C++程序设计

课程简介

课程沿革

课程要求

课程内容

教学内容	讲课学时	上机学时
计算机基础知识	4	
C++入门与基本数据类型	4	
表达式和语句	6	4
函数和程序结构	8	4
数组和结构	6	4
指针和引用	8	
面向对象程序设计	3	
类与构造函数	8	4
堆与拷贝构造函数	2	
继承	4	4
I/O流	3	4
合计	56	24

第1章 计算机基础知识

1.1 计算机的基本组成

1.1.1 计算机硬件

计算机的发展已经到了第四代，但其结构仍然是“存储程序式计算机”，又称冯.诺依曼式计算机。它一般由控制器、运算器、存储器、输入和输出设备等五大部分组成。

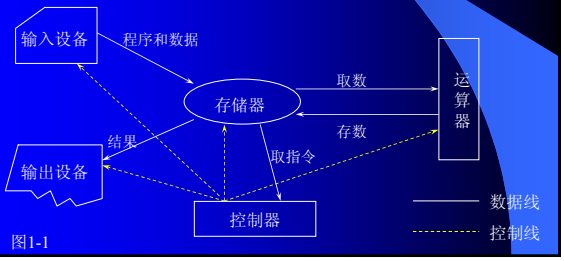


图1-1

- 存储器：用来存储程序和数据(又分：高速缓存、主存(内存)、辅助存储器(外存)及海量存储器(磁带、光盘)等)
- 运算器：用来执行规定的运算。
- 控制器：指挥各部件按程序要求实现自动操作

> CPU

- 输入/输出设备：用于输入原始数据和输出运算结果

1.1.2 计算机软件

即计算机中的程序、数据及有关文档的集合。

- 系统软件：直接控制和协调计算机硬件，其它软件均通过它发挥作用的一类软件。如操作系统、编译程序等。
- 应用软件：在特定应用领域，解决用户具体问题的软件。如各种管理信息系统、财务软件等。

1.1.3 关于地址、文件路径

1.2 关于二进制

1.2.1 什么是二进制

我们日常所用的计数制是十进制，即逢十进一，每一位的基数是十的若干次幂，如1980＝$1\times10^3+9\times10^2+8\times10^1+0\times10^0$

所谓二进制与之类似，但它是逢二进一，每一位的基数是二的若干次幂，如$(11010)_2=1\times2^4+1\times2^3+0\times2^2+1\times2^1+0\times2^0=(26)_{10}$

1.2.2 计算机中为什么要使用二进制

因为数字在计算机中是以电子器件的物理状态来表示的，二进制只需0和1两个数字符号，可以用电子器件的低电平和高电平两种不同的状态来表示。其运算电路容易实现且运行可靠。而要制造出具有10种稳定状态的电子器件来表示十进制中的10个数字符号是非常困难的。

1.2.3 二进制与十进制的相互转换

采用“除2取余法”将十进制整数转换成二进制整数：

采用“乘2取整法”将十进制小数转换为二进制小数。

例如十进制数39：

2 ⌊ 39	余数为1，即$a_0=1$
2 ⌊ 19	余数为1，即$a_1=1$
2 ⌊ 9	余数为1，即$a_2=1$
2 ⌊ 4	余数为0，即$a_3=0$
2 ⌊ 2	余数为0，即$a_4=0$
2 ⌊ 1	余数为1，即$a_5=1$
0	商为0，结束

结果为 $(39)_{10}=(a_5a_4a_3a_2a_1a_0)_2=(100111)_2$

例如十进制小数0.25：

0.25	
× 2	
0.5	整数部分为0，即$a_{-1}=0$
× 2	
1.0	整数部分为1，即$a_{-2}=1$
0.0	余下的小数部分为0，结束

结果为 $(0.25)_{10}=(0.a_{-1}a_{-2})_2=(0.01)_2$

对一般十进制数，将整数部分与小数部分分别转换，然后再组合起来。例如：$(39.25)_{10}=(100111.01)_2$

采用按幂次展开的方法即可将二进制数转换为十进制数。例如：

$(100111.01)_2=1\times2^5+0\times2^4+0\times2^3+1\times2^2+1\times2^1+1\times2^0+0\times2^{-1}+1\times2^{-2}=(39.25)_{10}$

1.2.4 八进制、十六进制及其与二进制的相互转换

由于二进制书写复杂，不易读，而八进制和十六进制与二进制有着很简单的对应关系，且书写较简单，所以常用它们表示。

- 八进制：“逢八进一”，0，1，2，3，4，5，6，7。
- 十六进制：“逢十六进一”，0，1，2，3，4，5，6，7，8，9，A，B，C，D，E，F。

与十进制之间的相互转换与二进制类似。

与二进制之间的相互转换：

(1)八进制到二进制：因为$8=2^3$，所以将每位八进制数用相应的三位二进制数代替即可，例如$(15.2)_8=(001101.010)_2=(1101.01)_2$

(2)十六进制到二进制：因为$16=2^4$，所以将每位十六进制数用相应的四位二进制数代替即可，例如$(2B1)_{16}=(001010110001)_2$

(3)二进制到八进制：从个位起每三位二进制数转换为一位八进制数即可，如最后一组不足三位时，添0补足三位。例如：

$(1101.01)_2 = (15.2)_8$

(4)二进制到十六进制：从个位起每四位二进制数转换为一位十六进制数即可，如最后一组不足四位时，添0补足四位。例如：

$(1101.01)_2 = (D.4)_{16}$

1.3 各种信息在计算机内部的表示方法——信息编码

1.3.1 字符编码

除数字外，各种字符，包括汉字，在计算机中也都是用二进制表示的，为便于处理，必须为字符、汉字等建立统一的编码标准。

8个二进制位为一个字节，1024个字节为1KB，1024KB为1MB。

- 字符：ASCII码，每个字符用一个字节表示。有一个码值。
- 汉字：国标码，BIG5码等，一个汉字用2个字节表示。

1.3.2 数字编码

数字在计算机内部用1～10个字节表示。

- 原码：用除2取余法得到的二进制数，称该数字的原码。
- 反码：将原码按位取反，得到的就是该数字的反码。
- 补码：正数的补码＝原码；负数的补码 ＝ 反码加1。

在计算机内部，任何数字均用其补码表示。

例：假如用1个字节表示数字，

则 12在计算机内部的编码为 00001100

而-12在计算机内部的编码为 11110100

最高位(红色)就称为符号位。

1.4 计算机语言—— 沟通人和计算机的桥梁

- 机器语言：计算机直接能懂的二进制形式的语言。
- 汇编语言：采用助记符，与机器指令一一对应，需要汇编程序。
- 高级语言：类自然语言风格，易学、易记，需要编译程序。
- C与C＋＋：高级语言，C＋＋是C的超集。见P3。

第2章 C语言概述

2.1 C语言的由来和发展

C语言的开发史源于高级语言和UNIX操作系统的发展要求。

早期的系统程序设计，使用的是汇编语言，其优点：

(1)能体现计算机硬件指令级的特性，表达能力强；

(2)运行效率高。

其缺点：可读性，可移植性及描述问题的性能不如高级语言。

这样很自然有如下想法：能否用具有足够表达能力的高级语言来进行系统软件的设计呢？Bell实验室做了这一尝试。

1970：在 PDP-11/20机上实现了B语言，并用它编写了UNIX系统的实用程序。

B<---BCPL<---CPL<---ALGOL 60

1969 1960

在B语言基础上，改进其缺陷，发展出了C语言，其设计目标：
(1)保持BCPL和B的精练性及接近硬件的特点
(2)恢复这些语言失去的通用性
1972：第一个C编译投入使用
1973：UNIX用C改写，加入多道程序功能，发生质变
现在，UNIX已得到广泛推广，成为公认的第一标准的操作系统，随着UNIX的进一步开发，C也交织在一起被迅速推广。
1983：对C扩充，发展为C＋＋。经过不断完善，现仍在发展中。

2.2 C语言主要特点

- 简洁、紧凑、使用方便灵活
- 运算符丰富
- 数据结构丰富
- 支持结构化程序设计
- 语法限制不太严格，编程自由度大
- 允许直接访问物理地址，接近底层
- 目标代码质量高，程序执行效率高
- 可移植性好

C＋＋语言包括了C的所有特征、属性和优点，同时改进了C的一些不足，并且支持面向对象的程序设计。

第3章 C语言程序简介

3.1 C语言程序例
(1)显示字符串——最简单的C程序 p7
说明了：(1)注解 (2)main函数 (3)语句括号{}

(2)输入两个整数，计算一个表达式的值 p8
说明了：(1)变量说明 (2)输入cin (3)计算表达式 (4)输出结果

(3)求两个数中较大者的平方根，稍复杂些 p9
说明了：(1)流程控制，if语句
(2)C程序结构，例中有一函数max(x,y)，负责求x，y的最大值(注意参数说明)。

由以上几例可以看出：
(1)C的基本程序单位是函数：每个C程序有且仅有一个main函数
main+若干子函数＝＝>C程序

(2)C 从语法上无函数与过程之分，统称为函数。无返回值的函数其实就是一个过程，C的新标准及C＋＋中要求将这种函数的返回值定义为void类型。
(3)所有函数的地位都是相同的，不允许嵌套定义。
(4)注解：增加易读性。
(5)自由书写格式：一语句可分几行，一行可写多个语句。一般要求“缩排”。

3.2 C语言语法描述

最常用的语法规则描述工具是Backus-Naur范式(BNF)和语法图两种，由于后者更加直观，本课采用语法图描述。
- 圆圈，即 ⬭ 或 ⬭ ：表示基本符号
- 方框，即 ▭ ：引用另一个语法图
- 箭头，即 ⟶ ：语法的合法走向

3.3字符和词法单位

一. 语言的最基本单位是字符：
字符——>词——>语句——>函数——>程序。C语言采用ASCII字符集。

二. 词法单位：
(1)标识符：在高级语言中，任何对象都有一个“名”——标识符
这些对象包括：函数、变量、符号常量、数组、数据类型、宏等
标识符其实就是字母打头的字母数字序列，注意：
(a)长度：早期的C编译程序只认前8个字符。VC允许256个字符。但一般不要超过31个字符为宜
(b)约定：变量名、函数名、和数据类型名等用小字母；
符号常量名及宏用大写字母；
下划线开头的名字常为系统程序(库函数用)，用户一般不用。
(c)要有意义：如用Sum表示“求和变量”，不要用aaa，A1，B12之类名字。参见p15～p16。

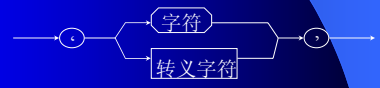
(2)关键字：也称保留字，程序员不可用之作为自己定义的变量或函数的名字。P12_表2-1。
(3)界限符：运算符、分隔符、语句括号等。
(4)常量：

- 数值常量
  - 整常数
  - 实常数

p17

- 字符常数
- 字符串常数
- 符号常数

(a)字符常量：



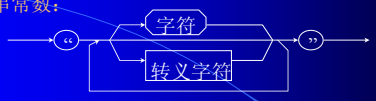
- 每个字符常数占一个字节
- 每个字符常数有一个编码数字值——ASCII码表中的码值
  例如：‘A’:65 ‘B’:66 ‘a’:97 ‘b’:98 . . .

- 字符常量可象数值量一样进行运算、比较
  例1：if(c>=‘A’&&c<=‘Z’) c=c+‘a’-‘A’;
  将字符变量c从大写字母转为小写字母。
  例2：‘0’-‘0’=0
  ‘1’-‘0’=1
  ‘2’-‘0’=2
  ⋮
  如果c是一数字字符，则c-‘0’就是它对应数字的数值。

- 转义字符：非图形字符(如回车、制表符等)及\‘“等字符，因它们有特殊作用，所以表示这些字符本身时，应：p18。

- 位模式：用来表示无法用其它方法表示的字符，\ddd：一至三位8进制数。
  例如：‘\001’表示码值为$(01)_8$的ASCII字符。
  ‘A’也可用‘\101’表示
  ‘\0’为空白符，即NUL，码值为0，在C语言中是字符串的结束标志。(与‘0’和空格符不同)

(b)字符串常数：



如“ABC”、“\n GOOD BYE”、“”等。
注意“P”与‘P’不同。前者为字符串，在机内存储形式为 | P | \0 |，而‘P’就是一常数P。

(c)符号常数：为便于编程，为某个常数所起的符号名。
例如：#define PI 3.1415926
参见p20常量定义，C＋＋中新增功能，同PASCAL中的常量。

3.4 基本的输入和输出处理
3.4.1 I/O流控制
这是C＋＋中新加的非常简单易用的输入、输出方式。
1)用插入操作符“<<”向标准输出流cout输出。 p22。

2) 用抽取操作符“>>”从标准输入流cin中输入。 p22。

3) 使用控制符
有时缺省输出格式不能满足要求，这时可以用控制符控制输出的格式。 p23表2-4。例ch2_1.cpp。

3.4.2 传统的C语言标准输入/输出函数
C语言没有专门的I/O命令，利用库函数实现数据的输入输出。

1) 用printf函数进行数据的输出
形式：printf(格式控制串，参数1，参数2，… …)
按格式控制串指定的格式在标准输出设备上输出各参数所指定的内容。 p28例及格式符介绍。

2) 用scanf进行变量数据的输入
格式：scanf(格式控制串，参数1，参数2，… …)
格式串的意义与printf中一样，但其转换方向是输入到变量。要特别注意参数中的取地址操作&是必不可少的。 p31例。

3) 用getchar和putchar进行单个字符的输入和输出。

- 形式：getchar( )

从终端输入字符流中取下一字符，并返回其代码值，用法：

```
char c;
c=getchar();
```

- 形式：putchar(c)

在终端上输出字符型变量c所代表的字符。
相当于printf(“%c”,c);

例：

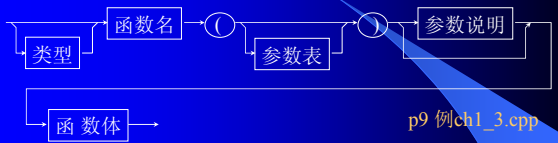
```
#include<stdio.h>
main()
{char c1,c2,c3;
 c1=getchar();  c2=getchar();  c3=getchar();
 putchar(c1);   putchar(c2);   putchar(c3);
 putchar('\n');
}
```

3.5 函数的初步介绍

函数是C程序的基本组成单位，有两种函数：

(1)系统定义的库函数(如printf、scanf等)

(2)用户在程序中定义的函数



p9 例ch1_3.cpp

调用与返回：从main函数的第一句开始执行，遇return语句返回.

3.6运行C语言程序的全过程

p6图1-1。　　编辑→编译→调试→运行

第4章 基本数据类型和运算表达式

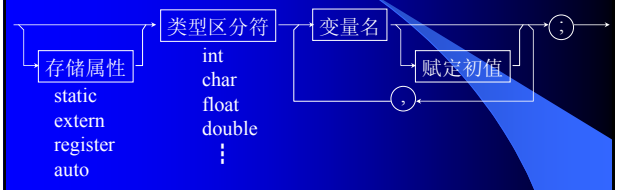
数据类型 (p14表2-2)

- 标准数据类型
  - 整型：短整、长整、无符号
  - 实型：单精度、双精度
  - 字符(char)
  - 逻辑型(bool)(C++)
- 组合数据类型
  - 枚举型
  - 数组
  - 结构(struct)
  - 联合(union)
  - 类(class)(C++)
- 指针类型
- 空类型(void)
- 用户定义数据类型

4.1 变量和类型说明

常量：固定不变的值，C中用符号常数来表示，无须定义类型；C＋＋中可以定义常量(const)。

变量：需要定义，说明其存储属性和数据类型。



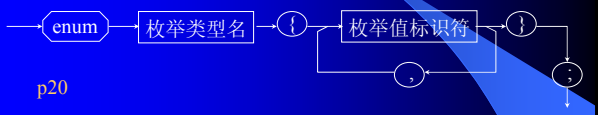
例：static　int　nward=0, nkayword;

4.2 标准数据类型　p14表2-2

- 注意整型和实型的几种不同方式。
- 注意各种数据类型的取值范围。

• 注意C无布尔类型。C语言在表达逻辑运算的结果时，用整数0代表逻辑“假”，用非0的整数值(如1)表示逻辑“真”。
C++中增添了布尔类型bool，但并非每个C++编译器都支持。

4.3 枚举类型
实际上是一种用户定义的数据类型，是C语言新版本中新增的。



p20

4.4 数据类型转换
C语言中，当表达式中对象类型不一致时，要进行转换，规则：
(1)char, short → int ; float →double (必定的转换)
(2)级别由低 → 高，表达式最后值是级别高的那个类型。p38
(3)赋值表达式E1＝E2，将E2结果值转换成E1类型后进行赋值。
(4)强制类型转换：(类型名)表达式。

虽然C在对不同类型对象混合运算时，能自动进行类型转换，但编程时仍应尽量避免。

4.5 运算符及运算表达式 *
表达式就是由操作数、运算符构成的一个操作序列，P35表3-1列出了C++的全部运算符的功能、优先级和结合性规则。

(1) 算术运算符：
+ - * / %(取模)
在C中，若 / 的两操作数都为整数，则为整除：

例：
```
int  i = 15;
float  f  ;
f = i/10;
printf (“f=%f\n”, f);
```
则结果为：f=1.000000　如果将第三条语句改为：
f = (float)i/10; 或f = i/10.0;
则结果为：f=1.500000

(2) 关系运算符
> >= < <= == !=
注意= =在程序中不要误用为= 。

(3) 逻辑运算符：
&& ‖ ！ 真值表如下(T表示“真”，F表示“假”)：

a	b	!a	!b	a&&b	a‖b
T	T	F	F	T	T
T	F	F	T	F	T
F	T	T	F	F	T
F	F	T	T	F	F

注意：
对mini <= value <= max C编译不会报错，但结果不对，所以编程时要注意，不要以为C可以这样用。
应该用mini<= value && value <= max进行判断。

(4) 增1和减1运算

即++和--运算。C特色，对于构造简单灵活的表达式很有用。

两种用法 { 前缀用法 ++i，--i　先加1，再引用i的值
　　后缀用法 i++，i--　先引用i的值，再加1

都是将变量i的值加1(减1)。

例：i=10;
　a=i++;
　则a的值为10，i的值为11。

　而a= --i;
　则a的值为9，i的值也为9。

　同样：s[i++]=c; 等价于 s[i]=c; i=i+1;
　　s[++i]=c; 等价于 i=i+1; s[i]=c;

注意：(1)++、-- 只适用于整型变量。
　(2)不引用表达式值，只需对变量加1(减1)时，两法相同。
　(3)注意副作用。p40~41，p49

(5) 字位逻辑运算符

&	按位“与”，两操作数按位求与
|	按位“或”，两操作数按位求或
^	按位“加”，两操作数按位求“异或”
<<	将左操作数“左移”右操作数给出的位数
>>	将左操作数“右移”右操作数给出的位数
~	按位求“反”

例：int i=4, j=6;　　i的内码00000000 00000100
　　j的内码00000000 00000110

则　k=i&j 的结果为4
　k=i|j 的结果为6
　k=i^j 的结果为2

- & 常用来屏蔽某些字位，如x=x & ~077 置x的后6为0。
- | 常用于将某些位置为1，如x=x | 02 将x的第2位置为1。
- <<和>>要求右数为正整数且小于左数的字位长度。
　例如 i<<3 相当于 i = i*8(2^3)

所以，字位逻辑运算使C语言能象汇编语言那样灵活地处理硬件机器字。

(6)赋值运算和自反赋值运算

(a)C语言中＝是运算符。p34

(b)C允许多重赋值：x=y= i*j;

(c)自反赋值，C独有的一套运算符。多数二目运算符(op)都有其对应的自反运算符op=，规则：p37

E1 op= E2 等价于 E1 = (E1) op (E2)

例如：i *= i*p+q 等价于 i = i*(i*p+q)
　a+=a-=a*a 等价于 a=a-a*a; a=a+a;

(7) 逗号运算符

用于将两个表达式组合成一个表达式：E1，E2
从左向右计算，结果的类型和值为E2的类型和值。

例如：t=2,t+3 则最后表达式的值为5，t的值为2。

- 在逗号有特殊意义的上下文中，逗号表达式必须加括号：

例如：f1(a,(t=2,t+3),c)

- 常用于for语句中，可控制多个初值或循环控制变量。

(8) 条件运算符

E1？E2：E3

先计算E1的值 { 如为真，计算E2，且此值是该表达式的值。
如为假，计算E3，且此值是该表达式的值。

例：z = (a>b) ? a : b　就是求a和b的最大值，将其赋值给变量z。

4.6 计算顺序和优先级

按照p35表3-1运算符的优先级和结合性：

- 先高优先级，后低优先级。
- 同优先级，按结合性从左到右或从右到左进行计算。
- C运算符及优先级别较多，必要时可加上括号以增强可读性和安全性。

第5章 语句和流程控制

在程序设计语言中，每条语句都是一条命令，要求计算机完成完成一定的功能。若干条语句的合理组合，就描述了我们解决某种问题的算法，构成程序。

5.1 表达式语句

语法：表达式;

注意：(1)关于；号，在C中它是语句的一部分，是语句的"终止符"，而不象PASCAL等语言那样是语句间的分隔符。

(2)要有实际意义，即通过语句的执行使程序的状态发生一定的变化。

如 i++; a=getchar();

而 a+b; 则没有任何意义。

5.2 复合语句和分程序

语法：{----; ----; ----;}　结束复合语句的}之后不需再加;号。

分程序：结构化程序设计的基本设施，变量作用域仅在{ }内部。

语法：
```
{
 变量类型说明
 ------;
 ------;
    ⋮
}
```

例如：{int c; c=getchar(); putchar(c);}

5.3 空语句

语法：仅有一个;号

为goto语句提供标号或辅助其它语句的书写。

5.4 if 语句

语法：if(表达式) 语句1
[else 语句2]

分号不能加

例：if(a>b) z=a;←分号不能少　if(a>b) {z=a; a++;}
else z=b;　else z=b;

p45例。

注意：(1)因为( )中是表达式，所以允许出现＝号，因此要特别当心不要将＝＝误写为＝。

(2)注意分号的问题，C语言中分号是语句的一部分。

(3)p46解决二义性。

5.5 循环语句

循环语句用于控制其它语句，使之重复执行若干次，称为循环。按照控制方式的不同，C语言中提供了三种不同的循环语句。

5.5.1 while语句

语法：while (表达式) 语句S

p54图4-1。例ch4_1.cpp。

5.5.2 do while语句

语法：do 语句S while(表达式)；

p56图4-2。例ch2_2.cpp。

5.5.3 for语句

语法：for([表达式1];[表达式2];[表达式3]) 语句S

p58图4-3，for的执行流程。

注意：C的for语句与其它高级语言中的for语句大不相同。

其它语言中，while语句的循环次数在运行过程中根据情况而定，而for语句则执行事先给定次数的循环。如：

```
FOR i=1 TO 10 DO …
```

但在C中，for语句同时具有while的功能，可以用while描述for的功能：

```
表达式1；
while(表达式2) {
语句S
表达式3；
}
```

总结：C中for的执行机理与while是一样的，但它比while更灵活，它不仅可以用于循环次数已经确定的情况，而且可用于循环次数不定的情况。p58灵活性。

例：for(i=0;(s[i]=getchar())!=EOF;i++);　等价于

```
i=0;
s[i]=getchar();
while(s[i]!=EOF) s[++i]=getchar();
```

- 相关语句：

(1) return语句：return(表达式)或return。

(2) break语句：退出循环体或switch语句。p64。

例：

```
while(1) {
------;
------;
if(---) break;
}
```

(3)continue语句：跳过剩余循环体，进入下一次循环。p64,65。

5.6 开关语句

语法：switch(表达式) {
case 常量1：成分子语句1；break；
case 常量2：成分子语句2；break；
⋮
case 常量n：成分子语句n；break；
[default： 成分子语句n+1]
}

注意：在case子句中要正确运用break语句。 p61。

例：switch(i) {
case 1:
case 2:
case 3:
case 4: a=a+b*10; break;
case 5: a=a*c;
case 6: a++;
}

5.7 goto语句

语法：goto 语句标号(标识符)

p65例。

注意：标号的作用域：当前函数。

5.8 语句使用实例

p66~p73。编程关键：

(1) 确定求解问题的算法。

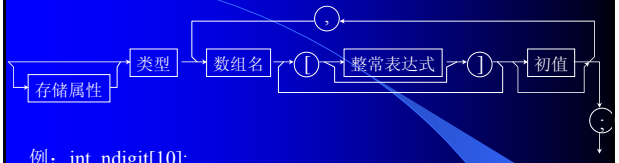
(2) 熟悉各种语句的功能和特性。

(3) 以合理的程序结构(如ch4_9.cpp)利用相关语句将算法描述出来。p73小结。

第6章 数组

数组是由相同类型的成分分量组成的有序数据集合。当需要使用大量集中在一起的数据来工作时，就可以使用数组来满足这一要求。

6.1 数组定义和数组元素



例：int ndigit[10];
static char s[LENG], ds[3][3];
int array[]={1,2,3,4,5};
p120图7-1例。

数组元素引用：数组是由连续的存储单元组成，起始地址为数组的第一个元素，每个元素都有一个下标，它是该元素距离第一个元素的偏移量。数组中特定的元素通过下标访问(p120图7-1)：

注意：(1) C语言中，数组下标是从0开始的，一个长度为n数组，其n个元素的下标分别为0～(n-1)。

ndigit[0], ndigit[1], …, ndigit[9]

ss[0][0], ss[0][1], ss[0][2], …, ss[2][2]　　p123 Fibonacci例。

(2) 当说明中表达式为空，只有[ ]时，数组长度由以下因素决定：

a.如同时给出初值，则由初值个数决定。

b.已在其它场合说明了它的长度(如形参、外部数组等)。

6.2 字符数组

这是C语言中最常用的数组，由于C语言中没有专门的字符串类型，对字符串的处理是通过字符数组进行的。

C语言约定每个字符串以'\0'作为结束标志。

如：char line[100];　可用来存放最长为99个字符的串。

例：p123 ch7-1.cpp。

注意：

(1)不能用scanf输入一个带空格的串(cin>>str也一样)。

如：char str[13];

scanf("%s",str);

如果输入How are you? 则str的内容为：

(2)可以用C的库函数gets(str)或C++的cin.getline(str, 80)输入带空格的串。p124例7_1.cpp。

(3)scanf中的输入项是数组名，不要再加&：

scanf("%s",str);

(4)字符串处理函数

puts(字符串)	strcmp(字符串1, 字符串2)
gets(字符数组)	strlen(字符串)
strcat(字符数组1, 字符数组2)	strlwr(字符串)
strcpy(字符数组1, 字符串2)	strupr(字符串)

6.3 多维数组

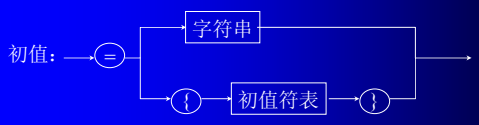
当组成一个数组的每个元素也是数组时，便为多维数组。

```
#define  I  3
#define  J  5
int tab[I][J];
```

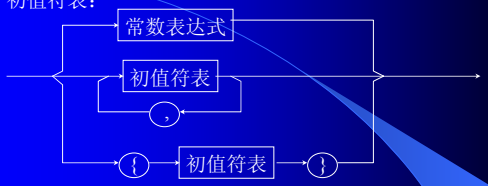
则tab是一个有I*J个元素的数组，共有I行、J列存放时是按行存放的。参见p130图7_4。

- 可以将多维数组看成是元素为数组的一维数组。

6.4 数组的初值



初值符表：

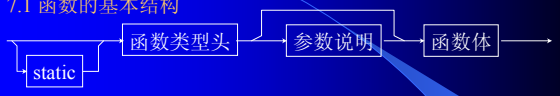


例：char string[]="INTRODUCTION";
char string[]={"INTRODUCTION"};
char string[]={'I','N','T','R', … ,'N','\0'};
int x[2][2]={{1,2,3},{4,5,6}};或{1,2,3,4,5,6};

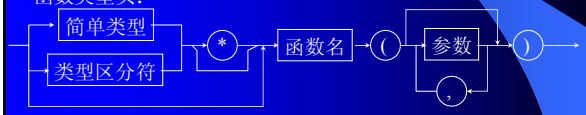
C限制：(1)静态数组 } 才能赋初值。但C++中自动数组也行。
(2)外部数组 } p125例ch7-2.cpp

第7章 函数和程序结构

7.1 函数的基本结构



函数类型头：



p78，几种不同类型的函数。

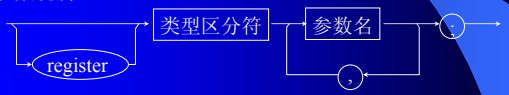
调用函数：

- 按语法图，C语言中不允许在一个函数内定义另一个函数。
- 函数之间的通信：
  (1)通过实参、形参。
  (2)通过返回值。
  (3)通过全程外部变量。

7.2 函数的参数

为使函数具备更大的灵活性，对不同变量进行相同功能的处理。

(1)参数说明：



与变量定义很类似，但存储属性不能是static或extern。

(2)形参的生存期：同所在函数的生存期。

(3)参数的值传送性质：实参、形参之间共有传值、传地址两种传递方式，C语言中只支持传值。

•由于是传值，所以在函数中对形参的改变不影响对应的实参。

•当形参为数组时，对应的实参应为数组名。此时仍然是传值，只不过该值恰好是数组的首地址，因为在C中，数组名就代表该数组的首地址，可以将其当常量来用。

•指针参数的处理：还是传值，此值是该指针所代表的地址值。

所以要切记，C语言中形、实参之间只有一种传递方式，即传值。

(4)注意参数的求值顺序。

(5)某些C编译器允许实参和形参的个数可以不同，当实参个数少时，多出的形参取出的是无意义的数据；当实参个数多时，多给的实参没有用。

(6)p84~85函数调用机制，注意其中形、实参间的值传送方式。

7.3 函数类型

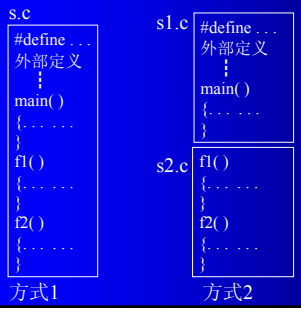
即函数返回值的类型。调用函数前必须对其说明，int型可缺省。

7.4 变量存储属性*

(1)C源文件的组成及编译运行环境

(i)源文件的组成：

一个C程序由若干函数、外部数据定义、预处理部分组成，它们可以存放在一个源文件中，也可以分散存放在多个源文件中：



分成多个源文件的好处：
- 便于多人合作(大工程)
- 便于调试(分步编译)

(ii)C编译

a. 直接编译：cc -o exec s1.c s2.c<回车>

b. 分步编译：先生成浮动安装码，再连接生成目标程序：

cc -c s1.c<回车>　　将生成s1.o(.obj)

cc -c s2.c<回车>　　将生成s2.o(.obj)

cc -o exec s1.o s2.o<回车>

采用分步编译时，如果对f2()中做了修改，只需重新编译s2.c，再与s1.o连接即可。

(2)变量的存储属性

即变量的生存期及作用域：auto, static, extern, register。

7.5 自动变量

即某函数中的局部变量。其生存期同所在函数；作用域在所在函数之内。它是在函数被调用时在栈中临时分配空间的。 p81及p85图8-5，注意其中的局部变量a, b, n, x等在栈中的空间分配。

7.6 外部变量

- 外部变量是在任何函数外部定义的变量，可被多个函数共用。
- 外部变量在程序运行期间“永远”存在，它是在编译时静态分配的空间。如p81程序段中的n。
- 可以通过外部变量在函数间隐式传递数据。
- 如果一个全程外部变量与一个函数(或分程序)中的自动变量同名，在C语言中是允许的，但它们各自的生存期和作用域是不同的。p110~111。

7.7 作用域规则

自动变量 —— 所在函数

外部变量 —— 源文件说明它的地方直到该文件的结束

注意：

(1)下述情况下必须使用extern说明：p101上的n。

a. 定义外部变量的程序和使用该变量的程序不在同一源文件。

b. 外部变量在定义之前就要被某一函数使用。

(2)在组成一个C程序的所有源文件中，对一个外部变量有且只能有一个定义，其余的为extern说明。(变量初值、数组长度等在变量定义时指明)。

s.c

```
int i;
char s[100];
float f=1.5;
main( )
{. . . . . .
}
f1( )
{. . . . . .
}
f2( )
{. . . . . .
}
```

方式1

s1.c

```
int i;
char s[100];
float f=1.5;
main( )
{. . . . . .
}
```

s2.c

```
extern int i;
extern char s[];
extern f;
f1( )
{. . . . . .
}
f2( )
{. . . . . .
}
```

方式2

注意变量的定义和变量的说明不同：

- 定义：要在编译时或在栈中分配空间。
- 说明：引用其它源文件或本文件后面定义的变量，不分配存储空间，只是说明该变量是有定义的。

7.8 静态变量

- 静态变量(static)分为两种情况：

(1)静态自动变量：延长其生存期至整个程序执行结束。(下次再进入该函数时值还在)。

(2)静态外部变量：缩小其作用域至所在的源文件。(多人合作时避免冲突，增加安全性)。

p102~104。

- 静态函数：函数在缺省情况下都是全程外部的，而静态函数类似于静态外部变量，是将函数的作用域缩小到其所在的源文件。p104。

7.9 寄存器变量

指定硬件寄存器作为变量的存储位置。

```
register int i;
i++;
```

快！对频繁使用的整型、字符型变量可用register。

注意：

(1)register只适用于auto和形参。

(2)为整型时，int可省。
(3)与机器硬件有关，不能保证该变量一定存放到寄存器中。
(4)不能进行取地址操作(&)，不能存放组合型变量。

例：auto_static()
```
{
 int auto_var=0;
 static int static_var=0;
 printf("auto=%d, static=%d\n", auto_var, static_var);
 ++auto_var;
 ++static_var;
}
main()
{
 int i;
 for(i=0;i<4;i++)
     auto_static();
}
```

7.10 函数的递归调用

函数可以直接或间接调用自身，称为递归调用。并不是每种程序设计语言都支持函数的递归调用，但PASCAL、C都支持。

例如计算阶乘n!，因为n! = n*(n-1)!，所以可以用递归函数实现对阶乘的计算。p87，88例。

注意：(1)p88。其中最重要的是之前设定递归结束条件。
(2)递归调用不是必须的，很多情况下可化为非递归。p89。
(3)递归调用可以简化程序的设计，使之易读，但执行效率并不高。

7.11 内联函数

C++新增。主要目的是提高程序的运行效率，作用类似于原C中的"宏"。p90。

- 定义方法：在一般的函数定义前加inline关键字。p91。

注意：(1)必须在第一次被调用前即声明为inline。p91。
(2)内联函数中不使用复杂的流程控制语句，如for, switch等，如果有，将被视为普通函数处理。

(3)注意它与#define所定义的"宏"的异同。p92。

7.12 重载函数

C++新增。目的是增加函数的灵活性，使程序更易读、易书写。p93求绝对值例。

当两个函数的参数个数、参数类型或返回值类型不同，但函数名相同时，称之为函数重载。

注意：(1)重载函数的匹配。p94。
(2)重载函数如果仅仅是返回值类型不同是不够的。p94。
(3)不能用typedef定义的类型名来区分重载函数中参数的类型。
(4)应该让重载函数对不同的参数完成相同的功能。

7.13 参数的默认值

当函数调用时，如果实参缺省，对应的形参就设为事先确定的默认值，便于编程。p95。

注意：(1)参数的默认值在函数声明中提供，当只有定义时，才可出现在函数定义中。

(2)默认参数的顺序规定：应从右至左逐渐定义，调用函数时只能从左向右匹配参数。p96。
(3)注意重载函数用默认参数时可能产生的二义性。p96。
(4)默认值可以是一个全局变量、全局常量、甚至是一个函数，但不能是局部变量。

第8章 指针

8.1 指针的概念

(1)定义：一个指针其实就是一个地址，相当于日常生活中的路标或指路牌，能够提供对变量空间的“间接”访问。

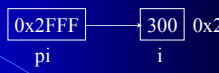
一个指针变量中存放的就是另一个数据对象的内存地址。

例：int *pi;　　定义了一个指向整型量的指针。

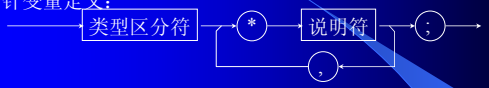
关于取地址运算符&：

作用：取变量名所指变量的地址。

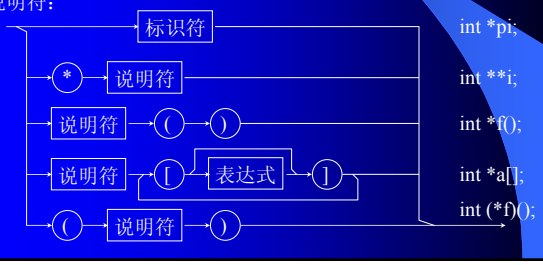
例：int i, *pi;
pi = &i;　则将pi指向变量i：



指针变量定义：



说明符：



int *pi;
int **i;
int *f();
int *a[];
int (*f)();

指针定义例子：

int *pi;	指向整型量的指针(类似的：char *pc; … …)
int *f();	函数f的返回值是一个指向整型量的指针
int (*pfi)();	pfi是指向函数的指针，该函数的返回值为int
int *api[4];	api是长度为4的指针数组，其每个元素是一个指向整型量的指针
int (*pai)[3];	pai是一个指向长度为3 的整型数组的指针
int *(*pfpi)();	pfpi是一个指向函数的指针，该函数的返回值为指向整型量的指针
int (*(*fap)[])();	fap是一个指向数组的指针，该数组是一个指针数组， 其每个元素是指向返回值为int的函数的指针

注意：要从内层逐层向外读。

(2)指针的使用

定义了一个指向特定类型的指针后，可以：

a. 赋值：否则无意义
```
int i, *pi;
pi = &i;
```

b. 引用：*pi —— 取pi所指变量的内容

所以：i=0; 等价于 *pi=0;

i++; 等价于 (*pi)++; 括号不能少，因为：

*pi++; 等价于 *pi; pi=pi+1;

c. 复写：
```
int x, *px, *py;
px = &x;
py = px;   使指向相同
```

注意：(1)指针变量可以用auto, register, static和extern说明。

(2)复写时(px=py)，两指针类型要一致。

(3)指针必须有指向。int *p; *p=2; 是错误的。

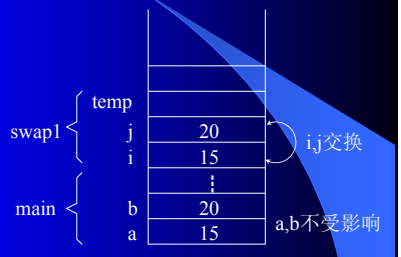
(4)静态/外部/指向基本类型的auto指针可以赋初值。

```
int k[5], i;
int *ptr1 = &k[1], *ptr2 = &i;
```

8.2 指针与函数

C的函数参数是传值的，直接传值无法将函数对形参的修改带出，但实际应用中常有这种需要，C语言中可以利用指针实现。以交换两个变量值的过程为例：

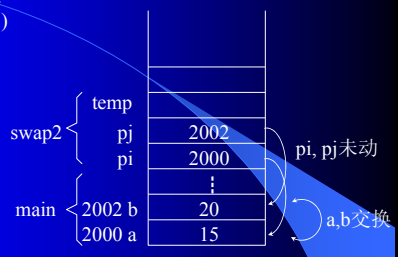
```
swap1(int i, int j)
{int temp;
 temp = i;
 i = j;
 j = temp;
}
main()
{int a=15, b=20;
 swap1(a,b);
}
```



但该程序无法实现通过调用函数swap1交换main中a和b的值的目的，因为：

用指针改写上述程序：

```
swap2(int *pi, int *pj)
{int temp;
 temp = *pi;
 *pi = *pj;
 *pj = temp;
}
main()
{int a=15, b=20;
 swap2(&a, &b);
}
```



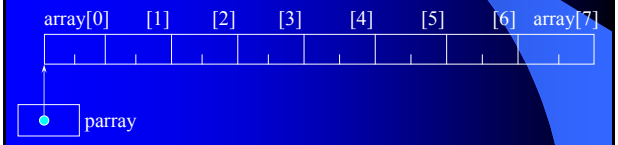
但该程序可以实现通过调用函数swap2交换main中a和b的值的目的，因为：

所以，虽然C本质上是传值的，但通过指针，令形、实参均为指针，便可使形、实参指向同一对象，从而实现在函数中改变实参所指变量的值。与PASCAL 中的传地址方式功能相当，但需要用户自己负责地址传递。这也是为什么scanf参数前要加&。

8.3 指针和数组

C语言中指针和数组是密不可分的，且同样的问题用指针处理速度更快。对数组元素的存取可以通过下标来确定元素，也可以通过指针运算来确定元素。从某种意义上说，指针就是数组，数组就是指针。

例：int array[8], *parray;
parray = array; 或
parray = &array[0]; 则可形成下图所示状态：



则：*parray 等价于 array[0] 等价于 *array 等价于 parray[0]
*(parray+i) 等价于 array[i] 等价于 *(array+i) 等价于 parray[i]
parray+i 等价于 &array[i] 等价于 &parray[i]

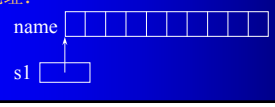
数组名与指向该数组首址的指针的区别在于：
数组名不是变量，不能array++，但parray++可以。

• 函数参数中的数组名 p162

当实参是数组名时，对应的形参有两种定义方法，以字符数组为例：

```
main( )                  slength(s1)              slength(s1)
{char name[10];          char s1[];               char *s1;
 ... ...                 {          或            {
 slength(name);            ... ...                  ... ...
 ... ...                 }                        }
}
```

形参的两种定义方法作用完全相同，因为在C中数组名就代表数组的首地址：



p162图8-7

结论：C语言中，指针和数组是密不可分的，C语言数组实际上就是指针，C用指针＋运算符[]模拟了数组的概念， C编译器只实现了指针， 并未实现数组， 所以在C中根本不会检查数组下标是否出界。

注意：如果需要对连续的多个数组元素进行处理，用指针方式访问元素比通过下标访问速度快，例如p154例中的方法2。

8.4 指针运算

指针之间有一些特定的运算，其结果总是与某一类型对象的地址有关：

(1)两指针在一定条件下可以比较。例如p1，p2指向同一数组中的元素。

(2)与整型量可以进行加减运算。p+n表示指针p所指位置之后第n 个对象的地址，即后移n个单位长度。编译程序将按类型不同将单位长度予以放大：char(1)、short(2)、int(2)、long(4)、float(4)、double(8)、数组(sizeof(数组))、结构(sizeof(结构))。

(3)两指针可进行减法，要求：指向同一数组中的元素。
p1-p2(p1>p2)的结果就是p1, p2之间元素的个数。

(4)除上述之外，其它运算非法。

8.5 指针数组及其初始化

当一数组中的元素为指针变量时，即成为指针数组。最常用的指针数组就是字符指针数组：char *lineptr[100];

典型应用：p171书名排序。

赋初值：类似前面的数组，p171,172。

```
main()
{static a[5]={1,3,5,7,9};
 int *num[5]={&a[0],&a[1],&a[2],&a[3],&a[4]};
 int **p, i;
 p = num;
 for(i=0;i<5;i++)
        {printf("%d\t", **p); p++;}
}
```

- p171图8-12：指针数组类似二维数组但有区别(列数不定)。按照C语言中指针和[]运算符的特点，在使用时感觉象二维数组。
- 应根据需要选用指针数组或多维数组：处理矩阵，用多维数组方便；处理多个字符串，用指针数组方便。

8.6 命令行参数

在操作系统中，运行一命令时往往可以指定一些选择项，以便增强命令的灵活性，如格式化命令、拷贝命令等。

用户所编程序经编译后，也可看作操作系统的一个命令，有时也需具备上述功能。

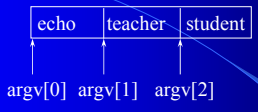
可以将一个 C程序中的main看作是操作系统所调用的函数，上述功能即可通过对main函数设置参数来实现。

以DOS/UNIX中的echo命令为例，说明如何使用main的参数：

```
>echo   teacher   student<回车>
命令名  选项1     选项2
```

C语言中main函数可以有两个参数，即argc和argv，其中argc表示命令行中参数的个数，argv是一个指针数组，指向各选项。

以上述echo命令为例，argc的值将为3(包括命令名在内)，argv的内容为：




```
main(int argc, char *argv[])
{
        while(--argc>0)
                printf("%s%c",*++argv, (argc>1)?' ':'\n');
}
```

main只能有上述这两个参数，这是固定的，由于argv是字符指针数组，所以当命令行选项是整型、浮点型等非字符串类型时，必须在程序中进行转换：

例如需要实现下述的命令行选项：

```
exec  5  6.3  student<回车>
```

```
main(int argc, char *argv[])
{
int i;
float f;
sscanf(argv[1],"%d",&i);
sscanf(argv[2],"%f",&f);
        ⋮
}
```

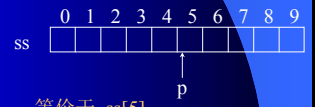
内存格式转换函数：
sscanf(char *, 格式控制串，参数1，...)
sprintf(char *, 格式控制串，参数1，...)

8.7 多级指针

上述argv，即字符指针数组就是多级指针的一种。当指针所指内容又是指针时，就称为多级指针。

8.7.1 进一步理解[ ]运算符

```
char ss[10],*p;
p = ss+5;
```



则 p[0] 等价于 *p 等价于 ss[5]
p[1] 等价于 *(p+1) 等价于 ss[6]
p[-1] 等价于 *(p-1) 等价于 ss[4]

8.7.2 进一步理解二维数组与指针

```
static int a[3][4]={{1,3,5,7},{9,11,13,15},{17,19,21,23}}, (*p)[4];
```

(1)多维数组的地址 谭p179

数组名a代表该数组的首址，由于二维数组实际上是元素又是数组的一维数组，所以 a 指向其第0个元素，逻辑含义是一个指向长度为4的整型数组的指针。同理a+1指向第一行；a+2指向第二行。

a[0]可以理解为第0行的数组名，指向 a[0][0]，所以逻辑含义是一个指向int的指针；同理a[1]、a[2]分别是第1、第2行的数组名。

所以：a+i的逻辑含义是int (*)[4]，a+i等价于&a[i]；
a[i]的逻辑含义是int *，a[i]等价于&a[i][0]；a[i]+j等价于&a[i][j]。

注意：不要将 &a[i]理解为a[i]单元的地址，因为并不存在a[i]这样一个变量，二维数组中真正存在的是a[i][j]，这只是一种计算地址的方法。&a[i] 和 a[i] 的地址值是一样的，但逻辑含义不同，&a[i]即a+i指向第i行，而a[i]即*(a+i)指向a[i][0]。

p = a;
则：p[i] 等价于 *(p+i) 等价于 a[i]
p[i][j] 等价于 *(*(p+i)+j) 等价于 *(*(a+i)+j) 等价于 a[i][j]

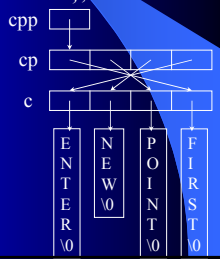
(2)二维数组作函数参数

形参的定义方法：p132或int (*grade)[4]。

8.7.3 一个综合的例子

```
char *c[] = {"ENTER","NEW","POINT","FIRST"};
char **cp[] = {c+3, c+2, c+1, c};
char ***cpp = cp;
main()
{
        printf("%s", **++cpp);
        printf("%s", *--*++cpp+3);
        printf("%s", *cpp[-2]+3);
        printf("%s\n", cpp[-1][-1]+1);
}
```

POINTERSTEW



8.7.4 结论

当一个指针指向了一片连续的存储区后：

(1)*p(一级)：指向某数据类型，相当于一维数组名。

p[i] 等价于 *(p+i)

(2)**p或(*p)[N](二级)：指向一级指针(行)，相当于二维数组名。

p[i] 等价于 *(p+i)　　一维

p[i][j] 等价于 *(p+i)[j] 等价于 *(p[i]+j) 等价于 *(*(p+i)+j)

(3)***p或(*p)[M][N](三级)：指向二级指针(面)，相当于三维数组名。

p[i] 等价于 *(p+i)　　二维

p[i][j] 等价于 *(p+i)[j] 等价于 *(*(p+i)+j)　　一维

p[i][j][k] 等价于 *(p+i)[j][k] 等价于 *(*(p+i)+j)[k] 等价于 *(*(*(p+i)+j)+k)

总之，(合法指针p)[i] 等价于 *(p+i)；而 p+i 等价于 &p[i]。

8.8 指向函数的指针

C语言中，函数也可以通过指针间接调用，指向函数的指针实际上就是相应函数的入口地址。p177。

第9章 引用

引用与指针有关，但又不相同。它是C＋＋中新增的，主要目的是为了更加方便对函数参数的传递。

9.1 什么是引用

引用实际上就是为某个变量或对象所建立的别名，对引用的修改实际上就是对相应变量或对象的修改。

```
int someInt;
int &rInt = someInt;
```

在声明时必须初始化 p185例ch9-1.cpp

注意：(1)若引用被声明为T&，则必须用T类型的变量或对象，或能够转换成T类型的对象进行初始化。

(2)若声明引用时所用的初始值不是左值，则会建立一个 T 类型的变量并用该初始值对其初始化，引用即和所建立的变量相关联。p188。

(3)允许建立对指针变量的引用。

```
int *a;  int * &p=a;  int b=8;  p=&b;  *p=10;
```

(4)不允许对void进行引用。

(5)不能建立引用的数组，也不能为数组名建立引用，因为数组名不是变量。

(6)引用只是某个变量的别名，本身不是变量，所以没有引用的引用，也没有引用的指针。

(7)有空指针，无空引用。int &ri = NULL; 非法

9.2 引用的操作

(1)取引用的地址，得到的将是其相关变量的地址。p186

(2)对引用的任何赋值，实际上都是对其相关变量的赋值。p187

(3)引用一旦初始化，就关联到一个固定的变量上，再也不分开，不能象指针那样通过复写操作指向另一个对象。

9.3 用引用传递函数参数

用引用做函数形参可以实现在函数体内改变对应实参值的目的，其效果与使用指针完全相同，但使用时更加简单(p190)。这实际上相当与PASCAL中参数的传地址方式。

注意：用引用传递函数参数时，注意可能带来的二义性。p191

9.4 用引用返回函数值

C＋＋允许函数的返回值为引用，注意p193有关引用返回的4种不同情况。要特别注意不要返回一个对局部变量的引用。

9.5 用const限定引用

使用指针或引用作为函数参数的好处主要是：

(1)使函数可以传回多个值。

(2)提高效率。

但是也有缺点：

这会使实参对应的变量处于随时会被修改的危险之中。有时只需要使用引用提高效率，但不希望实参被函数修改，这时就可以用const指针或引用对实参加以保护。 p198。

第10章 结构和联合

10.1 结构类型的定义

和数组一样，结构也是一种组合数据类型。与数组不同的是：数组是由一组相同类型的数据组成的；而结构则是有一组不同类型的数据所组成的。在实际应用中，有很多数据信息必须用结构来表示，例如人事记录、课程信息等。p204例说明其必要性。

(1)定义：struct 结构类型标识符 {

```
        成员1;        //可以是基本数据类型、
        成员2;          数组或另一个结构等
          ⋮
        } [结构变量名];
```

例：

```
struct date {
        int day, month, year;
        char mon_name[10];
        } d1, d2;
struct date d3;
```

(2)初值：类似数组赋初值的方式：

```
static struct date d1 = {12, 5, 1990, "MAY"};
```

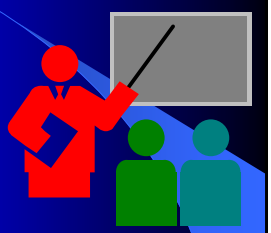
(3)对成员的访问：可以有三种方式

a. 结构变量名.

b. 结构指针名->

c. (*结构指针).

} 结构成员



例：

```
struct date d, *pd;
pd = &d;
d.day = 12;
pd->month = 5;
(*pd).mon_name = "MAY";
```

10.2 结构和函数

主要解决的是结构变量作为函数参数传递的情况。

由于C支持对结构变量的整体赋值(p206)，所以作为函数参数可以有两种用法：(1)形参为结构变量，此时形、实参整体赋值。

(2)形参为结构指针或引用。p212~213。

函数的返回值也可以是结构、结构指针、结构引用。

结构除了它是由用户定义的组合数据类型之外，其它方面与基本数据类型完全类似。

10.3 结构数组

当一个数组的每个元素都是结构时，就称为结构数组，经常用它来处理一系列的记录。如 p210 例 10_5.cpp所示的关于6个人的薪水情况的记录。

例10_6.cpp利用结构指针数组提高效率。

经验：应用中常会遇到这样的情况：

(1)该问题适合用数组解决

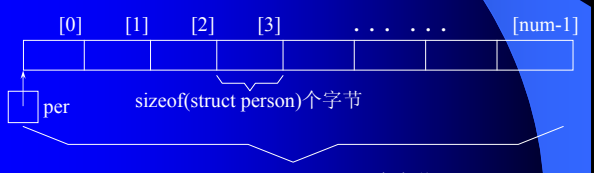
(2)数组的长度事先不好确定

这时希望能定义长度可变的数组，而C语言在定义数组时，要求长度必须是常量，所以不能用C本身的数组定义实现上述目的。

但是由于C的指针与数组间的特殊关系，所以利用指针和动态申请内存的库函数malloc( )便可实现上述目的。p155。

以例ch10_5.cpp中的结构数组为例，定义结构指针per：

```
#include<alloc.h>      //void *malloc(size_t  size)
struct  person  *per;
int  num;              //确定数组长度
    ┆
scanf("%d",&num);
per = (struct person *)malloc(sizeof(struct person)*num);
per[1].name = "John";
per[1].id = 13916;
    ┆                  //此后便可将per当成长度为num的数组来用
```



总之，在C语言中，只要一个指针指向了一片连续的存储区，即可将将它作为数组，连续的存储区既可通过数组定义静态获得，也可通过malloc或new动态获得。

10.4 指向结构的指针(结构指针)

即一个结构变量的首地址，结构指针的主要用途有三：

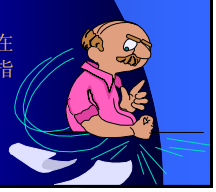
(1)10.2所述，用作函数参数。

(2)10.3所述，结合结构数组，引用其元素；或构造动态结构数组。 注意其单位长度是sizeof(struct ... ...)，C采用偶地址分配。

(3)下节所述，用于构造链表、树等数据结构。

10.5 树和表

树和表都是重要且很常用的数据结构，在数据结构课中将会详细介绍，为说明结构指针的用法，此处简单介绍链表结构。



10.5.1 什么是链表

(1)引用自身的结构：结构允许嵌套定义，即结构的一个成员是另外一种结构，但不能是该结构自身； 不过可以是指向自身的指针。p217。

(2)多个结构变量的组织：程序中经常需要同时处理多个结构相同的结构变量 (如某单位的人事信息)，这些结构变量的组织方式如下：

a.事先已知结构的个数，且程序运行过程中不变：用结构数组。

b.程序运行后才知结构个数，且不再变化：用动态结构数组 (结构指针＋malloc)。

c.自始至终也不知究竟会有多少个结构，或者在程序运行中个数会不停变化，此时结构随机散布在内存中，不是连续存放的：用链表或树等数据结构。

(3)链表结构：对于上述第三种应用情况，用引用自身的结构将多个结构串接起来，即构成链表结构。p219图10-2。

10.5.2 链表的创建与遍历

p219例ch10-11.cpp，构造并显示学生成绩信息。

注意：C语言中没有new操作符，申请新结点要自编一个函数：

```
struct student * getnode()
{
return((struct student *)malloc(sizeof(struct student)));
}
```

10.6.3 链表的插入与删除

p222例，图10-3、图10-4。

p224例，图10-5。

注意对链表的头、尾的处理，是最容易出错的地方。p225。

10.7 字段存取

节省空间的一种方法，允许将若干个状态量压缩到一个机器字中。其功能完全可以用字位逻辑运算来实现，但用字段存取更方便。例如结构date也可以用字段来表示：

```
struct date {
        char flag;      //闰年
        int day, month;
        };
```

```
struct date1 {
  unsigned  int  flag:1;
  unsigned  int  day: 5;
  unsigned  int  month:4;} d1;
```

10.8 联合类型

节省空间的另一种方法，定义类似于结构。可将其“想象”成结构，但所占空间不是各成员的总和，而是各成员中所占空间最大者，所以也称为“共用体”

```
union v_tag {
        int ival;
        float fval;
        char *pval;
        } uval, *p;
```

```
uval.ival = 5;
. . . . . .
p = &uval;
p->fval = 6.3
. . . . . .
```

10.9 类型定义

C允许用户用typedef定义数据类型，例如：

```
typedef  int  INTEGER;
INTEGER  i;
```

```
typedef struct date {
        int  day, month, year;
        char  mon_name[10];
        } DATE;
DATE d1, d2, *pd;
```

目的：方便编程，特别对一些复杂的定义，用typedef定义成一个新的类型，免得每次都写一大串：

```
typedef int (*(*(*POINTFUNC( ) )[ ] )( ) )[ ];
POINTFUNC check1, check2;
```

用途：(1)便于程序移植，将与硬件有关的数据类型用typedef重新定义。
(2)使程序清晰，易于理解。

第11章 C预处理程序

C的特色之一。通过预处理有效地扩充语言的能力，使程序易于开发、阅读、修改及移植。

11.1 #define语句

定义符号常数及“宏”。

11.1.1 定义符号常数

格式：#define 符号常数标识符 单词串标志

如：
```
#define TRUE 1
#define FALSE 0
```

注意：(1)该语句后不可用分号。
(2)不一定非在程序头上，可在任何地方出现，但名必须先定义，然后才可用。不存在所谓“局部”的问题。

好处：(1)程序的可扩展性，易读、易改：

```
#define MAXSIZE 100
char *sp[MAXSIZE];
. . . . . .
for(i=0; i<MAXSIZE; i++)
. . . . . .
```

(2)增强可移植性：

```
#define INT_SIZE 16        //machine dependent
```

11.1.2 宏定义及展开

即在#define中带变元的情况，称为“宏定义”(macro)

```
#define max(a,b) ((a)>(b)?(a):(b))
```

将来在程序中，语句 x = max(p+g, r+s); 将被替换为：

```
x = ((p+q)>(r+s)?(p+q):(r+s));
```

注意：宏定义会产生副作用，例如：

```
#define SQUARE(x) x*x
```

则y = SQUARE(v);　　y = v*v;
而y = SQUARE(p+q);　　y = p+q*p+q;　　//error
所以必须注意定义严格：

```
#define SQUARE(x) (x)*(x)
```

小结：(1)宏在使用时类似函数，但与函数不同，它不安全，但运行效率高。功能上相当于C＋＋中的inline函数。
(2)长的定义，如需分多行，行间用 \ 分隔。
(3)可重复定义，使用时以前面最近一次定义为准。
(4)可嵌套定义：

```
#define OK 1
#define GOOD OK
```

(5)可撤消定义：

```
#undef OK
```

11.2 #include语句

C可用 include 语句将另一个文件蕴含到当前源文件中。比如定义了多个符号常数和宏，且它们在多个程序程序中都要用到，这时可将它们收集起来存入一个文件，如 constant.h (一般以 .h 为后缀，取header之意)，在使用它的源文件头上：

```
#include “constant.h”
#include <stdio.h>              //注意与“ ”用法的区别，查找快。
```

优点：能将定义集中起来，供多个程序共同使用。在大程序中，易于修改，避免产生定义的不一致。

11.3 条件编译

可在程序中设置一些编译“开关”，对程序中的某些语句有选择地编译。

(1)#ifdef, #endif, #else和#ifndef语句

判断某个符号常数是否用#define定义过，以决定是否对其后的语句进行编译。

用途一：增强程序的可移植性。

例如，C语言在不同机器上int类型的长度不同，则编程时可以：

```
#ifdef  PDP11
#define  INT_SIZE  16          //PDP11上整型占2个字节
#else
#define  INT_SIZE  32          //其它机器上整型占4个字节
#endif
```

用户可根据情况选择是否要定义PDP11，为给PDP11加定义：

a. 在程序前加上：#define PDP11 1 甚至 #define PDP11即可。

b. 用编译命令的命令行参数(以UNIX为例)：

```
cc  my_prog.c  -D PDP11<回车>
```

用途二：在调试程序时，作为跟踪语句的开关：

```
#ifdef  DEBUG
printf(“***Test  information\n”);
. . . . . .
#endif
```

调试时用-D编译，成功后不用，目标代码中便可去除跟踪信息。

(2)#if语句

它通过了更一般的方法，可针对符号常数的值进行多种选择：

```
#if  MACHINE==1
. . . . . .
#else
#if  MACHINE==2
. . . . . .
#else
#if  MACHINE==3
. . . . . .
#endif
#endif
#endif
```

为符号常数赋值：

a. #define MACHINE 1(或2，3)

b. 编译命令行参数：

```
cc  my_prog.c  -D  MACHINE=2<回车>
```

第12章 操作系统上的C程序

实际上就是由各库函数构成的一个编程环境。

12.1 标准文件的输入 / 输出处理

操作系统将I /O设备也作为文件处理，任何C程序都包含三个标准文件：标准输入文件stdin，标准输出文件stdout，标准出错文件stderr。

(1)scanf等：从stdin输入数据。

(2)printf等：向stdout输出数据。

12.2 文件输入 / 输出

(1)将I / O重定向到文件

a. 输出重定向(>)：my_exec > ttt.dat<回车>

会将原输出到stdout的内容全部输出到文件ttt.dat中。

b. 输入重定向(<)：my_exec < input.dat<回车>

会将stdin重定向到文件input.dat，改从该文件中输入数据。

(2)关于文件结束

大多数库函数，在遇到文件结束时会返回标志EOF，在程序中应判断此标志，否则可能陷于死循环。例如程序：

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

假定编译后的可执行文件名为exec.exe，则

exec < infile.txt<回车>

的效果是在屏幕上将文件infile.txt的内容打印出来。

12.3 用于文件处理的特殊函数

对文件的简单处理用scanf、printf、putchar、getchar等函数加上I /O 重定向即可解决。但有时需要更大的灵活性、对文件进行复杂的操作，比如从两个文件中同时读数据，这时就需要下述的文件处理函数：

(1)fopen函数

```
FILE  *fp, *fopen();
fp = fopen("文件名", "r");  //或者"w"写、"a"添加
```

(2)getc和putc函数

```
char c;
c = getc(fp);
putc('\n', fp);
```

(3)fclose函数

```
fclose(fp);
```

文件操作全部完成后将文件关闭。

(4)feof函数

判断是否已读取到文件尾(结束返回1，否则返回0)：

```
if(feof(fp)  printf("Run out of data !\n");
```

(5)fscanf和fprintf

对文件按格式输入/ 输出：fscanf(fp, 格式控制串, 参数1, 参数2. .)

(6)fgets和fputs

```
fgets(char *buffer, n, fp);
```

直到读到'\n'或读满n个字符为止，读出的内容存入buffer，最后自动加'\0'。

```
fputs(char *buffer,  fp);
```

将buffer中的内容写入文件，直到写到'\0'为止，但'\0'不写入文件。

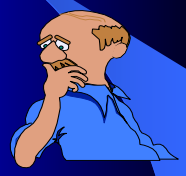
(7)fread和fwrite

int fread(char *pt, unsigned size, unsigned n, FILE *fp)

从fp所指文件中读取长度为size的 n个数据项，存到 pt 所指的内存区。返回所读数据项的个数，如遇文件结束或出错返回0。

int fwrite(char *pt, unsigned size, unsigned n, FILE *fp)

将pt所指的n*size个字节输出到fp所指的文件中。 返回所写数据项的个数。



例：struct student {

char name[10];

int age;

char addr[30];

} stud[40];

则用

fwrite(stud, sizeof(struct student), 40, fp);

即可将该结构数组写入fp所指的文件中。将来用

fread(stud, sizeof(struct student), 40, fp);

又可从文件中将该数组读出来。

第13章 类和面向对象程序设计

13.1 类的定义

类是面向对象程序设计的基础，通过类，实现数据及对其进行处理的函数 (方法)的封装，从而实现信息隐藏，屏蔽细节，提高程序的可重用性，提高程序设计的效率。

C++的类由C的结构发展而来：

(1) p232 C中表示存款帐户的结构Savings。

(2) p233 C++中表示存款帐户的类 Savings，注意除了数据成员外还有方法。说明类具有封装性，必须通过方法才能修改保护成员的值。

(3) 类与结构的区别。p233。

13.2 面向对象程序设计

13.2.1 为什么要引入类

这是程序开发方法发展的必然。80年代中开始出现的软件危机使得传统的程序设计方法面临挑战。

- 早期的结构化设计方法：

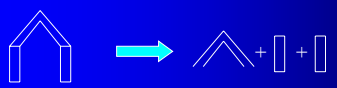
  程序 =(算法)+(数据结构)，以算法(过程)为主，见p234图11-1。

  缺点：不利于软件的重用，开发过程复杂，效率低。

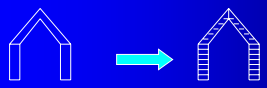
- 面向对象的设计方法：数据结构及其相关的算法被封装成一个类，对象是类的一个实例，这时：

  对象 = (算法+数据结构)　　程序 = (对象+对象+...)

  优点：大大提高了程序的可重用性，简化了程序设计，提高开发效率。对象成为程序的基本构件，编程就好象在搭积木。



结构化程序设计：
定制构件



面向对象程序设计：
利用已有标
准件搭积木

由于突破了软件设计思想上的障碍，软件产业得以飞速发展。C++是类机制实现得比较完善的一种高级语言。

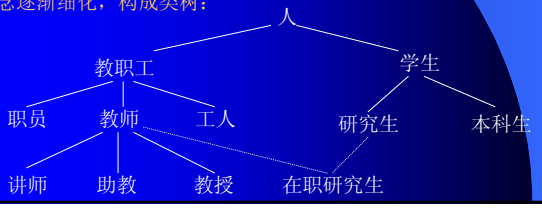
13.2.2 抽象

抽象就是具体事物的描述的一个概括，以隐藏事物固有的复杂性。将现实世界中的事物抽象成对象，以便编程。即忽略细节，只知接口。

P299，将Josephus问题抽象成两个类：Josephus类和Ring类。

13.2.3 分类

将具有某些共性的对象按一定规则归为一类，层层分类，使概念逐渐细化，构成类树：



其中子类可以继承父类的所有属性和方法。继承使得程序具有较高的可重用性，可以提高开发效率。总之：

结构化程序设计：按功能分割问题，强调的是处理过程的功能划分，每个功能都靠自己解决。如果一个功能与另一个已有功能的实现类似，也不能直接为我所用。

•面向对象程序设计：以数据为中心，按对象分割问题。对象被分成类，类又是层层分解的，子类可以继承父类的所有特性和行为，同样的对象可直接用于多个不同的程序。

•P293~310对Josephus问题的解答，注意体会结构化程序设计与面向对象程序设计的本质区别。

13.3 定义成员函数

即对象的方法。定义方法与前面的函数基本相同，只不过是在类中定义的。P236。

(1)在类定义中定义成员函数(inline)。

(2)在类定义之后定义成员函数。P238，这是典型的OO程序的写法。

(3)重载成员函数。P239。

13.4 调用成员函数

(1)调用：成员函数必须通过对象来调用，其形式类似于访问结构的一个成员，除此以外与调用普通函数完全相同。P240~241。

(2)在成员函数中访问成员(属性)：p243。

13.5 保护成员

对某些成员加以保护不允许从外部直接访问，只能通过接口访问，这也是封装的特性之一。

13.6 屏蔽类的内部实现

类实现了对数据及其行为的封装，要使用某个类，只要了解它的公共成员即可(接口)， 即使类的内部实现完全改变了，但只要接口不变，该类的使用方法就不会变。P247~251。

13.7 再论程序结构
p252

13.8 关于构造函数和析构函数

构造函数和析构函数是类的特殊成员函数。构造函数创建对象，并初始化其成员；析构函数负责撤销对象时的善后工作。

- 构造函数在创建对象时被自动调用，如果没有专门定义构造函数则调用缺省的构造函数。见p263例。
- 析构函数在对象被撤销时被自动调用，它没有返回值，没有参数，不能重载。见p269例。

13.9 关于堆

(1)关于堆区。P312。

(2)malloc / free和new / delete的区别

在不使用类时，两者在功能上是等价的(见p158)：

int *p = new int;　　等价于

int *p = (int *)malloc(sizeof(int));

但在创建对象时，new和malloc是有区别的。见p313。

第14章 I/O 流

14.1 printf和scanf的缺陷。P415。

14.2 I / O标准流类。P416。

14.3 文件流类。P418。

14.4 串流类。P420。

14.5 控制符。
用于控制输出的格式。P421~424。