目录

第一章　C 语言概述

一、C 语言特点

- 语言简洁、紧凑,使用方便、灵活。 32 个关键字、9 种控制语句,程序形式自由
- 运算符丰富。34 种运算符
- 数据类型丰富,具有现代语言的各种数据结构。
- 具有结构化的控制语句，是完全模块化和结构化的语言。
- 语法限制不太严格,程序设计自由度大。
- 允许直接访问物理地址,能进行位操作,能实现汇编语言的大部分功能,可直接对硬件进行操作。兼有高级和低级语言的特点 。
- 目标代码质量高,程序执行效率高。只比汇编程序生成的目标代码效率低 10%-20%。
- 程序可移植性好(与汇编语言比)。基本上不做修改就能用于各种型号的计算机和各种操作系统。

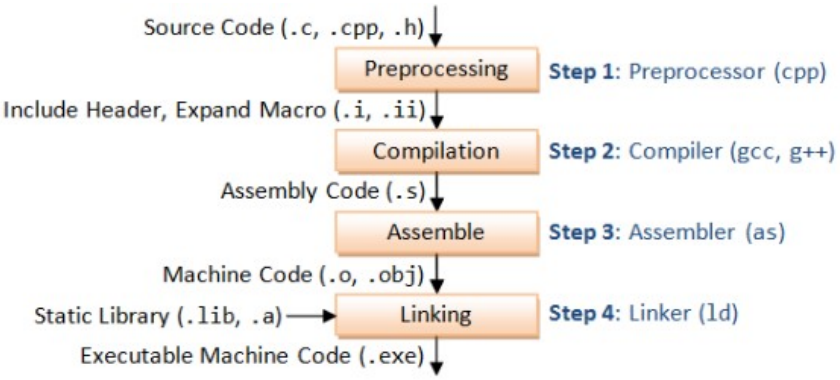
二、运行 C 程序的步骤

第一步：预处理：将所有的#include 头文件以及宏定义替换成其真正的内容

第二步：编译：不是指程序从源文件到二进制程序的全部过程，而是指将经过预处理之后的程序转换成特定汇编代码(assembly code)的过程

第三步：汇编：将上一步的汇编代码转换成机器码(machine code)

第四步：链接：将多个目标文以及所需的库文件(.so 等)链接成最终的可执行文件



第二章　算法的基本概念

2.1 算法的基本概念

一、程序设计概述

1、一个程序应该包含的内容：数据结构（对数据的描述）、算法（对操作的描述）

2、完整的程序设计：数据结构＋算法＋程序设计方法＋语言工具

二、算法的基本概念

1、算法分类：数值运算算法、非数值运算算法

2、算法的性质（面试问过）

- 有穷性：包含有限的操作步骤。
- 确定性：算法中的每一个步骤都应当是确定的。
- 有零个或多个输入：输入是指在执行算法时需要从外界取得必要的信息。
- 有一个或多个输出：算法的目的是为了求解，“解”就是输出。
- 有效性：算法中的每一个步骤都应当能有效地执行，并得到确定的结果 。

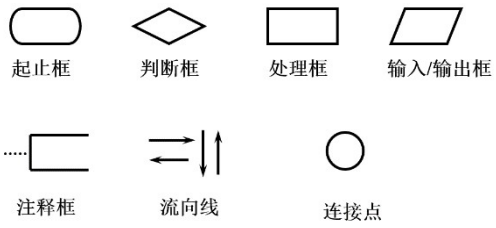
三、算法的表示

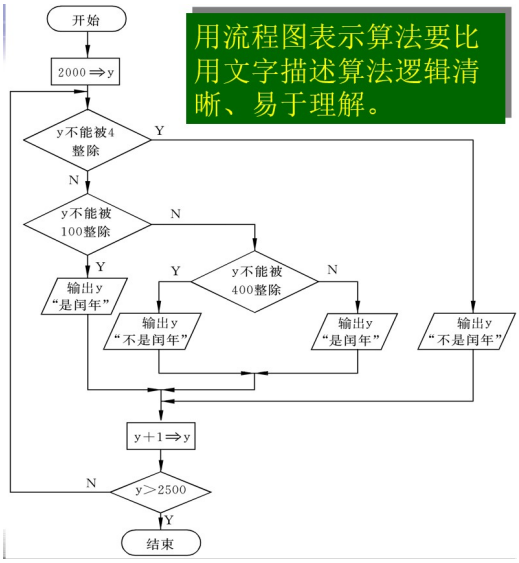
1、用自然语言表示算法（只能用于简单算法）

- 缺点：容易出现歧义；分支、循环不好描述

2、用流程图表示算法（传统流程图）

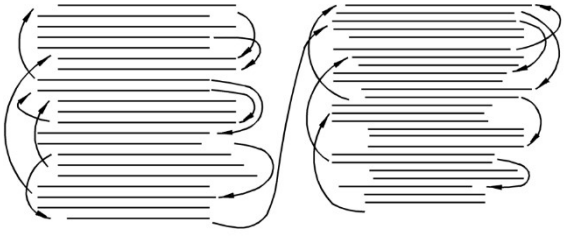
a)　概念





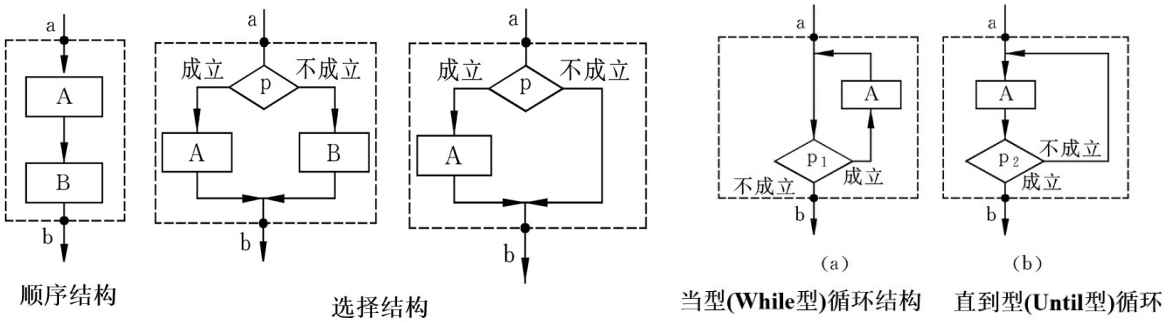
- 缺点：传统流程图用流程线指出各框的执行顺序，对流程线的使用没有严格限制。因此，使用者可以毫不受限制地使流程随意地转向，使流程图变得毫无规律，阅读者要花很大精力去追踪流程，使人难以理解算法的逻辑

传统流程图的流程可以是：



这种如同乱麻一样的算法称为BS型算法，意为一碗面条(A Bowl of Spaghetti)，乱无头绪。

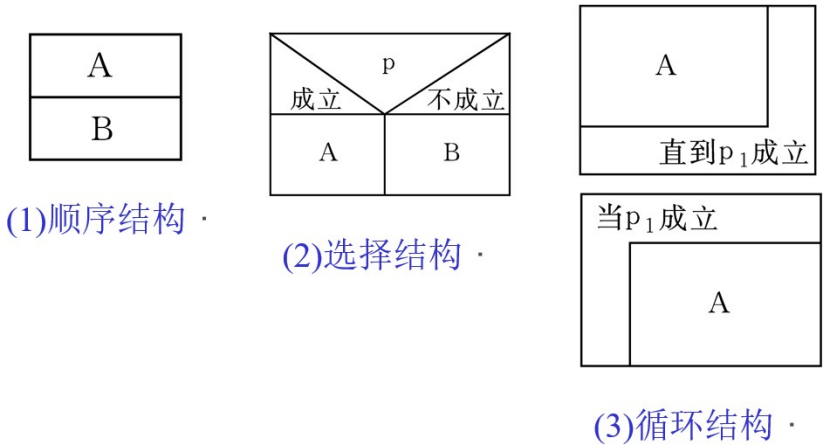
a) 基本结构的流程图规范（顺序结构、选择结构、循环结构）



顺序结构　　选择结构　　当型(While型)循环结构　　直到型(Until型)循环

3、用改进流程图表示算法——N-S 流程图

（去掉了流程线，全部算法写在一个矩形框内）



(1)顺序结构 ·　　(2)选择结构 ·　　(3)循环结构 ·

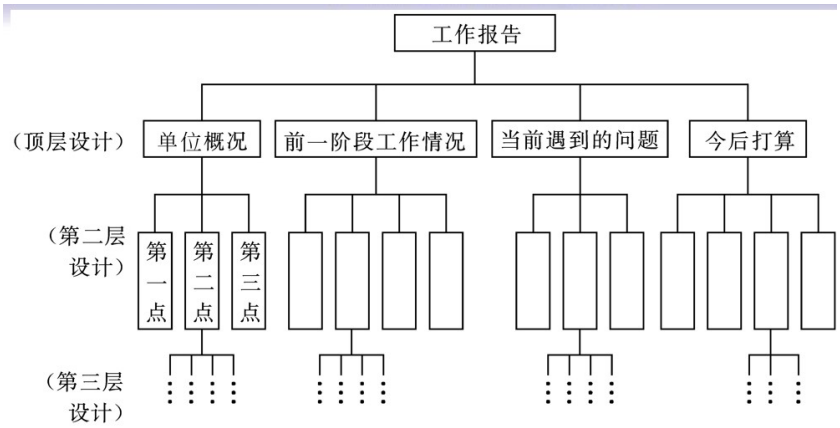
4、用伪代码表示算法

5、用计算机语言表示算法

2.2 结构化程序的设计方法

一、设计方法——自顶向下，逐步细化

1、内容



用这种方法逐步分解，直到作者认为可以直接将各小段表达为文字语句为止。这种方法就叫 做“自顶向下，逐步细化”。

2、优点

考虑周全，结构清晰，层次分明，作者容易写，读者容易看。如果发现某一部分中有一段内容不妥，需要修改，只需找出该部分修改有关段落即可，与其它部分无关。我们提倡用这种方法设计程序。这就是用工程的方法设计程序。

3、模块设计方法

- 在拿到一个程序模块以后，根据程序模块的功能将它划分为若干个子模块，如果这些子模块的规模还嫌大，还再可以划分为更小的模块。这个过程采用自顶向下方法来实现
- 子模块一般不超过 50 行。
- 划分子模块时应注意模块的独立性，即：使一个模块完成一项功能，耦合性愈少愈好

第三章　数据类型、运算符、表达式

3.1 数据类型

一、常量与变量

1、常量：程序运行过程，值不能被改变的量

a) 常量的数据类型：整型、实型、字符型、字符串

b) 常量的定义方法（无法用再用赋值语句赋值）

- const 关键字定义
- #define 定义

2、变量：反之

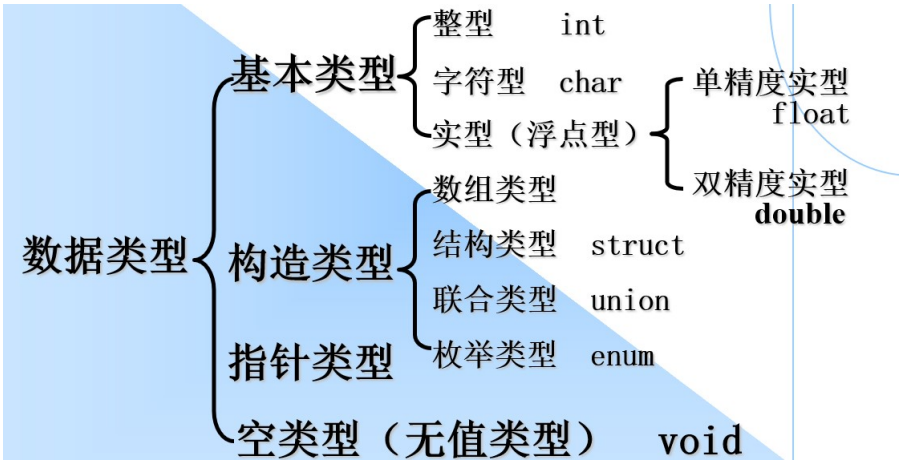
a) 变量命名规则

- 只能由字母、数字和下划线三种字符组成，且第一个字母不能是数字
- C 语言区分大小写

b) 变量名的本质：对应一个地址，在对程序编译连接时由编译系统给每一个变量名分配对应的内存地址。从变量中取值，实际上是通过变量名找到相应的内存地址，从该存储单元中读取数据。

- 注意：初始化不是在编译阶段完成的而是在程序运行时执行本函数时赋初值的，相当于有一个赋值语句

二、数据类型概述



三、整型数据

1、整型数据在内存中的存放格式：补码

2、整型常量（讲表示方法）

a) 区分不同进制

- 十进制：正常
- 八进制：0 开头。如 0123（等于 83）、-011（等于-9）
- 十六进制：0x 开头。如 0x123（等于 291）、-0x12（等于-10）

b) 区分不同数据类型

- unsigned int 型：整常量后面加字母 u 或 U。如 12345u，但是如果写成-12345u，

则先将-12345 转换成其补码 53191，然后按无符号数存储

- long 型：整常量后面加一个字母 l 或 L

3、整型变量

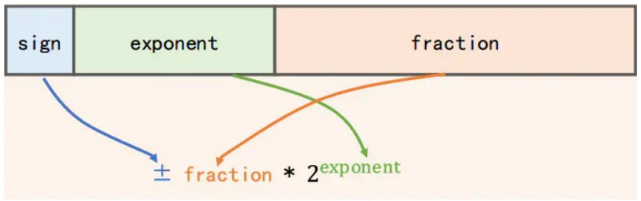
a) 分类

类型	类型说明符	长度	数的范围
基本型	int	2字节	-2^{15}~$2^{15}-1$
短整型	short	2字节	-2^{15}~$2^{15}-1$
长整型	long	4字节	-2^{31}~$2^{31}-1$
无符号整型	unsigned	2字节	0～$2^{16}-1$
无符号短整型	unsigned short	2字节	0～$2^{16}-1$
无符号长整型	unsigned long	4字节	0～$2^{32}-1$

四、浮点型数据

1、在内存中的存放格式：指数形式（4 字节）

- 数符、尾数、阶码



2、浮点型常量（讲表示方法）

a) 小数表示法：0.123

b) 指数表示法：3e-3

- 规范化的指数形式：在字母 e（或 E）之前的小数部分中，小数点有且仅有一位非零的数字

3、浮点型变量

a) 分类（编译器默认双精度）

类型	位数	数的范围	有效数字
float	32	10^{-37}～10^{38}	6~7 位
double型	64	10^{-307}~10^{308}	15~16位
long double	128	10^{-4931}~10^{4932}	18~19位

- 注意浮点型数据的有效数字，超过有效位数后，后面的数据都是无意义的。所以应当避免将一个很大的数和一个很小的数直接相加或相减，否则就会“丢失”小的数

五、字符型数据

1、在内存中的存储方式：存放对应的 ASCII 码

2、字符常量

a) 规范

- 用单引号包含的一个字符是字符型常量
- 只能包含一个字符

b) 特殊字符——转义字符，如\n

3、字符变量（char，1 字节）

六、字符串型数据

1、存放方式：除了存放每一位的 ASCII 码之外，末尾还有\0 作为字符串的结束标志

2、字符串常量：双引号

- 注意：‘ a ’和” a ”不同

3、字符串变量

3.2 运算符、表达式

一、基本概念

1、不同数据类型之间的运算

可以运算，但是在进行运算时，不同类型的数据要先转换成同一类型,然后进行运算（类型转换是由系统自动进行的）

2、运算符概述

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

二、算术运算符和算术表达式

1、除法运算符的比较

- / 除法
- % 求模（即余数，两侧均应为整型）

2、强制类型转换运算符

有两种类型转换

- 一种：在运算时不必用户指定，系统自动进行的类型转换，如 3+6.5
- 另一种: 是强制类型转换。当自动类型转换不能实现目的时，可以用强制类型转换。

3、自增运算符：i++与++i 的区别

三、赋值运算符

不同数据类型赋值时的处理方式

1、浮点型赋给整型时：舍弃浮点数的小数部分

2、整型数据赋给浮点型时：数值不变，但以浮点数形式存储到变量中

3、double 赋给 float 时：截取其前面 7 位有效数字，存放到 float 变量的存储单元（4 个字节）中。但应注意数值范围不能溢出。

4、字符型数据赋给整型变量时，由于字符只占 1 个字节，而整型变量为 2 个字节，因此将字符数据（8 个二进位）放到整型变量存储单元的低 8 位中

…

第四章　简单的 C 程序设计

一、基本概念

1、C 语句的分类：

- 控制语句：if、for、return 等
- 函数调用语句
- 表达式语句
- 空语句：一个分号
- 复合语句：由大括号括起来的多条语句

二、数据输入输出

1、概念：所谓输入输出是对于计算机而言的

- 输出：从计算机向外部输出设备(显示器,打印机)输出数据
- 输入：从输入设备(键盘,鼠标,扫描仪)向计算机输入数据.

2、输入输出语句

- 注意：C 语言本身不提供输入输出语句，他们都是由 C 函数库中的函数来实现的

字符输入函数：getchar 字符输出函数:putchar
格式输入函数：scanf 格式输出函数: printf
字符串输入函数:gets 字数串输出函数:puts

第五章　选择结构程序设计

一、关系、逻辑运算符（略）

优先级：算术运算符>关系运算符>条件运算符>赋值运算符

二、选择语句之——if 语句

1、if 语句三种基本形式（略）

2、else 的匹配规则

- Else 总是与它上面最近的，统一复合语句中的，未配对的 if 语句配对

```
例：                          例：
If()                          If()
      if()  语句1                   {if()  语句1}
else                          else
      if()  语句2                   if()  语句2
else    语句3                 else    语句3
```

当if和else数目不同时，可以加花括号来确定配对关系。

三、选择语句之——switch 语句

```
switch(grade)
{   case 'A' : printf ("85~100\n");
    case 'B' : printf ("70~84\n");
    case 'C' : printf ("60~69\n");
    case 'D' : printf ("<60\n");
    default  : printf ("error\n");
 }
```

第六章　循环结构程序设计

一、循环结构之——goto

1、用法

```
void main( )
    {       int i,  sum=0;
            i=1;
loop:    if(i<=100)
        { sum=sum+i;
          i++;
          goto loop;
        }
                printf("%d\\n" , sum);
    }
```

2、评价：一般不用（可读性差）

二、循环结构之——while

1、while

2、do while

- 通常情况下，二者等价
- 仅当但是如果 while 后面的表达式一开始就为假时，两种循环的结果是不同的。

三、循环结构之——for

四、循环相关的命令

1、break

2、continue

第七章　数组

一、一维数组、二维数组、多维数组（略）

二、字符数组

1、定义方式

- 字符串常量定义：char c [] ={"I am happy"};
- 字符定义：char c[10]={ 'I', ' ','a', 'm','', 'h', 'a', 'p', 'p', 'y'};

一般字符数组剩下的元素都填充\0，但是上面字符定义就没有（因为没空间了），这样会导致输出字符串时出现问题，因为输出的结束是根据\0 决定的

2、字符串处理函数（略）

第八章　函数

8.1 参数、变量的基本概念

一、基本概念

1、函数分类

a) 从用户使用角度：标准函数（库函数）、用户自定义的函数

b) 从函数形式：无参函数、有参函数

2、关于 main 函数

- C程序的执行是从 main 函数开始的，如果在 main 函数中调用其他函数，在调用后流程返回到 main 函数，在 main 函数中结束整个程序的运行
- 函数间可以互相调用，但不能调用 main 函数。main 函数是系统调用的

二、函数的参数及数据传递

1、主调函数和被调函数间数据传递的方式

- 形参和实参：单向的“值传递”
- 返回值
- 全局变量

2、形参和实参

a) 形式参数：函数声明时后面括号中的变量

- 未出现函数调用时，它们并不占内存中的存储单元。只有在发生函数调用时，函数 max 中的形参才被分配内存单元。
- 在调用结束后，形参所占的内存单元也被释放

b) 实际参数：主调函数中调用一个函数时，括号中写的具体内容

- 在调用时将实参的值赋给形参

3、数组作为参数时的数据传递过程

a) 数组元素作为参数——单向的“值传递”

b) 数组名作为参数——“地址传递”，相当于指针

三、变量

1、变量的分类

a) 按作用域分

	局部变量	全局变量
定义	函数内部定义的变量	函数之外定义的变量
有效范围	当前函数	从定义变量的位置开始到本源文件结束
对应的存储区	动态存储区	静态存储区

- 形参就是局部变量

b) 变量按存在的时间分

- 静态存储方式：在程序运行期间由系统分配固定的存储空间的方式
- 动态存储方式：在程序运行期间根据需要进行动态的分配存储空间的方式

存储空间的划分：程序区、静态存储区、动态存储区

2、变量的存储类型

a) 自动变量 auto：（默认是它，如形参、在函数中定义的变量）

- 调用函数时系统会给它们分配存储空间，调用结束时就自动释放

b) 静态变量 static

- 声明局部变量
  - 用它定义的局部变量，在函数调用结束后不会被释放（尽管如此，其他函数并不能引用它）
  - 只赋一次初值：编译时赋上设定的初值。以后每次调用函数时不再重新赋初值而只是保留上次函数调用结束时的值
  - 如果没有给处初值，编译时自动赋初值 0 （数值型变量）或'\0'（字符变量）这一点和 auto 变量有很大的不同。
- 声明全局变量：该变量只限于被本文件引用，而不能被其他文件引用

c) 寄存器变量 register（对于需要频繁使用的变量，可以这么定义）

- 变量不再存放在内存中，而是直接放在寄存器中
  （CPU 对寄存器的存取速度远高于对内存的存取速度）

d) Extern（用于全局变量的额外声明）

- 一个文件内声明：可以扩展全局变量的作用范围

```c
#include<stdio.h>

int func();

int main()
{
    func(); //1
    printf("%d",num); //2
    return 0;
}

int num = 3;

int func()
{
    printf("%d\n",num);
}
```

```c
#include<stdio.h>

int func();

int main()
{
    func(); //1
    extern int num;
    printf("%d",num); //2
    return 0;
}

int num = 3;

int func()
{
    printf("%d\n",num);
}
```

- 多文件程序中声明：将全局变量的作用域扩展到其他文件

mian.c

```c
#include<stdio.h>

int main()
{
    extern int num;
    printf("%d",num);
    return 0;
}
```

b.c

```c
#include<stdio.h>

void func()
{
    int num = 5;
    printf("fun in a.c");
}
```

3、变量的声明类型及特点

- 定义型声明：需要建立存储空间
- 引用型声明：不需要（extern）

8.2 函数的相关概念

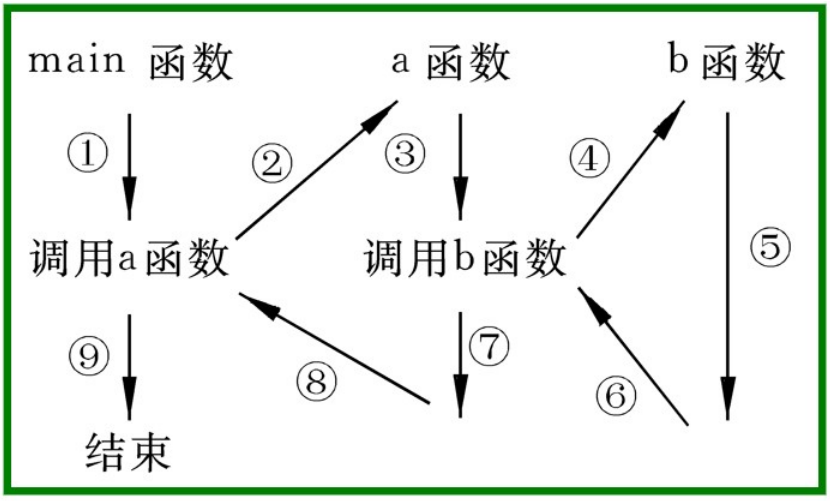
一、定义和声明的区别

- 定义是指对函数功能的确立，包括指定函数名，函数值类型、形参及其类型、函数体等，它是一个完整的、独立的函数单位。
- 声明的作用则是把函数的名字、函数类型以及形参的类型、个数和顺序**通知编译系统**，以便在调用该函数时系统按此进行对照检查。

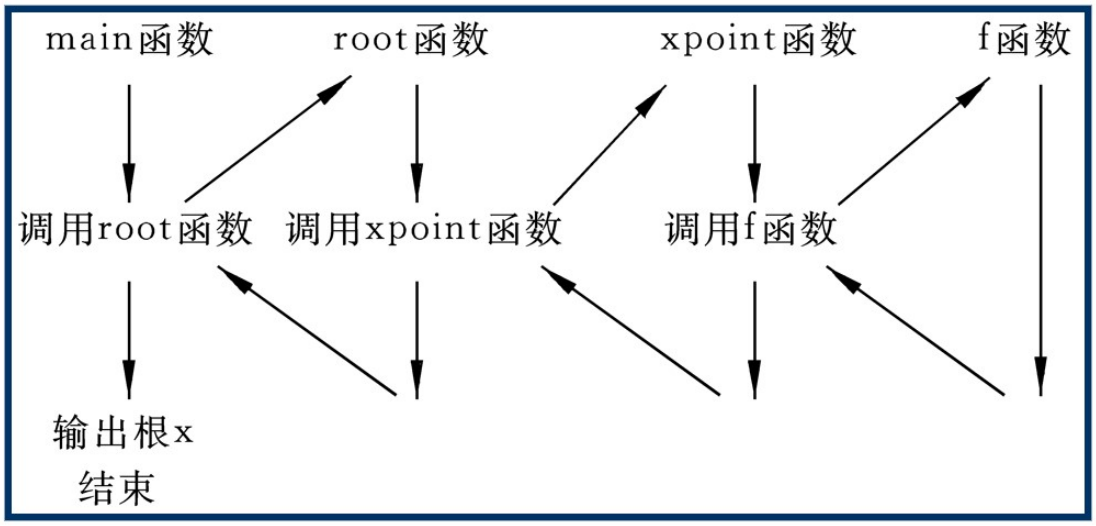
二、函数的调用

1、函数的一般调用（略）

2、函数的嵌套调用



3、函数的递归调用



三、内部函数和外部函数——按能否被其他文件调用

- 内部函数：加上 static
- 外部函数：加上 extern（默认是这种），注意再另一个调用它的文件中要用 extern 声明

第九章　预处理命令

一、概述

C 语言有三种预处理方式：宏定义、文件包含、条件编译

二、宏定义#define

1、宏定义类型

a) 不带参数的宏定义： # define PI 3.1415926

- 在预编译时将宏名替换成字符串，不作正确性检查，不分配空间

b) 带参数的宏定义：#define　S(a,b)　a*b。正文 area=S(3,2);

- 不是进行简单的字符串替换，还要进行参数替换
- 宏定义不是 C 语句，不必在行末加分号。如果加了分号则会连分号一起进行置换

2、作用域：定义命令之后到本源文件结束

- 可以用 # undef 命令终止宏定义的作用域

3、#define 和 typedef 的区别

- #define 是在预编译时处理的，它只能作简单的字符串替换
- typedef 是在编译时处理的，并不是作简单的字符串替换，而是采用如同定义变量的方法那样来声明一个类型

三、文件包含#include

在编译时并不是分别对两个文件分别进行编译，然后再将它们的目标程序连接的，而是在经过编译预处理后将头文件 format.h 包含到主文件中，得到一个新的源程序，然后对这个文件进行编译，得到一个目标（.obj）文件。被包含的文件成为新的源文件的一部分，而单独生成目标文件。

被包含文件（file2.h）与其所在的文件（即用#include 命令的源文件 file2.c），在预编译后已成为同一个文件(而不是两个文件)。因此，如果 file2.h 中有全局静态变量，它也在 file1.h 文件中有效，不必用 extern 声明

四、条件编译

```
(3) # if 表达式
      程序段 1
    # else
      程序段 2
    # endif
```

```
(1) # ifdef 标识符
      程序段 1
    # else
      程序段 2
    # endif
```

```
(2) # ifndef 标识符
      程序段 1
    # else
      程序段 2
    # endif
```

第十章　指针

一、void 类型的指针

二、指针与多维数组

对于 int a［3］［4］

表 示 形 式	含义	地 址
a	二维数组名，指向一维数组 a[0]，即0行首地址	2000
a[0]， *(a+0)， *a	0行0列元素地址	2000
a+1，&a［1］	1行首地址	2008
a［1］,*(a+1)	1行0列元素a[1][0]的地址	2008
A[1]+2, *(a+1)+2, &a[1][2]	1行2列元素a[1][2] 的地址	2012
*(a[1]+2), *(*(a+1)+2), a[1][2]	1行2列元素a［1］［2］的值	元素值为13

三、总结

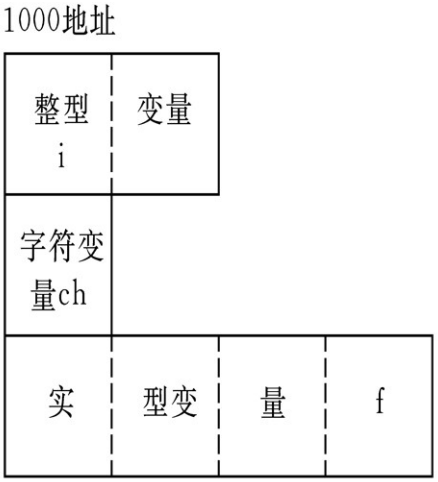
定义	含义
int i;	定义整型变量 i
int *p;	p 为指向整型数据的指针变量
int a[n];	定义整型数组 a，它有 n 个元素
int *p[n];	定义指针数组 p，它由 n 个指向整型数据的指针元素组成
int (*p)[n];	p 为指向含 n 个元素的一维数组的指针变量
int f();	f 为带回整型函数值的函数
int *p();	p 为带回一个指针的函数，该指针指向整型数据
int (*p)();	p 为指向函数的指针，该函数返回一个整型值
int **p;	p 是一个指针变量，它指向一个指向整型数据的指针变量

第十一章　结构体与共用体

一、结构体（略）

二、共用体 union

1、 概念：使几个不同的变量共占同一段内存的结构，如下



2、共用体变量的引用方式：不能引用共用体变量，而只能引用共用体变量中的成员

3、数据特点

- 同一个内存段可以用来存放几种不同类型的成员，但在每一瞬时只能存放其中一种，而不是同时存放几种
- 共用体变量中起作用的成员是最后一次存放的成员，在存入一个新的成员后原有的成员就失去作用。
- 共用体变量的地址和它的各成员的地址都是同一地址。
- 不能对共用体变量名赋值，也不能企图引用变量名来得到一个值，又不能在定义共用体变量时对它初始化。
- 不能把共用体变量作为函数参数，也不能使函数带回共用体变量，但可以使用指向共用体变量的指针
- 共用体类型可以出现在结构体类型定义中，也可以定义共用体数组。反之，结构体也可以出现在共用体类型定义中，数组也可以作为共用体的成员。

三、枚举类型 enum

第十三章　文件

一、文件的基本概念

1、定义：存储在外部介质(如磁盘磁带)上数据的集合

- 操作系统是以文件为单位对数据进行管理的

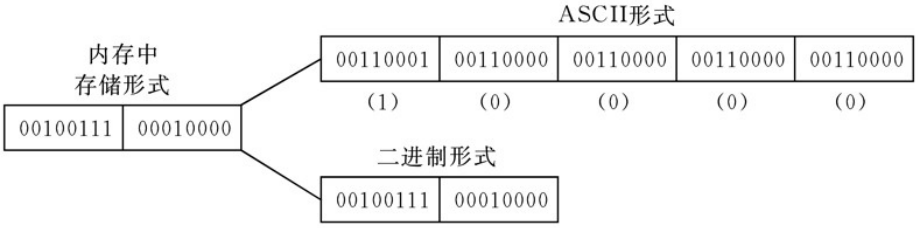
2、ASCII 文件和二进制文件的区别

a) ASCII 文件(文本文件)：每一个字节放一个 ASCII 代码

- 优点：便于对字符进行逐个处理，也便于输出字符
- 缺点：一般占存储空间较多，而且要花费转换时间

b) 二进制文件：把内存中的数据按其在内存中的存储形式原样输出到磁盘上存放

-



3、C 语言对文件的处理方法

- 缓冲文件系统（高级磁盘输入输出)：系统自动地在内存区为每一个正在使用的文件开辟一个缓冲区。
- 非缓冲文件系统（低级输入输出系统)：系统不自动开辟确定大小的缓冲区，而由程序为每个文件设定缓冲区。

二、文件的操作

1、文件类型指针

2、各种函数（略）