《计算机语言与程序设计》
复 习

清华大学 自动化系
索 津 莉
2024年6月21日

本节课教学目标

回顾，备考。

四个月过去了

```
#include <stdio.h>                                          /*文件包含*/

void  main( )                                               /*主函数 */
{                                                           /*函数体开始*/
        printf ( “I’ll be a professional programmer in 4 months!\n");   /*输出语句*/
}                                                           /*函数体结束*/
```

```
#include <stdio.h>

void  main( )
 {
            printf ( “I am a professional programmer now!\n");
}
```

四个月过去了

- 会调程序了么
- 找到程序的乐趣了么
- 可以应对考试了么
- 是一个程序员了么

本节课主要内容

- **程序设计流程**
- C语言知识点回顾
- C++语言知识点回顾
- 几道面试题
- 备考事项

程序设计流程

1. 定义目标

2. 设计程序

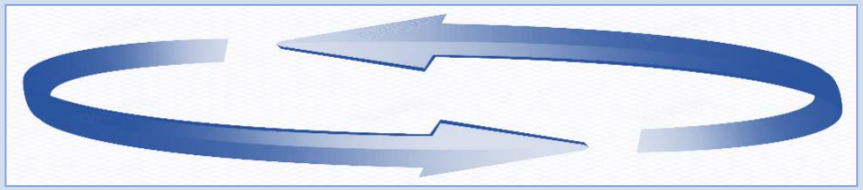
3. 编写代码

4. 编译链接

5. 运行程序

6. 测试调试

7. 维护修改



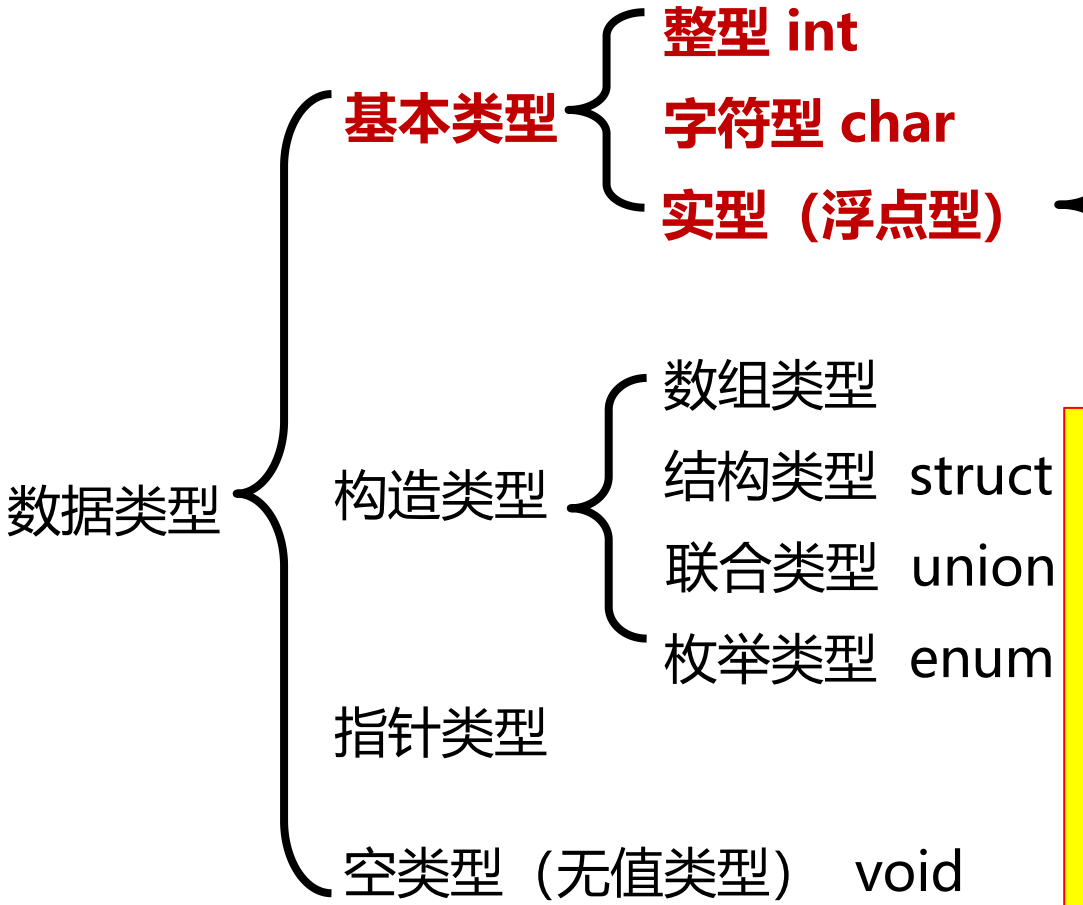
程序设计流程

- 编程中的“二八定律”

本节课主要内容

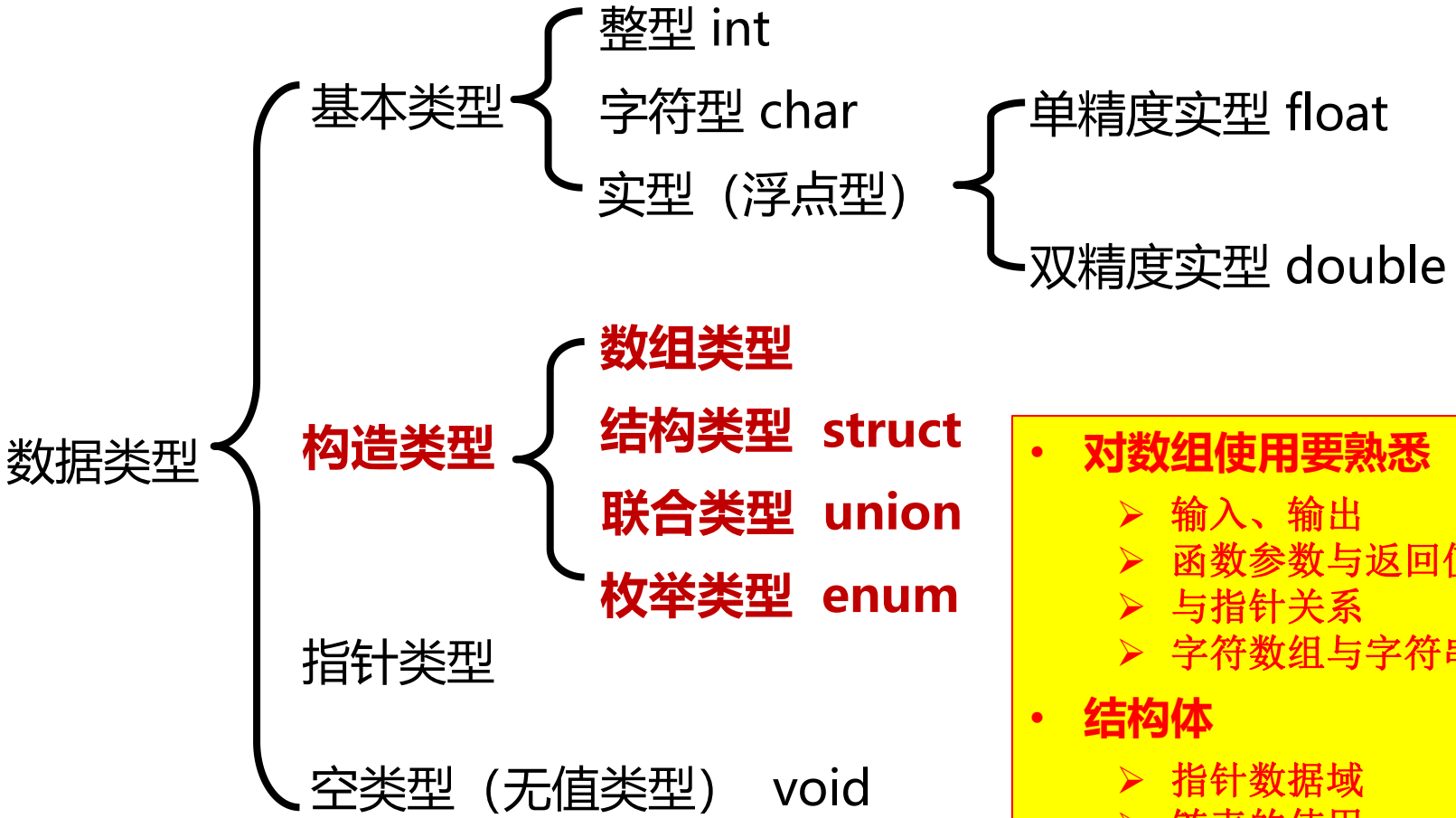
- 程序设计流程
- **C语言知识点回顾**
- C++语言知识点回顾
- 几道面试题
- 备考事项

1. 基础数据类型



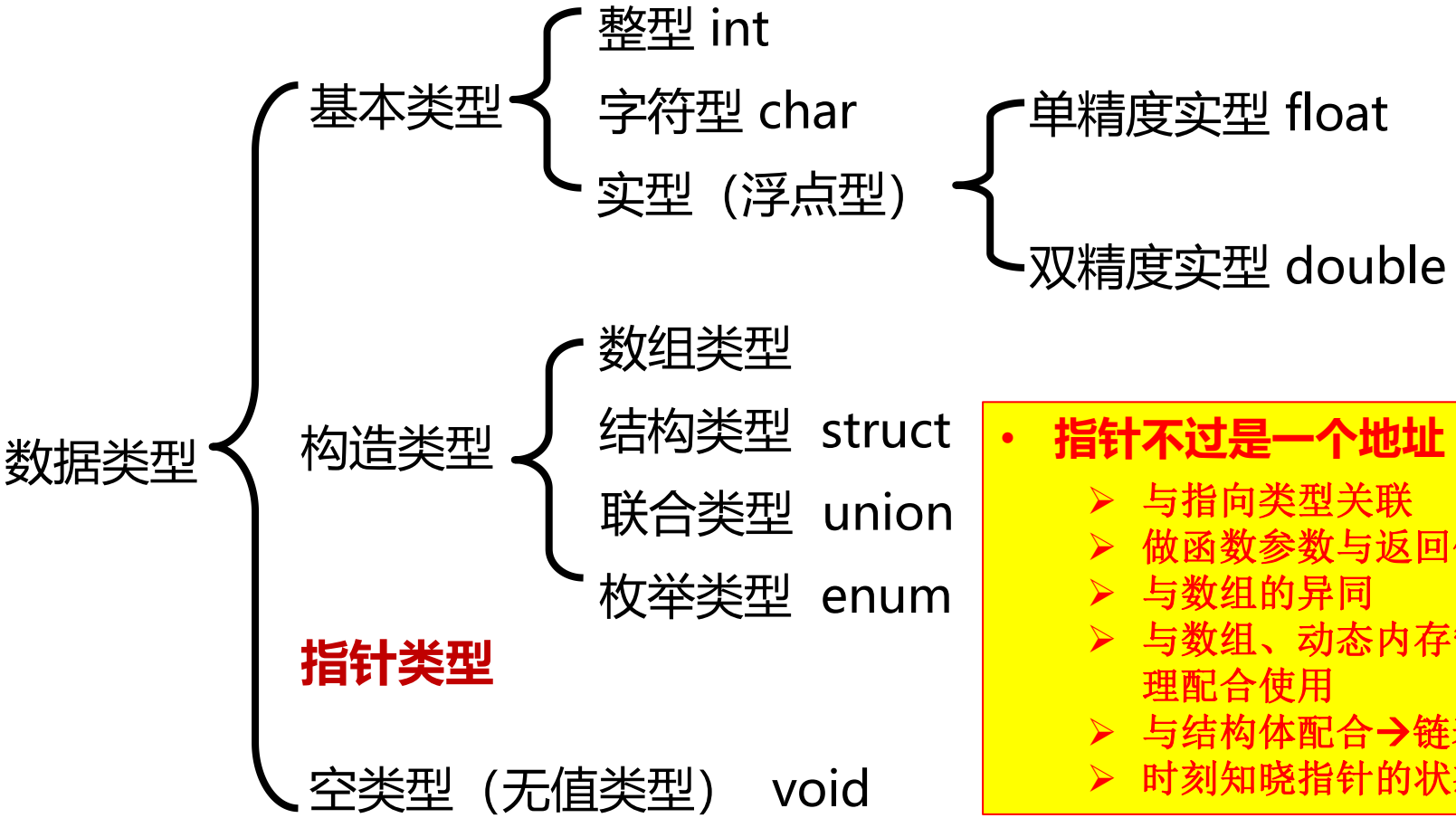
- **对整数符号、取值范围要敏感！**
- **字符在内存的存储**
  - 区分空格、空字符0、‘0’与“0”
- **如何判断两个浮点数是否相等?**
- **对浮点精度不要过高估计**

1. 基础数据类型



- **对数组使用要熟悉**
  - 输入、输出
  - 函数参数与返回值
  - 与指针关系
  - 字符数组与字符串
- **结构体**
  - 指针数据域
  - 链表的使用

1. 基础数据类型



- **指针不过是一个地址**
  - 与指向类型关联
  - 做函数参数与返回值
  - 与数组的异同
  - 与数组、动态内存管理配合使用
  - 与结构体配合→链表
  - 时刻知晓指针的状态

2. 格式化输入输出

格式字符	说　明	代码	输出
d,i	以带符号的十进制形式输出整数（正数不输出符号）	int a=567; printf (“%d”,a);	567
o	以八进制无符号形式输出整数（不输出前导符0）	int a=65; printf(“%o”,a);	101
x,X	以十六进制无符号形式输出整数（不输出前导符0x）,用x则a～f小写输出，用X则大写输出	int a=255; printf(“%x”,a);	ff
u	以无符号十进制形式输出整数	int a=567; printf(“%u”,a);	567
c	以字符形式输出，只输出一个字符	char a=65; printf(“%c”,a);	A
s	输出字符串	printf(“%s”, “ABC”);	ABC
f	以小数形式输出实数，隐含输出6位小数	float a=567.789; printf(“%f”,a);	567.789000
e,E	以指数形式输出实数	float a=567.789; printf(“%e”,a);	5.677890e+02
g,G	选用%f或%e格式中宽度较短的一种格式，不输出无意义的0	float a=567.789; printf(“%g”,a);	567.789

2. 格式化输入输出

printf

修饰符	功能
m	输出数据域宽：数据长度<m,左补空格;否则按实际输出
.n	对实数，小数点后位数(**四舍五入**)
	对字符串，实际输出位数
-	在域内左对齐（缺省右对齐)
+	有符号数的正数前显示(+)
0	左面不使用的空位置自动填0
#	显示八进制和十六进制前导符
l	在d,o,x,u前，输出精度为long型
	在e,f,g前，输出精度为double型

printf vs. scanf

printf	scanf
%[flags][width][.prec][hlL]type % [标志][输出最小宽度][.精度][长度]类型	%[flag]type
输出的字符数	读入的项目数
变量/常量/表达式的列表	地址列表

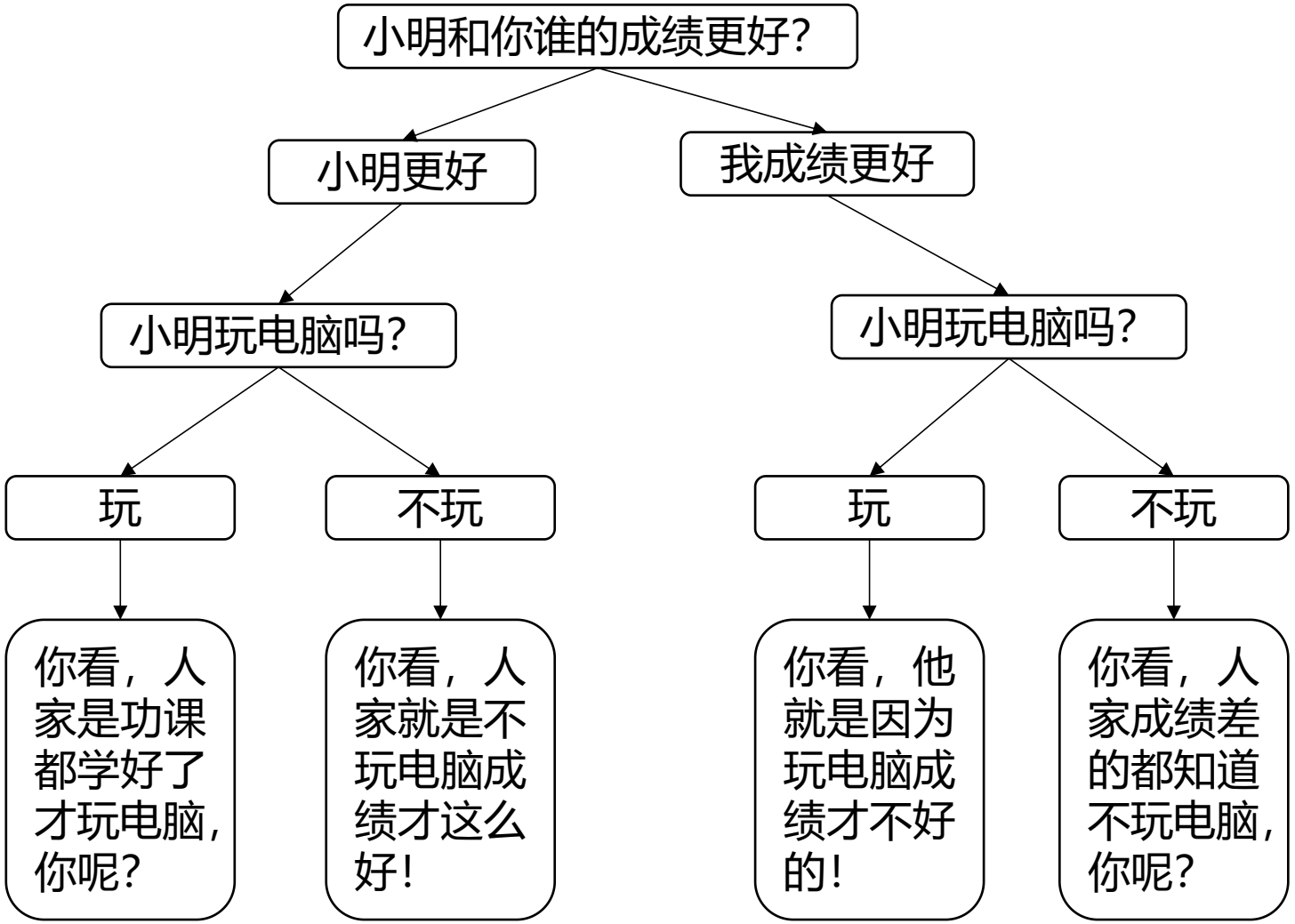
3. 运算符

C运算符分类：

1. 算术运算符 （+ - * / %）
2. 关系运算符 （> < == >= <= !=）
3. 逻辑运算符 （! && ||）
4. 条件运算符 （? ：）
5. 位运算符 （<< >> ~ | ^ &）
6. 赋值运算符 （= 及其扩展赋值运算符）
7. 逗号运算符 （, ）
8. 指针运算符 （*和&）
9. 求字节数运算符（sizeof）
10. 强制类型转换运算符（ (类型) ）
11. 分量运算符（. ->）
12. 下标运算符（ [ ] ）
13. 其他 （如函数调用运算符（））

- **不要太自信记忆力，该加括号加括号**
- **自增自减不要组合使用**
- **位运算与逻辑运算区分**
- **不要出现a>b>c这样的比较**

4. 条件分支


小明和你谁的成绩更好?
小明更好
我成绩更好
小明玩电脑吗?
小明玩电脑吗?
玩
不玩
玩
不玩
你看，人家是功课都学好了才玩电脑，你呢?
你看，人家就是不玩电脑成绩才这么好!
你看，他就是因为玩电脑成绩才不好的!
你看，人家成绩差的都知道不玩电脑，你呢?

4. 条件分支

```
#include <stdio.h>
int main()
{
        int x, t;
        scanf("%d", &x);

        if( … )
        {
                printf(“Execute if branch”);
        }
        else
        {
                printf(“Execute else branch”);
        }

        while ( … )
        {
                printf(“Enter iteration”);
        }
        return 0;
}
```

5. 循环控制：几种循环的比较

	while	do-while	for
设置循环初始值	语句前	语句前	语句前或表达式1
结束条件的判断	先判断后执行	先执行后判断	先判断后执行
循环控制的计	循环体内	循环体内	循环体内或表达式3

- 关于初始化
  - 用while和do-while循环时，循环变量初始化的操作应在while和do-while语句之前完成。
  - for语句可以在表达式1中实现循环变量的初始化。
- 关于循环中止
  - while循环、do-while循环和for循环，均可以用break语句跳出循环，用continue语句结束本次循环

5. 循环控制：循环的加速与中止

```
while(expr1)
{
        …
        continue;
        …
}
```

```
do
{
        …
        continue;
        …
} while(expr1);
```

```
for (expr1; expr2; expr3)
{
                …
                continue;
                …
}
```

```
while(expr1)
{
                …
                break;
                …
}
…
```

```
do
{
                …
                break;
                …
} while(expr1);
…
```

```
for (expr1; expr2; expr3)
{
                …
                break;
                …
}
…
```

5. 循环控制：熟记经典结构

```
#include <stdio.h>
int main()
{
        char x;
        scanf("%d", &x);

        while (x != -1)
        {
                .....
                scanf("%d", &x);
        }


        return 0;
}
```

- **经典结构控制模式要熟悉，避免“while（1）+break“等非主流的结构**

5. 循环控制：调试

```
#include <stdio.h>
int main()
{
        int x, t;
        scanf("%d", &x);

        iter = 0;
        while ( … )
        {
                iter++;
                if (iter > 100)
                        printf("Two many iterations!");
        }
        return 0;
}
```

```
#include<stdlib.h>
…
…
…

if (iter > 100)
        exit(1);
```

2. 程序调试：调试

```
#include <stdio.h>
int main()
{
        int x, iter;
        scanf("%d", &x);

        sum = 0;
        for (iter = 0; iter  < 100 && sum < 65535; iter++ )
        {
                printf(“iter = %d sum = %d\n”);

        }
        return 0;
}
```

6. 数组

- 数组名字的含义：首元素地址，常量
- 二维数组：含义、存储、访问

例如：int a[3][3]={ {1,2,3}, {4,5,6}, {7,8,9} };

地址	值	数组元素
3000H	1	a[0][0]
3004H	2	a[0][1]
3008H	3	a[0][2]
300CH	4	a[1][0]
3010H	5	a[1][1]
3014H	6	a[1][2]
3018H	7	a[2][0]
301CH	8	a[2][1]
3020H	9	a[2][2]

6. 数组

数组的初始化

例如： int a[10]={0,1,2,3,4,5,6,7,8,9};
int arr1[5]={1,4,5} , arr2[]={7,8,9};

例如： int a[3][4]={{1},{5},{9}};

```
1 0 0 0
5 0 0 0
9 0 0 0
```

例如： int a[3][4]={{1}, {0,6}, {0,0, 0,11}};

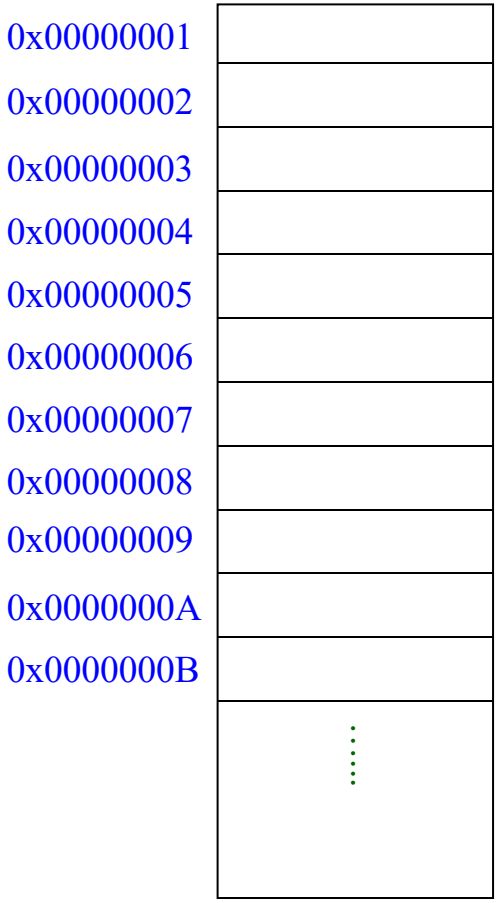
```
1 0 0 0
0 6 0 0
0 0 0 11
```

例如： int a[3][4]={{1},{5,6}};

```
1 0 0 0
5 6 0 0
0 0 0 0
```

7. 指针

- 计算机内存的每一个字节有一个编号，称之为“**地址编码**”。
- 地址编码是一种计算机特有的数据，C语言中用**指针类型**来表示这种地址数据。
- **变量(或代码)的地址称为该变量的“指针”。**
- **指针（类型的）变量**是一种特殊的变量，它是存放地址的。



7. 指针

- 指针等同于地址?
  - 变量的地址，可以称为变量的指针
  - 指针不完全等同于地址，指针虽是地址，但**有它关联的数据类型**

- 为什么要用指针?
  - 间接访问近乎机器指令，可极大**提高存取效率**
  - 结构化程序设计中，数据和代码都要求封装到函数内，指针成为**两个函数进行数据交换必不可少的工具**
  - 程序运行期间申请到的内存空间只有地址没有名称，因此指针成为**访问动态内存**的唯一工具
  - 可以描述**复杂的数据结构**
  - 可以简洁高效地进行**数组和字符串操作**

7. 指针

定义	含义
int i;	定义整型变量i
int *p;	p为指向整型数据的指针变量
int a[n];	定义整型数组a，它有n个元素
int *p[n];	定义指针数组p，它由n个指向整型数据的指针元素组成
int (*p)[n];	p为指向含n个元素的一维数组的指针变量
int f();	f为带回整型函数值的函数
int *p();	p为返回一个指针值的函数，该指针指向整型数据
int (*p)();	p为指向函数的指针，该函数返回一个整型值
int **p;	p是一个指针变量，它指向一个指向整型数据的指针变量

7. 指针

- 指针的状态与安全使用
  - 指针的值为0，又称空指针（null pointer）。

```
int *p=0;
*p=2; //空指针间接引用将导致程序产生严重的异常错误
```

  - 指向已知对象（经常与数组结合使用，注意不要“越界”）

```
int arr[10], *p=arr;
*(p+10) = 3; //谨防越界
```

  - 一个指针还没有初始化或赋值，称为“野指针”（wild pointer）

```
int *p; //p是野指针
*p=2; //几乎总会导致程序产生严重的异常错误
```

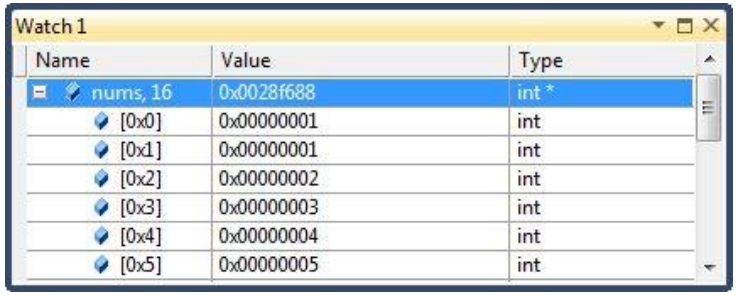
  - 指针运算后指向未知对象，那么该指针是无效的。

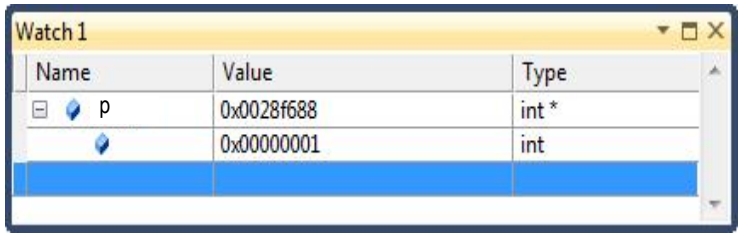
```
for (p=a;p<a+10;p++) scanf(“%d”,p); //运行后指针变无效
for (p=a;p<a+10;p++) printf(“%d ”,*p); //循环初始时，重新进行赋值
```

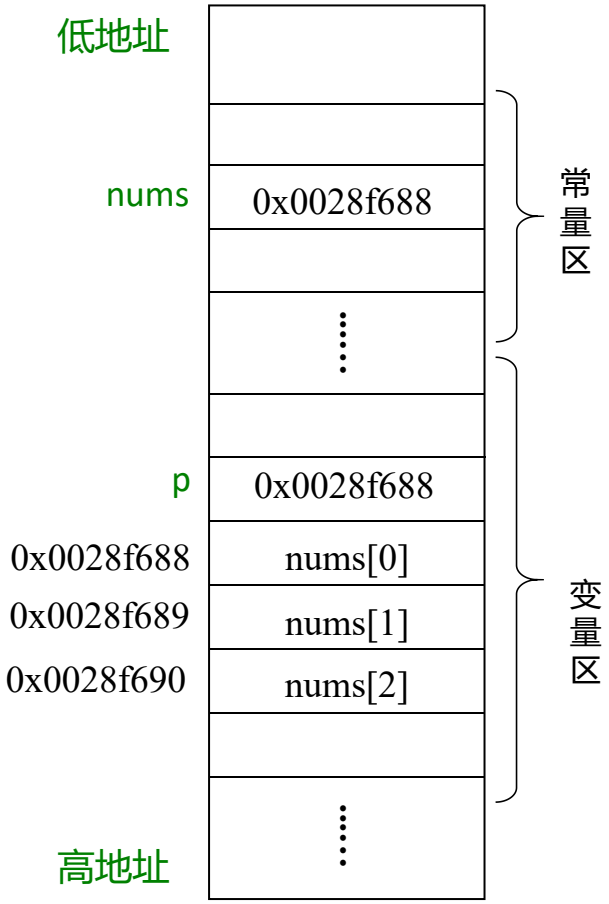
8. 指针与数组

- 指针与数组

```
int nums[16];
int *p=nums;
```







8. 指针与数组

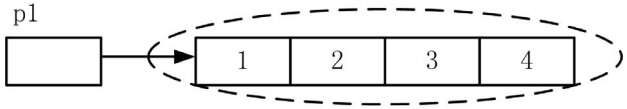
表 示 形 式	含义	等价地址
a	二维数组名，指向一维数组a[0]，即0行首地址	&a[0][0]
a+i， &a[i]	i行首地址	&a[i][0]
a[0], *(a+0), *a	0行0列元素地址	&a[0][0]
a[i],*(a+i)	i行0列元素a[i][0]的地址	&a[i][0]
a[i]+j, *(a+i)+j, &a[i][j]	i行j列元素a[i][j] 的地址	&a[i][j]
*(a[i]+j), *(*(a+i)+j), a[i][j]	i行j列元素a[i][j]的值	

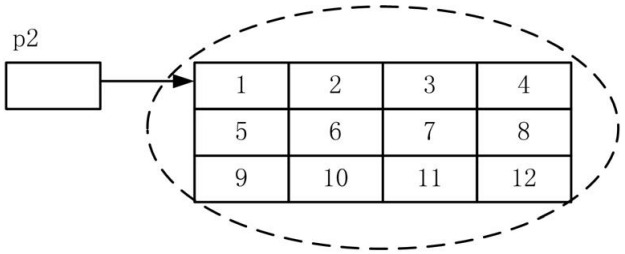
注意指向类型，事关赋值匹配、地址偏移计算

8.1 数组指针与指针数组

- **数组指针**：是**一个指针**变量，编译器为其分配**4字节**的存储空间，其指向类型是数组，*p就是该数组

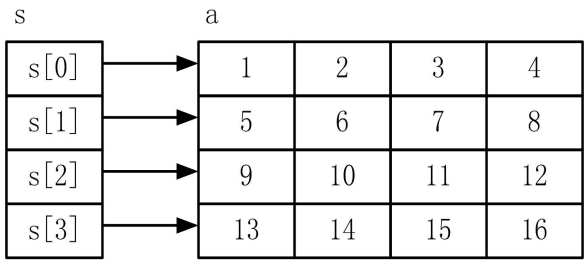
int (*p)[4];





- **指针数组**：是**一个数组**，编译器为其分配**4*N字节**的存储空间，其指向类型一般是数组

int *s[4];



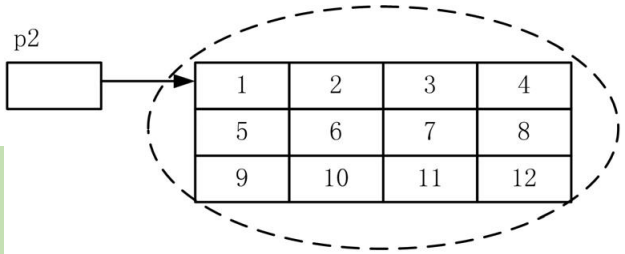
8.1 数组指针与指针数组：数组指针

- 初始化与赋值

```
int a[3][4], (*p)[4];
p=a; //*p是a[0]，即int[4]的数组
```

- 访问数组元素

多用*(*(p+i)+j), (*(p+i))[j], 偶尔用p[i][j];




```
#include <stdio.h>
void  main ( )
{
        int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};
        int  (*p)[4], i, j;
        p = a;
        scanf("i=%d,j=%d", &i, &j);
        printf("a[%d, %d]=%d\n", i,  j, *(*(p+i)+j));
}
```

8.1 数组指针与指针数组：指针数组

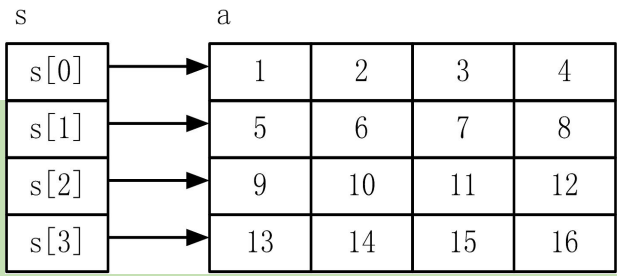
- 初始化：已知内存或者空指针

```
int *s[4]={NULL,NULL,NULL,NULL};//一维指针数组初始化

int a[4][4]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16}; //二维数组
int *s[4]={a[0],a[1],a[2],a[3]}; //一维指针数组初始化
```

- 访问数组元素

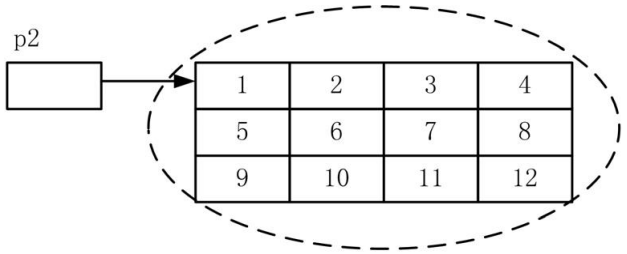
多用 s[i][j], *(s[i]+j)、偶尔用*(*(s+i)+j)



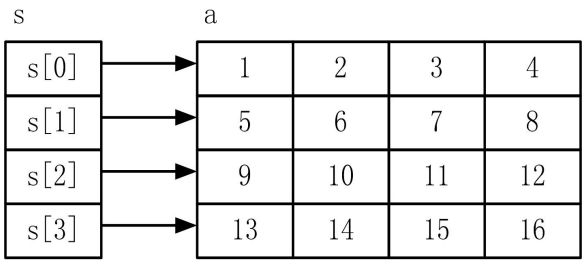

```
#include <stdio.h>
void main()
{
        int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};
        int i, j, *s[4]={a[0],a[1],a[2],a[3]}; //一维指针数组初始化
        for (i=0; i<3 ; i++) {
                for (j=0; j<4; j++)
                        printf("%2d ",s[i][j]); //s[i][j]等价于a[i][j], *(s[i]+j), *(*(s+i)+j)
                printf("\n");
        }
}
```

8.1 数组指针与指针数组

- 异
  - 概念不同
  - 存储不同
  - 赋值不同



- 同
  - 处理二维数组时级次相同
  - 访问元素方式相同



8.2 字符数组、字符指针、字符串：一维

- 概念区分
  - 字符数组：若干字符
  - 字符指针：一个指向字符的地址
  - 字符串：'\0'结尾的字符数组
- 联系
  - 定义一个字符数组，用字符串常量初始化　char str[]="C Language";
  - 定义一个字符指针，指向字符串常量　char *p = “C Language”；
  - 数组与指针结合使用

指针指向数组，使数组多了一种访问途径，但不能替代数组存储批量数据

```
#include <stdio.h>
int main()
{
        char str[100],*p=str;
        scanf("%s", str); //输入字符串
        while (*p) p++; //指针p指向到字符串结束符
        printf("strlen=%d\n",p-str); //输出字符串长度
        return 0;
}
```

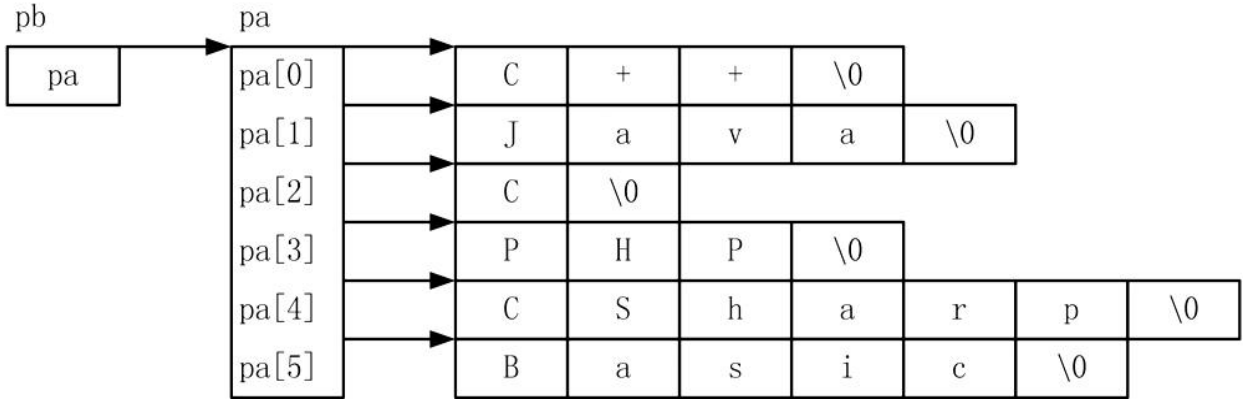
8.2 字符数组、字符指针、字符串：二维

- 二维字符数组与字符串指针
  - 用字符串数组存储若干个字符串时，要求每行包含的元素个数相等，因此需取最大字符串长度作为列数，浪费内存单元

sa

sa[0]	C	+	+	\0			
sa[1]	J	a	v	a	\0		
sa[2]	C	\0					
sa[3]	P	H	P	\0			
sa[4]	C	S	h	a	r	p	\0
sa[5]	B	a	s	i	c	\0	

  - 若使用字符指针数组，各个字符串按实际长度存储，指针数组元素只是各个字符串的首地址，不存在浪费内存问题



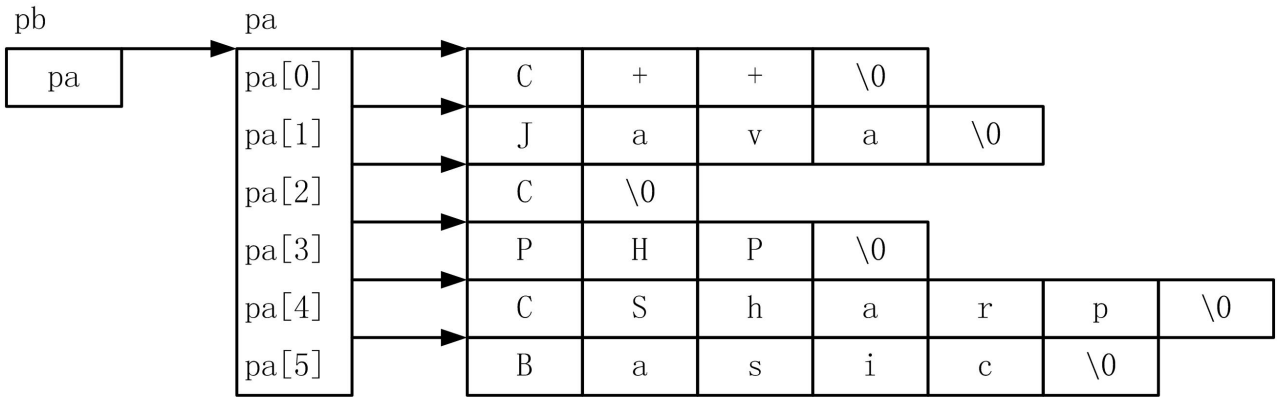
8.2 字符数组、字符指针、字符串：二维

- 字符串数组与字符串指针
  - 由于一个字符指针可以指向一个字符串，为了用指针表示字符串数组，有两种方案
    - 使用**指针数组**，例如

      char *pa[6]={"C++","Java","C","PHP","CSharp","Basic"};

      其中pa为一维数组，有6个元素，每个元素均是一个字符指针。
    - 定义**指向指针的指针**。

      char **pb=pa;



9. 函数与数组、函数与指针

- 如果有一个实参数组，想在函数中改变此数组中的元素的值，有四种方式

① 实参--数组
形参--数组

```
void  main()
{
      int a[10];
      f(a, 10);
}

void f(int x[], int n)
{
      …
}
```

② 实参--数组
形参--指针

```
void  main()
{
      int a[10];
      f(a, 10);
}

void f(int *x, int n)
{
      …
}
```

③ 实参—指针
形参--数组

```
void  main()
{
      int a[10], *p=a;
      f(p, 10);
}

void f(int x[], int n)
{
      …
}
```

④ 实参--指针
形参--指针

```
void  main()
{
      int a[10], *p=a;
      f(p, 10);
}

void f(int *x, int n)
{
      …
}
```

9. 函数与数组、函数与指针

- 二级指针的细节区分

```
#define ROW 3
#define COL 4

int main() {
        int a[ROW][COL]={0};
        //数组指针
        int (*p)[COL] = a;
        fun_1(p, ROW);

        //指针数组
        int *p[ROW] = {a[0], a[1], a[2]};
        fun_2(p, ROW, COL);

        //二级指针
        int **pp = a;
        fun_3(p, ROW, COL);
}
```

```
void fun_1(int (*pp)[4], int nRow)
//数组指针
{

}
```

```
void fun_2(int *pp[], int nRow, int nCol)
//指针数组，pp常量不可以加加减减
{

}
```

```
void fun_3(int **pp, int nRow, int nCol)
//二级指针
{

}
```

9. 函数与数组、函数与指针

- 跨函数内存申请：参数传递

```
void GetMemory(char *p, int num)
{
        p = (char *)malloc(sizeof(char)*num);
        //这里函数调用结束后内存丢失
}

void Test1(void)
{
    char *str = NULL;
    GetMemory(str, 50); //str仍然为NULL
    strcpy(str, "hello");  // 运行错误
}
```

```
void GetMemory2(char **p, int num)
{
    *p = (char *)malloc(sizeof(char) * num);
}

void Test2(void)
{
        char *str = NULL;
        GetMemory2(&str, 50); //参数是&str
        strcpy(str, "hello");
        printf("%s", str);
        free(str);}
```

深刻理解函数参数传递的本质

9. 函数与数组、函数与指针

- 跨函数内存申请：返回值

```
char *GetMemory3(int num)
{
    char *p = (char*)malloc(sizeof(char)*num);
    return p;
}

void Test3(void)
{
        char *str = NULL;
        str = GetMemory3(100);
        memset(str, 0, 100*sizeof(int));
        strcpy(str, "hello");
        printf("%s", str);
        free(str);
}
```

```
char *GetString(void)
{
    char p[] = "hello world";
    return p;  // 编译器将提出警告
}


void Test4(void)
{
        char *str = NULL;
        str = GetString(); //指向常量区
        printf("%s", str);
}
```

切忌传递来自栈的内存空间！！！

10. 链表

类型定义

```
typedef struct tagLNode //单链表结点类型
{
        ElemType data;
        struct tagLNode *next;
}LNode,*LinkList;
//LNode为单链表结构体类型，LinkList为单链表指针类型
```

等价于

```
struct tagLNode //struct tagLNode为单链表结点类型
{
        ElemType data;
        struct tagLNode *next;
}
typedef struct tagLNode Lnode; //LNode为单链表结构体类型
typedef struct tagLNode* LinkList;//LinkList为单链表指针类型
```

10. 链表

- 链表操作
  - 链表的创建
  - 链表的销毁
  - 链表访问：遍历、查找
  - 链表编辑：逆序、插入、删除

- **函数接口定义要深刻理解**
- **掌握各种操作的程序架构**
- **与动态内存管理的联合使用**
- **避免内存泄露**

10. 链表

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

int main()
{
        LinkList La,Lb,Lc;
        int
        j,a[4]={3,5,8,11},b[7]={2,6,8,9,11,15,20};
        InitList(&La); //创建空表La
        for(j=1;j<=4;j++) //表La中插入元素
                ListInsert(La,j,a[j-1]);

        InitList(&Lb); //创建空表Lb
        for(j=1;j<=7;j++) //表La中插入元素
                ListInsert(Lb,j,b[j-1]);

        MergeList(La,Lb,&Lc);
        return 0;
}
```

```
void InitList(LinkList *L) //构造一个空的单链表L
{
}
```

```
int ListInsert(LinkList L,int i,ElemType e)
{ //在第i个位置之前插入元素e
}
```

```
void MergeList(LinkList La,LinkList Lb,LinkList *Lc)
 {
}
```

10. 链表

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

int main()
{
        LinkList La,Lb,Lc;
        int
        j,a[4]={3,5,8,11},b[7]={2,6,8,9,11,15,20};
        La = InitList(); //创建空表La
        for(j=1;j<=4;j++) //表La中插入元素
                ListInsert(La,j,a[j-1]);

        Lb = InitList(); //创建空表Lb
        for(j=1;j<=7;j++) //表La中插入元素
                ListInsert(Lb,j,b[j-1]);

        Lc = MergeList(La,Lb);
        return 0;
}
```

```
DLinkList InitList() //构造一个空的单链表L
{
}
```

```
int ListInsert(LinkList L,int i,ElemType e)
{ //在第i个位置之前插入元素e
}
```

```
LinkList MergeList(LinkList La,LinkList Lb)
 {
}
```

本节课主要内容

- 程序设计流程
- C语言知识点回顾
- **C++语言知识点回顾**
- 几道面试题
- 备考事项

1. 面向对象的概念：类、对象、封装

```
class  Clock
{
public:
        void setTime(int newH, int newM, int newS);
        void showTime();
private:
        int m_hour, m_minute, m_second;
};
```

```
bool Clock::SetTime(unsigned int nHour, unsigned int  nMin, unsigned int nSec)
{
        if (nHour < 24 && nMin < 60 && nSec < 60 )
                m_hour = nHour, m_minute = nMin, m_second = nSec;
};
```

```
Clock  myClock;
bool res_1 = myClock.setTime(9, 35, 0); //返回true
bool res_2 = myClock.setTime(9, 85, 0); //返回false
myClock.hour = 25; //私有成员，不可以通过对象访问
```

2. 类的定义

```
//默认构造函数
my_array::my_array() {
    this->len = 10;
    this->space = new int[len];
}
```

```
//带参构造函数
my_array::my_array(int len) {
    if (len <= 0) {
        this->len = 0;
        return;
    }
    else {
        this->len = len;
        this->space = new int[this->len];
    }
}
```

```
//析构函数
my_array::~my_array() {
    if (this->space != NULL) {
        delete[]this->space;
        this->space = NULL;
        len = 0;
    }
}
```

```
//拷贝构造函数
my_array::my_array(const my_array &another){
    if (another.len >= 0) {
        this->len = another.len;
        this->space = new int[this->len];
        for (int i = 0; i < another.len; i++){
            this->space[i] = another.space[i];
        }
    }
}
```

```
void my_array::operator=(const my_array &another) {
    if (another.len >= 0) {
        this->len = another.len;
        this->space = new int[this->len];
        for (int i = 0; i < another.len; i++){
            this->space[i] = another.space[i];
        }
    }
}
```

3. 类的继承

- 三种继承方式的应用特点
  - 公有继承：从外部通过派生类对象可以访问基类的公有成员
  - 私有继承：从外部通过派生类对象不能访问基类的任何成员
  - 保护继承：从外部通过派生类对象不能访问基类的任何成员

继承方式/基类成员	public	protected	private
public (公有继承)	public	protected	不可见
protected (保护继承)	protected	protected	不可见
private (私有继承)	private	private	不可见
规律	**X继承方式，基类中Y权限成员在派生类中变为min(X, Y)权限**		**基类的private在派生类中一定不可见**

3. 类的继承

- **隐藏：**指派生类中的与基类变量**同名的变量**或与基类函数**同名的函数**(注意，**参数不一定相同**)会遮盖基类中的成员，使得在派生类中无法访问

```
#include <string>
using namespace std;
class Base {
public:
   Base(int Param1 = 0, int Paran2 = 1);
   ~Base();
   Base(const Base& aBase);
   Base& operator=(const Base& aBase);
   void PublicFunc();
   static const string ClassName;
protected:
   void ProtectedFunc();
   int ProtectedData;
private:
   void PrivateFunc();
   int PrivateData;
};
```

```
#include "Base.hpp"


class Derived : private Base {
public:
   static const string ClassName;
   //隐藏了基类同名函数
   void PublicFunc(){
      //使用作用域操作符，依然可以访问基类同名函数
      Base::ProtectedFunc();
   }
   //↓调整基类成员在派生类中的访问权限
   using Base::ProtectedFunc;
};
```

4. 概念甄别：重载、隐藏与覆盖甄别

- 成员函数被**重载**
  - 相同的范围(在同一个类中)
  - 函数名相同
  - 函数参数个数或类型不同
  - virtual关键字可有可无

```
class Person
{
        public void fun()
        {…}

        public void fun(string s)
        {…}

        public void fun(string s, int i)
        {…}
  }
```

4. 概念甄别：重载、隐藏与覆盖甄别

- 成员函数被**隐藏**
  - 不同的范围(分别位于派生类和基类)
  - 函数名字相同，参数可以相同，也可以不同
  - virtual关键字可有可无
  - 基类中的同名函数在派生类中不可见

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

class Base{
public:
        void fun(double ,int ){}
};
class Derive : public Base{
public:
        void fun(int ){};
}
```

```
int main()
{
        Derive pd;
        pd.fun(1);//Derive::fun(int )
        pd.fun(0.01, 1); //编译错误
        return 0;
}
```

4. 概念甄别：重载、隐藏与覆盖甄别

- **覆盖**是指派生类的虚函数覆盖基类的虚函数
  - 不同的范围(分别位于派生类和基类)
  - 函数名字相同
  - 函数参数相同
  - 基类的被覆盖函数必须有virtual关键字

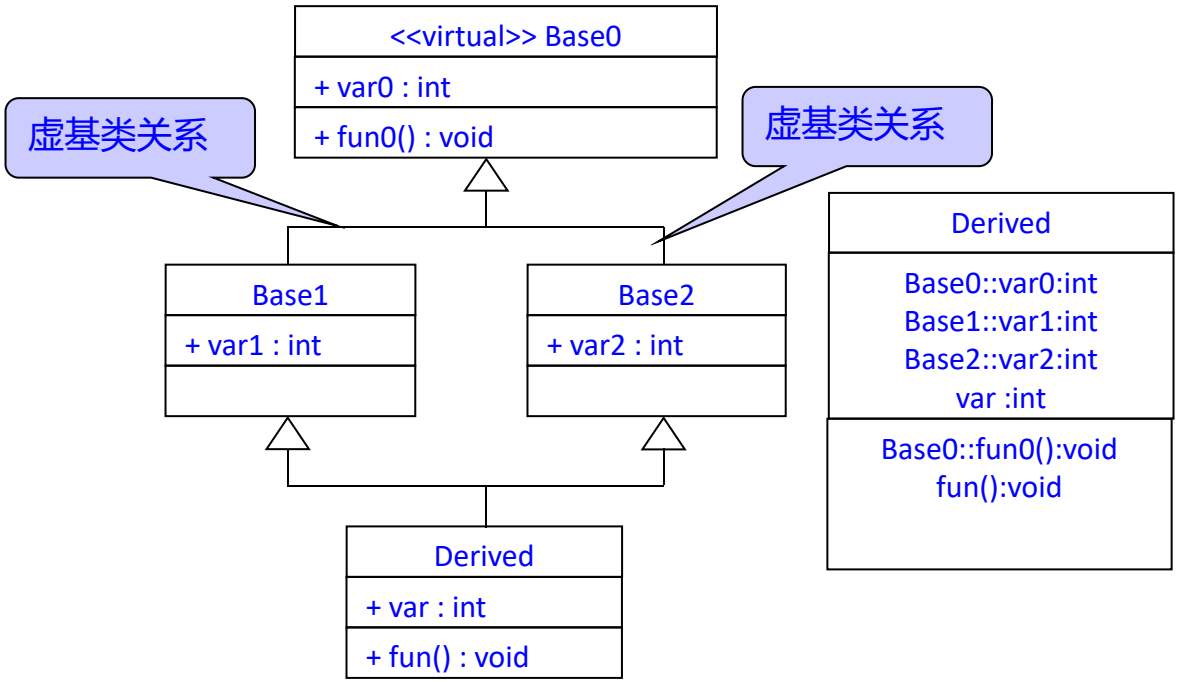
```
#include<iostream>
using namespace std;

class Base{
public:
    virtual void fun(int i){}
};
class Derived : public Base{
public:
    virtual void fun(int i){}
}
```

```
int main()
{
    Base * pb = new Derived();
    pb->fun(3); //Derived::fun(int)
    return 0;
}
```

5. 概念甄别：虚基类、虚函数

- **虚基类**是一种继承方式（如 class B1:virtual public B），主要用来解决**多继承时**可能发生的对同一基类**继承多次而产生的二义性问题**



注意：虚基类实际上是一种类的继承关系，而不是类本身的某种特性。

5. 概念甄别：虚基类、虚函数

- **虚函数**是实现多态的方式
  - 在基类中声明**同名的虚拟函数成员**，并在派生类进行覆盖
  - **基类指针（或引用）**指向**其派生类实例，**对基类虚函数**动态绑定**

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

class Base1                    //基类Base1定义
{
public:
        virtual void display() const; //虚函数
};

class Base2:public Base1
{
public:
        void display() const; //覆盖基类虚函数
};

class Derived: public Base2
{
public:
        void display() const; //覆盖基类虚函数
};
```

```
void fun(Base1 *ptr) //参数为指向基类对象的指针
{
        ptr->display(); //"对象指针->成员名"
}


int main()  //主函数
{
        Base1 base1;            //定义Base1类对象
        Base2 base2;            //定义Base2类对象
        Derived derived;        //定义Derived类对象
        fun(&base1);//用Base1对象的指针调用fun()
        fun(&base2);//用Base2对象的指针调用fun()
        fun(&derived);//用Derived对象指针调用fun()
        return 0;
}
```

本节课主要内容

- 程序设计流程
- C语言知识点回顾
- C++语言知识点回顾
- **几道面试题**
- 备考事项

几道面试题

- #define MAX_NUM 1000+1
  int Temp = Max_NUM*10; 请问：Temp的值是多少?

- 编程实践中，浮点型变量与“零值”一般怎么进行比较?
  怎么判断指针是不是空指针?

- int nArr[10] = {1, 6, 3};
  对nArr进行升序排序，结果是什么?

几道面试题

- char s[]=“Hello World”和
  char s[11]=“Hello World” 的区别是什么
- 想在子函数内部改变主函数里面定义的变量值有哪些方法?
- int (*p)[6] 与 int *p[6]的区别?
- 定义char s[]=“abc”； 则执行语句x = sizeof(s)后，x为多少?
  定义char s[5]=“abc”； 则执行语句x = strlen(s)后x的值是多少?
  定义 char *s= “abc”； 则执行语句x = sizeof(s)后，x为多少?

几道面试题

- int arr[100] = {1, 3, 42 ,24, 245, -3};

  请写一个子函数，对arr进行排序，接口应该怎么设计?

  int arr[3][20] = {{1, 3, 42} ,{24, 245}, {-3}};

  请写一个子函数，对arr各行按照行和排序，接口应该怎么设计?

- char arr[100]， *p;

  scanf(“%s”, &arr);

  scanf(“%s”, p);

本节课主要内容

- 程序设计流程
- C语言知识点回顾
- C++语言知识点回顾
- 几道面试题
- **备考事项**

考试注意事项

- 关键知识点要知晓
  - 数组、链表、字符串、排序等
  - 指针、数组做函数参数，基本原则
- 交卷
  - 格式规范（.c文件）、一道题保存一次
  - 不要轻易弃考、不要空程序、不要直接放弃
- 心态要好
  - 严肃、竞技感、谨慎
  - 不紧张（复制粘贴忘修改、多分号；=与==）

考试注意事项

- 利用好很有限的答题时间
  - 模块化编程，提高速度和代码质量

  - **先分模块或步骤**
  - **逐模块编写，不要尝试同时兼顾10行以上的代码**
  - **需要的变量慢慢添加，不可能一次性想到**

  - 及时调整，切勿恋战

  - **如果思路出现了越想越麻烦的状态，该调整方案了！**

考试注意事项

- 根据难度与分值统筹安排时间
  - 不要臆想几道题的难度次序
  - 事先通读试卷，分清必得分、争取分、放弃分
  - 包括子函数与主函数
  - Input、output等简单功能

- 利用好闲散时间
  - 提前进入考场
  - 利用好考前时间
  - 避免频繁切换窗口
  - 经典程序准备好
  - 三思而行、思路清晰

考试注意事项

- 缺失知识点

  - scanf、字符串查找比对...

- 注重理解

  - 理解为什么这样编
  - 理解才是真正掌握，是进行变通的基础

- 跟榜样学习

  - 有一些固定模式，比如碰到特定字符串退出
  - 比如匹配字符串、求和、交换等等
  - 好的程序好写、好调、好读
  - 不要OJ通过万事大吉

考试注意事项

- 复习也很重要
  - 老问题，新做法
  - 老问题，新速度
  - 老问题，新成绩

- 考场纪律
  - 瓜田李下，不要给监考与巡考想象空间

考好。