0311嵌入式

刘彪

刘阳

操作系统

董一弘 弘姐 13783482375

赵长江

挂起耗硬盘

函数的返回值需要用变量接受

几个常见单词

Semicolon:分号，union联盟，dynamic 动态 linked链表

几个常用变量

coun:计数 arr:数组 temp:临时 value:值 row:行 col:列 columns柱

volume 体积 menu：菜单 add：加 sub：减 mul：乘 dvd：除 buffer:缓冲区

message:日志消息 sock：锁定 client：客户 score：分数

Current：当前

Git 的使用

安装和初始化

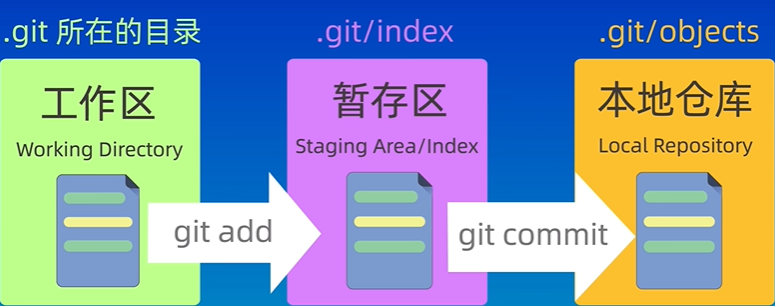
1. git config --global user.name "admin"
2. git config --global user.email 2076625002@qq.com
3. git config --global credential.helper store //保存用户名和密码
5. git config --global --list //查看配置信息

新建库

只需要把一个目录变成git可以管理的仓库

1. git init  //初始化，变为给git仓库
2. ls -a     //查看隐藏文件
3. ls -altr //查看文件的详情信息

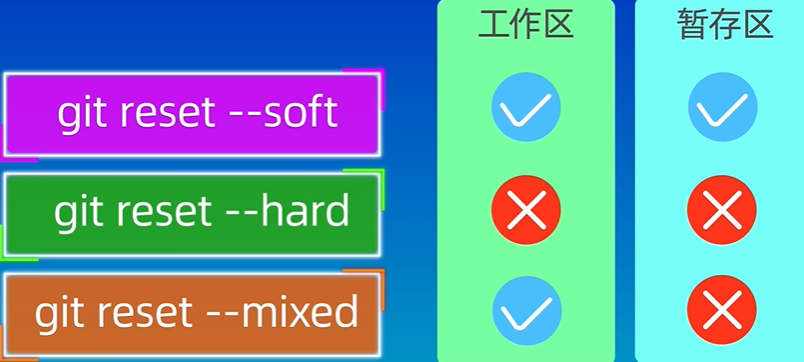
工作区域和文件状态



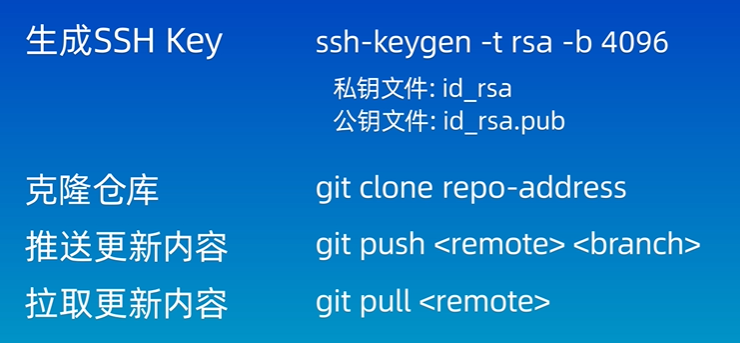
添加和提交文件

1. git  init  //创建仓库
2. git  status  //查看仓库状态
3. git  add  //添加到暂存区 ，可以使用通配符，如 add \*.txt
4. git add . //当前目录所有文件
5. git  commit -m “说明”  //提交，只会提交暂存区的文件
6. git log //查看提交命令
7. git log –oneline //查看简介的提交信息

git reset回退版本



1. git diff //对比工作区和暂存区
2. git ls-files //查看仓库的文件
3. git add file1.txt  //删除暂存区的file1.txt
4. git rm file1.txt



Vim使用

q！强制退出 /home/abc/zdh绝对路径

tree树

i-插前 hjkl-上下左右 a-插后 o-新增下一行

shift+o-新增上一行 shift+g-到最后一行 gg-到第一行

5j-当前下5行 5k-当前上5行

.-重复前次操作 u-撤销上一步 ctrl+r-恢复前次操作

dw-删除单词 cw-改变单词

w-下个单词首部 e-上个单词首部 b-上个单词首部

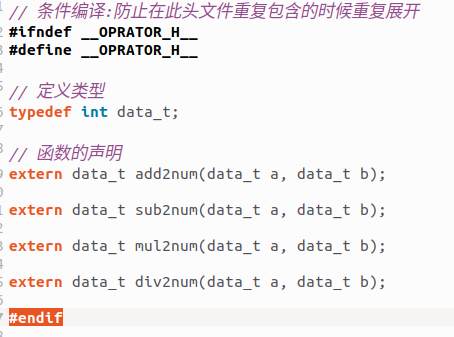
/-搜索 :b%s/旧/新/g-全局替换

yw-复制单词 p-粘贴 ci{-删除

ctrl+v可视化块 shift+v可视化行

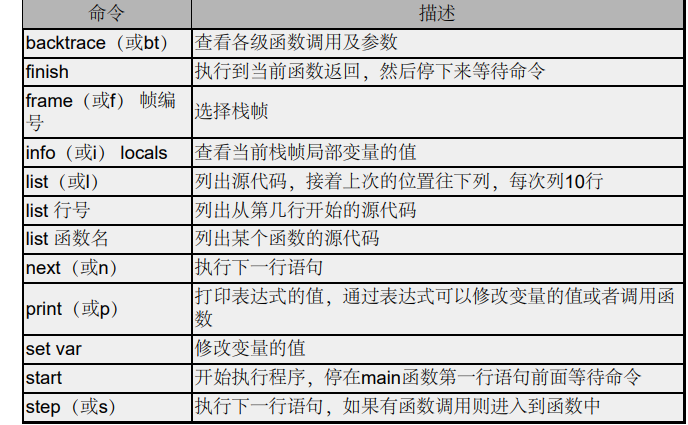
s删除光标字符并插入 shift+s 删除光标行

ctrl+ v +=：可以对齐



调试：在编译时要加上-g选项，生成的可执行文件才能用gdb进行调试：quit退出

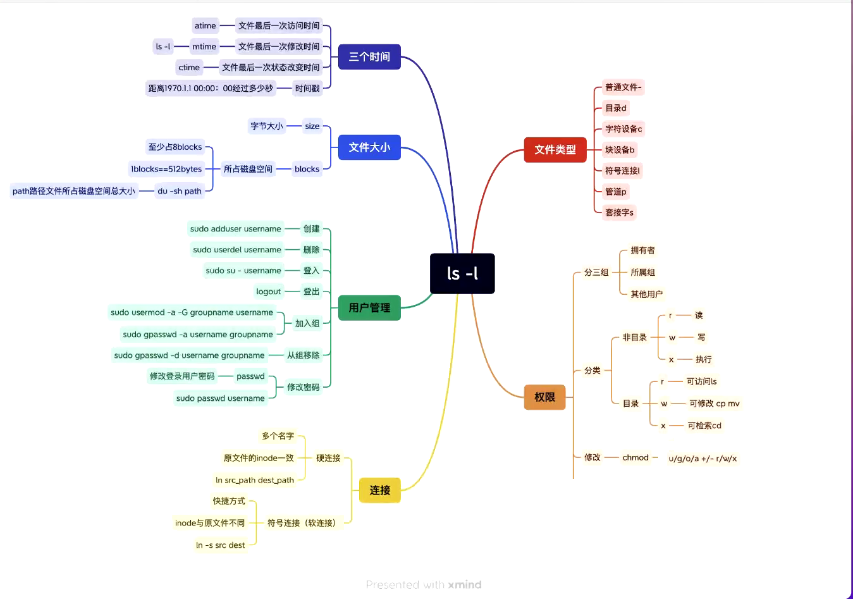




I locals 可以查看局部变量，i及其他的值。



Linux



sudo su----进入root模式，exit退出

home---普通用户目录用于存储个人文件和设置

bin-----二进制文件、

boot------系统启动时所需的所有文件

etc-----配置文件

dev----设备文件

lib------系统运行时共享库文件

media------可移动媒体设备

mnt------临时挂载点，用于挂载其他文件系统。例如，在安装新的文件系统或备份数据时，可以将其挂载到/mnt目录下进行操作。

opt------用于存放第三方应用程序目录

proc---- 虚拟文件系统、包含了当前运行中的进程和系统内核的详细信息。这些信息可以通过读取/proc目录中的文件来获取。

tmp------临时文件目录

usr------系统中许多用户程序和库文件。例如，用户安装的软件和系统自带的应用程序都在/usr目录下

whoami：显示当前会话用户名称 sudo su – user02:进入user02组

cat和less 文件名 查看文件

cd ..上一层目录 cd /根目录

ctrl+shift+加号 放大 ctrl+减号 缩小

mkdir创建目录

rmdir删除空目录 rm -r 删除非空目录

rm 删除文件 ctrl+l 清屏

touch 创建文件

ls -altr：所有文件和目录

man 是 linux 提供的一个手册，包含了绝大部分的命令、函数使用说明

man ascii查ASCII码

man operator所有的运算符

make 文件名 直接编译执行

tar cjvf a1.tar abc a.c b.c ----打包：

tar cjvf a1.tar(打包成什么名字) abc a.c b.c （打包什么东西）

tar xvf 文件名

find -name 文件名 查找文件

find 路径 -name \*文件名--所有带文件名字符的文件

ps aux或者ajx----查看本机进程

安装软件包：sudo apt install 《软件包名》

列出所有可安装的软件包： sudo apt list

列出所有可安装的软件包数量：sudo apt list|wc -l

更新软件包：sudo apt upgrade

vim /usr/src/linux-headers-5.19.0-32-generic/include/linux/list.h：查看内核链表代码

修改vim配置：vim /etc/vim/vimrc

网路配置文件cat /etc/network/interfaces

常用函数

|  |  |  |
| --- | --- | --- |
| 转义字符 | 意义 | ASCII码值（十进制） |
| \a | 响铃(BEL) | 7 |
| \b | 退格(BS) ，将当前位置移到前一列 | 8 |
| \f | 换页(FF)，将当前位置移到下页开头 | 12 |
| \n | 换行(LF) ，将当前位置移到下一行开头 | 10 |
| \r | [回车(CR) ，将当前位置移到本行开头](https://baike.baidu.com/item/%E5%9B%9E%E8%BD%A6/459029?fromModule=lemma_inlink) | 13 |
| \t | 水平制表(HT) （跳到下一个TAB位置） | 9 |
| \v | 垂直制表(VT) | 11 |
| \\ | 代表一个反斜线字符"\" | 92 |
| \' | 代表一个单引号（撇号）字符 | 39 |
| \" | 代表一个双引号字符 | 34 |
| \? | 代表一个问号 | 63 |
| \0 | 空字符(NUL) | 0 |
| \ddd | [1到3位八进制数所代表的任意字符](https://baike.baidu.com/item/%E5%85%AB%E8%BF%9B%E5%88%B6/4230825?fromModule=lemma_inlink) | 三位八进制 |
| \xhh | [十六进制所代表的任意字符](https://baike.baidu.com/item/%E5%8D%81%E5%85%AD%E8%BF%9B%E5%88%B6/4162457?fromModule=lemma_inlink) | 十六进制 |

清除缓存：setbuf(stdin, NULL);

清空屏幕：头文件#include<stdlib.h>，system(“clear”);

Windows中是system(“cls”)

函数名: sleep ------- 头文件#include <windows.h> /#include <unistd.h>

功 能: 执行挂起指定的秒数---- sleep(50); 秒为单位的

随机数：头文件#include<stdlib.h>，#include<time.h>

srand(time(NULL))：只能放在main函数m

srand((unsigned int)time(NULL))

int ret1 = rand() % 10 + 1;//生成1~10的随机数

int ret2 = rand() % 100 + 1;//生成1~100的随机数

int ret3 = rand() % 34 + 66;//生成66~99的随机数

int ret4 = rand() % (n - m + 1) + m;//生成m~n的随机数

检测按键：kbhit（）-------头文件<conio.h>--while(!kbhit())，不是标准库，非linux

检测键盘是否有键按下。 如果有键按下，则返回对应键值；否则返回零。 kbhit不等待键盘按键。无论有无按键都会立即返回。

Linux即读取键盘字符但是不显示出来，代替conio.h 主要使用 getch () 函数

1. system("stty -echo");
2. c=getchar();
3. system("stty echo");

平方根：头文件#includ<math.h>

x=(int)sqrt(i+100);//强转为int型，默认为浮点型x=sqrt(i);----应该在命令行 执行编译命令时加入 -lm

n次方：// pow函数的作用:求base的n次方

1. int res = pow(3, 3);//3的3次方

isspace()：属于 <ctype.h> 头文件，用于检查一个字符（char）是否为空白字符

main函数参数： int main(int argc, char \*argv[])---------int argc: 这是一个整型变量，表示命令行参数的个数（Argument Count），包括文件名-------char \*argv[]: 这是一个指向字符串，如果在命令行中执行 ./program arg1 arg2，那么 argv[1] 将指向 "arg1"，argv[2]

套接字：网络通信的编程接口

提供了应用程序与网络之间的接口，

使用端口号和IP地址来标识应用程序在网络中的通信终点

常见错误：

{-前出现错误------expected ‘=’, ‘,’, ‘;’, ‘asm’ or ‘\_\_attribute\_\_’ before ‘{’ token

应该在命令行执行编译命令时加入-lm——1.c:(.text+0x33)：对‘某某’未定义的引用

数组作为参数传给函数时，是传给数组的地址，而不是传给整个的数组空间，因而sizeof(a)这句话会报错‘sizeof’ on array function parameter ‘a’ will return size of ‘int \*’

缺少地址：format ‘%d’ expects argument of type ‘int \*’,but argument 2 has type ‘int’

-o写错了：gcc: error: -E or -x required when input is from standard input

数组没限定大小：non-static initialization of a flexible array member

没有主函数：undefined reference to `main'

少一个占位符：too many arguments for format

没有头文件：implicit declaration of function ‘srand’

不兼容的指针类型：passing argument 1 of ‘strlen’ from incompatible pointer type

Tinyhttpd

http请求由三部分组成，分别是：起始行、消息报头、请求正文

请求报文起始行由以下三部分组成

方法（Method）：比如GET、POST、PUT、DELETE等。

请求URI（Request-URI）：资源的统一资源标识符，包括可能的查询字符串。

HTTP版本（HTTP-Version）：如HTTP/1.1。

1. 实例：GET /path/to/page?param1=value1&param2=value2 HTTP/1.1

响应报文起始行（Status Line）： 响应报文起始行由以下三部分组成：

HTTP版本（HTTP-Version）：同请求报文一样，表示使用的HTTP协议版本。

状态码（Status-Code）：三位数字，表示服务器对请求的处理结果。

状态消息（Reason-Phrase）：对状态码的简短描述。

1. 实例：HTTP/1.1 200 OK

套接字用于监听连接并处理客户端请求。

1. //会创建一个用于网络通信的套接字对象，并将其赋值给变量 httpd。
2. httpd = socket(PF\_INET, SOCK\_STREAM, 0);// 是在创建一个套接字
4. //PF\_INET 表示使用 IPv4 协议族。
5. //SOCK\_STREAM 表示创建一个面向流的套接字，用于实现基于 TCP（传输控制协议）的连接。

在编程中，创建套接字通常涉及以下步骤：

1.使用系统调用（如socket()函数）创建套接字。

2.绑定（bind()函数）套接字到本地地址（IP地址和端口号）。

3.对于服务器端）监听（listen()函数）来自客户端的连接请求。

4.对于服务器端）接受（accept()函数）客户端的连接请求，得到新的通信套接字。

5.对于客户端）连接（connect()函数）到服务器的套接字。

6.使用send()和recv()（或其他相关函数）进行数据的发送和接收。

最后，使用close()函数关闭不再需要的套接字。

套接字是用来收发数据的接口，它关联了通信双方的地址信息（如IP地址和端口号），并且提供了读写数据的接口

网路编程函数：

struct sockaddr\_in：处理网络通信的地址，[头文件](https://so.csdn.net/so/search?q=%E5%A4%B4%E6%96%87%E4%BB%B6&spm=1001.2101.3001.7020)#include <sys/socket.h>

主要功能：用来表示和存储IPv4网络地址

socklen\_t 主要用于表示套接字地址结构，用于确保在各种平台上都能正确表示结构体的长度，头文件：<sys/socket.h>

struct stat：它用于存储文件或文件系统对象的状态信息

原码反码补码：

计算机只能做加法运算, 不能做减法和乘除法

一个负数的反码, 是将该数的原码除了符号位以外的其它位取反

一个负数的补码, 就是它的反码 + 1

例如: -12的原码、反码和补码分别为

0000 0000 0000 0000 0000 0000 0000 1100 // 12二进制

1000 0000 0000 0000 0000 0000 0000 1100 // -12原码

1111 1111 1111 1111 1111 1111 1111 0011 // -12反码

1111 1111 1111 1111 1111 1111 1111 0100 // -12补码

正数的原码、反码和补码都是它的二进制

例如: 12的原码、反码和补码分别为

0000 0000 0000 0000 0000 0000 0000 1100

0000 0000 0000 0000 0000 0000 0000 1100

0000 0000 0000 0000 0000 0000 0000 1100

因为对于减法来说,如果使用原码结果是不正确的,所以才引入了反码

源码如何转换为反码：

符号位不变, 其它位按位取反

反码如何转换为补码：

反码+1

补码如何转换为反码：

补码-1

反码如何转换为源码：

符号位不变, 其它位按位取反

scanf 读取

1.scanf函数会先从输入缓冲区中获取数据, 如果输入缓冲区中没有数据, 那么程序就会阻塞, 等待用户输入数据

2.如果用户输入了数据, scanf函数会将用户输入的数据放到输入缓冲区中, 然后再从输入缓冲区中获取数据

3.如果输入缓冲区中已经有数据了, 那么就不会要求用户输入数据, 会直接从输入缓冲区中获取数据

1. **int** a=0,b=0;
2. **char** str[30];
3. printf("b=%d\n",b);
4. scanf("%d",&a);  //输入100 helloworld
5. scanf("%d",&b);
6. scanf("%s",str);
7. printf("a=%d,b=%d,str=%s\n",a,b,str);
9. 回车后，缓存区内有：100 helloworld，匹配发a匹配，赋值给a。helloworld匹配b，数据类型不同，匹配失败，缓存指针不动，helloworld继续匹配str，匹配成功，输出

setbuf函数:setbuf（stdin，null）;

用于清空输入缓冲区, 可以在所有平台使用(Windows/Linux/Max OS)

每次遇见scanf函数就可以重新输入

scanf("格式控制字符串", 地址列表);

把字符串读入数组不要使用 &

字符 \* 出现意味着赋值屏蔽

scanf格式串中的空白表示“跳过零个或多个空白字符”

%c允许 scanf 函数和 printf 函数对单个字符进行 读/写 (%c前要加空格）读入字符前，scanf 函数不会跳过空白字符

对于 scanf 函数来说，\n 等同于空格，\b等同于回退符

不能写成in=scanf（“%d”,&a)

scanf注意点:

1.scanf函数的第二个参数, 只能接受变量地址, 不能接受变量名称

2.scanf函数的格式化字符串不能以\n结尾, 如果以\n结尾, 那么scanf函数永远 无法结束

3.scanf函数的格式化字符串中,如果除了格式化字符串还有别的字符串, 那么必须 原样输入, 否则接收的数据就不对

1. scanf("num = %i", &num);  //必须输入num = 数字 ——才能对，其他报错

4.scanf函数如果需要同时接收多个非字符类型的数据,可以通过空格,TAB,回车作 为分隔符

1. scanf("%i,%i", &num, &value); //只能输入1个字节
2. scanf("%i%i", &num, &value); //能输入2给字节

printf函数

（C语言不看怎么存, 只看怎么取）

short的长度不能大于int，int的长度不能大于long

char一定为为8位(1字节)，毕竟char是我们编程能用的最小数据类型

字符类型：

char 1字节

short 2字节

int 4字节

long 8字节

float 4字节

double 8字节

signed-有符号标识符 用于明确说明, 当前保存的数据可以是有符号的, 一般情 况下很少使用-2^31 ~ 2^31 - 1

unsigned-无符号标识符 用于明确说明, 当前不能保存有符号的值, 只能保存0和正数--%u 0-2^32

%d 有符号10进制整型

%i 有符号10进制整型

%u 无符号10进制整型

%o 无符号8进制整型

%#x 无符号16进制整型

%#X 无符号16进制整型-------#输出“0x”

%f 单、双精度浮点数(默认保留6位小数)

%lf 双精度保留15-16位

%e / E 以指数形式输出单、双精度浮点数

%g / G 以最短输出宽度,输出单、双精度浮点数

%c 字符

%s 字符串

%p 地址

%[] ：在⽅括号中指定⼀组匹配的字符（⽐如 scanf（“%10s”，name） ），遇到不在集合之中的字符，匹配将会 停⽌

如何输出被类型说明符说明的数据

short --> %hi %hd

long --> %li %ld

long long --> %lli %lld

宽度：

格式: printf("a = %[宽度]类型", a);-----int=999(实际宽度3），%5d（指定宽度）

如果实际宽度>指定宽度输出实际宽度

如果实际宽度<指定宽度，输出空格再输出实际宽度（空空999）

标志：printf("a = %[标志][宽度]类型", a);

+:当输出值为正数时,在输出值前面加上一个+号, 默认不显示

空格:输出值为正数时,在输出值前面加上空格, 为负数时加上负号

-:左对齐, 默认右对齐

0:右对齐时, 用0填充宽度.(默认用空格填充)，即指定宽度>实际宽度，前补0不 补空格

精度：格式: printf("a = %[精度]类型", a);

%.10f:保留小数点后10位。单精度有效是6-7位，超出为垃圾数，双精度是15-16位.

动态指定保留小数位数：格式: printf("a = %.\*f", a);

double a=3.21212; printf("a=%.\*f",2,a);//3.21

长度：格式: printf("a = %[长度]类型", a);

hh -char, h-short int, l-long int , ll-long long int

如:printf("c = %d\n", c); printf("d = %ld\n", d)——l：代表long int

|  |  |  |
| --- | --- | --- |
| 长度 | 修饰类型 | 含义 |
| hh | d、i、o、u、x | 输出char |
| H | d、i、o、u、x | 输出 short int |
| L | d、i、o、u、x | 输出 long int |
| ll | d、i、o、u、x | 输出 long long int |

转义字符：格式: printf("%f%%", 3.1415); //输出%必须添加一个转移字符

printf("%f%%", 3.1415); // 输出结果3.1415%

运算符

单目运算：只有一个操作数 如 : i++;

1. x=10；x+=x-=x-x；//x=20   从右向左

双目运算：有两个操作数 如 : a + b;，双目运算符前后需空格

三目运算： ?:——条件表达式 ?:结果A : 结果B;

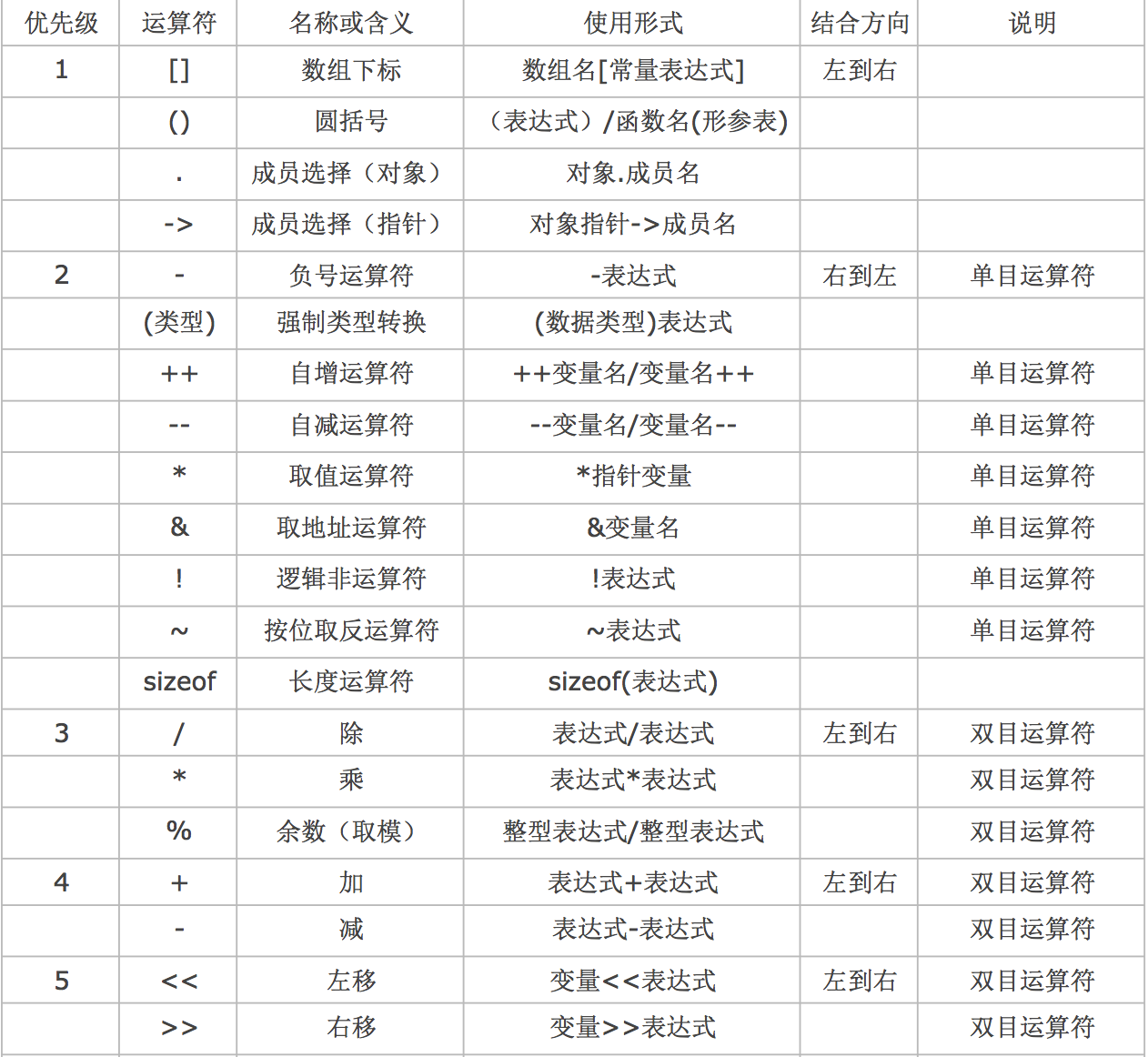
多目运算： 逗号

sizeof可以用来计算一个变量或常量、数据类型所占的内存字节数，标准格式: sizeof(常量 or 变量);

在C语言中逗号“,”也是一种运算符,称为逗号运算符。该运算符的特点是从左到右依次执行每个子表达式，并且整个表达式的值等于最后一个子表达式的值

逗号运算符会从左至右依次取出每个表达式的值, 最后整个逗号表达式的值等于最后一个表达式的值

格式: 表达式1，表达式2，… …，表达式n;





注意： sizeof不是一个函数, 是一个运算符, 该运算符的优先级是2

（type）类型转换类型

类型转换:

|  |  |
| --- | --- |
| **强制类型转换(显示转换)** | **自动类型转换(隐式转换)** |
| (需要转换的类型)(表达式) | 1.算数转换 2.赋值转换 |

强制类型转换(显示转换)

1. // 将double转换为int
2. **int** a = (**int**)10.5;

算数转换:

系统会自动对占用内存较少的类型做一个“自动类型提升”的操作, 先将其转换为当前算数表达式中占用内存高的类型, 然后再参与运算

1. // 当前表达式用1.0占用8个字节, 2占用4个字节
2. // 所以会先将整数类型2转换为double类型之后再计算
3. **double** b = 1.0 / 2;

赋值转换:

1. // 赋值时左边是什么类型,就会自动将右边转换为什么类型再保存
2. **int** a = 10.6;

注意点: 参与计算的是什么类型, 结果就是什么类型, 类型转换并不会影响到原有变量的值(6==6==6//0——6==6//1)

位运算符

&： 二进制数值的每一位，同时为1结果是1，否则是0，从左到右运算，出现0后面 的不再运算

|： 只有要一位为1，结果为1,从左到右运算，出现1后面的不在运算

1. **int** x=0, y=0, z=0;
2. ++x||++y||++z;
3. printf("%d %d %d\n", x, y, z);//1,0,0

^：异或 相同为0，不同为1

~：按位取反 0为1，1为0

注意：取0用|（或），取1为&（与）

按位左移 <<

运算的步骤:左移, 移出去的部分删除, 在最后添0

规律: 左移多少位就是乘以2的多少次幂

注意点: 左移运算可能会改变数值的正负性

按位右移 >>

运算的步骤: 向右移动, 移出去的部分删除,在最高的后面补最高位

规律: 右移多少位就是除以2的多少次幂

1. **int** res = 9 >> 2;
2. printf("res = %i\n", res);  //res=2----9/4=2

利用位运算符,判断奇偶

1. **int** num = 5;
2. **if**((num&1) == 0){//5是0011&(与上)0001=1
3. printf("YES");
4. }**else**{
5. printf("NO");
6. }
7. **return** 0;

进制间转换

十进制转二进制

拼凑：100=2^6+2^5+2^2==1100100；

计算：对2求余，最终将所有余数倒序

二进制转十进制

累加：110001011=2^8+2^7+2^3+2^1+2^0=256+128+8+2+1

二进制转八进制

自右向左每3位为一组，分别对应一个八进制数

11 100 001 010---03412

二进制转十六进制

自右向左每3位一组

0x开头，0~9 a~f --------111 0000 1010 ------0x70a

数组

注意：

1. 定义数组必须放大小或者参数（a[]={1,2,3}）
2. a[],[]放变量不能初始化.
3. 只能在定义数组的时候进行一次性（全部赋值）的初始化
4. 如果只定义了变量, 没有对变量进行初始化, 那么变量中保存的是垃圾数据
5. 如果先定义后初始化, 那么被初始化的元素会变成初始化的数据, 没有被初始化的元素还是垃圾数据
6. 如果定义的同时初始化, 那么被初始化的元素会变成初始化的数据, 没有被初始化的元素会变成0

注意：定义的同时初始化，没有初始化的元素会自动赋值给零

1. **char** ch[5]={'1','2'};//ch[5]={'1','2','0','0','0'}

如果定义数组后,没有初始化,数组中是有值的,是随机的垃圾数,所以如果想要正确使用数组应该要进行初始化。

在新版本的C语言标准中, 数组的元素个数可以是常量, 也可以是变量(不能初始化)

数组名称: 是常量不能自增和变量名称一样, 可以通过数组名称找到数组对应的存储空间, 然后操作这块存储空间

数组的存储细节

和变量一样, 会从内存地址大的开始分配存储空间

和变量一样, 数组的地址就是数组占用内存空间最小的地址

和变量不一样, 给数组每一个元素分配存储空间, 是从数组占用存储空间最小的地址开始分配

1. 分配储存空间——a[0]>a[1]>a[2];

和变量一样, 存储数据也会从每个元素内存地址大的开始存储

1. 存储数据——a[2]>a[1]>a[0]

数组中的sizeof():

对数组使用 sizeof 运算符,数组元素个数×数组数据类型的大小

sizeof(数组名称)得到的是数组占用的总大小

sizeof(数组名称[索引])得到的是数组中每一个元素的大小

数组的长度 = 数组占用的总字节数 / 数组元素占用的字节数—— sizeof(a)/sizeof(int)或者sizeof(a)/sizeof(a[0]);

数组元素作为函数参数：

数组的元素作为函数实参a(a[0](实参))，与同类型的简单变量作为实参一样，如果 是基本数据类型void a(int a（形参）), 那么形参的改变不影响实参

基本数据类型作为函数的参数

\* char int float double

在函数内修改形参, 不会影响到外界的实参

数组名作为函数参数

\*数组名作为函数的参数, 在函数内修改形参, 会影响到外界的实参

由于数组名 == &数组名 == &数组名[0] a==&a==&a[0]==&a[0][0];

所以这个地方相当于把数组的地址传递给了形参

1. change2(arr);  //arr把数组的地址传到change2，change2会根据地址找到arr数组的存储空间，进而进行修改数组的值。

a，数组名是首元素地址

a[0]:是二维数组的第一个元素

a[2][2]={1,2,3,4}--------a[0]={1,2},a[1]={3,2}but 用%s打印a[0]=1234;

数组名作函数参数的注意点

在函数形参表中,允许不给出形参数组的长度

当数组名作为函数参数时, 因为自动转换为了指针类型，所以在函数中无法动态计 算除数组的元素个数，不能在函数中计算传入数组的长度

1. **void** test(**int** arr[])
2. {
3. **int** sz2 = **sizeof**(arr)/**sizeof**(arr[0]);
4. printf("sz2 = %d\n", sz2);//1
5. }
7. **int** main()
8. {
9. **int** arr[10] = {1,2,3,4,5,6,7,8,9,10};
10. **int** sz1 = **sizeof**(arr)/**sizeof**(arr[0]);
11. printf("sz1 = %d\n", sz1);//10
12. test(arr);//数组传参传递的是数组⾸元素的地址
13. }

%zu 这是因为 sizeof 的返回值类型是 size\_t 类型（unsigned int），%zu 是专门为这种类型设置的转换说明

二维数组作为函数参数，在被调函数中不能获得其有多少行，需要通过参数传入，即a[][2]和a[1][2]√，a[1][]×

排序：

1.计数排序

找出待排序数组最大值

定义一个索引最大值为待排序数组最大值的数组

遍历待排序数组, 将待排序数组遍历到的值作新数组索引

在新数组对应索引存储值原有基础上+1

法一：

1. // 遍历待排序数组
2. **for**(**int** i = 0; i < len; i++){
3. // 取出待排序数组当前值
4. **int** index = nums[i];
5. // 将待排序数组当前值作为排序数组索引
6. // 将用于排序数组对应索引原有值+1
7. newNums[index] = newNums[index] +1;
8. }
10. // 计算待排序数组长度
11. **int** len2 = **sizeof**(newNums) / **sizeof**(newNums[0]);
12. // 输出排序数组索引, 就是排序之后结果
13. **for**(**int** i = 0; i < len2; i++){
14. **for**(**int** j = 0; j < newNums[i]; j++){
15. printf("%i\n", i);
16. }
17. }

2.选择排序

假设按照升序排序

用第0个元素和后面所有元素依次比较

判断第0个元素是否大于当前被比较元素, 一旦小于就交换位置

第0个元素和后续所有元素比较完成后, 第0个元素就是最小值

排除第0个元素, 用第1个元素重复1~3操作, 比较完成后第1个元素就是倒数第 二小的值

以此类推, 直到当前元素没有可比较的元素, 排序完成

1. //选择排序
2. **void** selectSort(**int** numbers[], **int** length) {
4. // 外循环为什么要-1?
5. // 最后一位不用比较, 也没有下一位和它比较, 否则会出现错误访问
6. **for** (**int** i = 0; i < length; i++) {
7. **for** (**int** j = i; j < length - 1; j++) {
8. // 1.用当前元素和后续所有元素比较
9. **if** (numbers[i] < numbers[j + 1]) {
10. //  2.一旦发现小于就交换位置
11. swapEle(numbers, i, j + 1);
12. }
13. }
14. }
15. }

3.冒泡排序

假设按照升序排序

从第0个元素开始, 每次都用相邻两个元素进行比较

一旦发现后面一个元素小于前面一个元素就交换位置

经过一轮比较之后最后一个元素就是最大值

排除最后一个元素, 以此类推, 每次比较完成之后最大值都会出现再被比较所有元 素的最后

1. // 冒泡排序
2. **void** bubbleSort(**int** numbers[], **int** length) {
3. **for** (**int** i = 0; i < length; i++) {
4. // -1防止`角标越界`: 访问到了不属于自己的索引
5. **for** (**int** j = 0; j < length - i - 1; j++) {
6. //  1.用当前元素和相邻元素比较
7. **if** (numbers[j] < numbers[j + 1]) {
8. //  2.一旦发现小于就交换位置
9. swapEle(numbers, j, j + 1);
10. }
11. }
12. }
13. }

4.插入排序：

假设按照升序排序

从索引为1的元素开始向前比较, 一旦前面一个元素大于自己就让前面的元素先后 移动

直到没有可比较元素或者前面的元素小于自己的时候, 就将自己插入到当前空出来 的位置

1. //  1.从第一个元素开始依次取出所有用于比较元素
2. **for** (**int** i = 1; i < len; i++)
3. {
4. // 2.遍历取出前面元素进行比较
5. **for**(**int** j = i; j > 0; j--)
6. {
7. // 3.如果前面一个元素大于当前元素,就交换位置
8. **if**(nums[j-1] > nums[j]){
9. **int** temp = nums[j];
10. nums[j] = nums[j - 1];
11. nums[j - 1] = temp;
12. }**else**{
13. **break**;
14. }
15. }
16. }

5.希尔排序

希尔排序可以理解为插入排序的升级版, 先将待排序数组按照指定步长划分为几个 小数组

步骤一：间隔分组（通常为总长度的一半）

步骤二：组内排序

步骤三：重新设置间隔分组（为前一次分组的一半）

步骤四：插入排序

1. //2.计算步长
2. **int** gap = len / 2;
3. **do**{
4. //  1.从第一个元素开始依次取出所有用于比较元素
5. **for** (**int** i = gap; i < len; i++)
6. {
7. // 2.遍历取出前面元素进行比较
8. **int** j = i;
9. **while**((j - gap) >= 0)
10. {
11. printf("%i > %i\n", nums[j - gap], nums[j]);
12. // 3.如果前面一个元素大于当前元素,就交换位置
13. **if**(nums[j - gap] > nums[j]){
14. **int** temp = nums[j];
15. nums[j] = nums[j - gap];
16. nums[j - gap] = temp;
17. }**else**{
18. **break**;
19. }
20. j--;
21. }
22. }
23. // 每个小数组排序完成, 重新计算步长
24. gap = gap / 2;
25. }**while**(gap >= 1);

查找

1.折半查找

定义中间值，最大值，最小值

查找值与中间值比较，查找值>中间值，让最大值=中间值+1

反之一样，若中间值与中间值一样，则返回中间值

1. **while** (min <= max) {
2. // 如果mid对应的值 大于 key, 那么max要变小
3. **if** (values[mid] > key) {
4. max = mid - 1;
5. // 如果mid对应的值 小于 key, 那么min要变
6. }**else** **if** (values[mid] < key) {
7. min = mid + 1;
8. }**else** {
9. **return** mid;
10. }
11. // 修改完min/max之后, 重新计算mid的值
12. mid = (min + max) \* 0.5;
13. }
14. **return** -1;

二维数组：

二维数组的长度=（行）sizeof(a)/sizeof(a[0]);(列）sizeof(a[0])/sizeof(a[0][0])(sizeof(a[0]));

二维数组的定义：数据类型 数组名[一维数组的个数][一维数组的元素个数]

初始化：定义的同时初始化，先定义后初始化，按行分段赋值，按行连续赋值，第一维的长度，指定元素的初始化。

1. **int** a[3][4] = {0};
2. printf("sizeof(a)=%ld\n",**sizeof**(a));  //48
3. printf("sizeof(a[0][0])=%ld\n",**sizeof**(a[0][0])); //4
4. printf("sizeof(a[0])=%ld\n",**sizeof**(a[0])); //16
5. printf("sizeof(a[0]+1)=%ld\n",**sizeof**(a[0]+1));  //8   a[0]实际上被视为int\*类型
6. printf("sizeof(\*(a[0]+1))=%ld\n",**sizeof**(\*(a[0]+1))); //4 a[0]+1指针向前移动一位，指向数组a[0]中的第二个元素
7. printf("sizeof(a+1)=%ld\n",**sizeof**(a+1)); //8
8. printf("sizeof(\*(a+1))=%ld\n",**sizeof**(\*(a+1))); //16 指针向前移动一位，即第二个一维数组a[1]
9. printf("sizeof(&a[0]+1)=%ld\n",**sizeof**(&a[0]+1));  //8
10. printf("sizeof(\*(&a[0]+1))=%ld\n",**sizeof**(\*(&a[0]+1))); //16
11. printf("sizeof(\*a)=%ld\n",**sizeof**(\*a)); //16
12. printf("sizeof(a[3])=%ld\n",**sizeof**(a[3])); //16

二维数组与函数：

1.基本数据类型和函数

在函数内修改形参, 不会影响到外界实参

2.一维数组和函数

在函数内修改形参, 会影响到外界实参

3.二维数组和函数

在函数内修改形参, 会影响到外界实参

总结:

形参的数组必须有值（a[1][1]）或者a[][1];

以后要想知道在函数中修改形参会不会影响到外界的实参, 是不是看一下传递的是 不是地址就可以了（a=&a=&a[0]=&a[0][0]）

如果传递的是地址, 那么在函数内修改形参, 会影响到外界实参

字符串

字符串的存储取决于声明方式和作用域

定义字符串：

1.利用数组定义字符串

1. **char** str1[] = {'l','n','j','\0'};

2.利用双引号定义字符串

1. **char** str2[] = "lnj"; // {'l','n','j','\0'};

注意：

1. //   char str1[] = {'l','n','j'}; // 不是一个字符串, 因为不是以\0结尾的
2. //   char str1[3] = {'l','n','j'}; // 不是一个字符串, 因为不是以\0结尾的
3. // 在定义的同时部分初始化, 没有被初始化的元素会自动赋值为0

// 而'\0'的ASCII码值就是0, 所以以下是字符串

1. **char** str1[4] = {'l','n','j'}; // {'l','n','j', 0}; -> {'l','n','j','\0'};

字符串输出：

如果字符数组中存储的是一个字符串, 那么字符数组的输入输出将变得简单方便,不必使用循环语句逐个地输入输出每个字符

可以使用printf函数和scanf函数一次性输出输入一个字符数组中的字符串,使用的格式字符串为“%s”,表示输入、输出的是一个字符串 字符串的输出

1. **char** chs[] = "lnj";
2. printf("%s\n", chs);  //inj

注意点: \0引发的脏读问题

1. **char** name[] = {'c', 'o', 'o', 'l' , '\0'};
2. **char** name2[] = {'l', 'n', 'j'};
3. printf("name2 = %s\n", name2); // 输出结果: lnjcool

字符串输入

1. **char** a[10]={0};
2. // 输入字符串不需要&
3. scanf("%s",a);
5. printf("%s\n",a);

getchar()函数的作用是从标准输入键盘中读取一个字符，并返回的是一个 int 类型的值而不是 char 类型的值（用来判断字符串的个数，）--不需要回车就能得到输入字符

putchar: 向屏幕输出一个字符

typedef 类型定义（typedef int int32）

sizeof（类型） 表示字节数

注意：一个数加上字符型‘0’自动转换成字符型，

对一个字符串数组, 如果不做初始化赋值, 必须指定数组长度

字符串的结尾是：‘\0’

当用scanf函数输入字符串时,字符串中不能含有空格,否则将以空格作为串的结束符

注意：获得长度，使用时长度需减一，因为字符串是以数组方式存储

字符串常用方法

使用输入输出的字符串函数,在使用前应包含头文件"stdio.h"

使用其它字符串函数则应包含头文件"string.h"

字符串输出函数:puts --------格式: puts(字符数组名)

功能:把字符数组中的字符串输出到显示器。即在屏幕上显示该字符串。

优点:自动换行,可以是数组的任意元素地址

缺点:不能自定义输出格式, 例如 puts("hello %i");

1. **char** ch[] = "lnj";
2. puts(ch); //输出结果: lnj
3. Puts函数完全可以由printf函数取代。当需要按一定格式输出时,通常使用printf函数

字符串输入函数:gets------格式: gets (字符数组名)

功能:从标准输入设备键盘上输入一个字符串。

1. **char** ch[30];
2. gets(ch); // 输入:lnj
3. puts(ch); // 输出:lnj
4. 可以看出当输入的字符串中含有空格时,输出仍为全部字符串。说明gets函数并不以空格作为字符串输入结束的标志,而只以回车作为输入结束。这是与scanf函数不同的。
5. 注意gets很容易导致数组下标越界，是一个不安全的字符串操作函数

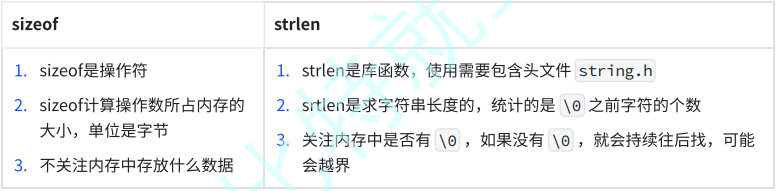
字符串长度

利用sizeof字符串长度

因为字符串在内存中是逐个字符存储的,一个字符占用一个字节,所以字符串的结束 符“\0”长度也是占用的内存单元的字节数。

1. **char** name[] = "it666";
2. **int** size = **sizeof**(name);// 包含\0
3. printf("size = %d\n", size); //输出结果:6

利用系统函数--------格式: strlen(字符数组名) strlen不能应用字符

1. **char** name[] = "it666";
2. **size\_t** len = strlen(name2);
3. printf("len = %lu\n", len); //输出结果:5

字符串连接函数:strcat-------格式: strcat(字符数组名1,字符数组名2)

功能:把字符数组2中的字符串连接到字符数组1 中字符串的后面,并删去字符串1 后的串标志 “\0”。本函数返回值是字符数组1的首地址。

1. **char** oldStr[100] = "welcome to";
2. **char** newStr[20] = " lnj";
3. strcat(oldStr, newStr);
4. puts(oldStr); //输出: welcome to lnj"
5. 本程序把初始化赋值的字符数组与动态赋值的字符串连接起来。要注意的字符数组1应定义足够的长度,否则不能全部装入被连接的字符串。

字符串拷贝函数:strcpy-------格式: strcpy(字符数组名1,字符数组名2) - 功能:把字符数组2中的字符串拷贝到字符数组1中。串结束标志“\0”也一同拷贝。 字符数名2, 也可以是一个字符串常量。这时相当于把一个字符串赋值被覆盖一个字符数组。

1. **char** oldStr[100] = "welcome to";
2. **char** newStr[50] = " lnj";
3. strcpy(oldStr, newStr);
4. puts(oldStr); // 输出结果:  lnj // 原有数据会被覆盖
5. 本函数要求字符数组1应有足够的长度,否则不能全部装入所拷贝的字符串。

字符串比较函数:strcmp-------格式: strcmp(字符数组名1,字符数组名2)

功能:按照ASCII码顺序比较两个数组中的字符串,并由函数返回值返回比较结果。

字符串1=字符串2,返回值=0;

字符串1>字符串2,返回值>0;

字符串1<字符串2,返回值<0。

1. printf("%d", strcmp(oldStr, newStr));  //输出结果:相同为0

另外：strcasecmp 是C语言中用于比较两个字符串，它忽略了字母的大小写

1. **if** (strcasecmp(str1, str2) == 0) {
2. printf("两个字符串相等\n");
3. } **else** {
4. printf("两个字符串不相等\n");
5. }

字符串分割：strtok()——char \*strtok(char \*str, const char \*delim);

功能：用于将一个字符串分割成多个子字符串。它通过在指定的分隔符处插入空字符 （'\0'）来实现分割

1. **char** str[] = "This is a sample string";
2. **char** \*token;
4. // 分割字符串
5. token = strtok(str, " \n\t");  //分割空格\n\t
6. **while** (token != NULL) {
7. printf("%s\n", token);
8. token = strtok(NULL, " \n\t");//这里NULL以便 strtok() 函数继续从上次分割的位置开始分割
9. }

函数

函数的传参形式有两种

1. 按值传递参数
2. #include<stdio.h>
3. **void** swp(**int** a,**int** b)
4. {
5. **int** c=a;
6. a=b;
7. b=c;
8. }
9. **int** main()
10. {
11. **int** x=100,y=200;
12. printf("交换前\n");
13. printf("x=%d\n",x);//100
14. printf("y=%d\n",y);//200
15. swp(x,y);
16. printf("交换后\n");
17. printf("x=%d\n",x);//100
18. printf("y=%d\n",y);//200
20. }
21. 按地址传递参数
22. //2、按地址传递参数
23. #include<stdio.h>
24. **void** swp(**int** \*a,**int** \*b)
25. {
26. **int** c=\*a;
27. \*a=\*b;
28. \*b=c;
29. }
30. **int** main()
31. {
32. **int** x=100,y=200;
33. swp(&x,&y);
34. printf("交换后\n");
35. printf("x=%d\n",x);//200
36. printf("y=%d\n",y); //100
37. }

指针

定义指针变量（本身会带一个地址，存储空间会指向&a的地址）-------数据类型 \*变量名称;

指针初始化：

定义的同时进行初始化

1. **int** a = 5;
2. **int** \*p = &a;

先定义后初始化

1. **int** a = 5;
2. **int** \*p;
3. p=&a;

把指针初始化为NULL

1. **int** \*p=NULL;
2. **int** \*q=0;

不合法的初始化:

指针变量专门用来存储地址的。

指针变量只能存储地址, 不能存储其它类型

1. **int** \*p;
2. p =  250; // 错误写法

给指针变量赋值时,指针变量前不能再加“\*”

1. **int** \*p;
2. \*p=&a; //错误写法

即：指针变量int \*p; p=&a;p就是指针变量

注意点:

多个指针变量可以指向同一个地址

指针的指向是可以改变的

指针没有初始化里面是一个垃圾值,这时候我们这是一个野指针

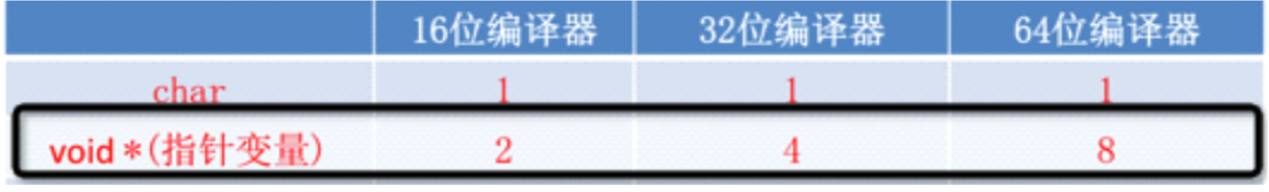
野指针可能会导致程序崩溃，野指针访问你不该访问数据

所以指针必须初始化才可以访问其所指向存储区域

在不是定义变量的时候 \*是一个操作符,代表访问指针所指向存储空间

指针类型

在同一种编译器环境下,一个指针变量所占用的内存空间是固定的



二级指针

如果一个指针变量存放的又是另一个指针变量的地址,则称这个指针变量为指向指 针的指针￼变量。也称为“二级指针”

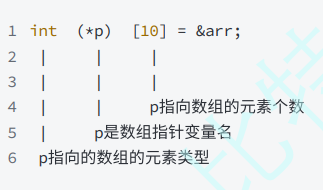
1. **int** a=100;
2. **int** \*p=&a;
3. **int** \*\*pp=&p;
4. printf("\*\*pp=%d\n",\*\*pp);//100
5. printf("\*pp=%p\n",\*pp);  //\*pp==p==&a==a的地址
6. printf("pp=%p\n",pp);//pp的地址比\*pp地址大4个字节

指针和数组

1. **int** a[5]={100,200,300,400,500};
2. **int** \*p=NULL;
3. p=a;
4. **for**(**int** i=0;i<5;i++){
5. printf("\*(a+%d)=%d\n",i,\*(a+i));//100,200,300,400,500
6. }
7. **for**(**int** i=0;i<5;i++){
8. printf("\*(p+%d)=%d\n",i,\*(p+i));//100,200,300,400,500
9. }
10. //p[i]==a[i]==\*(a+i)==\*(p+i)

13. **int** arr[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};
14. **for**(**int** i=0;i<3;i++){
15. **for**(**int** j=0;j<4;j++){
16. printf("\*(a+%d)+%d=%p\n",i,j,\*(arr+i)+j);//arr数组的地址
17. }
18. }
19. **for**(**int** i=0;i<3;i++){
20. **for**(**int** j=0;j<4;j++){
21. printf("\*(\*(a+%d)+%d)=%d\n",i,j,\*(\*(arr+i)+j));//arr数组的值
22. }
23. }
24. //&a+1==a+1 跳过了整个数组的长度（地址）
25. -----&a+1地址是会增加整个数组大小的长度，a+1增加一个int类型大小
26. //a+i==&a[i]
27. //\*(a+i)==a[i]
28. //\*(a+i)+j==&a[i][j]
29. //\*((a+i)+j)==a[i][j]
30. //\*[]:是取值运算符，不同于\*，带偏移量，a[b] 🡺\*(a+b)🡺b[a]
31. ---> arr(int \*) + 1 == arr(int \*) + 1(int \*) //将1转换为int\*类型

数组指针



1. **int** \*p1[10];  //是一个数组，数组里的每个元素都是一个整型指针
2. **int** (\*p2)[10];//是一个指针，它指向一个包含10个整型元素的数组

指针与字符串

定义字符串的两种方式

字符数组

1. **char** string[]=”I love lnj!”;
2. printf("%s\n",string);

字符串指针指向字符串

1. // 数组名保存的是数组第0个元素的地址, 指针也可以保存第0个元素的地址
2. **char** \*str = "abc"

字符串指针使用注意事项

可以查看字符串的每一个字符

1. har \*str = "lnj";
2. **for**(**int** i = 0; i < strlen(str);i++)
3. {
4. printf("%c-", \*(str+i)); // 输出结果:l-n-j
5. }

不可以修改字符串内容

1. //使用字符数组来保存的字符串是保存栈里的,保存栈里面东西是可读可写,所有可以修改字符串中的的字符
2. //使用字符指针来保存字符串,它保存的是字符串常量地址,常量区是只读的,所以我们不可以修改字符串中的字符
3. **char** \*str = "lnj";
4. \*(str+2) = 'y'; // 错误

不能够直接接收键盘输入

1. // 错误的原因是:str是一个野指针,他并没有指向某一块内存空间
2. // 所以不允许这样写如果给str分配内存空间是可以这样用 的
3. **char** \*str;
4. scanf("%s", str);

指针字符串不能修改，内存在常数区不能修改，数组字符串存在栈区可以修改。

指针字符串，指针存在栈区，字符串存在常数据，当在新建指针字符串，会在同一常 数区寻找字符，如果相同则指向同一地址，否则新的字符串继续按顺序存在常数区

数组字符串，都存在栈区，能修改

函数和指针

数组名保存的就是数组的地址, 所以可以直接将数组名赋值给一个指针变量

函数名和数组名非常相似, 函数名保存的也是函数的地址, 所以可以直接将函数 名赋值给一个指针变量

指向函数指针

指针函数的定义-------格式: 返回值类型 (\*指针变量名)(形参1, 形参2, ...);

1. **int** sum(**int** a,**int** b)
2. {
3. **return** a + b;
4. }
6. **int** (\*p)(**int**,**int**);
7. p = sum;  //用指针变量指向函数

指针函数定义技巧

1、把要指向函数头拷贝过来

2、把函数名称使用小括号括起来

3、在函数名称前面加上一个\*

4、修改函数名称

使用步骤：

1. 定义指针函数
2. 指向函数
3. 使用函数
4. 打印
5. //定义指针函数
6. **int** (\*p2)();
7. //指向函数
8. p2=minus;
9. //获取返回值
10. **int** ret=p2(4,1);
11. printf("%d\n",ret);

还可以这样用

1. **int** ret= calculate(4,3,sum);
2. printf("ret=%d",ret);//  7

5. //定义a,b变量和指针变量函数，，将a,b值返回到指针变量函数中
6. **int** calculate(**int** a,**int** b,**int**(\*fn)(**int** a,**int** b)){
7. **return** fn(a,b);
8. }
9. **int** minus(**int** a, **int** b){
10. **return** a - b;
11. }
12. **int** sum(**int** a, **int** b){
13. **return** a +b;
14. }

函数指针的解引用

1. **int** (\*pf)(**int**, **int**) = &Add; // pf 是指向一个接受两个整数参数并返回整数的函数的指针
3. //两者相同
4. **int** result = (\*pf)(3, 5); // 相当于调用 Add(3, 5)，result 为 8
5. **int** result2 = pf(4, 6); // 直接调用，result2 为 10

函数指针数组：

1. **int** (\*parr1[3])();

结论: 指针变量指向谁, 调用的就是谁 函数名就是地址

只要将函数名赋值给一个指针变量之后, 我们就可以像利用函数名操作函数一样, 利用指针变量操作函数

指针函数易错点，返回栈内存的错误，返回局部变量地址错误

可以使用 const来表明函数不会改变指针参数所指向的对象。

解引用：使用指针访问或修改指针指向的内存内容时，这个过程就是

指针-指针，两个指针必须指向同一块空间，指针减指针的结果是两个指针之间相隔的元素个数

解引用操作符 \* 用于访问指针变量所指向的内存地址中的数据

指针（地址）只能与指针（地址）比较，地址+-整数=地址，也就是指针。

\*&a == a本身

结构体

什么是结构体?

结构体和数组一样属于构造类型

数组是用于保存一组相同类型数据的, 而结构体是用于保存一组不同类型数组的

定义结构体类型

格式：

1. **struct**　结构体名{
2. 类型名1　成员名1;
3. 类型名2　成员名2;
4. ……
5. 类型名n　成员名n;
6. };

实例

1. **struct** Student {
2. **char** \*name; // 姓名
3. **int** age; // 年龄
4. **float** height; // 身高
5. };

定义结构体变量：---------格式: struct 结构体名 结构体变量名;

先定义结构体类型，再定义变量

1. **struct** Student {
2. **char** \*name;
3. **int** age;
4. };
6. **struct** Student stu;

定义结构体类型的同时定义变量

1. **struct** Student {
2. **char** \*name;
3. **int** age;
4. } stu;

匿名结构体定义结构体变量

1. **struct** {
2. **char** \*name;
3. **int** age;
4. } stu;

第三种方法与第二种方法的区别在于,第三种方法中省去了结构体类型名称,而直接 给出结构变量,这种结构体最大的问题是结构体类型不能复用

结构体成员访问-------结构体变量名.成员名

1. **struct** Student {
2. **char** \*name;
3. **int** age;
4. };
5. **struct** Student stu;
6. // 访问stu的age成员
7. stu.age = 27;
8. printf("age = %d", stu.age);

结构体类型作用域

结构类型定义在函数内部的作用域与局部变量的作用域是相同的，从定义的那一行 开始, 直到遇到return或者大括号结束为止（内部结构体=局部变量）

结构类型定义在函数外部的作用域与全局变量的作用域是相同的，从定义的那一行 开始,直到本文件结束为止

结构体数组

结构体数组和普通数组并无太大差异, 只不过是数组中的元素都是结构体而已

格式: struct 结构体类型名称 数组名称[元素个数]

1. **struct** Student {
2. **char** \*name;
3. **int** age;
4. };
5. **struct** Student stu[2];

结构体指针

一个指针变量当用来指向一个结构体变量时,称之为结构体指针变量

格式: struct 结构名 \*结构指针变量名

1. // 定义一个结构体类型
2. **struct** Student {
3. **char** \*name;
4. **int** age;
5. };
7. **struct** Student stu = {“lnj", 18};   // 定义一个结构体变量
8. **struct** Student stu1[2] = {{“lnj", 18},{"jnm",19}};  // 定义一个结构体变量s数组
10. **struct** Student \*p，\*p2;  // 定义一个指向结构体的指针变量

13. p = &stu;  // 指向结构体变量stu
14. printf("name=%s, age = %d \n", stu.name, stu.age);//方式1：结构体变量名.成员名
15. printf("name=%s, age = %d \n", (\*p).name, (\*p).age);//方式2：(\*指针变量名).成员名
16. printf("name=%s, age = %d \n", p->name, p->age);//方式3：指针变量名->成员名


20. p2 = stu1/&stu1;  // 指向数组
21. printf("name=%s, age = %d \n", stu[0].name, stu[0].age); //方式1：结构体变量名[下标].成员名
22. printf("name=%s, age = %d \n", p[0].name,p[0].age);//方式2：指针变量名[下标].成员名
23. // 方式3：指针变量名->成员名

26. **return** 0;

结构体指针访问结构体成员:(\*结构指针变量).成员名--------结构指针变量->成 员名(用熟)

结构体指针不能初始化，遇见结构体指针变量，调用函数时传地址print

指针获取数据：p->data, (\*p).data

指针获取地址：p.next=&a;

指针中p代表此空间的地址，\*p代表此空间的数据

结构体变量占用的内存空间永远是所有成员中占用内存最大成员的倍数

例一

1. **struct** Person{
2. **int** age; // 4
3. **char** ch; // 1
4. **double** score; // 8
5. };
6. **struct** Person p;
7. printf("sizeof = %i\n", **sizeof**(p)); // 16
9. //占用内存最大属性是score, 占8个字节, 所以第一次会分配8个字节
10. //将第一次分配的8个字节分配给age4个,分配给ch1个, 还剩下3个字节
11. //当需要分配给score时, 发现只剩下3个字节, 所以会再次开辟8个字节存储空间
12. //一共开辟了两次8个字节空间, 所以最终p占用16个字节

例二

1. **struct** Person{
2. **int** age; // 4
3. **double** score; // 8
4. **char** ch; // 1
5. };
6. **struct** Person p;
7. printf("sizeof = %i\n", **sizeof**(p)); // 24
8. //占用内存最大属性是score, 占8个字节, 所以第一次会分配8个字节
9. //将第一次分配的8个字节分配给age4个,还剩下4个字节
10. //当需要分配给score时, 发现只剩下4个字节, 所以会再次开辟8个字节存储空间
11. //将新分配的8个字节分配给score, 还剩下0个字节
12. //当需要分配给ch时, 发现上一次分配的已经没有了, 所以会再次开辟8个字节存储空间
13. //一共开辟了3次8个字节空间, 所以最终p占用24个字节

结构体嵌套定义

成员也可以又是一个结构,即构成了嵌套的结构

1. **struct** Date{
2. **int** month;
3. **int** day;
4. **int** year;
5. }
6. **struct**  stu{
7. **int** num;
8. **char** \*name;
9. **char** sex;
10. **struct** Date birthday;
11. Float score;
12. }
14. **struct** stu s={1,"xhd","x",1,2,3,12.1};
15. printf("s.birthday.month=%d",s.birthday.month);

注意:结构体不可以嵌套自己变量,可以嵌套指向自己这种类型的指针

结构体和函数

结构体虽然是构造类型, 但是结构体之间赋值是值拷贝, 而不是地址传递

1. **struct** Person{
2. **char** \*name;
3. **int** age;
4. };
5. **struct** Person p1 = {"lnj", 35};
6. **struct** Person p2;
7. p2 = p1;  // 不能p2=&p1
8. p2.name = "zs"; // 修改p2不会影响p1
9. printf("p1.name = %s\n", p1.name); // lnj
10. printf("p2.name = %s\n", p2.name); //  zs

结构体变量作为函数形参时也是值传递, 在函数内修改形参, 不会影响外界实参

1. #include <stdio.h>
2. **void** test(**struct** Person per);
3. **struct** Person{
4. **char** \*name;
5. **int** age;
6. };
8. **int** main()
9. {
10. **struct** Person p1 = {"lnj", 35};
11. printf("p1.name = %s\n", p1.name); // lnj
12. test(p1);
13. printf("p1.name = %s\n", p1.name); // lnj
14. **return** 0;
15. }
16. **void** test(**struct** Person per){
17. per.name = "zs";
18. }

共用体

和结构体不同的是, 结构体的每个成员都是占用一块独立的存储空间, 而共用体所有的成员都占用同一块存储空间

和结构体一样, 共用体在使用之前必须先定义共用体类型, 再定义共用体变量

定义共用体类型格式:

1. **union** 共用体名{
2. 数据类型 属性名称;
3. 数据类型 属性名称;
4. ...   ....
5. };

特点: 由于所有属性共享同一块内存空间, 所以只要其中一个属性发生了改变, 其它的属性都会受到影响

1. **union** Test{
2. **int** age;
3. **char** ch;
4. };
5. **union** Test t;
6. printf("sizeof(p) = %i\n", **sizeof**(t));
8. t.age = 33;
9. printf("t.age = %i\n", t.age); // 33
10. t.ch = 'a';
11. printf("t.ch = %c\n", t.ch); // a
12. printf("t.age = %i\n", t.age); // 97

注意：只能个体初始化，不能整体初始化

枚举

“枚举”的类型。在“枚举”类型的定义中列举出所有可能的取值, 被说明为该“枚举”类型的变量取值不能超过定义的范围。

格式：

1. **enum**　枚举名　{
2. 枚举元素1,
3. 枚举元素2,
4. ……
5. };

C语言编译器会将枚举元素(spring、summer等)作为整型常量处理，称为枚举常量。

枚举元素的值取决于定义时各枚举元素排列的先后顺序。默认情况下，第一个枚举元素的值为0，第二个为1，依次顺序加1。

也可以在定义枚举类型时改变枚举元素的值

1. **enum** Season {
2. Spring,
3. Summer,
4. Autumn,
5. Winter
6. };
7. // 也就是说spring的值为0，summer的值为1，autumn的值为2，winter的值为3

1. **enum** Season {
2. Spring = 9,
3. Summer,
4. Autumn,
5. Winter
6. };
7. // 也就是说spring的值为9，summer的值为10，autumn的值为11，winter的值为12

static与extern对函数的作用

内部函数:只能在本文件中访问的函数

外部函数:可以在本文件中以及其他的文件中访问的函数

默认情况下所有的函数都是外部函数

static 作用：声明一个内部函数

1. **static** **int** sum(**int** num1,**int** num2);

extern作用：声明一个外部函数

1. **extern** **int** sum(**int** num1,**int** num2);

注意点:由于默认情况下所有的函数都是外部函数, 所以extern一般会省略如果只有函数声明添加了static与extern, 而定义中没有添加static与extern, 那么无效

计算机运算过程分析

1.通过地址线找到对应地址的存储单元

2.通过控制线发送内存读取指令

3.通过数据线将内存中的值传输到CPU寄存器中

4.在CPU中完成计算操作

5.通过地址线找到对应地址的存储单元

6.通过控制线发送内存写入指令

7.通过数据线将计算结果传输到内存中

预处理指令

所有预处理指令都以符号“#”开头，并且结尾不用分号

宏定义-----格式:#define 标识符 字符串

1. #include <stdio.h>
3. //宏名一般用大写字母，以便与变量名区别开来，但用小写也没有语法错误
4. // 源程序中所有的宏名PI在编译预处理的时候都会被3.14所代替
5. #define R  3.0
6. #define PI 3.14
7. //定义一个宏时可以引用已经定义的宏名
8. #define L  2\*PI\*R
9. #define S  PI\*R\*R
11. //可用宏定义表示数据类型,使书写方便
12. #define String char \*
14. // 根据圆的半径计radius算周长
15. **float** girth(**float** radius) {
16. **return** 2 \* PI \*radius;
17. }
19. **int** main ()
20. {
21. **float** g = girth(2);
23. printf("周长为：%f", g);
24. **return** 0;
25. }
27. //其中的“#”表示这是一条预处理命令。凡是以“#”开头的均为预处理命令。“define”为宏定义命令。“标识符”为所定义的宏名。“字符串”可以是常数、表达式、格式串等

宏名的有效范围是从定义位置到文件结束。如果需要终止宏定义的作用域，可以用 #undef命令

带参数的宏定义----格式: #define 宏名(形参表) 字符串

1. // 第1行中定义了一个带有2个参数的宏average，
2. #define average(a, b) (a+b)/2//宏名和参数列表之间不能有空格，否则空格后面的所有字符串都作为替换的字符串.
4. **int** main ()
5. {
6. // 第4行其实会被替换成：int a = (10 + 4)/2;，
7. **int** a = average(10, 4);
8. // 输出结果为：7是不是感觉这个宏有点像函数呢？
9. printf("平均值：%d", a);
10. **return** 0;
11. }

带参数的宏在展开时，只作简单的字符和参数的替换，不进行任何计算操作。所以在 定义宏时，一般用一个小括号括住字符串的参数。

1. #include <stdio.h>
2. // 下面定义一个宏D(a)，作用是返回a的2倍数值：
3. #define D(a) 2\*a
4. // 如果定义宏的时候不用小括号括住参数
6. **int** main ()
7. {
8. // 将被替换成int b = 2\*3+4;，输出结果10，如果定义宏的时候用小括号括住参数，把上面的第3行改成：#define D(a) 2\*(a)，注意右边的a是有括号的，第7行将被替换成int b = 2\*(3+4);，输出结果14
10. **int** b = D(3+4);
11. printf("%d", b);
12. **return** 0;
13. }

计算结果最好也用括号括起来

1. #include <stdio.h>
2. // 下面定义一个宏P(a)，作用是返回a的平方
3. #define Pow(a) (a) \* (a) // 如果不用小括号括住计算结果
5. **int** main(**int** argc, **const** **char** \* argv[])      {
6. // 代码被替换为:int b = (10) \* (10) / (2) \* (2);
7. // 简化之后：int b = 10 \* (10 / 2) \* 2;，最后变量b为:100
8. **int** b = Pow(10) / Pow(2);
10. printf("%d", b);
11. **return** 0;
12. }
13. #include <stdio.h>
14. // 计算结果用括号括起来
15. #define Pow(a) ( (a) \* (a) )
17. **int** main(**int** argc, **const** **char** \* argv[])      {
18. // 代码被替换为:int b = ( (10) \* (10) ) / ( (2) \* (2) );
19. // 简化之后：int b = (10 \* 10) / (2 \*2);，最后输出结果：25
20. **int** b = Pow(10) / Pow(2);
22. printf("%d", b);
23. **return** 0;
24. }

关键字

typedef关键字-------格式: typedef 原类型名 新类型名;

C语言不仅􏰀供了丰富的数据类型,而且还允许由用户自己定义类型说明符,也就是说允许由用户为数据类型取“别名”。

1. **typedef** **int** INTEGER
2. INTEGER a; // 等价于 int a;

注意：typedef定义结构体，必须声明别名，用别名声明变量直接声明，大名声明变量需要struct

1. **typedef** **struct** student4
2. {
3. **int** id;
4. **char** name[20];
5. **int** math;
6. }stu4;
8. **struct** student4 s4={18,"tt",70};
9. stu4 s5={17,"c",10};//加struct报错

const关键字

可以定义const常量,具有不可变性

修饰常指针

const int \*A; //const修饰指针,A可变,A指向的值不能被修改

int const \*A; //const修饰指向的对象,A可变,A指向的对象不可变

int \*const A; //const修饰指针A, A不可变,A指向的对象可变

const int \*const A;//指针A和A指向的对象都不可变

1. \*\*\*\*\*未验证\*\*\*\*
2. 先看“\*”的位置
3. 如果**const** 在 \*的左侧 指针指向的内容不能通过指针修改,但是指针变量本身内容可以改变。
4. 如果**const** 在 \*的右侧 表示指针变量内容不能修改,但是指针指向的内容，可以通过指针改变。
5. 如果在“\*”的两侧都有**const** 标识指向和值都不能改。

内存管理

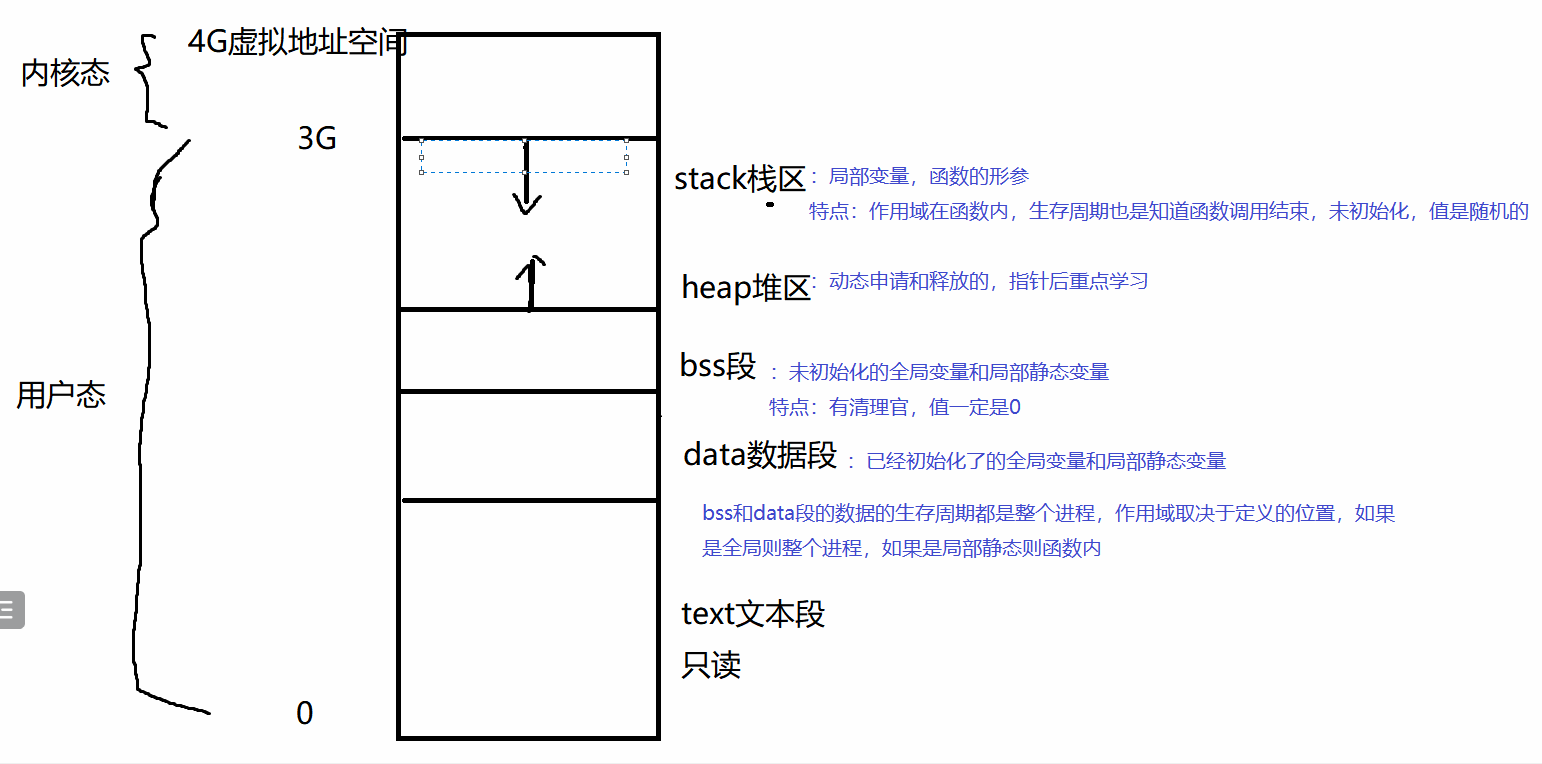
代码区：这个区域是只读的，通常是由操作系统加载程序时分配的这个区域通常是共享的

全局区：全局变量是在函数外部定义的变量，它们在整个程序的执行过程中都是可见的

常量区：用于存储字符串常量和其他常量数据，它们在程序执行过程中是不可修改的

栈区：存放函数的参数值、局部变量等，会自动释放

堆区：就是通过new、malloc、[realloc](https://so.csdn.net/so/search?q=realloc&spm=1001.2101.3001.7020)分配的内存块，编译器不会负责它们的释放工作，需要用程序区释放。



malloc函数

|  |  |
| --- | --- |
| 函数声明 | void \* malloc(size\_t \_Size);//需用指针 |
| 所在文件 | stdlib.h |
| 函数功能 | 申请堆内存空间并返回,所申请的空间并未初始化。 |
| 常见的初始化方法是 | memset 字节初始化。 |
| 参数及返回解析 |  |
| 参数 | size\_t \_size 表示要申请的字符数 |
| 返回值 | void \* 成功返回非空指针指向申请的空间 ，失败返回 NULL |

1. // 1.申请3块4个字节存储空间
2. **int** \*p;
3. P = (**int** \*)malloc(**sizeof**(**int**) \* 3);
4. // 2.使用申请好的3块存储空间
5. p[0] = 1;
6. p[1] = 3;
7. p[2] = 5;
8. printf("p[0] = %i\n", p[0]);
9. printf("p[1] = %i\n", p[1]);
10. printf("p[2] = %i\n", p[2]);
11. // 3.释放空间
12. free(p);

free函数

注意: 通过malloc申请的存储空间一定要释放, 所以malloc和free函数总是成对出现

|  |  |
| --- | --- |
| 函数声明 | void free(void \*p); |
| 所在文件 | stdlib.h |
| 函数功能 | 释放申请的堆内存 |
| 参数及返回解析 |  |
| 参数 | void\* p 指向手动申请的空间 |
| 返回值 | void 无返回 |

calloc函数

|  |  |
| --- | --- |
| 函数声明 | void \*calloc(size\_t nmemb, size\_t size); |
| 所在文件 | stdlib.h |
| 函数功能 | 申请堆内存空间并返回，所申请的空间，自动清零 |
| 参数及返回解析 |  |
| 参数 | size\_t nmemb 所需内存单元数量 |
| 参数 | size\_t size 内存单元字节数量 |
| 返回值 | void \* 成功返回非空指针指向申请的空间 ，失败返回 NULL |

1. **int** \*p = calloc(3, **sizeof**(**int**));
2. // 2.使用申请好的3块存储空间
3. p[0] = 1;
4. p[1] = 3;
5. p[2] = 5;
6. printf("p[0] = %i\n", p[0]);
7. printf("p[1] = %i\n", p[1]);
8. printf("p[2] = %i\n", p[2]);
9. // 3.释放空间
10. free(p);

realloc函数

|  |  |
| --- | --- |
| 函数声明 | void \*realloc(void \*ptr, size\_t size); |
| 所在文件 | stdlib.h |
| 函数功能 | 扩容(缩小)原有内存的大小。通常用于扩容，缩小会会导致内存缩去的部分数据丢失。 |
| 参数及返回解析 |  |
| 参数 | void \* ptr 表示待扩容(缩小)的指针， ptr 为之前用 malloc 或者 calloc 分配的内存地址。 |
| 参数 | size\_t size 表示扩容(缩小)后内存的大小。 |
| 返回值 | void\* 成功返回非空指针指向申请的空间 ，失败返回 NULL。 |

1. // 1.申请4个字节存储空间
2. **int** \*p = malloc(**sizeof**(**int**));
3. printf("p = %p\n", p);
4. // 如果能在传入存储空间地址后面扩容, 返回传入存储空间地址
5. // 如果不能在传入存储空间地址后面扩容, 返回一个新的存储空间地址
6. p = realloc(p, **sizeof**(**int**) \* 2);
7. printf("p = %p\n", p);
8. // 2.使用申请好的空间
9. \*p = 666;
10. printf("\*p = %i\n",  \*p);
11. // 3.释放空间
12. free(p);

链表

静态链表：**静态链表是用数组来实现链式存储结构，静态链表实际上就是一个结构体数组**

**连续的内存空间，数据元素随机存放**

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
5. // 1.定义链表节点
6. **typedef** **struct** node{
7. **int** data;
8. **struct** node \*next;
9. }Node;
10. **int** main()
11. {
13. // 2.创建链表节点
14. Node a;
15. Node b;
16. Node c;
18. // 3.初始化节点数据
19. a.data = 1;
20. b.data = 3;
21. c.data = 5;
23. // 4.链接节点
24. a.next = &b;
25. b.next = &c;
26. c.next = NULL;
28. // 5.创建链表头
29. Node \*head = &a;
31. // 6.使用链表
32. **while**(head != NULL){
33. **int** currentData = head->data;
34. printf("currentData = %i\n", currentData);
35. head = head->next;
36. }
37. **return** 0;
38. }

动态链表：动态分配空间，动态存储

文件流

FILE 结构体

FILE 结构体是对缓冲区和文件读写状态的记录者，所有对文件的操作，都是通过FILE 结构体完成的

* fopen()：打开文件。
* fclose()：关闭文件。
* fread()：从文件中读取数据。
* fwrite()：向文件中写入数据。
* fseek()：设置文件指针的位置。
* ftell()：获取文件指针的当前位置。
* rewind()：将文件指针重置到文件开头。
* fgets()：从文件中读取一行数据。
* fputs()：向文件中写入一行数据。
* fprintf()：向文件中格式化输出数据。
* fscanf()：从文件中格式化读取数据。

注意：以写入模式打开文件，这将清空文件内容

fileopen函数--------以 mode 的方式，打开一个 filename 命名的文件，返回一个指向该文件缓冲的 FILE 结构体指针。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| mode | 处理方式 | 当文件不存在时 | 当文件存在时 | 向文件输入 | 从文件输出 |
| r | 读取 | 出错 | 打开文件 | 不能 | 可以 |
| w | 写入 | 建立新文件 | 覆盖原有文件 | 可以 | 不能 |
| a | 追加 | 建立新文件 | 在原有文件后追加 | 可以 | 不能 |
| r+ | 读取/写入 | 出错 | 打开文件 | 可以 | 可以 |
| w+ | 写入/读取 | 建立新文件 | 覆盖原有文件 | 可以 | 可以 |
| a+ | 读取/追加 | 建立新文件 | 在原有文件后追加 | 可以 | 可以 |

注意点:Windows如果读写的是二进制文件，则还要加 b,比如 rb, r+b 等。 unix/linux 不区分文本和二进制文件

fclose函数----fclose()用来关闭先前 fopen()打开的文件.

fprintf()函数----- 用于将格式化的数据写入到指定的文件中

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

1. **FILE** \*file = fopen("output.txt", "w");
2. **if** (file != NULL) {
3. **int** value = 123;
4. **float** pi = 3.14159265;
5. fprintf(file, "Integer value: %d, PI value: %.2f\n", value, pi);
6. fclose(file);
7. }

fscanf()函数------fscanf() 函数则是用于从指定的文件流中按照特定格式读取数据

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

1. **FILE** \*file = fopen("input.txt", "r");
2. **if** (file != NULL) {
3. **int** intValue;
4. **float** floatValue;
5. **if** (fscanf(file, "Integer value: %d, Float value: %f", &intValue, &floatValue) == 2) //fscanf() 将尝试从 "input.txt" 文件中读取格式化数据
6. {
7. printf("From file: Integer value is %d, Float value is %.2f\n", intValue, floatValue);
8. }
9. fclose(file);
10. }
11. **return** 0;

一次读写一个字符

写入：

int fputc (int ch, FILE \* stream );----- 将 ch 字符，写入文件。

1. // 1.打开一个文件
2. **FILE** \*fp = fopen("test.txt", "w+");
4. // 2.往文件中写入内容
5. **for**(**char** ch = 'a'; ch <= 'z'; ch++){
6. // 一次写入一个字符
7. **char** res = fputc(ch, fp);
8. printf("res = %c\n", res);
9. }
11. // 3.关闭打开的文件
12. fclose(fp);

注意：int 写入成功，返回写入成功字符，如果失败，返回 EOF

读取：

int fgetc ( FILE \* stream )----- 从文件流中读取一个字符并返回。

1. // 1.打开一个文件
2. **FILE** \*fp = fopen("test.txt", "r+");
3. // 2.从文件中读取内容
4. har res = EOF;
5. **while**((res = fgetc(fp)) != EOF){
6. printf("res = %c\n", res);
7. }
8. // 3.关闭打开的文件
9. fclose(fp);
10. **return** 0;

判断文件末尾

int feof( FILE \* stream );----- 判断文件是否读到文件结尾

1. // 1.打开一个文件
2. **FILE** \*fp = fopen("test.txt", "r+");
3. // 2.从文件中读取内容
4. **char** res = EOF;
5. // 注意: 由于只有先读了才会修改标志位,
6. // 所以通过feof判断是否到达文件末尾, 一定要先读再判断, 不能先判断再读
7. **while**((res = fgetc(fp)) && (!feof(fp))){
8. printf("res = %c\n", res);
9. }
10. // 3.关闭打开的文件
11. fclose(fp);

注意：返回值int ---- 0 未读到文件结尾，非零 读到文件结尾，feof 这个函数， 是去读标志位判断文件是否结束的。而标志位只有读完了才会被修改, 所以如果先判断 再读标志位会出现多打一次的的现象

所以企业开发中使用feof函数一定要先读后判断, 而不能先判断后读

一次读写一行字符

写入一行：

int fputs(char \*str,FILE \*fp)----- 把 str 指向的字符串写入 fp 指向的文件中。

1. **FILE** \*fw = fopen("test.txt", "w+");
2. // 注意: fputs不会自动添加\n
3. fputs("lnj\n", fw);
4. fputs("it666\n", fw);
6. /\*\*\*\*\*\*\* fputs写入时遇到\0就会自动终止写入 \*\*\*\*\*\*\*/
7. fputs("lnj\0it666\n", fw);
8. fclose(fw);
9. **return** 0;

读取一行：

char \*fgets(char \*str,int length,FILE \*fp)----最多读lenth-1字符，遇见\n或EOF，读入结束，字符串读入后在最后加一个‘\0’字符。

1. **FILE** \*fp = fopen("test.txt", "w+");
2. // 注意: fputs不会自动添加\n
3. fputs("it666\n", fp);
5. // 注意: FILE结构体中读写指针每读或写一个字符后都会往后移动
6. rewind(fp);// 将FILE结构体中的读写指针重新移动到最前面
7. **char** str[1024];
8. // 从fp中读取4个字符, 存入到str中
9. // --------fgets最多只能读取N-1个字符, 会在最后自动添加\0
10. fgets(str, 4, fp);  //fgets是char类型的
12. /\*\*\*\*\*\*\*\*读取到EOF自动结束 \*\*\*\*\*\*\*\*\*/
13. // 每次从fp中读取1024个字符, 存入到str中
14. // 读取到文件末尾自动结束
15. while(fgets(str, 1024, fp)){
16. printf("str = %s", str);
17. }
18. \*/
19. printf("str = %s", str); // it6

一次读写一块数据

写入一块数据:

int fwrite(void \*buffer, int num\_bytes, int count, FILE \*fp)

1. **FILE** \*fp = fopen("test.txt", "wb+");
2. // 注意: fwrite不会关心写入数据的格式
3. **char** \*str = "lnj\0it666";
4. /\*
5. \* 第一个参数: 被写入数据指针
6. \* 第二个参数: 每次写入多少个字节
7. \* 第三个参数: 需要写入多少次
8. \* 第四个参数: 已打开文件结构体指针
9. \*/
10. fwrite((**void** \*)str, 9, 1, fp);
12. fclose(fp);
13. **return** 0;

一次读取一块数据

int fread(void \*buffer, int num\_bytes, int count, FILE \*fp)

1. #include <stdio.h>
3. **int** main()
4. {
6. **FILE** \*fp = fopen("test.txt", "wb+");
7. **int** ages[4] = {1, 3, 5, 6};
8. fwrite(ages, **sizeof**(ages), 1, fp);
9. rewind(fp);
10. **int** data;
11. **while**(fread(age, **sizeof**(**int**), 1, fp) > 0){  //fread返回的是字符的数量
12. printf("data = %i\n", data);
13. }
14. **return** 0;
15. }

注意：读取时num\_bytes应该填写读取数据类型的最小单位, 而count可以随意写,如果读取时num\_bytes不是读取数据类型最小单位, 会引发读取失败

读写结构体

1. **typedef** **struct**{
2. **char** \*name;
3. **int** age;
4. **double** height;
5. } Person;
7. Person p1 = {"lnj", 35, 1.88};
8. **FILE** \*fp = fopen("person.stu", "wb+");
9. fwrite(&p1, **sizeof**(p1), 1, fp);
11. rewind(fp);
12. Person p2;
13. fread(&p2, **sizeof**(p2), 1, fp);
14. printf("name = %s\n", p2.name);
15. printf("age = %i\n", p2.age);
16. printf("height = %lf\n", p2.height);
18. //数组结构体读写
19. Person ps[] = {
20. {"zs", 18, 1.65},
21. {"ls", 21, 1.88},
22. {"ww", 33, 1.9}
23. };
24. **FILE** \*fp = fopen("person.stu", "wb+");
25. fwrite(&ps, **sizeof**(ps), 1, fp);
27. rewind(fp);
28. Person p;
29. **while**(fread(&p, **sizeof**(p), 1, fp) > 0){
30. printf("name = %s\n", p.name);
31. printf("age = %i\n", p.age);
32. printf("height = %lf\n", p.height);
33. }

二进制文件