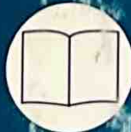
REPORT PAD

NAME C++ /C

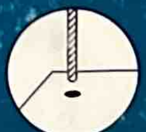
SUBJECT

YEAR



可平摊

易撕取



打孔，方便装订

金光集团APP中国企业
亚龙纸制品(昆山)有限公司
江苏省昆山经济技术开发区新南片
www.yalongpaper.net www.opexpress.com

PAA2B51H0AM904 拍纸本
企业标准 Q/320583WYYL002 252 x 179mm

Maslino®

个人资料

姓　　名 Name: 刘帮主

地　　址 Address: 奋斗

电　　话 Tel: 奋斗

籍　　贯 Nationality: 奋斗

星　　座 Constellation:

血　　型 Blood type:

出生年月 Birth Date:

身份证号 I.D.Card:

公司/学校名称 Company/school Name:

公司/学校电话 Company/school Tel:

公司/学校网页 Company/school Web:

公司/学校传真 Company/school Fax:

E-mail:

QQ:

MSN:

其他Other:

轻轻松松做笔记

节约每一寸纸

采用国际先进装订方式，可平摊，易撕取，使用方便，充分利用每一寸纸

一. 电子器件的两种稳定可转换的物理状态

二进制 0/1：整型实型、字符型及指令的存储、处理及传送的形式.

八进制 0~7 / 十六进制 0~9, A~F. ⇔ 日常生活：十进制 0~9

1. 二进制数 → 十进制数：

每一位根据位权 × 2^i, $i \in N$

十进制数 → 二进制数：

整数部分：除2取余法(精确) + 小数部分：乘2取整法(一定精度)

2. 十六进制数 → 十进制数：

每一位根据位权 × 16^i, $i \in N$.

十进制数 → 十六进制数：

整数部分：除16取余法(精确) + 小数部分：乘16取整法(一定精度).

八进制数与十进制数间的转换类似.

3. 十六进制数、八进制数 ⇔ 二进制数.

∵ $16=2^4$, $8=2^3$, ∴ 四位$()_2$=一位$()_{16}$, 三位$()_2$=一位$()_8$.

注意：$()_2$转换为$()_{16}$或$()_8$时 以小数点为中心向左右以四位(三位)为单位展开转换.

不够的单位以0补之.

NO.

Date

二、二进制数的算术与逻辑运算

1. 加法：0+0=0，0+1=1+0=1

   1+1=10　逢二进一，向高位进位

2. 减法：0-0=1-1=0，1-0=1

   0-1=1　同时向高位借1

3. 乘法：0×0=0，0×1=1×0=0，1×1=1

   除法：可进行竖式除法，用到乘法和减法运算

4. 或（逻辑加）：$0\vee0=0$，$0\vee1=1\vee0=1\vee1=1$ ……并联、并集

5. 与（逻辑乘）：$1\wedge1=1$，$0\wedge1=1\wedge0=0\wedge0=0$ ……串联、交集

6. 非（逻辑否定）：$\bar{1}=0$，$\bar{0}=1$ ……补集

7. 异或运算：$0\oplus0=1\oplus1=0$，$0\oplus1=1\oplus0=1$

三、计算机中数的表示——定点数、浮点数

1. 定点数

{整数：定点整数 / 纯小数：定点小数} 最高二进制位是符号位，0正1负
一般有8k位，$k\in Z$ = k字节 = 2k十六进制位

由于定点小数中小数点默认为在符号位后，不占二进制位，故与一定点整数与之在二进制定点数表示下相同，此时应视具体情况确定数。

1)原码：即以上定点数的计数法，符号位+数，不能直接用于运算(同号不能相减)

用n个二进制位来存一原码，能表示 $\pm(2^0+2^1+\cdots+2^{n-1})=\pm(2^n-1)$ 以内的整数

↓

2)反码：{正数的反码与原码同 / 负数的反码对该数原码除符号位外各位取反} 原码转换为补码或偏移码中起中介作用

3) 补码 {正数的补码=反码=原码; 负数的补码=反码最后一位+1}

用n个二进制位来存一补码，能表示 $+(2^0+\cdots+2^{n-1})=+2^{n-1}-1$

及 $-(2^0+2^1+2^2+\cdots+2^{n-1})-1=-2^{n-1}$ 以内的数.

计算机中，所有加减运算都化为补码的加法运算，符号位参与运算，

计算结果为补码，可转回原码.

4) 偏移码(移码) = 补码的符号位取反. (0负1正)

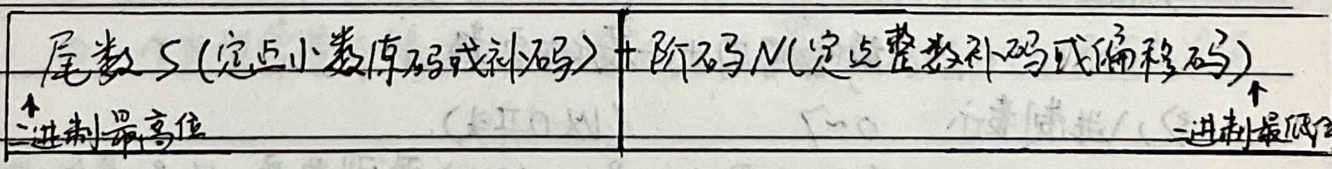
用n个二进制位来存一移码，能表示的整数范围 $-2^{n-1}\sim2^{n-1}-1$.

移码加减运算后再取移码才是移码形式的结果.

2. 浮点数.

二进制数 $P=S\times2^N$，S：定点小数，N：定点整数

(尾数) (阶码)



NO.

Date . .

四. 基本数据类型常量

常量：程序执行中值不变的量。

编译系根据表示形式来识别常量的数据类型，不同数据类型的常量在计算机中所占字节数不同.（字符型：1B；整型：2B/4B(长)；实型：8B）

1. 整型常量

有/无符号 基本整型常量	计算机中二进制补码表示	→ 2B、十六位 有符号 $-2^{15}\sim 2^{15}-1$；无符号 $0\sim 2^{16}-1$
有/无符号 短整型常量		→ 2B、十六位 有符号 $-2^{15}\sim 2^{15}-1$；无符号 $0\sim 2^{16}-1$
有/无符号 长整型常量		→ 4B 三十二位 有符号 $-2^{31}\sim 2^{31}-1$；无符号 $0\sim 2^{32}-1$

输入计算机时的表示：

1) 十进制表示：0~9，+ or −.（正整数前可不加+）.
长整型常量后加一"L"，不加L但超基本整型常量的数也认为是"L".

2) 十六进制表示. 0~9，a~f.（以0x开头）
前面可用+、−表示正负数.

3) 八进制表示 0~7 （以0开头）.
前面可用 +.−号. 但对整型常量的符号位含于相应的十六进制或八进制数中，故加+(−)的十六(八)进制数未必是正(负)数，因此，十六/八进制一般用于无符号整型常量.

2. 实型（浮点型）常量

1) 十进制表示：0~9，+ or −.和小数点（必有）

2) 科学记数法. $A_1 e B_1$ B_1是整数，可正可负. A_1可为任小数 ← 输入时

计算计中 实型常量 $P = S \times 2^N$. ← 计算机中

S：尾数 N：阶码.

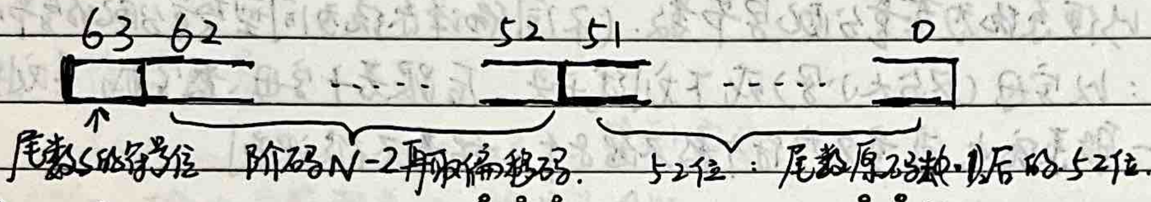
即 $P = 0.1\ldots\ldots \times 2^N$.
↑ 前两位已定.

NO.
Date · ·

例：十进制实型数97.6875

$(97.6875)_{10} = (1100001.1011)_2$

$= (0.11000011011)_2 \times 2^7$



0~63 共64位，一个实型常量占8B.（16个十六进制位）

注意：① 尾数的符号位放在64个二进制位的最前一位（63位）

② 尾数原码除符号位后第一位必是"1"（任何数都可实现此），故可省去此位，计算时再加上.

③ 阶码先转化为二进制原码，再减去(10)₂即(2)₁₀，所得11位数再取偏移码.

④ 一个实型常量可精确至小数点后53位（含除去首位必有的1），但仍有误差.

3. 字符型常量：

一对单引号（单撇号）括起来的单个字符或反斜杠开头的（转义字符）

如 'a'、'*'、'\n'、'\''

一个字符型常量用1B（八位）存放，存放的是该字符的ASCII码值（2^8种）

例 { 6是整型常量，占2B，在计算机中为$(110)_补$ = 0000 0000 0000 0110.

6L是长整型常量，占4B，在计算机中为 0……0 0110.（28个）

0.6e1是实型常量，占8B，在计算机中为 $(0.110)\times 2^3$ = 0 10…01 10……0.（9个）（51个）

'6' 是数字字符，占1B，在计算机中为其ASCII码值54 = $(110110)_2$.

$= (00110110)_2$.

No.
Date

C语言数据类型：整型、实型、字符型、枚举型、构造型、指针型.
（整型、实型、字符型：基本数据类型.）

五. 基本数据类型变量.

变量：程序执行过程中值可变的量，对应于计算机中的某个内存空间.
不同变量所占字节不同，故定义变量时指出变量的数据类型(变量名)
以便系统为变量分配字节数.（不同编译系统为同型变量分配的字节可异）
变量名：以字母（区分大小写）或下划线开头，后跟若干字母、数字或下划线.
一般系统中，变量名字符个数不超8个，超者不被识别.

1. 整型变量.

短整型2B { short(+变量名) 或 short int(+变量名) ⇒ 有符号 $-2^{15}\sim2^{15}-1$（有符号位）
unsigned short 或 unsigned short int ⇒ 无符号 $0\sim2^{16}-1$ }

基本整型2B { int ⇒ 有符号 $-2^{15}\sim2^{15}-1$
unsigned int 或 unsigned. ⇒ 无符号 $0\sim2^{16}-1$ }

长整型4B { long 或 long int ⇒ 有符号 $-2^{31}\sim2^{31}-1$
unsigned long 或 unsigned long int ⇒ 无符号 $0\sim2^{32}-1$ }

注意. ① 一个类型说明语句可同时定义多个变量，变量间用“，”分隔。
定义变量时也可为变量赋初值（初始化），在程序中可通过重新
赋值改变所赋初值.
② 未对变量赋值前，这些变量值是随机的.
③ 对各整型变量数据类型，存放不下输入数据时，从后往前取数.
（即按小数点向左向右取）
④ 各类型盛满了的数都视作补码表示！
（最高位被数值填了）.

NO.

Date · ·

2. 实型变量.

{ 单精度型：6~7位有效数字
双精度型：15~16位有效数字. } 尾数中 ~ float 4B
~ double 8B

3. 字符型变量.

char ~1B ~ ASCII码 ~ 整型常量（低8位：1B）

{ %c：输出字符型数据的格式说明符
%d：输出基本整型数据的格式说明符

故C中的字符数据（ASCII码）与整数（1B：0~2^8-1）之间可通用.
↳（0~126） ⟺双射⟺ ↳ 原码取低8位的值（0~255）→（0~126）

六. 数据的输入与输出.

{ 输入输出的设备：键盘/显示器…
输入输出数据的格式：整型、实型、字符型…
输入输出的具体内容.

调用"stdio.h"库中的输入输出 { printf（"%格式…"，内容） 格式输出
scanf（"%格式…" &地址） 格式输入
putchar（…） 字符输出
getchar（…） 字符输入.

1. 格式输出函数.

1）整型格式说明符 { 十进制：%d（基本）、%ld（长）、%u（无符号基本）、%lu（无符号长）
八进制：%o（基本）、%lo（长）、加"#"在%后输出 0….
十六进制：%x（基本）、%lx（长）、加"#"在%后输出 0x…

（2B 4B）

百分号后可加 +（加号）、-（左对齐）、m（总位数）.

亚龙纸制品

NO.

Date ·) ·

8B（作为双精度）

2）实型格式说明符 { ~~小数 数字 十进制小数点~~ 实数式：%f
指数式：%e
由系统依据宽度m最小决定：%g

%后可加 m.n m表宽度即总位数（小数点算一位）、n表小数点后的位数
（不输默认为取六位，取数时四舍五入）

3）字符型格式说明符 1B
%c

注：① 除 %、d、f、c、s（字符串）、l…以外，其它双引号中的字符按原样输出
② 先计算各项目（常量/变量/表达式）的值，按各项目应占的字节数依次将它们存入"输出缓冲区"，根据各格式说明符依次从"输出缓冲区"取出数据，若是非格式说明符则按原字符输出。
③ 计算机内存数时皆存 整型（补码）、实型（原码+偏移码）、字符（ASCII码的补码）。
数据的低字节部分（从左向右 1B 1B数）存于"存储空间"的后面。
数据的高字节部分 …… 前面。
如 0X 00 00 00 01（1B 1B 1B 1B） ⇒ 0X 01 00 00 00
（输入时） （存储空间中：补码）
④ long 强制转成 int 时在存储空间中只取前 2B（从左向右）
（顺序已反的补码形式）

2. 格式输入函数

1）整型：与输出同
2）实型：{ 单精度 4B：%f 或 %e
双精度 8B：%lf
不能用 m.n 表位数

3) 字符型：%c，%mc.

注意：① 内存地址表中的项是变量地址，彼此用","分隔，如 &a，&b，&c.

② 格式说明符间无其他字符，输入数据时两数据间用"␣"、"Table"或"↵"分隔；若有其他字符，则输完其他字符再输数据.

③ 一个字符型变量只能存放一个字符. 截取第一个字符赋给 char (%mc)（前1B）

④ 不能用 m.n 指定小数点后的宽度.

⑤ 一个C程序开始执行时，C系统就在计算机内存中开辟输入缓冲区，暂存从键盘输入的数据。当执行到一个 scanf() 时，就检查输入缓冲区是否有数据.

if 已有(上一scanf剩下没读完的)，依次按格式说明符从中取出并转为二进制码，再存于内存地址表的相应地址。(由高位向低位取)

else 没有(输入缓冲区位空)，待用户用键盘输入并高敲〈回车〉或〈换行〉后，依次按格式说明符中还没用过的，从输入缓冲区中取数据……

if 从输入缓冲区中取数据，遇〈回车〉/〈换行〉then 输入缓冲区清空.

so 在输入函数的"格式控制"一栏中，最后不能加换行符'\n'!

3. 字符输出函数

在当前光标位置处输出 c 所表示的一个字符.

putchar(c)　　c = 字符型常量 / 字符型变量 / 整型变量 / 整型表达式 / 转义字符.
（要加左右单撇号）.　　（变为ASCII码）

4. 字符输入函数.

接收从键盘输入的一个字符. {char x; x = getchar();}
= {char x; scanf("%c", &x);}.

一个getchar()只按顺序从输入缓冲区接收一个字符，剩下的给下面用，以〈回车〉结束

NO.

Date . .

七. 运算—运算符—表达式

1. 赋值运算"=": ~~赋值表~~ 赋值语句:

左值(变量名) = 表达式 左值(变量名)=表达式;

功能: 计算右边表达式的值赋给左边, 即从右至左执行.

说明: ①"="两端数据类型不一致时系统自动类型转换。

② 允许在"="前加其他运算符构成复合赋值运算符.

e.g "+=" "-=" "*=" "/=" "%="

x = x/2 等价于 x/=2.

2. 算术运算. "+(双目:加; 单目:取正)"

"-(双目:减; 单目:取负)".

"*(双目:乘)"

"/(双目:除. 整型除整型取整商)"

"%(双目:取余, 只用于整型相除)".

算术表达式: 用算符将运算对象连接的式子.

说明: ①不同数据类型进行混合运算时, C系统按一定顺序在运算过程中自动逐步进行数据类型转换.

转换顺序: (低) int → unsigned → long → double (C中所有实型格式)

↑ char(按ASCII码值运算)或short ↑ float.

②. 强制类型转换: (类型名)(表达式).

e.g.1 (int)(x-y), 将表达式x-y的结果强制转换成int.

e.g.2 (double)x/y. 将x强制转换成double.

3. 关系运算.

"<"小于 "<="小于等于 ">"大于 ">="大于等于.

NO.

Date · ·

== 等于　　!= 不等于.

说明：关系表达式的值为1（条件满足，命题是真的）或0（条件不满足，命题是假的）

4. 逻辑运算（对真值"非0"和"0"的运算，非0表真，0表假，既是真值也是一般十进制数）

"&&" 与　"||" 或　"!" 非.

- 非0 && 0 == 0，非0 && 非0 == 1，0 && 0 == 0.
- 非0 || 0 == 1，非0 || 非0 == 1，0 || 0 == 0
- ! 非0 == 0，!0 = 1.

说明：各运算优先级（先算 → 后算）.

! → 算术运算符 → 关系运算符 → && 与 || → 赋值运算符.

5. 增1与减1运算.

功能：　$x=++n \Leftrightarrow n=n+1; x=n;$

　　　$x=n++ \Leftrightarrow x=n; n=n+1;$

说明：++ -- 只用于 int 或 char（ASCII码值），不能用于常量或表达式.

6. sizeof 运算

功能：将表达式计算结果或某种类型的量所占的字节数算出.

sizeof（表达式），　sizeof（类型名）

说明：sizeof 运算符可出现在表达式中.

e.g.

```
#include <stdio.h>
main
{ char a, b;  scanf("%d,%d", &a, &b);
    b= a+ sizeof((a+b)/a);
  putchar(b);
}
```

亚龙纸制品

NO.

Date . .

7. 逗号运算（顺序求值运算）

功能："，"作为分隔符；作为顺序求值运算符.

- 作为分隔符
  - eg1：一个变量说明语句可同时定义多个变量，其间用"，"分隔.
  - eg2：printf () 中各参数用"，"分隔.
- 作为顺序运算符：子表达式1，子表达式2，…，子表达式n.

  从左到右计算各子式的值，算到子式n的值（各子式间可能含相同的变量，故从左到右算时，变量值会变且不断传递以影响后面的子表达式的值）.即为逗号表达式的值!

说明："，"是所有运算符中级别最低的（最后运算）

eg.
```
#include <stdio.h>
main()
{ int a,k; ~~scanf("%d",&a);~~
  printf("%d\n", a=((k=3,4*k),k+sizeof(int)));
}
```

8. 条件运算符.（三目运算 or 三元运算）.

表达1？表达式2：表达式3.

功能：表达式1的值非0（真）时，取式2的值；为0（假）时，取式3的值.

说明：1) 运算优先级：单目>双目>三目>…

! 与 ++、-- 与 sizeof > 算术运算符 > 关系运算符 > && 与 ||

\> 条件运算符 > 赋值运算符 > 逗号运算符.

2). 条件运算符的结合方向为从右到左（赋值运算也是！）.

如. a>b ? a : c>d ? c : d

⇕

a>b ? a : (c>d ? c : d)

八、编译预处理

1. 宏(Macro)定义

~~八、宏(Macro)定义、编译预处理~~

1) 符号常量定义

功能：为减少程序中重复书写某些字符串的工作量，将此字符串定义成一个符号常量。 #define 符号常量名 字符串

(一般是大写字母) (末尾不加";"，单独占一行)

其作用域直到出现"#undef 符号常量名"截止。

eg
```
#define PI 3.14159
main()
{…}
#undef PI
```

2) 带参数的宏定义

功能：#define 宏名(参数表) 字符串(含参)

说明：为保证运算中不出现基于运算顺序的错误，

⇒{将字符串中的参数用括号括起来 / 将整个字符串部分用括号括起来} ⇒ 合理的计算顺序

2. 文件包含命令

文件包含：一个源文件(.c)可将指定的另一个源文件包括进来。

#include <文件名.扩展名>

功能：将指定文件的内容读到该命令所在处后一起被编译 ⇒ 避免重复操作

{.h：头文件(库函数) / .c：源文件：可用来定义新的函数或变量}

说明：1) #include 欲包含的文件内容被修改后，包括此文件的源文件需重新编译。

2) #include 可嵌套。

3) 一般将有公式性质的符号常量、带参数的宏定义、外部变量等放入此种文件中~

亚龙纸制品

NO.

Date . .

3.条件编译命令.

功能：对源程序.c中的部分内容在满足一定条件时才进行编译.

或：当满足一定条件时对一部分语句进行编译.

当条件不满足时对另一部分语句进行编译.

目标：使同一个源程序.c在不同编译条件下能产生不同目标文件.obj.

1) #ifdef 标识符
   程序段1 (不用花括号)
#else
   程序段2 (不用花括号).
#endif.

or

#ifdef 标识符
   程序段1;
#endif.

若"标识符"被定义过，编译程1；否则编译程2.

2). #ifndef 标识符
   程序段1
#else.
   程序段2
#endif

or

#ifndef 标识符
   程序段1;
#endif.

若"标识符"没被定义过，编译程1；否则…… 与1)恰好相反！

3). #if 常量表达式
   程序段1
#else
   程序段2
#endif.

or

#if 常量表达式
   程序段1;
#endif.

若"常量表达式值为真"(非0)，程1；否则程2.

任一常量表达式都能算出其值的(非0 or 0)！
(非0 → #if，0 → #else)

九. ~~循环结构~~ 顺序结构.

1. 语句.

1). 表达式语句：表达式；{ eg1：y = x*x+3； c = getchar()；
eg2：printf("%d；%d\n", a, c>d?c:d)；

2) 空语句：只包括一个分号.

3) 流程控制语句：break；continue；…

4) 函数返回语句：return；

2. 复合语句：{说明语句；语句部分；}.

说明语句 ↓ 如变量类型说明.
语句部分 ↓ 一系列可执行的语句.

说明：1). 一个复合语句语法上等于一个独立语句.

2) 复合语句作为一个语句可放在其他复合语句内部.

☆ 3). 一个复合语句内的"说明语句"只适合于本层中该说明语句以后的部分(包括其内层的复合语句)，在该复合语句外不起作用.

若该复合语句中未对"x"进行说明(外层已对"x"说明过了)，则"x"在该层中被赋的值可带到外层(同样可带到本层的内层).

⟹ 将本层看成外层的一个赋值语句而已！

十、选择结构.

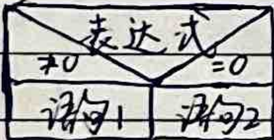
1. 单路分支选择结构

if (表达式) 语句/复合语句;

功能: 若表达式值为非0(真) 则执行表达式后的语句, 再执行if语句后的语句.

若表达式值为0, 则直接执行 if语句后的语句.

2. 两路分支选择结构.



if (表达式) 语句1;
else 语句2; → else的意思是($\overline{\text{表达式}}$)

功能: 很明显!

说明: 为严谨, 此三种都合法
- if (表达式) 语句; else ; ⇔ if (表达式) 语句;
- if (表达式); else 语句; ⇔ if ($\overline{\text{表达式1}}$) 语句;
- if (表达式); else ; ⇔ 空操作";"

3. 多路分支选择结构.

1) if (表达式1) 语句1;
if (表达式2) 语句2;
......

注意 表达式之间的条件要互斥, 没有交集(除非需要).

2). if (表达式1) 语句1;
else if (表达式2) 语句2;
...
else if (表达式n) 语句n;
else 语句n+1;

注意表达式之间的条件可不互斥(可有交集)
但"else"会帮表达式i去掉了其与i-1的交集.

NO.

Date . .

3). 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；

}

说明：① 语句i可以是复合语句.

② 若要在执行完语句i后跳出switch结构，在语句i后加break；

否则执行完语句i后，系统还会进入"case 常量表达式i+1"的执行.

→ 多个case可共用一组执行语句.

→ 各case的语句最后，一般加break语句，除非要共用case i+1后的语句.

③ 可不加"default"，这时，若(表达式)值不等于任何case后常量表达式的值，系统跳出switch；

default 与 case i 间位置可互换，但其后语句可能需修改.

④ "case i"中各i的值应各不相同（类似于映射：常量表达式i→语句i）.

且各i必须是"有限且离散"的（只取int或char）.

4) 算法举例：一元二次方程的求解.

① 自然语言描述：一元二次方程 $Ax^2+Bx+C=0$ 由 A.B.C 唯一确定，故应先输入 A.B.C.

如果 $A=0 \rightarrow Bx+C=0$ { $B=0$，方程无意义，输出"ERR".
$B\neq0$，方程有一实根 $x=-\frac{C}{B}$.

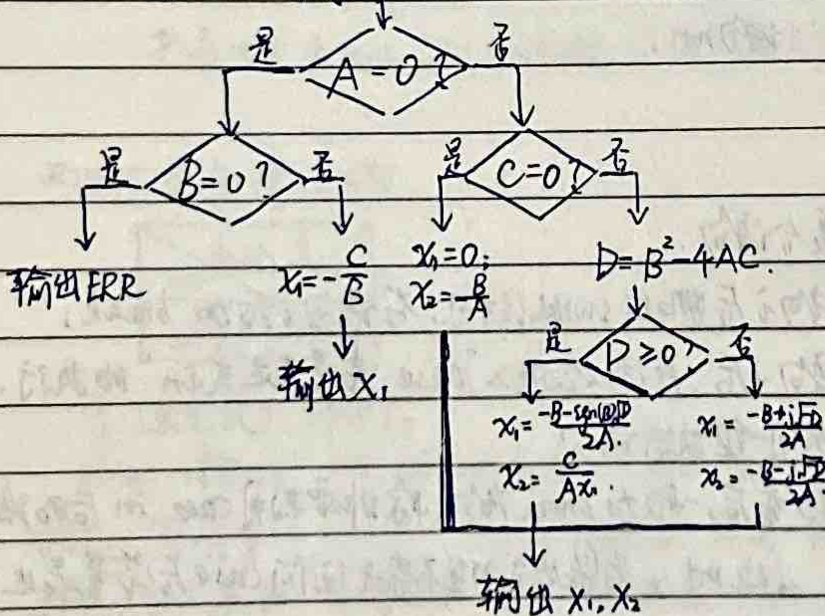
如果 $A\neq0$. { ~~$B=0 \rightarrow Ax^2+C=0$. { $A*C<0$，方程有两实根~~
~~$D=B^2-4AC$~~

见下页.

NO.
Date · ·

$\begin{cases} C=0, & x_1=0, x_2=-\frac{B}{A} \text{ 两实根} \\ C\neq 0, & D=B^2-4AC. \begin{cases} D\geqslant 0, & x_1=(-B-\mathrm{sgn}(B)\sqrt{D})/(2A), x_2=C/(Ax_1) \\ D<0, & x_{1,2}=(-B\pm j\sqrt{-D})/(2A) \text{ 共轭复根} \end{cases} \end{cases}$

② 流程图



③ 源程序

```
#include <stdio.h>
#include <math.h>.
main()
{ double A, B, C, D, X1, X2;
  printf("please input A, B, C:");
  scanf("%lf%lf%lf", &A, &B, &C);
  if (A==0)
     { if (B==0) printf("ERR\n");
       else printf("X1=%f\n", -C/B);
     }
```

```
else
{ if (C==0)
    { x1=0; x2=-B/A; }
  else
    { D=B*B-4*A*C;
      if (D>=0) { if (B>=0)  x1=(-B-sqrt(D))/(2*A);
                  else       x1=(-B+sqrt(D))/(2*A);
                  x2=C/(A*x1);
                }
      else { printf("x1=(-%f+jsqrt(-%f))/(2*%f)\n", B, D, A);
             printf("x2=-%f-jsqrt(-%f)/(2*%f)", B, D, A);
           }
    }
  printf("x1=%f\n x2=%f", x1, x2);
}
}
```

十一. 循环结构

当型循环

逻辑表达式（当条件满足）	
	循环体

当逻辑表达式 == 非0时，执行循环体，执行完后，再次回到逻辑表达式（判断条件）

…

直到逻辑表达式 == 0（条件不满足）.

直到型循环.

	循环体.
逻辑表达式（直到条件满足）	

先执行循环体，然后判断条件（计算逻辑表达式），若逻辑表达式 != 0 则退出循环，若逻辑表达式 == 0，继续执行循环体

…… 直到逻辑表达式 != 0（条件满足）

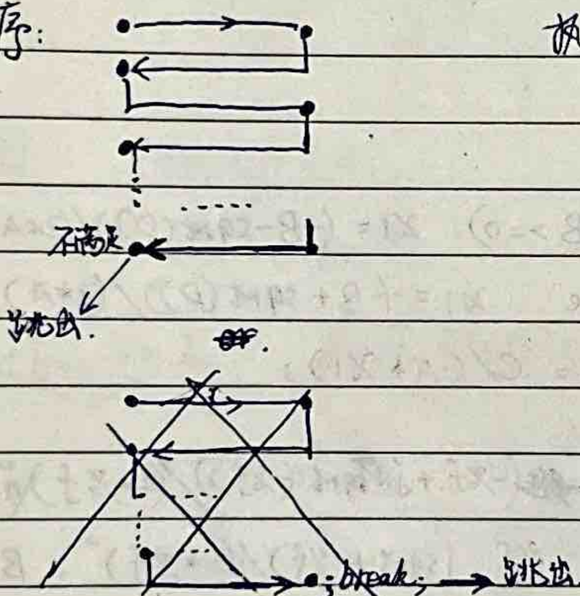
NO.

Date

1. while语句、for语句 —— 当型.

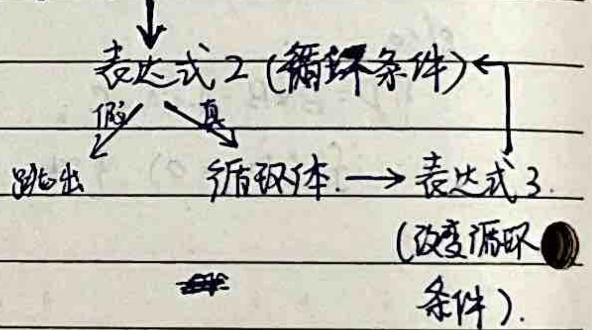
1) while (表达式) 循环体语句.

执行顺序:



2) for (表达式1; 表达式2; 表达式3) 循环体

执行顺序:



~~若循环体语句加break;~~

说明: ① while和for的循环体语句都可以是复合语句.

② for (表达式1; 表达式2; 表达式3) 循环体语句

⇕

```
表达式1;
while (表达式2)
{ {循环体语句}
  表达式3; /* 改变循环条件, 直到表达式2不满足 */
}
```

⇕

表达式1; for (; 表达式2; 表达式3) 循…;

⇕

表达式1; for (; 表达式2;) {循…; 表达式3;}

③ for语句中, 表达式1与表达式2可放"for"外, 其中两个";"不能省!

④ for中循环次数是确定的, while则不易事先确定.

常用"for"形式:

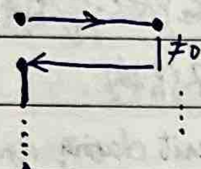
- for (i=初值; i<=终值; i=i+步长) 循环体
- for (i=初值; i>=终值; i=i-步长) 循环体.

↑ "当型"

2. do-while语句 —— 特殊直到型.

do 循环体 while(表达式);

执行顺序:



←— 直到满足时跳出循环.

说明: ① while中循环体可一次也不执行(逻辑表达式一开始就为0).
do-while中循环体至少一次.

② 显然,while与do-while的循环体中必有改变条件的语句.

③ do{循环体;} while(表达式);中,(表达式)后一定要加分号";"!

3. 循环的嵌套:顺序输出3~100间的所有素数.

算法: for(n=3;n<100;n=n+2)
用 2到√N间的所有整数除N,若都除不尽,则N为素,若其中之一除尽,则N为合;同时,要注意输出的所有素数的排列美观.

程序:

```
#include <stdio.h>
#include <math.h>
main()
{ int j=0,n,i,flag;       /*j用来记已打出素数的个数,n是3~100间的奇数,
  for(n=3;n<100;n=n+2)      i是2到[√n]间的整数用来试除n,
   { i=2;flag=0;            flag:当n为素数时由0被置为1.*/
    for(;i<sqrt((double)n);i=i+1)
      {if(n%i==0) flag=1;
       if(flag==0) {j=j+1;printf("%d",n);}
      }if(j%10==0) printf("\n");
   }
}
```

亚龙纸制品

NO.

Date . .

4. 算法举例.

break语句: { 跳出switch结构(放case(常量表达式)语句;之后).
退出当前循环结构.(在当前循环中应包括一个条件结构→when to break?)

continue语句: 结束本次循环的执行,但不退出循环结构.

(similarly, An "if" unit during when to "continue the present circle" is needed!)

①列举算法:

列举所有可能情况.用问题中给定的条件检验哪些是需要的,
常用于解决"是否存在"或"有多少种可能"等问题.

②试探算法:

若需列举的量事先不知道,只能从初始情况开始,往后逐步进行试探
直到满足给定的条件为止.

③密码问题:

加密: 报文中每个英文字母替换为其后的第k个字母.非英文字母不变,
若已超过'Z'或'z',则将循环到字母表的开始.

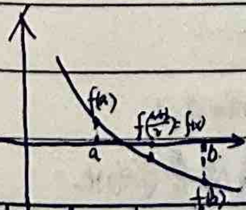
解密: 加密的逆运算.

```
#include <stdio.h>.
main()
{char c; int k; /*步长*/
 printf("input k:");
 scanf("%d", &k);
 getchar();   /*读取输入k时留下的<回车>*/
 c=getchar(); /*输入报文(一行字符),并读取第一个字符*/
```

```
while(c!='\n')   /*一行字符未读完*/.
{if((c>='a' && c<='z')||(c>='A' && c<='Z'))
    {c=c+k;
     if(c>'z'||(c>'Z' && c<='Z'+k))   /*若超上限,则循环到字母表开始*/.
     c=c-26;
    }
  printf("%c",c);   /*输出加密后的字符*/.
  c=getchar();      /*依次读取下一个字符*/.
 }
}
```

```
#include <stdio.h>.
main()
{ char c; int k; printf("input k:"); scanf("%d",&k);
  getchar(); c=getchar();
  while(c!='\n')
   {if((c>='a' && c<='z')|| c>='A' && c<='Z'))
      {c=c-k; if((c<'a' && c>='a'-k)||c<'A') c=c+26;}
   printf("%c",c); c=getchar();
   }
}.
```

④ 对分法求方程实根.



1) 取有根区间的中点 $\frac{a+b}{2}$ 赋给 x.　（已知 $f(a)f(b)<0$）.
2) 若 $f(x)=0$，则 x 就是根，输出 x
3) 若 $f(a)f(x)<0$，则根在 $[a,x]$ 内，$b=x$；若 $f(x)f(b)<0$，则…$[x,b]$ 内，$a=x$.
4) 若 $|a-b|<\varepsilon$，输出 x 的值，否则从 1) 开始重复执行

亚龙纸制品

NO.

Date

有时，在 [a, b] 内可能有多个实根，此时需将逐步扫描与对分法结合起来用。

while (x_1<=b)

1) 从区间左端点 $x=a$ 开始，以 h 为步长，逐步往后搜。(子区间 $[x_k, x_{k+1}=x_k+h]$)。

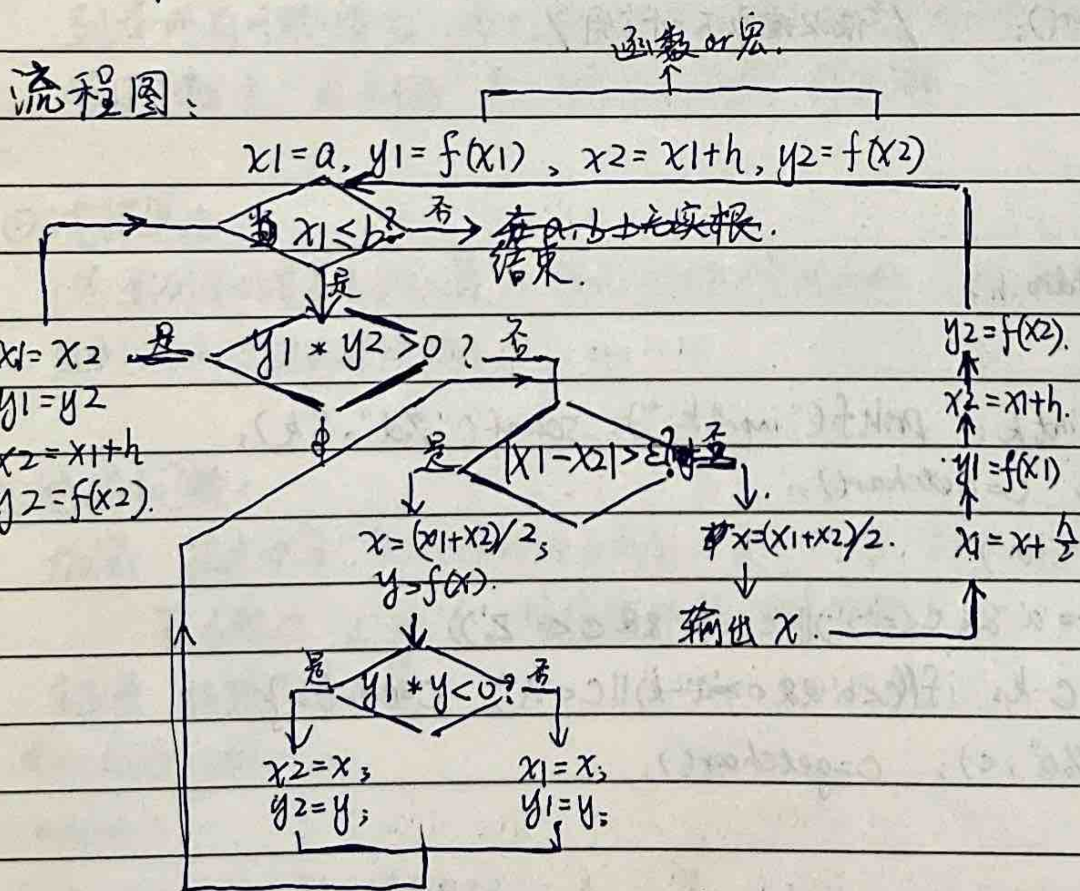
2) 若 $f(x_k)=0$，则 x_k 为一个实根，print 之，且从 $x_k+\frac{h}{2}$ 开始往后再搜。

3) 若 $f(x_{k+1})=0$，则 x_{k+1} 为一个实根，print 之，且从 $x_{k+1}+\frac{h}{2}$ 开始往后再搜。

4) 若 $f(x_k)f(x_{k+1})>0$，则说明当前子区间内无实根或 h 选得过大，放弃本子区间 (→可能造成根的丢失，故要合理选择步长)，从 x_{k+1} 开始往后搜。

5) 若 $f(x_k)f(x_{k+1})<0$，则当前区间内有实根，用对分法求之。

流程图：



⑤ 迭代法求方程实根.

1) 将欲求实根的方程改写成便于迭代的格式 $x=\phi(x)$.

2) 初步估计方程实根的一个初值 x_0，作如下迭代：$x_{n+1}=\phi(x_n)$，$n=0,1,2,\cdots$

直到满足条件 $|x_{n+1}-x_n|<\varepsilon$ 或达到迭代最大次数仍不满足精度要求为止.

输入初值，精度要求
↓
输入最大迭代次数
↓
$x_0 = x$
$x = \phi(x_0)$
$M = M-1$
↓
$|x - x_0| < \varepsilon \ || \ M = 0$ ？ —否→（返回）
↓是
输出"PAIL" ←是— $M=0$ ？ —否→ 输出 x.

十二. C语言中模块用——函数——来实现

{ 标准库函数：<math.h> <stdio.h> <string.h> <stdlib.h>等库中的函数
自定义函数：用户自编，用以解决专门问题，存于各种文件(.c)中.

函数的定义：类型标识符 函数名(形参表)
类型标识符 形参表;
{ 说明部分
语句部分
return(返回值);
}

说明：①.函数中返回语句的形式为 return(表示式) 或 return 表达式。
作用是将表达式的值作为函数值返回给调用函数. 其中表达式
数据类型(在说明部分中定义的)与函数类型要一致，若函数是无类型
(void型)的，则return后的表达式可略(不返回函数值，只是完成某种功能).
②形式参数表有多个，每个间用"，"分开.
③ 形参表中可直接对形参进行类型说明，但每个形参前都要有数据类型说明!
如 P(int n) →后不加";"!

NO. inner peace

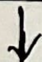
Date . .

④ 一个C程序有且只有一个main()，其它函数可任意多，当main()开始执行时，遇到函数就将其调用。(从x·c文件中)

⑤ 一个C程序中所有函数可放在一个文件中，也可放在多个文件中。

1. 函数的调用

区别函数的定义与函数的调用：

定义：类型标识符 函数名(类型标识符 形参1，类型标识符 形参2……)

调用：函数名(实参表列)。

函数原型：类型标识符 函数名(形参1类型，形参2类型……)。

↓

实质是在调用函数中对被调用函数进行的说明。

↓

作用是便编译系统易于检查"函数类型"、"参数个数"、"参数类型"与被调用函数中的是否匹配。

说明：① 函数调用 { 可以出现在表达式中：有函数值返回
也可单独作为一个语句：无函数值返回，只是进行某一操作。如void型。

② 被调用函数通过"函数原型"的形式在调用函数的"说明部分"进行说明

③ { 编译：按程序中语句的先后顺序进行，无论main()在哪。
执行：从main()开始

故若被调用函数在调用函数之前进行定义 / 在调用函数之前已由别的函数(也调用该被调用函数)说明了被调用函数的"函数原型" }时，可以不在调用函数中对被调用函数作"函数原型"的说明。

④ { 形参是定义函数时用字母代表的一组参数，只起象征作用。
实参是调用该函数时要进行实际操作的一组变量，可以是表达式，它们的类型和个数应与函数中的形参对应并用"，"间隔彼此。

NO.

Date . .

nothing better than an example :

```
#include <math.h>          /*此函数要调用math库中的sqrt()函数*/
sushu(int n)               /*函数名和形参的定义与说明*/.
{ int k, i, flag;          /*说明部分：说明了辅参与函数值的类型与代表符*/.
  k=(int)sqrt((double)n);      /*
  i=2                            语
  flag=1;                        句
  while ((i<=k)&&(flag==1))      部
    { if (n%i==0) flag=0;        分
      i=i+1;                     */
    }
  return (flag);           /*返回函数值1(素数) or 0(合数)*/.
}                          /*此sushu()函数的作用是判断形参n是(1)否(0)
                             是素数*/.

#include <stdio.h>
void main()                /*正规的main()应是void main()，因为main()是无类型的*/.
{ int k, sushu(int)        /*被调函数的说明放在“说明部分”*/.
  for(k=3;k<100;k=k+2)     /*实参是k，与定义sushu()中的n是一个东西但与n无关*/.
  if (sushu(k)==1) printf("%d\n", k);
}                          /*如果主函数void main()在sushu()之后，可不要说明sushu(int)*/
                           /*若sushu()在另一个文件x.c中，在void main()前还应include之*/
```

NO.

Date . .

2. 形式参量(在被调用函数中)与实际参量(在调用函数中)的结合方式.

调用模块: 被调用模块:

实参 ⇌ 形参

①地址结合: "调用模块"调用"被调用模块"时,

(数据双向传递). 不是将实参值直接传送给被调用模块中的形参,

而是将存放实参的地址传给形参.—形参与实参地址相同

—在被调用程序中对形参的操作(语句部分)就是对实参的操作.

说明: 被调用函数中改变了形参值,同时就改变了调用函数中的实参值,

故这种地址结合的方式下实参一般只能为变量(有固定的内存地址,

而表达式则无).

②.数值结合: "调用模块"调用"被调用模块"时,

(数据单向传递) 直接将实参值传给形参并被存放在形参地址中.

—形参与实参地址互不相同— 在被调用程序中对形参的

操作(语句部分)不影响调用程序的实参值.

说明: 当被调用程序执行完,返回调用程序时,被调用程序中

形参的新值不能传回调用程序,故实参不受形参影响,既可以是

变量,又可以是表达式。

C语言中,形参为简单变量时,均采用数值结合!

一个函数只能通过函数名返回一个函数值!

③局部变量: 在函数内部定义的变量,只在该函数范围内有效.

{不同函数局部变量可重名,互不影响;

形参即是局部变量,故调用时只能将实参传送给形参,

返回时不能将形参传回给实参变量.

NO.

Date

④全局变量：在函数外定义的变量，属于程序中的所有函数。

{ 可实现地址结合的功能（双向传递）。
{ 若局部变量若与全局变量同名，则局部变量作用域内全局变量无用。

3、变量的存储类型

一个用户程序在计算机中的存储分配：

程序区	：程序语句
静态存储区	：程序开始时分配的固定存储单元（eg.全局变量）。
动态存储区	：函数调用过程中进行动态分配的存储单元（eg.形参）。

变量与函数的基本量属性：

{ 数据类型：整、实、字符。
{ 存储类型：自动(auto)、静态(static)、寄存器(register)、外部(extern)

① 不作存储类型说明，默认为auto型，存于动态存储区。

② static说明的局部变量是局部静态变量，被调用后保留原值待下次调用。（做迭代时可能有用）。 存于静态存储区。

③ 如果在全局变量有效范围之外需要用全局变量，则应事先用extern说明。

eg：用swap()函数实现两个变量的值的交换：

```
~~extern int x,y;~~   #include <stdio.h>
main()
{extern int x,y; /*定义x,y为外部变量,这样,后面定义
scanf("x=%d,y=%d",&x,&y);的全局变量x,y在此也适用*/
swap();
printf("x=%d,y=%d\n",x,y);
}
```

亚龙纸制品

NO.

Date . .

```
int x, y;   /*在函数名前面定义的变量是全局变量,对定义后的所有函数适用
swap()      /* 没有形参,只返回某一操作(即交换两变量值),默认为整型*/
{int t;
 t=x; x=y; y=t;
 return;
}.
```

④. 如果在全局变量所在文件之外的另一个文件(.c)也需要用该全局变量,亦可用 extern 在本文件中标出.

eg.

```
/*file1.c*/
int x, y;   /* file1中的全局变量x.y*/
#include <stdio.h>
main()
{scanf("x=%d, y=%d", &x, &y);
 swap();    /*将在file2中定义*/
 printf("x=%d; y=%d\n", x, y);
}
```

```
/*file2.c*/
extern int x, y;   /*定义了x, y是外部变量,这样,本来全局变量作用
swap()               不到的file2文件中 全局变量x.y也生效了*/
{int t;
 t=x; x=y; y=t;
 return;
}.       /* 作用与③的例子相同 */
```

动态变量：在调用该函数时系统会给它们分配存储空间，在函数调用结束时自动释放这些存储空间，即动态地分配、释放空间

静态变量：程序开始时给分配存储区，程序执行完毕就释放.

⑤ 同一程序的两个函数文件中不能同时定义相同的全局变量，否则，系统会指出"x,y为重复定义".

⑥ 将全局变量定义为静态(static)变量，这样，该全局变量只能被本文件中的函数引用，不会被其它文件中的函数引用（包括函数的外部(extern)变量）.

4. 内部函数与外部函数.

内部函数：只能被本文件中其它函数调用的函数.

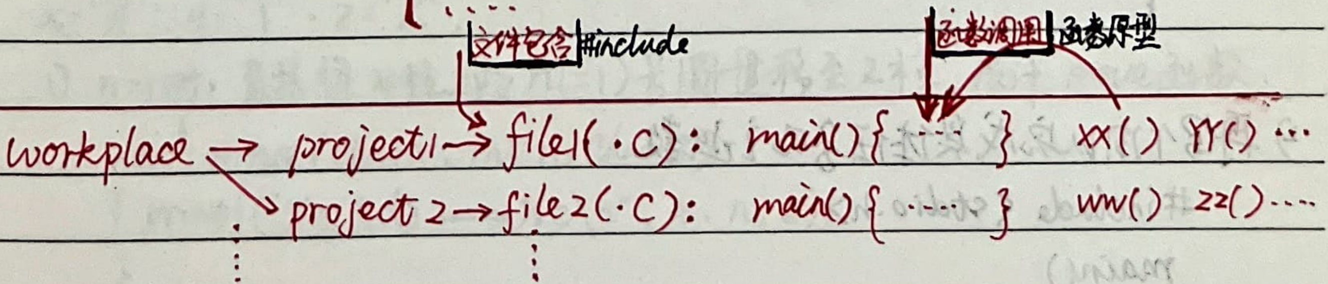
形式：static 函数类型 函数名(形参表).

外部函数：能被其他文件中的函数调用的函数，可不写extern.

形式：extern 函数类型 函数名(形参表).

小结：模块化程序设计：

- 头文件：<math.h> { double cos(double x)，double log(double x) ... }　<stdio.h> { ~~getchar~~ int getchar()，int putchar(char ch) ... }
- 用户自定义文件：<xx.c> { 函数类型 函数名(形参类型 形参1，形参类型 形参2 ...) ..... }



eg 在project1中，包含了很多个文件(.c)，其中只有唯一的一个main()函数。若在project1内调用函数，直接在调用函数中标明被调用函数的函数原型即可，无论被调用函数在哪个文件中；若要调用project1所含函数以外的函数，则应把被调用函数所在的文件用#include包含进project1.

亚龙纸制品

No.

Date

5. 算法举例

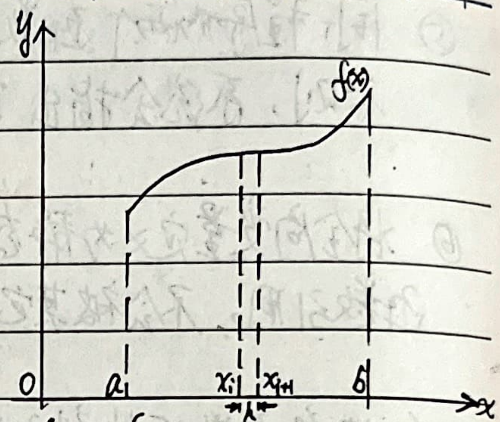
①梯形法求定积分. $S=\int_a^b f(x)dx$

$\approx \sum_{i=0}^{n-1}[f(x_i)+f(x_{i+1})]\cdot\frac{h}{2}$. 其中 $h=\frac{b-a}{n}$

$=\frac{h}{2}[\sum_{i=0}^{n-1}f(x_i)+\sum_{i=1}^{n}f(x_i)]$

$=\frac{h}{2}[\sum_{i=1}^{n-1}f(x_i)+f(a)+\sum_{i=1}^{n-1}f(x_i)+f(b)]$

$=\frac{h}{2}[f(a)+f(b)]+h\sum_{i=1}^{n-1}f(x_i)$, 其中 $f(x_0)=f(a)$, $f(x_n)=f(b)$.



1) 首先写对任何连续的f(x)都适用的用以求定积分的函数

```
double integ(double a, double b, int n)   /*定义integ函数及形参*/
{ int k;   /*用来计 1~n-1个数,以累加f(xi) */
  double h, s, p, x, f(double);   /* f(double)是被积函数f(x)
                                      的函数原型 */
  h=(b-a)/n;
  s= h*(f(a)+f(b))/2;
  p=0.0   /* p是double型的,用来累加f(xi) */
  for(k=1; k<n; k=k+1)
    { x= a+k*h; p=p+f(x); }   /* x是auto型变量,每次
  s=s+p*h;                      新值覆盖旧值,存于
  return(s);  /*返回积分值*/     动态存储区 */
}
```

2) 再写个用以完成具体任务的主函数.

```
#include <stdio.h>
main()
{ int n;
  double a=___, b=___, s, tab(double, doulo, int);
```

```
printf ("input n:");
scanf ("%d", &n);
s=tab(a, b, n)        /* 将tab()函数经代入实参a、b、n后的返回值赋给s*/
printf ("s=%f\n", s);
}
```

3) 最后需要写出具体的被积函数f():

```
#include <math.h>.
double f(double x)
{double y;
 y=______;
 return(y);
}
```

以上有 ____ 的地方都需用户视具体情况填写.

② 递归法解hanoi塔问题.

设计函数hanoi(n, x, y, z): 将x柱上的1~n号盘通过y移到z柱,

int n 表示圆盘个数,编号分别为1~n; char x, y, z 表示柱子名称,初始时 x='X', y='Y', z='Z'.

1) n=1时,直接将x柱上的n(=1)号圆盘移至z柱,设计move函数:

```
void move(char x, int n, char z)
{ printf("%c(%d) ——> %c\n", x, n, z);
}
```

2) n>1时,操作如下:

NO.

Date . .

① 将x柱上的上n-1个盘借助z柱移至y. 这仍是个hanoi塔问题：

hanoi (n-1, x, z, y);

② 此时，x上是第n号盘(最大的盘)，y上是n-1个盘，需将x上的第n号盘移至z，即

move (x, n, z);

③ 此时，x柱上无盘，y未变，z上是第n号盘，需将y上的n-1个盘借助于x移至z上，问题解决.

hanoi (n-1, y, x, z);

可以看出：hanoi(n, x, y, z) = hanoi(n-1, x, z, y). move(x, n, z).
hanoi (n-1, y, x, z).

一个n阶hanoi分解为了一个move(相当于1阶hanoi)和两个n-1阶hanoi，又，一个n-1阶hanoi可分解为两个n-2阶hanoi和一个1阶hanoi …… 一个n阶hanoi可分解为

$2^{n-1} + (2^0+2^1+\cdots+2^{n-2}) = 2^n-1$ 个1阶hanoi(即move).

3) 首先写出move()：

```
void move (x, n, z)    /*将"第n号盘"从x柱移至z柱*/.
int n;
char x, z;
{ printf("%c(%d) → %c\n", x, n, z);
```

4) 再写出hanoi()：

```
void hanoi(n, x, y, z)
int n;
char x, y, z;
{ void move (char, int, char);    /*对被调用函数move()进行的
                                    函数原型的说明*/.
```

```c
    if (n==1) move (x, n, z);
    else
      { hanoi (n-1, x, z, y);    /* 并未解决问题，只是将问题规模
                                    缩小(从n阶变为n-1阶)，但问题
        move (x, n, z);
        hanoi (n-1, y, x, z);       性质不变，直至 n==1 时，问题才可解决 */
      }
}        /* nothing to return */
```

5). 编写主函数 main()，提供初始态数据以解决具体问题.

```c
#include <stdio.h>
main()
{ int n;                                  
  char x='X', y='Y', z='Z';                } 说明部分
  void hanoi (int, char, char, char);
  printf("input n=");
  scanf ("%d", &n);                        } 语句部分.
  hanoi (n, x, y, z);
}
```

△ 数据类型
- 非派生类型：void型、整型、浮点型
- 派生类型：函数、数组、指针、结构、联合、枚举.

△ 存储类型

类	作用域	生命周期	连接性	关键字
auto	块	自动	内部	auto or none
register	块	自动	内部	register
static(块作用域)	块	静态	内部	static
static(文件作用域)	文件	静态	内部	static
extern	文件	静态	外部	extern

亚龙纸制品

NO.

Date . .

十三、数组

数组：相同数据类型元素的集合

一维数组的定义：类型说明符 数组名[常量表达式]；

二维数组的定义：类型说明符 数组名[常量表达式1][常量表达式2]；

说明：① 数组元素又称为下标变量。a[i]表示数组a[n]中第i个元素。

② 常量表达式的值是数组中元素的个数。对于n维数组，

元素个数为 $\prod_{i=1}^{n}$(sizeof(数组名)/sizeof(数据类型))。

③ 数组的命名规则与变量名相同。

④ 常量表达式必为整型，可包含符号常量(ASCII码)，不能是变量/形参！

⑤ 多维数组中，由最末维到最前维依次计下标

如 a[3][4] 的存储顺序：a[0][0] → a[0][1] → a[0][2] …… a[1][0]

…… a[2][0] …… a[3][0] …… a[3][4]。

1. 给数组元素提供数据：

1) 用赋值语句依次赋值：a[i] = xx

2) 用输入函数在循环结构中给数组元素赋值：eg a[n]

```
for (i=0; i<n; i++)  scanf("%d", &a[i]);
```

3) 定义时直接赋值。但只能对"静态存储"的数组初始化，即外部(extern)型和静态(static)型。如：

```
static int a[5]={1, 3, 5, 6, 10};
```

在对程序进行编译连接时就给予分配存储空间。

~~说明~~：

大多数微机编译系统中，亦可对局部动态(auto)数组初始化，如：

```
int a[5]={1, 3, 5, 6, 10};
```

在运行时才给予分配存储空间。

NO.
Date · ·

说明：
- static int a[10]={1,2,3,4,5} 中，未赋值的数组元素自动被赋为0.
- int a[10]={1,2,3,4,5} 中，未赋值的数组元素被~~系统随机赋值~~ 自动被赋为0.
- int a[10] 中，没有一个元素被赋初值，所有元素被赋随机值.
- static int a[] ={1,2,3,4,5} 中，由于没有常量表达式，自动定义为 a[5].

算法举例：矩阵相乘.

```c
#include<stdio.h>
void main(void)
{ int i,j,k, c[2][3];          /*两数组的积的行数和列数是已知的*/
  static int a[2][4] = {1,2,3,4,5,6,7,8};
  static int b[4][3] = {1,2,3,4,5,6,7,8,9,10,11,12};
  for (i=0; i<2; i++)          /* a[0][], a[1][] */
  for (j=0; j<3; j++)          /* b[][0], b[][1], b[][2] */
     { c[i][j]=0;   /*对动态数组的初始化*/
       for(k=0; k<4; k++)
           c[i][j] = c[i][j] + a[i][k] * b[k][j]     /* Cij = Σ(k=1..4) aik bkj */
     }
  for (i=0; i<2; i++)
  {for (j=0; j<3; j++)   printf("%6d", c[i][j]);   /*按2x3的格式打出*/
   printf("\n");          /*打一行空一行*/
  }
}
```

NO.

Date . .

2. 字符数组与字符串.

1)字符数组.

定义 { char 数组名[常量表达式]; 一维字符数组
char 数组名[常量表达式1][常量表达式2]; 二维字符数组.

赋值：规则与一般数组相同，未赋式的被自动赋为'\0'(ASCII码中的0字符)
但未赋任何值的动态字符数组中的元素是随机的.

元素：一个字符数组元素只存一个字符.

2)字符串常量.

特征：用一对双撇号括起来的一串字符，最后包括一个占1B的'\0'结束符.
可以用字符串常量对字符数组进行初始化，但不能用其对动态字符数组赋值.

eg: static char a[]="how do you do?";
⇔ static char a[]={'h','o','w',' ','d','o',' ','y','o','u',' ','d','o','?','\0'};

3)字符数组与字符串的输入与输出.

{ %c → 'ch'
%s → "......"

输入输出一个字符(%c)：

输入项：数组元素的地址，如 &a[3]。不要用单撇号括起。

输出项：数组元素，如 a[3]. 不用空格分隔(一次只读入一个char)

输入输出一个字符串(%s)：

输入项：数组名，相邻两字符串间用"空格"分隔，系统在"空格"前自动加'\0'.
若超出定义a[n]时的n个字符(包括'\0') { 若后面无数据，自动突破n
若后面有数据(如b[m])，被覆盖

系统自动在句末(空格前)加的!

输出项：数组名，系统遇'\0'则止.
→空格 回车

NO.

Date · ·

字符串输入输出实例见 textbook P202-204

4). 字符串处理函数 ∈ <string.h>.

puts (字符数组名): 输出一个字符串到显示屏.
- 遇'\0'则停输
- 功能同于 printf ("%s", 字符数组名) 或 putchar(...) × n (有n个putchar几个).

gets (字符数组名): 读入一个字符串到已定义的动态字符数组, 返回字符数组名(地址)
- 遇'↵'则停读.
- 功能同于 scanf ("%s", 字符数组名) 或 ~~getchar~~ ...=getchar() ...=getchar() ...

strcat (字符数组1/~~字符串常量~~, 字符数组2/字符串常量): 连接
- 字符数组1的长度[n]须够大以容纳被连接的字符串.
- 连接后自动取消字符数组1后的结束符'\0' (少了1B).
- 字符数组2也可是字符串常量, eg strcat (a, "abefg").

strcpy (字符数组1, 字符数组2/字符串常量): 拷贝.
- 字符数组1的长度[n]须够大以容纳被拷入的字符串.
- 只能用赋值语句对数组进行初始化, 而不能定义后再赋值, 但此函数可做到
- strcpy (s1, s2, n) 可将 s2 的前 n 个字符拷入 s1.

strcmp (字符串1, 字符串2): 比较字符串
- 按字典顺序比较 (~~...~~ ASCII码)
- 相等 → 0, a1[] > a2[] → 第一个不同字符的A码值, a1[] < a2[] → (负的)第一个不同的A码值.

出的被覆盖!

亚龙纸制品

NO.

Date . .

strlen(字符串)：测字符串长度.

- 可以在括号中(实参)写字符数组名，也可写字符串常量"……"
- 不包括系统自添的'\0'与系统自添的'\0'.
  (不够定义长度时) (空格前、末尾的)

strlwr(字符串) 大写→小写

strupr(字符串) 小写→大写.

3. 数组名作为函数形参.

调用函数和被调函数中分别说明和定义数组，其数组名可以不同但类型必须一致.

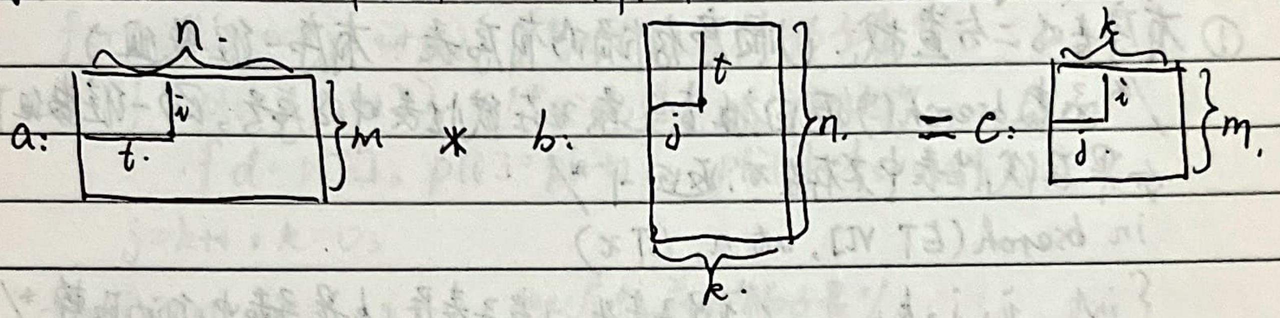
- 形参变量与实参变量间是数值结合：实参→形参⇒返回值，实参未变.
- 形参数组与实参数组间是地址结合：实参⇔形参⇒返回值，实参与形参同变.

- 数组形参定义：在被调函数中定义一个数组变量.
- 数组被调说明：为数组分配存储空间.
- 调用函数中数组名：该数组存储空间的首地址.(数组中第一个元素在计算机中的位置)

说明①在被调函数中说明的形参数组，包括在调用函数说明部分的函数原型中的形参数组，系统都不为其分配存储空间(即形参中数组名只是形式上的，真正为之分配存储空间的是实参数组)。在调用过程中，该形参数组名将与实参数组名结合，即形参数组名中存放的是实参数组的首地址.

②. 编译系统对~~形参数组~~作为形参的数组的大小不作检查，由上所指，调用时只将~~实参数组~~作为实参的数组的首地址传给形参数组，因此，为了通用性，被调函数中作为形参的数组不指定大小。如a[]、b[].
但为了控制数组的使用范围，要在被调函数中另设一个变量n.(形参数组大小

例：用二维数组写一个通用的两矩阵相乘的函数.




```
matmul(a, b, c, m, n, k)
int m, n, k, a[], b[], c[]      /* 整型一般变量与数组变量的声明 */
{int i, j, t, s;    /* i:1,…,m; j:1,…,k; t:1,…,n; s:计矩阵C中的元 */
for(i=0; i<m; i++)
for(j=0; j<k; j++)
    {s=i*k+j; c[s]=0;
    for(t=0; t<n; t++)   c[s]=c[s]+a[i*n+t]*b[t*k+j];
  }
return;
}
```

再写一个main()对具体问题进行处理：

```
#include <stdio.h>
main()
{int i,j, c[2][3];    /* 实参c[2][3]*, i,j 均是局部变量 */
static int a[2][4]={1,2,3,4,5,6,7,8};
static int b[4][3]={1,2,3,4,5,6,7,8,9,10,11,12};
int matmul(int a[], int b[], int c[], int, int, int);
matmul(a,b,c,2,4,3);
for(i=0; i<2; i++)
    {for(j=0; j<3; j++)  printf("%5d", c[i][j]);  printf("\n");}
 printf("\n");
}
```

亚龙纸制品

NO.

Date

4. 算法举例：

① 有序表的二分查找.（顺序存储的有序表：有序一维数组）

/* 函数bserch()返回被查元素x在线性表中的序号，即一维数组下标，如果在线性表中不存在x，返回-1 */.

```
in bserch(ET v[], int n, ET x)
{int  i, j, k;        /*i是子表头, j是子表尾, k是子表中, 自动取整*/
  i=1; j=n;          /*初始化表头表尾*/
  while (i<=j)
    {k = (i+j)/2;
      if (v[k-1]==x) return(k-1);
      if (v[k-1]>x) j=k-1; else i=k+1
    }.
  return (-1)
}.
```

② 冒泡排序

排序对象：线性表（一维数组）

原理：相邻数据的交换逐步将线性表变成有序.

工作量：设表长为n，最坏情况下，经 $\frac{n}{2}$ 遍从前往后扫和 $\frac{n}{2}$ 遍从后往前扫，需要比较的次数为 $\frac{1}{2}n(n-1)$.

```
bubsort(ET p[], int n)
{int m, k, j, i;
  ET d;
  k=0; m=n-1.
  while (k<m)  /*子表未空*/.
```

```
{ j=m-1; m=0;
  for(i=k; i<=j; i++)   /*从前往后扫描子表*/
    if (p[i] > p[i+1])   /*发现逆序进行交换*/
      { d=p[i]; p[i]=p[i+1]; p[i+1]=d; m=i;}
    j=k+1; k=0;
  for(i=m; i>=j; i--)   /*从后往前扫描子表*/
    if (p[i-1] > p[i])   /*发现逆序进行交换*/
      {d=p[i]; p[i]=p[i-1]; p[i-1]=d; k=i;}
}
return;
}
```

③. 选择排序.

原理：从线性表中选出最小元素，换到表的最前面，然后对剩下的子表采用同样的方法直到子表为空。

工作量：对长度为n的序列，要扫描n-1遍，最坏情况下需比较 $\frac{1}{2}n(n-1)$次.

```
selesort (p, n)
int n;   ET p[];
{int i, j, k;  ET d;
  for (i=0; i<=n-2; i=i+1)
    {k=i;
     for(j=i+1; j<=n-1; j=j+1)
       {if (p[j]< p[k]) k=j;}
     if (k!=i) {d=p[i]; p[i]=p[k]; p[k]=d;}
    }
  return;
}
```

亚龙纸制品

NO.

Date . .

④插入排序.

内容：将无序序列中的各元素依次插入到已经有序的线性表中.

原理：{ 表中只含1个元素：有序表.
表中前j-1个元素已有序，将第j个元素插入前面的有序子表中.

工作量：与"冒泡"相同，需$n(n-1)$次比较.

```
insort(p,n)
int n; ET p[];
{int j,k; ET t;
 for(j=1; j<n; j++)
 {t=p[j]; k=j-1;
  while((k>=0)&&(p[k]>t))
     { p[k+1]=p[k] ; k=k-1; }
  p[k+1]=t;
 }
 return;
}
```

NO.

Date · ·

十四、指针

每台计算机都包含可寻址的存储单元

1. 给标识符赋予数据，然后用标识符来操作数据的内容.

2. 指针作为一种非常有效的数据访问方法。包含可用于访问数据的地址.

指针常量：取用本身已存在的计算机地址集。不能改变、仅能使用。
确定地对应着计算机中的每个存储单元。

指针值：一变量的地址（&data）就是这个变量所占据的第一个字节位置.
它的指针值就是它占据的第一个字节的地址。

指针变量：可以将一个变量的地址存储在另一个变量中（赋值语句），
这就是指针变量.

NULL：定义于stdio.h中。一个不指向任何变量的指针含有的是特殊
的空指针常量.

① 通过指针访问变量：

既然已有一个变量及指向该变量的指针，如何将两者联系起来？

即，如何使用该指针？

C语言提供了间接运算符（*），用指针值去引用（寻址）它当前指向的变量

间接运算符：* 与地址运算符：& 相反，一个寻码，一个取址。

② 指针声明和定义：

数据声明：

类型	标识符（变量名）

指针声明：

类型 *	标识符（指针变量名）

用（*）来声明指针变量，这样使用（*）时
它不是一个运算符，而是一个编译器语法符号

亚龙纸制品

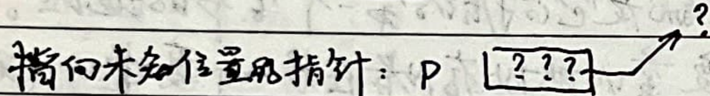
注意：某种类型的指针只能指向该种类型的变量
因为不同类型的变量的地址特点不一样。

③ 初始化指针变量.

在启动程序时，所有未初始化的变量包含的都是无用的垃圾.
同样，未初始化的指针将包含一些未知的内存单元。

图解：

未知变量：a ???



所以，如同最好对变量进行初始化一样，在声明和定义时就初始化指针：

int a; int* p = &a; 然后再在适当时候初始化 a 或者由键盘输入 a 的值.

④ 一个指针在不同时候指向n个不同变量/一个变量使用多个指针：

```
#include <stdio.h>
int main (void)
{ int a, b, c;  int* p;
 scanf ("%d %d %d", &a, &b, &c);
 p = &a;
 printf ("%3d\n", *p);
 p = &b;
 printf ("%3d\n", *p);
 p = &c;
 printf ("%3d\n", *p);
return 0; }
```

```
#include <stdio.h>
int main (void)
{ int a; int* p=&a, *q=&a, *r=&a
 scanf ("%d", &a);
 printf ("%d\n", *p);
 printf ("%d\n", *q);
 printf ("%d\n", *r);
 return 0;
}
```

指针的应用:

指针最为有用的应用在于函数。当将指针作为函数参数时,~~指针~~ 被调函数中对指针所指向的内容进行的操作,都能通过地址传回主调函数中,无论指针指向基本类型数据(int、char、double…)或数组、结构体、函数、甚至指针。否则,当只对同级(数据一级、地址二级、地址的地址三级…)的数据在被调函数中被操作时,只进行数值传递,即改变只发生在被调函数内(不体现在返回值上),主调函数中原来定义的变量不发生改变。

指针的大小:

所有指针的大小都相同。每个指针变量都包含机器中一个内存单元地址。(指针常量)

但指针所引用到的变量大小,显然地,有可能不同,取决于被引用数据的类型。

指向void类型的指针: 任何引用类型的指针都可被赋给一个指向void类型的指针,并且一个指向void的指针也可被赋给任何引用类型的指针。但由于void指针没有对象类型,不能被间接引用,除非被强制类型转换。

eg:
```
void* p;   int* q;
q = (int*) p;
```

指针的重要性:

作为一条通用的规则,如果值将发生改变,它必须作为指针来传递,如果不会被改变,那它可以作为值来传递。这样做,可以保护数据而不遭到偶然破坏。

亚龙纸制品

NO.

Date . .

1. 指向数组元素(一维、二维…)的指针

① 数组名就是指向第一个数组元素的常量：a ⟷ &a[0]

可以定义一个指针变量，并通过赋给它数组名的方式（数组名 ⟷ &数组名[0]）
将其初始化为指向数组中的第一个元素：p = a；p[j] = a[j]
可以定义一个指针变量，并通过赋给它 &数组名[i] 的方式
将其初始化为指向数组中的第i个元素，此时 a[i+j] ⟷ p[j]

② 除了以上的"索引"方法（即 a[i] 中的中括号就是索引的方式），还可以通过指针运算来得到数组 a[] 中任一元素的地址：

给定指针 p，p±n 为与 p 相距 n 个元素的地址（指针）← 必须在同一数组内
p±n 的意义是，在 p 指向的内存地址上，左(右)移 n * sizeof(p[0]) 个字节
所以 *(a+n) 与 a[n] 所表示的值相同

指针运算规则：

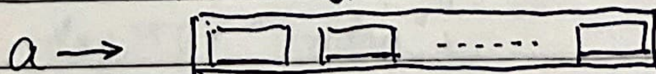
当一个操作数为指针，另一个操作数为整型数时，才可使用加法
eg. p+5 5+p
当两个操作数都为指针时允许用减法，其差值为两指针间元素个数
eg p1 - p2.
前一个操作数为指针，后一个操作数为整型数时，亦可用减法
eg p-5
用一元增量或减量运算符也合法：
eg p++ --p
当两个操作数为指向同一数据类型的指针时，允许使用关系运算符
eg p1 >= p2 其表达式值仍是 0 or 1.
p == NULL
p != NULL

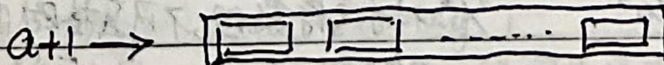
例：有序表的二分查找（search an ordered list using Binary Search）

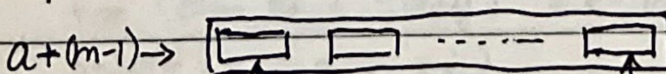
```
int binarysearch (int a[], int* end, int x, int** Mid)
{ int *first, *mid, *last;

  first = a;     /* first = &a[0] */
  last = end;    /* a[]的表尾作为初值赋给last */
  while (first <= last)    /* 表头在表尾之前 */
    { mid = first + (last - first)/2;    /* 求出表中地址 */
      if (x > *mid )          /* 目标 > 表中值 */
         first = mid+1;        /* 表中后一数作为表头 */
      else if (x < *mid)      /* 目标 < 表中值 */
         last = mid-1;         /* 表中前一数作为表尾 */
      else                    /* x = *mid，已找到表中的目标元素 */
         first = last+1        /* 改变指针所指向的地址，跳出循环 */
    }
  *Mid = mid;                 /* 将目标元素的地址保存 */
  return (x == *mid);         /* 找到：返回1；未找到：返回0 */
}
```

③. 指针与二维数组:

$*(*(a+i)+j) \leftrightarrow a[i][j]$







$*(a+m-1)$ 或 $a[m-1]$.　$*(a+m-1)+(n-1)$ 或 $a[m-1]+(n-1)$.

亚龙纸制品

NO.

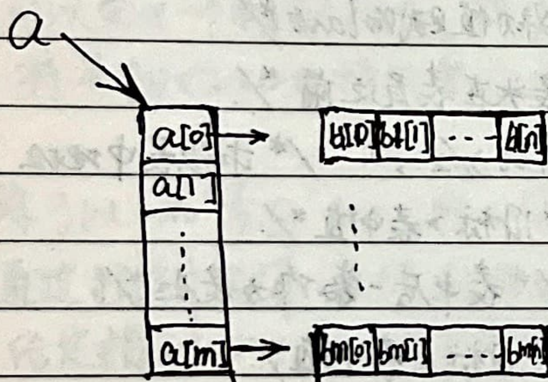
Date . .

④ 将数组传递给函数.

此时, 传数组名或指向数组的指针都可。

只要函数中进行操作的对象是数组元素, 则可实现地址结合.

2. 指针数组.



即数组中的每个元素都是指针. 定义与声明如下.

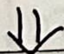
类型 * 数组名 [大小] ;

3. 字符串常量与指针.

字符串常量在内存中的存储有地址 ~~(...)~~ (needless to say)

因此, 可以通过指针来引用字符串常量。

由于字符串常量是一个字符数组, 其本身就是一个指向字符串首元素的指针常量. 也可定义一个指针变量, 将字符串首地址赋予之.



eg. char *p ; p = "Hello".　　/* 对于字符数组, 不可直接赋字符串 只能用 strcpy() 将串复制到数组 */

"Hello"[0] → H

"Hello"[1] → e

"Hello"[5] → '\0'.

NO.

Date . .

① 字符串的声明与初始化：

字符串声明 { char a[]：声明了一个字符数组，内存自动分配，可根据需要读入数据
char* a：为指针变量分配了存储空间但未为字符串本身分配空间
在使用字符串前，必须为其分配存储空间。

字符串初始化 { char a[] = "Good day";
char* a = "Good day";
char a[9] = {'G','o','o','d',' ','d','a','y','\n'};

② 字符串指针数组的应用：打印一星期的七天。

```
# include <stdio.h>
int main (void)
{ char* day[7];              /*定义字符串指针数组*/
  char** last ; int i;

  day[0] = "Sunday";
  day[1] = "Monday";
  day [2] = "Tuesday";
  day [3] = "Wednesday";
  day [4] = "Thursday";
  day [5] = "Friday";
  day[6] = "Saturday";
  last = day+6;              /*字符串指针数组的最后一个元素的地址*/
  for (char ** i = day; i <= last; i++)
  printf ("%s\n", *i);
  return 0;
}
```

亚龙纸制品

NO.

Date . .

4. 指向数组的指针.

指向一维数组的指针：类型 (*数组名)[大小]；

→更确切地说是指针名.

与指针数组的声明差一对括弧.

这是一个二级的数据结构：指针指向数组名，数组名指向数组首个元素的地址.

eg. int a[2][4]; ⟹

00	01	02	03
10	11	12	13

int (*p)[4]; ⟹ p → | 0 | 1 | 2 | 3 |

p = a;

由于a就是 a[2][4] 二维数组中首行的地址，所以p存被赋值后指向了a[2][4]的首行，即 p = &a[0]，且已有 a[0] = &a[0][0]

所以 p++ = &a[1].

进而有 *(*(p+i)+j) = a[i][j] .

(等同于 p = p+i . (*p)[j] = a[i][j]

5. 指针作为函数参数

这一点已在"指针的应用"中讲过，是地址传合的有效方式.

6. 返回指针的函数.

声明：返回类型 *函数名 (形参表).

eg. char *day = {"I", "love", "my", "parents"};

```
char *f (int i)
{ return (i<1 || i>3) ? day[0] : day[i] ; }
```

/*f() 函数返回了一个字符串 (字符串常量的指针) */

注意：函数返回的指针在该函数被调用的域内有确切的指向，切不可返回指向函数内部的局部变量的指针或指向void的指针。

7. 指向函数（返回值）的指针 —— 函数指针.

函数 { 类型：返回值的类型
　　　 地址：入口地址. ← 定义函数指针：

函数返回类型（*函数指针名）（形参表）.　　赋值：函数指针名=函数名.

① 只有返回类型和形参类型都相同的函数的地址才能赋给相应函数指针
② C不允许把函数作参数，但有了函数指针，就可把函数返回值的地址传给被调函数，实现地址结合上的函数作为被调函数的参数.

8. 指向结构体的指针.

通过这样的指针可实现被调函数返回一个结构体，即返回多个不同类型的数据.

定义：

① struct xxx
{
field list
};
struct xxx *p;

② struct xxx
{
field list
} *p;

③ struct
{
field list
} *p;

9. 包含指针的结构体：用起来亦很方便且节约内存.

NO.

Date . .

十五. 存储分配函数

当需要为一个对象预留存储位置时，C语言提供了两种选择
- 静态存储分配
- 动态存储分配.

1. 存储器的使用

- 程序存储器：main及所有被调函数所使用到的内存.
- 数据存储器：全局变量和常量、局部变量、动态数据存储.
  - 栈式存储器：在某一时刻，同函数可能有多个版本处于被调用状态(如递归时)，此时已为该函数的局部变量的多个"copy"分配了空间，这类存储体即栈式存储器
  - 堆式存储器：分配给程序一块未被使用的内存。用于在程序执行期间的赋值操作。当有来自存储分配函数的请求时分配存储空间.

2. 静态存储分配：

定义变量、数组、指针和流时，源程序中完全为之指定好存储空间，程序运行时，内存预留的字节数不可改变.

3. 动态存储分配.

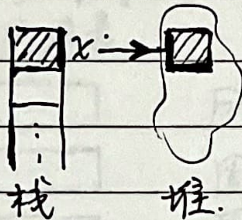
使用 malloc()、calloc()、realloc() 函数为数据分配存储空间。它们返回一个地址，因此，要访问在动态存储中的数据(引用在堆中分配的内存)，须使用指针；数据处理完后，须释放(free)内存.

图解：

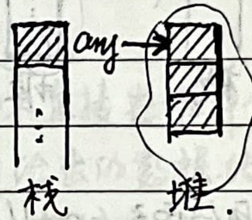
静态存储分配： int x;

int ary[3];

动态存储分配： int *x;
x=malloc(…);



int *ary;
ary=calloc(…);



{malloc、calloc、realloc、free} ∈ stdlib.h

① 块式存储分配(malloc)

malloc() 分配一块内存，在参数中指定所包含的字节数(通常用 n* sizeof(类型)填入)；将一个void指针返回给所分配存储的第一个字节，用户根据需要初始化这个被返回到malloc外的指针。如果未成功，malloc返回一个NULL指针。

eg
```
int *p;
p= (int *) malloc(6*sizeof(int));
```
(int *)：将void指针强制转化成int指针.

6*sizeof(int)：申请分配的字节数.

检查是否申请成功：
```
if(!(p=(int *) malloc(6*sizeof(int))))
    exit(100);
```
/*如果申请成功，p指向堆上的一段存储空间；未成功，p不含任何有效数据，并以错误码100退出该程序*/.

NO.

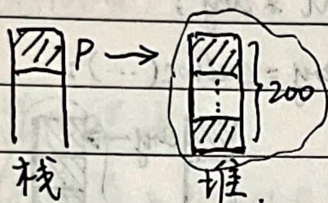
Date

②. 邻接存储分配 (calloc)

主要为数组分配内存，形参为元素个数，每个元素的字节数。

如 int *p;

p= (int*) calloc (200, sizeof(int));



③ 存储再分配 (realloc)

对于先前已在堆中用malloc被分配的存储空间，realloc删除或扩展这一空间从而改变块的大小（通过重新指定块的大小）。

如果由于存储空间无法扩展，则 realloc 重新在堆中开一个块，并将原来欲改变大小的块中的数据复制到新块上，并删除旧块。

P= realloc (原块的地址, n*sizeof(类型));

设原块大小为m.

- 若 n > m，新开 n-m个空间
- 若 n < m，删除尾端 m-n个空间

④. 释放申请的内存. (free)

free 的 declaration:

void free(void *p); /* p是申请的存储空间的首地址 */

free释放的不是指针，而是free指向的堆中的内容。free不会改变指针中的值(地址值)，但释放后再使用指针会导致逻辑错误，故

NO.

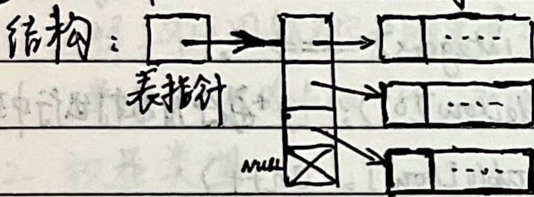
Date

释放存储之后，最好将指针值设为NULL以清空指针.

应用：动态数组.

特点：存储不规则数组，列宽和行宽自行设定. 在分配好行指针(用calloc)后，程序询问每行入口数，然后根据用户通过键盘所提供的数据进行填表.

结构：



所有数组都不直接在堆中分配空间，因为由堆所给出的数据结构受限于计算机存储空间.

函数1：创建一个指针数组，每个指针指向一个整型数组(首元素).

形参：无

返回：指针数组的数组名.

```
int ** build (void)
{ int rn, cn, row;  int** table;
  printf("\nEnter the number of rows in the table: ");
  scanf ("%d", &rn);                      /* 动态数组的行数 */
  table = (int**) calloc (rn+1, sizeof(int*));  /* 分配一个指针数组空间 */
  for (row=0; row < rn; row++)
    { printf ("Enter number of integers in row %d:" row+1);
      scanf ("%d", &cn);                  /* 输入第row行的列数 */
      table [row] = (int *) calloc (cn+1, sizeof(int));  /* 每行分配一个数组 */
      table [row][0] = cn;
    }                                     /* 每行第一个元素赋为列数 */
  table[row] = NULL;                      /* 最后一行指向NULL */
  return table;                           /* 返回这个二维数组 */
}
```

亚龙纸制品

NO.

Date

函数2：将数组的每一行填满根据函数1中cn所指示的个数的数据.
形参：已创建的二维动态数组.
返回：无.

```
void fill (int** table)
{ int row=0, cn=1;                          /*从第1行第2列开始填入数据*/
  while (table[row] != NULL)                /*函数1中已令table[末行]=NULL*/
    { printf ("\n row %d (%d integers) ====> ",
              row+1, table[row][0]);         /*每行前数打出行中列数*/
      for (cn=1 ; cn <= *table[row] ; cn++)
          scanf ("%d", table[row]+cn);       /*按每行第一个数,即指示的列数,输入相应个数的数*/
      row++;
    }
  return;
}.
```

函数3：main

```
#include <stdio.h>
#include <stdlib.h>
int ** build (void);
void fill (int** );                         /*函数调用声明:即函数原型*/
int main (void)
{ int ** table;
  table = build();                          /*将build()创建的动态二维数组
  fill (table);                              赋给指向指针的指针table,并
  return 0;                                  用fill()填充二维数组*/
}
```

十六、枚举、结构、联合及类型定义声明.

1. 类型定义:

可以将任何类型(基本、派生)定义成一个新名字，可易于理解.
推荐将重定义的类型名写为大写，以区别已有的数据类型.

eg. typedef int INTEGER;
typedef char* STRING;

这样,可以用新名字声明变量了: STRING stringPtrAry[20].

类比: #define PI 3.1415926.

2. 枚举类型.

基于标准整型的自定义类型，在此类型中，每一整型值被赋予一个标识符(枚举常量)，以提高程序的可读性。如果没有特地声明这样枚举常量的值，它们被按顺序默认为 0.1.2 …

① 声明: enum 类型名{枚举常量表};

eg enum color{red.blue.green,white};
(red → 0, blue → 1, green → 2, white → 3.)

② 定义变量: enum color skycolor; enum color flagcolor; …

③ 赋值: skycolor = blue; flagcolor = red; …
(blue → 1, red → 0)

④ 比较: 枚举变量间、枚举标识符与枚举变量间可以有"==、>、<"的关系.
多路分支: 由于枚举常量/变量对应于整数，所以可以将枚举类型用于任何对整型数据的操作.

```
eg. enum day {Sun, Mon, Tue, Wed, Thu, Fri, Sat}
    enum day date;          /*定义枚举变量*/.
    switch (date)
    {case Sun: ……; break;
     case Mon: ……; break;
      ⋮
    }
```

NO.

Date . .

⑤ 枚举类型的转化.

{ 隐式：赋值： int x; enum color y; x= ~~BLUE~~ blue; y=2 /* may be warned */
{ 显式：强制： enum color y; y= (enum color) 2;

⑥ 初始化.

{ 若未人为初始化，则系统自动按顺序 0.1.2 … 初始化枚举常量.
{ 人为初始化只需在定义时对枚举常量用赋值语句，若某个枚举常量被赋值 a，则之后的未被人为赋值，则系统自动为之赋 a+1, a+2, a+3 … 直到又遇到被赋了值的枚举常量.

eg. enum month {Jan=1, Feb, Mar …. Dec};
Feb → 2, Mar → 3, …… Dec → 12.

⑦ 匿名枚举类型：即无类型名的枚举类型.

eg. enum {space=' ', comma=',', colon=':', ……};
enum {OFF, ON}
OFF → 0, ON → 1

⑧ 输入输出：

其时 C 自动将枚举类型转化成需要输出的格式，输入相同.

eg. enum day {Sun, Mon, … Sat};
enum date1;
scanf ("%d", &date1);
printf ("%d", date1);

NO.

Date · ·

3. 结构体：

具有单一名字的相关元素的集合，元素类型可能不同。

① 定义方法1：

struct 结构体类型名

{元素1；元素2；元素3；……；}；

定义方法2：

typedef struct

{元素1；元素2；……；}结构体类型名；

② 结构体变量的声明：

struct 结构体类型名 结构体变量名；

结构体类型名 结构体变量名；

或者：struct 结构体类型名 ←--(可有可无)

{……}结构体变量名1，结构体变量名2 …… ；

③ 初始化

eg struct everytype

{ int a; double b; char c; int d[5]; char *p;};

struct everytype k =

{ 1, 2.4, 'A', {5,6,7,8,9}, "luckey"};

④ 访问结构体中的元素：

优先级最高的运算符："."

eg. scanf("%d %f %c", &k.a, &k.b, &k.c);

⑤ 结构体的复制：

相同类型的不同结构体变量间可以相互复制，只需用"="。

亚龙纸制品

NO.

Date . .

⑥ 定义结构体指针，之后可用指针来访问结构体.

eg.
```c
struct day
{ int year; int month; int date;};
struct day day1, *calendar;
calendar = &day1;
```

这样：结构体变量day1中的元素有以下三种方式可以访问.

```c
day1.year
(*calendar).year
calendar -> year.      /* 使用间接选择操作符 */
```

⑦ 结构体的嵌套.

eg.
```c
struct day
{int year; int month; int date;};
struct time
{ struct day a; int hour; int min; int sec;};
```

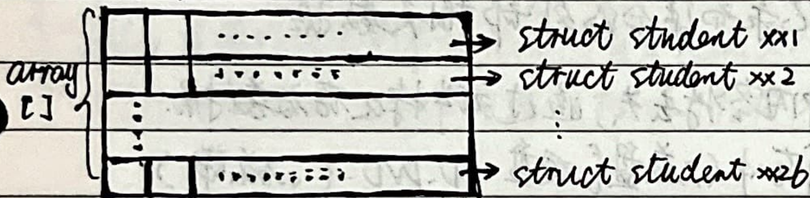
```c
// 初始化. struct time now
{{2008, 12, 31}, {16, 56, 59}};
// 引用. printf("%d/%d/%d %d:%d:%d",
        now.a.month, now.a.date, now.a.year,
        now.hour, now.min, now.sec);
```

⑧ 包含数组或指针的结构体.

eg.
```c
struct
{ char name[100];
  int score[4];
} Thomas = {"Thomas", {98, 88, 92, 86}};
```

```
int* p;  int total;
p = Thomas.score;
total = *p + *(p+1) + *(p+2) + *(p+3);
```

⑨. 结构体数组.



⑩ 将结构体的地址传给函数(作为实参).

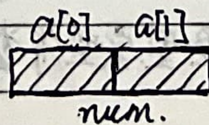
```
struct type
{ ...... ;}name, *p;
struct type name = {......};
p = &name;
f(p);
```

→ 将具体的 type 型名为 name 的结构体的地址传给了 f().

4. 联合体.

不同类型的数据共享(其中字节最大的数据的)内存.

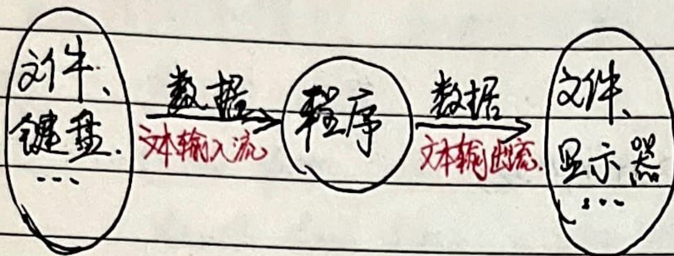
如 union share
```
{ char a[2];
  short num;
};
```



所以,访问内存中的数据(即访问联合体中的数据)时,一次只有一个数据(元素)存于内存中. 初始化时一次也只能有一个(第一个数据)会被执行.

No.
Date

十七. 文本输入/输出. ~~与二进制输入/输出~~



1. 文件：为保存数据记录而集合的外部相关数据.
   - △关机时主存中内容将丢失，通过文件持久存放数据.
   - △文件存于辅存中（磁盘{硬盘、CD、DVD…}、磁带）.
   - △文件是数据不能一次全部转移，须分批读写.

缓冲区：读取数据时，数据：外部设备→内存.
写数据时，数据：内存→外部设备.
   - △一块临时存储字段.
   - △根据程序需求，对主机与物理设备间的通信进行同步.
   - △读入的非程序当前所需的数据的存放地.

文件信息表：定义于stdio.h中的FILE类型结构体，包含
文件所对应的操作系统名，当前字符在文件中的位置……

2. 流：数据的来源或目的地是文件（包括程序源文件）或
物理设备（键盘，扫描仪…），数据以流的形式输入输出
   - 文本流：由字符序列组成，每行字符后有个'\n'（换行符）.
   - 二进制流：由整型实型等数据组成，使用各自的存储表示.

3. 流-文件处理.
   - 文件：独立存在的实体，以操作系统所知的名字命名.
   - 流：由程序创建的实体.

为了在程序中使用文件，必须将程序的流名字与操作系统下的文件名关联起来.

NO.

Date . .

①.创建流：

FILE* p;　　/*定义了一个指向FILE型结构体的指针*/.

/*P即(FILE)文件类型流指针，指向流(缓冲区中的地址)*/

②.打开一个文件：

- △用标准打开函数实现，这样流与该文件彼此关联.
- △FILE类型的结构体将记录所有有关该文件的信息。
- △打开函数返回文件结构体的地址，存在p中.

③.使用流名字.

使用流指针p，访问相关文件，进行读写.

④.关闭流.

使用标准关闭函数解除流名字与文件名字间的关联，删除文件内容.

说明：C语言提供标准流。在stdio.h中定义了3种流指针.

- stdin：指向标准输入流(eg. scanf())
- stdout：指向标准输出流(eg. printf())
- stderr：指向标准错误流.(~~EOF~~)

程序开始时自动创建，终止时自动关闭.

4. 标准输入/输出函数中的

文件打开/关闭，格式化输入/输出，字符输入/输出

⑴ 文件打开/关闭

fopen("文件名","打开模式");

返回：存储文件信息的文件结构体的地址，失败则返回NULL.

亚龙纸制品

NO.

Date . .

文件名：Microsoft Windows中，是一个字符串
最多由8个字符 + 3个字符的扩展名组成.

打开模式：告诉C如何使用这个文件（r、w、a…）.

eg.
```
#include <stdio.h>
...
int main(void)
{ FILE* p;
   p= fopen("A:\\MYFILE.DAT", "w");
   ...
}
```
/* 由于反斜杠(\)是转义符，为了在输出时得到一个反斜杠，在编码时用两个 */

eg. r：只读模式.
- 文件存在：在起始处标记.
- 不存在：返回NULL空指针.

w：只写模式.
- 文件存在：打开并删除原有数据.
- 不存在：创建之.

a：追加模式.
- 文件存在：在结尾处标记.（新数据被追加到文件末）.
- 不存在：创建之，逻辑上与只写模式相同.

NO.
Date

fclose（流指针）;
返回：成功关闭返回0，失败则返回EOF（∈stdio.h）.

打开和关闭错误测试.

```
# include <stdio.h>
# include <stdlib.h>
...
int main (void)
{ FILE* p;
  ....
  if((p = fopen ("XX.DAT", "r")) == NULL)
    {printf ("ERROR"); exit(100);}
  ...
  if (fclose(p) == EOF)
    {printf("ERR"); exit (102);}
  ...
}
```

② 格式化输入/输出.
scanf：从键盘获取文本流，转为数值存到变量中.
printf：从程序获取数值，转为文本流示于 screen.
即 scanf（"格式控制字符串"，地址表）;
printf（"格式控制字符串"，变量表）; } 终端输入/输出

NO.

Date . .

一般输入输出：fscanf / fprintf.

fscanf（流指针，"格式控制字符串"，地址表）；

fprintf（流指针，"格式控制字符串"，变量表）；

stdin

fscanf：读入来自文件（或终端）的流，将转换后的值存到地址表找到的变量表. 返回：转换数据的数目，失败返回EOF

fprintf：将内部数据转换成字符串写入文件（或终端）

返回：已写入文件中的字符数，失败返回EOF

↓

stdout. or EOF.

说明：1)输入流存于缓冲区，直到按'↵'（回车），系统才传递流中的数据.

即流末端总有一个'↵'

2)默认下，scanf将'↵'放在缓冲区中，若欲清空缓冲区.

- getchar();
- 用scanf多读一个字符 %c.
- 下一个scanf会忽略上一个scanf留下的'↵'.

例子：1). 读取并打印文件.

```
#include <stdio.h>.
#include <stdlib.h>.
int maci void.
{ FILE* P;          /*指向输入流*/.
  int a;            /*存放文件中的整型数据以待打出*/.
  P= fopen ("good.DAT", "r");   /*以只读模式打开*/
  if (!P)           /*文件没打开：流未创建*/.
   { printf ("Could not open file \n");
     exit (101);
   }
```

```
while(fscanf(p,"%d",&a)==1)        /*文件中的数据未读完*/
    printf("%d",a);                /*每次用a存放从文件中读取的数并打出，
return 0;                            下一次读数时a又被新的数覆盖*/
}
```

2). 拷贝文件.

```
int main(void)
{ FILE* in,*out;                   /*in：从已存在文件中读取数据放入流中*/
  int a;                           /*out：向用只写模式创的新文件中传数据的流*/
                                   /*a：存放从旧文件中读取的数据，输至新文件中*/
  printf("Running file copy\n");
  in = fopen("2008.DAT","r");      /*将流与通过只读打开的文件关联*/
  if(!in)
     {printf("Could not open input file\n"); exit(101);}
  out = fopen("2009.DAT","W");     /*以只写创建新文件并用写入的流与之关联*/
  if(!out)
     {printf("Could not open output file\n"); exit(102);}
  while(fscanf(in,"%d",&a)==1)     /*2008.DAT中还有未读的数据*/
     fprintf(out,"%d\n",a);        /*通过流out向2009.DAT写入数据*/
  if(fclose(out)==EOF)
     {printf("Could not close output file\n"); exit(201);}
  printf("File copy complete\n")
  return 0;
}
```

亚龙纸制品

NO.
Date . .

3). 追加文件.

```
#include <stdio.h>
#include <stdlib.h>
int main (void).
{ FILE* p;
  int a;
  p= fopen ("2008.DAT", "a");
  if (!p)
   {printf ("Could not add to input file\n"); exit (100);}.
  while ~~(fscanf (p, "%d", &a) ==1)~~ (scanf ("%d", &a)==1)
      fprintf ( p, "%d\n",a);
  return 0;
}.
```

③ 字符输入/输出

1)- 读字符函数. fgetc().

形参：通过以读方式打开的文件的流指针

返回：读到的一个字符(从流指针中).

eg: 从文件 2009.dat 中顺序读入字符并打在屏幕上.

```
#include <stdio.h>
#include <stdlib.h>.
int main (void).
{ FILE * p;
  char c;
   if((p=fopen ("2009.dat", "r")) ==NULL)
     {printf ("Could not open the file\n"); exit(100); }
```

```
C= fgetc (p);                /*从流中读取一个字符*/
while (C! =EOF)
   { putchar(c);
     C=fgetc(p);
   }
fclose(p);                   /*关闭流*/
return 0;
}
```

2). 写字符函数 fputc()

形参：字符型常量/变量/表达式 + 流指针

返回：向指定的文件中写字符，若写成功则返回已输出的字符，否则返回 EOF.(常量，∈stdio.h).

eg.
```
#include <stdio.h>
#include <stdlib.h>
int main (void).
{ FILE *p;
  char c;
  P= fopen ("2009.DAT", "W");
  if(!P)
    {printf ("cannot open the file\n"); exit (0);}
  C=getchar();               /*从键盘(终端)的指定流stdin中
                               读取一个字符 */.
  while (c!='\n')
    {fputc (c, p);           /*将字符加入"写入流中*/.
     c=getchar();
    }
  fclose(p);   return 0;
}
```

亚龙纸制品

NO.

Date . .

3). 读字符串函数 fgets() : fgets (string, n, p)

形参: { string: 字符串指针.
n: 整型, 表示从指定的文件读取 n-1 个字符存到 string 指向的空间.
p: 读取流.

返回: 存放字符串的首地址, 遇文件结束或出错返回 NULL.

4). 写字符串函数 fputs() : fputs(string, p)

形参: 将写入的字符串指针. 及流指针 (指向以 w 方式打开的文件).

返回: 写成功返回 0, 否则返回非 0 值.

说明: 用 fputs() 将字符串写到文件中时, 字符串中最后的结束符 '\0'
并不写入文件, 也不自动加换行符 '\n'.

NO.

Date · ·

十八. 二进制输入/输出

{文本：用字符的ASCII码值存储：结束：EOF、feof(p)=1 } 未结束feof(p)=0
二进制：数据的二进制形式. 结束：feof(p)=1.

r+：读/写 { 文件存在：打开 在起始位处标记.
不存在：返回NULL空指针.

w+：读/写 { 文件存在：删除原有内容，打开
不存在：创建new one.

a+：读/追加写 { 文件存在：打开. 在结尾处标记.
不存在：创建之.

现在有了六种打开文件的方式，如果打开的是文本文件，则六种模式为 r、w、a、r+，w+，a+，如果打开的是二进制文件，则六种模式为 rb、wb、ab、rb+，wb+，ab+.

数据块读写函数.

读：fread()：int fread (char* ptr, unsigned size, unsigned n, FILE* p).

- ptr：存放读取的数据的内存首地址.
- size：每个数据项的字节数：sizeof()
- n：数据的个数.
- p：指向与打开的文件关联的流.

写：fwrite()：int fwrite (char* ptr, unsigned size, unsigned n, FILE* p).

- ptr：从以此为首地址的内存中读数据输向文件.
- size：同fread()
- n：同fread()
- p：流指针.(wb/ab/wb+/ab)

NO.

Date

文件的定位函数:

一个文件被打开后，系统为该文件设置一个读写指针指示当前读写位置，当进行一次读写操作后，读/写指针自动发生变化。 C提供了改变文件中读/写指针的函数.

① void rewind(FILE *p);

将文件的读写指针移至文件开头，清除文件结束标志EOF.

② int fseek(p, offset, base)

1) { FILE *p 流指针
long offset: 偏移量.
int base: 基准位置.

2) 副作用: 将读/写指针移到以base指示的位置为基准、以offset为偏移量(-:向前; +:向后)的位置.

3) 返回: 成功:返回当前位置(int), 失败:返回-1(int).

stdio.h ⇒ { SEEK_SET 或 0: 文件首.
SEEK_CUR 或 1: 当前读写位置.
SEEK_END 或 2: 文件尾 } base.

③ long ftell(FILE *p); 返回当前读写位置.

总结:

文本文件 { 所有数据是可识别的图形化字符(ASCII)
数据的每行以一个换行符('\n')结尾.
文件结尾有一个文件结束符(EOF == -1).

二进制文件 { 数据存储在文件中和存储在内存中格式一样，不先转换成文本文件打开
没有行结束和行开始符.
不存在文件结束符EOF.

十九、命令行参数.

main() 要么不带参数(void)，要么带两个参数:

int main(int argc, char *argv[])

这两个参数代表用户需要传递给main的数据.

- argc 指定了字符串指针数组argv中元素个数，不由键盘输入
- argv 是一个字符串指针数组，其中的元素由键盘输入.

argv数组中第一个元素指向程序的文件名.(创建时命名的).
最后个元素是NULL指针. 表示数组末尾.
其余元素是指向用户输入的字符串的指针.

eg.
```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main(int argc, char* argv[])
{ printf("The number of arguments:%d\n", argc);
  printf("The name of the program:%s\n", argv[0]);
  for (int i=1; i<argc; i++)
      printf("User Value No. %d:%s\n", i, argv[i]);
  return 0;
}
```
/*运行时在命令中加上字符串(以空格分界，若想输出带空格的串，须将整体加双引号输入)*/

main函数是被操作系统调用的，返回值也是返回给操作系统，当main()完成后，控制权交还给操作系统.

亚龙纸制品

二十. 运算中的位运算.

1. 优先级表.

操作符	描述	例子	副作用	结合性	优先级
	标识符	data			
	常量	3.14159	N	—	16
	括号表达式	(a+b)			
[]	数组下标	arr[i]	N		
f()	函数调用	hanoi(x,y)	N/Y		
.	直接成员选择	day.hour	N	左→右	16
->	间接成员选择	ptr->hour	N		
++ --	后缀增量、减量	a++	Y		
++ --	前缀增量、减量	++a	Y		
sizeof	字节数	sizeof(int)	N		
~	按位取反	~a	N		
!	非	!a	N	右→左	15
+ -	一元加、减	+a	N		
&	地址	&a	N		
*	间接引用	*ptr	N		
()	强制类型转换	(int)ptr	N	右→左	14
* / %	乘、除、模	a*b	N	左→右	13
+ -	加法、减法	a+b	N	左→右	12
<< >>	左移位、右移位	a<<3	N	左→右	11
< <= > >=	比较	a<5	N	左→右	10
== !=	等、不等	a==b	N	左→右	9
&	按位与	a&b	N	左→右	8
^	按位异或	a^b	N	左→右	7

NO.

Date · ·

操作符	描述	例子	副作用	结合性	优先级
|	按位或	a|b	N	左→右	6
&&	逻辑与	a&&b	N	左→右	5
||	逻辑或	a||b	N	左→右	4
?:	条件	a?x:y	N	右→左	3
= += -=		a=5			
*= /= %=	赋值	a%=b	Y	右→左	2
>>= <<=		a>>=2			
&= ^= |=		a|=b			
,	逗号	a,b,c	N	左→右	1

2. 位运算符:

C包含可以在"位级别"操纵数据的运算符.

① 逻辑位运算符

- 按位与 &
- 按位或 |
- 按位异或 ^

} 2元

- 按位取反 ~ : 一元

} 使用前需确保固定大小的数据类型, 否则不知哪位与哪位相对并参与运算.

② 移位运算符

- 右移 >>: 左边为被移位数, 右边为待移动的位数. } 溢出的丢弃, 移入0或1
- 左移 <<: 左边为被移位数, 右边为待移动的位数. } 溢出的丢弃, 移入0.

③ 一个应用, 旋转:

把一个数的左(右)n位移到右(左)边形成一个新数.

eg:

NO.

Date

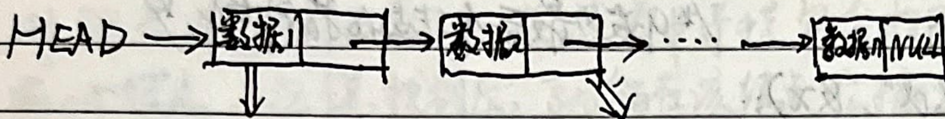
```c
/* Test driver for rotate left and right.
      Written by: Hugo
      Date:   10th Jan.
 */
#include <stdio.h>
#include <stdint.h>
uint16_t rotate16Left (uint16_t num, int n);
uint16_t rotate16Right (uint16_t num, int n);

int main (void)
{ uint16_t num = 0X2345;
   printf ("Original:  %#06x\n", num);
   printf ("Rotated Left: %#06x\n", rotate16Left (num, 4));
   printf ("Rotated Right ;%#06x\n", rotate16Right (num, 4));
}

uint16_t rotate16Right (uint16_t num, int n)
{ return ( (num << n) | (num >> 16 - n)); }

uint16_t rotate16Left (uint16_t num, int n)
{ return ((num >> n) | (num << 16-n)); }
```

NO.

Date · ·

二十一. 链表.

1. 结构.



数据域：存放数据元素 指针域：存放下一个结点元素的地址.

节点结构体.

```
struct 结构体名
{数据成员表;
 struct 结构体名* 指针变量名;
};
```

① HEAD: 头指针; NULL: 空指针. HEAD=NULL时为空表.

② 各数据元素间的前后关系是由各结点的指针域来指示的.

2. 申请:

(struct 结构体名 *) malloc(存储区字节数) …… free(p)

eg.

```
#include <stdio.h>
#include <stdlib.h>
struct node   /*定义结点类型*/
{int d;
 struct node *next;
};

int main (void)
{int x;
 struct node *head, *p, *q;
```

亚龙纸制品

NO.

Date . .

```
head = NULL            /*置链表头指针为空*/.
q = NULL;              /*q设为最后一个结点的指针域*/
scanf("%d", &x);
while(x>0)             /*以输入负数为终止链表的信号*/
{p = (struct node *) malloc (sizeof (struct node));  /*申请一个结点*/
  if (p == NULL)       /*内存满,未申请成功*/
    {printf("can't get memory!\n");
     exit(1);
    }
  p->d = x;            /*置当前结点的数据域为输入的正整数x*/
  p->next = NULL;      /*置当前结点的指针域为空*/
  if (head == NULL)
      head = p;        /*若当前链表为空,则将头指针指向第一个申请
                         到的结点*/
  else q->next = p;    /*否则将当前结点链接在最后*/.
  q = p;               /*置当前结点为链表最后一个结点*/
  scanf("%d", &x);     /*给下一个结点的数据域输数*/.
}
p = head;  /*从链表前结点开始,打印各结点的元素值,并删除*/.
while (p != NULL)
    {printf("%5d", p->d);
     q = p;
     p = p->next
     free(q);          /*释放删除结点的空间*/.
    }
 printf("\n");          return 0;
}
```

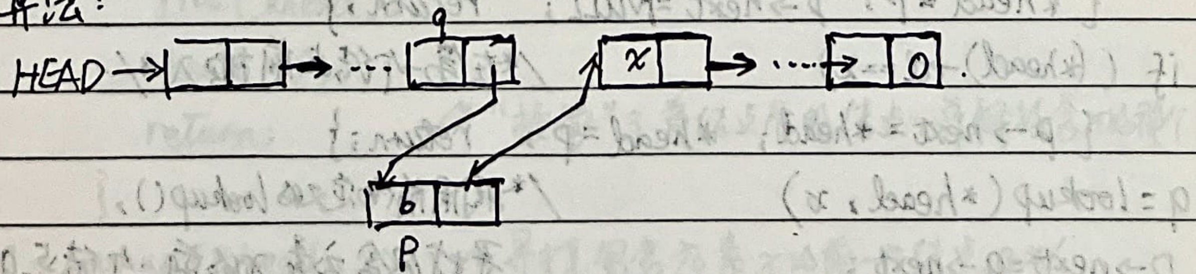
3. 在链表中查找指定元素.

对链表进行扫描查找，在链表中寻找包含指定元素值的前一个结点，这样，找到后，就能在该结点后插入新结点或删除该结点后的一个结点.

```
struct node                    /*定义结点类型*/.
{ET d;                         /*ET为数据元素类型名*/.
 struct node *next;
};

struct node *lookup(struct node *head, ET x)
{ struct node *p;              /*结构体指针函数:
  p=head;                        返回node型结构体的一个地址*/.
  while((p->next != NULL) && (p->next->d != x))
        p=p->next;
  return (p);                  /*一但找到了x,跳出循环,通过lookup
}                                函数返回之*/.
```

4. 插入一个新结点.

算法:



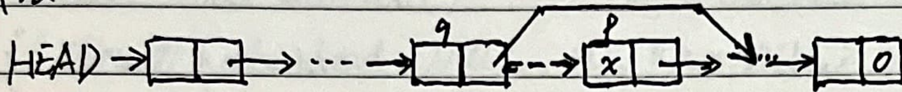
代码:

亚龙纸制品

```c
#include <stdio.h>
#include <stdlib.h>
struct node
{ET d;
  struct node *next;
};

void putin(struct node **head, ET x, ET b)
{ struct node *p, *q;
  /* 将头指针的形参定义为二维指针的意义是实现地址传递，
     即当函数被调用后，被处理过的链表(*head指向的)已带有
     被插入的结点；若head不是指针的指针，则函数体外该链表没有变化*/
p=(struct node *) malloc(sizeof(struct node));     /*申请一个新结点*/
if (p==NULL)                                        /*未申请成功*/
   {printf("can't get memory!\n");
    exit(1);
   }
p->d=b;                                    /*置结点的数据域为b*/
if (*head==NULL)                           /*链表为空*/
   { *head=p; p->next=NULL;  return;}
if ((*head)->d==x)                         /*在第一个结点前插入*/
   { p->next=*head; *head=p;  return;}
q=lookup(*head, x)                         /*调用前面定义的lookup(),
p->next=q->next;                             寻找包含元素x的前一个结点q */
q->next=p;                                 /*结点p插入到结点q之后 */
return;
}
```

5. 删除一个结点.

算法:



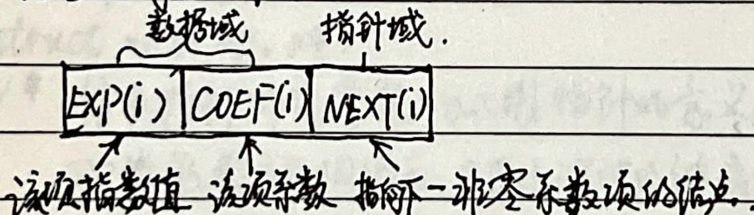
代码:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{ET d;
 struct node *next;
};

void delete (struct node **head, ET x)
    /*在头指针*head的链表中删除包含元素x的结点*/
{ struct node *p, *q;
  if (*head==NULL) /*空表*/
     {printf("This is an empty list.\n");  return;}
  if ((*head)->d ==x) /*首结点就是欲删除的结点*/
     { p=(*head)->next;
       free(*head);
       *head =p;
       return;        /*"接管"了首结点后的结点,并释放含x的那个首结点*/
     }
  q=lookup(*head, x)  /*寻找包含元素x的前一个结点q*/
     if (q->next ==NULL) /*链表中没有包含元素x的结点*/
        {printf(" node not in the list.\n"); return;}
```

```c
    p=q->next;
    q->next=p->next;
    free(p);
    return;
}
```

6. 用链表进行多项式的表示与运算.

设多项式为 $P_n(x)=a_n x^n+a_{n-1}x^{n-1}+\cdots+a_1 x+a_0$.




```c
#include <stdio.h>
#include <stdlib.h>.
struct node
{ int exp;
  double coef;
  struct node *next;
};
```

非零系数项的多项式:

$P_m(x)=a_m x^{e_m}+a_{m-1}x^{e_{m-1}}+\cdots+a_1 x^{e_1}$

其中 $a_k\neq 0\ (k=1,2,\ldots,m)$, $e_m>e_{m-1}>\cdots>e_1\geq 0$.

N0.

Date . .

① 多项式链表的生成

```
struct node *creat()   /*返回生成链表的头指针*/
{ struct node *head=NULL, *p, *k=NULL;
  int e;
  double a;
  printf("Enter exp and coef:");
  scanf("%d%lf", &e, &a);
  while (e>=0)          /*以输入一个负数为终止信号*/
   { p=(struct node *) malloc(sizeof(struct node));
     p->exp=e; p->coef=a; p->next=NULL;
     if (head==NULL)  head=p;
     else    k->next=p;
     k=p;
     printf("Enter exp and coef:");
     scanf("%d%lf", &e, &a);
   }
  return head;
}
```

② 多项式链表的释放.

```
void throw(struct node *head)    /*以需释放的多项式链表为形参*/
{ struct node *p, *k=head;
  while(k!=NULL).
    {p=k->next;
     free(k);
     k=p;
    }
}
```

NO.

Date . .

③ 多项式的输出

```
void output(struct node *head)
{ struct node *p=head;        /*给结构体指针初始化*/
  while(p!=NULL)
    { printf("%d,%lf\n", p->exp, p->coef);
      p=p->next;
    }
}
```

④ 多项式相加.

算法：假设多项式 $A_m(x)$ 与 $B_n(x)$ 已用链表表示，头指针分别为 AH 和 BH，多项式 C(x) 用另一链表表示，头指针为 CH.

从 $A_m(x)$ 和 $B_n(x)$ 的第一个结点开始检测：

1) 若两个多项式中对应结点的指数值相等，则将它们的系数相加，若和不为零，则形成新结点的一个数据域，然后再检测两链表中的下一个结点.

2) 若两个多项式中对应结点的指数值不等，则将指数值大的那个结点中的指数与系数定为新结点中的相应量，链入 CH 链表的表尾. 然后再检测指数值小的链表中的当前结点与指数值大的链表中的下一个结点.

代码：

```
struct node *addpoly(struct node *ah, struct node *bh)
        /*形参是两个被加多项式*/
{ struct node *k=NULL, *p, *m, *n, *ch=NULL;
  int e; double d;
  m=ah; n=bh;
```

NO.

Date · ·

```
while (m!=NULL && n!=NULL)
  { if (m->exp==n->exp)
      { d=m->coef+n->coef;
        e=m->exp;
        m=m->next; n=n->next;
      }
    elseif (m->exp > n->exp)
      { d=m->coef, e=m->exp;
        m=m->next;
      }
    else
      { d=n->coef; e=n->exp;
        n=n->next;
      }
    if (d!=0)
      { p=(struct node *)malloc(sizeof(struct node));
        p->exp=e; p->coef=d;
        p->next=NULL;
        if(ch==NULL) ch=p;
        else k->next=p;
        k=p;
      }
  }
```

亚龙纸制品

No.
Date

```
while(m!=NULL)
{ p=(struct node*) malloc (sizeof(struct node));
  p->exp=m->exp; p->coef=m->coef; p->next=NULL;
  m=m->next;
  if (ch==NULL) ch=p;
  else k->next=p;
  k=p;
}
while(n!=NULL)
{ p=(struct node*) malloc (sizeof(struct node));
  p->exp=n->exp; p->coef=n->coef; p->next=NULL;
  n=n->next;
  if(ch==NULL) ch=p;
  else k->next=p;
  k=p;
}
return ch;
}
// 主函数
void main()
{ struct node *ah, *bh, *ch
  ah=creat();     bh=creat();
  printf("ah=");  output(ah);
  printf("bh=");  output(bh);
  ch=addpoly(ah,bh);
  printf("ch=");  output(ch);
  throw(ch); throw(bh); throw(ah);
}
```