endl;
    for (int i = 0; i < argc; i++) {
        cout << "argv[" << i << "] = " << argv[i] << endl;
    }
    return 0;
}
```

说明：运行时输入 ./program a b c ，argv[0]为程序路径，argv[1]-argv[3]为输入参数。

6.8使用原则

- 始终明确指针指向的内存地址，避免野指针。
- 动态分配的内存必须手动释放，防止内存泄漏。
- 指针未初始化时，不可用 * 操作取值。

VII 结构体

7.1 绪论

生活中存在大量由不同性质数据构成的实体（如日期、通讯录、人等），数组难以准确描述这类复杂实体，结构体应运而生。

7.2 结构体基本知识

7.2.1 定义

- 结构体是一组相关的不同类型数据的集合，可自定义新的数据类型

```
struct 结构类型名
{
类型 成员名 1;
类型 成员名 2;
……
类型 成员名 n;
}; //分号不能少
```

例如

```
为了描述日期，可以定义如下结构：
struct date
{
int year;  // 年：整型作为结构中的成员
int month;  // 月
int day;  // 日
};
```

- 编译时结构定义不分配内存，声明结构变量时才分配；在程序中，结构的定义可以在一个函数的内部，也可以在所有函数的外部。

7.2.2 变量声明

有两种方式：

- 先定义结构类型再声明变量，如 struct date {int year; int month; int day; }; struct date today; ；
- 定义结构类型的同时声明变量，如 struct date {int year; int month; int day; } today; 。

7.2.3 嵌套

这里定义结构类型为 person，man 为结构变量。在 person 结构中成员 name 是字符串存储姓名，sex 为字符型，表示性别，birthday 为前面定义的日期 date 类型的结构。birthday 这个成员就是一个嵌套的结构类型定义。

```
struct person /* 定义 person 结构类型 */
{
char name[30];
char sex;
struct date birthday; /* 结构的成员又是结构，即结构的嵌套定义 */
} man;
```

7.2.4 初始化

可对结构变量整体初始化，嵌套结构初始化需逐层进行。

```
struct date
{
int year;
int month;
int day;
};
struct person
{
char name[30];
char sex;
struct date birthday;
};
struct date today = { 2001, 10, 1};
struct person man = { "zhang",'M',{1980,3,28} };
```

7.2.5 成员访问

- 用“.”运算符，如 today.year=2010; ;
- 嵌套结构需逐层访问，如 man.birthday.year=1980;
- 结构成员可参与各类运算，结构变量整体仅支持赋值“=”和取地址“&”操作，且赋值时两侧类型需相同。

例:输入今天的日期，然后输出该日期。

```
//date_struct.cpp
#include <iostream>
using namespace std;
int main()
{
    struct date
    {
        int year;
        int month;
        int day;
};
    struct date today; /* 说明结构变量 today */
    cin>>today.year>>today.month>>today.day;
    cout<<"Year: "<<today.year
        <<" Month: "<<today.month
        <<" Day: "<<today.day
        <<endl;
    return 0;
}
```

7.2.6 注意

- 结构体类型的变量在内存依照其成员的顺序顺序排列，所占内存空间的大小是其全体成员所占空间的总和。
- 在编译时，仅对变量分配空间，不对类型分配空间。
- 对结构体中各个成员可以单独引用、赋值，其作用与变量等同。

  格式：变量名 . 成员名 student1 . num

- 不能对结构体变量整体赋值或输出，只能分别对各个成员引用。

```
cin>>student1;
cin>>student1.num; student1.num=100;
```

- 可以将一个结构体变量整体赋给另外一个相同类型的结构体变量。

```
student2=student1;
```

- 嵌套的结构体变量必须逐层引用。

```
student1.birthday.day=25;
```

- 结构体变量中的成员可以同一般变量一样进行运算。

```
student1.birthday.day++; student1.score+=60;
```

例题：

定义日期结构体，声明并初始化变量，修改成员值后输出。

```
#include <iostream>
using namespace std;

int main() {
    // 定义日期结构体
    struct date {
        int year;
        int month;
        int day;
    };
    // 初始化结构体变量
    struct date today = {2024, 10, 13};
    // 修改成员值
    today.day = 14;
    // 输出成员
    cout << "修改后的日期: " << today.year << "年" << today.month << "月" << today.day <<
"日" << endl;
    return 0;
}
```

解析:1. 先定义 date 结构体，包含年、月、日三个整型成员；

2. 声明变量 today 并初始化为 2024 年 10 月 13 日；

3. 通过“.”运算符修改 day 为 14；
4. 输出修改后的日期，体现结构体成员的访问与修改操作。

例题：(结构体嵌套)定义日期结构体和人员结构体（包含日期类型成员），初始化并输出人员信息。

```
#include <cstring>
using namespace std;

int main() {
    // 定义日期结构体
    struct date {
        int year;
        int month;
        int day;
    };
    // 定义人员结构体（嵌套日期结构体）
    struct person {
        char name[30];
        char sex;
        struct date birthday;
    };
    // 初始化嵌套结构体变量
    struct person man = {"Zhang San", 'M', {1990, 5, 20}};
    // 输出信息
    cout << "姓名：" << man.name << endl;
    cout << "性别：" << man.sex << endl;
    cout << "生日：" << man.birthday.year << "年" << man.birthday.month << "月" <<
man.birthday.day << "日" << endl;
    return 0;
}
```

解析：

1. 先定义 date 结构体，再定义 person 结构体，其成员 birthday 为 date 类型；
2. 初始化 man 时，对嵌套的 birthday 成员需用大括号逐层初始化；
3. 访问嵌套成员需通过多次“.”运算符，如 man.birthday.year 。

7.3 结构体数组

7.3.1 定义

有两重关系：结构中可包含数组成员；也可用结构类型作为数组元素类型.

```
struct person
{
char name[30];
char sex;
struct date birthday;
} man;
struct person student[100];
```

7.3.2 成员访问

- 同一般的数组一样，结构数组中每个元素的起始下标从 0 开始，数组名称表示该结构数组的存储首地址。
- 结构数组存放在一连续的内存区域中，它所占内存数目为结构类型的大小乘以数组元素的个数。

例如:我们要将数组 man 中的 3 号元素赋值为:"Peter",'M',1980,3,1，就可以使用下列语句

```
struct person man[100];
strcpy(man[3].name, "Peter");
man[3].sex='M';
man[3].birthday.year=1980;
man[3].birthday.month=3;
man[3].birthday.day=1;
```

例题:定义学生结构体数组，存储 3 名学生信息，输出所有学生信息。

```
#include <iostream>
#include <cstring>
using namespace std;

int main() {
    // 定义学生结构体
    struct Student {
        int num;
        char name[20];
        char sex;
        int age;
    };
    // 初始化结构体数组
    struct Student stu[3] = {
        {10101, "Li Lin", 'M', 18},
        {10102, "Zhang Fang", 'M', 19},
        {10104, "Wang Min", 'F', 20}
    };
    // 遍历数组输出信息
    for (int i = 0; i < 3; i++) {
        cout << "学号: " << stu[i].num << " 姓名: " << stu[i].name
            << " 性别: " << stu[i].sex << " 年龄: " << stu[i].age << endl;
    }
    return 0;
}
```

解析 ：

1. 定义 Student 结构体，包含学号、姓名等成员；
2. 声明结构体数组 stu 并初始化 3 名学生数据；
3. 通过循环遍历数组，用“数组下标+.”运算符访问每个元素的成员并输出。

7.4 结构体与指针

7.4.1 结构体指针定义

- 两重关系:结构中可包含指针成员；也可定义指向结构体的指针.

```
struct somebody
{
char name[30];
int score;
int *ptr;
};
struct person *man;
```

- 结构体指针是指向一种结构类型的指针变量，它是结构体在内存中的首地址，结构指针具有一般指针的特性，如在一定条件下两个指针可以进行比较，也可以与整数进行加减.但在指针操作时应注意：进行地址运算时的放大因子由所指向的结构的实际大小决定.

```
struct 结构体类型 *结构体变量;
```

```
struct date *today;
struct address *home;
```

7.4.2 成员访问

- 通过结构体指针访问成员有两种方式： (*指针变量).成员名 和 指针变量->成员名 ，如 (*ppt).x=0; 和 ppt->x=0;
- **通过结构指针访问成员可以采用运算符"->"进行操作.**

7.4.3 指向结构体数组的指针

可通过指针遍历结构体数组，访问数组中元素的成员。

例题 1：用结构体指针访问结构体变量及成员。

```
#include <iostream>
#include <cstring>
using namespace std;

int main() {
    struct point {
        int x;
        int y;
    };
    struct point pt = {3, 5};
    struct point *ppt = &pt; // 定义结构体指针并指向 pt

    // 两种指针访问成员的方式
    (*ppt).x = 10;
    ppt->y = 20;

    cout << "pt.x = " << pt.x << ", pt.y = " << pt.y << endl;
    return 0;
}
```

解析：

1. 定义 point 结构体和变量 pt ，初始化坐标为(3,5)；
2. 定义指针 ppt 指向 pt ；
3. 用 (*ppt).x 和 ppt->y 两种方式修改成员值，最终输出修改后的坐标。

例题 2：有 3 个学生的信息，放在结构体数组中，要求输出全部学生的信息。

```
#include <stdio.h>
struct Student
{
    int num;
    char name[20];
    char sex;
    int age;
};
struct Student stu[3] = {
    {10101, "Li Lin", 'M', 18},
    {10102, "Zhang Fang", 'M', 19},
    {10104, "Wang Min", 'F', 20}
};
int main()
{
    struct Student *p;
    printf(" No. Name              sex age\n");
    for (p = stu; p < stu + 3; p++)
       printf("%5d %-20s %2c %4d\n",
              p->num, p->name,
              p->sex, p->age);
    return 0;
}
```

7.5 结构体作函数参数

有三种方式：

1. 用结构体变量的成员作参数，属于“值传递”，需保证实参与形参类型一致。
2. 用结构体变量作实参，按顺序传递全部内容，形参需为同类型结构体变量，开销大且形参修改不影响实参。
3. 用指向结构体变量（或数组元素）的指针作实参，传递地址，效率高。

例题 1：有 n 个结构体变量，内含学生学号、姓名和 3 门课程的成绩。

要求输出平均成绩最高的学生的信息(包括学号、姓名、3 门课程成绩和平均成绩)。

解题思路：将 n 个学生的数据表示为结构体数组。按照功能函数化的思想，分别用 3 个函数来实现不同的功能：

- 用 input 函数输入数据和求各学生平均成绩
- 用 max 函数找平均成绩最高的学生
- 用 print 函数输出成绩最高学生的信息

• 在主函数中先后调用这 3 个函数，用指向结构体变量的指针作实参。

最后得到结果。

• 本程序假设 n=3

```
#include <stdio.h>
#define N 3

struct Student
{
    int num;
    char name[20];
    float score[3];
    float aver;
};

void input(struct Student stu[]);
struct Student max(struct Student stu[]);
void print(struct Student stud);

int main()
{
    struct Student stu[N], *p = stu;
    input(p);
    print(max(p));
    return 0;
}

void input(struct Student stu[])
{
    int i;
    printf("请输入各学生的信息：学号、姓名、三门课成绩:\n");
    for (i = 0; i < N; i++)
    {
        scanf("%d %s %f %f %f",
              &stu[i].num,
              stu[i].name,
              &stu[i].score[0],
              &stu[i].score[1],
              &stu[i].score[2]);
        stu[i].aver = (stu[i].score[0] + stu[i].score[1] + stu[i].score[2]) / 3.0;
    }
}
struct Student max(struct Student stu[])
{
    int i, m = 0;
    for (i = 0; i < N; i++)
    {
        if (stu[i].aver > stu[m].aver)
           m = i;
    }
    return stu[m];
}

void print(struct Student stud)
{
    printf("\n 成绩最高的学生是：\n");
    printf("学号:%d\n", stud.num);
    printf("姓名:%s\n", stud.name);
    printf("三门课成绩:%.1f, %.1f, %.1f\n",
          stud.score[0], stud.score[1], stud.score[2]);
    printf("平均成绩:%6.2f\n", stud.aver);
}
```

例题 2：用指针作参数，修改结构体变量的值。

```
#include <iostream>
#include <cstring>
using namespace std;

struct Student {
    int num;
    char name[20];
};

// 用指针作参数修改结构体成员
void modifyStudent(struct Student *p) {
    p->num = 10105;
    strcpy(p->name, "Zhao Wei");
}

int main() {
    struct Student stu = {10101, "Li Lin"};
    modifyStudent(&stu); // 传递结构体变量地址
    cout << "修改后：学号=" << stu.num << ", 姓名=" << stu.name << endl;
    return 0;
}
```

解析：1. 定义 Student 结构体和修改函数 modifyStudent ，函数参数为结构体指针；
2. 主函数中声明并初始化 stu ，调用函数时传递其地址；
3. 函数内部通过指针修改结构体成员，主函数中输出修改后的结果，体现地址传递的优势。

7.6 自定义类型名（typedef）

1. 标准类型有：
- int、char、float、double、long
- 以及结构体(struct)、共用体(union)、枚举类型(enum)等

2. 此外还可以定义新的类型来代替已有的类型名，如：
- typedef int INTEGER; 指定 INTEGER 代表 int 类型
- typedef float REAL; 指定 REAL 代表 float 类型

例题：用 typedef 简化结构体定义与变量声明。

```
#include <iostream>
#include <cstring>
using namespace std;
// 用 typedef 自定义结构体类型名
typedef struct {
    char name[12];
    char sex;
} Person;
int main() {
    Person student; // 直接用自定义类型名声明变量
    strcpy(student.name, "Chen Yu");
    student.sex = 'F';
    cout << "姓名: " << student.name << " 性别: " << student.sex << endl;
    return 0;
}
```

解析： 1. 用 typedef 将匿名结构体定义为 Person 类型；

2. 直接用 Person 声明变量 student ，简化代码，提高可读性；
3. 对变量成员赋值并输出。

7.7 枚举类型

7.7.1 定义

- 如果一个变量只有几种可能的值，则可以定义为枚举类型
- 所谓“枚举”就是指把可能的值一一列举出来，变量的值只限于列举出来的值的范围内

7.7.2 使用枚举类型例

- 声明枚举类型用 enum 开头 例如： enum Weekday{sun,mon,tue, wed,thu,fri,sat};
- 声明了一个枚举类型 enum Weekday，然后可以用此类型来定义变量 ： enum Weekday workday,weekend;

7.7.3 注意

- 编译器对枚举类型的枚举元素按常量处理，故称枚举常量。不要因为它们是标识符(有名字)而把它们看作变量，不能对它们赋值。 例如：

```
sun=0; mon=1; 错误
```

- 枚举元素为常量，不可赋值；默认值按顺序为 0、1、2……，也可人为指定；枚举变量的值仅限于枚举元素范围内，可用于判断比较。

例题 1：

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    enum color { red=1, green, blue };
    enum color favorite_color;

    /* ask user to choose color */
    printf("请输入你喜欢的颜色: (1. red, 2. green, 3. blue): ");
    scanf("%d", &favorite_color);

    switch (favorite_color)
    {
    case red:
        printf("你喜欢的颜色是红色");
        break;
    case green:
        printf("你喜欢的颜色是绿色");
        break;
    case blue:
        printf("你喜欢的颜色是蓝色");
        break;
    default:
        printf("你没有选择你喜欢的颜色");
    }
    return 0;
}
```

例题 2：
定义星期枚举类型，判断输入的星期值对应的星期几。

```
#include <iostream>
using namespace std;

int main() {
    // 定义星期枚举类型
    enum Weekday {sun, mon, tue, wed, thu, fri, sat};
    enum Weekday workday;
    int n;
    cout << "请输入 0-6 的数字：";
    cin >> n;
    workday = (enum Weekday)n; // 类型转换
    switch (workday) {
        case sun: cout << "星期日" << endl; break;
        case mon: cout << "星期一" << endl; break;
        case tue: cout << "星期二" << endl; break;
        default: cout << "其他星期" << endl;
    }
    return 0;
}
```

解析：
1. 定义 Weekday 枚举类型，元素默认值为 0（sun）到 6（sat）；
2. 将输入的整数转换为枚举类型赋值给 workday ；
3. 通过 switch 语句判断并输出对应的星期名称。

7.8 引用

7.8.1 定义

- 引用是变量的别名（小名）
- 对引用操作等同于对原始变量的操作
- 其中原变量名必须是一个已定义过的变量
- max 与 refmax 在内存中占用同一地址，即同一地址两个名字

7.8.2 特性

1. 引用和指针的区别
   - 引用在定义的时候必须要初始化
   - 引用一旦被初始化后不得再作为其它变量的别名引用
2. 引用和指针的关系

> 当&a 的前面有类型符时(如 int &a)，它必然是对引用的声明；
> 如 果前面无类型符(如 cout<<&a;)，则是取变量的地址
> 对引用执行取地址操作，结果只能是所引用的变量地址

3. 不合法的引用

企图建立引用的数组	int & a[9];
企图建立指向引用的指针	int & *p;
企图建立引用的引用	int & &px;
企图建立 void 应用	void & a=x;
用类型或 NULL 来初始化	int & a=int; int & a =NULL;

7.8.3 引用作为函数参数

主要用作函数参数，可实现对实参的直接操作，如交换函数中使用引用可交换两个变量的值。

例题：用引用作函数参数，交换两个整数的值。

```
#include <iostream>
using namespace std;

// 引用作参数交换变量
void Swap(int &x, int &y) {
    int tmp = x;
    x = y;
    y = tmp;
}

int main() {
    int a = 123, b = 456;
    cout << "交换前：a=" << a << ", b=" << b << endl;
    Swap(a, b); // 直接传递变量，引用关联实参
    cout << "交换后：a=" << a << ", b=" << b << endl;
    return 0;
}
```

解析：

1. 定义 Swap 函数，参数为整数引用；
2. 主函数中调用 Swap 时，引用 x 、 y 分别关联 a 、 b ；
3. 函数内部修改引用的值，实质修改原变量，实现交换效果。

7.9 数据类型小结

基本数据类型难以解决所有问题，复合数据类型（如结构体、数组、指针）由基本数据类型派生而来，抽象数据类型在复合数据类型基础上增加数据操作，进而进化为“类（Class）”。

Ⅷ 略（不考）

IX 类与对象

9.1 面向对象编程基础和类的定义

9.1.1 面向过程 vs 面向对象

特性	面向过程	面向对象
核心思想	围绕功能/算法展开	围绕对象（数据 + 操作）展开
程序组成	程序 = 算法 + 数据结构	程序 = 对象集合 + 消息
数据与操作关系	数据与操作分离	数据与操作封装为对象
适用场景	简单程序、线性流程	复杂程序、模块化开发

表 1: 面向过程与面向对象对比

重点

对象的两大核心要素：1. 属性（Attribute）：对象的静态特征（如“学生”的姓名、成绩）；2. 行为（Behavior）：对象的动态操作（如“学生”的选课、考试）。

9.1.2 定义和结构

类的定义

1. 类是一种复杂的数据类型，它是将不同类型的数据和与这些数据相关的运算封装在一起的集合体。
2. 类将一些数据及与数据相关的函数封装在一起，使类中的数据得到很好的“保护”。在大型程序中不会被随意修改。

```
class  Student //类名
{ private : //私有
    char Name[20];
    float Math;
    float Chiese;
    public : //公有
    float  average;
    void SetName(char *name);
    void  SetMath(float math);
    void  SetChinese(float ch);
    float GetAverage(void);
}; //分号不能省
```

重点

1. 用关键字 priviate 限定的成员称为私有成员，对私有成员限定在该类的内部使用，即只允许该类中的成员函数使用私有的成员数据
2. 对于私有的成员函数，**只能被该类内的成员函数调用**
3. 类就相当于私有成员的作用域
4. 每一个限制词 (private 等) 在类体中可使用**多次**。一旦使用了限制词，该限制词一直有效，直到下一个限制词开始为止
5. 如果未加说明，类中成员默认的访问权限是 private，即私有的
6. 在定义一个类时, 在其类体中又包含了一个类的完整定义，称为类的嵌套

9.1.3 作用域

定义和分类

作用域是指程序中所说明的标识符在哪一个区间内有效，即在哪一个区间内可以使用或引用该标识符。
在 C++ 中，作用域共分为五类：块作用域、文件作用域、函数原型作用域、函数作用域和类的作用域。

块作用域:

1. 我们把用花括号括起来的一部分程序称为一个块。在块内说明的标识符，只能在该块内引用，即其作用域在该块内，开始于标识符的说明处，结束于块的结尾处。
2. 在一个函数内部定义的变量或在一个块中定义的变量称为局部变量。

9.2 类与对象的定义及使用

9.2.1 类的定义格式

```
class 类名 {
    访问控制符:  // private（默认）、public、protected
    数据成员;  // 属性，如变量、数组等
    成员函数声明/定义;  // 行为，如函数、方法等
};  // 注意：类定义末尾必须加英文分号
```

类定义的关键注意点：

1. 类仅为“数据类型模板”，定义时不分配内存；
2. 数据成员不能在类内直接初始化（如 ‘int x=10;‘ 非法）；
3. 访问控制符作用域：从当前控制符开始，到下一个控制符结束。

9.2.2 访问控制符权限

访问控制符	类内访问	类外访问	派生类访问
private	可访问	不可访问	不可访问
public	可访问	可访问	可访问
protected	可访问	不可访问	可访问

表 2: 访问控制符权限范围

9.2.3 成员函数的两种定义方式

```
1  // 方式1：类内定义（自动为内联函数，适合短函数）
2  class Student {
3      public:
4      void ShowScore() {  // 类内定义
5          cout << "Math:" << math << ",Chinese:" << chinese << endl
               ;
6      }
7      private:
8      float math, chinese;
9  };
10
11 // 方式2：类外定义（需用作用域解析符::）
12 class Teacher {
13     public:
14     void SetName(char* name);  // 类内声明
15     private:
16     char name[20];
17 };
18 // 类外定义
19 void Teacher::SetName(char* name) {
20     strcpy(this->name, name);  // this指针指向当前对象
21 }
```

9.2.4 对象变量的定义与使用

重点

在定义类时，只是定义了一种导出的数据类型，**并不为类分配存储空间**，所以，在定义类中的数据成员时，不能对其初始化

例如class a{int a=1;};是不允许的

正确的赋值方式应该是在定义类的变量后，由类内函数对类内数据进行修改，具体如下：

```
        // 1. 对象定义格式：存储类型 类名 对象名1, 对象名2;
        Student st1;           // 局部对象（栈内存）
        static Student st2;    // 静态对象（全局数据区，仅初始化一次）
        Student* pst = new Student();  // 动态对象（堆内存，需delete释放）

        // 2. 对象成员访问方式
        // （1）直接访问（仅公有成员）：对象名.成员名
        st1.ShowScore();  // 调用公有成员函数

        // （2）指针访问：指针名->成员名
        pst->ShowScore();

        // （3）私有成员访问：通过公有成员函数间接访问
        class Student {
            public:
            void SetMath(float m) { math = m; }  // 公有函数间接修改私有成员
            private:
            float math;
        };
        st1.SetMath(95.5);  // 合法：通过公有函数修改私有成员
        // st1.math = 95.5;  // 非法：类外直接访问私有成员
```

全局类变量的定义如下：

```
    class  A {
        float  x,y;
        public:
        void Setxy( float a, float b  ){  x=a;   y=b;   }
        void  Print(void) {  cout<<x<< '\t' <<y<<endl;  }
    } a1,a2;
```

9.3 对象的使用

9.3.1 赋值和引用

核心方法

1. 一个对象的成员就是该对象的类所定义的成员，有成员数据和成员函数，引用时同结构体变量类似，用“.”运算符
2. 用成员选择运算符“.”只能访问对象的公有成员，而不能访问对象的私有成员或保护成员。若要访问对象的私有的数据成员，只能通过对象的公有成员函数来获取

```
class A {
    float x,y;
    public:
    float m,n;
    void Setxy( float a, float b ){ x=a; y=b; }
    void Print(void) { cout<<x<<'\t'<<y<<endl; }
};
void main(void)
{
    A a1,a2;
    a1.m=10; a1.n=20; //为公有成员数据赋值
    a1.Setxy(2.0,5.0);//通过公有函数为私有数据赋值
    a1.Print();
}
```

注意

1. 同类型的变量之间支持整体赋值，如 `A a1,a2; a1.m=10; a1=a2;`
2. 可以定义类类型的指针，类类型的引用，对象数组，指向类类型的指针数组和指向一维或多维对象数组的指针变量
3. 一个类的对象，可作为另一个类的成员

9.3.2 类的作用域

类类型的**作用域**：在函数定义之外定义的类，其类名的作用域为文件作用域；而在函数体内定义的类，其类名的作用域为块作用域。**对象的作用域与前面介绍的变量作用域完全相同**，全局对象、局部对象、局部静态对象等。

```
class A //全局变量
{
    float x,y;
    public:
    float m,n;
```

```
        void Setxy( float a, float b ){ x=a; y=b; }
        void Print(void) { cout<<x<<'\t'<<y<<endl; }
    }a3,a4;//全局内可用

    void main(void)
    {
        A a1,a2;  //a1,a2为局部变量，只能在main内使用
        class B  //只能在main之内使用
        {
            int i,j;
            public :
            void Setij(int m, int n){ i=m; j=n; }
        };
        B b1,b2;
        a1.Setxy(2.0, 5.0);
        b1.Setij(1,2);
    }
```

9.3.3 类的嵌套

在定义一个类时，在其类体中又包含了一个类的完整定义，称为类的嵌套。**类是允许嵌套定义的。**

```
class A
{
    class B //B嵌套在A之内
    {
        int i,j;
        public :
        void Setij(int m, int n){ i=m; j=n; }
    };
    float x,y;
    public:
    B b1,b2; //嵌套的类的对象
    void Setxy( float a, float b ){ x=a; y=b; }
    void Print(void) { cout<<x<<'\t'<<y<<endl; }
};
```

9.3.4 类的具体应用

```
// 嵌套类定义
class A {
    class B {
        int i,j;
        public :
        void Setij(int m, int n){ i=m; j=n; }
    };
    float x,y;
    public:
    B b1,b2;
    void Setxy( float a, float b ){ x=a; y=b; }
    void Print(void) { cout<<x<<'\t'<<y<<endl; }
};

// 三角形类定义及实现
class Triangle{
    private:
    float a,b,c;  //三边为私有成员数据
    public:
    void Setabc(float x, float y, float z);//置三边的值
    void Getabc(float &x, float &y, float &z);//取三边的值
    float Perimeter(void);//计算三角形的周长
    float Area(void);//计算三角形的面积
    void Print(void);//打印相关信息
};

void Triangle::Setabc (float x,float y,float z)
{ a =x; b=y; c=z;  }//置三边的值

void Triangle::Getabc (float &x,float &y,float &z) //取三边的值
{ x=a;  y=b;   z=c;}

float Triangle::Perimeter (void)
{ return (a+b+c)/2; }//计算三角形的周长

float Triangle::Area (void)  //计算三角形的面积
{
    float area, p;
    p= Perimeter();
    area=sqrt((p-a)*(p-b)*(p-c)*p);
```

```
41          return area;
42      }
43
44      void Triangle::Print(void) //打印相关信息
45      { cout<<"Peri="<<Perimeter()<<'\t'<<"Area="<<Area()<<endl;}
46
47      void main(void)
48      {
49          Triangle Tri1; //定义三角形类的一个实例（对象）
50          Tri1.Setabc (4,5,6);    //为三边置初值
51          float x,y,z;
52          Tri1.Getabc (x,y,z);  //将三边的值为x,y,z赋值
53          cout<<x<<'\t'<<y<<'\t'<<z<<endl;
54          cout<<"s="<<Tri1.Perimeter ()<<endl;//求三角形的周长
55          cout<<"Area="<<Tri1.Area ()<<endl;//求三角形的面积
56          cout<<"Tri1:"<<endl;
57          Tri1.Print ();//打印有关信息
58      }
```

9.3.5 成员函数的重载

重点

1. 类中的成员函数可以带有缺省的参数，也可以重载成员函数
2. 重载时，函数的形参必须在类型或数目上不同

```
1           // 类Stu，展示函数重载（成绩计算相关）
2           class Stu
3           {
4               char Name[20];
5               float Chinese;
6               float Math;
7               float English;
8               float Physical;
9               public:
10              float Average(void);//语文、数学平均成绩
11              float Average(int n);//四门课的平均成绩
12              float Sum(void);//语文、数学总分
13              float Sum(int n);//四门课总分
14              void Show(void);
15              void SetStudent(char*,float,float);//置姓名、语文、数学初值
```

```
16              void SetStudent(char *, float,float,float,float);//置姓名、成
                    绩
17              void SetName(char *);
18              char *GetName(void);
19          };
```

注：可以有缺省参数的成员函数，若形参不完全缺省，则必须从形参的右边开始缺省

9.3.6 类的指针的定义和赋值

类的名称不是地址，需要用&取地址。指针引用时需要->，不能用”.”

```
1           class A{
2               float x,y;
3               public:
4               float Sum(void) { return x+y; }
5               void Set(float a,float b) { x=a; y=b;}
6               void Print(void) {cout<<"x="<<x<<'\t'<<"y="<<y<<endl; }
7           };
8           void main(void)
9           {
10              A a1,a2;
11              A *p;         //定义类的指针
12              p=&a1;        //给指针赋值
13              p->Set(2.0, 3.0); //通过指针引用对象的成员函数
14              p->Print();
15              cout<<p->Sum()<<endl;
16              a2.Set(10.0, 20.0);  a2.Print();
17          }
```

9.3.7 返回引用类型的成员函数

重点

当函数声明为 `float &get()`时，return x; 返回的是变量 x 的引用（即 x 本身的“别名”）。通过这个引用，外部可以直接修改 x 的值（例如 obj.get() = 10; 会直接改变 obj 内部的 x）。

当函数声明为 `float get()`时，return x; 返回的是变量 x 的值拷贝。外部得到的是一个与 x 相等的临时值，修改这个返回值不会影响 x 本身（例如 obj.get() = 10; 会报错，因为临时值不能被赋值）。

总结：加`&`,`return`的就是 x 本身, 不加`&return`的就是 x 的值

```
class A{
    float x,y;
    public:
    float &Getx(void) { return x; } //返回x的引用
    void Set(float a,float b) { x=a; y=b; }
    void Print(void) { cout<<x<<'\t'<<y<<endl; }
};
void main(void)
{
    A a1,a2;
    a1.Set(3,5);
    cout<<"a1:␣";       a1.Print();
    a1.Getx()=30;    //将a1对象中的x成员赋值
    cout<<"changed␣a1:␣";
    a1.Print();
}
```

9.3.8 this 指针

核心特性

1. 自动生成与传递：定义类时，编译器会在成员函数中自动生成 this 指针，并在调用函数时隐式传递，开发者无需手动声明或传递。

2. 指向当前对象：this 始终指向调用该成员函数的对象

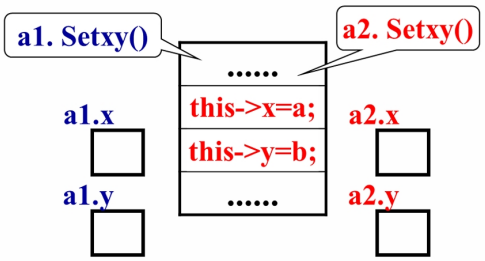
3. 可显式使用：在成员函数中，可通过 this-> 成员的形式显式访问当前对象的成员（通常用于参数与成员变量同名的场景）。

this 指针

■ 不同对象占据内存中的不同区域，它们所保存的数据各不相同，但对成员数据进行操作的成员函数的程序代码均是一样的。

```
class A{
      int x,y;
   public:
    void Setxy(int a, int b)
   { x=a;  y=b;}
};
A  a1, a2;
```

```
a1.Setxy(1,2);
a2.Setxy(3,4);
```



系统自动将对象的指针带到成员函数中

63

图 1: 使用说明

重点

当对一个对象调用成员函数时，编译程序先将对象的地址赋给 this 指针，然后调用成员函数，每次成员函数存取数据成员时，也隐含使用 this 指针

9.3.9 缺省

```
class A{
    float x,y;
    public:
    float Sum(void) { return x+y; }
    void Set(float a,float b=10.0) { x=a; y=b;}
    void Print(void) {cout<<"x="<<x<<'\t'<<"y="<<y<<endl; }
};
void main(void)
{
    A a1,a2;
    a1.Set(2.0,4.0);  //无缺省 x=2.0,y=4.0
    cout<<"a1:␣";       a1.Print();
    cout<<"a1.sum="<<a1.Sum()<<endl;
    a2.Set(20.0);    //缺省, x=20.0,y=10.0
    cout<<"a2:␣";       a2.Print();
    cout<<"a2.sum="<<a2.Sum()<<endl;
}
```

程序分析：类 A 的 Set 函数中，参数 b 被指定了缺省值 10.0。这意味着调用 Set 时，若不传入 b 的实参，会自动使用 10.0 作为默认值。

程序执行时，a1 调用 Set 时传入了两个参数（2.0 和 4.0），因此 a1 的 x=2.0、y=4.0；a2 调用 Set 时仅传入一个参数 20.0，y 则使用缺省值 10.0，即 a2 的 x=20.0、y=10.0。这种特性让函数调用更灵活，可根据场景选择是否省略部分参数，同时保证代码的可读性和简洁性。

9.4 构造函数与析构函数

9.4.1 构造函数与析构函数的定义

重点

1. 构造函数和析构函数是在类体中说明的两种特殊的成员函数
2. 构造函数是在**创建对象**时，使用给定的值来将对象初始化
3. 析构函数的功能正好相反，是在系统释放对象前，对对象做一些善后工作
4. **构造函数可以带参数、可以重载，同时没有返回值**
5. 构造函数是类的成员函数，系统约定构造函数名必须与类名相同。**构造函数提供了初始化对象的一种简单的方法**

9.4.2 构造函数（初始化对象）

提示

1. 构造函数的核心作用：在对象创建时自动调用，完成数据成员的初始化。
2. 构造函数的函数名必须与类名相同
3. 在定义构造函数时，**不能指定函数返回值的类型，也不能指定为 void 类型**
4. 一个类可以定义若干个构造函数（不同的初始化值）。当定义多个构造函数时，必须满足函数重载的原则
5. 若定义的类要说明该类的对象时，构造函数必须是公有的成员函数。如果定义的类仅用于派生其它类时，则可将构造函数定义为保护的成员函数
6. 由于构造函数属于类的成员函数，它对私有数据成员、保护的数据成员和公有的数据成员均能进行初始化

```
class Circle
{
    private:
        float radius;  // 半径（私有成员）
    public:
        Circle() // 1. 无参构造函数（缺省构造函数）
        {
            radius = 5.0;  // 初始化半径为默认值
        }
        Circle(float r) // 2. 带参构造函数（可重载）
        {
            radius = r;  // 用参数初始化半径
        }
        Circle(float r = 5.0) // 3. 带缺省参数的构造函数
        {
            radius = r;
        }

        float GetArea()
        {
        return 3.14 * radius * radius;
        }
};

// 对象初始化（自动调用对应构造函数）
Circle c1;           // 调用无参/缺省构造函数
Circle c2(3.0);      // 调用带参构造函数
Circle* c3 = new Circle(4.5);  // 动态对象初始化
```

构造函数的关键规则

1. 函数名与类名完全相同，无返回类型（不可写 void）；
2. 可重载（参数类型/个数不同），但缺省构造函数只能有一个；
3. 全局对象：main 函数执行前初始化；局部对象：定义时初始化；静态对象：首次定义时初始化。
4. 每一个对象必须要有相应的构造函数，若没有显式定义构造函数，**系统默认缺省的构造函数，格式为 className::className() ，该构造函数不会给数据赋值**。

另外，产生对象时，系统必定要调用构造函数。所以任一对象的构造函数必须唯一，在类中，若定义了没有参数的构造函数，或各参数均有缺省值的构造函数也称为缺省的构造函数，**缺省的构造函数只能有一个**，且部分参数缺省的构造函数不算缺省的构造函数。

```
class A{
        float x,y;
public:
  A(float a, float b){x=a;y=b;cout<<"初始化自动局部对象\n";}
  A(){ x=0; y=0; cout<<"初始化静态局部对象\n";}
  A(float a){ x=a; y=0; cout<<"初始化全局对象\n";}
 void Print(void){ cout<<x<<'\t'<<y<<endl; }
};
A a0(100.0);//定义全局对象
void f(void)
{ cout<<" 进入f()函数\n";A a2(1,2);
   static A a3; //初始化局部静态对象
}
void main(void)
{   cout<<"进入main函数\n";
   A a1(3.0, 7.0);//定义局部自动对象
      f();   f(); }
```

```
初始化全局对象
进入main函数
初始化自动局部对象
进入f()函数
初始化自动局部对象
初始化局部静态变量
进入f()函数
初始化自动局部对象
```

图 2: 示例

new 运算符

1. 可以使用 new 运算符来动态地建立对象。建立时要自动调用构造函数，以便完成初始化对象的数据成员。最后返回这个动态对象的起始地址
2. 用 new 运算符产生的动态对象，在不再使用这种对象时，必须用 delete 运算符来释放对象所占用的存储空间
3. 用 new 建立类的对象时，可以使用参数初始化动态空间

```
1   #include <iostream>
2   using namespace std;
3
4   class A {
5       float x, y;
6       public:
7       A(float a, float b) { x = a; y = b; }
8       A() { x = 0; y = 0; }
9       void Print(void) { cout << x << '\t' << y << endl; }
10  };
11
```

```
        void main(void) {
            {
                A *pa1, *pa2;
                pa1 = new A(3.0, 5.0);
                pa2 = new A;
                pa1->Print();
                pa2->Print();
                delete pa1;
                delete pa2;
            }
        }
```

9.4.3 析构函数

析构函数的特点

1. 析构函数是成员函数，函数体可写在类体内，**也可写在类体外**
2. 析构函数是一个特殊的成员函数，函数名必须与类名相同，并在其前面加上字符~，以便和构造函数名相区别
3. 析构函数不能带有任何参数，**不能有返回值，不指定函数类型**
4. 一个类中，**只能定义一个析构函数，析构函数不允许重载**
5. 析构函数是在撤消对象时由系统自动调用的。在程序的执行过程中，**当遇到某一对象的生存期结束时，系统自动调用析构函数**，然后再收回为对象分配的存储空间

```
#include <iostream>
using namespace std;

class A {
    float x, y;
    public:
    A(float a, float b) {
        x = a;
        y = b;
        cout << "调用非缺省的构造函数\n";
    }
    A() {
        x = 0;
        y = 0;
        cout << "调用缺省的构造函数\n";
    }
    ~A() {
```

```
18              cout << "调用析构函数\n";
19          }
20          void Print(void) {
21              cout << x << '\t' << y << endl;
22          }
23      };
24
25      void main(void) {
26          A a1;
27          A a2(3.0, 30.0);
28          cout << "退出主函数\n";
29      }//此时自动调用析构函数
30
31      程序输出内容为:
32      以上程序输出内容为:
33      调用缺省的构造函数
34      调用非缺省的构造函数
35      退出主函数
36      调用析构函数
37      调用析构函数
```

不同存储类型的对象调用构造函数及析构函数

1. 对于全局定义的对象（在函数外定义的对象），在程序开始执行时，调用构造函数；到程序结束时，调用析构函数
2. 对于局部定义的对象（在函数内定义的对象），当程序执行到定义对象的地方时，调用构造函数；在退出对象的作用域时，调用析构函数
3. 用 static 定义的局部对象，在首次到达对象的定义时调用构造函数；到程序结束时，调用析构函数

特别的，对于用 new 运算符动态生成的对象，在产生对象时调用构造函数，**只有使用 delete 运算符来释放对象时，才调用析构函数**。若不使用 delete 来撤消动态生成的对象，程序结束时，对象仍存在，并占用相应的存储空间，即系统不能自动地调用析构函数来撤消动态生成的对象。

用 new 运算符来动态生成对象数组时，自动调用构造函数，而用 delete 运算符来释放 p1 所指向的对象数组占用的存储空间时，在指针变量的前面必须加上 [ ]，才能将数组元素所占用的空间全部释放。否则，只释放第 0 个元素所占用的空间。

```
#include <iostream>
using namespace std;

class A {
    float x, y;
    public:
    A(float a = 0, float b = 0) {
        x = a;
        y = b;
        cout << "调用了构造函数\n";
    }
    void Print(void) {
        cout << x << '\t' << y << endl;
    }
    ~A() {
        cout << "调用了析构函数\n";
    }
};

void main(void) {
    cout << "进入main()函数\n";
    A *pa1;
    pa1 = new A[3];  // 开辟数组空间
    cout << "\n完成开辟数组空间\n\n";
    delete[] pa1;    // 必须用[]删除开辟的空间
    cout << "退出main()函数\n";
}

程序输出内容如下
进入main()函数
调用了构造函数
调用了构造函数
调用了构造函数
完成开辟数组空间
调用了析构函数
调用了析构函数
调用了析构函数
退出main()函数
```

9.4.4 拷贝构造函数

可以在定义一个对象的时候用另一个对象为其初始化，**即构造函数的参数是另一个对象的引用**，这种构造函数常为完成拷贝功能的构造函数. 如果没有定义完成拷贝功能的构造函数，编译器自动生成一个隐含的完成拷贝功能的构造函数，依次完成类中对应数据成员的拷贝

```
class Rectangle {
    private:
    float width, height;
    public:
    // 普通构造函数
    Rectangle(float w, float h)
    {
        width = w;
        height = h;
    }

    // 拷贝构造函数（参数为同类型引用）
    Rectangle(const Rectangle & obj) //&不能省
    {
        width = obj.width;  // 拷贝源对象的宽度
        height = obj.height;  // 拷贝源对象的高度
    }

    float GetArea()
    {
        return width * height;
    }
};

// 调用拷贝构造函数的场景
Rectangle rec1(3, 4);        // 普通构造
Rectangle rec2 = rec1;       // 拷贝构造（赋值式初始化）
Rectangle rec3(rec1);        // 拷贝构造（括号式初始化）
```

注意

若未显式定义拷贝构造函数，编译器自动生成“浅拷贝构造函数”：
浅拷贝：仅逐字节拷贝数据成员（若含动态内存，会导致“双重释放”问题）；
深拷贝：需显式定义拷贝构造函数，为新对象重新分配动态内存（避免浅拷贝问题）。

9.5 继承与派生

继承性是面向对象程序设计中最重要的机制。这种机制提供了无限重复利用程序资源的一种途径。通过 C++ 语言中的继承机制，可以扩充和完善旧的程序设计以适应新的需求。这样不仅可以节省程序开发的时间和资源，并且为未来程序增添了新的资源。

```
class Student{
    int num;
    char name[30];
    char sex;
public:
    void display() {   //对成员函数display的定义
        cout<<"num: "<<num<<endl;
        cout<<"name: "<<name<<endl;
        cout<<"sex: "<<sex<<endl;
    }
};

class Student1 : public Student{  // 使用继承，避免重复定义
    int num;                      // 此行原来已有
    char name[20];                // 此行原来已有
    char sex;                     // 此行原来已有
    int age;
    char addr[20];
public:
    void display(){ // 重写display函数
        cout<<"num: "<<num<<endl;   // 此行原来已有
        cout<<"name: "<<name<<endl; // 此行原来已有
        cout<<"sex: "<<sex<<endl;   // 此行原来已有
        cout<<"age: "<<age<<endl;
        cout<<"address: "<<addr<<endl;
    }
};
```

这里的类 Student1 里面有一些内容是类 Student 中已经包括的。实际上，定义类 Student1 时，可以利用原来定义的类 Student 作为基础，再加上新的内容即可，以减少重复的工作量。C++ 提供的继承机制就是为了解决这个问题。

在 C++ 中所谓“继承”就是在一个已存在的类的基础上建立一个新的类。已存在的类称为“基类 (base class)”或“父类 (father class)”。新建立的类称为“派生类 (derived class)”或“子类 (son class)”

我们可以通过以下代码实现继承机制，从而简化之前的代码：

```
class Student1: public Student{ //声明基类是Student
private:
    int age;  //新增加的数据成员
    string addr;  //新增加的数据成员
public:
    void display_1(){  //新增加的成员函数
        cout<<"age: "<<age<<endl;
        cout<<"address: "<<addr<<endl;
    }
};
```

通过继承机制，可以利用已有的数据类型来定义新的数据类型。所定义的新的数据类型不仅拥有新定义的成员，而且还同时拥有旧的成员。我们称已存在的用来派生新类的类为基类，又称为父类。由已存在的类派生出的新类称为派生类，又称为子类。

当从已有的类中派生出新的类时，可以对派生类做以下几种变化：

- 可以继承基类的成员数据或成员函数。
- 可以增加新的成员变量。
- 可以增加新的成员函数。
- 可以重新定义已有的成员函数。
- 可以改变现有的成员属性。

在 C++ 中有二种继承：单一继承和多重继承。当一个派生类仅由一个基类派生时，称为单一继承；而当一个派生类由二个或更多个基类所派生时，称为多重继承。

下面展示了派生的一般格式：

```
class ClassName: Access BaseClassName{
            //ClassName: 派生类名; Access: 继承方式; BaseClassName: 基类名
            //Access中, public: 公有基类; private: 私有基类; protected: 保护基类
    private:
    ......;    //私有成员说明
    protected:
    ......;    //保护成员说明
    public:
    ......;    //公有成员说明
} //派生类中新增加的成员
```

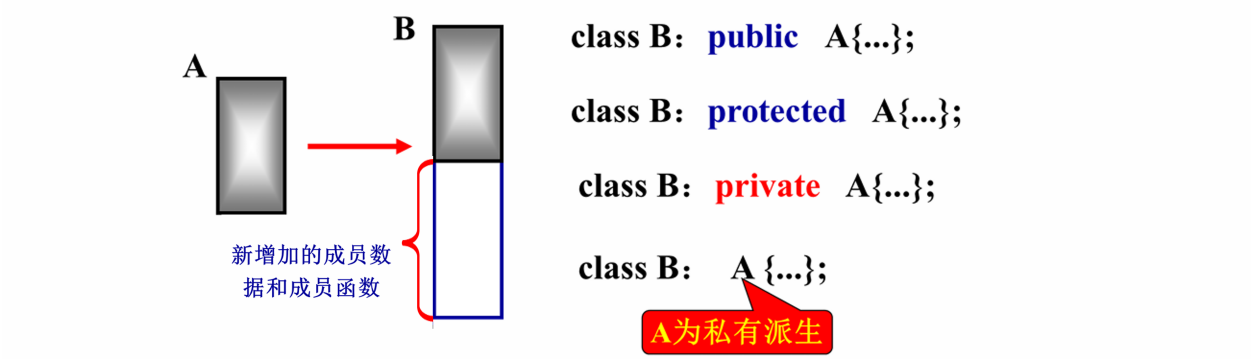


图 3: 派生示意图

派生并不是简单的扩充，有可能改变基类的性质。派生方式有三种：公有派生 (public)、保护派生 (protected) 和私有派生 (private)。在没有明确类型时，默认方式是私有派生。

下面我们分别介绍三类派生方式。

9.5.1 公有派生

公有派生的一般形式为：

```
class ClassName: public BaseClassName
```

公有派生时，基类成员在派生类中的访问权限如下表所示：

基类成员属性	派生类中	派生类外
公有	可以引用	可以引用
保护	可以引用	不可引用
私有	不可引用	不可引用

```
// x是基类私有，在派生类和类外中不能直接引用
// y是基类保护，在派生类中可以直接引用。而在类外不可直接引用
// z是基类公有，在派生类中和类外均可直接引用
class A {
    int x;  // 私有成员
protected: int y;  // 保护成员
public: int z;  // 公有成员
    A(int a,int b,int c){x=a;y=b;z=c;}  // 基类初始化
    int Getx(){return x;} // 返回x
    int Gety(){return y;} // 返回y
    void ShowA(){cout<<"x="<<x<<"\t"<<"y="<<y<<"\t"<<"z="<<z<<"\n";}
};

class B:public A{  // 公有继承
    int m,n;
public:
```

```
    B(int a,int b,int c,int d,int e):A(a,b,c){m=d;n=e;}
    void Show(){
        cout<<"m="<<m<<"\t"<<"n="<<n<<"\n";
        cout<<"x="<<Getx()<<"\t"<<"y="<<y<<"\t"<<"z="<<z<<"\n";  // x是基类私有，不能
            直接引用，需要通过Getx()函数访问
    }
    int Sum(){
        return (Getx()+y+z+m+n);  // y和z可以直接引用，x需要通过Getx()
    }
};

void main(void) {
    B b1(1,2,3,4,5);
    b1.ShowA();
    b1.Show();
    cout<<"Sum="<<b1.Sum()<<"\n";
    cout<<"x="<<b1.Getx()<<"\t";       // x私有，通过公有函数访问
    cout<<"y="<<b1.Gety()<<"\t";       // y保护，通过公有函数访问
    cout<<"z="<<b1.z<<"\n";            // z公有，可以直接访问
}
```

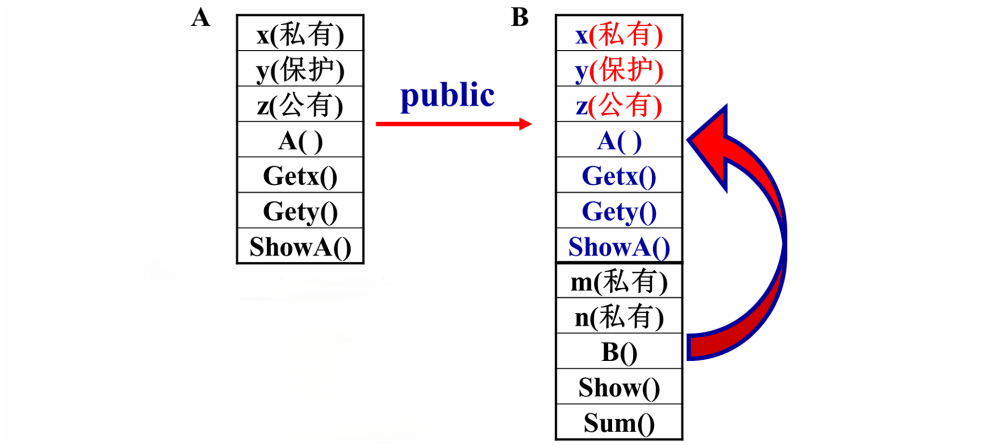


图 4: 公有派生示意图

由上面的代码和示意图，可以看到 x 在类 B 新增加的成员中不能直接调用，y 在类 B 中可以调用，z 在整个文件中可以调用。对类 B 的对象初始化即是对 x、z、m、n 等全部成员的初始化。

9.5.2 私有派生

私有派生时，基类成员在派生类中的访问权限如下表所示：

基类成员属性	派生类	派生类外
公有	可以引用	不可引用
保护	可以引用	不可引用
私有	不可引用	不可引用

```
class A {
    int x;  // 私有成员
protected: int y;  // 保护成员
public: int z;  // 公有成员
    A(int a,int b,int c){x=a;y=b;z=c;}  // 基类初始化
    int Getx(){return x;} // 返回x
    int Gety(){return y;} // 返回y
    void ShowA(){cout<<"x="<<x<<"\t"<<"y="<<y<<"\t"<<"z="<<z<<"\n";}
};
class B:private A{  // 私有派生
    int m,n;
public:
    B(int a,int b,int c,int d,int e):A(a,b,c){m=d;n=e;}
    void Show(){
        cout<<"m="<<m<<"\t"<<"n="<<n<<"\n";
        cout<<"x="<<Getx()<<"\t"<<"y="<<y<<"\t"<<"z="<<z<<"\n";
        // x是基类私有，不能直接引用
    }
    int Sum(){
        return (Getx()+y+z+m+n);  // y和z在私有继承中只能在类内直接访问
    }
};

void main(void)
{
    B b1(1,2,3,4,5);
    A a1(1,2,3);

    a1.ShowA();  // A类对象可以直接调用ShowA

    // b1.ShowA();  // 错误：私有继承后基类公有成员在派生类外不可访问
    b1.Show();      // 正确：调用派生类的公有成员函数

    cout<<"Sum="<<b1.Sum()<<"\n";
    //cout<<"x="<<b1.Getx()<<"\t";  // 错误：私有继承后x在类外不可直接访问
    //cout<<"y="<<b1.Gety()<<"\t";  // 错误：私有继承后y在类外不可直接访问
    //cout<<"z="<<b1.z<<"\n";      // 错误：私有继承后z在类外不可直接访问
}
```

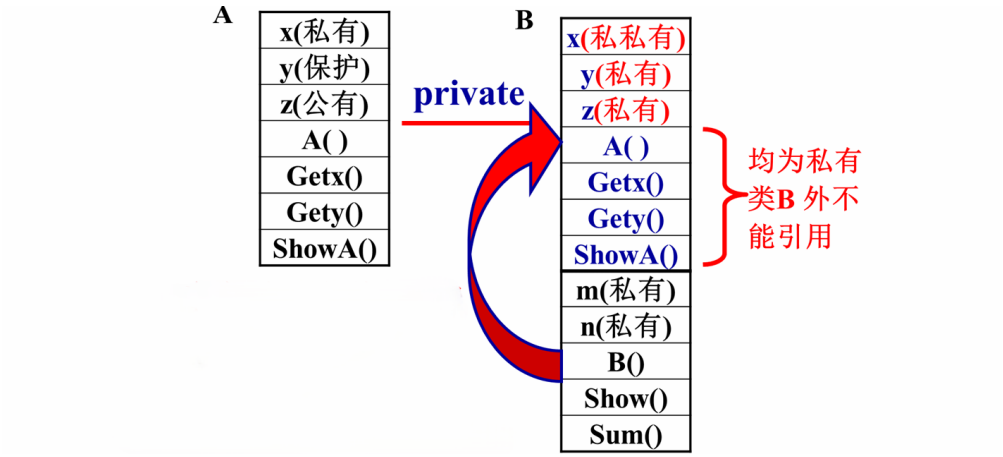


图 5: 私有派生示意图

由上面的代码和示意图，可以看到，y 和 z 可以在派生类中直接使用，但是无法在派生类 B 外直接引用。对类 B 的对象初始化即是对 x、y、z、m、n 等全部成员的初始化。

9.5.3 保护派生

保护派生时，基类成员在派生类中的访问权限如下表所示：

基类成员属性	派生类	派生类外
公有	可以引用	不可引用
保护	可以引用	不可引用
私有	不可引用	不可引用

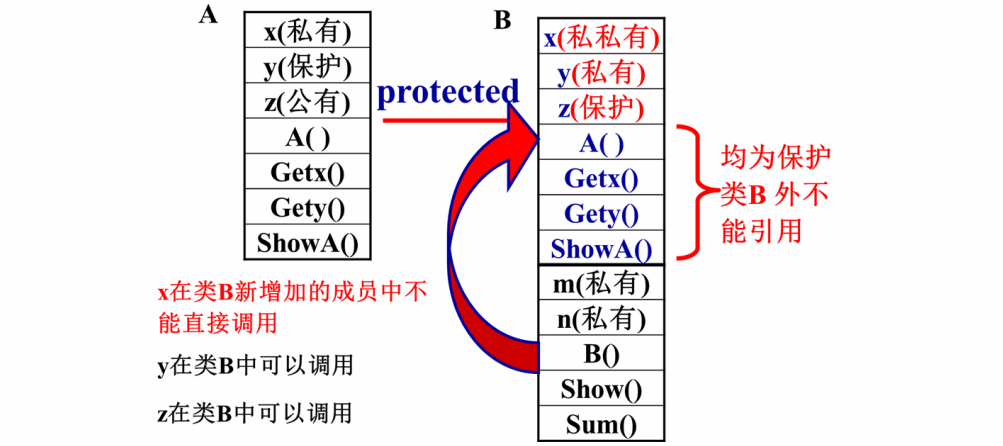


图 6: 保护派生示意图

需要说明的是，protected 成员是一种具有血缘关系内外有别的成员。它对派生类的对象而言，是公开成员，可以访问，对血缘外部而言，与私有成员一样被隐蔽。

9.5.4 抽象类与保护的成员函数

当定义了一个类，这个类只能用作基类来派生出新的类，而不能用这种类来定义对象时，称这种类为抽象类。当对某些特殊的对象要进行很好地封装时，需要定义抽象类。

将类的构造函数或析构函数的访问权限定义为保护的时，这种类为抽象类。

当把类中的构造函数或析构函数说明为私有的时，所定义的类通常是没有任何实用意义的，一般情况下，不能用它来产生对象，也不能用它来产生派生类。

```
class A {
    int x, y;
protected:
    A(int a,int b){x=a;y=b;}  // 基类初始化
public:
    void ShowA(){cout<<"x="<<x<<"\t"<<"y="<<y<<"\n";}
};
class B: public A{
    int m;
    // A a1; //在派生类中也不可以定义A的对象，实际上还是类外调用
public:
    B(int a,int b,int c):A(a,b)  // 可以在派生类中调用A的构造函数
    {m=c;}
    void Show(){ cout<<"m="<<m<<"\n"; ShowA(); }
};

void main(void){
    B b1(1,2,3); // 可以定义派生类对象
    b1.Show();
    // A aa; // 不可定义A的对象，因为A的构造函数是protected
}
```

在上面的程序中，任何时候都不能定义 A 的对象，但可以在初始化类 B 的对象时初始化原类 A 中的成员，因为 A() 在类 B 中是可以被调用的。

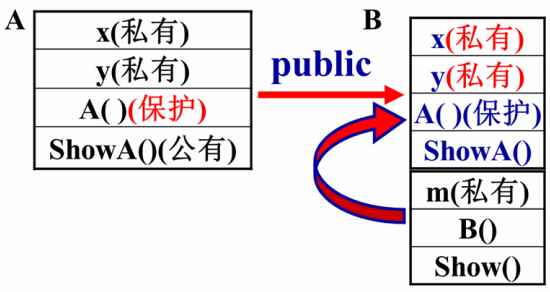


图 7: 抽象类与保护成员函数示意图

9.5.5 多重继承

在程序中，可以用多个基类来派生一个类。多重继承是单一继承的扩展。

```
class 类名 : <Access>类名1, ...., <Access>类名n{
    private: ......; // 私有成员说明;
    public: ......; // 公有成员说明;
    protected: ......; // 保护的成员说明;
};
//例:
class D : public A, protected B, private C{
    .... // 派生类中新增加成员
};
```

下面是一个使用多重继承的例子：

```
class A{
    int x1,y1;
public:
    A(int a,int b) { x1=a; y1=b; }
    void ShowA(void){ cout<<"A.x="<<x1<<"\t"<<"A.y="<<y1<<endl; }
};
class B{
    int x2,y2;
public:
    B(int a,int b) {x2=a; y2=b; }
    void ShowB(void){ cout<<"B.x="<<x2<<"\t"<<"B.y="<<y2<<endl; }
};
class C:public A,private B{  // 公有继承A，私有继承B
    int x,y;
public: C(int a,int b,int c,int d,int e,int f):A(a,b),B(c,d) {x=e; y=f; }
        void ShowC(void){
        cout<<"C.x="<<x<<"\t"<<"C.y="<<y<<endl;
        ShowA();  // 正确：公有继承，可以在派生类中调用基类A的公有成员
        ShowB();  // 正确：私有继承，可以在派生类中调用基类B的公有成员
    }
};  // 添加缺失的分号

void main(void){
    C c(1,2,3,4,5,6);
    c.ShowC();
    c.ShowA();  // 正确：公有继承，可以在类外调用基类A的公有成员
    // c.ShowB();  // 错误：私有继承，不能在类外调用基类B的公有成员
}
```

9.5.6 继承的应用

在平面上作两个点，连一直线，求直线的长度和直线中点的坐标。

基类为 Dot，有两个公有数据成员，即平面上的坐标 (x,y)，同时有构造函数及打印函数。

派生类为 Line，有两个基类 Dot 对象，分别存放两点的坐标，同时，从基类继承了一个 Dot 数据，存放直线中点的坐标。

首先分析基类和派生类之间的关系：

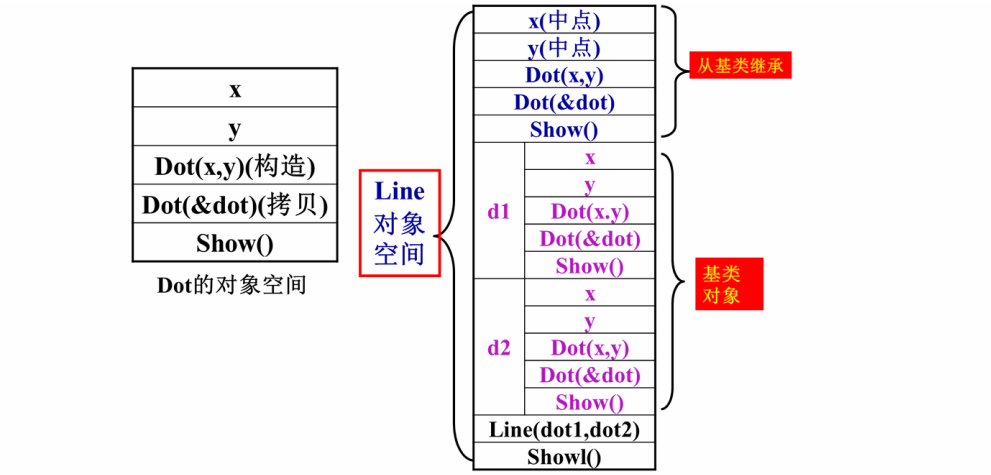


图 8: 抽象类与保护成员函数示意图

```
class Dot{
public:
    float x,y;
    Dot(float a=0, float b=0){ x=a; y=b;}  // 用坐标初始化Dot对象
    void Show(void){cout<<"x="<<x<<"\t"<<"y="<<y<<endl;}
};

class Line: public Dot{
    Dot d1,d2;  // 在Line中新说明的成员
public:
    Line(Dot dot1,Dot dot2):d1(dot1),d2(dot2){  // 对成员初始化
        x=(d1.x+d2.x)/2;
        y=(d1.y+d2.y)/2;
    }  // x, y是继承基类的成员

    void Showl(void){
        cout<<"Dot1: "; d1.Show();
        cout<<"Dot2: "; d2.Show();
        cout<<"Length="<<sqrt((d1.x-d2.x)*(d1.x-d2.x)+(d1.y-d2.y)*(d1.y-d2.y))<<endl;
        cout<<"Center: "<<"x="<<x<<"\t"<<"y="<<y<<endl;
    }
};
```

```
void main(void){
    float a,b;
    cout<<"Input Dot1: \n"; cin>>a>>b;
    Dot dot1(a,b);  // 调用Dot的构造函数
    cout<<"Input Dot2: \n"; cin>>a>>b;
    Dot dot2(a,b);
    Line line(dot1,dot2);
    line.Showl();
}
```

在这里需要介绍“赋值兼容”规则。即可以将派生类对象的值赋值给基类对象，反之不允许。下面的示意图对此进行了解释说明。

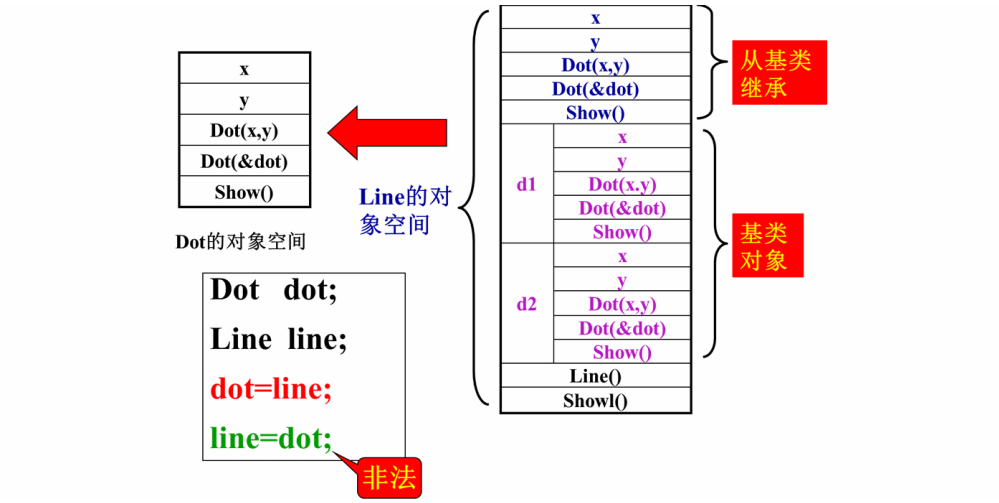


图 9: 赋值兼容规则

9.6 运算符重载与模板

所谓函数的重载是指完成不同功能的函数可以具有相同的函数名。C++ 的编译器是根据函数的实参来确定应该调用哪一个函数的。下面的例子展示了函数的重载：

```
int fun(int a, int b) //函数1
{ return a+b; }
int fun(int a)        //函数2
{ return a*a; }

void main(){
    cout << fun(3,5) << endl;  //调用函数1
    cout << fun(5) << endl;    //调用函数2
}
```

```
int sum, a=3, b=2;
```

```
sum = a + b;     // (int) = (int) + (int)

float add, x=3.2, y=2.5;
add = x + y;     // (float) = (float) + (float)

char str[4], c1[2]="a", c2[2]="b";
str = c1 + c2;  // (char *) = (char *) + (char *) - 这是错误的!
```

```
class A{
    float x, y;
public:
    A(float a = 0, float b = 0) { x = a; y = b; }
};

void main(void){
    A a(2,3), b(3,4), c;
    c = a + b;  // 错误: 两个对象不能使用+, 必须重新定义+
}
```

9.6.1 运算符重载

在 C++ 编程中，数据类型运算遵循严格的类型匹配规则。如上例所示，对于基本数据类型如整型 (int) 和浮点型 (float)，编译器能够自动识别数据类型并执行相应的加法运算。整型变量相加得到整型结果，浮点型变量相加得到浮点型结果，这都是由语言本身支持的。

然而，当遇到字符数组 (char *) 等用户自定义或复杂数据类型时，情况就不同了。示例中的字符数组相加操作 str = c1 + c2 实际上是不合法的，因为编译系统中的运算符"+" 本身并不具备处理字符数组相加的功能。

要使这类表达式能够正确运算，必须通过运算符重载来重新定义"+" 运算符的行为。运算符重载就是赋予已有的运算符多重含义。C++ 通过重新定义运算符，使它能够用于特定类的对象执行特定的功能。

运算符的重载从另一个方面体现了 OOP 技术的多态性，且同一运算符根据不同的运算对象可以完成不同的操作。

为了重载运算符，必须定义一个函数，并告诉编译器，遇到这个重载运算符就调用该函数，由这个函数来完成该运算符应该完成的操作。这种函数称为运算符重载函数，它通常是类的成员函数或者是友元函数。

运算符的操作数通常也应该是类的对象。

```
<类名> operator<运算符>(<参数表>)
{    // 函数体     }

//例如
A operator + (A &);  // 重载了类A的"+"运算符
//A 为返回类型,  operator + 为函数名, (A &) 为运算对象
```

其中，operator 是定义运算符重载函数的关键字，与其后的运算符一起构成函数名。

```
//没有重载运算符的例子
class A{
    int i;
public:
    A(int a=0) { i=a; }
    void Show(void) {
        cout << "i=" << i << endl;  // 需要添加输出语句
    }
    void AddA(A &a, A &b){
        i = a.i + b.i;
    }
};

void main(void){
    A a1(10), a2(20), a3;
    a1.Show();
    a2.Show();
    // a3 = a1 + a2;  // 不可直接运算，需要重载+运算符
    a3.AddA(a1, a2);  // 调用专门的功能函数来实现相加
    a3.Show();
}
```

```
//使用重载运算符的例子
class A{
    int i;
public:
    A(int a=0){ i=a; }
    void Show(void){ cout<<"i="<<i<<endl; }
    void AddA(A &a, A &b){ // 利用函数进行类之间的运算
        i = a.i + b.i;
    }
    A operator +(A &a){ // 重载运算符+
        A t;
        t.i = i + a.i;
        return t;
    }
};

void main(void){
    A a1(10), a2(20), a3;
    a1.Show();
    a2.Show();
    a3 = a1 + a2; // 重新解释了加法，可以直接进行类的运算
    //相当于a3 = a1.operator+(a2);
    a3.AddA(a1, a2); // 调用专门的功能函数
    a3.Show();
}
```

由上面两种实现类相加的方式示例，看出重载运算符和一般函数的相同点在于：

- 均为类的成员函数。
- 实现同一功能。

它们的区别在于：

- 函数的形式不同。
- 调用的对象不同。例如在上面的例子中，无重载运算符函数是由 a3 调用的，重载运算符函数是由 a1 调用的。

运算符重载的本质是重新定义运算符在特定类对象间的运算行为。其核心机制在于：当遇到重载的运算符时，由左操作数对象调用相应的运算符重载函数，右操作数作为参数传入该函数。运算完成后，将函数的返回值赋给运算结果对象，从而完成整个运算过程。

当用成员函数实现运算符的重载时，运算符重载函数的参数只能有二种情况：没有参数或带有一个参数。

对于只有一个操作数的运算符 (如 ++)，在重载这种运算符时，通常不能有参数；

而对于有二个操作数的运算符，只能带有一个参数。这参数可以是对象，对象的引用，或其它类型的参数。

在 C++ 中，不允许重载有三个操作数的运算符。

在 C++ 中, 大多允许运算符重载，不允许重载的运算符如下所示：

- 作用域操作符::
- 条件操作符?:
- 点操作符.
- 指向成员操作的指针操作符 ->* , . *
- 预处理符号 #

并且，只能对 C++ 中已定义了的运算符进行重载，而且，当重载一个运算符时，该运算符的优先级和结合律是不能改变的。下面的程序说明了这一点。

```
class room{
    float Length;
    float Wide;
public:
    room(float a=0.0, float b=0.0){ Length=a; Wide=b;}
    void Show(void){ cout<<"Length="<<Length<<"\t"<<"Wide="<<Wide<<endl; }
    void ShowArea(void){ cout<<"Area="<<Length*Wide<<endl; }
    room operator+(room &);  // 重载运算符+, 函数原型
```

```
};

room room::operator + (room &rr){ // 重载运算符，函数定义
    room rr;
    rr.Length = Length + r.Length;
    rr.Wide = Wide + r.Wide;
    return rr;
}

void main(void){
    room r1(3,2), r2(1,4), r3, r4;
    r1.Show();
    r2.Show();
    r3 = r1 + r2;
    r3.Show();
    r4 = r1 + r2 + r3;
    r4.Show();
    // 运算符的优先级和结合律是不能改变的
}
```

9.6.2 单目运算符的重载

只具有一个操作数的运算符为单目运算符，最常用的为 ++ 及–。可以看出，虽然运算后对象 a 的值一致，但先自加或后自加的重载运算符函数的返回值不一致，必须在重载时予以区分。

当 ++ 为前置运算时，它的运算符重载函数的一般格式为：

```
<type> operator ++()
{   ......;}
```

当 ++ 为后置运算时，它的运算符重载函数的一般格式为：

```
<type>  operator ++(int)
{   ......;}
```

例子如下：

```
class A{
    float x, y;
public:
    A(float a=0, float b=0){ x=a; y=b; }
    A operator ++(){// 重载前置++运算符
        A t;
        t.x = ++x;  // 先自增再赋值
        t.y = ++y;
        return t;
    }
    A operator ++(int) { // 重载后置++运算符
```

```
        A t;
        t.x = x++;   // 先赋值再自增
        t.y = y++;
        return t;
    }
};

void main(void){
    A a(2,3), b;
    b = ++a;   // 调用前置++
    b = a++;   // 调用后置++
}
```

两种方式的存储分析如下：

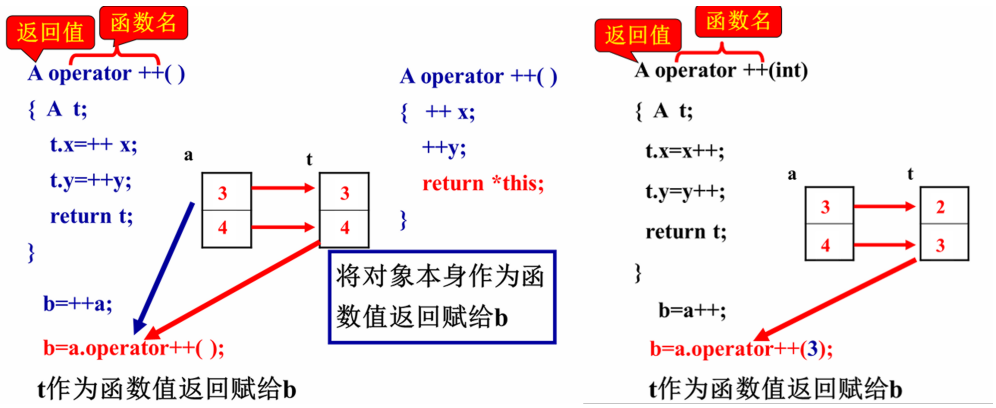


图 10: 存储分析

示例：

```
class incount{
    int c1, c2;
public:
    incount(int a = 0, int b = 0) { c1 = a; c2 = b; }
    void Show(void) { cout << "c1=" << c1 << "\t" << "c2=" << c2 << endl; }

    // 前置++
    incount operator++(){
        c1++;
        return *this;
    }

    // 后置++
    incount operator++(int){
        incount c;
        c.c1 = c1;
        c.c2 = c2;
        c1++;
        c2++;
        return c;
    }
};

void main(void){
    incount ic1(10, 20), ic2(100, 200), ic3, ic4;
    ic1.Show();
    ic2.Show();
    ic3 = ++ic1;  // ic3 = ic1.operator++()
    ic4 = ic2++;  // ic4 = ic2.operator++(0)
    ic3.Show();
    ic4.Show();

    // 预期结果：
    // ic1.c1=10 ic1.c2=20
    // ic2.c1=100 ic2.c2=200
    // ic3.c1=11 ic3.c2=200
    // ic4.c1=100 ic4.c2=200
}
```

9.7 运算符重载为友元函数

友元函数是在类外的普通函数，与一般函数的区别是可以调用类中的私有或保护数据。将运算符的重载函数定义为友元函数，参与运算的对象全部成为函数参数。

语法：

```
friend <类型说明> operator<运算符>(<参数表>)
```

示例：

```
class A
{int i;
public:
    A(int a=0)  {i=a; }
    void Show(void)  {cout<<"i="<<i<<endl;}
    friend A operator +(A &,A &);//友元函数，两个参数，为引用
};
A operator +(A &a,A&b)
    {A t;
    t.i=a.i+b.i;
    return t;}
void main(void)
{
    A a1(10),a2(20),a3;
    a1.Show();
    a2.Show();
    a3=a1+a2;//重新解释了加法，可直接进行类的运算
    a3.Show();
}
```

++ 为前置运算时，它的运算符重载函数的一般格式为：

A operator ++(A &a) ……; ++ 为后置运算时，它的运算符重载函数的一般格式为：

A operator ++(A &a, int) ……;

示例：

```
class incount
{
    int c1, c2;
public:
    incount(int a = 0, int b = 0) { c1 = a; c2 = b; }
    void Show(void) { cout << "c1=" << c1 << '\t' << "c2=" << c2 << endl; }
    friend incount operator ++(incount &); //前置
    friend incount operator ++(incount &, int); //后置
};

incount operator++(incount &c)
{
    c.c1++; c.c2++; return c;
```

```
}

incount operator++(incount &c, int)
{
    incount cc;
    cc = c;
    c.c1++; c.c2++; return cc;
}

void main(void)
{
    incount ic1(10, 20), ic2(100, 200), ic3, ic4;
    ic1.Show();
    ic2.Show();
    ic3 = ++ic1; // ic3 = operator(ic1)
    ic4 = ic2++; // ic4 = operator(ic2, n)
    ic3.Show();
    ic4.Show();
}
```

注：对双目运算符，重载为成员函数时，仅一个参数，另一个被隐含；重载为友元函数时，有两个参数，没有隐含参数。
一般来说，单目运算符最好被重载为成员函数；对双目运算符最好被重载友元函数。

9.7.1 模板

模板是一个泛型编程的概念，即不考虑类型的一种编程方式，模板分为模板函数和模板类。

人们需要编写很多个形式和功能都很相似的函数来实现某些功能只因为类型不同，就需要重载，很多个版本，写起来就比较麻烦，也比较嗦，此时，就可以使用模板函数来解决这种问题。

模板函数：建立一个通用的函数，其返回值类型，形参的类型都不确定指定而是采用虚拟类型来代替，对于功能相同的函数，就无须重复写代码模板函数和普通函数长相及使用方法非常类似，唯一的区别是类型参数化。
语法：

```
    template <typename T> //T代表起的类型名，  template 关键字
T findmax(T arr[], int len)
{
T val = arr[0];
…
}
template <class T> //typename 和 class 是声明虚拟类型的关键字
```

示例：

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

template <typename T>
T findmax(T arr[], int len)
{
    T val = arr[0];
    for(int i = 0; i < len; i++)
    {
        if(arr[i] > val)
        {
            val = arr[i];
        }
    }
    return val;
}

int main()
{
    int arr[4] = {1, 5, 7, 80};
    int result1 = findmax<int>(arr, 4);
    //在使用函数名为findmax<int>时，表示把int类型带入模板
    printf("MAX = %d\n", result1);

    double abc[3] = {1.0, 5.1, 6.2};
    double result2 = findmax<double>(abc, 3);
    printf("MAX = %.1f\n", result2);

    return 0;
}
```

9.7.2 模板函数的调用：

（1）第一种：显式类型调用函数名 < 实际类型 1，实际类型 2>（参数列表）；
如：findmax<int>(1,2);
（2）第二种：隐式类型调用函数名（参数）；
如： indmax(1,2)

9.7.3 模板函数的重载：

和普通函数的重载相似、在同一个作用域，函数名相同参数列表不同的多个函数参数不同：个数不同，类型不同，顺序不同

注：（1）前提条件函数名必须相同（2）返回值不能决定调用哪个函数体（3）形参列表的不同才是决定是否重载的重要原因

9.7.4 类模板

类模板，可以定义相同的操作，拥有不同数据类型的成员属性。模板类的作用和模板函数的相同，都是为了减少代码量方便操作.

通常使用 template 来声明。告诉编译器，碰到 T 不要报错，表示一种泛型.

一个类模板 (类生成类) 允许用户为类定义个一种模式，使得类中的某些数据成员、默认成员函数的参数，某些成员函数的返回值，能够取任意类型 (包括系统预定义的和用户自定义的)。

语法:

```
    template<typename 类型参数>
    class 类名
    {
    类成员说明
    };
```

示例:

```
    template <typename T>
class Complex {
private:
    T a;
    T b;

public:
    //构造函数
    Complex(T a, T b)
    {
        this->a = a;
        this->b = b;
    }

    //运算符重载
    Complex<T> operator+(Complex &c)
    {
        Complex<T> tmp(this->a + c.a, this->b + c.b);
        cout << tmp.a << endl;
        cout << tmp.b << endl;
        return tmp;
    }
};

int main()
{
    //对象的定义，必须声明模板类型，因为要分配内容
    Complex<int> a(10, 20);
    Complex<int> b(20, 30);
    Complex<int> c = a + b;

    return 0;
}
```

注意：

1. 类模板，实际上是建立一个通用类，内部的数据成员，成员函数的返回值类型和参数类型不具体说明，用类型参数来代表。
2. 类模板，实际上是建立一个通用类，内部的数据成员，成员函数的返回值类型和参数类型不具体说明，用类型参数来代表。
3. 类模板不编译, 会在编译期根据使用方式，生成对应的类代码
4. 类模板中没有使用到的成员方法不会在编译器生成对应的指令
5. 类模板使用时必须加上模板类型参数，类模板无法自己推导类型参数

X 2024笔试真题及答案解析

西安交通大学考试题

课　程　计算机程序设计（A 卷）

学　　院　人工智能学院　　考试日期　2024 年 12 月 4

专业班号　人工智能 2401，人工智能 2402

姓　　名　　学　号　　期中　　期末

一、选择题（20×2 = 40 分）

1. 下面哪位同学对编程的描述更全面（　　）
   A. 编程就是写算法　　B. 编程就是对输入数据进行各自处理后输
   B. 编程就是编写高级语言代码　　D. 编程包括数据组织、算法设计及具体语言

2. 下面描述不正确的是（　）
   A. 机器语言是面向计算机硬件的 0、1 组成的代码　B. 高级编程语言包括机器语言
   C. 程序设计中代码注释能促进代码可读性　　D. 编译器对代码注释不

3. 如果程序员想在计算机存储 “I Love XJTU”，问这个字符串消耗内存多少个 bi
   A. 9　　B. 72　　C. 88　　D. 96

4. 若从终端上输入：**Hi, xjtu！，hello！**后回车，则程序运行结果是(　　)

```
#include <stdio.h>
void main()
{
    char c;
    c = getchar();
    while ( c!= '！ ')
    {
       putchar(c);
       c=getchar();
    }
}
```

   A. Hi　　B. Hi,　　C. Hi,xjtu　　D. Hi, xjtu！，hello！

5. 下列合法的字符常量是（　）
   B. ‘\? ’

变量 a、b、c 的最终值是（　　）

A. a= 14　　b=8　　c=13　　B. a=14　　b=8　　c=14

C. a= 13　　b=8　　c=11　　D. a= 14　　b=8　　c=12

7. 在下列条件语句中，只有一个在功能上与其他三个语句不等价（其中 s1 和 s2 表某个语句），这个不等价的语句是？（　　）

A. if (a) s1; else s2;　　B. if (!a) s2; else s1;

C if (a ! = 0) s1; else s2;　　D. if (a = = 0) s1; else s2;

8. 请请问下列代码中 while 和 do-while 循环体内代码分别执行了几次？i 分别是（　　）

```
void main()
{
   int i=0;
   int sum=0;
   int total=0;

   while(sum < 16)
          {
              sum += i;
              i++;
          }
   cout<<i<<endl;
   do
     {
              total += i;
              i++;
     }while(i <= 8);
   cout<<i<<endl;
 }
```

A. while 循环体内执行了 16 次, do-while 循环体内执行了 8 次，i 为 16，8

B. while 循环体内执行了 15 次, do-while 循环体内执行了 7 次，i 为,16，23

C. while 循环体内执行了 7 次, do-while 循环体内执行了 8 次，i 为 7，23

D. while 循环体内执行了 7 次, do-while 循环体内执行了 2 次，i 为 7，9

9. 已知字符'a'的 ASCII 码是 97，若有：int a=19,　b=3;　　float　c=2,　d;

d = a%b + a/c + b + 'q';

则执行上述表达式语句后，变量 d 的值是（　　）

西 安 交 通 大 学 考 试 题

10. 请问以下函数执行结果为 （ ）

```
int i;
int sum = 0;
 for (i = 0; i<=30; i++){
 if (i%5==0)
     {
          i=i+1;
          continue;
      }
      sum++;
 }
 cout<<sum<<'\t'<<i<<'\t';
```

A. 12, 32; B. 18, 31; C. 18, 20; D. 18,

11. 若用数组名作为函数调用的实参，传递给形参的是：（ ）

A. 数组的首地址 B. 数组中第一个元素的值

C. 数组中的全部元素的值 D. 数组元素的个数

12. 关于函数，下列说法正确的是：

A. 函数返回值不能是指针 B. 函数的形参可以是结构体

C. 函数形参不能包括引用 D.函数的返回值不能为指向函数的指

13. 如下所示的代码中，程序执行后与 a[i]不等价的是 （ ）

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

A. p[i]; B *(a+i); C *(p+i) D *p[i]

14. 定义 float a[10]={0,1,2,3,4,5,6,7,8,9}， 已知 a 的地址为 0x0018ff20, 请问&a[
&a[8]的值分别是, (float 类型在此计算机中占 4 个字节)（ ）

A. 0x0018FF28, 0x0018FF40 B. 0x0018FF22, 0x0018FF3

C. 0x0018FF24, 0x0018FF40 D. 0x0018FF24, 0x0018FF4

15.下列代码不属于函数重载的是 （ ）

A. double test1 (double x); double test1 (double & x);

B. long test2 (int n, float m); double test2(float n, float m);

C. long test3(int &n, int x); long test3(float n, float x);

16. 下列关于类与对象表述正确的是 （　　）。

A. 类是对一类数据的抽象，既能包含成员数据，又能包含对数据的操作函数

B. 类是对象的实例化，对象定义后是有具体内存空间的

C. 结构体的定义和类的定义内容是一样的

D. 面向对象的编程是由一个一个对象组成的，对象之间的调用按顺序执行的

17. 已知如下代码：

```
class A {
    float x, y;
    public:
    A( ) {x = 0; y = 0};
    A(float a, float b) {x = a; y = b;}
    A(float c) {x = c;}
    ~A( ) {}
}
A a0(100.0);
void f ()
{
    A ab(10.0, 20.0);
}
void main ()
{
    A obj;
    f( );
}
```

下列叙述中，错误的是（　　）。

A. A()　A(float a, float b)　A(float c) 三个函数均是构造函数

B. A a0(100)是全局对象，在 main 函数之前执行

C. f( )函数也是在 main 函数之前执行

D. 这段程序中 ~A( )析构函数会被调用不止 1 次

18. 关于面向对象相关概念理解不正确的是（　　）。

A. 类的继承性提供了重复利用程序资源的一种途径，包括私有、保护和公开三类继承方式，三者的最大区别是继承数据的权限不同

B. 在类的成员变量定义时，一般包括整型、浮点型及结构体等数据，新的类或对象在其内部的定义是不被允许的

C. 类的构造函数可以定义多个，且其函数名都是一模一样的，析构函数在类的运行过程中可以被多次调用

D. 类的构造函数与析构函数在一定程度上其功能可以用普通函数来实现，即定义在类外的函数实现

西 安 交 通 大 学 考 试 题

19. 已知如下代码：

```
class A {
        float x, y;
        public:
        A(float a, float b) {x = a; y = b;}
        float Getx() {return x;}
        float Gety() {return y;}
        float sum() {return x+y;}
}
float sumxy (A &a) {return a.Getx()+a.Gety()}
float sum(A &a) {(return a.x + a.y)}
void main ()
{
        A t1(1,2), t2(10,20), t3(100,200);
        cout<<t1.sum()<<endl;
        cout<<sum(t2)<<endl;
        cout<<sumxy(t3)<<endl;
}
```

下列叙述中，不正确的是（　　）。

A.该类在运行是不会调用默认的构造函数 A（）;

B.该类中的成员函数 sum 在使用时容易于类外的函数 sum 混淆，最好不要这样编程

C. sumxy 函数是一般函数，与类 A 无关

D. main 函数中 t1.sum() 和 sum(t2)分别调用了类内函数和类外的普通函数

20. 关于模版的使用描述错误的是 （　　）。

A．模板是一个泛型编程的概念，即不考虑类型的一种编程方式，模板分为模板函数和模板类

B. 模版函数的优点包括：类型无关有很高的可复用性、平台无关的，可移植

C. 类模板，实际上是建立一个通用类，内部的数据成员，成员函数的返回值类型和参数类型不具体说明，用类型参数来代表

D. 当使用类模板定义对象时，系统不会根据实参的类型去替换模板中的类型参数，而需要程序员手工做

二、程序分析、填空与设计题（共 60 分）

1. 如下程序为数据交换功能，请写出两种方法实现交换的子函数，并写出程序运行果。另外，程序中存在错误，请仔细通读该程序代码，并修改掉程序错误之处，明错误原因（10 分）

```
#include <iostream>
using namespace std;
void Swap1(int x, int y);
void Swap2(int x, int y);
int main()
{
    float  a=123.8，  b=456.9;
    cout<<"--------before swap--\n";
    cout<<"a\t"<<"\t"<<a<<endl;
    cout<<"b\t"<<"\t"<<b<<endl;

    Swap(a,b);

    cout<<"---------after swap---\n";
    cout<<"a=\t"<<"\t"<<a<<endl;
    cout<<"b=\t"<<"\t"<<b<<endl;
    return 0;
}
第一种 swap 的实现
{
    _____;
    _____;
    _____;
    _____;
}
第二种 swap 的实现
{
    _____;
    _____;
    _____;
    _____;
}
```

修改程序的一处错误为：_____

错误原因：_____

_____

_____

西 安 交 通 大 学 考 试 题

2. 阅读如下代码，支出程序中存在的错误，并用 if-else 结构写一段代码重新实现该码的功能（10 分）

```
#include <iosteam>
int main()
{
   int score;
   scanf("%d", &score);
   switch(score)
   {
        case score >= 90:
                cout<<"A\n"; break;
        case score >= 80:
                cout<<"B\n"; break;
        case score >= 70:
                cout<<"C\n"; break;
        case score >= 60:
                cout<<"D\n"; break;
        default:
                cout<<"E\n"; break;
   }

   return 0;
}
```

修改程序的错误之为：_____

错误原因说明：_____

用 if-else 结构重新编写上面的代码：

3. 如下代码,请补全程序，该代码是由用户输入学生信息到一个结构体数组（input 数），max 函数实现了在该学生结构体数组中找最高分数，然后 print 函数打印 。据代码片段补充代码，提示（1）为函数返回类型，（2）为指向结构体的指针，该码片段以指针地址传递实现，（3）（4）（5）均为函数参数传递(5 分)

```
include <stdio.h>
struct Student
{   int num;
    char name[20];
    float score[3];
    float aver;
};
void input(struct Student stu[]);
   (1)   max(struct Student stu[]);
void print(struct Student stu);
int main ( )
{
    struct Student stu[3];
    (2)      ;
    input (  (3)      );
    struct Student tmp = max(   (4)   )
    print(   (5)      );
    return 0;
 }
```

4. 给定一个整型数组 vec，其声明语句为 int vec [] = { 88, 11, 77, 44, 22}；经过调用如函数后，请写出 vec 内部值的排列情况（5 分）

```
void Mystery(int vec[])
{
     int len = 5;
     int tmp = vec[len - 1];
     for (int i = 1; i < len; i++)   {
                vec[i] = vec[i – 1];
     }
     vec[0] = tmp;
}
```


西 安 交 通 大 学 考 试 题

5. 下面的伪代码描述了对一个整型数组 vec 排序的过程，请将排序过程中数组的状态写出来，直到程序正常终止。初始状态已经给出，不一定需要用完格子（5

```
for (i = 0; i < vec size; i = i + 1)
{
    minPosition = i
    for (j = i; j < vec size; j = j + 1)
     {
        if (vec[j] < vec[minPosition])
        minPosition = j
     }
   temp = vec [i]
   vec[i] = vec[minPosition]
   vec[minPosition] = temp
   print(vec)   // 每次执行此函数时，写出当前数组内部的值
}
```


6. 请阅读如下代码，**基于注释**补全函数。（5 分）

```
class MytestClass
 {
      int i;
 public:
      MytestClass(int a=0){ i=a;}
      void Show (void){  cout<<"i="<<i<<endl;   }
        _____________________     //利用专门的功能函数进行类之间的运算
        _____________________     //利用重载运算符进行类之间的运算
 };
      _____________________//专门的功能函数的实现
      _____________________
```

```
 void main(void)
   {
        MytestClass a1(50), a2(80), a3;
        a3=a1-a2;                    //重新解释了加法，可以直接进行类的运算
        a3. Myfunc (a1,a2);          //调用专门的功能函数
        a3.Show ();
   }
```

7. 阅读如下代码，写出程序所有运行输出 （5 分）

```
class   TestMyClass
{
 private：
    float     x,y;
 public:
    TestMyClass (float a, float b){
        x=a;   y=b;   cout<<"初始化自动局部对象\n";}
    TestMyClass (){
        x=0;   y=0;   cout<<"初始化静态局部对象\n";}
    TestMyClass (float   a){
        x=a;   y=0;   cout<<"初始化全局对象\n"; }
    void   Print(void) { cout<<x<<'\t'<<y<<endl; }
};

TestMyClass   a0 (100.0);

void function(void){
    cout<<"  进入 function ()  函数\n";
    TestMyClass   a2(1,2);
    static   TestMyClass   a3; }

void main(void){
      cout<<"进入 main 函数\n";
      TestMyClass   a1(3.0,   7.0);
      cout<<"调用子函数\n";
      function ();
      function ();
      cout<<"main 函数执行结束\n";
```

西 安 交 通 大 学 考 试 题

8. 阅读如下代码，按提示补全类模版等（5 分）

```
#include <iostream>
   （1）          //模板声明
class ArrayTemTest
{
    bool m_HaveInit;              //判断该类是否已经初始化
    （2）  *m_Array;     //模版类型的成员数据
    int m_Row, m_Column       //行数
    public:
    （3）      ; //空的构造函数
    };

  （4）          //模板声明
  （5）          //构造函数声明
  {
      m_HaveInit = false;
      m_Array = NULL;
      m_Column = m_Row = 0;
  }
```

9. 请用类实现一个学生考勤管理小系统，请首选定义一个考勤系统类，
类的成员变量包括：需要考勤的 10 名学生的姓名，学号，已签到次数，和该人
成绩（其中成绩包括该学生 5 门课的成绩），
类的成员函数包括：（1）设置学生姓名、学号、已签到次数和成绩；
（2）显示学生姓名、学号、已签到次数和成绩；
（3）计算每个学生的 5 门口平均成绩并显示;
（4）对所有签到学生的签到次数排序并输入显示；
（5）使用 main 函数实例化，并依次调用成员函数完成信息
入与展示，注意所有信息通过用户输入，编写必要的与用户交互的界面代码。
出完整的 C++代码，务必注意语法不要出现错误，把题目所需功能写完善，如
功能则扣分。（10 分）

2024 程序设计笔试答案解析

1　选择题答案及解析 (20×2=40 分)

第 1 题

答案：D

解析：

- A 选项：编程不仅是写算法，还需数据组织和代码实现，描述片面；
- B 选项：仅提及“输入数据处理后输出”，未涵盖算法设计、数据结构等核心环节，不全面；
- C 选项：编写高级语言代码只是“具体实现”环节，忽略数据组织和算法设计，不完整；
-D 选项：编程的核心流程包括**数据组织**（如选择合适的数据结构）、**算法设计**（解决问题的步骤）和**具体语言实现**（用代码落地），描述最全面。

第 2 题

答案：B

解析：

- A 选项：机器语言是直接面向硬件的二进制代码（0 和 1），表述正确；
- B 选项：高级语言与机器语言是不同层级的语言，机器语言是低级语言，高级语言需编译/解释为机器语言才能执行，二者无“包含”关系，表述错误；
- C 选项：代码注释用于说明逻辑，不影响程序执行，但能提升可读性，表述正确；
- D 选项：编译器仅处理可执行代码，忽略注释内容，表述正确。D

第 3 题

答案：D

解析：C 字符串中共有 9 个字母，2 个空格，另外字符串结尾系统自动添加'\0'这个字符，共 12 个字符，一个字符占一个 byte, 一个 byte 等于 8 个 bit，故 8*12=96

第 4 题

答案：C

解析：

- 程序逻辑：用 getchar() 逐字符读取输入，putchar() 输出字符，直到遇到’!’ 停止；
- 输入数据：“Hi, xjtu!,hello!”，读取顺序为’H’→’i’→’,’→’ ’→’x’→’j’→’t’→’u’→’!’，遇到’!’ 时循环终止；
- 输出 Hi,xjtu

第 5 题

答案：B

解析：

- 字符常量规则：用单引号（' '）包裹单个字符，不可为字符串或多个字符；
- A 选项 N：无单引号，不是字符常量；
- B 选项 \? ：符合字符常量规则；
- C 选项 'hello'：双引号包裹多个字符，是字符串常量，不是字符常量；
- D 选项 "xjtu"：双引号包裹，是字符串常量，不是字符常量。

第 6 题

答案：A

解析：

- 表达式逻辑：三目运算符 "(条件) ? 表达式 1 : 表达式 2"，条件为真时执行表达式 1，否则执行表达式 2；
- 初始值：a=13，b=8，c=9；
- 条件判断：b==8 为真，执行表达式 1 "(c++, a++)"（逗号表达式，从左到右执行，最终结果为 a++ 的值）；
- 运算过程：

1. c++：c 先参与运算，后自增，此时 c=9（参与运算后变为 10）；
2. a++：a 先参与运算，后自增，此时 a=13（参与运算后变为 14）；
3. 三目运算符结果为 a++ 的值（13），赋值给 c，故 c 最终为 13；

- 最终值：a=14，b=8，c=13，故选 A。

第 7 题

答案：D

解析：

- 核心逻辑：判断 "a 是否为真（非 0）"，真则执行 s1，假则执行 s2；
- A 选项：if(a) s1; else s2 → a 非 0 执行 s1，0 执行 s2；
- B 选项：if(!a) s2; else s1 → a 为 0 执行 s2，非 0 执行 s1（与 A 等价）；
- C 选项：if(a!=0) s1; else s2 → a 非 0 执行 s1，0 执行 s2（与 A 等价）；
- D 选项：if(a==0) s1; else s2 → a 为 0 执行 s1，非 0 执行 s2（与 A 相反，不等价）。

第 8 题

答案：D

解析：

- 先分析 while 循环（sum < 16）：
- 初始：i=0，sum=0；
- 循环 1：sum += 0 → sum=0，i=1；
- 循环 2：sum +=1 → sum=1，i=2；
- ... 循环 7：sum +=6 → sum=0+1+2+3+4+5+6=21 16，循环终止；
- 结论：while 循环执行 7 次，i=7。

- 再分析 do-while 循环（i <=8）：
- 初始 i=7，do-while 先执行循环体再判断；
- 循环 1：total +=7 → total=7，i=8；
- 循环 2：total +=8 → total=15，i=9；
- 判断 i=9 >8，循环终止；
- 结论：do-while 执行 2 次，i=9。
- 最终：while 执行 7 次，do-while 执行 2 次，i=7（while 结束）→ 9（do-while 结束），故选 D。

第 9 题

答案：A

解析：

- 已知条件：`'a'ASCII=97 → 'q'ASCII=97+16=113；a=19，b=3，c=2（float）；`
- 表达式拆解：`d = a%b + a/c + b + 'q';`
- 分步计算：

1. `a%b：19%3=1`（整数取余）；2. a/c：19/2=9.5（float 除法）；
3. b：3（整数）；
4. 'q'：113（ASCII 值）；
- 总和：1 + 9.5 + 3 + 113 = 126.5，故选 A。

第 10 题

答案：B

解析：步骤 2：模拟循环过程

循环中 i 的取值及处理：

1. i 是 5 的倍数时： i=0、5、10、15、20、25、30 ，共 7 次：

- 每次执行 i=i+1 （ i 变为 1、6、11、16、21、26、31 ），再 continue ，跳过 sum++ 。
- 注意： i=30 时，执行 i=30+1=31 后，循环条件 i<=30 不成立，循环终止。

2. i 不是 5 的倍数时：总循环次数（ 0 30 共 31 次）- 5 的倍数次数（7 次）= 24 次？实际需考虑 i 额外 +1 的影响：

- 当 i 是 5 的倍数时， i 会额外 +1，因此后续 i 的取值会“跳过”一个数（如 i=0 → i=1 ，跳过了 i=0 的 sum++ ，且下一次循环 i 从 2 开始）。
- 最终实际参与 sum++ 的次数是 18 次（ i 从 0 到 30 的过程中，共 18 次非 5 倍数且未被 continue 跳过的情况）。

步骤 3：最终值

- sum ：共执行 18 次 sum++ ，结果为 18 ；
- i ：最后一次 i=30 （5 的倍数），执行 i=30+1=31 后，循环终止， i 最终为 31 。

因此，执行结果是 sum=18，i=31 ，答案选 B。

第 11 题

答案：A

解析：

- 数组名作为函数实参时，传递的是**数组首元素的地址**（即指针），而非数组元素值或个数；

- A：数组首地址，正确；
- B：首元素值，错误；
- C：全部元素值，错误；
- D：元素个数，错误（需单独传递个数）。

第 12 题

答案：B

解析：

- A：函数返回值可以是指针（如 int* func()），错误；
- B：函数形参可以是结构体（如 void func(struct Student s)），正确；
- C：函数形参可以是引用（如`void func(int& x)`），错误；
- D：函数返回值可以是函数指针（如 int (*func())(int)），错误。

第 13 题

答案：D

解析： - 已知：int a[5]=0,1,2,3,4，int *p=a（p 指向 a 首地址）；

- 等价关系：a[i] = *(a+i) = *(p+i) = p[i]（数组下标与指针偏移等价）；
- A：p[i]，与 a[i] 等价；
- B：*(a+i)，与 a[i] 等价；
- C：*(p+i)，与 a[i] 等价；
- D：*p[]，p 是一级指针，*p[] 表示指针数组（二级指针），语法错误且不等价。

第 14 题

答案：A

解析：

- 数组地址计算：数组元素地址 = 首地址 + 下标 × 元素字节数；
- 已知：a 首地址 =0x0018ff20，float 占 4 字节；
- 计算：`&a[2]`应为 a+4*2=0x0018FF28;`&a[8]`应为 a+4*8=0x0018FF40
- 选 A

第 15 题

答案：A

解析： - 函数重载规则：函数名相同，**参数个数不同**或**参数类型不同**（与返回值无关，与参数引用/值传递无关）；

- A：`double test1(double x)`与`double test1(double &x)` → 参数类型均为 double, 不构成重载；
- B：long test2(int n, float m) 与 double test2(float n, float m) → 参数类型不同（int vs float），构成重载；
- C：`long test3(int &n, int x)`与`long test3(float n, float x)` → 参数类型不同（int vs float），构成重载；
- D：char* test4(const char *str, unsigned n) 与 char* test4(const char *str) → 参数数量不同，构成重载。

第 16 题

答案：A

解析：

- A：类是对数据和操作的抽象（封装数据和成员函数），正确；
- B：对象是类的实例化，类无内存空间，对象有内存空间，表述颠倒，错误；
- C：结构体（C++ 中）默认访问权限为 public，类默认访问权限为 private，内容不同，错误；
- D：面向对象编程是“对象交互”，而非“顺序执行”，错误。

第 17 题

答案：C

解析： - 代码逻辑：类 A 有 3 个构造函数，全局对象 a0，函数 f() 中局部对象 ab，main() 中局部对象 obj；

- A：三个函数均为构造函数（与类名相同，无返回值），正确；
- B：全局对象 a0 在 main() 前初始化，正确；
- C：f() 函数是普通函数，在 main() 中调用时执行，不在 main() 前执行，错误；
- D：析构函数调用次数 = 对象个数（a0、obj、ab 共 3 个），调用不止 1 次，正确。

第 18 题

答案：C

- A：继承分 public、protected、private，区别是子类对父类成员的访问权限，正确；
- B：类的成员变量不能是自身类的对象（可是指针/引用），正确；
- C：析构函数仅在对象销毁时调用 1 次（每个对象 1 次），不能多次调用，错误；
- D：构造/析构函数的功能可通过普通函数实现（如 init()/destroy()），正确。

第 19 题

答案：C

解析：

- 代码逻辑：类 A 有带参构造函数，成员函数 sum()，类外函数 sumxy() 和 sum()；
- A：类 A 无默认构造函数，创建对象时用带参构造，不调用默认构造，正确；
- B：类内 sum() 与类外 sum() 同名，易混淆，不推荐，正确；
- C：sumxy() 函数参数为`A&`（类 A 的引用），与类 A 相关（操作类 A 的成员），错误；
- D：t1.sum() 调用类内成员函数，sum(t2) 调用类外普通函数，正确。

第 20 题

答案：D

解析：

- A：模板分函数模板和类模板，泛型编程，正确；
- B：函数模板可复用、可移植，正确；
- C：类模板用类型参数表示数据/返回值类型，正确；

- D：类模板实例化时，系统会根据实参类型自动推导并替换类型参数（如 Array<int> arr），无需手工替换，错误。

2 程序分析、填空与设计题答案及解析 (共 60 分)

第 1 题：数据交换函数（10 分）

程序错误修正

错误：数据类型不匹配：Swap1、Swap2 形参为 int，但实参为 float（a=123.8，b=456.9），应将形参改为 float

交换函数实现

1. 引用传递实现（**Swap1**）：

```
void Swap(float &x, float &y)
{
 float temp = x;
 x = y;
 y = temp;
}
```

2. 指针传递实现（**Swap2**）：

```
void Swap(float *x, float *y)
{
 float temp = *x;
 *x = *y;
 *y = temp;
}
```

程序输出结果

```
----before swap--
a=      123.8
b=      456.9
----after swap--
a=      456.9
b=      123.8
```

第 2 题：switch 语句错误修正（10 分）

程序错误修正

- **错误：case 表达式非法：**switch 的 case 表达式必须是“常量表达式”（如 90、80），不能是“score>=90”（布尔表达式）。

if-else 重写代码

```
#include <iostream>
using namespace std;
int main() {
 int score;
 cin >> score; // 或scanf("%d", &score);
 if (score >= 90) {
  cout << "A\n";
 } else if (score >= 80) {
  cout << "B\n";
 } else if (score >= 70) {
  cout << "C\n";
 } else if (score >= 60) {
  cout << "D\n";
 } else {
  cout << "E\n";
 }
 return 0;
}
```

第 3 题：结构体数组函数填空（5 分）

填空答案

1. (1) **struct Student**（max 函数返回结构体类型）；
2. (2) **struct Student *p = stu**（定义指向结构体数组的指针）；
3. (3) **stu**（input 函数参数为结构体数组名）；
4. (4) **stu**（max 函数参数为结构体数组名）；
5. (5) **tmp**（print 函数参数为 max 返回的结构体变量）。

完整代码片段

```
#include <stdio.h>
struct Student {
 int num;
 char name[20];
 float score[3];
```

```
 float aver;
};
void input(struct Student stu[]);
struct Student max(struct Student stu[]); // (1)
void print(struct Student stu);

int main() {
 struct Student stu[3];
 struct Student *p = stu; // (2)
 input(p); // (3)
 struct Student tmp = max(p); // (4)
 print(tmp); // (5)
 return 0;
}
```

第 4 题：数组排列结果（5 分）

答案：vec = 22, 88, 88, 88, 88

解析： - 函数逻辑：1. len=5，tmp=vec[4]（最后一个元素 22）；

2. for(i=1; i<5; i++)：vec[i] = vec[i-1]（元素向后复制）；

- i=1：vec[1] = vec[0] → 88→88；
- i=2：vec[2] = vec[1] → 11→88；
- i=3：vec[3] = vec[2] → 77→88；
- i=4：vec[4] = vec[3] → 44→88；

3. vec[0] = tmp（22）；

- 最终数组：22, 88, 88, 88, 88。

第 5 题：选择排序中间状态（5 分）

初始数组：[9, 4, 8, 5, 1, 2, 3, 7, 6]

排序步骤（选择排序：每次选最小元素放当前位置）：1. i=0：minPosition=4（元素 1），交换 vec[0] 和 vec[4] → [1, 4, 8, 5, 9, 2, 3, 7, 6]；

2. i=1：minPosition=5（元素 2），交换 vec[1] 和 vec[5] → [1, 2, 8, 5, 9, 4, 3, 7, 6]；
3. i=2：minPosition=6（元素 3），交换 vec[2] 和 vec[6] → [1, 2, 3, 5, 9, 4, 8, 7, 6]；
4. i=3：minPosition=5（元素 4），交换 vec[3] 和 vec[5] → [1, 2, 3, 4, 9, 5, 8, 7, 6]；
5. i=4：minPosition=5（元素 5），交换 vec[4] 和 vec[5] → [1, 2, 3, 4, 5, 9, 8, 7, 6]；
6. i=5：minPosition=8（元素 6），交换 vec[5] 和 vec[8] → [1, 2, 3, 4, 5, 6, 8, 7, 9]；
7. i=6：minPosition=7（元素 7），交换 vec[6] 和 vec[7] → [1, 2, 3, 4, 5, 6, 7, 8, 9]；
8. i=7：minPosition=7（元素 8），无交换 → [1, 2, 3, 4, 5, 6, 7, 8, 9]；
9. i=8：minPosition=8（元素 9），无交换 → [1, 2, 3, 4, 5, 6, 7, 8, 9]。

第 6 题：类运算函数补全（5 分）

专门的功能函数（Myfunc）

```
// 专门的功能函数声明
void Myfunc(MytestClass a1, MytestClass a2);
// 重载减法运算符声明（成员函数形式）
MytestClass operator-(const MytestClass& other);
```

重载减运算符（operator-）

```
// 专门的功能函数实现
void MytestClass::Myfunc(MytestClass a1, MytestClass a2)
{
 this->i = a1.i - a2.i;
}
// 重载减法运算符的函数实现
MytestClass MytestClass::operator-(const MytestClass& other) {
 MytestClass temp;
 temp.i = this->i - other.i;
 return temp;
```

完整类代码

```
#include<bits/stdc++.h>
using namespace std;
class MytestClass{
 int i;
 public:
 MytestClass(int a=0){i = a;}
 void Show(void){cout<<"i="<<i<<endl;}
 void Myfunc(MytestClass & a1,MytestClass & a2);
 friend MytestClass operator-(const MytestClass & a1,const MytestClass & a2);
};
void MytestClass::Myfunc(MytestClass & a1,MytestClass & a2){
 i = a1.i - a2.i;
}
MytestClass operator-(const MytestClass & a1,const MytestClass & a2){
 MytestClass t;
 t.i=a1.i-a2.i;
 return t;
}
int main(){
```

```
 MytestClass a1(50),a2(80),a3;
 a3 = a1-a2;
 a3.Show();
 a3.Myfunc(a1,a2);
 a3.Show();
}
```

第 7 题：程序运行输出（5 分）

输出顺序（按对象初始化和函数调用顺序）：

1. 全局对象 a0 初始化 → 调用 TestMyClass(float a) → 输出“初始化全局对象”；
2. 进入 main() → 输出“进入 main 函数”；
3. main() 中 a1 初始化 → 调用 TestMyClass(float a, float b) → 输出“初始化自动局部对象”；
4. 调用子函数 → 输出“调用子函数”；
5. 进入 function() → 输出“进入 function () 函数”；
6. function() 中 a2 初始化 → 调用 TestMyClass(float a, float b) → 输出“初始化自动局部对象”；
7. function() 中 a3（静态局部对象）初始化 → 调用 TestMyClass() → 输出“初始化静态局部对象”；
8. 第一次 function() 结束（a2 销毁，a3 不销毁）；
9. 第二次调用 function() → 输出“进入 function () 函数”；
10. function() 中 a2 初始化 → 输出“初始化自动局部对象”；
11. a3 已初始化（静态对象仅初始化 1 次），不输出；
12. 第二次 function() 结束（a2 销毁）；
13. main() 执行结束 → 输出“main 函数执行结束”；
14. 全局对象 a0、main() 中 a1、function() 中 a3 销毁（析构函数无输出）。

完整输出：

```
初始化全局对象
进入main函数
初始化自动局部对象
调用子函数
进入function()函数
初始化自动局部对象
初始化静态局部对象
进入function()函数
初始化自动局部对象
main函数执行结束
```

第 8 题：类模板补全（5 分）

填空答案

1. (1) **template <typename T>**（模板声明，T 为类型参数）；

2. (2) **T**（成员数据为模板类型指针）；
3. (3) **ArrayTemTest()**（空构造函数声明）；
4. (4) **template <typename T>**（构造函数的模板声明）；
5. (5) **ArrayTemTest<T>::ArrayTemTest()**（构造函数定义）。

完整类模板代码

```
#include <iostream>
template <typename T> // (1)
class ArrayTemTest {
 bool m_HaveInit;
 T *m_Array; // (2)
 int m_Row, m_Column;
 public:
 ArrayTemTest(); // (3)
};

template <typename T> // (4)
ArrayTemTest<T>::ArrayTemTest() { // (5)
 m_HaveInit = false;
 m_Array = NULL;
 m_Column = m_Row = 0;
}
```

第 9 题：学生考勤管理系统（10 分）

完整 C++ 代码：

```
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;

// 学生考勤系统类
class StudentAttendance {
 private:
 // 成员变量：10名学生的信息
 string name[10];    // 姓名
 string id[10];      // 学号
 int attendCount[10];// 已签到次数
 float scores[10][5];// 5门课成绩
 int studentNum = 10;// 学生人数（固定10人）
```

```
public:
// (1) 设置学生信息
void setStudentInfo() {
 for (int i = 0; i < studentNum; i++) {
  cout << "===== 输入第" << i+1 << "名学生信息 =====" << endl;
  cout << "姓名: ";
  cin >> name[i];
  cout << "学号: ";
  cin >> id[i];
  cout << "已签到次数: ";
  cin >> attendCount[i];
  cout << "5门课成绩（空格分隔）: ";
  for (int j = 0; j < 5; j++) {
   cin >> scores[i][j];
  }
  cout << endl;
 }
}

// (2) 显示学生信息
void showStudentInfo() {
 cout << "===== 所有学生信息 =====" << endl;
 for (int i = 0; i < studentNum; i++) {
  cout << "第" << i+1 << "名学生: " << endl;
  cout << "姓名: " << name[i] << "\t学号: " << id[i] << "\t已签到次数: " << attendCount[i] << endl;
  cout << "5门课成绩: ";
  for (int j = 0; j < 5; j++) {
   cout << scores[i][j] << " ";
  }
  cout << endl << "------------------------" << endl;
 }
}

// (3) 计算并显示每个学生的平均成绩
void calculateAverageScore() {
 cout << "===== 学生平均成绩 =====" << endl;
 for (int i = 0; i < studentNum; i++) {
  float sum = 0;
  for (int j = 0; j < 5; j++) {
   sum += scores[i][j];
  }
  float average = sum / 5;
```

```
    cout << name[i] << "（学号: " << id[i] << "）的平均成绩: " << average << endl;
   }
   cout << endl;
  }

  // (4) 对签到次数排序并显示（按签到次数降序，保留学生信息关联）
  void sortAttendCount() {
   // 定义结构体存储学生信息（用于排序）
   struct Student {
    string name;
    string id;
    int attendCount;
   };
   Student stu[10];
   // 复制信息到结构体数组
   for (int i = 0; i < studentNum; i++) {
    stu[i].name = name[i];
    stu[i].id = id[i];
    stu[i].attendCount = attendCount[i];
   }
   // 排序（降序）
   sort(stu, stu + studentNum, [](Student a, Student b) {
    return a.attendCount > b.attendCount;
   });
   // 显示排序结果
   cout << "===== 签到次数排序（降序） =====" << endl;
   for (int i = 0; i < studentNum; i++) {
    cout << "姓名: " << stu[i].name << "\t学号: " << stu[i].id << "\t签到次数: "
    << stu[i].attendCount } <<  endl;
   cout << endl;
  }
};

// (5) main函数实例化并调用
int main() {
 StudentAttendance system;
 int choice;
 while (true) {
  // 交互界面
  cout << "===== 学生考勤管理系统 =====" << endl;
  cout << "1. 录入学生信息" << endl;
  cout << "2. 显示学生信息" << endl;
```

```
    cout << "3. 计算平均成绩" << endl;
    cout << "4. 签到次数排序" << endl;
    cout << "5. 退出系统" << endl;
    cout << "请输入操作序号: ";
    cin >> choice;
    cout << endl;

    // 选择操作
    switch (choice) {
     case 1:
     system.setStudentInfo();
     break;
     case 2:
     system.showStudentInfo();
     break;
     case 3:
     system.calculateAverageScore();
     break;
     case 4:
     system.sortAttendCount();
     break;
     case 5:
     cout << "退出系统成功! " << endl;
     return 0;
     default:
     cout << "输入错误, 请重新选择! " << endl;
     break;
    }
   }
  }
```

功能说明：

1. 录入学生信息：通过循环输入 10 名学生的姓名、学号、签到次数和 5 门课成绩；
2. 显示学生信息：遍历输出所有学生的完整信息；
3. 计算平均成绩：对每个学生的 5 门课成绩求和后取平均并显示；
4. 签到次数排序：用结构体存储学生信息，调用 sort 函数按签到次数降序排序并显示；
5. 交互界面：通过 while 循环和 switch 提供菜单选择，支持重复操作。

XI 易错点和库函数附录表

1. 易错点提醒

错误	描述	建议
编码格式错误	使用了中文符号如逗号，引号，括号等	建议先本地调试排除问题
变量未赋值	变量没有赋予初值，导致接下来运算时具有随机初值出错	养成初值设定习惯如 `int x;`
输出错误	小数精度错误，空格数量错误，输出内容不符要求	建议直接 copy 样例输出，在其基础上使用 printf 改为自己输出
读取鲁棒性差	读取时不能面对所有情形，如英语人名中间带空格，不能仅使用 cin 读取	在输入内容可能有空格隔开时，使用 getline 或者 getchar 读取输入
数组越界错误	下标超过数组范围（注意第一个元素下标为 0）	情况允许时可使用略大的数组存储，使用遍历时检查开始结尾的下标正确性
指针未设置空	未设置为空的指针会导致意想不到的错误	使用前赋值为 nullptr；在删除指针后也要及时赋为 nullptr
循环条件错误	循环条件设置错误可能导致死循环；结果是程序一直运行；或者条件未考虑完全，导致漏了起始或终止情况	while 循环如果用变量标志结束，记得更新变量；检查循环的起始和终止条件。
分号逗号错误	`for(int i=0;i<n;i++);int a,b;//`其中符号可能出错	逗号一般表示地位相同内容（如参数），分号一般分割语句。
等号错误	连等号才表示相等含义，等号表示赋值；由于 c++ 中的浮点数并不精准，不建议用 `0.1==0.05+0.05`。	采用两者之差小于一个较小值来比较 `abs(0.3-(0.1+0.2))<1e-5`
比较语法错误	连续使用比较符号:`if(0<=a<=10)`，这只在 python 有效	c++ 只用一侧的比较符号
除法问题	`a/b` 其中 a,b 均为整型则整除，结果为整型；任一为浮点型则结果为浮点型	对于变量采用 double 另赋值；对于常数直接加小数点（1.0/3）

2 库函数附录 (include< 函数名 >)

2.1 注意点

1. 在使用标准库函数之前，注意使用 using namespace std; 否则在调用函数时记得使用 std:: 指明函数命名空间，说明函数是标准库中的函数。2. 推荐本地下载 c++ 参考手册，可以直接根据要用的具体函数找到库函数。

2.2 万能库函数

对于 GCC 编译器：include<bits\stdc++.h> 包含了大多数 c++ 中的标准库函数

2.2.1 使用说明

1. 该头文件包含了绝大多数 c++ 中的标准库函数，在竞赛中常常使用。
2. 适合在编译时间不够或者记不起标准库函数时使用。
3. 因为其包含过多文件，导致占用空间以及加载时间过多，在较大的工程中不推荐使用。

2.3 输入输出库 (<iostream>)

Table 2: 输入输出函数汇总

函数原型	功能描述	示例代码
cin >> var	从标准输入读取数据	int x; cin >> x;
cout << expr	向标准输出写入数据	cout << "Hello: " << x;
cerr << expr	向标准错误输出写入	cerr << "Error message";
getline(cin, str)	读取整行文本	string s; getline(cin, s);

2.4 字符串处理库 (<string>)

Table 3: 字符串处理函数汇总

函数原型	功能描述	示例代码
str.length()	返回字符串长度	string s="hello"; int len=s.length();
str.size()	同 length()	int size = s.size();
str.empty()	检查字符串是否为空	if(s.empty()) cout << " 空字符串";
str.substr(pos, len)	提取子字符串	string sub = s.substr(1, 3);
str.find(substr)	查找子字符串位置	int pos = s.find("ell");
str.append(str2)	追加字符串	s.append(" world");
str.replace(pos, len, str2)	替换子字符串	s.replace(0, 1, "H");
str.c_str()	转换为 C 风格字符串	const char* cstr = s.c_str();
to_string(val)	数值转字符串	string num = to_string(123);
stoi(str)	字符串转整数	int x = stoi("456");
stod(str)	字符串转双精度浮点数	double d = stod("3.14");

2.5 数学运算库 (<cmath>)

Table 4: 数学函数汇总

函数原型	功能描述	示例代码
sqrt(x)	平方根	double r = sqrt(16.0); // 4.0
pow(x, y)	x 的 y 次幂	double p = pow(2, 3); // 8.0
abs(x)	绝对值（整数）	int a = abs(-5); // 5
fabs(x)	绝对值（浮点数）	double f = fabs(-3.14); // 3.14
ceil(x)	向上取整	double c = ceil(3.2); // 4.0
floor(x)	向下取整	double f = floor(3.9); // 3.0
round(x)	四舍五入	double r = round(3.6); // 4.0
sin(x), cos(x), tan(x)	三角函数	double s = sin(3.14159/2);
log(x), log10(x)	自然对数、常用对数	double l = log10(100); // 2.0
exp(x)	e 的 x 次幂	double e = exp(1); // 约 2.718

2.6 算法库 (<algorithm>)

Table 5: 算法函数汇总

函数原型	功能描述	示例代码
sort(begin, end)	排序	sort(arr, arr+5);
find(begin, end, val)	查找元素	auto it = find(v.begin(), v.end(), 5);
reverse(begin, end)	反转序列	reverse(s.begin(), s.end());
max(a, b), min(a, b)	最大/最小值	int m = max(3, 7); // 7
swap(a, b)	交换两个值	swap(x, y);
count(begin, end, val)	统计出现次数	int c = count(v.begin(), v.end(), 2);
copy(src_begin, src_end, dest)	复制序列	copy(a, a+5, b);
fill(begin, end, val)	填充序列	fill(arr, arr+10, 0);

2.7 容器操作 (<vector>, <array>)

Table 6: 容器操作函数汇总

函数原型	功能描述	示例代码
v.push_back(val)	向向量末尾添加元素	v.push_back(10);
v.pop_back()	删除向量末尾元素	v.pop_back();
v.size()	返回元素数量	int n = v.size();
v.empty()	检查是否为空	if(v.empty()) ...
v.clear()	清空所有元素	v.clear();
v.begin(), v.end()	返回迭代器	for(auto it=v.begin(); it!=v.end(); ++it)
v.front(), v.back()	首尾元素引用	int first = v.front();
v.insert(pos, val)	在指定位置插入	v.insert(v.begin()+1, 5);
v.erase(pos)	删除指定位置元素	v.erase(v.begin()+2);

2.8 实用函数 (<cstdlib>, <ctime>)

Table 7: 实用函数汇总

函数原型	功能描述	示例代码
rand()	生成随机数	int r = rand() % 100;
srand(seed)	设置随机种子	srand(time(0));
time(0)	获取当前时间戳	long t = time(0);
exit(code)	退出程序	exit(0);
system(command)	执行系统命令	system("pause");

2.9 补充库函数

Table 8: 补充函数汇总

头文件/函数原型	功能描述	示例代码
<cctype>	包含以下字符处理函数	
isalpha(c)	判断字符是否为字母	if(isalpha(ch)) cout << "是字母";
isdigit(c)	判断字符是否为数字	if(isdigit(ch)) cout << "是数字";
toupper(c)	转换为大写字母	char upper = toupper('a'); // 'A'
tolower(c)	转换为小写字母	char lower = tolower('A'); // 'a'
<iomanip>	输出流操纵算子函数，包含 setprecision 等常用函数（设置输出精度）	
setprecision(n)	设置浮点数精度	cout << setprecision(2) << 3.14159; // 3.14
fixed	固定小数点表示	cout << fixed << setprecision(2) << 3.14159;
scientific	科学计数法表示	cout << scientific << 1234.56;
setw(n)	设置字段宽度	cout << setw(10) << "Hello";