**Easy搞定**

全国计算机等级考试二级

C语言

# 考试要求

## 1、报名条件

1、进入 计算机等级考试报名系统 （以北京为例）

网址为：<http://ncre.bjeea.cn/web/login.jsp>

2、阅读和同意报名协议后，进入当前批次计算机等级考试报名资料的填写，按要求进行详细资料的填写，选择合适自己的考试科目和等级，报考条件

3、报考时，要选择合适自己的考点，最好离住址较近，方便现场确认和考试。填写完毕后，在：我保证以上信息是真实和正确的前面打勾，检查报考信息无误后，点击提交报考信息

## 2、考试要求

**1、科目：**语言程序设计类（C、C++、Java、Visual Basic、Web、Python）、数据库程序设计类（Access、MySQL）、办公软件高级应用（MS Office 高级应用）共九个科目。

于九个科目中选择一个参加考试并过关即可。

**2、形式：**完全采取上机考试形式。各科上机考试时间均为 120 分钟，满分 100 分。

**3、获证条件：**总分不低于 60 分。

## 3、考题格式

**C语言程序设计：**

（1）单项选择题，40题，40分（含公共基础知识部分10分）；

（2）程序填空题，2~3个空，18分；

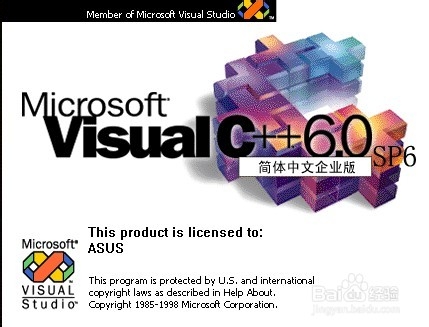
（3）程序改错题，2~3处错误，18分；

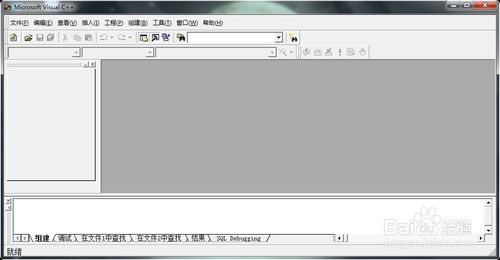
（4）程序设计题，1题，24分。

# 开发环境

Visual C++6. 0集成：

Microsoft Visual C++6.0作为新手C语言编程软件，被大家广为使用，然而许多人为拷贝来的C++6.0安装包如何安装感到苦恼，因此许多同学都是以安装失败，安装不成功而告终。接下来我就介绍下如何快速安装Microsoft Visual C++6.0。

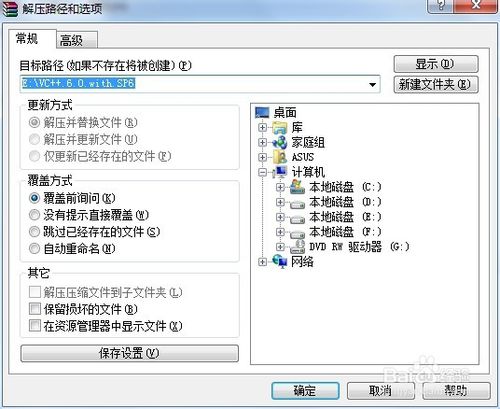
[](http://jingyan.baidu.com/album/a501d80ce2a7ecec630f5eb8.html?picindex=1)

[](http://jingyan.baidu.com/album/a501d80ce2a7ecec630f5eb8.html?picindex=2)

## 1、解压压缩包

首先将拷贝来的压缩包解压，【右击文件】选择解压文件，选择路径进行解压。

**注意事项：**不用管是否是光盘映像文件，直接点击右击解压即可。

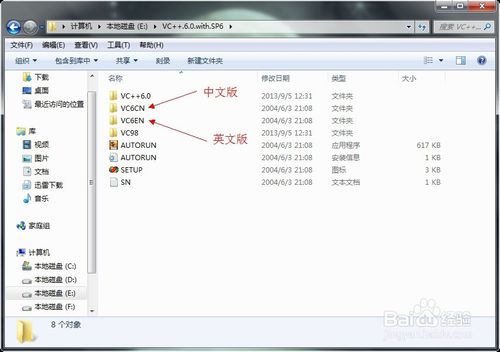
[](http://jingyan.baidu.com/album/a501d80ce2a7ecec630f5eb8.html?picindex=3)

[](http://jingyan.baidu.com/album/a501d80ce2a7ecec630f5eb8.html?picindex=16)

## 2、选择【VC 6.0 CN】

**注意事项：**VC6CN为中文版

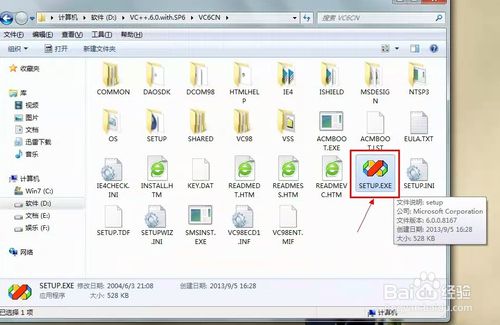
      VC6EN为英文版

[](http://jingyan.baidu.com/album/a501d80ce2a7ecec630f5eb8.html?picindex=4)

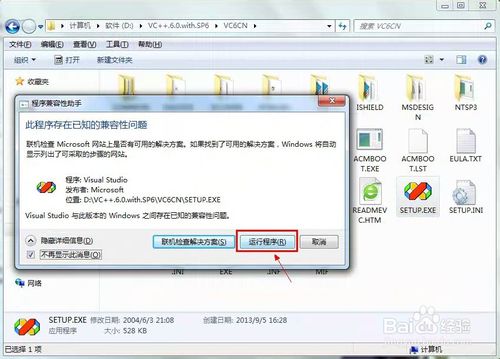
## 3、安装程序

在诸多软件中找到可运行.exe软件

如下图中的【SETUP.EXE】

[](http://jingyan.baidu.com/album/a501d80ce2a7ecec630f5eb8.html?picindex=5)

 如果出现以下窗口，表示计算机与此软件兼容性存在问题，不过不必担心，正常的笔记本都会出现此种情况，点击【运行程序】

[](http://jingyan.baidu.com/album/a501d80ce2a7ecec630f5eb8.html?picindex=6)

 继续点击下一步。

[](http://jingyan.baidu.com/album/a501d80ce2a7ecec630f5eb8.html?picindex=7)

 认真阅读完协议后，点击【接受协议】

点击下一步。

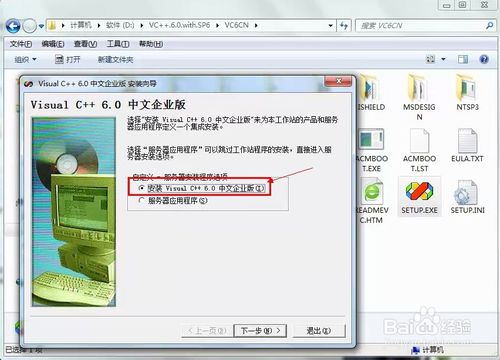
[](http://jingyan.baidu.com/album/a501d80ce2a7ecec630f5eb8.html?picindex=8)

 此处姓名与公司名字可以填写可以不填，随意即可。

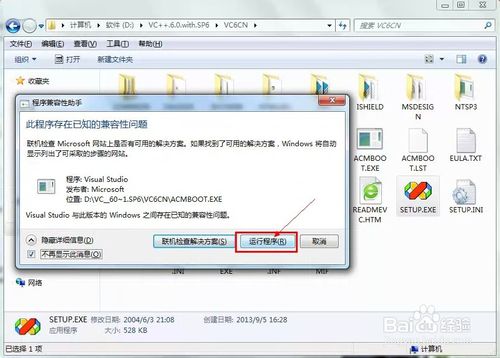
一般【直接使用默认的】点击下一步。

[](http://jingyan.baidu.com/album/a501d80ce2a7ecec630f5eb8.html?picindex=9)

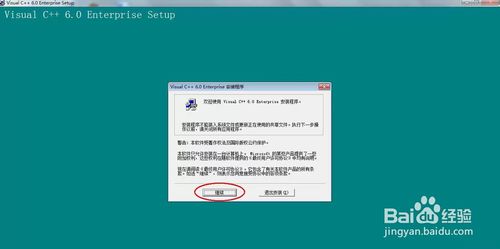
 安装Visual C++ 6.0 中文企业版。

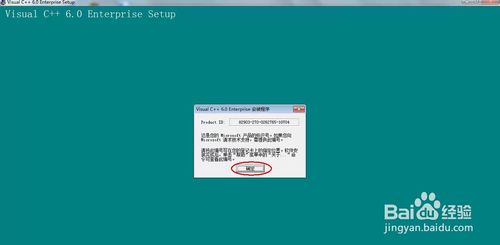
[](http://jingyan.baidu.com/album/a501d80ce2a7ecec630f5eb8.html?picindex=10)

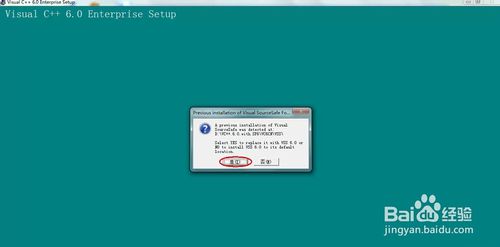
 则又可能出现兼容性问题，照样继续运行。

[](http://jingyan.baidu.com/album/a501d80ce2a7ecec630f5eb8.html?picindex=11)

 接下里连续的三个窗口点击【继续】→【确定】→【是】

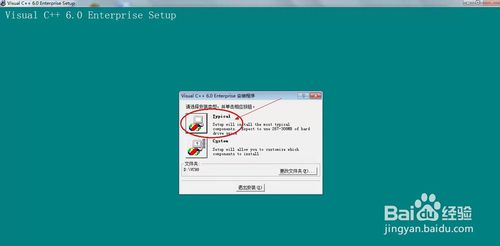
[](http://jingyan.baidu.com/album/a501d80ce2a7ecec630f5eb8.html?picindex=12)

[](http://jingyan.baidu.com/album/a501d80ce2a7ecec630f5eb8.html?picindex=13)

[](http://jingyan.baidu.com/album/a501d80ce2a7ecec630f5eb8.html?picindex=14)

 此处选择【Typical】继续安装。

**注意事项：**【Typical】表示安装的同时还有其他相关软件会给你安装进去，能够使你的电脑更加融洽的运用此软件。

[](http://jingyan.baidu.com/album/a501d80ce2a7ecec630f5eb8.html?picindex=15)

 【打钩后】点击OK

# 考试内容

## 1、C语言程序的结构

### 1.1 程序的构成,main函数和其他函数

1.1.1 main函数：

* 一个完整的C语言程序，是由一个、且只能有一个main()函数(又称主函数，必须有)和若干个其他函数结合而成（可选）。
* main函数是C语言程序的入口，程序是从main函数开始执行。

1.1.2 其他函数

* 从函数定义的角度看，函数可分为系统函数和用户定义函数：
* 系统函数，即库函数：这是由编译系统提供的，用户不必自己定义这些函数，可以直接使用它们，如我们常用的打印函数printf()。
* 用户定义函数：用以解决用户的专门需要。

### 1.2源程序的书写格式

**编写C语言代码：hello.c**

#include <stdio.h>

int *main*()

{

//这是第一个C语言代码

*printf*("hello world\n");

return 0;

}

C语言的源代码文件是一个普通的文本文件，但扩展名必须是.c。

### 1.3头文件,数据说明,函数的开始和结束标志以及程序中的注释

1.3.1 include头文件包含

* #include的意思是头文件包含，#include <stdio.h>代表包含stdio.h这个头文件
* 使用C语言库函数需要提前包含库函数对应的头文件，如这里使用了printf()函数，需要包含stdio.h头文件
* 可以通过man 3 printf查看printf所需的头文件

#include< > 与 #include ""的区别：

* < > 表示系统直接按系统指定的目录检索
* "" 表示系统先在 "" 指定的路径(没写路径代表当前路径)查找头文件，如果找不到，再按系统指定的目录检索

1.3.2 main函数

* 一个完整的C语言程序，是由一个、且只能有一个main()函数(又称主函数，必须有)和若干个其他函数结合而成（可选）。
* main函数是C语言程序的入口，程序是从main函数开始执行。

1.3.3 {} 括号，程序体和代码块

* {}叫代码块，一个代码块内部可以有一条或者多条语句
* C语言每句可执行代码都是"**;**"分号结尾
* 所有的#开头的行，都代表预编译指令，预编译指令行结尾是没有分号的
* 所有的可执行语句必须是在代码块里面

1.3.4 注释

* //叫行注释，注释的内容编译器是忽略的，注释主要的作用是在代码中加一些说明和解释，这样有利于代码的阅读
* /\*\*/叫块注释
* 块注释是C语言标准的注释方法
* 行注释是从C++语言借鉴过来的

1.3.5printf函数

* printf是C语言库函数，功能是向标准输出设备输出一个字符串
* printf(“hello world\n”);//\n的意思是回车换行

1.3.6 return语句

* return代表函数执行完毕，返回return代表函数的终止
* 如果main定义的时候前面是int，那么return后面就需要写一个整数；如果main定义的时候前面是void，那么return后面什么也不需要写
* 在main函数中return 0代表程序执行成功，return -1代表程序执行失败
* int main()和void main()在C语言中是一样的，但C++只接受int main这种定义方式

### 1.4 C语言的风格

1.4.1 缩进

* Tabs(制表符)是8个字符的大小，因此缩进也应该是8个字符的大小。
* 缩进背后的思想是:清楚地定义一个控制块从哪里开始，到哪里结束。

1.4.2放置花括号

* C程序中另一个要主意的就是花括号的放置。与缩进尺寸不同的是，关于如何放置花括号没有技术上的理由。
* 另外一种情况，就是函数：函数应当把左右括号都放在行首。
* 注意这种花括号的放置减少了空行的数目，并没损害可读性。因此，当屏幕上不可以有很多空行时，你就有更多的空行来安插注释。

1.4.3 标识符命名规则

* 标识符不能是关键字。
* 标识符只能由字母、数字、下划线组成。
* 第一个字符必须为字母或下划线。
* 标识符中字母区分大小写。

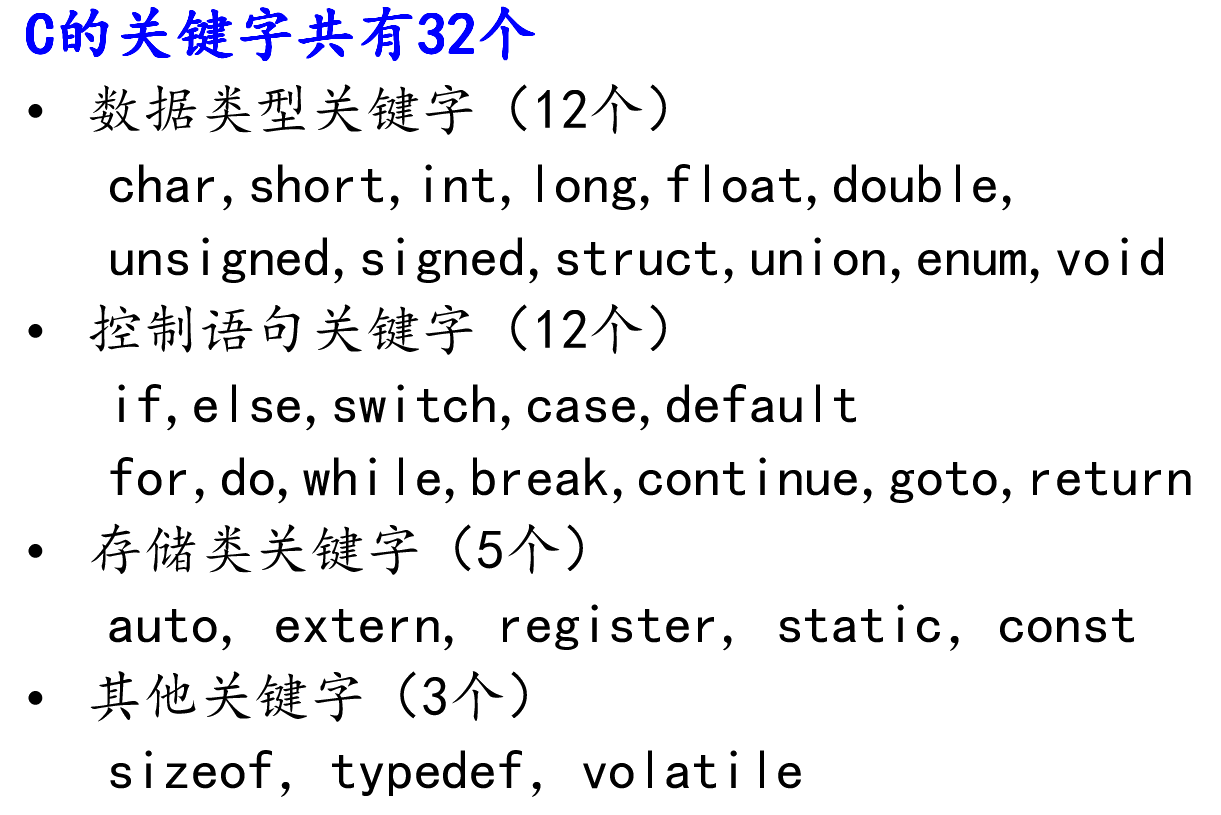
1.4.4 注释

* 注释是有用的，但过量的注释则是有害的。不要试图在注释中解释你的代码是如何工作的：把代码是如何工作的视为一件显然的事情会更好些，而且，给糟糕的代码作注释则是在浪费时间。

## 2、数据类型及其运算

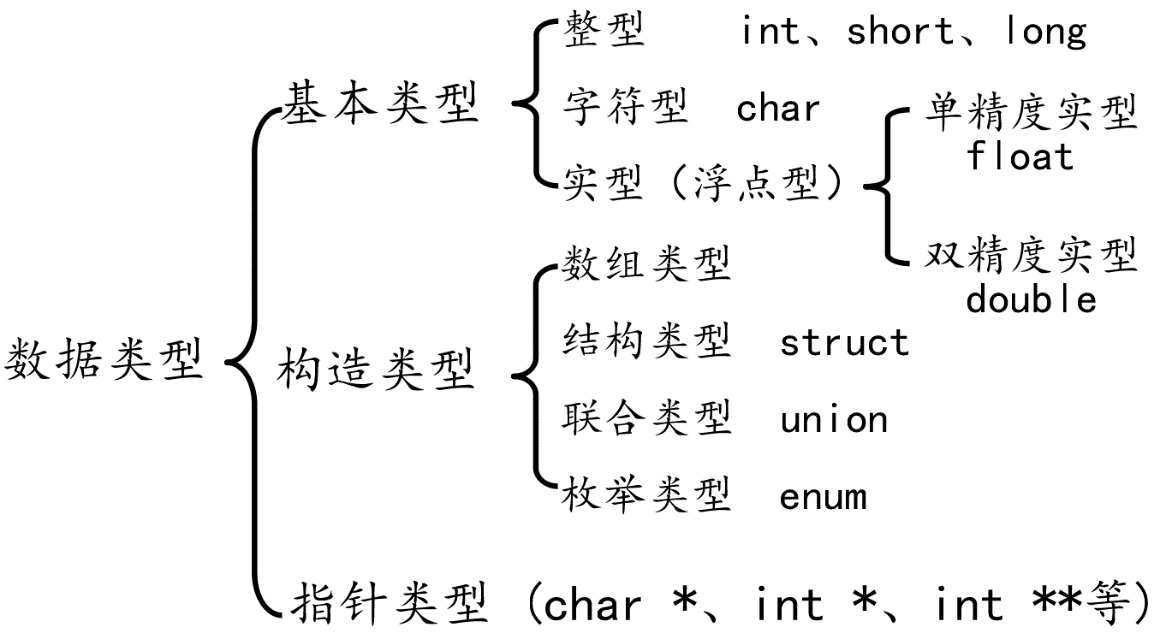
### 2.1 C的数据类型

#### 2.1.1 C语言关键字



#### 2.1.2 数据类型

数据类型的作用：编译器预算对象（变量）分配的内存空间大小。



##### 1） 整型：int

###### 一、 整型变量的定义和输出

| **打印格式** | **含义** |
| --- | --- |
| %d | 输出一个有符号的10进制int类型 |
| %o(字母o) | 输出8进制的int类型 |
| %x | 输出16进制的int类型，字母以小写输出 |
| %X | 输出16进制的int类型，字母以大写输出 |
| %u | 输出一个10进制的无符号数 |

#include <stdio.h>

int *main*()

{

int a = 123; //定义变量a，以10进制方式赋值为123

int b = 0567; //定义变量b，以8进制方式赋值为0567

int c = 0xabc; //定义变量c，以16进制方式赋值为0xabc

*printf*("a = %d\n", a);

*printf*("8进制：b = %o\n", b);

*printf*("10进制：b = %d\n", b);

*printf*("16进制：c = %x\n", c);

*printf*("16进制：c = %X\n", c);

*printf*("10进制：c = %d\n", c);

unsigned int d = 0xffffffff; //定义无符号int变量d，以16进制方式赋值

*printf*("有符号方式打印：d = %d\n", d);

*printf*("无符号方式打印：d = %u\n", d);

return 0;

}

###### 二、 整型变量的输入

#include <stdio.h>

int *main*()

{

int a;

*printf*("请输入a的值：");

//不要加“\n”

*scanf*("%d", &a);

*printf*("a = %d\n", a); //打印a的值

return 0;

}

###### 三、 short、int、long、long long

| **数据类型** | **占用空间** |
| --- | --- |
| short(短整型) | 2字节 |
| int(整型) | 4字节 |
| long(长整形) | Windows为4字节，Linux为4字节(32位)，8字节(64位) |
| long long(长长整形) | 8字节 |

注意：

* 需要注意的是，整型数据在内存中占的字节数与所选择的操作系统有关。虽然 C 语言标准中没有明确规定整型数据的长度，但 long 类型整数的长度不能短于 int 类型， short 类型整数的长度不能长于 int 类型。
* 当一个小的数据类型赋值给一个大的数据类型，不会出错，因为编译器会自动转化。但当一个大的类型赋值给一个小的数据类型，那么就可能丢失高位。

##### 2） 字符型：char

###### 一、 字符变量的定义和输出

字符型变量用于存储一个单一字符，在 C 语言中用 char 表示，其中每个字符变量都会占用 1 个字节。在给字符型变量赋值时，需要用一对英文半角格式的单引号(' ')把字符括起来。

字符变量实际上并不是把该字符本身放到变量的内存单元中去，而是将该字符对应的 ASCII 编码放到变量的存储单元中。char的本质就是一个1字节大小的整型。

#include <stdio.h>

int *main*()

{

char ch = 'a';

*printf*("sizeof(ch) = %u\n", sizeof(ch));

*printf*("ch[%%c] = %c\n", ch); //打印字符

*printf*("ch[%%d] = %d\n", ch); //打印‘a’ ASCII的值

char A = 'A';

char a = 'a';

*printf*("a = %d\n", a); //97

*printf*("A = %d\n", A); //65

*printf*("A = %c\n", 'a' - 32); //小写a转大写A

*printf*("a = %c\n", 'A' + 32); //大写A转小写a

ch = ' ';

*printf*("空字符：%d\n", ch); //空字符ASCII的值为32

*printf*("A = %c\n", 'a' - ' '); //小写a转大写A

*printf*("a = %c\n", 'A' + ' '); //大写A转小写a

return 0;

}

###### 二、 字符变量的输入

#include <stdio.h>

int *main*()

{

char ch;

*printf*("请输入ch的值：");

//不要加“\n”

*scanf*("%c", &ch);

*printf*("ch = %c\n", ch); //打印ch的字符

return 0;

}

###### 三、 ASCII对照表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **ASCII值** | **控制字符** | **ASCII值** | **字符** | **ASCII值** | **字符** | **ASCII值** | **字符** |
| 0 | NUT | 32 | (space) | 64 | @ | 96 | 、 |
| 1 | SOH | 33 | ! | 65 | A | 97 | a |
| 2 | STX | 34 | " | 66 | B | 98 | b |
| 3 | ETX | 35 | # | 67 | C | 99 | c |
| 4 | EOT | 36 | $ | 68 | D | 100 | d |
| 5 | ENQ | 37 | % | 69 | E | 101 | e |
| 6 | ACK | 38 | & | 70 | F | 102 | f |
| 7 | BEL | 39 | , | 71 | G | 103 | g |
| 8 | BS | 40 | ( | 72 | H | 104 | h |
| 9 | HT | 41 | ) | 73 | I | 105 | i |
| 10 | LF | 42 | \* | 74 | J | 106 | j |
| 11 | VT | 43 | + | 75 | K | 107 | k |
| 12 | FF | 44 | , | 76 | L | 108 | l |
| 13 | CR | 45 | - | 77 | M | 109 | m |
| 14 | SO | 46 | . | 78 | N | 110 | n |
| 15 | SI | 47 | / | 79 | O | 111 | o |
| 16 | DLE | 48 | 0 | 80 | P | 112 | p |
| 17 | DCI | 49 | 1 | 81 | Q | 113 | q |
| 18 | DC2 | 50 | 2 | 82 | R | 114 | r |
| 19 | DC3 | 51 | 3 | 83 | S | 115 | s |
| 20 | DC4 | 52 | 4 | 84 | T | 116 | t |
| 21 | NAK | 53 | 5 | 85 | U | 117 | u |
| 22 | SYN | 54 | 6 | 86 | V | 118 | v |
| 23 | TB | 55 | 7 | 87 | W | 119 | w |
| 24 | CAN | 56 | 8 | 88 | X | 120 | x |
| 25 | EM | 57 | 9 | 89 | Y | 121 | y |
| 26 | SUB | 58 | : | 90 | Z | 122 | z |
| 27 | ESC | 59 | ; | 91 | [ | 123 | { |
| 28 | FS | 60 | < | 92 | / | 124 | | |
| 29 | GS | 61 | = | 93 | ] | 125 | } |
| 30 | RS | 62 | > | 94 | ^ | 126 | ` |
| 31 | US | 63 | ? | 95 | \_ | 127 | DEL |

ASCII 码大致由以下两部分组成：

* ASCII 非打印控制字符： ASCII 表上的数字 0-31 分配给了控制字符，用于控制像打印机等一些外围设备。
* ASCII 打印字符：数字 32-126 分配给了能在键盘上找到的字符，当查看或打印文档时就会出现。数字 127 代表 Del 命令。

##### 3） 实型(浮点型)：float、double

实型变量也可以称为浮点型变量，浮点型变量是用来存储小数数值的。在C语言中， 浮点型变量分为两种： 单精度浮点数(float)、 双精度浮点数(double)， 但是double型变量所表示的浮点数比 float 型变量更精确。

| **数据类型** | **占用空间** | **有效数字范围** |
| --- | --- | --- |
| float | 4字节 | 31位有效数字 |
| double | 8字节 | 63～64位有效数字 |

由于浮点型变量是由有限的存储单元组成的，因此只能提供有限的有效数字。在有效位以外的数字将被舍去，这样可能会产生一些误差。

不以f结尾的常量是double类型，以f结尾的常量(如3.14f)是float类型。

#include <stdio.h>

int *main*()

{

//传统方式赋值

float a = 3.14f; //或3.14F

double b = 3.14;

*printf*("a = %f\n", a);

*printf*("b = %lf\n", b);

//科学法赋值

a = 3.2e3f; //3.2\*1000 = 3200，e可以写E

*printf*("a1 = %f\n", a);

a = 100e-3f; //100\*0.001 = 0.1

*printf*("a2 = %f\n", a);

a = 3.1415926f;

*printf*("a3 = %f\n", a); //结果为3.141593

return 0;

}

#### 2.1.3 常量

常量：

* 在程序运行过程中，其值不能被改变的量
* 常量一般出现在表达式或赋值语句中

|  |  |
| --- | --- |
| 整型常量 | 100，200，-100，0 |
| 实型常量 | 3.14 ， 0.125，-3.123 |
| 字符型常量 | ‘a’,‘b’,‘1’,‘\n’ |
| 字符串常量 | “a”,“ab”，“12356” |

#### 2.1.4 变量

变量：

* 在程序运行过程中，其值可以改变
* 变量在使用前必须先定义，定义变量前必须有相应的数据类型

### 2.2 C运算符的种类、运算优先级和结合性

#### 2.2.1 常用运算符分类

| **运算符类型** | **作用** |
| --- | --- |
| 算术运算符 | 用于处理四则运算 |
| 赋值运算符 | 用于将表达式的值赋给变量 |
| 比较运算符 | 用于表达式的比较，并返回一个真值或假值 |
| 逻辑运算符 | 用于根据表达式的值返回真值或假值 |
| 位运算符 | 用于处理数据的位运算 |
| sizeof运算符 | 用于求字节数长度 |

#### 2.2.2 算术运算符

| **运算符** | **术语** | **示例** | **结果** |
| --- | --- | --- | --- |
| + | 正号 | +3 | 3 |
| - | 负号 | -3 | -3 |
| + | 加 | 10 + 5 | 15 |
| - | 减 | 10 - 5 | 5 |
| \* | 乘 | 10 \* 5 | 50 |
| / | 除 | 10 / 5 | 2 |
| % | 取模(取余) | 10 % 3 | 1 |
| ++ | 前自增 | a=2; b=++a; | a=3; b=3; |
| ++ | 后自增 | a=2; b=a++; | a=3; b=2; |
| -- | 前自减 | a=2; b=--a; | a=1; b=1; |
| -- | 后自减 | a=2; b=a--; | a=1; b=2; |

#### 2.2.3 赋值运算符

| **运算符** | **术语** | **示例** | **结果** |
| --- | --- | --- | --- |
| = | 赋值 | a=2; b=3; | a=2; b=3; |
| += | 加等于 | a=0; a+=2; | a=2; |
| -= | 减等于 | a=5; a-=3; | a=2; |
| \*= | 乘等于 | a=2; a\*=2; | a=4; |
| /= | 除等于 | a=4; a/=2; | a=2; |
| %= | 模等于 | a=3; a%2; | a=1; |

#### 2.2.4 比较运算符

C 语言的比较运算中， “真”用数字“1”来表示， “假”用数字“0”来表示。

| **运算符** | **术语** | **示例** | **结果** |
| --- | --- | --- | --- |
| == | 相等于 | 4 == 3 | 0 |
| != | 不等于 | 4 != 3 | 1 |
| < | 小于 | 4 < 3 | 0 |
| > | 大于 | 4 > 3 | 1 |
| <= | 小于等于 | 4 <= 3 | 0 |
| >= | 大于等于 | 4 >= 1 | 1 |

#### 2.2.5 逻辑运算符

| **运算符** | **术语** | **示例** | **结果** |
| --- | --- | --- | --- |
| ! | 非 | !a | 如果a为假，则!a为真；  如果a为真，则!a为假。 |
| && | 与 | a && b | 如果a和b都为真，则结果为真，否则为假。 |
| || | 或 | a || b | 如果a和b有一个为真，则结果为真，二者都为假时，结果为假。 |

#### 2.2.6 运算符优先级

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **优先级** | **运算符** | **名称或含义** | **使用形式** | **结合方向** | **说明** |
| **1** | **[]** | 数组下标 | 数组名[常量表达式] | 左到右 | -- |
| **()** | 圆括号 | (表达式）/函数名(形参表) | -- |
| **.** | 成员选择（对象） | 对象.成员名 | -- |
| **->** | 成员选择（指针） | 对象指针->成员名 | -- |
| **2** | **-** | 负号运算符 | -表达式 | **右到左** | 单目运算符 |
| **~** | 按位取反运算符 | ~表达式 |
| **++** | 自增运算符 | ++变量名/变量名++ |
| **--** | 自减运算符 | --变量名/变量名-- |
| **\*** | 取值运算符 | \*指针变量 |
| **&** | 取地址运算符 | &变量名 |
| **!** | 逻辑非运算符 | !表达式 |
| **(类型)** | 强制类型转换 | (数据类型)表达式 | -- |
| **sizeof** | 长度运算符 | sizeof(表达式) | -- |
| **3** | **/** | 除 | 表达式/表达式 | 左到右 | 双目运算符 |
| **\*** | 乘 | 表达式\*表达式 |
| **%** | 余数（取模） | 整型表达式%整型表达式 |
| **4** | **+** | 加 | 表达式+表达式 | 左到右 | 双目运算符 |
| **-** | 减 | 表达式-表达式 |
| **5** | **<<** | 左移 | 变量<<表达式 | 左到右 | 双目运算符 |
| **>>** | 右移 | 变量>>表达式 |
| **6** | **>** | 大于 | 表达式>表达式 | 左到右 | 双目运算符 |
| **>=** | 大于等于 | 表达式>=表达式 |
| **<** | 小于 | 表达式<表达式 |
| **<=** | 小于等于 | 表达式<=表达式 |
| **7** | **==** | 等于 | 表达式==表达式 | 左到右 | 双目运算符 |
| **!=** | 不等于 | 表达式!= 表达式 |
| **8** | **&** | 按位与 | 表达式&表达式 | 左到右 | 双目运算符 |
| **9** | **^** | 按位异或 | 表达式^表达式 | 左到右 | 双目运算符 |
| **10** | **|** | 按位或 | 表达式|表达式 | 左到右 | 双目运算符 |
| **11** | **&&** | 逻辑与 | 表达式&&表达式 | 左到右 | 双目运算符 |
| **12** | **||** | 逻辑或 | 表达式||表达式 | 左到右 | 双目运算符 |
| **13** | **?:** | 条件运算符 | 表达式1?  表达式2: 表达式3 | **右到左** | 三目运算符 |
| **14** | **=** | 赋值运算符 | 变量=表达式 | **右到左** | -- |
| **/=** | 除后赋值 | 变量/=表达式 | -- |
| **\*=** | 乘后赋值 | 变量\*=表达式 | -- |
| **%=** | 取模后赋值 | 变量%=表达式 | -- |
| **+=** | 加后赋值 | 变量+=表达式 | -- |
| **-=** | 减后赋值 | 变量-=表达式 | -- |
| **<<=** | 左移后赋值 | 变量<<=表达式 | -- |
| **>>=** | 右移后赋值 | 变量>>=表达式 | -- |
| **&=** | 按位与后赋值 | 变量&=表达式 | -- |
| **^=** | 按位异或后赋值 | 变量^=表达式 | -- |
| **|=** | 按位或后赋值 | 变量|=表达式 | -- |
| **15** | **，** | 逗号运算符 | 表达式,表达式,… | 左到右 | -- |

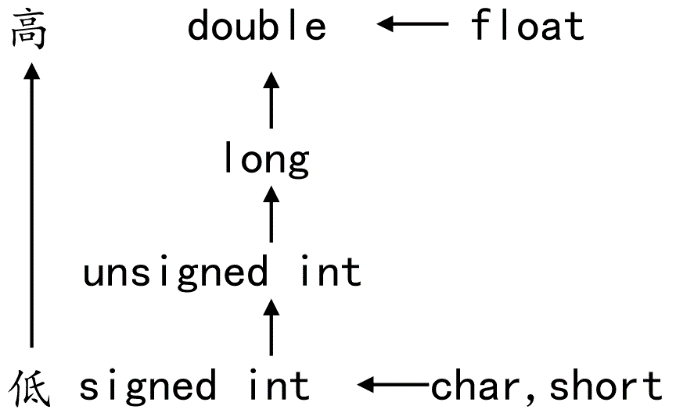
### 2.3 不同类型数据间的转换与运算

数据有不同的类型，不同类型数据之间进行混合运算时必然涉及到类型的转换问题。

转换的方法有两种：

* 自动转换(隐式转换)：遵循一定的规则,由编译系统自动完成。
* 强制类型转换：把表达式的运算结果强制转换成所需的数据类型。

类型转换的原则：占用内存字节数少(值域小)的类型，向占用内存字节数多(值域大)的类型转换，以保证精度不降低。



#### 2.3.1 隐式转换

#include <stdio.h>

int *main*()

{

int num = 5;

*printf*("s1=%d\n", num / 2);

*printf*("s2=%lf\n", num / 2.0);

return 0;

}

#### 2.3.2 强制转换

强制类型转换指的是使用强制类型转换运算符，将一个变量或表达式转化成所需的类型，其基本语法格式如下所示：

(类型说明符) (表达式)

#include <stdio.h>

int *main*()

{

float x = 0;

int i = 0;

x = 3.6f;

i = x; //x为实型, i为整型，直接赋值会有警告

i = (int)x; //使用强制类型转换

*printf*("x=%f, i=%d\n", x, i);

return 0;

}

### 2.4 C表达式类型

**注意：(赋值表达式,算术表达式,关系表达式,逻辑表达式,条件表达式,逗号表达式)和求值规则。二级各科考试的公共基础知识大纲及样题见高等教育出版社出版的《全国计算机等级考试二级教程——公共基础知识(2013年版)》的附录部分。**

## 3、基本语句

### 3.1 表达式语句,空语句,复合语句

#### 3.1.1 表达式语句

C程序设计语言的表达式语句由表达式加上分号“；”组成。

分别由以下几种类：算术表达式、赋值表达式、逗号表达式、关系表达式、逻辑表达式 等...

其一般形式为：

**表达式；**

执行表达式语句就是计算表达式的值。

#### 3.1.2 空语句

* 空语句是什么也不执行的语句，一般由分号“；”或大括号"{ }"组成。
* 在程序中空语句可用来作空循环体。
* 一般推荐使用大括号"{ }"来表示空语句，这样更直观

#### 3.1.3 复合语句

* 把多个语句用括号{}括起来组成的一个语句称复合语句。 在程序中应把复合语句看成是单条语句，而不是多条语句
* 复合语句内的各条语句都必须以分号“;”结尾；此外，在括号“}”外不能加分号。

### 3.2 输入输出函数的调用

#### 3.2.1 printf函数

printf是输出一个字符串。

printf格式字符：

| **打印格式** | **对应数据类型** | **含义** |
| --- | --- | --- |
| %d | int | 接受整数值并将它表示为有符号的十进制整数 |
| %hd | short int | 短整数 |
| %hu | unsigned short | 无符号短整数 |
| %o | unsigned int | 无符号8进制整数 |
| %u | unsigned int | 无符号10进制整数 |
| %x,%X | unsigned int | 无符号16进制整数，x对应的是abcdef，X对应的是ABCDEF |
| %f | float | 单精度浮点数 |
| %lf | double | 双精度浮点数 |
| %e,%E | double | 科学计数法表示的数，此处"e"的大小写代表在输出时用的"e"的大小写 |
| %c | char | 字符型。可以把输入的数字按照ASCII码相应转换为对应的字符 |
| %s | char \* | 字符串。输出字符串中的字符直至字符串中的空字符（字符串以'\0‘结尾，这个'\0'即空字符） |
| %p | void \* | 以16进制形式输出指针 |
| %% | % | 输出一个百分号 |

printf附加格式：

| **字符** | **含义** |
| --- | --- |
| l(字母l) | 附加在d,u,x,o前面，表示长整数 |
| - | 左对齐 |
| m(代表一个整数) | 数据最小宽度 |
| 0(数字0) | 将输出的前面补上0直到占满指定列宽为止不可以搭配使用- |
| m.n(代表一个整数) | m指域宽，即对应的输出项在输出设备上所占的字符数。n指精度，用于说明输出的实型数的小数位数。对数值型的来说，未指定n时，隐含的精度为n=6位。 |

#include <stdio.h>

int *main*()

{

int a = 100;

*printf*("a = %d\n", a);//格式化输出一个字符串

*printf*("%p\n", &a);//输出变量a在内存中的地址编号

*printf*("%%d\n");

char c = 'a';

*putchar*(c);//putchar只有一个参数，就是要输出的char

long a2 = 100;

*printf*("%ld, %lx, %lo\n", a2, a2, a2);

long long a3 = 1000;

*printf*("%lld, %llx, %llo\n", a3, a3, a3);

int abc = 10;

*printf*("abc = '%6d'\n", abc);

*printf*("abc = '%-6d'\n", abc);

*printf*("abc = '%06d'\n", abc);

*printf*("abc = '%-06d'\n", abc);

double d = 12.3;

*printf*("d = \' %-10.3lf \'\n", d);

return 0;

}

#### 3.2.2 scanf函数

* scanf通过%转义的方式可以得到用户通过标准输入设备输入的数据。

#include <stdio.h>

int *main*()

{

char ch1;

char ch2;

char ch3;

int a;

int b;

*printf*("请输入ch1的字符：");

ch1 = *getchar*();

*printf*("ch1 = %c\n", ch1);

*getchar*(); //测试此处getchar()的作用

*printf*("请输入ch2的字符：");

ch2 = *getchar*();

*printf*("\'ch2 = %ctest\'\n", ch2);

*getchar*(); //测试此处getchar()的作用

*printf*("请输入ch3的字符：");

*scanf*("%c", &ch3);//这里第二个参数一定是变量的地址，而不是变量名

*printf*("ch3 = %c\n", ch3);

*printf*("请输入a的值：");

*scanf*("%d", &a);

*printf*("a = %d\n", a);

*printf*("请输入b的值：");

*scanf*("%d", &b);

*printf*("b = %d\n", b);

return 0;

}

## 4、选择结构程序设计

### 4.1 if 语句

#### 4.1.1 if语句



#include <stdio.h>

int *main*()

{

int a = 1;

int b = 2;

if (a > b)

{

*printf*("%d\n", a);

}

return 0;

}

#### 4.1.2 if…else语句



#include <stdio.h>

int *main*()

{

int a = 1;

int b = 2;

if (a > b)

{

*printf*("%d\n", a);

}

else

{

*printf*("%d\n", b);

}

return 0;

}

#### 4.1.3 if…else if…else语句



#include <stdio.h>

int *main*()

{

unsigned int a;

*scanf*("%u", &a);

if (a < 10)

{

*printf*("个位\n");

}

else if (a < 100)

{

*printf*("十位\n");

}

else if (a < 1000)

{

*printf*("百位\n");

}

else

{

*printf*("很大\n");

}

return 0;

}

### 4.2 switch语句

#### 4.2.1 switch语句

#include <stdio.h>

int *main*()

{

char c;

c = *getchar*();

switch (c) //参数只能是整型变量

{

case '1':

*printf*("OK\n");

break;//switch遇到break就中断了

case '2':

*printf*("not OK\n");

break;

default://如果上面的条件都不满足，那么执行default

*printf*("are u ok?\n");

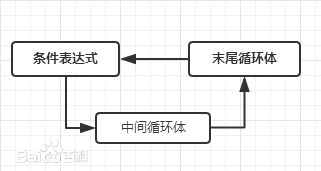
}

return 0;

}

## 5、循环结构程序设计

### 5.1 for循环结构



#include <stdio.h>

int *main*()

{

int i;

int sum = 0;

for (i = 0; i <= 100; i++)

{

sum += i;

}

*printf*("sum = %d\n", sum);

return 0;

}

### 5.2 while和do-while循环结构

#### 5.2.1 while语句



#include <stdio.h>

int *main*()

{

int a = 20;

while (a > 10)

{

*scanf*("%d", &a);

*printf*("a = %d\n", a);

}

return 0;

}

#### 5.2.2 do…while语句



#include <stdio.h>

int *main*()

{

int a = 1;

do

{

a++;

*printf*("a = %d\n", a);

} while (a < 10);

return 0;

}

### 5.3 continue语句和break语句

#### 5.3.1 break语句

在switch条件语句和循环语句中都可以使用break语句：

* 当它出现在switch条件语句中时，作用是终止某个case并跳出switch结构。
* 当它出现在循环语句中，作用是跳出当前内循环语句，执行后面的代码。
* 当它出现在嵌套循环语句中，跳出最近的内循环语句，执行后面的代码。

#include <stdio.h>

int *main*()

{

int i = 0;

while (1)

{

i++;

*printf*("i = %d\n", i);

if (i == 10)

{

break; //跳出while循环

}

}

int flag = 0;

int m = 0;

int n = 0;

for (m = 0; m < 10; m++)

{

for (n = 0; n < 10; n++)

{

if (n == 5)

{

flag = 1;

break; //跳出for (n = 0; n < 10; n++)

}

}

if (flag == 1)

{

break; //跳出for (m = 0; m < 10; m++)

}

}

return 0;

}

#### 5.3.2 continue语句

在循环语句中，如果希望立即终止本次循环，并执行下一次循环，此时就需要使用continue语句。

#include<stdio.h>

int *main*()

{

int sum = 0; //定义变量sum

for (int i = 1; i <= 100; i++)

{

if (i % 2 == 0) //如果i是一个偶数，执行if语句中的代码

{

continue; //结束本次循环

}

sum += i; //实现sum和i的累加

}

*printf*("sum = %d\n", sum);

return 0;

}

### 5.4 循环的嵌套

循环语句之间可以相互嵌套：

#include <stdio.h>

int *main*()

{

int num = 0;

int i, j, k;

for (i = 0; i < 10; i++)

{

for (j = 0; j < 10; j++)

{

for (k = 0; k < 10; k++)

{

*printf*("hello world\n");

num++;

}

}

}

*printf*("num = %d\n", num);

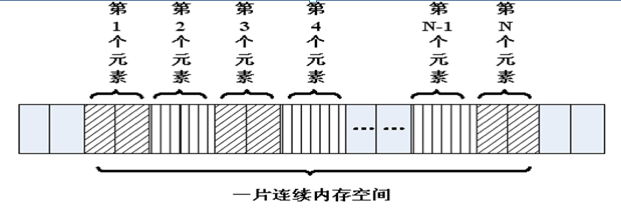
return 0;

}

## 6、数组的定义和引用

在程序设计中，为了方便处理数据把具有相同类型的若干变量按有序形式组织起来——称为数组。

数组就是在内存中连续的相同类型的变量空间。同一个数组所有的成员都是相同的数据类型，同时所有的成员在内存中的地址是连续的。



数组属于构造数据类型：

* 一个数组可以分解为多个数组元素：这些数组元素可以是基本数据类型或构造类型。

int a[10];

struct Stu boy[10];

* 按数组元素类型的不同，数组可分为：数值数组、字符数组、指针数组、结构数组等类别。

int a[10];

char s[10];

char \*p[10];

通常情况下，数组元素下标的个数也称为维数，根据维数的不同，可将数组分为一维数组、二维数组、三维数组、四维数组等。通常情况下，我们将二维及以上的数组称为多维数组。

### 6.1 一维数组和二维数组的定义、初始化和数组元素的引用。

#### 6.1.1 一维数组的定义和使用

* 数组名字符合标识符的书写规定(数字、英文字母、下划线)
* 数组名不能与其它变量名相同，同一作用域内是唯一的
* 方括号[]中常量表达式表示数组元素的个数

int a[3]表示数组a有3个元素

其下标从0开始计算，因此3个元素分别为a[0],a[1],a[2]

* 定义数组时[]内最好是常量，使用数组时[]内即可是常量，也可以是变量

#include <stdio.h>

int *main*()

{

int a[10];//定义了一个数组，名字叫a，有10个成员，每个成员都是int类型

//a[0]…… a[9]，没有a[10]

//没有a这个变量，a是数组的名字，但不是变量名，它是常量

a[0] = 0;

//……

a[9] = 9;

int i = 0;

for (i = 0; i < 10; i++)

{

a[i] = i; //给数组赋值

}

//遍历数组，并输出每个成员的值

for (i = 0; i < 10; i++)

{

*printf*("%d ", a[i]);

}

*printf*("\n");

return 0;

}

#### 6.1.2 一维数组的初始化

在定义数组的同时进行赋值，称为初始化。全局数组若不初始化，编译器将其初始化为零。局部数组若不初始化，内容为随机值。

int a[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };//定义一个数组，同时初始化所有成员变量

int a[10] = { 1, 2, 3 };//初始化前三个成员，后面所有元素都设置为0

int a[10] = { 0 };//所有的成员都设置为0

//[]中不定义元素个数，定义时必须初始化

int a[] = { 1, 2, 3, 4, 5 };//定义了一个数组，有5个成员

#### 6.1.3 数组名

数组名是一个地址的常量，代表数组中首元素的地址。

#include <stdio.h>

int *main*()

{

int a[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };//定义一个数组，同时初始化所有成员变量

*printf*("a = %p\n", a);

*printf*("&a[0] = %p\n", &a[0]);

int n = sizeof(a); //数组占用内存的大小，10个int类型，10 \* 4 = 40

int n0 = sizeof(a[0]);//数组第0个元素占用内存大小，第0个元素为int，4

int i = 0;

for (i = 0; i < sizeof(a) / sizeof(a[0]); i++)

{

*printf*("%d ", a[i]);

}

*printf*("\n");

return 0;

}

### 6.2 字符串与字符数组

#### 6.2.1 字符数组与字符串区别

* C语言中没有字符串这种数据类型，可以通过char的数组来替代；
* 字符串一定是一个char的数组，但char的数组未必是字符串；
* 数字0(和字符‘\0’等价)结尾的char数组就是一个字符串，但如果char数组没有以数字0结尾，那么就不是一个字符串，只是普通字符数组，所以字符串是一种特殊的char的数组。

#include <stdio.h>

int *main*()

{

char c1[] = { 'c', ' ', 'p', 'r', 'o', 'g' }; //普通字符数组

*printf*("c1 = %s\n", c1); //乱码，因为没有’\0’结束符

//以‘\0’(‘\0’就是数字0)结尾的字符数组是字符串

char c2[] = { 'c', ' ', 'p', 'r', 'o', 'g', '\0'};

*printf*("c2 = %s\n", c2);

//字符串处理以‘\0’(数字0)作为结束符，后面的'h', 'l', 'l', 'e', 'o'不会输出

char c3[] = { 'c', ' ', 'p', 'r', 'o', 'g', '\0', 'h', 'l', 'l', 'e', 'o', '\0'};

*printf*("c3 = %s\n", c3);

return 0;

}

#### 6.2.2 字符串的初始化

#include <stdio.h>

// C语言没有字符串类型，通过字符数组模拟

// C语言字符串，以字符‘\0’, 数字0

int *main*()

{

//不指定长度, 没有0结束符，有多少个元素就有多长

char buf[] = { 'a', 'b', 'c' };

*printf*("buf = %s\n", buf); //乱码

//指定长度，后面没有赋值的元素，自动补0

char buf2[100] = { 'a', 'b', 'c' };

char buf[1000]={“hello”};

*printf*("buf2 = %s\n", buf2);

//所有元素赋值为0

char buf3[100] = { 0 };

//char buf4[2] = { '1', '2', '3' };//数组越界

char buf5[50] = { '1', 'a', 'b', '0', '7' };

*printf*("buf5 = %s\n", buf5);

char buf6[50] = { '1', 'a', 'b', 0, '7' };

*printf*("buf6 = %s\n", buf6);

char buf7[50] = { '1', 'a', 'b', '\0', '7' };

*printf*("buf7 = %s\n", buf7);

//使用字符串初始化，编译器自动在后面补0，常用

char buf8[] = "agjdslgjlsdjg";

//'\0'后面最好不要连着数字，有可能几个数字连起来刚好是一个转义字符

//'\ddd'八进制字义字符，'\xdd'十六进制转移字符

// \012相当于\n

char str[] = "\012abc";

*printf*("str == %s\n", str);

return 0;

}

#### 6.2.3 字符串的输入输出

由于字符串采用了'\0'标志，字符串的输入输出将变得简单方便。

#include <stdio.h>

int *main*()

{

char str[100];

printf("input string1 : \n");

scanf("%s", str);//scanf(“%s”,str)默认以空格分隔

printf("output:%s\n", str);

return 0;

}

## 7、函数

C 程序是由函数组成的，我们写的代码都是由主函数 main()开始执行的。函数是 C 程序的基本模块，是用于完成特定任务的程序代码单元。

从函数定义的角度看，函数可分为系统函数和用户定义函数两种：

* 系统函数，即库函数：这是由编译系统提供的，用户不必自己定义这些函数，可以直接使用它们，如我们常用的打印函数printf()。
* 用户定义函数：用以解决用户的专门需要。

### 7.1 库函数的正确调用

* 导入头文件
* 使用库函数，例如printf scanf

#### 7.1.1函数的调用：产生随机数

当调用函数时，需要关心5要素：

* 头文件：包含指定的头文件
* 函数名字：函数名字必须和头文件声明的名字一样
* 功能：需要知道此函数能干嘛后才调用
* 参数：参数类型要匹配
* 返回值：根据需要接收返回值

#include <time.h>

***time\_t* time(*time\_t* \*t);**

功能：获取当前系统时间

参数：常设置为NULL

返回值：当前系统时间, *time\_t* 相当于long类型，单位为毫秒

#include <stdlib.h>

**void srand(unsigned int seed);**

功能：用来设置rand()产生随机数时的随机种子

参数：如果每次seed相等，rand()产生随机数相等

返回值：无

#include <stdlib.h>

**int rand(void);**

功能：返回一个随机数值

参数：无

返回值：随机数

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

int *main*()

{

*time\_t* tm = *time*(*NULL*);//得到系统时间

*srand*((unsigned int)tm);//随机种子只需要设置一次即可

int r = *rand*();

*printf*("r = %d\n", r);

return 0;

}

### 7.2 函数的定义方法

#### 7.2.1 函数定义格式

函数定义的一般形式：

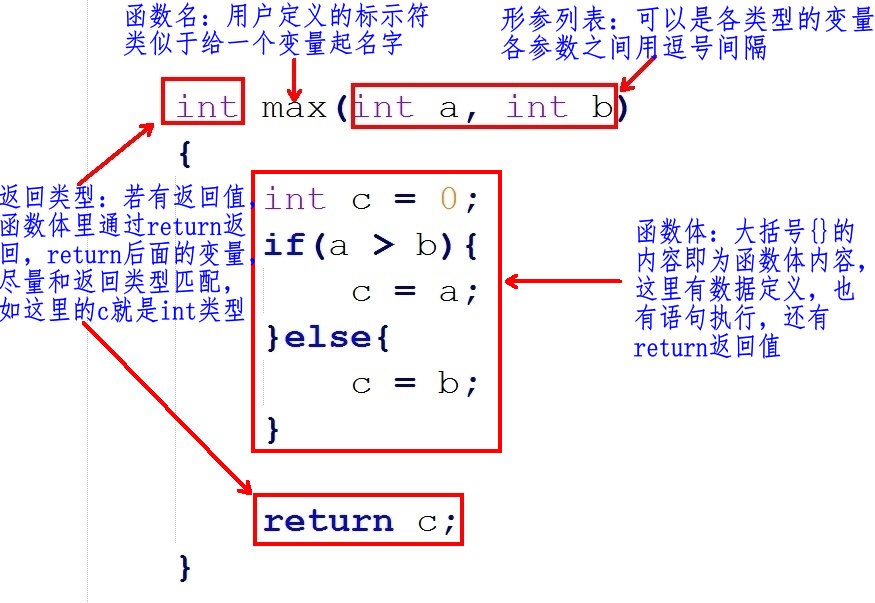
返回类型 函数名(形式参数列表)

{

数据定义部分;

执行语句部分;

}



#### 7.2.2 函数名字、形参、函数体、返回值

##### 1) 函数名

理论上是可以随意起名字，最好起的名字见名知意，应该让用户看到这个函数名字就知道这个函数的功能。注意，函数名的后面有个圆换号()，代表这个为函数，不是普通的变量名。

##### 2) 形参列表

在定义函数时指定的形参，在未出现函数调用时，它们并不占内存中的存储单元，因此称它们是形式参数或虚拟参数，简称形参，表示它们并不是实际存在的数据，所以，形参里的变量不能赋值。

void *max*(int a = 10, int b = 20) // error, 形参不能赋值

{

}

在定义函数时指定的形参，必须是，类型+变量的形式：

//1: right, 类型+变量

void *max*(int a, int b)

{

}

//2: error, 只有类型，没有变量

void *max*(int, int)

{

}

//3: error, 只有变量，没有类型

int a, int b;

void *max*(a, b)

{

}

在定义函数时指定的形参，可有可无，根据函数的需要来设计，如果没有形参，圆括号内容为空，或写一个void关键字：

// 没形参， 圆括号内容为空

void *max*()

{

}

// 没形参， 圆括号内容为void关键字

void *max*(void)

{

}

##### 3) 函数体

花括号{ }里的内容即为函数体的内容，这里为函数功能实现的过程，这和以前的写代码没太大区别，以前我们把代码写在main()函数里，现在只是把这些写到别的函数里。

### 7.3 函数的类型和返回值

#### 7.3.1 无参函数调用

如果是调用无参函数，则不能加上“实参”，但括号不能省略。

// 函数的定义

void test()

{

}

int main()

{

// 函数的调用

test(); // right, 圆括号()不能省略

test(250); // error, 函数定义时没有参数

return 0;

}

#### 7.3.2 有参函数调用

a)如果实参表列包含多个实参，则各参数间用逗号隔开。

// 函数的定义

void test(int a, int b)

{

}

int main()

{

int p = 10, q = 20;

test(p, q); // 函数的调用

return 0;

}

b)实参与形参的个数应相等，类型应匹配(相同或赋值兼容)。实参与形参按顺序对应，一对一地传递数据。

c)实参可以是常量、变量或表达式，无论实参是何种类型的量，在进行函数调用时，它们都必须具有确定的值，以便把这些值传送给形参。所以，这里的变量是在圆括号( )外面定义好、赋好值的变量。

// 函数的定义

void test(int a, int b)

{

}

int main()

{

// 函数的调用

int p = 10, q = 20;

test(p, q); // right

test(11, 30 - 10); // right

test(int a, int b); // error, 不应该在圆括号里定义变量

return 0;

}

#### 7.3.3 返回值

函数的返回值是通过函数中的return语句获得的，return后面的值也可以是一个表达式。

1. 尽量保证return语句中表达式的值和函数返回类型是同一类型。

int *max*() // 函数的返回值为int类型

{

int a = 10;

return a;// 返回值a为int类型，函数返回类型也是int，匹配

}

b)如果函数返回的类型和return语句中表达式的值不一致，则以函数返回类型为准，即函数返回类型决定返回值的类型。对数值型数据，可以自动进行类型转换。

double *max*() // 函数的返回值为double类型

{

int a = 10;

return a;// 返回值a为int类型，它会转为double类型再返回

}

注意：如果函数返回的类型和return语句中表达式的值不一致，而它又无法自动进行类型转换，程序则会报错。

c)return语句的另一个作用为中断return所在的执行函数，类似于break中断循环、switch语句一样。

int *max*()

{

return 1;// 执行到，函数已经被中断，所以下面的return 2无法被执行到

return 2;// 没有执行

}

d)如果函数带返回值，return后面必须跟着一个值，如果函数没有返回值，函数名字的前面必须写一个void关键字，这时候，我们写代码时也可以通过return中断函数(也可以不用)，只是这时，return后面不带内容( 分号“；”除外)。

void *max*()// 最好要有void关键字

{

return; // 中断函数，这个可有可无

}

### 7.4 形式参数与实在参数,参数值的传递

#### 7.4.1 函数的形参和实参

* 形参出现在函数定义中，在整个函数体内都可以使用，离开该函数则不能使用。
* 实参出现在主调函数中，进入被调函数后，实参也不能使用。
* 实参变量对形参变量的数据传递是“值传递”，即单向传递，只由实参传给形参，而不能由形参传回来给实参。
* 在调用函数时，编译系统临时给形参分配存储单元。调用结束后，形参单元被释放。
* 实参单元与形参单元是不同的单元。调用结束后，形参单元被释放，函数调用结束返回主调函数后则不能再使用该形参变量。实参单元仍保留并维持原值。因此，在执行一个被调用函数时，形参的值如果发生改变，并不会改变主调函数中实参的值。

#### 7.4.2 函数执行流程

#include <stdio.h>

void print\_test()

{

*printf*("this is for test\n");

}

int main()

{

print\_test(); // print\_test函数的调用

return 0;

}

1. **进入main()函数**
2. **调用print\_test()函数：**
3. 它会在main()函数的前寻找有没有一个名字叫“print\_test”的函数定义；
4. 如果找到，接着检查函数的参数，这里调用函数时没有传参，函数定义也没有形参，参数类型匹配；
5. 开始执行print\_test()函数，这时候，main()函数里面的执行会阻塞( 停 )在print\_test()这一行代码，等待print\_test()函数的执行。
6. **print\_test()函数执行完( 这里打印一句话 )，main()才会继续往下执行，执行到return 0, 程序执行完毕。**

### 7.5 函数的正确调用,嵌套调用,递归调用

#### 7.5.1 函数正确调用

#include <stdio.h>

int max(int x, int y); // 函数的声明，分号不能省略

// int max(int, int); // 另一种方式

int main()

{

int a = 10, b = 25, num\_max = 0;

num\_max = max(a, b); // 函数的调用

*printf*("num\_max = %d\n", num\_max);

return 0;

}

// 函数的定义

int max(int x, int y)

{

return x > y ? x : y;

}

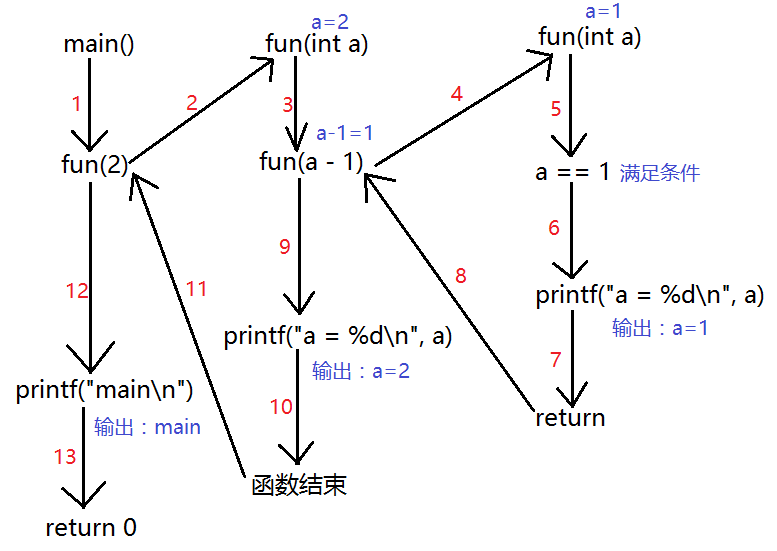
#### 7.5.2 函数嵌套

|  |
| --- |
| void funB**(**int b**){**  printf**(**"b = %d\n"**,** b**);**  **}**  void funA**(**int a**){**  funB**(**a **-** 1**);**  printf**(**"a = %d\n"**,** a**);**  **}**  int main**(**void**){**  funA**(**2**);**  printf**(**"main\n"**);**  **return** 0**;**  **}** |

#### 7.5.3 递归函数

|  |
| --- |
| void fun**(**int a**){**    **if** **(**a **==** 1**){**  printf**(**"a = %d\n"**,** a**);**  **return;** //中断函数很重要  **}**  fun**(**a **-** 1**);**  printf**(**"a = %d\n"**,** a**);**  **}**  int main**(**void**){**    fun**(**2**);**  printf**(**"main\n"**);**  **return** 0**;**  **}** |

函数的调用流程如下：



### 7.6 局部变量和全局变量

#### 7.6.1 局部变量

局部变量也叫auto自动变量(auto可写可不写)，一般情况下代码块{}内部定义的变量都是自动变量，它有如下特点：

* 在一个函数内定义，只在函数范围内有效
* 在复合语句中定义，只在复合语句中有效
* 随着函数调用的结束或复合语句的结束局部变量的声明声明周期也结束
* 如果没有赋初值，内容为随机

#include <stdio.h>

void test()

{

//auto写不写是一样的

//auto只能出现在{}内部

auto int b = 10;

}

int *main*(void)

{

//b = 100; //err， 在main作用域中没有b

if (1)

{

//在复合语句中定义，只在复合语句中有效

int a = 10;

*printf*("a = %d\n", a);

}

//a = 10; //err离开if()的复合语句，a已经不存在

return 0;

}

#### 7.6.2 全局变量

* 在函数外定义，可被本文件及其它文件中的函数所共用，若其它文件中的函数调用此变量,须用extern声明
* 全局变量的生命周期和程序运行周期一样
* 不同文件的全局变量不可重名

## 8、编译预处理

### 8.1 宏定义和调用(不带参数的宏,带参数的宏)

#### 8.1.1 无参数的宏定义(宏常量)

如果在程序中大量使用到了100这个值，那么为了方便管理，我们可以将其定义为：

const int num = 100; 但是如果我们使用num定义一个数组，在不支持c99标准的编译器上是不支持的，因为num不是一个编译器常量，如果想得到了一个编译器常量，那么可以使用：

#define num 100

在编译预处理时，将程序中在该语句以后出现的所有的num都用100代替。这种方法使用户能以一个简单的名字代替一个长的字符串，在预编译时将宏名替换成字符串的过程称为“宏展开”。宏定义，只在宏定义的文件中起作用。

|  |
| --- |
| #define PI 3.1415  void test**(){**  double r **=** 10.0**;**  double s **=** PI **\*** r **\*** r**;**  printf**(**"s = %lf\n"**,** s**);**  **}** |

说明：

1. 宏名一般用大写，以便于与变量区别；
2. 宏定义可以是常数、表达式等；
3. 宏定义不作语法检查，只有在编译被宏展开后的源程序才会报错；
4. 宏定义不是C语言，不在行末加分号；
5. 宏名有效范围为从定义到本源文件结束；
6. 可以用#undef命令终止宏定义的作用域；
7. 在宏定义中，可以引用已定义的宏名；

#### 8.1.2 带参数的宏定义(宏函数)

在项目中，经常把一些短小而又频繁使用的函数写成宏函数，这是由于宏函数没有普通函数参数压栈、跳转、返回等的开销，可以调高程序的效率。

宏通过使用参数，可以创建外形和作用都与函数类似地类函数宏(function-like macro). 宏的参数也用圆括号括起来。

|  |
| --- |
| #define SUM(x,y) ((x)+(y))  void test**(){**    //仅仅只是做文本替换 下例替换为 int ret = ((10)+(20));  //不进行计算  int ret **=** SUM**(**10**,** 20**);**  printf**(**"ret:%d\n"**,**ret**);**  **}** |

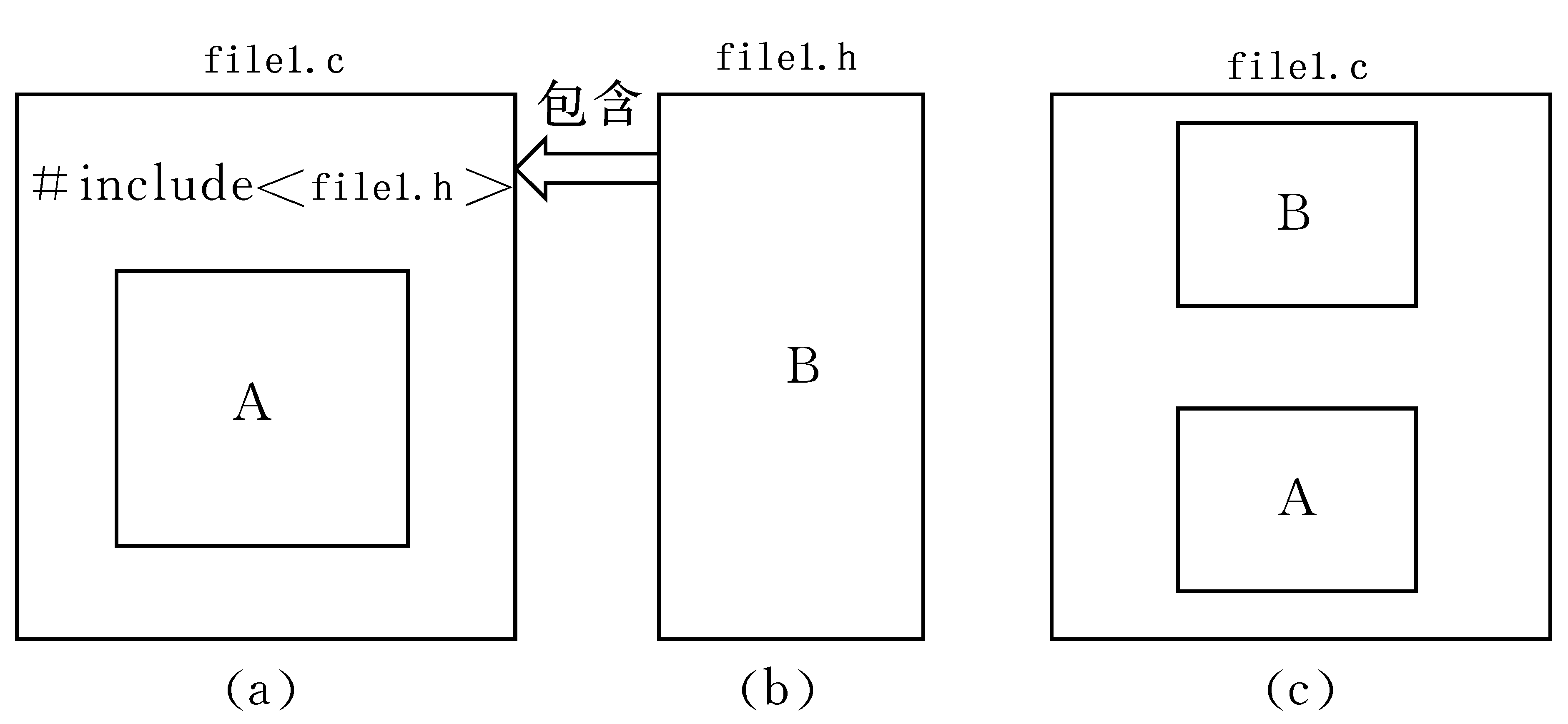
**注意:**

1. 宏的名字中不能有空格，但是在替换的字符串中可以有空格。ANSI C允许在参数列表中使用空格；
2. 用括号括住每一个参数，并括住宏的整体定义。
3. 用大写字母表示宏的函数名。
4. 如果打算宏代替函数来加快程序运行速度。假如在程序中只使用一次宏对程序的运行时间没有太大提高。

### 8.2 文件包含处理

#### 8.2.1 文件包含处理

“文件包含处理”是指一个源文件可以将另外一个文件的全部内容包含进来。Ｃ语言提供了#include命令用来实现“文件包含”的操作。



#### 8.2.2 #incude<>和#include""区别

* "" 表示系统先在file1.c所在的当前目录找file1.h，如果找不到，再按系统指定的目录检索。
* < > 表示系统直接按系统指定的目录检索。

注意：

1. #include <>常用于包含库函数的头文件；

2. #include ""常用于包含自定义的头文件；

3. 理论上#include可以包含任意格式的文件(.c .h等) ，但一般用于头文件的包含；

## 9、指针

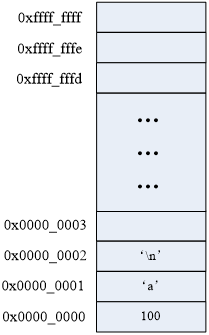
### 9.1 地址与指针变量

#### 9.1.1 内存地址

* 将内存抽象成一个很大的一维字符数组。
* 编码就是对内存的每一个字节分配一个32位或64位的编号（与32位或者64位处理器相关）。
* 这个内存编号我们称之为内存地址。

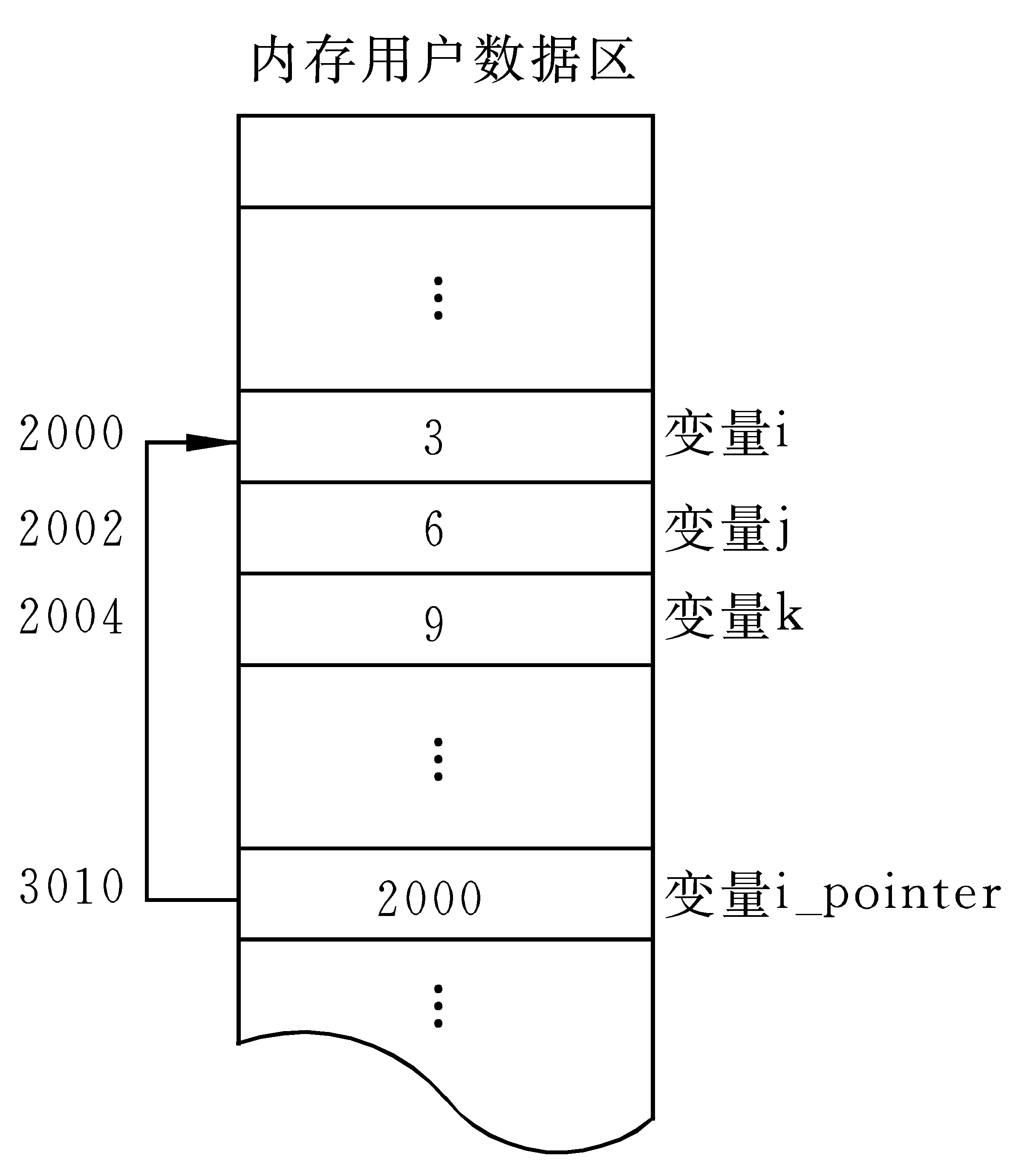
内存中的每一个数据都会分配相应的地址：

* char:占一个字节分配一个地址
* int: 占四个字节分配四个地址
* float、struct、函数、数组等



#### 9.1.2 指针和指针变量

* 内存区的每一个字节都有一个编号，这就是“地址”。
* 如果在程序中定义了一个变量，在对程序进行编译或运行时，系统就会给这个变量分配内存单元，并确定它的内存地址(编号)
* 指针的实质就是内存“地址”。指针就是地址，地址就是指针。
* 指针是内存单元的编号，指针变量是存放地址的变量。
* 通常我们叙述时会把指针变量简称为指针，实际他们含义并不一样。



### 9.2指针引用

#### 9.2.1 指针变量的定义和使用

* 指针也是一种数据类型，指针变量也是一种变量
* 指针变量指向谁，就把谁的地址赋值给指针变量
* “\*”操作符操作的是指针变量指向的内存空间

#include <stdio.h>

int *main*()

{

int a = 0;

char b = 100;

*printf*("%p, %p\n", &a, &b); //打印a, b的地址

//int \*代表是一种数据类型，int\*指针类型，p才是变量名

//定义了一个指针类型的变量，可以指向一个int类型变量的地址

int \*p;

p = &a;//将a的地址赋值给变量p，p也是一个变量，值是一个内存地址编号

*printf*("%d\n", \*p);//p指向了a的地址，\*p就是a的值

char \*p1 = &b;

*printf*("%c\n", \*p1);//\*p1指向了b的地址，\*p1就是b的值

return 0;

}

注意：&可以取得一个变量在内存中的地址。但是，不能取寄存器变量，因为寄存器变量不在内存里，而在CPU里面，所以是没有地址的。

#### 9.2.2 通过指针间接修改变量的值

int a = 0;

int b = 11;

int \*p = &a;

\*p = 100;

*printf*("a = %d, \*p = %d\n", a, \*p);

p = &b;

\*p = 22;

*printf*("b = %d, \*p = %d\n", b, \*p);

### 9.3 用指针作函数参数

#### 9.3.1 函数形参改变实参的值

#include <stdio.h>

void swap1(int x, int y)

{

int tmp;

tmp = x;

x = y;

y = tmp;

*printf*("x = %d, y = %d\n", x, y);

}

void swap2(int \*x, int \*y)

{

int tmp;

tmp = \*x;

\*x = \*y;

\*y = tmp;

}

int *main*()

{

int a = 3;

int b = 5;

swap1(a, b); //值传递

*printf*("a = %d, b = %d\n", a, b);

a = 3;

b = 5;

swap2(&a, &b); //地址传递

*printf*("a2 = %d, b2 = %d\n", a, b);

return 0;

}

#### 9.3.2 数组名做函数参数

数组名做函数参数，函数的形参会退化为指针：

#include <stdio.h>

void printArrary(int \*a, int n)

{

int i = 0;

for (i = 0; i < n; i++)

{

*printf*("%d, ", a[i]);

}

*printf*("\n");

}

int *main*()

{

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

int n = sizeof(a) / sizeof(a[0]);

//数组名做函数参数

printArrary(a, n);

return 0;

}

### 9.4 返回地址值的函数

#include <stdio.h>

int a = 10;

int \*getA()

{

return &a;

}

int *main*()

{

\*( getA() ) = 111;

*printf*("a = %d\n", a);

return 0;

}

### 9.5 指针数组,指向指针的指针

#### 9.5.1 数组名

数组名字是数组的首元素地址，但它是一个常量：

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

*printf*("a = %p\n", a);

*printf*("&a[0] = %p\n", &a[0]);

//a = 10; //err, 数组名只是常量，不能修改

#### 9.5.2 指针操作数组元素

#include <stdio.h>

int *main*()

{

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

int i = 0;

int n = sizeof(a) / sizeof(a[0]);

for (i = 0; i < n; i++)

{

//printf("%d, ", a[i]);

*printf*("%d, ", \*(a+i));

}

*printf*("\n");

int \*p = a; //定义一个指针变量保存a的地址

for (i = 0; i < n; i++)

{

p[i] = 2 \* i;

}

for (i = 0; i < n; i++)

{

*printf*("%d, ", \*(p + i));

}

*printf*("\n");

return 0;

}

#### 9.5.3 指针加减运算

##### 1) 加法运算

* 指针计算不是简单的整数相加
* 如果是一个int \*，+1的结果是增加一个int的大小
* 如果是一个char \*，+1的结果是增加一个char大小

#include <stdio.h>

int *main*()

{

int a;

int \*p = &a;

*printf*("%d\n", p);

p += 2;//移动了2个int

*printf*("%d\n", p);

char b = 0;

char \*p1 = &b;

*printf*("%d\n", p1);

p1 += 2;//移动了2个char

*printf*("%d\n", p1);

return 0;

}

通过改变指针指向操作数组元素：

#include <stdio.h>

int *main*()

{

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

int i = 0;

int n = sizeof(a) / sizeof(a[0]);

int \*p = a;

for (i = 0; i < n; i++)

{

*printf*("%d, ", \*p);

p++;

}

*printf*("\n");

return 0;

}

##### 2) 减法运算

示例1：

#include <stdio.h>

int *main*()

{

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

int i = 0;

int n = sizeof(a) / sizeof(a[0]);

int \*p = a+n-1;

for (i = 0; i < n; i++)

{

*printf*("%d, ", \*p);

p--;

}

*printf*("\n");

return 0;

}

示例2：

#include <stdio.h>

int *main*()

{

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

int \*p2 = &a[2]; //第2个元素地址

int \*p1 = &a[1]; //第1个元素地址

*printf*("p1 = %p, p2 = %p\n", p1, p2);

int n1 = p2 - p1; //n1 = 1

int n2 = (int)p2 - (int)p1; //n2 = 4

*printf*("n1 = %d, n2 = %d\n", n1, n2);

return 0;

}

#### 9.5.4 指针数组

指针数组，它是数组，数组的每个元素都是指针类型。

#include <stdio.h>

int *main*()

{

//指针数组

int \*p[3];

int a = 1;

int b = 2;

int c = 3;

int i = 0;

p[0] = &a;

p[1] = &b;

p[2] = &c;

for (i = 0; i < sizeof(p) / sizeof(p[0]); i++ )

{

*printf*("%d, ", \*(p[i]));

}

*printf*("\n");

return 0;

}

## 10、结构体与共同体

### 10.1 用typedef 说明一个新类型

typedef为C语言的关键字，作用是为一种数据类型(基本类型或自定义数据类型)定义一个新名字，不能创建新类型。

* 与#define不同，typedef仅限于数据类型，而不是能是表达式或具体的值
* #define发生在预处理，typedef发生在编译阶段

#include <stdio.h>

typedef int INT;

typedef char BYTE;

typedef BYTE T\_BYTE;

typedef unsigned char UBYTE;

typedef struct type

{

UBYTE a;

INT b;

T\_BYTE c;

}TYPE, \*PTYPE;

int *main*()

{

TYPE t;

t.a = 254;

t.b = 10;

t.c = 'c';

PTYPE p = &t;

*printf*("%u, %d, %c\n", p->a, p->b, p->c);

return 0;

}

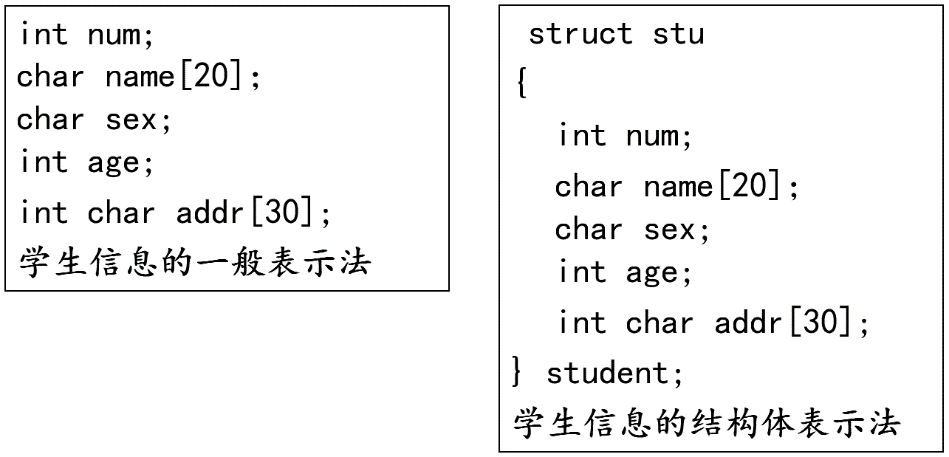
### 10.2 结构体类型数据的定义和成员的引用

#### 10.2.1 概述

数组：描述一组具有相同类型数据的有序集合，用于处理大量相同类型的数据运算。

有时我们需要将不同类型的数据组合成一个有机的整体，如：一个学生有学号/姓名/性别/年龄/地址等属性。显然单独定义以上变量比较繁琐，数据不便于管理。

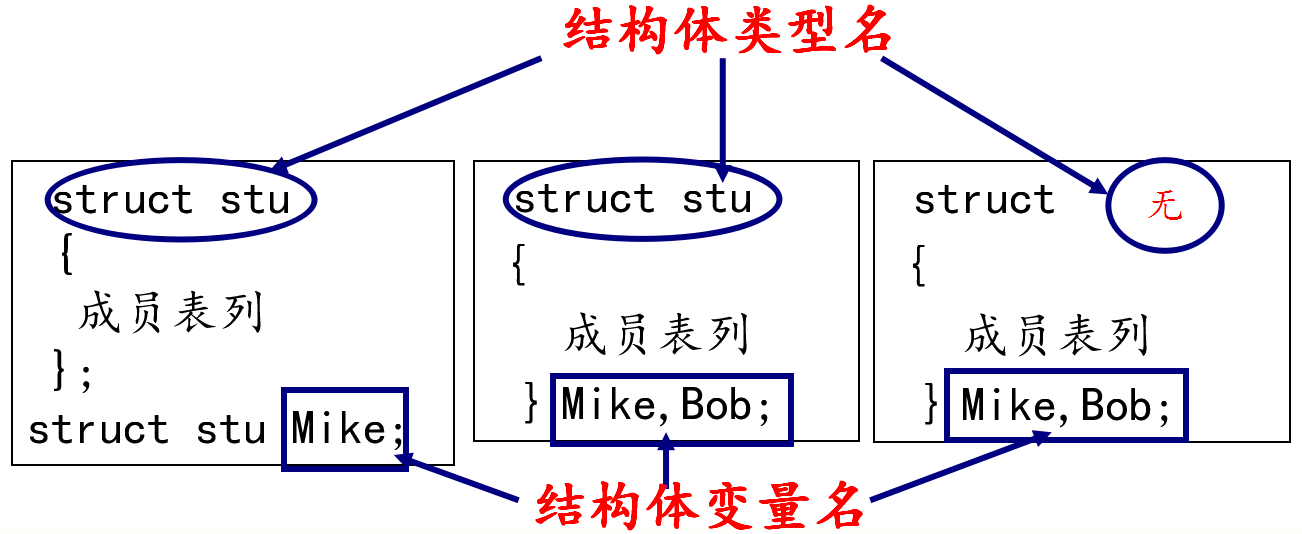
Ｃ语言中给出了另一种构造数据类型——结构体。



#### 10.2.2 结构体变量的定义和初始化

定义结构体变量的方式：

* 先声明结构体类型再定义变量名
* 在声明类型的同时定义变量
* 直接定义结构体类型变量（无类型名）



结构体类型和结构体变量关系：

* 结构体类型：指定了一个结构体类型，它相当于一个模型，但其中并无具体数据，系统对之也不分配实际内存单元。
* 结构体变量：系统根据结构体类型（内部成员状况）为之分配空间。

//结构体类型的定义

struct stu

{

char name[50];

int age;

};

//先定义类型，再定义变量（常用）

struct stu s1 = { "mike", 18 };

//定义类型同时定义变量

struct stu2

{

char name[50];

int age;

}s2 = { "lily", 22 };

struct

{

char name[50];

int age;

}s3 = { "yuri", 25 };

#### 10.2.3 结构体成员的使用

#include<stdio.h>

#include<string.h>

//结构体类型的定义

struct stu

{

char name[50];

int age;

};

int *main*()

{

struct stu s1;

//如果是普通变量，通过点运算符操作结构体成员

*strcpy*(s1.name, "abc");

s1.age = 18;

*printf*("s1.name = %s, s1.age = %d\n", s1.name, s1.age);

//如果是指针变量，通过->操作结构体成员

*strcpy*((&s1)->name, "test");

(&s1)->age = 22;

*printf*("(&s1)->name = %s, (&s1)->age = %d\n", (&s1)->name, (&s1)->age);

return 0;

}

#### 10.2.4 结构体数组

#include <stdio.h>

//统计学生成绩

struct stu

{

int num;

char name[20];

char sex;

float score;

};

int *main*()

{

//定义一个含有5个元素的结构体数组并将其初始化

struct stu boy[5] = {

{ 101, "Li ping", 'M', 45 },

{ 102, "Zhang ping", 'M', 62.5 },

{ 103, "He fang", 'F', 92.5 },

{ 104, "Cheng ling", 'F', 87 },

{ 105, "Wang ming", 'M', 58 }};

int i = 0;

int c = 0;

float ave, s = 0;

for (i = 0; i < 5; i++)

{

s += boy[i].score; //计算总分

if (boy[i].score < 60)

{

c += 1; //统计不及格人的分数

}

}

*printf*("s=%f\n", s);//打印总分数

ave = s / 5; //计算平均分数

*printf*("average=%f\ncount=%d\n\n", ave, c); //打印平均分与不及格人数

for (i = 0; i < 5; i++)

{

*printf*(" name=%s, score=%f\n", boy[i].name, boy[i].score);

// printf(" name=%s, score=%f\n", (boy+i)->name, (boy+i)->score);

}

return 0;

}

#### 10.2.5 结构体套结构体

#include <stdio.h>

struct person

{

char name[20];

char sex;

};

struct stu

{

int id;

struct person info;

};

int *main*()

{

struct stu s[2] = { 1, "lily", 'F', 2, "yuri", 'M' };

int i = 0;

for (i = 0; i < 2; i++)

{

*printf*("id = %d\tinfo.name=%s\tinfo.sex=%c\n", s[i].id, s[i].info.name, s[i].info.sex);

}

return 0;

}

#### 10.2.6 结构体赋值

#include<stdio.h>

#include<string.h>

//结构体类型的定义

struct stu

{

char name[50];

int age;

};

int *main*()

{

struct stu s1;

//如果是普通变量，通过点运算符操作结构体成员

*strcpy*(s1.name, "abc");

s1.age = 18;

*printf*("s1.name = %s, s1.age = %d\n", s1.name, s1.age);

//相同类型的两个结构体变量，可以相互赋值

//把s1成员变量的值拷贝给s2成员变量的内存

//s1和s2只是成员变量的值一样而已，它们还是没有关系的两个变量

struct stu s2 = s1;

//memcpy(&s2, &s1, sizeof(s1));

*printf*("s2.name = %s, s2.age = %d\n", s2.name, s2.age);

return 0;

}

### 10.3 共用体(联合体)

* 联合union是一个能在同一个存储空间存储不同类型数据的类型；
* 联合体所占的内存长度等于其最长成员的长度倍数，也有叫做共用体；
* 同一内存段可以用来存放几种不同类型的成员，但每一瞬时只有一种起作用；
* 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的值会被覆盖；
* 共用体变量的地址和它的各成员的地址都是同一地址。

#include <stdio.h>

//共用体也叫联合体

union Test

{

unsigned char a;

unsigned int b;

unsigned short c;

};

int main()

{

//定义共用体变量

union Test tmp;

//1、所有成员的首地址是一样的

printf("%p, %p, %p\n", &(tmp.a), &(tmp.b), &(tmp.c));

//2、共用体大小为最大成员类型的大小

printf("%lu\n", sizeof(union Test));

//3、一个成员赋值，会影响另外的成员

//左边是高位，右边是低位

//低位放低地址，高位放高地址

tmp.b = 0x44332211;

printf("%x\n", tmp.a); //11

printf("%x\n", tmp.c); //2211

tmp.a = 0x00;

printf("short: %x\n", tmp.c); //2200

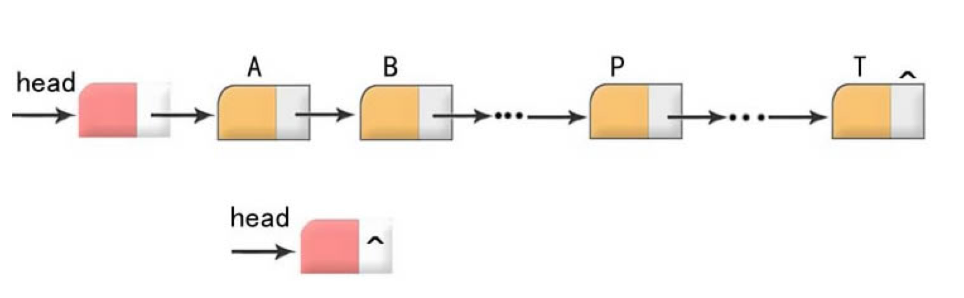
printf("int: %x\n", tmp.b); //44332200

return 0;

}

### 10.4 通过结构体构成链表,单向链表的建立,结点数据的输出、删除与插入

#### 10.4.1 什么是链表



* 链表是一种常用的数据结构，它通过指针将一些列数据结点，连接成一个数据链。相对于数组，链表具有更好的动态性（非顺序存储）链式存储。
* 数据域用来存储数据，指针域用于建立与下一个结点的联系。
* 建立链表时无需预先知道数据总量的，可以随机的分配空间，可以高效的在链表中的任意位置实时插入或删除数据。
* 链表的开销，主要是访问顺序性和组织链的空间损失。

**数组和链表的区别：**

|  |
| --- |
| 数组：一次性分配一块连续的存储区域。  优点：随机访问元素效率高  缺点：1) 需要分配一块连续的存储区域（很大区域，有可能分配失败）  2) 删除和插入某个元素效率低  链表：无需一次性分配一块连续的存储区域，只需分配n块节点存储区域，通过指针建立关系。  优点：1) 不需要一块连续的存储区域  2) 删除和插入某个元素效率高  缺点：随机访问元素效率低 |

#### 10.4.2 创建链表

使用结构体定义节点类型：

|  |
| --- |
| **typedef** struct \_LINKNODE  **{**  int id**;** //数据域  struct \_LINKNODE**\*** next**;** //指针域  **}**link\_node**;** |

编写函数：link\_node\* init\_linklist()

建立带有头结点的单向链表，循环创建结点，结点数据域中的数值从键盘输入，以 -1 作为输入结束标志，链表的头结点地址由函数值返回.

|  |
| --- |
| **typedef** struct \_LINKNODE**{**  int data**;**  struct \_LINKNODE**\*** next**;**  **}**link\_node**;**  link\_node**\*** init\_linklist**(){**    //创建头结点指针  link\_node**\*** head **=** **NULL;**  //给头结点分配内存  head **=** **(**link\_node**\*)**malloc**(sizeof(**link\_node**));**  **if** **(**head **==** **NULL){**  **return** **NULL;**  **}**  head**->**data **=** **-**1**;**  head**->**next **=** **NULL;**  //保存当前节点  link\_node**\*** p\_current **=** head**;**  int data **=** **-**1**;**  //循环向链表中插入节点  **while** **(**1**){**    printf**(**"please input data:\n"**);**  scanf**(**"%d"**,&**data**);**  //如果输入-1，则退出循环  **if** **(**data **==** **-**1**){**  **break;**  **}**  //给新节点分配内存  link\_node**\*** newnode **=** **(**link\_node**\*)**malloc**(sizeof(**link\_node**));**  **if** **(**newnode **==** **NULL){**  **break;**  **}**  //给节点赋值  newnode**->**data **=** data**;**  newnode**->**next **=** **NULL;**  //新节点入链表，也就是将节点插入到最后一个节点的下一个位置  p\_current**->**next **=** newnode**;**  //更新辅助指针p\_current  p\_current **=** newnode**;**  **}**  **return** head**;**  **}** |

#### 10.4.3 遍历链表

编写函数：void foreach\_linklist(link\_node\* head)

顺序输出单向链表各项结点数据域中的内容：

|  |
| --- |
| //遍历链表  void foreach\_linklist**(**link\_node**\*** head**){**  **if** **(**head **==** **NULL){**  **return;**  **}**  //赋值指针变量  link\_node**\*** p\_current **=** head**->**next**;**  **while** **(**p\_current **!=** **NULL){**  printf**(**"%d "**,**p\_current**->**data**);**  p\_current **=** p\_current**->**next**;**  **}**  printf**(**"\n"**);**  **}** |

#### 10.4.4 插入节点

编写函数: void insert\_linklist(link\_node\* head,int val,int data).

在指定值后面插入数据data,如果值val不存在，则在尾部插入。

|  |
| --- |
| //在值val前插入节点  void insert\_linklist**(**link\_node**\*** head**,** int val**,** int data**){**    **if** **(**head **==** **NULL){**  **return;**  **}**  //两个辅助指针  link\_node**\*** p\_prev **=** head**;**  link\_node**\*** p\_current **=** p\_prev**->**next**;**  **while** **(**p\_current **!=** **NULL){**  **if** **(**p\_current**->**data **==** val**){**  **break;**  **}**  p\_prev **=** p\_current**;**  p\_current **=** p\_prev**->**next**;**  **}**  //如果p\_current为NULL，说明不存在值为val的节点  **if** **(**p\_current **==** **NULL){**  printf**(**"不存在值为%d的节点!\n"**,**val**);**  **return;**  **}**  //创建新的节点  link\_node**\*** newnode **=** **(**link\_node**\*)**malloc**(sizeof(**link\_node**));**  newnode**->**data **=** data**;**  newnode**->**next **=** **NULL;**  //新节点入链表  newnode**->**next **=** p\_current**;**  p\_prev**->**next **=** newnode**;**  **}** |

#### 10.4.5 删除节点

编写函数: void remove\_linklist(link\_node\* head,int val)

删除第一个值为val的结点.

|  |
| --- |
| //删除值为val的节点  void remove\_linklist**(**link\_node**\*** head**,**int val**){**  **if** **(**head **==** **NULL){**  **return;**  **}**  //辅助指针  link\_node**\*** p\_prev **=** head**;**  link\_node**\*** p\_current **=** p\_prev**->**next**;**  //查找值为val的节点  **while** **(**p\_current **!=** **NULL){**  **if** **(**p\_current**->**data **==** val**){**  **break;**  **}**  p\_prev **=** p\_current**;**  p\_current **=** p\_prev**->**next**;**  **}**  //如果p\_current为NULL，表示没有找到  **if** **(**p\_current **==** **NULL){**  **return;**  **}**    //删除当前节点： 重新建立待删除节点(p\_current)的前驱后继节点关系  p\_prev**->**next **=** p\_current**->**next**;**  //释放待删除节点的内存  free**(**p\_current**);**  **}** |

#### 10.4.6 销毁链表

编写函数: void destroy\_linklist(link\_node\* head)

销毁链表，释放所有节点的空间.

|  |
| --- |
| //销毁链表  void destroy\_linklist**(**link\_node**\*** head**){**  **if** **(**head **==** **NULL){**  **return;**  **}**  //赋值指针  link\_node**\*** p\_current **=** head**;**  **while** **(**p\_current **!=** **NULL){**  //缓存当前节点下一个节点  link\_node**\*** p\_next **=** p\_current**->**next**;**  free**(**p\_current**);**  p\_current **=** p\_next**;**  **}**  **}** |

## 11、位运算

### 11.1 位运算符的含义和使用

可以使用C对变量中的个别位进行操作。您可能对人们想这样做的原因感到奇怪。这种能力有时确实是必须的，或者至少是有用的。C提供位的逻辑运算符和移位运算符。在以下例子中，我们将使用二进制计数法写出值，以便您可以了解对位发生的操作。在一个实际程序中，您可以使用一般的形式的整数变量或常量。例如不适用00011001的形式，而写为25或者031或者0x19.在我们的例子中，我们将使用8位数字，从左到右，每位的编号是7到0。

#### 11.1.1 位逻辑运算符

4个位运算符用于整型数据，包括char.将这些位运算符成为位运算的原因是它们对每位进行操作，而不影响左右两侧的位。请不要将这些运算符与常规的逻辑运算符(&& 、||和!)相混淆，常规的位的逻辑运算符对整个值进行操作。

#### 11.1.2 按位取反~

一元运算符~将每个1变为0，将每个0变为1，如下面的例子：

|  |
| --- |
| **~(**10011010**)**  01100101 |

假设a是一个unsigned char，已赋值为2.在二进制中，2是00000010.于是-a的值为11111101或者253。请注意该运算符不会改变a的值，a仍为2。

|  |
| --- |
| unsigned char a **=** 2**;** //00000010  unsigned char b **=** **~**a**;** //11111101  printf**(**"ret = %d\n"**,** a**);** //ret = 2  printf**(**"ret = %d\n"**,** b**);** //ret = 253 |

#### 11.1.3 位与（AND）: &

二进制运算符&通过对两个操作数逐位进行比较产生一个新值。对于每个位，只有两个操作数的对应位都是1时结果才为1。

|  |
| --- |
| **(**10010011**)**  **&** **(**00111101**)**  **=** **(**00010001**)** |

C也有一个组合的位与-赋值运算符：&=。下面两个将产生相同的结果：

|  |
| --- |
| val **&=** 0377  val **=** val **&** 0377 |

#### 11.1.4 位或（OR）: |

二进制运算符|通过对两个操作数逐位进行比较产生一个新值。对于每个位，如果其中任意操作数中对应的位为1，那么结果位就为1.

|  |
| --- |
| **(**10010011**)**  **|** **(**00111101**)**  **=** **(**10111111**)** |

C也有组合位或-赋值运算符： |=

|  |
| --- |
| val **|=** 0377  val **=** val **|** 0377 |

#### 11.1.5 位异或: ^

二进制运算符^对两个操作数逐位进行比较。对于每个位，如果操作数中的对应位有一个是1(但不是都是1)，那么结果是1.如果都是0或者都是1，则结果位0.

|  |
| --- |
| **(**10010011**)**  **^** **(**00111101**)**  **=** **(**10101110**)** |

C也有一个组合的位异或-赋值运算符： ^=

|  |
| --- |
| val **^=** 0377  val **=** val **^** 0377 |

现在让我们了解一下C的移位运算符。移位运算符将位向左或向右移动。同样，我们仍将明确地使用二进制形式来说明该机制的工作原理。

#### 11.1.6 左移 <<

左移运算符<<将其左侧操作数的值的每位向左移动，移动的位数由其右侧操作数指定。空出来的位用0填充，并且丢弃移出左侧操作数末端的位。在下面例子中，每位向左移动两个位置。

|  |
| --- |
| **(**10001010**)** **<<** 2  **(**00101000**)** |

该操作将产生一个新位置，但是不改变其操作数。

|  |
| --- |
| 1 **<<** 1 **=** 2**;**  2 **<<** 1 **=** 4**;**  4 **<<** 1 **=** 8**;**  8 **<<** 2 **=** 32 |

左移一位相当于原值\*2.

#### 11.1.7 右移 >>

右移运算符>>将其左侧的操作数的值每位向右移动，移动的位数由其右侧的操作数指定。丢弃移出左侧操作数有段的位。对于unsigned类型，使用0填充左端空出的位。对于有符号类型，结果依赖于机器。空出的位可能用0填充，或者使用符号(最左端)位的副本填充。

|  |
| --- |
| //有符号值  **(**10001010**)** **>>** 2  **(**00100010**)** //在某些系统上的结果值  **(**10001010**)** **>>** 2  **(**11100010**)** //在另一些系统上的解雇  //无符号值  **(**10001010**)** **>>** 2  **(**00100010**)** //所有系统上的结果值 |

### 11.2 简单的位运算

#### 11.2.1 打开位

已知：10011010：

将位2打开

flag | 10011010

|  |
| --- |
| **(**10011010**)**  **|(**00000100**)**  **=(**10011110**)** |

将所有位打开。

flag | ~flag

|  |
| --- |
| **(**10011010**)**  **|(**01100101**)**  **=(**11111111**)** |

#### 11.2.2 关闭位

flag & ~flag

|  |
| --- |
| **(**10011010**)**  **&(**01100101**)**  **=(**00000000**)** |

#### 11.2.3 转置位

转置(toggling)一个位表示如果该位打开，则关闭该位；如果该位关闭，则打开。您可以使用位异或运算符来转置。其思想是如果b是一个位(1或0)，那么如果b为1则b^1为0，如果b为0，则1^b为1。无论b的值是0还是1,0^b为b.

flag ^ 0xff

|  |
| --- |
| **(**10010011**)**  **^(**11111111**)**  **=(**01101100**)** |

#### 11.2.4 交换两个数不需要临时变量

|  |
| --- |
| //a ^ b = temp;  //a ^ temp = b;  //b ^ temp = a  **(**10010011**)**  **^(**00100110**)**  **=(**10110101**)**  **(**10110101**)**  **^(**00100110**)**  10010011    int a **=** 10**;**  int b **=** 30**;** |

#### 11.2.5移位运算符

移位运算符能够提供快捷、高效（依赖于硬件）对2的幂的乘法和除法。

|  |  |
| --- | --- |
| number << n | number乘以2的n次幂 |
| number >> n | 如果number非负，则用number除以2的n次幂 |

## 12、文件操作

### 12.1 文件类型指针(FILE类型指针)

在C语言中用一个指针变量指向一个文件，这个指针称为文件指针。

typedef struct

{

short level; //缓冲区"满"或者"空"的程度

unsigned flags; //文件状态标志

char fd; //文件描述符

unsigned char hold; //如无缓冲区不读取字符

short bsize; //缓冲区的大小

unsigned char \*buffer;//数据缓冲区的位置

unsigned ar; //指针，当前的指向

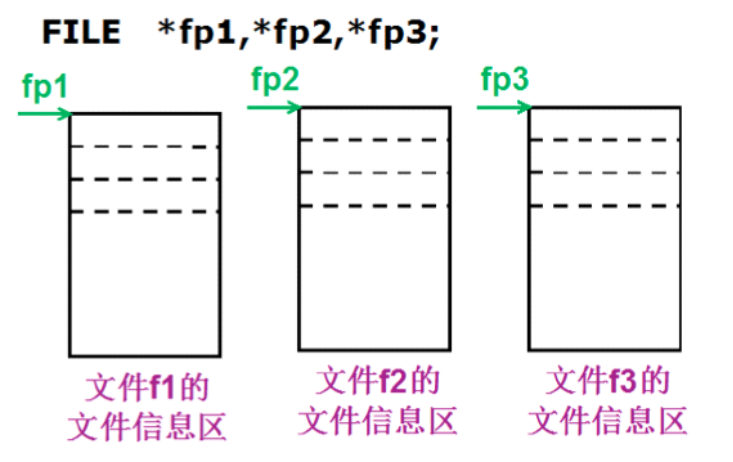
unsigned istemp; //临时文件，指示器

short token; //用于有效性的检查

}FILE;

FILE是系统使用typedef定义出来的有关文件信息的一种结构体类型，结构中含有文件名、文件状态和文件当前位置等信息。

声明FILE结构体类型的信息包含在头文件“stdio.h”中，一般设置一个指向FILE类型变量的指针变量，然后通过它来引用这些FILE类型变量。通过文件指针就可对它所指的文件进行各种操作。



C语言中有三个特殊的文件指针由系统默认打开，用户无需定义即可直接使用:

* stdin： 标准输入，默认为当前终端（键盘），我们使用的scanf、getchar函数默认从此终端获得数据。
* stdout：标准输出，默认为当前终端（屏幕），我们使用的printf、puts函数默认输出信息到此终端。
* stderr：标准出错，默认为当前终端（屏幕），我们使用的perror函数默认输出信息到此终端。

### 12.2 文件的打开与关闭(fopen, fclose)

#### 12.2.1 文件的打开

任何文件使用之前必须打开：

#include <stdio.h>

**FILE \* fopen(const char \* filename, const char \* mode);**

功能：打开文件

参数：

filename：需要打开的文件名，根据需要加上路径

mode：打开文件的模式设置

返回值：

成功：文件指针

失败：NULL

第一个参数的几种形式:

*FILE* \*fp\_passwd = *NULL*;

//相对路径：

//打开当前目录passdw文件：源文件(源程序)所在目录

*FILE* \*fp\_passwd = *fopen*("passwd.txt", "r");

//打开当前目录(test)下passwd.txt文件

fp\_passwd = *fopen*(". / test / passwd.txt", "r");

//打开当前目录上一级目录（相对当前目录）passwd.txt文件

fp\_passwd = *fopen*(".. / passwd.txt", "r");

//绝对路径：

//打开C盘test目录下一个叫passwd.txt文件

fp\_passwd = *fopen*("c:/test/passwd.txt","r");

第二个参数的几种形式(打开文件的方式)：

| **打开模式** | **含义** |
| --- | --- |
| r或rb | 以只读方式打开一个文本文件（不创建文件，若文件不存在则报错） |
| w或wb | 以写方式打开文件(如果文件存在则清空文件，文件不存在则创建一个文件) |
| a或ab | 以追加方式打开文件，在末尾添加内容，若文件不存在则创建文件 |
| r+或rb+ | 以可读、可写的方式打开文件(不创建新文件) |
| w+或wb+ | 以可读、可写的方式打开文件(如果文件存在则清空文件，文件不存在则创建一个文件) |
| a+或ab+ | 以添加方式打开文件，打开文件并在末尾更改文件,若文件不存在则创建文件 |

注意：

* b是二进制模式的意思，b只是在Windows有效，在Linux用r和rb的结果是一样的
* Unix和Linux下所有的文本文件行都是\n结尾，而Windows所有的文本文件行都是\r\n结尾
* 在Windows平台下，以“文本”方式打开文件，不加b：
  + 当读取文件的时候，系统会将所有的 "\r\n" 转换成 "\n"
  + 当写入文件的时候，系统会将 "\n" 转换成 "\r\n" 写入
  + 以"二进制"方式打开文件，则读\n写都不会进行这样的转换
* 在Unix/Linux平台下，“文本”与“二进制”模式没有区别，"\r\n" 作为两个字符原样输入输出

int main(void)

{

FILE \*fp = NULL;

// "\\"这样的路径形式，只能在windows使用

// "/"这样的路径形式，windows和linux平台下都可用，建议使用这种

// 路径可以是相对路径，也可是绝对路径

fp = fopen("../test", "w");

//fp = fopen("..\\test", "w");

if (fp == NULL) //返回空，说明打开失败

{

//perror()是标准出错打印函数，能打印调用库函数出错原因

perror("open");

return -1;

}

return 0;

}

#### 12.2.2 文件的关闭

任何文件在使用后应该关闭：

* 打开的文件会占用内存资源，如果总是打开不关闭，会消耗很多内存
* 一个进程同时打开的文件数是有限制的，超过最大同时打开文件数，再次调用fopen打开文件会失败
* 如果没有明确的调用fclose关闭打开的文件，那么程序在退出的时候，操作系统会统一关闭。

#include <stdio.h>

**int fclose(FILE \* stream);**

功能：关闭先前fopen()打开的文件。此动作让缓冲区的数据写入文件中，并释放系统所提供的文件资源。

参数：

stream：文件指针

返回值：

成功：0

失败：-1

*FILE* \* fp = *NULL*;

fp = *fopen*("abc.txt", "r");

*fclose*(fp);

### 12.3 文件的读写

#### 12.3.1 按照字符读写文件fgetc、fputc

##### 1）写文件

#include <stdio.h>

**int fputc(int ch, FILE \* stream);**

功能：将ch转换为unsigned char后写入stream指定的文件中

参数：

ch：需要写入文件的字符

stream：文件指针

返回值：

成功：成功写入文件的字符

失败：返回-1

char buf[] = "this is a test for fputc";

int i = 0;

int n = strlen(buf);

for (i = 0; i < n; i++)

{

//往文件fp写入字符buf[i]

int ch = fputc(buf[i], fp);

printf("ch = %c\n", ch);

}

##### 2）文件结尾

在C语言中，EOF表示文件结束符(end of file)。在while循环中以EOF作为文件结束标志，这种以EOF作为文件结束标志的文件，必须是文本文件。在文本文件中，数据都是以字符的ASCII代码值的形式存放。我们知道，ASCII代码值的范围是0~127，不可能出现-1，因此可以用EOF作为文件结束标志。

#define *EOF* (-1)

当把数据以二进制形式存放到文件中时，就会有-1值的出现，因此不能采用EOF作为二进制文件的结束标志。为解决这一个问题，ANSI C提供一个feof函数，用来判断文件是否结束。feof函数既可用以判断二进制文件又可用以判断文本文件。

#include <stdio.h>

**int feof(FILE \* stream);**

功能：检测是否读取到了文件结尾。判断的是最后一次“读操作的内容”，不是当前位置内容(上一个内容)。

参数：

stream：文件指针

返回值：

非0值：已经到文件结尾

0：没有到文件结尾

##### 3）读文件

#include <stdio.h>

**int fgetc(FILE \* stream);**

功能：从stream指定的文件中读取一个字符

参数：

stream：文件指针

返回值：

成功：返回读取到的字符

失败：-1

char ch;

#if 0

while ((ch = fgetc(fp)) != EOF)

{

printf("%c", ch);

}

printf("\n");

#endif

while (!feof(fp)) //文件没有结束，则执行循环

{

ch = fgetc(fp);

printf("%c", ch);

}

printf("\n");

#### 12.3.2 按照行读写文件fgets、fputs

##### 1）写文件

#include <stdio.h>

**int fputs(const char \* str, FILE \* stream);**

功能：将str所指定的字符串写入到stream指定的文件中，字符串结束符 '\0' 不写入文件。

参数：

str：字符串

stream：文件指针

返回值：6

成功：0

失败：-1

char \*buf[] = { "123456\n", "bbbbbbbbbb\n", "ccccccccccc\n" };

int i = 0;

int n = 3;

for (i = 0; i < n; i++)

{

int len = fputs(buf[i], fp);

printf("len = %d\n", len);

}

##### 2）读文件

#include <stdio.h>

**char \* fgets(char \* str, int size, FILE \* stream);**

功能：从stream指定的文件内读入字符，保存到str所指定的内存空间，直到出现换行字符、读到文件结尾或是已读了size - 1个字符为止，最后会自动加上字符 '\0' 作为字符串结束。

参数：

str：字符串

size：指定最大读取字符串的长度（size - 1）

stream：文件指针

返回值：

成功：成功读取的字符串

读到文件尾或出错： NULL

char buf[100] = 0;

while (!feof(fp)) //文件没有结束

{

memset(buf, 0, sizeof(buf));

char \*p = fgets(buf, sizeof(buf), fp);

if (p != NULL)

{

printf("buf = %s", buf);

}

}

#### 12.3.3 按照格式化文件fprintf、fscanf

##### 1）写文件

#include <stdio.h>

**int fprintf(FILE \* stream, const char \* format, ...);**

功能：根据参数format字符串来转换并格式化数据，然后将结果输出到stream指定的文件中，指定出现字符串结束符 '\0' 为止。

参数：

stream：已经打开的文件

format：字符串格式，用法和printf()一样

返回值：

成功：实际写入文件的字符个数

失败：-1

fprintf(fp, "%d %d %d\n", 1, 2, 3);

##### 2）读文件

#include <stdio.h>

**int fscanf(FILE \* stream, const char \* format, ...);**

功能：从stream指定的文件读取字符串，并根据参数format字符串来转换并格式化数据。

参数：

stream：已经打开的文件

format：字符串格式，用法和scanf()一样

返回值：

成功：参数数目，成功转换的值的个数

失败： - 1

int a = 0;

int b = 0;

int c = 0;

fscanf(fp, "%d %d %d\n", &a, &b, &c);

printf("a = %d, b = %d, c = %d\n", a, b, c);

#### 12.3.4 按照块读写文件fread、fwrite

##### 1）写文件

#include <stdio.h>

**size\_t fwrite(const void \*ptr, size\_t size, size\_t nmemb, FILE \*stream);**

功能：以数据块的方式给文件写入内容

参数：

ptr：准备写入文件数据的地址

size： size\_t 为 unsigned int类型，此参数指定写入文件内容的块数据大小

nmemb：写入文件的块数，写入文件数据总大小为：size \* nmemb

stream：已经打开的文件指针

返回值：

成功：实际成功写入文件数据的块数目，此值和 nmemb 相等

失败：0

typedef struct Stu

{

char name[50];

int id;

}Stu;

Stu s[3];

int i = 0;

for (i = 0; i < 3; i++)

{

sprintf(s[i].name, "stu%d%d%d", i, i, i);

s[i].id = i + 1;

}

int ret = fwrite(s, sizeof(Stu), 3, fp);

printf("ret = %d\n", ret);

##### 2）读文件

#include <stdio.h>

**size\_t fread(void \*ptr, size\_t size, size\_t nmemb, FILE \*stream);**

功能：以数据块的方式从文件中读取内容

参数：

ptr：存放读取出来数据的内存空间

size： size\_t 为 unsigned int类型，此参数指定读取文件内容的块数据大小

nmemb：读取文件的块数，读取文件数据总大小为：size \* nmemb

stream：已经打开的文件指针

返回值：

成功：实际成功读取到内容的块数，如果此值比nmemb小，但大于0，说明读到文件的结尾。

失败：0

typedef struct Stu

{

char name[50];

int id;

}Stu;

Stu s[3];

int ret = fread(s, sizeof(Stu), 3, fp);

printf("ret = %d\n", ret);

int i = 0;

for (i = 0; i < 3; i++)

{

printf("s = %s, %d\n", s[i].name, s[i].id);

}

### 12.4文件的定位

#include <stdio.h>

**int fseek(FILE \*stream, long offset, int whence);**

功能：移动文件流（文件光标）的读写位置。

参数：

stream：已经打开的文件指针

offset：根据whence来移动的位移数（偏移量），可以是正数，也可以负数，如果正数，则相对于whence往右移动，如果是负数，则相对于whence往左移动。如果向前移动的字节数超过了文件开头则出错返回，如果向后移动的字节数超过了文件末尾，再次写入时将增大文件尺寸。

whence：其取值如下：

SEEK\_SET：从文件开头移动offset个字节

SEEK\_CUR：从当前位置移动offset个字节

SEEK\_END：从文件末尾移动offset个字节

返回值：

成功：0

失败：-1

#include <stdio.h>

**long ftell(FILE \*stream);**

功能：获取文件流（文件光标）的读写位置。

参数：

stream：已经打开的文件指针

返回值：

成功：当前文件流（文件光标）的读写位置

失败：-1

#include <stdio.h>

**void rewind(FILE \*stream);**

功能：把文件流（文件光标）的读写位置移动到文件开头。

参数：

stream：已经打开的文件指针

返回值：

无返回值

typedef struct Stu

{

char name[50];

int id;

}Stu;

//假如已经往文件写入3个结构体

//fwrite(s, sizeof(Stu), 3, fp);

Stu s[3];

Stu tmp;

int ret = 0;

//文件光标读写位置从开头往右移动2个结构体的位置

fseek(fp, 2 \* sizeof(Stu), SEEK\_SET);

//读第3个结构体

ret = fread(&tmp, sizeof(Stu), 1, fp);

if (ret == 1)

{

printf("[tmp]%s, %d\n", tmp.name, tmp.id);

}

//把文件光标移动到文件开头

//fseek(fp, 0, SEEK\_SET);

rewind(fp);

ret = fread(s, sizeof(Stu), 3, fp);

printf("ret = %d\n", ret);

int i = 0;

for (i = 0; i < 3; i++)

{

printf("s === %s, %d\n", s[i].name, s[i].id);

}

# 四、 考试方式

## 1、时长

上机考试,考试时长120分钟,满分100分。

## 2、题型及分值

单项选择题40分(含公共基础知识部分10分)、操作题60分(包括填空题、改错题及编程题)。