**1 《C语言基础语法》**

**1 输入输出**

printf(“string\_1””string\_2”);

prinf(cptr);字符串指针

printf(“Hello World %d”, a); 也可以

scanf(“%a”, &a); 输入的是地址, 否则没法修改其内容 (本质是getchar()构造的函数)

**2 数据类型**

数据类型表示该类型的数据占用多少字节 (它们的存储方式), C语言里最小的数据占用1 byte.

数据扩展或截断时 (char ⬄ int): <data\_type>表示扩展时1或0, 和读取时最高位是符号还是数值

(Tip: 可以用char得到一个byte的内容, 数据的截断和扩展字位)

(数据是补码形式存储)

小端存储: 数据的低位存储在低地址, 高位存储在高地址

大端存储: 数据的高位存储在低地址, 低位存储在高地址

数据的存储方式：

static修饰的数据(或函数)只能在该文件内部访问(私有化), static数据变量记忆上次运算结果

extern修饰的数据(或函数)在file\_1声明, 在file\_2里定义, 从而可被其他file调用(最后在.h文件声明)

左移右移操作符只能对于正负整数, 不能对浮点数, 因此对于负整数而言左右移动的是他的补码

**按位操作符只是针对整数**

**3 运算符**

① 数值运算符: +, - \* /, %, +=, \*=, ++ …….(注意优先级)

② 逻辑运算符: &&, ||, !

③ 比较运算符: ==, !=, <=, >, ……

④ 位运算符: ~, &, |, ^, <<, >>

⑤ 三目运算符: A? B : C (当A成立时, 则B, 否则C)

转义字符: \字符表示其他含义。如\n, \0 ......

*(Note that: 上述A, B, C, …都是表达式)*

**4 基础语法**

(1) 顺序语句

逗号语句: (A, B, C, …, N) (运算从左向右, 最终是以D的结果)

运算按照上下顺序执行

switch(整型变量){

case A: 表达式; break;

……;

default: 表达式;

}

(2) 分支语句

if(判断表达式){……}

else if(判断表达式){……}

else{……}

(3) 循环语句

while(循环条件){……}

当循环条件!=0时才进入循环

for(初始条件; 判断条件; 变量改变) {……}

*Tip: 上述条件可以没有, 但要有 ;*

建议: 初始条件的变量在外面初始化, 如 int i = 0; for(i; i<10; i++){……}

do{……} while(循环条件);

当循环条件!=0时才进入循环

break结束所在的循环体

continue跳转出当前所在的循环, 进入所在的下一次循环

(4) go to语句

goto Label; ……; Label: 直接从goto跳转到Label然后执行后面的程序

Label: ……; goto Label: 直接从goto跳转到Label然后执行后面的程序

**2《C语言函数》**

**1 函数的结构**

① 创建函数: <data\_type> fun\_name(argument形参数列表){……; return result}

② 调用函数: fun\_name(实参数值); (得到的是该函数的返回值)

函数调用时, 在调用的空间里创建形参变量然后将赋值变量的值拷贝给这些形参, 因此函数里使用的是这些被赋了值的形参, 当函数使用完这些形参及时销毁, (函数里的参数都是局部变量存在栈区, 返回值是单独的寄存器, Ref: 函数栈帧)

当函数的创建在调用的后面, 则需要再调用之前声明该函数 <data\_type> fun\_name(arguments);

递归函数是指函数的返回值是该函数自己, 但实参值发生改变。

**3《C语言数组》**

**1 数组结构**

① 创建数组: <data\_type> arr\_name[num] = {初始化};

初始化一个值, 则全部值都一样 arr\_name[num] = {0或‘\0‘}, 数据类型: **<data\_type> [ ]**

如果<data\_type> arr\_name[ ] = {a, b, c}; 初始化后系统认为该数组有3个元素

② 数值的使用: arr\_name[i] = \*(arr\_name+i) = i[arr\_name];  **(数组名是首元素地址 = &首元素)**

对应的是数组的第i+1元素 (元素标号从0开始)

数组里的元素也可以是数组, 从而构成**多重数组**, 元素的使用arr\_name[i][j] = \*(\*(arr\_name+i)+j);

多重数组定义**只可以忽略第一个数量,** 如: arr[ ]**[2]** = {{e11, e12}, {e21, e22}, ……};

**2 字符串与数组关系**

① 定义字符串: const char\* str = “abc”; 等价char\* str = “abc”; 用const是为了规范

② 数组定义字符串: char str\_arr[ ] = “abc”; 等价char str\_arr[ ]={‘a’, ‘b’, ‘c’, ‘\0’};

Note that: 指针定义的字符串无法修改内容因为是const, 数组定义的可以修改内容

也可以char arr[]**[3]**={“ab”, “cd”}; 等价char arr[]**[3]**={{‘a’, ‘b’, ‘\0’},{‘c’, ‘d’,’\0’}};

关键字sizeof() 计算变量的字节(byte)大小 (对应函数的参数是数组, 但实际是数组名(首元素地址))

函数 strlen() 计算数组里元素的个数, 遇到 ’\0’ 才停止

"string\_1" == "string\_2"比较的是地址

应该: strcmp (string\_1, string\_2);

然而对于int arr[] = {1, 2, 3}; strlen(arr)≠3; 因为没有 ‘\0’

在C99之后可以int n = 5; int arr[n] = {……};

**4 《C语言指针》**

**1 指针**

① 指针变量(简称: 指针): <data\_type>\* ptr = &variable; 变量ptr存储的是变量variable的**首字节地址**

常数量 (直接嵌入内存) 和 宏(在于编译时被替换)没有地址

const变量 和 字符串有地址 *~~(char\* ptr\_1 = “abc”; char\* ptr\_2 = “abc”; ⇨ ptr\_1==ptr\_2)~~*

Note that:

* + - sizeof (指针) = 8byte (64位) 或4byte (32位)
    - 不同的数据类型占有的字节大小不同, 又有大小端存储, 因此首地址可能是这些数据的高位或地位字节
    - char只有1 byte从而没有大小端存储的区别
    - data\_type\* ptr = NULL; 该指针变量ptr为空值 (不指向任何地址, 也称为: 悬空指针)

② 指针的类型:

(1) 指针变量±1对应的是<data\_type>步长, 即地址移动sizeof (data\_type)步长

(2) \*ptr (解引用)后访问该首字节后面多少个字节, 即访问sizeof(data\_type)的空间

Note that: char是一个最小访问空间 (1 byte), 可以用它来强转得到首地址里的数据

**指针只要类型一样, 和运算步长一样就是同一类指针(如: 数组名==它内部元素类型的指针)**

void\* 没有具体类型的指针, 但可以接受任何类型的指针，但是不能解引用操作 (因为没有具体类

型就无法知道解引用后访问的内存大小，也不能±整数进行步长运算)，所以要在解引用前进行类型转换 (转换为某个具体类型)。

int a = 10;

强制转换只是临时改变了ap的使用方法, 但实际的ap还是void\*类型。

void\* ap = &a;

\*(int\*) ap;

const <data\_type>\* ptr = &variable; 无法修改ptr内部的数据由于是const

<data\_type>\* const ptr = &variable; 无法修改ptr的指向由于是const

const <data\_type>\* const ptr = &variable; 无法修改ptr指向和ptr地址里的数据内容

**2指针数组&数组指针**

① 指针数组:

<data\_type>\* ptr\_arr[num] = {ptr\_1, ptr\_2, ……}; 数组的元素是 <data\_type>\*

类型: <data\_type>\* [ ] ptr\_arr是变量名

* + - 数组名ptr\_arr是首元素地址⇨ ptr\_arr = &ptr\_1; 因此ptr\_arr + 1 = &ptr\_2;

如果是多重数组, ptr\_arr 是 首元素地址(首个子数组的地址, 即 &arr\_1)

* + - 结合数组的使用方法调用里面的元素: \*ptr\_arr[i] = \*\*(ptr\_arr+i) = \*i[ptr\_arr]; (元素是地址)

(作为函数的参数: int arr[] 或 int\* arr / int arr[][3] 或 int(\*arr)[], 都是首元素地址)

② 数组指针:

<data\_type> (\*arr\_ptr)[num] = &arr; arr\_ptr是数组arr[num]的指针变量

类型: <data\_type> (\* ) [ ]

* + - 数组指针arr\_ptr±1的步长是整个数组的字节长度 (sizeof(arr))
    - 解引用里面的元素: (\*arr\_ptr)[i] = \*((\*arr\_ptr) + i) = i[(\*arr\_ptr)]; ((\*arr\_ptr) = arr数组名)

Tip: int(\*(\*arr\_ptr)[10])[5]表示: 数组指针指向包含10个元素的数组arr, 每个元素也是数组指针(指向含有5个元素的数组指针arr\_sub)

**3 指针函数&函数指针**

① 指针函数:

<data\_type>\* ptr\_func(arguments){…..; return <data\_type>\* ptr;} 返回值是个指针

类型: <data\_type>\* (arguments) func为变量名

注意: 如int\* fun(){int x=10; return &x;}此处的返回值是一个野指针, 由于局部变量x在函数结束

后就销毁了, 导致该变量地址不存在

② 函数指针:

<data\_type> (\*func\_ptr)(arguments)= &func 或 func (函数名就是本体的指针)

此处: <data\_type> func(arguments){……}

类型: <data\_type> (\*)(arguments) func\_ptr是变量名

* + - 可以结合指针数组里面存放函数指针
    - 一般不要用void类型的函数指针强转化为其他类型的函数指针

回调函数: 函数funA通过它的函数指针funA\_ptr被另一个地方(可以是另一个函数通过函数指针参数)在某个情况下调用(funA)，则称函数A是这个地方的回调函数 (也称: 被该地方回调的函数)

**5《C语言动态内存》**