**c语言**

**python: <https://github.com/taizilongxu/interview_python>**

### 待处理

 int len = stringLength(s);

  char \*result = (char\*)malloc(len);

//这种声明字符串的写法看不懂. 但是和下面的效果一样.

 int len = stringLength(s);

  char\* result = malloc(len);

### 分号;

每一行代码结尾的分号; 一定不能少. 写完一段代码之后,先检查分号是不是齐全了.

### 数组

数组的声明: type arrayName[arraySize]

比如: int balance[5] = {1,2,3,4,5}//数组arr的长度5, 每个元素都是整数, 数组里的元素都是同一类型

int balance[5] = {1,2,3,4,5}

获取数组中的元素: arr[0], arr[1]

传递数组给函数:

void myFun(int \*arr)

void myFun(int arr[10]) 传递一个长度为10的数组

void myFunc(int arr[]) 传递一个长度未知的数组

### 字符串

: char是字符串类型

int stringLength(char \*string) {

  int len = 0;

  while (string[len] != '\0') {

    len += 1;

  }

  return len;

}

int

find(char\* s1, char s2) {

  for (int i = 0; i < stringLength(s1); i++) {

    if (s1[i] == s2) {

      return i;

    }

  }

  return -1;

}

char是变量值类型, (char \*string) 说明 \*string的类型是char, \*代表指针对应的地址. 那么string就是指针,

  printf("stringLength： %d\n", stringLength("aaaa") ); // 4

Char\* 代表char类型的指针, 那么传进来的s1本身就是指针. \*s1指向s1对应的值. s1[1]相当于\*(s+1)也就是s1指针里值的第二个字符.

C 语言中的字符串

C 语言中的字符串实际上是一个以 '\0' 结尾的特殊字符数组

（注意 '\0' 是一个特殊表示方法，因为打不出这个字符所以用 \0 表示）

声明的方式如下（注意必须使用双引号）

这一句声明了一个字符串（实际上是一个特殊字符数组）

它包含了 4 个字符（ a b c 和末尾自动加的 '\0'）

char \*s = "abc";

实际上内存中分配了 4 字节，s 是首字节的地址

由于 s 实际上只是一个地址（指针），并不能知道字符串总长度

所以 C 语言设计为 '\0' 表示结尾（否则不可能知道什么时候结束）

### 指针与数组

上面说了 s 是一个指针，又是一个特殊字符数组

实际上在很多情况下，可以把指针当作数组

s[0] 是 'a'，相当于 \*s

s[1] 是 'b'，相当于 \*(s+1)

s[2] 是 'c'，相当于 \*(s+2)

s[3] 是 '\0'，相当于 \*(s+3)

但是需要注意的是下面这样的写法是一个特殊写法

char \*s = "abc";

这样的情况下，只能读取 s[0] s[1] 不能写入 s[0]

比如 s[0] = 'z' 这样的写法是不可以的，不支持修改，只能读取

所有的字符串都是指针类型, 根本就没有所谓的字符串的变量值一说. 字符串就是一个字符数组, 字符数组的第一个元素的内存地址就是字符串的代表, 也是字符串的指针. 要往函数里传递时,只能传递这个指针, 然后在函数中, 挨个获取字符串里的字符.进行处理.所以字符串传递时必定是指针类型char\*, 字符传递是才是char类型

### 位运算符

位运算符作用于位，并逐位执行操作。&、 | 和 ^ 的真值表如下所示：

p q p & q p | q p ^ q

0 0 0 0 0

0 1 0 1 1

1 1 1 1 0

1 0 0 1 1

0xff == 255

### **杂项运算符 ↦ sizeof & 三元**

下表列出了 C 语言支持的其他一些重要的运算符，包括 sizeof 和 ? :。

运算符 描述 实例

sizeof() 返回变量的大小。 sizeof(a) 将返回 4，其中 a 是整数。

& 返回变量的地址。 &a; 将给出变量的实际地址,也就是变量的指针.

\* 指向一个变量。 \*a; 将指向一个变量。a是指针, \*a是指针对应的变量值.

? : 条件表达式 如果条件为真 ? 则值为 X : 否则值为 Y

&machineCode: machineCode指那个变量, &machineCode指变量的指针

\*a: a是指针, \*a就是指针对应的变量.

int var = 20; // var是变量值20;

int \*ip; // ip就是指针.

ip = &var; // 此时ip就是指针. &var获取20的内存地址.

printf("var 变量的地址： %p\n", &var ); //内存地址

printf("ip 变量的地址： %p\n", ip ); //内存地址

printf("ip 变量的值： %p\n", \*ip ); //变量值20

### malloc

// sizeof() 返回变量的大小。返回变量所占的空间位数,

//sizeof(int) 返回int类型占据的空间位数, 因为分高低位 两个数,所以 \*2.

// sizeof(int) \* 2 两个整数所占的内容空间,

// malloc(num) 申请num个内存空间,

// malloc前面的 (int\*) 应该是限制元素类型.

// malloc返回一个指向它的指针

int \*output = (int\*)malloc(sizeof(int) \* 2);

这里的output就是指针对应的值了.

“->”是一个整体，它是用于指向结构体子数du据zhi的指针，用来取子数据。

machineCode->len; 就类似于 machineCode.len;

char \*res = malloc(num). 申请num个内存空间用来保存数据, 返回申请的内存空间的收个空间的地址res, 内存空间里保存的数据格式是char字符, res就是字符串的指针.

### realloc

C 库函数 **void \*realloc(void \*ptr, size\_t size)** 尝试重新调整之前调用 **malloc** 或 **calloc** 所分配的 **ptr** 所指向的内存块的大小。

### **多个C语言文件的执行:**

main.c main.h utils.c utils.h 4个c文件

gcc main.c utils.c -o run.exe // 生成可执行文件run.exe

./run.exe // 执行run.exe文件

### **typedef**

typedef 关键字，您可以使用它来为类型取一个新的名字

size\_t 这是无符号整数类型，它是 sizeof 关键字的结果, 相当于+int. 正int

### strncat

char \*strncat(char \*dest, const char \*src, size\_t n)

把 src 所指向的字符串追加到 dest 所指向的字符串的结尾，直到 n 字符长度为止。

### 指针

char \*raw 我理解是这么个意思： 声明一个 raw 变量，这个变量是一个指向 char 类型的指针。 字符串在内存中是存在一块连续的内存空间里面的，要访问这 个字符串需要知道这块连续内存的起始地址 这个起始地址就是这个这个指针指向的地址。 然后一直读这个内存…读到 \0 就知道这个字符串结束了

struct \_aeList {

// \*buffer表示 buffer是这个连续内存空间的初始地址, \*\*buffer表示 这一块联系内存空间存储的是指针.

// 第二个\* 表示 buffer这个值本身是指针.

// 第一个\* 表示数组里存储的数据类型是 指针

void \*\*buffer;

size\_t length;

size\_t capacity;

};

如果一个指针,不知道他的具体的数据类型的话, 那么就没办法使用这个数据. 指针只是一个内存地址, 不知道具体类型, 后面的数据就不知道怎么解读, 也就没办法使用, 这种时候只能先用别的数据, 比如前后的数据判断好他的数据类型才能继续使用.

### **单引号和双引号, 字符和字符串**

大多数单引号表示的是一个字符，双引号表示的是表示一个字符串. 单引号和双引号是不能判断的.

“a” 和 ‘a’ 不相等.

“字符串” 的类型是 指针. 要往函数中传字符串的话, 参数类型限定为 指针.

int

ensureChar(char \*a, char \*b, char \*message) {

  if (a != b) {

    printf("测试失败 %s %s %s\n", a, b, message);

  } else {

    printf("测试成功\n");

  }

  return 1;

}

 ensureChar("a", "a", "双-双");

char a = ‘a’ // char是在数据类型里算 整数类型. -128 到 127 或 0 到 255.

// char类型只能是单个字符.代表这个字符本身, 和字符串是完全不一样的两个数据类型

char c; // 声明 char 变量

c = 'A'; // 定义 char 变量

printf("c 的值为 %c", c);

Void

showChar(char )

字符类型作为参数往函数中传的时候, 参数类型是 char,不是指针.

bool

applyCompare(int expression[2], char op) {

  if (op == '>') {

    return expression[0] > expression[1];

  } else {

    return false;

  }

}

bool r = applyCompare(testList, '=');

### **Bool类型**

#include <stdbool.h>

void

logBool(bool aBool) {

  printf("aBool %s\n", aBool ? "true": "false");

}

logBool(r);

#define 定义宏

C 预处理器不是编译器的组成部分，但是它是编译过程中一个单独的步骤。简言之，C 预处理器只不过是一个文本替换工具而已，它们会指示编译器在实际编译之前完成所需的预处理。我们将把 C 预处理器（C Preprocessor）简写为 CPP。

所有的预处理器命令都是以井号（#）开头。它必须是第一个非空字符，为了增强可读性，预处理器指令应从第一列开始。

宏延续运算符（\）

一个宏通常写在一个单行上。但是如果宏太长，一个单行容纳不下，则使用宏延续运算符（\）。例如：

#include <stdio.h>

#define message\_for(a, b) \

printf(#a " and " #b ": We love you!\n")

int main(void)

{

message\_for(Carole, Debra); // Carole and Debra: We love you!

return 0;

}

多行字符串宏

#define \_GuaStringMulti(s) #s

#define GuaStringMulti(s) \_GuaStringMulti(s)

char t[] = GuaStringMulti(

jump\_if\_great @label1\n

@label1\n

);

t == "jump\_if\_great @label1\n @label1\n "

### 枚举

enum DAY

{

MON=1,

TUE=2,

WED =3,

THU=4,

FRI=5,

SAT=6,

SUN=7

};

int main()

{

enum DAY day; // 声明枚举类型DAY, 变量day

day = WED;

printf("%d",day);

return 0;

}

### unsigned char

未声明的字符类型. 常用语保存 一个8位的二进制, 8位2进制,刚好一个字节, char类型也是一个字节.