



Mo	Tu	We	Th	Fr	Sa	Su
----	----	----	----	----	----	----

Memo No. _____

Date

/ /

C / C++ 语言程序设计

任课教师：邓凡

授课单位：计算机学院

授课班级：车辆1401-02



第 3 章 程序设计基础

授课学时： 学时

一、单元教学目标：

1. 理解 C 语言程序的三种基本结构
2. 掌握格式输入和输出函数
3. 掌握字符数据的输入和输出的常用函数
4. 熟练掌握顺序结构程序设计方法
5. 熟练掌握 if、if-else 语句
6. 熟练掌握 if 语句的嵌套
7. 熟练掌握 switch 语句
8. 熟练掌握选择结构程序设计方法
9. 熟练掌握 for、while 和 do-while 语句
10. 熟练掌握 break 和 continue 语句
11. 熟练掌握循环语句的嵌套
12. 熟练掌握循环结构程序设计方法

★ 重点：

- ① scanf() 和 printf() 函数
- ② 程序的三个基本结构
- ③ if 语句的两种形式
- ④ break 和 continue 语句
- ⑤ for、while 和 do-while 三种语句。

★ 难点：

- ① 选择结构的嵌套
- ② 循环结构的嵌套



Mo Tu We Th Fr Sa Su

Memo No. 26

Date / /

二、单元教学内容：

本单元主要需要学生掌握数据输入和输出函数的调用规则及格式控制字符的正确使用、掌握三种程序设计基本结构，理解结构化程序设计的方法和步骤。

1. 介绍程序结构和语句。

① 顺序结构 ② 选择结构 ③ 分支结构

2. 掌握数据的输入与输出。

{ printf ()
scanf ()

字符输入/输出函数 { getchar ()
putchar ()

3. 顺序结构的程序设计

4. 选择结构的程序设计

if - else 和 switch 语句

5. 循环结构的程序设计

for、while 和 do-while 语句

break 和 continue 语句。

★ 6. 选择结构的嵌套，循环结构的嵌套

7. 了解程序设计风格。

三、单元教学重点和难点。

教学重点：

1. 数据的输入/输出函数：scanf () 和 printf ()
getchar () 和 putchar ()



Mo Tu We Th Fr Sa Su

Memo No. 27
Date / /

2. 程序设计的三种结构

其中选择结构和循环结构是学生掌握的重点。

教学难点：

1. 选择结构的嵌套和循环结构的嵌套。

解决方法：多举实例，一步步分析每一行代码，帮助学生将以往的数学思维转换为程序语言的思维，并让学生多上机实践。

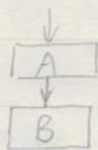
四. 教学过程

学习一门语言，学会了词汇，就要开始学习如何写语句了。

引入：程序结构和语句。

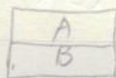
三种程序结构

顺序结构

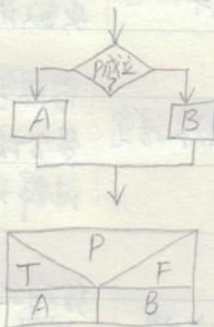


流程图

N-S结构图



选择结构





Mo Tu We Th Fr Sa Su

Memo No. 28

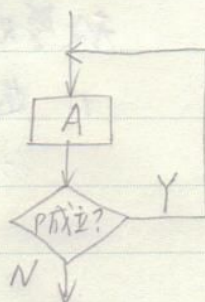
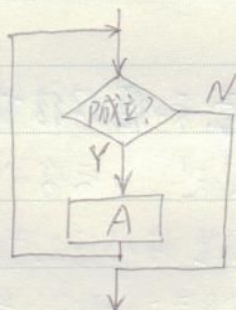
Date / /

循环结构

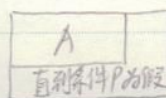
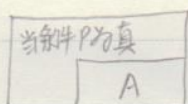
当型循环

直到型循环

流程图



N-S 结构图

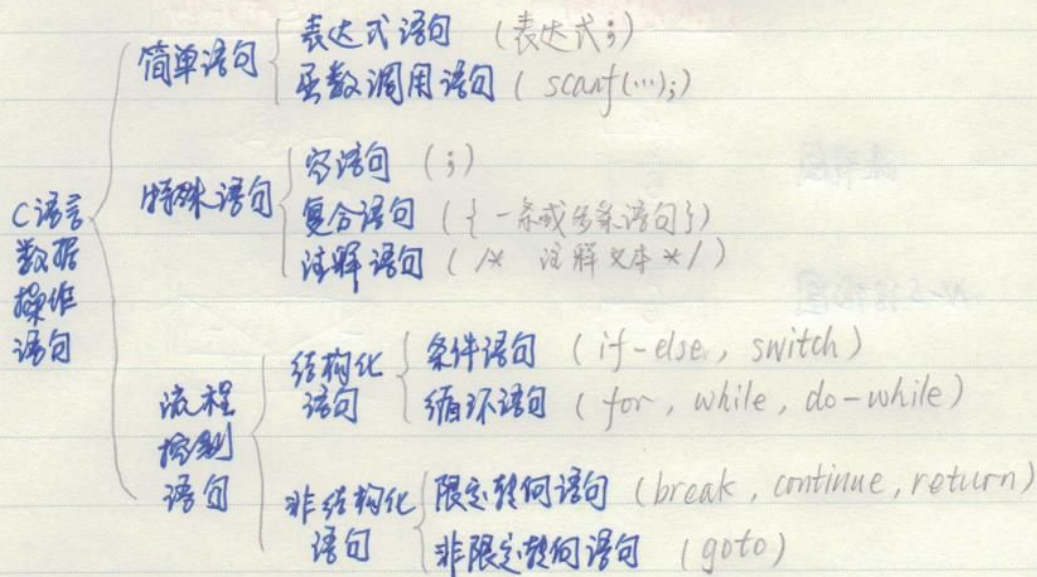


C 语句概述:

A. 程序包括数据描述和数据操作.

数据类型
和初值

为提供的数据进行加工





Mo Tu We Th Fr Sa Su

Memo No.

29

Date

/ /

b. 程序设计步骤:

① 分析问题.

② 确定算法.

③ 编写程序.

④ 调试运行.

引入: 数据的输入与输出.

a. 数据的输入输出是程序设计中用最普遍的基本操作.

b. 程序运行所需的数据通常要从外部设备(如键盘、文件等)输入, 程序运行的结果通常也要输出到外部设备(如显示器、打印机等).

c. 输入输出是用户与程序之间交互的主要手段.

d. 输入输出的标准库函数:

`scanf()`, `printf()`, `getchar()`, `putchar()`

上述函数的原型放在头文件 `stdio.h` 中.

`#include <stdio.h>`

printf() 函数

1. 标准输出函数, 输出变量、常量和表达式的值.

`printf(格式控制字符串, 输出项表);`

① 普通字符, 输出提示信息:

`printf("My name is ABC!\n");`

② 格式说明, 用于指定输出格式:

`% [格式修饰] 格式字符`

标志、类型修饰, 输出最小宽度和精度

`printf("x=%4d\n", x);`



Mo Tu We Th Fr Sa Su

Memo No. 30

Date / /

2. 格式字符: 见B8 表3.1

3. 格式修饰符:

① l: 输出长整型

② m: 指定数据输出的宽度

→ 如果是小数, 宽度包括小数点

%5d → □□□24

③ .n: 对于实型数据, 指定输出n位小数;

对于字符串, 指定左端截取n个字符输出.

④ +: 右对齐, 输出符号位

⑤ -: 左对齐.

scanf()函数

1. 标准输入函数, 是为数据的动态赋值, 即在程序运行过程中接受输入数值.

scanf(格式控制字符串, 输入项表);

同printf()函数

eg1: scanf("%d%d", &a, &b);

以空白符(空格, 跳格键或回车键)分隔.

eg2: scanf("%d,%d", &a, &b);

以逗号分隔

eg3: scanf("a=%d,b=%d", &a, &b);

a=4, b=8 (原样输入).

2. &勿忘! (数组名和指针变量除外).

3. double类型数据, 用%f或%le

4. 实型数据输入时不能规定宽度(即精度).

5. 尽量不要出现'\n', '\t'等字符.



Mo Tu We Th Fr Sa Su

Memo No. 31

Date / /

putchar() 函数

1. 输出一个字符.

2. 使用时 `#include <stdio.h>`

3. `putchar(字符);`

同样适用

getchar() 函数

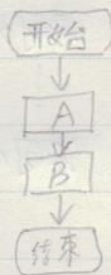
1. 输入一个字符

2. `c = getchar();`

3. 交互输入

4. 两个 `getchar()` 连续使用时, 必须连续输入两个字符.

引入: 顺序结构程序设计.



a. 最基本、最简单的结构, 又称线性结构.

b. 语句顺序被执行.

eg. 交换 a 和 b 的值 (采用临时变量)

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

eg. 见 P65 例 3.5 和例 3.6

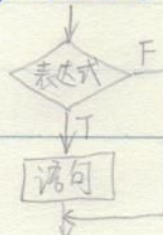
引入: 选择结构的程序设计

选择结构 { if 语句 { if 的一般形式
if 的嵌套
switch 语句 { switch 的一般形式
break 语句

if 选择结构

1. if 三种基本形式: 单分支、双分支和多分支.

2. 单分支: `if (表达式) 语句;`



eg: P66, 例 3.7
3.8

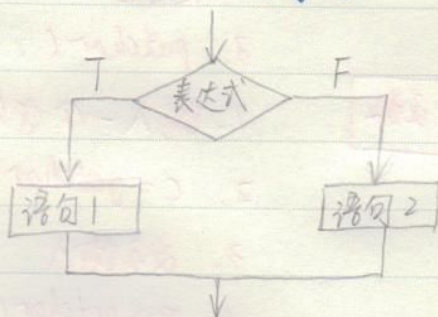


Mo	Tu	We	Th	Fr	Sa	Su
----	----	----	----	----	----	----

Memo No. 32

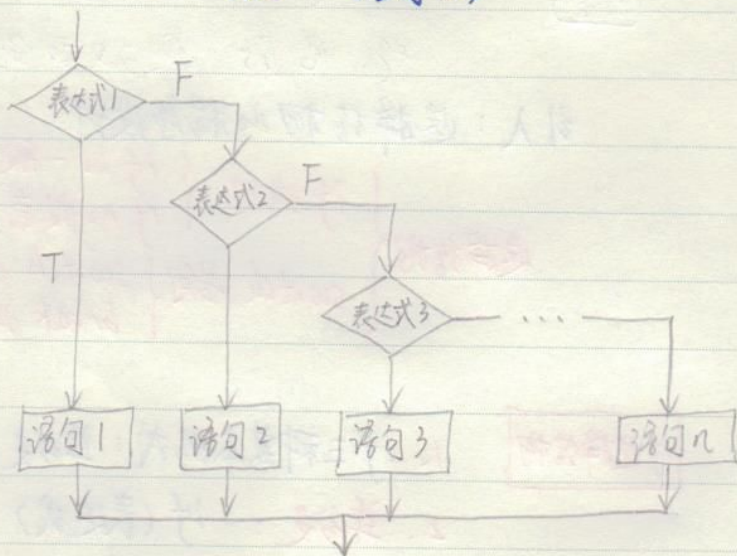
Date / /

3. 双分支: if (表达式) 语句1;
else 语句2;



eg: P67 例 3.9

4. 多分支: if (表达式1) 语句1;
else if (表达式2) 语句2;
else if (表达式3) 语句3;
⋮
else if (表达式m) 语句m;
else 语句n;



eg: P68 例 3.10



Mo Tu We Th Fr Sa Su

Memo No. 33

Date / /

Switch 选择结构

1. 多分支选择语句，又称开关语句。

2. switch (表达式)

{ case 常量表达式 1 : 语句 1;

case 常量表达式 2 : 语句 2;

⋮

case 常量表达式 n : 语句 n;

default : 语句 n+1;

}

eg: P. 例 3.11

3. break 语句:

{ 用在 switch 语句中: 跳出 switch 语句;

{ 用在循环语句中: 跳出本层循环。

4. switch 与 break 语句结合 控制流程程序的分支。

5. 注①: switch (表达式) 括号勿忘!

②: case 后 5: 冒号勿忘

③: case 后的常量表达式必须互不相同

④: case 后的常量表达式必须与 switch 后的表达式类型一致

⑤ switch 语句的花括号勿忘!

但每个 case 后若包含多条语句, 则不同花括号括起来, 系统自动执行完。

⑥ switch 语句可以用条件语句实现, 但反之不然。 (∵ switch 的表达式只能是整型或实符型, 条件语句中表达式可以是任意类型)。



Mo Tu We Th Fr Sa Su

Memo No. 34
Date / /

⑦ 多个 case 可共用一组执行语句。

eg: P12 例 3.12

⑧ 当使用了 break 语句后, 各个 case 和 default 语句的先后顺序可以变动, 不会影响程序的执行结果。

选择结构的嵌套

1. if 语句的嵌套

重点学习 if - else 的配对问题

eg: P13 例 3.13 - 3.15

2. switch 与 if 的嵌套

eg: P14 例 3.16

3. switch 语句的嵌套

eg: P15 例 3.17

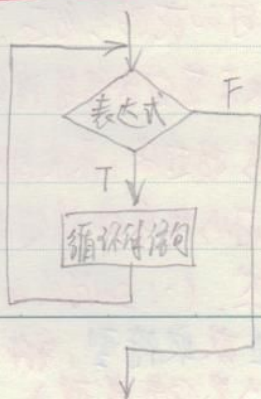
4. 综合案例

eg: P16 例 3.18 - 3.19

引入: 循环结构的程序设计

while 语句

1. while (表达式) 语句;



首先计算表达式的值, 若非 0, 则执行循环体语句, 如此反复, 直到表达式的值为假 (0)。

eg: P19 例 3.20



2. 注①: 先判断表达式, 后执行语句;

循环体可能执行多次, 也可能一次也不执行。

②: 只要表达式的值为真(非0)即可继续循环。

例如: $x=10; \text{while}(x!=0) \ x--; \quad (x=0)$

等价于: $x=10; \text{while}(x) \ x--; \quad (x=0)$

又 $x=10; \text{while}(x--); \quad (x=-1)$

③ 循环之前, 循环变量应用值。

④ 循环体如果包括多条语句, 应用大括号括起来!

⑤ 注意 while 与 for 语句的区别。

⑥ 适用于事先不知道循环次数的情形。

⑦ 注意循环次数以及循环变量的终止值, 该值可能在后面的语句中用到。

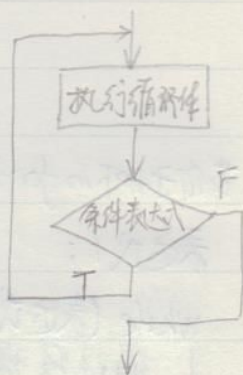
do-while 语句

1.

do

语句;

while (表达式);



首先执行循环体一次, 再判断表达式的值, 若为真(非0), 则继续执行循环体的语句, 再判断表达式的值, 如此重复, 直到表达式的值为假(0)时, 结束循环。

eg: Pg 例 3.21



Mo Tu We Th Fr Sa Su

Memo No. 36
Date / /

2. 注①: do-while 语句后面勿忘分号 (;).

②: do 和 while 之间由多个语句组成, 则必须用大括号括起来.

③: 循环体中须有改变循环变量的语句, 以便结束循环.

④: 循环体至少执行一次.

for 语句

1. for (表达式1; 表达式2; 表达式3)
语句;

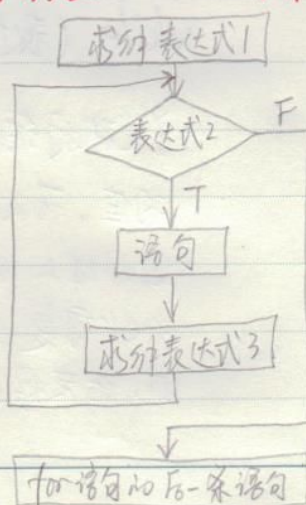
可部分或全部省略, 但分号不能省略.

2. 表达式1: 给循环变量赋初值

表达式2: 循环条件 (关系或逻辑表达式).

表达式3: 修改循环变量的值.

3. 首先计算表达式1的值, 再判断表达式2, 如果其值为非0, 则执行循环体语句, 并计算表达式3, 之后再去判断表达式2, 直到其值为0时结束循环.



等价于如下的 for 语句:

表达式1;

```
while (表达式2)
{ 循环体语句;
  表达式3;
}
```

qq: 183 1103.22



Mo	Tu	We	Th	Fr	Sa	Su
----	----	----	----	----	----	----

Memo No. 37
Date / /

4. 省略for语句的表达式:

- ① $\text{for}(i;)\Leftrightarrow \text{while}(1)$ 无限循环(死循环)
- ② $\text{for}(; \text{表达式}2;)\Leftrightarrow \text{while}(\text{表达式}2)$
- ③ $\text{for}(\text{表达式}1; ; \text{表达式}3)\Leftrightarrow$
 $\text{表达式}1; \text{while}(1)\{ \dots \text{表达式}3; \}$

5. 几种循环的比较:

- ① while 和 do-while 的表达式只有1个, 而 for 的表达式有3个;
- ② while 和 for 先判断条件再执行循环体;
 do-while 先执行循环体再判断循环条件.
- ③ $\left\{ \begin{array}{l} \text{while: 多用于循环次数不定的情况;} \\ \text{do-while: 多用于至少要运行一次的情况;} \\ \text{for: 多用于要赋初值或循环次数固定的情况.} \end{array} \right.$

Continue 语句
break 语句

1. 跳转语句, 在循环语句的循环体中使用, 进行循环的流程控制.

2. **Continue 语句:** 中断循环体的本次执行, 即执行下一次循环.

break 语句: 终止本层循环, 转到后续语句执行.

```
while(表达式1)
{
    语句1;
    if(表达式2) break;
    语句2;
}
```

```
while(表达式1)
{
    语句1;
    if(表达式2) continue;
    语句2;
}
```



Mo Tu We Th Fr Sa Su

Memo No. 38
Date 1/1

```
for(表达式1; 表达式2; 表达式3)
{
    语句1;
    if(表达式4) break;
    语句2;
}
```

```
for(表达式1; 表达式2; 表达式3)
{
    语句1;
    if(表达式4) continue;
    语句2;
}
```

引入：循环的嵌套

1. 二重循环、三重循环

2. 注①：不同层次一般要采用不同的循环变量。

外层— i ，内层— j

②：外循环每执行一遍，内循环从初值到终值完整执行一周。

③：内层循环一定要采用缩进的书写格式。

eg: p87 例 3.16-3.21

引入：循环结构程序举例

eg: p90 例 3.28-3.31

引入：程序设计风格

① 合理加入空行

② 适当使用注释

③ 标识符的命名要见名知义。eg: 指针: 加前缀 p

④ 同类变量的定义，每一条语句各占一行，便于识别和加入注释。

⑤ 变量赋初值采用就近原则，最好在定义变量的同时赋以初值。

⑥ 判等时，数值写左边，变量写右边。



Mo Tu We Th Fr Sa Su

Memo No.

39

Date

/ /

⑦. if, else, switch, for, while, do 等关键字加上其后的条件、括号独占一行，并“{”或“}”独占一行或合占一行，以保持括号配对。

⑧ 缩进对齐，{ } 垂直左对齐

⑨ 语句不宜太长，若太长须断行，行尾加“\”。

总结：

1. 三种程序设计基本结构：顺序、选择、循环；
2. if：实现单路、双路和多路分支；
3. switch：比if更简便地实现多路分支；
4. for：用于循环次数确定的计数循环结构；
5. while 和 do-while：用于循环次数不确定，由执行过程中条件变化控制循环次数的循环结构；
6. break：跳出 switch 结构或跳出循环；
7. continue：只能用于循环结构，使控制立即转去执行下一次循环。
8. goto：无条件转向指定语句继续执行。





Mo Tu We Th Fr Sa Su

Memo No. 41

Date / /

第 4 章 数组

授课学时: 学时

一、单元教学目标:

1. 熟练掌握一维数组的定义、初始化和引用.
2. 掌握二维数组的定义、初始化和引用.
3. 熟练掌握字符数组的定义、初始化和引用
4. 熟练掌握字符串概念及其输入输出
5. * 了解字符串处理函数
6. 熟练掌握指针与一维和二维数组的关联
7. 熟练掌握指针的运算.
8. 熟练掌握指针与字符串
9. * 了解指针数组
10. * 理解多级指针

★ 重点:

1. 数组的定义和引用;
2. 指针与一维和二维数组的关联.

★ 难点:

1. 不同的排序方法;
2. 字符串与一般字符数组的特征和使用方法之间的区别;
3. 二维数组与指针的关联.

二、单元教学内容:



Mo	Tu	We	Th	Fr	Sa	Su
----	----	----	----	----	----	----

Memo No. 42

Date / /

本单元需要学生掌握一维、二维数组的定义、初始化和引用，熟悉数组元素在内存中的存储方式，掌握通过指针访问一维、二维数组元素的方法，掌握字符数组、字符串与指针、指针数组的相关概念和应用。

1. 一维数组

定义、引用、初始化、排序

2. 二维数组

定义、引用、初始化

3. 字符数组与字符串

字符数组与字符串的关联，字符串处理函数

4. 指针与数组

指针的运算、指向一维、二维数组的指针、指针与字符串

5. 指向指针的指针

二级指针、多级指针

三、单元教学重点和难点：

教学重点：

1. 一维数组的定义、引用和初始化

2. 排序 冒泡排序

3. 字符数组与字符串的关联，会使用字符串处理函数。

4. 指针与数组的关联。指向一维、二维数组的指针



Mo Tu We Th Fr Sa Su

Memo No. 43

Date / /

教学难点:

1. 冒泡排序的掌握:

解决方法: 采用具体实例讲解, 演示冒泡排序每一轮的过程和结果.

2. 字符数组与字符串的关联, 以及指针与数组的关联:

解决方法: 首先须扎实掌握数组的基本知识, 其次采用画图的形式直观地给学生展示出它们的关联和区别, 并多举实例帮助学生理解和掌握.

四、教学过程:

大量具有相同性质的数据如何处理? eg: P103 例4.1

引入: 一维数组

a. 定义: 数据类型 数组名 [整型常量表达式]

b. 引用: 数组名 [下标表达式] $0 \sim n-1$

c. 初始化: 数据类型 数组名 [整型常量表达式]
 $= \{ \text{常量表达式, 常量表达式, } \dots \}$

注①: 数组是个多值变量, 由一组同名但不同下标的元素构成.

②: 一个数组被顺序存放在一块连续的内存中, 用数组名和下标就可以唯一地确定某个数组元素.

③ `int x[7]; x    int x(5); x`
`int n=4; int x[n]; x`



Mo Tu We Th Fr Sa Su

Memo No. 44

Date / /

④: 注意数组下标越界.

⑤: 须使用循环语句逐个输出各下标变量,
不能用一个语句输出整个数组.

⑥ `int x[5] = {1, 2, 3, 4, 5};` ✓

`int x[] = {1, 2, 3, 4, 5};` ✓

`int x[5] = {1, 2, 3};` (`x[3]=0, x[4]=0`)

`int x[5] = {1, 2, 3, 4, 5, 6};` X

一维数组的应用举例:

eg: P106 例 4.2-4.3

★ 冒泡排序:

eg: P107 例 4.4

引入: 二维数组

a. 定义: 数据类型 数组名 [整型常量表达式1] [整型常量表达式2]

b. 引用: 数组名 [下标表达式1] [下标表达式2].

c. 初始化:

① `int x[3][3] = { {1, 2, 3}, {2, 3, 4}, {3, 4, 5} };`

② `int x[3][3] = { 1, 2, 3, 2, 3, 4, 3, 4, 5 };`

③ `int x[3][3] = { {1, 2}, {2, 4}, {4, 5} };`

部分赋值, 其余元素为0.

④ `int x[][3] = { 1, 2, 3, 2, 3, 4, 3, 4, 5 };`

只能省略第1维大小 (省略, 可以计算得到).

⑤ `int x[][3] = { {1, 2}, {2, 4}, {4, 5} };` 补0!

$\begin{pmatrix} 1 & 2 & 0 \\ 2 & 4 & 0 \\ 4 & 5 & 0 \end{pmatrix}$



Mo Tu We Th Fr Sa Su

Memo No. 45
Date / /

二维数组的应用举例:

eg: P111 例4.6 矩阵与转置矩阵.

引入: 字符数组与字符串.

a. 定义: ^{一维:} char 数组名 [整型常量表达式]

^{二维:} char 数组名 [整型常量表达式1] [整型常量表达式2]

char a[10];

char b[3][4];

b. 初始化:

① char c[10] = {'p', 'r', 'o', 'g', 'r', 'a', 'm'};

其中 c[7], c[8], c[9] 均为 '\0'

② char c[] = {'p', 'r', 'o', 'g', 'r', 'a', 'm'};

数组长度为 7.

c. 引用:

① char c[] = {'p', 'r', 'o', 'g', 'r', 'a', 'm'};

c[5] = 'p'; ✓

② char c[5];

c = {'A', 'B', 'C'}; X

c[0] = 'A'; c[1] = 'B'; c[2] = 'C'; ✓

d. 字符串: 在计算机应用中, 需要处理大量的文字信息, 在计算机中这些文字信息通常用字符串表示.

① 字符串常量: 用双引号括起来的一串字符



就是字符串常量，其末尾由系统自动加一个字符串结束符 `'\0'` (ASCII码值为0)

eg. "program": 共有7个字符，占8个字节的存储单元

eg. "": 空串，长度为0，占1个字节的存储单元

△ 通常用一个字符数组存放一个字符串常量。

② 字符数组与字符串的区别。

字符数组: 其中每个元素都可以存放任意的字符，并不要求最后一个字符必须是 `'\0'`。
字符串: 必须以 `'\0'` 结束。

★ 一个字符型一维数组中的内容能否作为字符串使用，关键要看数组有效字符后面是否加入了串结束符 `'\0'`。

③ 通过赋初值为字符数组赋字符串。

(1) 所赋初值个数少于元素个数时

`char c[10] = {'p', 'r', 'o', 'g', 'r', 'a', 'm'};`

系统自动加 `'\0'`

也可 `char c[10] = {'p', 'r', 'o', 'g', 'r', 'a', 'm', '\0'};`

但 `char c[7] = {'p', 'r', 'o', 'g', 'r', 'a', 'm'}; X`

(2) 若采用单个字符赋初值来决定数组大小的定义

形式，一定要人为加入 `'\0'`。

`char c[] = {'p', 'r', 'o', 'g', 'r', 'a', 'm', '\0'};`

但 `char c[] = {'p', 'r', 'o', 'g', 'r', 'a', 'm'}; X`

系统为数组c开辟7个存储单元，没有存放 `'\0'`。



(3) 可以在定义时直接赋字符串常量, 不用人为加 '\0', 但必须有存放 '\0' 的空间。

```
char str1[15] = {"I am happy"}; ✓
```

大括号可省

也可以:

```
char str2[] = "I am happy";
```

系统自动添加 '\0', str2 的长度为 11。

但

```
char str3[10] = "I am happy";
```

 X

④ 字符串的输入和输出。

(1) 采用 %c, 每次输入或输出一个字符。

(2) 采用 %s, 每次输入或输出一个字符串。

eg.

```
char str1[80];  
scanf("%s", str);
```

 不用 &

输入为 How are you, 则 str 中存放 How

```
get(str);
```

输入为 How are you, 则 str 中存放
How are you.

(3) %s — printf 中须用字符数组名, 而不是数组元素,

```
printf("%s", str);
```

 ✓

(4) %s 遇到 '\0' 时结束输出

eg.

```
char c[10] = {"china"};  
printf("%s", c)
```

 输出前 5 个字符。

(5) 字符串包含多个 '\0' 时, 遇到第一个 '\0' 时结束输出。



Mo	Tu	We	Th	Fr	Sa	Su
----	----	----	----	----	----	----

Memo No. 48

Date / /

eg. `char c[10] = {"Chi\0na"};`
`printf("%s", c);` 输出为 Chi

(6) `puts(str);`

可以把字符数组 `str` 中 '\0' 前的内容全部输出,
并在输出完字符串后系统自动换行。

e. 字符串处理函数 `#include <string.h>`

(1) `strcpy(str1, str2);` 拷贝

① `str2` 即可以是数组名及数组元素地址,
也可以是指向字符串的指针, 还可以是
字符串常量

② `str1` 的长度应大于 `str2` 的长度,
`str1` 的 '\0' 也一同复制过去。

③ 两个字符数组之间不能直接赋值
`char str1[10]; char str2[10] = "abc";`
`str1 = str2;` X
只能用 `strcpy(str1, str2);`

(2) `strcat(str1, str2);` 连接

① `str1` 的长度应大于 `str1` 和 `str2` 的长度之和;

② `str1` 和 `str2` 均应该没有 '\0'。

(3) `strcmp(str1, str2);` 比较

对两个字符串自左向右逐个比较 `str1` 和
`str2` 的对应字符的 ASCII 码值的大小,
直到出现不同的字符或遇到 '\0' 为止。



- $$\begin{cases} ① = 0, & \text{str1} == \text{str2} \\ ② > 0, & \text{str1} > \text{str2} \\ ③ < 0, & \text{str1} < \text{str2} \end{cases}$$

注意：不能直接用关系运算符比较两个字符串的大小！

另转用 `strcmp()` 函数！

(4) `strlen(str)`; 求字符串长度

eg: `char str[20] = "good";`

`printf("%d\n", strlen);` 输出为4, 不包括'\0'

(5) `strlwr(str)`; > 大小写字母转换

`strupr(str)`;

eg: `printf("%s", strlwr("AbCd"));`

输出结果为abcd

f. 字符串数组:

- (1) 可用二维字符数组的形式建立字符串数组，行下标决定字符串的个数，列下标决定串的最大长度。

eg. `char str-array[30][80];`

字符串数组 `str-array`，可以存放30个字符串，串的最大长度为80。

eg. `gets(str-array[2]);`

另需标明行下标就可以访问字符串。

等价于 $\Leftrightarrow$

`gets(str-array[2][0]);`



Mo	Tu	We	Th	Fr	Sa	Su
----	----	----	----	----	----	----

Memo No. 50
Date / /

引入：指针与数组

- 数组与指针联系紧密；
- 数组名表示数组在内存中的首地址，其类型为数组元素类型的指针；
- 可以用指针变量指向数组或数组元素，也就是把数组的首地址或某一数组元素的地址放到一个指针变量中；
- 任何能用数组下标完成的操作都可由指针来实现，通过指针就可以对数组及数组元素进行操作。

A. 指针运算：

1. 算术运算：

(1) 指针与整数的加减运算：

- $[\text{指针变量} P \pm n]$ ：指将指针 P 当前所指向的位置向前或向后移动 n 个数据的位置。
- 对于 `char` 型， $P+1$ 相当于当前地址加1个字节
- `int` 型， $P+1$ “ 加4个字节
- `float` 型， $P+1$ “ 加4个字节

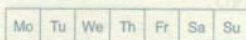
$P \pm n$ 相当于地址： $P \pm n * \text{sizeof}(\text{数据类型})$

(2) 自增、自减运算：（也是地址运算）

- $P++$ ：指针指向下一个数据的起始位置；
- $P--$ ：指针指向上一个数据的起始位置。

$*P++$ （ $*$ 和 $++$ 优先级相同，右结合）

“ $*(P++)$



$P1 - P2$: $P1$ 和 $P2$ 之间间隔元素的数目 n .

alo]

AT 371

4567

1

$P_2 - P_1$ 的值为 3

• $<$ 或 $>$: 比较两个指针指向的地址的大小关系

- $==$ 或 $!=$: 判断两个指针是否指向同一地址.

$$0 \quad 1 \quad 2 \quad \dots \quad i \quad \dots \quad j \quad \dots$$
 $2\tau_i$ $2\tau_i$

0717

0717

人

77

1

7

(1) $P1 < P2$ (或 $P1 > P2$) : $P1$ 所指元素位于 $P2$ 所指元素之前 (或之后)

(2) $P1 == P2$ (或 $P1 != P2$): $P1$ 和 $P2$ 指向同一数组元素的地址 (或 $P1$ 和 $P2$ 不指向同一数组元素的地址).

(3) 指针不能与一般数值进行关系运算, 但指针可以和零 (NULL 字符) 之间进行等于或不等于的关系运算. 例如:

$p == 0$; $p != 0$; $p == \text{NULL}$; $p != \text{NULL}$;

判断指针 P 是否为 NULL 指针。



Mo Tu We Th Fr Sa Su

Memo No. 52

Date / /

b. 指向一维数组的指针.

1. 一维数组的地址.

- 数组名是个不占内存的地址常量, 代表整个数组的存储首地址.

- $a[i]$ 的地址 $\&a[i]$ 或 $a+i$

下标形式

指针形式

$*(a+i)$ 取 $(a+i)$ 地址中的内容, 即 $a[i]$.

2. 一维数组指针的定义.

eg. `int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};`

`int *p;`

这时指针变量 p 并没有指向任何对象, 只有当数组 a 的起始地址赋值给变量 p 时, 指针 p 才表示指向数组 a , 称 **指针 p 为指向数组 a 的指针.**

两种实现方法:

(1) 用数组名作首地址.

`p = a;` // 赋值

又如: `int b[30];`

`int *p = b;` // 初始化

(2) 用数组第 0 个元素的地址做首地址.

`p = &a[0];`

注意数据类型一致:

`float a[30];`

`int *p = &a[0];` X 数据类型不一致



3. 利用指针引用一维数组元素:

若有 $\text{int } a[30]; \text{ int } *p = a;$ 则

- (1) $p+i$ 和 $a+i$ 就是数组元素 $a[i]$ 的地址, 它们指向数组 a 的第 $i+1$ 个元素 (即 i 号元素).
- (2) 指向数组元素的指针变量也可以带下标. $p[i]$ 与 $*(p+i)$ 等价

$a[i]$ 、 $*(a+i)$ 、 $p[i]$ 、 $*(p+i)$ 完全等价.

- (3) $*(p+i)$ 或 $*(a+i)$ 就是 $p+i$ 或 $a+i$ 所指向的数组元素 $a[i]$.

- (4) 指针变量可以通过对本身值的改变, 来实现对不同地址的操作, (如 $p++$) 但数组名则不行. $a++ \times \quad a = a+2 \times$

指针常量, 其值不能改变.

所以, 引用一个数组元素有两种方法:

(1) 下标法: $a[i]$, $p[i]$

(2) 指针法: $*(a+i)$, $*(p+i)$

注意

(1) 指针与一维数组的关系.

- $\&a[i]$, $a+i$, $p+i$: 引用 $a[i]$ 的地址
- $a[i]$, $*(a+i)$, $*(p+i)$, $p[i]$: 引用 $a[i]$
- $p++$, $p = p+1$: 表示 p 指向下一个元素, 即 $a[i]$
- $*p++$, $*(p++)$: 先取 p 所指向的存储单元内容 $*p$, 再使 p 指向下一个存储单元.



Mo Tu We Th Fr Sa Su

Memo No. 54

Date / /

• $*++P, *(++P)$: 先使指针 P 指向下一个存储单元, 然后取改变后的指针 P 所指向的存储单元内容 $*P$.

• $(*P)++$: 取指针 P 所指的存储单元作为该表达式的结果值, 然后使 P 所指对象的值加 1, 即 $*P = *P + 1$; 指针 P 的内容不变.

(2) 使用指针引用数组元素时下标不能越界.

(编译系统不对数组元素引用下标越界进行判断!)

< eg 4.11 P24 重点讲解! >

C. 指向二维数组的指针.

1. 二维数组的地址.

• a 和 $\&a[0]$ 都表示数组的首地址, 也是数组第 0 行的首地址, $\therefore a = \&a[0]$

• $a+i$ 和 $\&a[i]$ 表示第 i 行的首地址,

$\therefore a+i = \&a[i]$ $\therefore a+i = \&a[i]$

• $a[i]$: 第 i 行一维数组中第 0 列元素的地址.

• $a[i]+j$: 第 i 行第 j 列的地址.

• 第 i 行第 j 列的地址:

$\&a[i][j] = a[i] + j = *(a+i) + j$



• 若 n 表示二维数组的第二维的长度, 有

$$\begin{aligned} a[i][j] &= *(a[i] + j) = *(*(a + i) + j) \\ &= *(a[0] + i * n + j) \end{aligned}$$

• 特别注意 $a+i$ 和 $*(a+i)$ 的区别:

<1> $a+i$: 二维数组第 i 行的行地址, 存储的是第 i 行的起始地址, 该指针每增加 1, 则指针跳过 1 行数组元素的存储空间;

<2> $*(a+i)$: 等价于 $a[i]$, 相当于指向二维数组第 i 行中的第 0 列数组元素的列地址, 该指针每增加 1, 指针跳过一个数组元素的存储空间.

总结

二维数组 a 的各种表示形式及其含义

- ① a : 数组 a 的起始地址.
- ② $a+i$, $\&a[i]$: 第 i 行的起始地址.
- ③ $*(a+i)$, $a[i]$: 第 i 行第 0 列元素的起始地址.
- ④ $*(a+i)+j$, $a[i]+j$, $\&a[i][j]$: 第 i 行第 j 列元素的起始地址.
- ⑤ $*(*(a+i)+j)$, $*(a[i]+j)$, $a[i][j]$: 第 i 行第 j 列元素的值.



2. 指向二维数组元素的指针变量.

利用指向二维数组元素的指针变量,
可以完成二维数组数据的操作处理:

- ① 定义与二维数组相同类型的指针变量;
- ② 在指针变量与要处理的数组元素之间建立关联;
- ③ 使用指针的运算就可以访问到任何一个数组元素。

3. 指向由几个元素组成的一维数组的指针变量.

- 一个二维数组相当于多个一维数组,
通过指向整个一维数组的指针变量,也可以
完成二维数组的操作处理.
- 定义一个指向由几个元素组成的一维数组
指针变量的一般形式:

数据类型 (*变量名)[整型常量]

`int (*P)[4];`

- ① 表示定义一个指向由4个整型数组
元素组成的一维数组的指针变量P;
- ② 指针变量P存储该一维数组的
起始地址.

又如: $\left. \begin{array}{l} \text{int } a[3][4]; \\ \text{int } (*P)[4]; \\ P = a; \end{array} \right\} \Rightarrow \begin{array}{l} P \rightarrow a[0] \\ P+1 \rightarrow a[1] \\ P+2 \rightarrow a[2] \end{array}$



又如: `int a[3][3] = {{0,1,2}, {3,4,5}, {6,7,8}};`

`int (*p)[3];`

`p = a[0];` X

`p = &a[0][0];` X

`p = a;` ✓

`int (*p)[10];`

`p = a;` X

4. 指针与字符串.

• 指向字符串首地址的指针变量称作字符串指针 (字符类型的指针)

• `char *变量名;`

eg1. `char *p;`

`char s[] = "abc";`

`p = s;`

eg2. `char *p;`

`p = "abc";`

• 字符数组与字符指针变量的区别:

① 字符数组占用若干个字节, 每一个字节存放一个字符;

字符指针变量用于存放字符串的首地址, 占用4个字节.

② 赋值方式不同:

eg1: `char s[15] = {"Hello"};` ✓

`char s[15]; s = {"Hello"};` X



eg2: `char *p = "Hello";` ✓

`char *p; p = "Hello";` ✓

5. 指针数组

• **数据类型 * 数组名 [整型常量表达式]**

eg: `char *a[3] = {"abc", "bcd", "g"};`

注①: `[]` 比 `*` 优先级高;

注②: 指针数组 `a` 中含 3 个指针 `a[0]`、`a[1]` 和 `a[2]`, 分别表示三个字符串的起始地址。

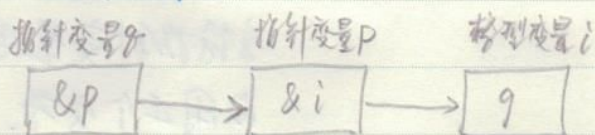
C. 指向指针的指针

eg: `int i = 9, *p;`

`p = &i;`



• 如果指针变量 `p` 的地址存储在指针变量中, 通过指针 `p` 访问变量 `i`, 中间必须通过两次指向运算, 这种访问方式称为“二级间接”访问方式。



• 二级指针的定义形式:

数据类型 ** 指针变量名;



eg. 上图的指针变量可以定义为

`int **p;`

其中:

- ① `p` 是一个指向指针的指针变量;
- ② 指针变量也是变量, 需要占据内存单元, 因而有相应的地址, 就可以再定义另外一种“指针”指向这个地址, 这种“指针”就是指向指针的指针.

eg. P31 例 4.17

总结:

1. 数组的定义、初始化和赋值方法及数组的用途.
2. 数组的所有元素均按顺序存放在一个连续的存储空间中, 数组名就是该存储空间的首地址.
3. 数组元素值的获取有两种方式:
 - ① 定义时初始化;
 - ② 循环中依次赋值.
4. 数组下标取值范围: $0 \sim n-1$.
5. 字符数组与字符串的联系与区别.
6. 指针与数组的联系与区别.



Mo	Tu	We	Th	Fr	Sa	Su
----	----	----	----	----	----	----

Memo No. 60

Date / /

又... ..

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...