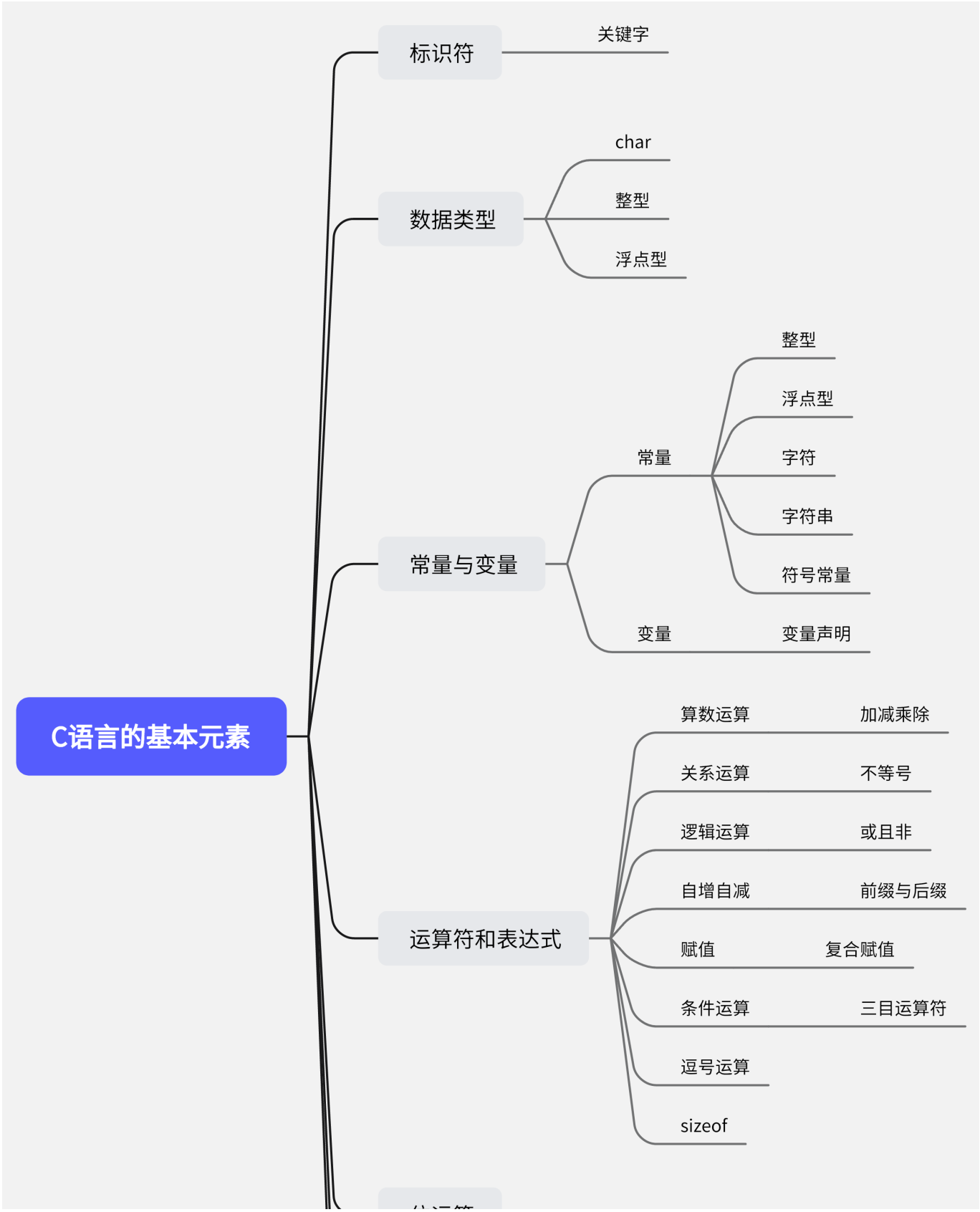
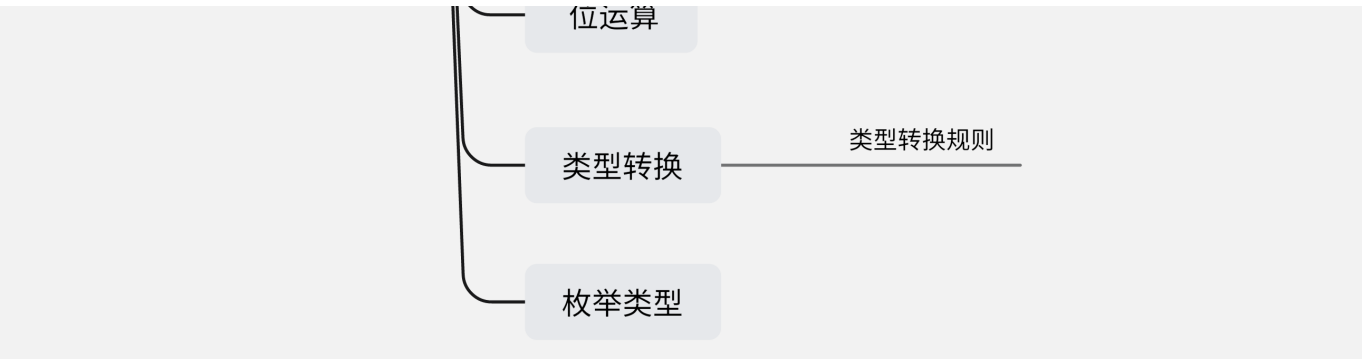
第二章 C语言的基本元素

思维导图





概要

本章是C语言最为基础的一章，很多知识在后续章节还会用到，如结构与联合的内容仍为基本元素，指针中也会用到类型转换等等。此外本章的内容也比其他章节更为琐碎，不易于记忆，但~~越是不好记的东西老师越爱考~~。

本章的主要内容可分为两类：数据和运算。数据包括基本元素类型、常量与变量、枚举类型（最早接触到的构造类型）；运算分为初等数学运算（加减乘除、小于大于等于）、计算机逻辑运算（或且非、赋值）、位运算（自己归纳的，不足之处请谅解）。

本章的复习我设计了三个模块：书中细碎知识、重点知识和例题（错题）。其中重点知识是结合例题来讲的，以求同时加深对两方面内容的理解。

---

1 边角知识

注意：这一部分的知识多数是老师课上所说的"这两年没考过"的知识，或课本上没有，课件上补充的知识。期末考到的概率不大，作为考试题的主题的概率则更低，酌情复习即可。

1.1 字符集

C语言字符集是7位ASCII码的子集，包括大小写英文字母、数字、特殊字符和空白字符（这个感觉重要一些）。

- 特殊字符 29 个，包括!、''、#、+、-、=等，主要为标点符号、数学记号。
- 空白字符包括：空格、换行符(\n)、水平制表符、垂直制表符、换页符，它们被打印出来时，页面上显示的是空白。

其他字符（如汉字）只能出现在注释语句、字符常量或字符串常量等。

1.2 C99新增数据类型

- long long 竞赛题、算法题用得多，64位，8byte，懂得都懂。输入输出使用%lld。
- bool类型，用得不少，只有0和非0两个值，0则为false，非0（哪怕为负数）则为true。
- 复数类型，见得不多，一般用_Complex表示实部，_Imaginary表示虚部，均为浮点型。使用复数运算和复数函数需要包含头文件<complex.h>

---

2 重点知识与例题分析

2.1 关键字与标识符

这一部分的考题通常以判断某个字符串能否作为标识符为主。**注意**：关键字实际上是特殊的标识符，但作为考题出现时，关键字不可作为标识符

关键字共32个如下：

```
/*************变量声明类***************/
auto
const
extern
register
static
typedef

char
double
enum
float
int
long
short
signed
struct
unsigned
union
void
/*************常用函数类*************/
break
case
continue
default
do
else
for
goto
if
return
sizeof
switch
while
/**********其他不知道干嘛的类**********/
volatile    //说明变量在程序执行中可被隐含地改变
```

例1

```
（学解复习题）下列选项均不是C语言关键字的是()
A.define,IF,type
B.getc,char,printf
C.include,scanf,case
D.if,struct,type
```

解析：刚刚手打了一遍关键字，这下~~一眼看出答案子~~。有些程序设计时常用的字段，如 include包含头文件，define 宏定义等并不是关键字，此外注意**大写的都不是关键字。**

本题答案选 A 。

例2

```
(学解考试宝典)以下不能定义为用户标识符的是()
A._3com_
B.int
C.Void
D.STatic
```

解析：题很简单，这里重点说一下用户 标识符的定义规则：

1. 下划线或字母开头
2. 只能包含数字、下划线、字母
3. 不能是关键字

注意：关键字若有部分字母或全部是大写，如本题中 Void ，则可以作为标识符。

本题选B。

2.2 数据类型

一般以判断常量的类型为主，需要注意整型常量的前后缀和浮点型常量的后缀。

- 前缀： （整型常量特有）表示进制类型。0表示八进制，0x或0X表示十六进制，若小写则表示十六进制数中a~f也为小写，大写类似。
- 后缀： 表示类型。对整数，有 L / l（长整型）、 U / u （无符号）或二者结合；对浮点型：f / F （单精度）、l / L （长双精度），没有后缀时默认双精度。
- 浮点型常量可省略整数部分，默认整数部分为 0；也可省略小数部分，默认小数部分为0.

有一种特殊表示形式：科学计数法，如2E10即2×10^{10}.需要注意 e / E 前面的部分不能同时缺少整数和小数，后面的部分（阶码）只能为整数。

例3

```
(2015-2016期末T3)常量'1'、1、1.0的类型分别是()
A.char,int,float
B.int,char,float
C.char,int,double
D.int,char,double
```

解析：~~阴间！！！~~这竟是期末考试题，无语住了。对于浮点型常量，没有后缀时，默认为双精度（见上方高亮或课本P33）所以 1.0 是 double 型。

本题答案选C。

例4

```
(头歌作业题改编)判断数据类型
1..12
2.'"'（一对单引号里一个双引号）
3.'''（三个单引号）
4.''（一对单引号）
5.""（一对双引号）
```

解析：~~依然很阴间~~。首先第一个又有题号又有小数点确实很影响辨认；后四个是同类的，需要记住：一对单（双）引号里不能有单独的单（双）引号，需要转义，一对单引号不能单独存在，但一对双引号单独存在可表示空字符串。至于为什么，~~还是去问电脑吧~~。

本题答案：浮点型常量、字符常量、非法常量、非法常量、字符串常量。

2.3 常量与变量

2.2已经判断了数据的类型，那这里常常判断数据是否合法(主要指常量)。这里值得注意的则是**常量的前缀**~~(不代表后缀不重要)~~。前缀一般包括0,0x,0b，表示了数据的进制表示法。0为八进制，0x为十六进制，0b为二进制，不能出现比自己进制数还大的数。

> 十六进制有两种前缀：0x和0X，前者表示10~15的数字时用小写字母，后者则用大写字母。
> 如果这个错了，亲测编译器是能通过的，但课本上既然这样写~~我觉得也是非法的吧~~

科学计数法也是可能出现的内容，前面已经介绍过，学会两者之间的转换是必要的。科学计数法的表示形式为

$$[整数部分][.][小数部分]e/E[\pm]n[后缀]$$

这里的**省略**需要格外注意。整数和小数部分可以没有，但不能二者均无。

对于$double$常量，可以没有整数或小数其中之一，但不能缺少小数点。因为$double$类型没有后缀，所以如果只有整数部分不加小数点，如`12`，会被认为是整型。

字符常量中转义字符很重要。即使是字符串常量，如`"abcd\0"`，这里的`\0`也是一个字符而非是`\`和`0`。

常见转义字符有三种：

1. 约定俗成的字母表示，如`\n \t`
2. 三位八进制数字表示，如`\101`
3. 二位十六进制

例5

```
(学解考试宝典)下面4个选项中，均是不合法的转义字符的选项是()
A.'\011','\f','\'
B.'\abc','\101','\x1f'
C.'\1011','\','\aa'
D.'\','\\','\xf'
```

解析：'\f'是翻页符，单独一个''是不合法的。十六进制转义符必须有x前缀，八进制转义符最多不超过三位。故C均不合法。

本题答案选C。

例6

```
(2014-2015期末)合法的转义字符有()
A.'\45'    B.'\0'    C.'\18'    D.'\0xa'
```

解析：十六进制转义字符前缀是'x'。

本题答案选AB

例7

```
(2013-2014期末)关于023584UL最准确的解释是()
A.无符号长整型常量      B.长整型常量
C.有符号整型常量        D.非法常量
```

解析：不要顾此失彼。注意到"UL"的后缀，我们很容易想到这可能是无符号长整型，但'0'前缀限定了该数是八进制，而常量中出现了8，因此这实际上是一个非法变量。

本题答案选D。

2.4 运算符与表达式

有了各种类型的常量和变量，下一步自然要开始运算。

需要注意的是运算符的优先级和结合性，这个一般会在附表给出，不过最好熟悉。相同优先级的表达式，从结合方向向对方方向进行运算。如右结合即从右向左运算。

运算符	结合性	运算符	结合性
() [] -> .	左结合	^	左结合
! ~ ++ -- + - * & (类型) sizeof	右结合		
* / %	左结合	&&	左结合
+ -	左结合	|	
<< >>	左结合	?:	右结合
< <= > >=	左结合	= += -= *= /= %= &= ^= |= <<= >>=	右结合
== !=	左结合	,	左结合
&	左结合		

注意：从上到下，从左到右是优先级顺序，注意第二栏里 '*' 是指针运算符，'+'、'-' 是取正、取负运算符。

这里强调一下 序列点 的应用。对于自增和自减(专指左值，即i++而非++i)，序列点前仍用其原值，序列点后则使用新值。序列点位于：

1. &&、||、?: 和 , 运算符的第一个操作数结尾处。
2. 完整表达式的结束，包括：表达式语句的分号处，do、while、if、switch、for 语句的表达式的右括号处，for语句的两个分号处，return语句的表达式末尾分号处。

对于逗号运算符的用法，从左到右对每一个表达式运算，以最后一个逗号后的表达式作为整个表达式的值。

例8

```
(2016-2017期末)设int x=3,y=2;,表达式(x++,y++)计算后，x和y的值分别是()
A.3、2    B.4、3    C.4、2    D.3、3
```

解析：表达式结束是序列点，因此结束后自增自减的操作已经生效。

本题答案选B。

例9

```
(学解考试宝典)设变量x,y,z均为double类型且已正确赋值，下面不能正确表示数字式子
x/y/z的C语言表达式是()
A.x/y*z                B.x*(1/(y*z))
C.x/y*1/z              D.x/y/z
```

解析：实际上也考察了类型转换。由于x,y,z已经确定为 `double` 型变量，那么 `1/(y*z)` 这种式子就不会发生强制转化为整型(分母为浮点型)。按优先级从左至右算即可。

本题答案选A。

2.5 位运算

这是课本上加了一个 `*` 的章节，按照课程组组长的意思，这是非计算机学生不作要求，而计算机学生应当重点掌握的章节。

位运算主要有以下几种运算方式：

1. `按位或(|)`：存在1，则值为1，否则为0；
2. `按位与(&)`：全为1，值才为1，否则为0；
3. `按位取反(~)`：是1则0，是0则1；

4. `按位异或(^)`：不同为1，相同为0(可视作无进位二进制加法)。
5. `左移位(<<)`：高位丢弃，低位补0；
6. `右移位(>>)`：低位丢弃，高位补0或1(与机器有关)。

同时本节还有一个重要内容：**整数在机内的表示。**

- 对于无符号整数，所有位数均用来表示数据的值，而有符号整数以最高位作为符号位；
- 正数在机内是以原码表示的，负数在机内是以补码表示的
- `原码`就是高位符号位，低位二进制表示其绝对值；符号位除外其他位取反则为`反码`；反码加一即为`补码`。

例10

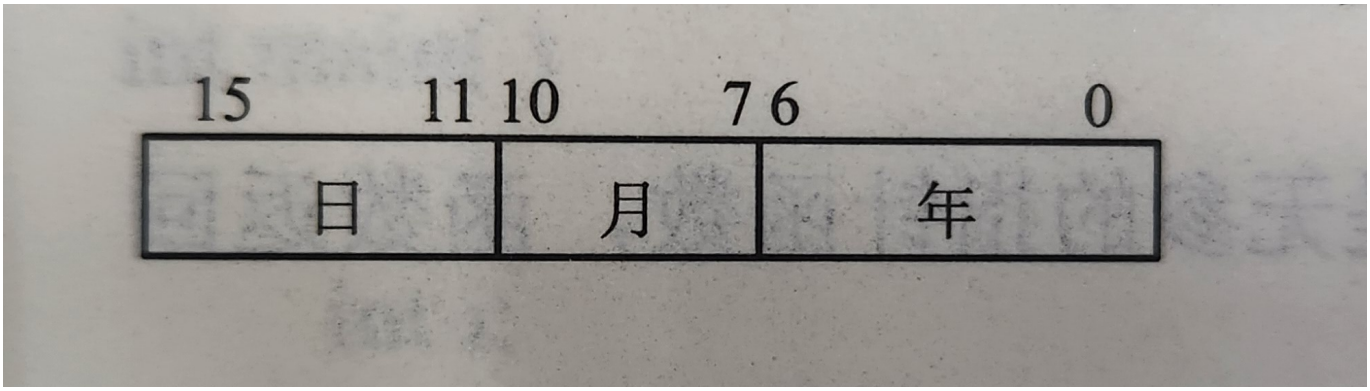
```
(2019-2020期末)改错题
将p的高字节作为低字节，q的低字节作为高字节组成一个新的短整型数k
#include<stdio.h>
int main(){
    short p=0x8034,q=0x5678;
    k=p>>8|q<<8;
    printf("%hx\n",k);
    return 0;
}
```

解析：对于数的右移位，为防止机器原因造成高位补1带来的影响，可以让p右移八位后的结果与`0xFF`取与，再与q取或。

正确答案：`k=p>>8&0xFF|q<<8`。

例11

```
(2019-2020期末)写一个表达式，将表示21世纪日期的日(day)、月(month)和年(year)
三个整数(如19年12月24日)压缩存储在一个短整型数x中，存储格式如下图所示：
```



解析：压缩的方式不只是字段结构，位运算也可以起到压缩的作用。我们只需要让日、月左移位到高位即可。

正确答案：`x=day<<11|month<<7|year`。

其实无论字段结构还是位运算，都是将其他数存储在另一个数空余的二进制位上。

例12

```
(2015-2016期末)表达式1|2|4>>2的值是()
A.0    B.1    C.3    D.4
```

解析：看上去考位运算，实际上考优先级。按位或的优先级高于右移位，因此从左至右顺序计算即可。

本题答案选B。

2.6 类型转换

按照课程组组长的意思，类型转换只需注意"高位溢出"即可，但这里想把类型转换的一般过程梳理一下。

1. 整数提升：对于`char`和`short`，首先会引起整数提升，自动转换成`int`或`unsigned`
2. 算术转换：对于双目运算符的求值，首先两个操作数独立整数提升，若提升后两个操作数类型不同，就会发生算术转换，总原则为：值域较窄的类型向值域较宽的类型转换，这里值域指能表示值得最大范围。

   ```
   char/short -> int -> unsigned -> long -> unsigned long -> long long
   -> unsigned long long -> float -> double -> long double
   ```

3. 赋值转换：右操作数的值被转换为左操作数的类型。

4. 强制类型转换：实际上是一个运算符，在前面优先级表上可查。

注意：浮点数转换为整数时，截去小数部位，不存在四舍五入等运算规律。

```
没有错题
```

2.7 枚举类型

主要注意枚举类型的定义和使用。

```
enum[枚举名]{标识符[=常量表达式],标识符[=常量表达式],...}
```

如`enum weekday{SUN,MON,TUE,WED,THU,FRI,SAT};`

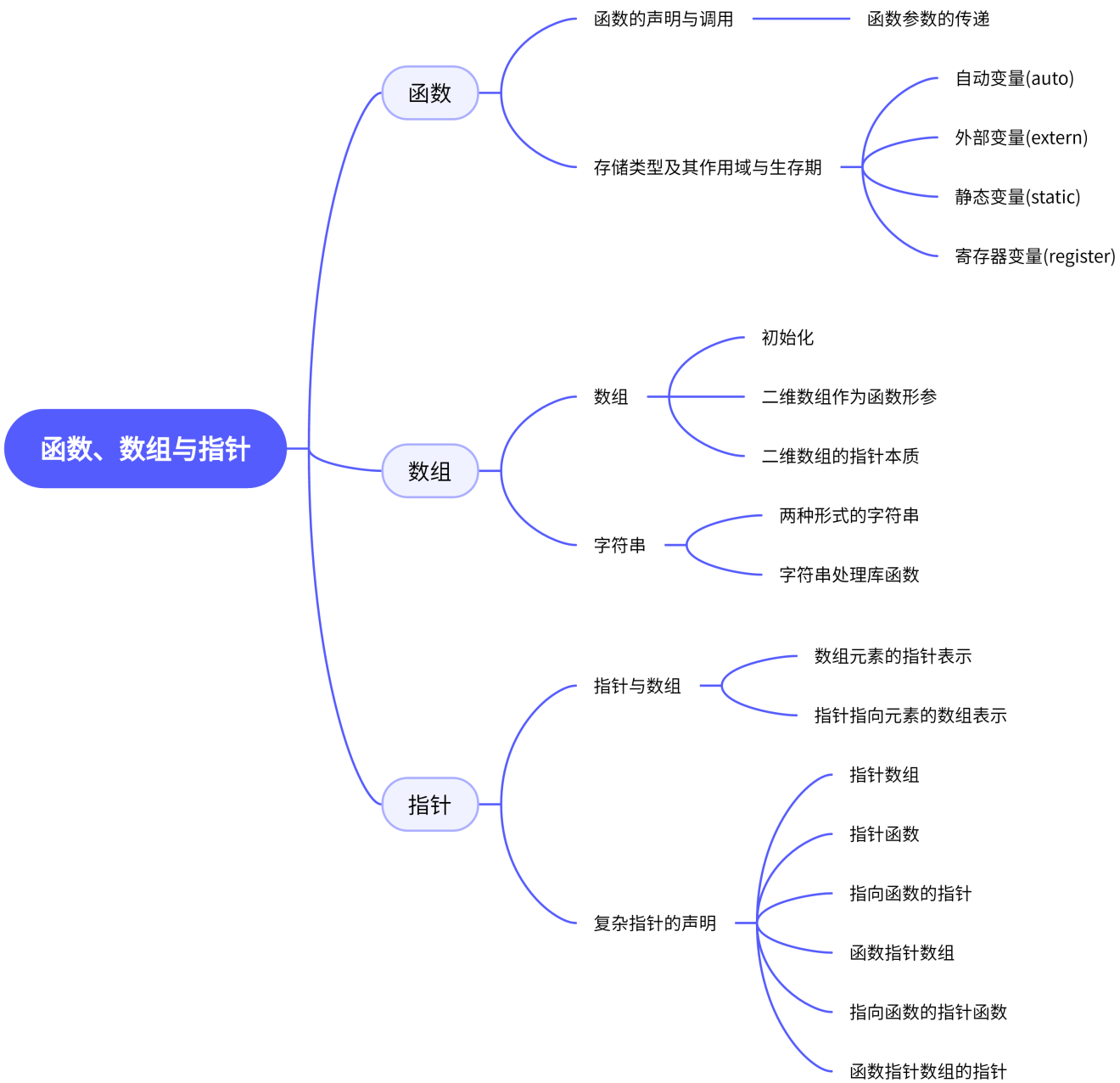
枚举类型内部的标识符实际上是枚举常量，如果没有特殊赋值，则从0开始按1递增，若其中一个常量赋值为k，则下面的常量从k开始按1递增。

```
没有错题
```

END

函数、数组与指针

思维导图



概述

函数是贯穿C语言后半部分的一章，很多功能的实现依托于函数。正因此函数本身处于一个**很重要但又无需重点复习**的章节，正如流程控制一样，其存在只是为了实现一种功能。而在函数章节需要重点复习的，实际上是四种变量的存储类型和作用域与生存期(与课程组老师不谋而合)。这里也会把函数与带参宏定义进行比较。

数组是连接C语言上下部分的一章，数组本身作为储存变量的容器，其性质与它存储的变量类型的性质一致。需要注意的是二维数组作为**函数形参**的表示方式以及**字符串**的相关处理。同时也可以思考一下数组的指针本质。

指针则是C语言集大成的一章，也是课程的最大重难点。课程组老师说过

> 指针搞不好，考过很难

大概就是这个意思。这一章需要复习的就太多了。各种指针的声明及实际作用都需要熟练掌握，但最重要的是明白每个指针的类型和它所指向的对象。

1 边角知识

注意：这一部分的知识多数是老师课上所说的"这两年没考过"的知识，或课本上没有，课件上补充的知识，但不包括老师说的"完全不考"的知识。期末考到的概率不大，作为考试题的主题的概率则更低，酌情复习即可。

如果复习尚未完成，建议先看第二部分，再看这里的边角知识。

1.1 条件编译与断言

是编写程序进行调试时常用到的语句。主要有以下几组：

1. #if 指令

   #if 常量表达式 程序段

   #elif 常量表达式 程序段

   #else 程序段

   #endif

   当常量表达式不为0时执行对应的程序段，用法类似if函数，但容易添加和删除，适合调试时使用。

2. #ifdef 指令

   #ifdef/#ifndef 标识符 程序段

   #elif 常量表达式 程序段

   #else 程序段

   #endif

ifdef(或ifndef)用于检测标识符是否被(或未被)#define宏定义。

也可用`#if defined(标识符)`

3. 断言assert

`assert(condition)`若condition为真，则什么也不发生，否则程序终止运行。

1.2 指针变量的大小

任意类型的指针变量均占用4个字节(32位系统)或8个字节(64位系统)

1.3 typedef

能看懂即可。typedef定义的一般形式为：

```
typedef 类型区分符 说明符;
```

例如：`typedef int bool`，即`bool`是基于`int`的一个分支，定义了新的类型名

`typedef char * string`，给`char *`一个别名`string`。

---

2 重点知识与例题分析

2.1 函数的声明与调用

函数声明的形式为

```
类型名 函数名(参数类型 [标识符],参数类型 [标识符],...);
```

函数调用时参数的传递方式为值传递，区别于`引用传递`。前者只传值，后者传递的是地址。

例1

```
(同学问的题)实现交换x,y的值的功能，下面函数是否正确
1.调用：swap(&x,&y);
  声明：void swap(int *x,int *y);
2.调用：swap(x,y);
  声明：void swap(int &x,int &y);
函数体均为：
{
    int t;t=x;x=y;y=t;
}
```

解析：函数的参数形式与其**调用形式**有关。但无论什么形式，只要注意到函数的值传递，能做到改变的是数本身(原地址上的数)而非传递的数即可。

`int *x=&a`好理解，做到了传地址，`int &a=b`实际上是在C++里学的引用声明，a改变，则b也相应改变。

故两个函数都是能实现传递功能的。

2.2 存储类型及其作用域与生存期

四种变量存储类型分别为：自动变量`auto`，外部变量`extern`，静态变量`static`，寄存器变量`register`。

1. 自动变量是最常使用的量，函数内部的声明允许省略关键字auto。其作用域为**块范围**，局限于定义它的代码块，退出块时变量的值就丢失了。重新进入块时编译器会为变量再次分配空间，但原先的值已经没有了。

2. 外部变量在函数外定义，定义时**不能**使用关键字extern，只是在引用时加上extern。它的作用域是文件范围，从定义处开始到源文件结束。若想引用其他文件的或定义点前的外部变量，需要加上extern。

3. 静态变量可以是全局变量，也可以是局部变量。静态变量和外部变量一样放在静态数据区生存期与其相同，但作用域不同。静态变量的作用域是块作用域，但程序执行期间它的值不会消失，有记忆性，再次进入块时，依然是原先的值。如果它在块中被定义，那么只做一次赋初值，再次进入块时不再赋初值，沿用上次在块中最后的计算值。

4. 寄存器变量只能用于定义局部变量，存储在寄存器中，提高执行速度，其余特性等同于自动变量。

例2

```
(2019-2020期末)程序改错
函数fac_sum计算并返回1!+2!+...+n!。例如，第一次调用fac_sum(4)返回33，第二次调用
fac_sum(3)返回9.
unsigned long fac_sum(int n){
    static unsigned long s=0,f=1;
    int i=1;
    do{
        f*=++i;
        s+=f;
    }while(i<=n);
    return s;
}
```

解析：实际考察存储类型。程序中do-while是在f、i初值为1，s初值为0时可实现阶乘之和。但是静态变量static使得函数返回后 s、f 的值仍为上次的计算结果，再次进入时不会再被赋初值，影响了函数功能的实现。

正确答案：`unsigned long long s=0,f=1`。

本题也可能改编成写程序运行结果题，要注意避免想当然，无论如何按照计算机执行顺序读一遍。

2.3 二维数组及其应用

二维数组实质上等价于指向数组的指针而非二级指针。其数组名首先是首行地址，其次才可看做首元素地址。

二维数组的逻辑结构是具有行和列的二维数据分布，类似于矩阵，两个下标可以看作是它们的坐标。而其存储结构是一维的，先存储首行元素，紧接着存储第二行、第三行……所以二维数组的任意值可以使用一级指针来访问。

二维数组作为函数的形参时，正确的声明形式如下：

```
数据类型名 数组名[][整型常量表达式]
```

即列数不可省略。

如果使用const、static、extern对数组进行定义或声明，那么数组中的**每一个元素**都具有这些储存类型的性质。

数组在进行初始化时，如果整体未赋初值，那么每个元素的具体值不可知，如果只对某些元素赋值，那么其余值为0。初始化赋值时也可以空缺特定的数，空缺的数为0。对于二维数组，第一维

下标可以省略，第二维下标不能省略。

例3

```
(2016-2017期末)对二维整型数组A的部分元素初始化的形式有()
A.int A[2][3]={{,2,3},{4,6}}    B.int A[2][3]={1,2,3,4}
C.int A[][]={{1,2},{4,5}}       D.int A[][3]={{1,2,3},{4,5}}
```

解析：二维数组初始化可以缺行，但不能缺列，故C错误。数组初始化时，缺失元素不能位于初值列表的前面或中间，只能在最后，即只有后面的可以省略，A错误

本题答案选BD。

例4

```
(学解考试宝典)若有说明：int a[][4]={1,2,3,4,5,6,7,8,9,10,11,12};则数组第一维
的大小为()
A.2        B.3        C.4        D.不能确定的值
```

解析：当第一维下标缺失时，正如一维数组的下标缺失，系统会根据你的初值列表确定你的数组大小。这里一共有12个元素，列数为4，故行数自然为3。

本题答案选B。

2.4 字符串操作

字符串是最常见的数据结构之一。字符串的实现方式有两种，字符数组或指针形式。对指针形式的字符串，赋初值后就不能再修改，如

```
char *s="aaaa";
s[0]='b';
```

这是不合法的。

对于字符数组定义的字符串，注意字符串的末尾有一位`\0`，确保不会发生越界。

"string.h"里有许多字符串处理库函数，要明白每个函数是什么意思。

```
int strlen(char *s);/*求字符串s的长度，不包括'\0'*/
char *strcat(char *s,const char *t);
//字符串连接，把t放到s的结尾并返回s的地址
int strcmp(const char *s1,const char *s2);
//字符串比较，字符串相同则返回0，若s1的ASCII码大于s2，则返回大于零的值
//否则返回值小于0
char *strcpy(char *t,const char *s);
//字符串复制，将字符串t的内容复制给字符串s，并返回s的地址
char *strstr(const char *s,const char *t);
//查找子串，s中若有t出现，则返回t在s中第一次出现的位置，否则返回NULL
```

还有一些平常写程序时可能会用到的好用的函数，在"stdlib.h"里。

```
char *itoa(int iv,char nstr[],int radix);
//将iv的值转化为字符串，进制是radix
char *ltoa(long iv,char nstr[],int radix);
//只是把int变成了long
int atoi(const char nstr[]);
//把字符串nstr转化为整型数，函数返回转换后的值
int atol...//同理
double atof(const char nstr[]);
//把字符串nstr转化为双精度浮点数
```

例5

```
(学解考试宝典)下列程序运行的结果为()
#include<stdio.h>
#include<string.h>
main(){
    int i;
    char a[]="How are you!";
    for(i=0;a[i];i++)
        if(a[i]==' ')
            strcpy(a,&a[i+1]);
    printf("%s\n",a);
}
A.are you!    B.Howareyou!    C.areyou!    D.you!
```

解析：程序运行结果题避免想当然！如果不认真读，很可能以为这个主函数是遇到空格就把后面的字符串取代整个字符串，就会理所当然地选D。但注意，当程序运行到第一次发现空格时，i=3，这是将"are you!"赋给a，继续循环，此时i=4，a[i]='y'，后面没有空格，于是"are you!"就是最终结果。

例6

```
(学解考试宝典)若有以下说明语句，则输出结果是()
    char sp[]="\t\b\0123\n";
    printf("%d",strlen(sp));
A.11        B.7        C.6        D.5
```

解析：~~还得是学解啊~~。'\t','\b','\n'都好理解，是三个转义字符。八进制转义字符最多有三位，故'\0123'肯定不是一个转义字符。由于转义字符的最大匹配机制，1,2均未超出八进制数字范围，故取到三位，即'\012'是一个字符，'3'是一个字符。

如果是"\0183"，那么'\01'是一个字符，'8','3'各是一个字符。

本题答案选D。

2.5 指针与数组

一级指针可以模拟一维数组。而虽说二维数组是指向数组的指针，但依然可以用指针数组、二级指针去模拟二维数组。

例7

```
(2019-2020期末)已知int a[5][3],表达式a和a[0]有何异同？写一个表达式用于计算
二维数组a的行数。
```

解析：对于二维数组，a和a[0]都是二维数组的首行，实际上也都是数组首元素的地址，但作为指针，a指向一个包含三个元素的数组，而a[0]指向这个数组的第一个元素(一维数组的数组名是首元素的地址)。

计算行数：`sizeof(a)/sizeof(a[0])`。

例8

```
(2014-2015期末)设有声明：int a[2][3]={{1,2,3},{4,5,6}},*p=&a[0][0];则表达式
的值为5的选项有()
A.*(a[1]+1)    B.*(p+4)    C.*(a+1)[1]    D.p[1][1]
```

解析：顾此失彼是大忌。A、B都好理解。对于C，由于[]的优先级高于*，因此该表达式计算顺序是`*((a+1)[1])`，指到了a逻辑上的的第3行。而p作为一级指针，没有这种用法。

作为对照，如下程序段：

```c
#include<stdio.h>

int main(){
    int a[3][3]={{1,2,3},{4,5,6},{7,8,9}};
    int (*p)[3]=a;
    //int **p=a;运行错误
    printf("%d %d %d",*(a+1)[1],(*(a+1))[1],p[1][1]);
    return 0;
}
```

运行结果为`7 5 5`。

本题答案选AB。

2.6 复杂指针声明

这里介绍除了指向基本类型的一级、二级指针外的其他所有形式的指针。

1. 指针数组

   `类型区分符 *数组名 [常量表达式]`，如`int *p[3]`。

   这句话的意思是定义一个具有三个元素的数组，数组元素的类型为整型指针。

   指针数组可以用来模拟二维数组、字符串数组。

2. 指针函数

   `类型区分符 *函数名(形参表)`，如`int *pfun(int,int)`。

   这句话的意思是声明一个函数，函数的参数为两个`int`型，返回值的类型为`int *`。

   最大的特点是返回指针。

3. 指向函数的指针

   `类型区分符 (*标识符)(形参表)`，如`int (*comp)(char *,char *)`。

   这句话的意思是定义一个指向函数的指针，该指针指向的函数有两个`char *`类型的参数，返回值是`int`型。

   主要用于**散转程序**，实际上往往是以指向函数的指针数组形式出现。

4. 指向数组的指针

已经提过多次，是二维数组的指针本质。

`类型区分符 (*标识符)[常量表达式]`，如`int (*p)[3]`。

这句话的意思是，定义一个指向数组的指针，该指针指向的数组是有三个元素的一维数组，元素的类型是`int`型。

主要用于模拟二维数组。

5. 函数指针数组(指向函数的指针数组)

   `类型区分符 (*标识符[常量表达式])(参数列表)`，如`char *(*pf[3])(char *p)`。

   这句话的意思是，定义一个有三个元素的数组，数组的元素类型为指针，这些指针指向以`char *p`为参数，返回值为`char *`的函数。

   用于散转程序

6. 指向函数的指针函数

   ~~开始阴间起来了。~~

   `类型区分符 (*标识符(参数列表))(参数列表)`，如`int (*f(int))(int *,int)`。

   这句话的意思是，声明一个函数，该函数以一个`int`为参数，返回值为**指向函数的指针**，该指针指向以一个`int *`，一个`int`类型为参数，返回值为int的函数。

7. 函数指针数组的指针

   `类型区分符 (*(*标识符)[常量表达式])(参数列表)`，如`char *(*(*pf)[3])(char *p)`

   这句话的意思是，定义一个指针，该指针指向具有三个元素的数组，元素类型为指向函数的指针，这些指针分别指向一个以`char *p`为参数，返回值为`char *`的函数。

判断方法：

1. 首先判断声明/定义的是函数还是指针，如果指针后面紧跟参数列表，则是函数声明。

   如果是指针，判断是不是指针数组，如果指针后面直接跟`[常量表达式]`，则是定义了一个指针数组。

2. 然后看指向的是函数还是数组，如果指针加上括号后紧跟的是参数列表，则指向的是函数，然后找参数和返回值；如果紧跟的是`[常量表达式]`，则指向数组，然后找数组的元素类型。

3. 对于指向函数的指针，对其返回值重新判断；对于指向数组的指针，对其元素类型重新判断。

掌握这些例子和方法，判断或写出指针声明就会变得相对有迹可循。

例9

```
(2015-2016期末)有声明int *(*p[10])(void)，写出p的完整含义。
```

解析：首先看指针后面，未紧跟参数列表，而是紧跟中括号常量，故这实质上是声明指针，而且是指针数组。再来看它的指向，后面跟的是`(void)`，故它们指向的是函数。

最终确定答案：定义一个具有10个元素的数组p，数组中元素的类型是指针，这些指针分别指向一个以`void`为参数，返回值为`int *`的函数。

2.7 不同型指针移动

我们已知指针的移动是因其类型不同而变化的，对于同一个`p+1`，如果指针类型为`char *`型，那么它移动一个字节；如果指针类型为`int *`，那么它移动四个字节。

在实际应用中，并不总是用指针指向与它类型相同的元素，例如我们可以用`char *`型指针指向`int`元素，起到取出它的每一个字节的作用。

例10

```
(2015-2016期末)设有声明：
int a=0x12345678,*pa=&a;
unsigned char *pc=(unsigned char *)pa;
则表达式*(pc+2)的值是()
A.0x12    B.0x34    C.0x56    D.0x78
```

解析：不得不说到高位优先与低位优先的问题。

> 大多数计算机按高位优先顺序存储32位的数，但基于Intel CPU的计算机按低位优先顺序存储32位的数。来源

~~然而我高贵的AMD锐龙5800也是低位优先。~~

也就是说，对于相同的数据，比如`short a=1`，

高位优先：`00000000 00000001`

低位优先：`00000001 00000000`

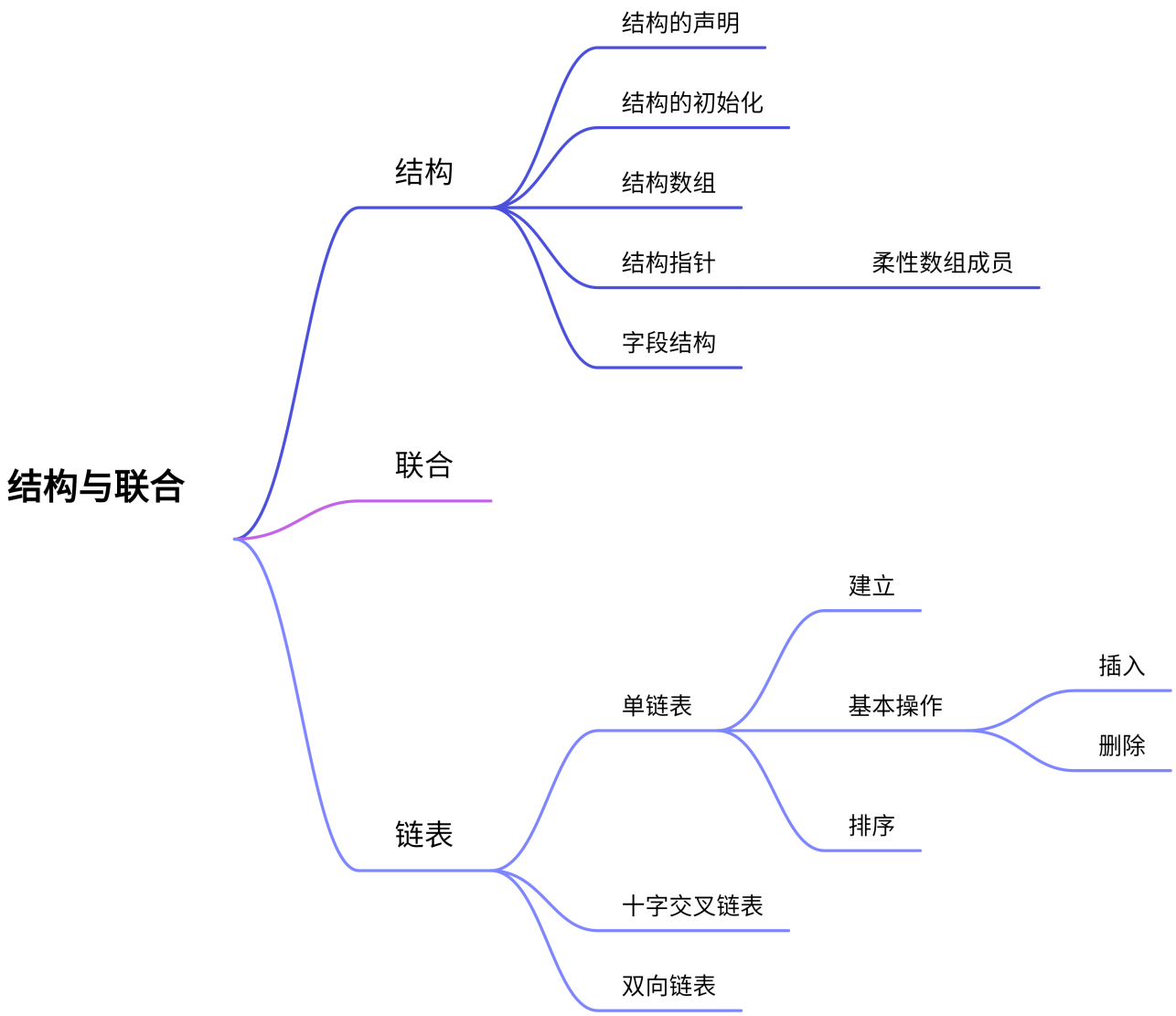
注意：高低位以八位(一个字节)为界。

那么这一题就不用多说了，pc向右移动两个`char *`的空间，也就是`0x34`。

END

结构与联合

思维导图



概述

结构与联合是同指针一样的C语言集大成的章节。首先作为一种构造类型，结构与联合体的内部元素大部分是第二章接触到的基本类型(也可能是第八章的指针)，那么这些内部元素**相对独立**地拥有自己原有的特性。而作为整体，结构自身有其特有的意义，同时也有“结构数组”、“结构指针”这些类型。这些特性使得结构体成为实现 `链表` 的最佳方式之一，而链表也是许多 `数据结构` 的实现方式。因此结构是综合运用C语言所学的，为其他课程做铺垫的重要一章。

1 边角知识

1.1 柔性数组成员

C99新增功能，支持动态结构类型，结构中最后一个成员可以是不给出为大小的数组，这种数组成员在标准中称为柔性数组成员。如：

```
struct stud{
    char name[20];
    int score[];
}
```

但是这种结构的空间不包含柔性数组的空间，即如果要使用数组`score`，需要用`malloc`为其分配空间。如：

```
struct stud *p;
p=(struct stud *)malloc(sizeof(struct stud)+3*sizeof(int))
//如果数组的大小是3的话
```

---

2 重点知识与例题分析

2.1 结构的定义与初始化

结构和数组一样，都是一种构造类型，但结构内数据的类型可以不同。结构声明的一般形式为：

```
struct 结构类型名{
    成员声明表
}[标识符列表];
```

我们同样可以在结构声明时进行初始化，如：

```
struct planet{
    char name[10];
    double diameter;
    int moons;
} x={"Jupiter",142987,79};
```

这样的初始化是顺序的，分别表示x的name、diameter、moons。

结构也可以嵌套，这里不展开举例。

结构也可构造为数组，称为结构数组。结构数组的初始化在形式上与二维数组类似，如：

```
struct stud students[]{
    {"20160805001","Zhang",'m',{5,6,1998},0};
    {"20160805002","Li",'f',{8,20,1998},0};
}
```

与二维数组的初始化一样用到了双层花括号(最内层包含三个数字的花括号是因为`stud`结构体中含有结构类型的元素)，也与二维数组一样，内层的括号可以不写出来，只是方便辨认。

```
没有错题
```

2.2 字段结构

第二章位运算提到，字段结构也可以实现空间压缩。事实上压缩空间是字段结构的主要内容，它的一般形式是：

```
struct 类型名{
    类型区分符 标识符:位数;
    ...
}标识符;
```

这表示所有的元素公用一个[类型区分符]的空间，位数之和可以不足一个[类型区分符]的位数，但一旦超越这个位数，就会多占用一个[类型区分符]的空间。比如：

```
struct{
    unsigned short a:4;
    unsigned short b:4;
    unsigned short c:4;
}flag;
```

上面的字段结构位数之和为12，不足16，仍按一个`unsigned short`的大小分配，但如果结构里新增一句：

```
    unsigned short d:8;
```

那么就要分配两个`unsigned short`的空间。字段结构的每一个元素都要保证不能超出所规定的二进制位数。

例1

```
(2012-2013期末)设有以下声明：
struct T{
    unsigned short a:1;
    unsigned short b:2;
    unsigned short c:3;
    unsigned short d:4;
    unsigned short e:6;
}x,*p=&x
则下面对字段变量各成员赋值正确的有()
A. x.a=2    B.p->b=3    C.*p.c=4    D.x.d=5
```

解析：最常见的错误超出位数，这里A便不合法。其次还要注意每个表达式是否是合法的赋值表达式。C项中`*`的优先级比`.`要低，因此先运算点运算符，在取`*`，显然是不合法的。

本题答案选BD。

例2

```
(2019-2020期末)请写出下面程序的运行结果。
#include<stdio.h>
char a[]="0123456789abcdef";
struct s{
    unsigned char s1:4;
    unsigned char s2:4;
    unsigned char s3:4;
    unsigned char s4:4;
};
union u{
    short sh;
    struct s st;
}n;
int main(){
    int m=0x1234;
    n.sh=m;
    printf("%hd\n",n.sh);
    putchar(a[n.st.s1]),putchar(a[n.st.s2]),putchar(a[n.st.s3]),putchar(a[n.st.s4]);
    return 0;
}
```

解析：综合考察联合与字段结构。在联合体中，`(struct s)st`与`sh`所占空间相同，均为2字节，故联合体大小为两字节。这里依然考到了指针里出现的低位优先。由于`sh`与`st`共享内存，故

对`sh`赋值也是对`st`赋值。`st`中一个字段大小恰为4位，也就是一个十六进制位，因此从低位到高位，s1~s4分别为4,3,2,1，在数组a中分别对应4,3,2,1。

本题答案为：4660，4321。

2.3 联合

前面字段结构的例题已经涉及了联合的特征。联合同结构一样也是一种构造类型，但联合的所有成员共享一段内存，所有成员的首地址相同，联合的大小由所占字节数最大的成员决定。对于联合同样有"大小端"，即高位/低位优先的说法。课本上有例题利用union判断CPU的大小端，但实际做题时好像默认低位优先。

例3

```
(2012-2013期末)设有如下声明：
union U{
    long a;
    short b;
    char c;
    char s[20];
}v={0x01020304},*p=&v;
则下列选项正确的是()
A.printf("%d\n",sizeof(v))输出4     B.printf("%d\n",p->s[0])输出4
C.printf("%d\n",v.c)输出1           C.printf("%x\n",v.b)输出102
```

解析：联合的大小由占字节数最大的成员决定，联合U中，最大字节为20(char s[20])，故`sizeof(v)==20`，由于低位优先，s[0]、c指向最低8位，即4；b指向最低十六位，即0x304。

本题答案选B。

2.4 链表

期末串讲时课程组老师说只会考单向链表，因此这里不再介绍十字交叉链表和双向链表~~(但双向链表确实好用)~~。

链表的建立、插入、删除、排序操作要记牢。

```
//基本类型
struct List{
    int val;
    struct List *Node;
};
```

```
//建立
void CreateList(){
    struct List *head=NULL,*p,*tail=NULL;
    int x;
    scanf("%d",&x);
    while(x){
        p=(struct List*)malloc(sizeof(struct List));
        p->data=x;
        if(head==NULL)
            head=p;
        else    tail->next=p;
        tail=p;
        scanf("%d",&x);
    }
    if(tail!=NULL)
        tail->next=NULL;
}
```

删除时，注意需要有一个额外的指针指向当前节点的上一个节点，否则链表将断开。

```
题目详见“常见题型与分析”
```

END