

C语言

2016年5月12日 星期四 上午8:15

计算机原理、数据结构、网络原理、操作系统原理

Lua脚本语言

功能强大、快速、轻量级、可嵌入的脚本语言(无需编译)

程序语法结构简单，用关键数组来实现强大的数据描述功能，同时提供一些可扩展的语义特征
由解释器先翻译成Lua虚拟机(基于寄存器实现)指令并运行，使用一个采用增量策略的GC来管理
在Lua中一切都是变量



Lua脚本语言
简明教程

BASIC-C

操作系统：管理系统硬件资源和软件资源的系统软件

软件下载(破解版)：www.macx.cn

*.dmg(虚拟光驱)

*.app(绿色版)

*.pkg(自解压安装包)

终端访问U盘：`cd /Volumes`

磁盘格式：Mac OS,FAT(<4G),ExFAT,NTFS(compressed win+mac)

LaunchPad：完整的Dock

文本编辑：不能用于编写程序(因为富文本格式会添加控制符)

MissionControl：多桌面切换

符号常量 `#define PI 3.1415926`

动态类型，
理内存。

关键字 `const int page = 100;`

区别：指针和变量作用域

`int a=(1+2,2+3,3+4)=>3+4=7`

`return 0, 10;` $\rightarrow 10$,表达式返回的是最右边表达式的值

\sim : 一个数取反后的值 = $-1 - (\text{这个数})$

\wedge : 异或 eg: `0101^0011 = 1010` (不同为1,相同为0)

... 表范围

eg: `case 'a'...'z' : printf("%c",ch);break;`

应用程序 \rightarrow 系统调用(API) \rightarrow 操作系统 \rightarrow 驱动程序(硬件) \rightarrow 磁盘

C \rightarrow 编译 \rightarrow 汇编语言 \rightarrow 翻译 \rightarrow 机器指令

`gcc -E *.c` 查看编译前文件内容

`#include<>`

`<>`:加载系统功能

`"`:加载自定义功能, 优先查找当前文件夹下的文件, 再去C函数库查找

`chmod 777 file`

`rwxrwxrwx` `777` `r(4)w(2)x(1)`

反码:正数反码与原码相同,负数反码是除符号位外对其原码逐位取反

补码:正数补码与其原码相同,负数补码是在其反码末位加1

原码:二进制定点表示法,0(+) 1(-) 其余表示数值大小

使用变量前要初始化

```
int a, b = 10;
int c = a + b; //未初始化变量参与运算使 c 成为垃圾值
printf("%d\n", a);

printf("%lu\n", sizeof(float)); //4
printf("%lu\n", sizeof(long)); //8
```


数据类型：访问内存的方式

访问变量的过程：找到变量地址再根据变量类型(读取sizeof(type)个字节)访问变量内容

数据类型(关键字、所占字节数、数域范围、格式控制符)

格式控制符

%c char

%d int,short

%ld long

%f float,double

%s char[] 遇到空格或回车表示结束

%u unsigned int,unsigned short

%lu unsigned long

%p 指针

%hd short

%#o 八进制

%#x 十六进制

%02d 不足2位，前面补0，超出不影响

%.2lg 保留2位小数去掉无意义的0

++ / --：变量自增或自减，不能用于常量

++x 先执行递增运算，再计算表达式的值

x++ 先计算表达式值，再执行递增运算

```
printf("%d\t", i); //把i放到输出缓冲区，并不显示在屏幕上  
sleep(3); //睡眠3秒 <unistd.h>  
fflush(0); //强制输出缓冲区内容并清空，遇到\n也会输出|
```

```
scanf("%*c"); //清除键盘缓冲区
```

scanf("%*[\n]"); 清理回车之前的缓冲区

```
for (i=0; i<4; i++) {  
  
    scanf("%c",&arr[i]); //输入和输出都有字符缓冲区  
    printf("第%d个数组为%c\n",i,arr[i]);
```



```
}
```

```
//求积 可能溢出, 需要用比int更大的long来接收
printf("求积: %d*%d=%ld\n", number1, number2, number1*(long)number2);
//求商
printf("求商: %d/%d=%lf\n", number1, number2, number1/(double)number2);
```

变量使用常用错误汇总

- 1) 使用未经声明的变量。
- 2) 使用不符合C标识符命名规则的变量。
- 3) 使用未经初始化的变量。

表示long字面值, 需要以L或l结尾 `long a = 10000000000L;`

char类型使用常见问题汇总

- 1) 字符类型不可以存储中文。
- 2) char类型的值可以作为整数类型直接使用。
- 3) 表示char的字面值, 需要用单引号将字符引起来。

java char 占2字节, 采用Unicode C 中占1字节

```
char c = 'abc'; //单引号引多个字符时, 只保留最后一个
//c=' '; 只能引一个字符
printf("%c\n", c); //c
```

浮点类型 (float、double) 使用常见问题汇总

- 1) 浮点数的字面值为double类型。
 - 2) 浮点数存在舍入误差问题。
- float -> 4 double -> 8 long double -> 16
- `%f / %.2f %lf %Lf %g` : 去掉无意义的0

```
bool rs=a>b&&b<++c; //c没有++, 在编译时&&b<++c被删除了
```


转义字符 printf()

\a 振铃

\b 退格

\f 换页

\n \r 将光标移到行首

\t \v

\\ \? \' \"

%%

sizeof 应用

sizeof(bool)=1; //只有0&1

```
printf("表达式最终运算结果的类型所占空间: %lu\n", sizeof(1+0.6*2.1));  
printf("表达式x=1不会执行: %lu\n", sizeof(x=1));
```

运算符

```
//取余:余数的符号仅与被除数有关,与除数无关  
printf("%d,%d,%d", -5%3, 5%-3, -5%-3); //-2,2,-2  
printf("%d\n", 5.5%3); //浮点数没有余数
```

参数是调用程序传递给函数的值,只能是基本数据类型,引用类型需要用指针
声明一个指向目标数组的指针(数组第一个元素的地址)并传递给函数

```
int getRandom(int m, int n){  
    swap(&m, &n); //交换,保证m<n  
    srand((unsigned)time(0)); //种种子  
    return rand()%(n-m+1)+m; //返回(m~n)范围内的随机数  
}
```

数组乱序

```
for (int i=0; i<size; i++) { //1、arr[0] <--> arr[0~9]  
    int x = rand()%(size-i)+i; //2、arr[1] <--> arr[1~9]  
    swap(&arr[x], &arr[i]); //... 10、arr[9] <--> arr[9]
```

使用数组前要初始化且不能越界

```
int k[] = {[5]=100}; //求数组长度  
printf("数组长度: %lu\n", sizeof(k)/sizeof(k[0]));
```



```
int f[5] = {[2]=10, [4]=20}; //C99标准新增标准
```

二维数组：数组的集合

- 1、`int arr[2][2] = {...}; int arr[][2] = {...};` 依次赋值自动补0
- 2、数组名指向数组第0个元素的地址(常量指针)，且不可改变，不能赋值

函数

- ## 1、声明、定义、调用

实参与形参

值传递：实参赋值给形参后，形参在函数中发生变化不会影响实参(不会传回给实参)

地址传递：在函数中修改指针指向的变量可以改变主函数中的变量，且函数可返回多个值

- 1、无形参时，实参会被忽略，void时不可传入实参
- 2、有形参时，传入的实参个数必须匹配
- 3、实参类型与形参类型不一致是，以形参为准进行转换

变量作用域

- 1、变量的有效范围，何时创建何时销毁，由最近一对{}确定
- 2、局部变量 (auto) int a = 10; 不要返回局部变量地址，销毁后地址空间是公用的，会被下一
- 3、全局变量 作用域从定义行开始到文件结束
- 4、就近原则

auto

static: 静态变量, 只初始化一次, 延长生命周期到main()结束, 但作用域还是在函数内, 但可

register: CPU寄存器变量, 提高访问速度

extern

指针类型 64位机占8字节,32位机占4字节

char *ch : ch是一个指向char类型变量的指针，而不是一个类型为char的变量

```
int *p_rate; p_rate = &rate; *p_rate(间接取值)=rate(直接取值)
```

```
data[] data=&data[0]
```

```
int* p = data; p[i] = *(p+i) = *(data+i) = data[i]
```

增加 增加 减少

个局部变量征用

以用指针访问

指针递增、递减

指针预算: 赋值(&)、间接运算(*解除引用)、求地址、递增/减、求差、比较

$\text{int}^* + 1 = \text{地址} + 4$; $\text{char}^* + 1 = \text{地址} + 1$ 运算单位由指针类型决定

$*(\text{array}) == \text{array}[0] \dots *(\text{array} + n) == \text{array}[n]$

$\text{arr}[i] == *(\text{arr} + i) == *(i + \text{arr}) == i[\text{arr}]$

```
54 * 传过来的数组名是第0个元素的首地址，只需要用一个指针型的变量接收即可
55 * 所以需要给定数组的长度
56 */
57 void printArr(int* arr, int size){
58     printf("sizeof(arr)=%lu\n", sizeof(arr));
59     for (int i=0; i<size; i++) {
60         printf("%d\t", *(arr+i)+1); //arr[i]
```

进程空间: 进程是内存中正在运行的程序，在内存中占据的空间

1、代码区(text): 常量，存放程序代码，只读

2、全局区: 全局变量、静态变量(未初始化自动清0)，可读写，空间不会被释放，直到程序结束

3、栈区(stack):

4、堆区(heap):

字符数组: 字符类型的数组，每个数组元素保存一个字符

字符指针: 字符指针变量，指向字符串中的第一个字符的地址

```
char str2[] = "world"; //长度是6，代码区world复制到栈中的数组中
str2[0] = 'h';
char* str3 = "world"; //指向代码区world的首地址
//str3[0] = 'h'; //不可修改代码区字面量
str3 = "hello"; //重定向
```

```
char* str5; //str5未初始化，是垃圾值
scanf("%s", str5); //如果str5指向代码区，会崩溃
```

`char* name=NULL;`

—>`char name[100];`

`scanf("%10s", name);` //会崩溃，因为name不指向任何区域，输入内容没地方存放

```
int* pb = 100; //也是字面量，不可修改
*pb = 200; ⚠️ Unused variable 'pb'
```


野指针

为避免野指针，通常要将指针初始化为NULL,用完后也为其赋值NULL

给指针变量初始化：1、char* p = (char*)malloc(sizeof(char)); 2、char* p = NULL;

给数组初始化：int a[10] = {10}; memset(a, 0, sizeof(a));

结构体指针初始化：pstu = (struct student*)malloc(sizeof(struct student));

assert(NULL != p);

free(p); //p所指的内存被释放(从私有堆到共有堆)，但p所指的地址和指向堆空间的内容都不变

p=NULL;

字符串函数

1、字符串复制 strcpy(str, "hello"); strncpy(str, "world", 3);

2、字符串长度 strlen(str)

3、字符串拼接 strncat/strcat(str1, str2);将str2拼接到str1的后面，去掉str1后的\0

4、字符串比较 int rs = strcmp(str1, str2); Unicode比较

宏定义、宏函数

#define PI 3.14

__LINE__、__FILE__、__DATE__、__TIME__、__STDC__

#define UPPER(ch) ((ch) >= 'a' && (ch) <= 'z' ? (ch)-32 : ch)

#define PRINT(x, y) printf("#x"=%d, "#y"=%d\n", x, y)

#define SWAP(x, y, T) {T t = x; x = y; y = t;} //C语言实现泛型

注意：宏函数参数要用括号({}), 不要用++、-- 运算符

... 可变参数

连接符号

原样输出

/ 换行符

获取方法参数的宏__argc__() 检查宏参数个数取决于10...1，最多支持10个参数

它的好处不仅将计算在预处理时搞定，不拖延到运行时的cpu；更重要的是编译检查(参数个数)

#define _ARGCOUNT(...) _ARG_AT(_0, __VA_ARGS__, 10, 9, 8, 7, 6, 5, 4

#define _ARG_AT(N,...) _CONCAT##N(__VA_ARGS__)

◦
, 3, 2, 1)


```
#define _CONCAT_0(_0,_1,_2,_3,_4,_5,_6,_7,_8,_9,...) _HEAD_FIRST(__VA_ARGS__)
#define _HEAD_FIRST(first,...) first
```

条件编译

```
#if #else #endif #elif
```

```
#ifdef #ifndef #undef
```

多文件编程

```
*.h *.c -> main.c
```

```
#ifndef xxx_h //防止重复定义,编译xxx.c时会定义xxx_h,然后主函数中又会定义
```

```
#define xxx_h //#ifndef就可以跳过函数声明,防止重复定义
```

```
.....
```

```
#endif
```

文件操作

```
FILE* fp;
```

```
fp = fopen("xxx", "rb") == NULL
```

```
fclose(fp)
```

```
r(read)、w(write)、a(append)、t(text)、b(binary)、+(rw)
```

```
字符读写: fgetc(fp) fputc('a', fp)
```

```
字符串读写: fgets(str, 10, fp) fputs("abc", fp)
```

```
数据块读写: fread(buf, size, count, fp) fwrite
```

```
格式化读写: fscanf(fp, "%s", str) fprintf
```

```
rewind(fp); 将文件内部位置指针移到文件首
```

```
fseek(fp, 位移量, 起始点);
```

```
feof(fp)
```

```
ferror(fp)
```

结构体

1、自定义数据类型,可以包含不同类型的成员

```
typedef struct{
```

A_ARGS__)

```
    char name[100];  
    int age;
```

```
}Student;
```

2、不可整体赋值，只能逐个赋值

3、内存分配

4、结构体作为函数形参(值传递、地址传递)

5、结构体属于值类型，可以以指针形式存在，但规范用于值类型(基本数据类型)

```
Teacher* pt = (Teacher*)malloc(sizeof(Teacher));
```

```
Teacher t = {"亚里士多德", 40, "Philosophy"};
```

```
void print(Teacher t){  
    printf("sizeof(t)=%lu\n", sizeof(t));  
    printf("name=%s,age=%d,course=%s\n", t.name, t.age, t.course);  
}
```

```
void set(Teacher* t, char* name, int age, char* course){  
    t->name = name;  
    t->age = age;  
    t->course = course;  
}
```

```
set(pt, "斯宾诺莎", 38, "哲学");
```

//pt->name是指向代码区“斯宾诺莎”的指针，记住代码区内容不可修改

```
printf("@斯宾诺莎=%p,pt->name=%p\n", "斯宾诺莎", pt->name);
```

联合

```
typedef union{  
    int x;//size->4,共用同一块内存空间  
    char ch;  
}Union;
```

枚举

```
typedef enum {  
    SPRING = 1,  
    SUMMER,  
    AUTUMN,  
    WINTER  
} Season;
```

1、自定义数据类型，每个值都是整形常量，默认值从0开始并依次+1

2、只能在声明时初始化(修改枚举值)

void* 指针

```
void* p; //可以指向任何类型的数据
p = &x;
//先强转成int*的指针(指定访问内存方式), 在取值
printf("%d\n", *(int*)p);
//根据不同类型, 指定相应的指针内存访问方式(泛型)
void output(void* p, Type t);
```

函数指针

```
void(*f)() = function; //void->返回类型, *f->指针, ()->形参
double invoke(double(*f)(int, double), int x, double d){
    return f(x, d); //3个形参分别是: 函数指针, 形参 x、d
}
```

指针函数

```
int* func();
```

分配内存函数

- 1、malloc()
- 2、calloc() 分配的空间会自动清零
- 3、realloc() 堆申请的空间进行扩容

先判断当前指针是否有足够连续空间, 有则扩大mem_address,
否则新开辟空间, 将原有数据放到新区域并释放原区间内存

```
const int* p = &a; //指针指向的值变为只读, 不能*p=20, 可以a = 20
int* const p = &a; //指针本身变为只读
const int* const p = &a; //指针本身和所指向的值都不能改变
```

栈中变量值有2种修改方式: 1、a=10(直接修改) 2、*p=10(指针修改)

```
const int i = 10; //i在栈区, 编译时遇到i就直接改为10
int* p0 = &i;
*p0 = 20; //通过指针方式修改了i的值
printf("i=%d,*p0=%d\n", i, *p0); //10,20
```

数据类型之间转换

数值转换函数

- 1、`sprintf(str, "%lg", d);`//数值转换为字符串
- 2、`sscanf(strDouble, "%lf", &d);`//字符串转换为数值

可变参数

```
void printData(int num, ...){
    va_list ap;//声明list->ap <stdarg.h>
    va_start(ap, num);//将num个参数放入ap中
    for (int i=0; i<num; i++) {
        int arg = va_arg(ap, int);//根据可变参数的类型int分别取出
        printf("%d\t", arg);//打印可变参数
    }
    printf("\n");|
    va_end(ap);//销毁对象
}
```

日期时间函数

`clock_t t = clock();`//从程序开始运行到clock函数被调用之间的时间(微妙)

函数: `clock, difftime, mktime, time`

```
time_t t1;//time_t t1->long t1
time(&t1);//将当前秒数放入变量t1中
printf("%s", ctime(&t1));//ctime获取本地时间
printf("%s", asctime(gmtime(&t1)));//世界世界(England)
char buf[100];|
strftime(buf, 100, "%Y-%m-%d %H:%M:%S 星期%w", localtime(&t1));
```

