凌云物网智科实验室

Development

C 语 言 基 础

WE HAVE MADE GREAT ACHIEVEMENTS

C 程 序 开 发

01

C程序入门

C语言历史

Speech/Training Topics

C语言的起源以及类似C语言的编程语言的历史简直不要太漫长，我简单总结列表如下：

- 1.CPL（Combined Programming Language） - 1963
- CPL是1963年剑桥大学发明的

- 2.BCPL（Base Combined Programming Language） - 1967
- 剑桥的Matin Richards 对CPL做了简化，推出了BCPL

- 3.B（B Programming Language） - 1969
- 20世纪60年代，贝尔实验室的研究员Ken Thompson（肯·汤普森）对BCPL又做了改进发明了B语言，并使用B语言编了个游戏 - Space Travel。他想偷着玩自己编写的这个游戏，所以背着老板找到了台空闲的机器 PDP-7，但是这台机器没有操作系统，于是Thompson着手为PDP-7开发操作系统，后来这个OS被命名为 - UNIX。

- 4.C（C Programming Language） - 1972
- Ken Thompson在贝尔实验室的另外一个好基友Dennis Ritchie（丹尼斯·里奇）也想玩这个游戏，但因为自己的机器并不是PDP-7，这样就不能跟好基友一起愉快地玩耍了。于是他又在B的基础上设计出了能在不同机器之间移植的编程语言--C语言，随后他们一起用C语言重新改写了UNIX操作系统，从此以后，UNIX将在操作系统领域一统江湖。

- 5.C++（C plus plus Programming Language） - 1983
- 还是贝尔实验室的人，Bjarne Stroustrup（本贾尼·斯特劳斯特卢普） 在C语言的基础上推出了C++，它扩充和完善了C语言，特别是在面向对象编程方面。一定程度上克服了C语言编写大型程序时的不足。

- 6.Java（Java Programming Language） - 1995
- Sun公司的Patrick Naughton的工作小组研发了Java语言，主要成员是James Gosling（詹姆斯·高斯林）

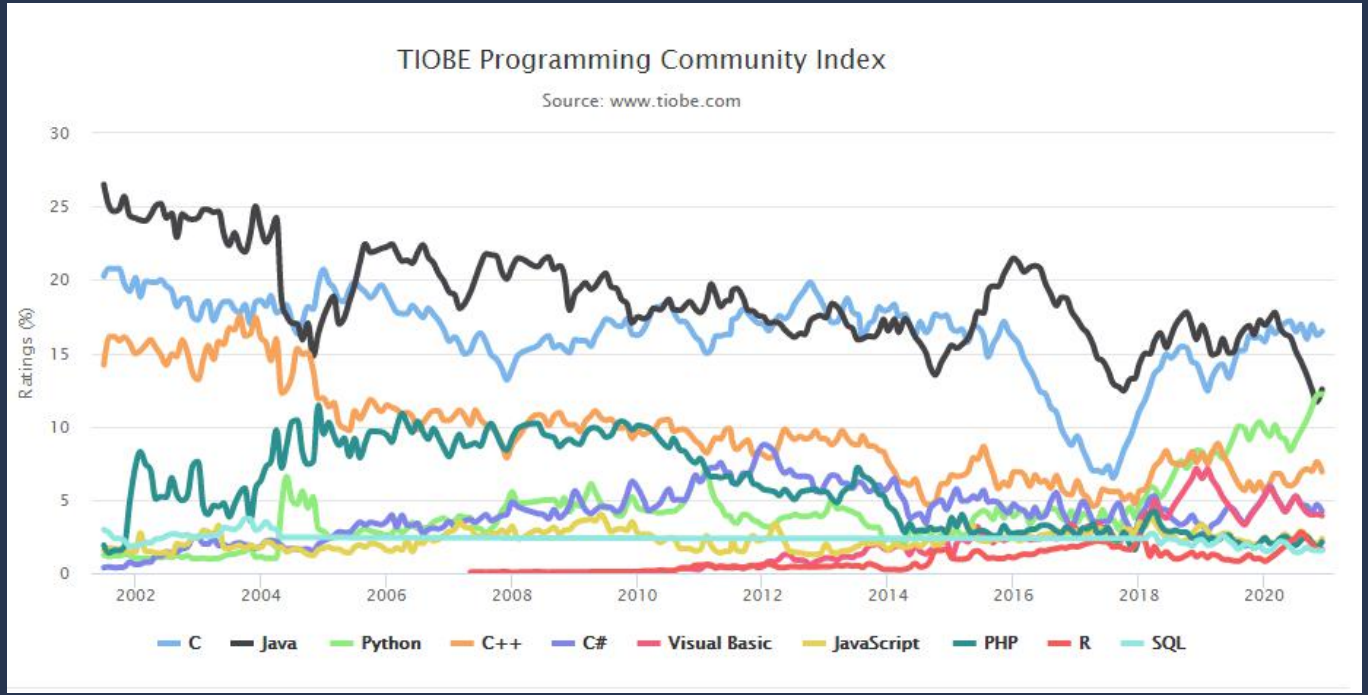
- 7.C#（C Sharp Programming Language） - 2000
- Microsoft公司的Anders Hejlsberg（安德斯·海尔斯伯格）发明了C#，他也是Delphi语言之父。

编程语言排行榜

Speech/Training Topics

C语言从1972年伴随着UNIX而生以来，由于其与硬件打交道的独特能力以及编程语言性能优势，在编程语言排行榜中都一直名列前茅。即使到了2020年的今天，随着5G、无人驾驶等物联网技术蓬勃发展，C语言在编程语言趋势排行榜TIOBE上再次排行第一！

https://www.tiobe.com/tiobe-index/

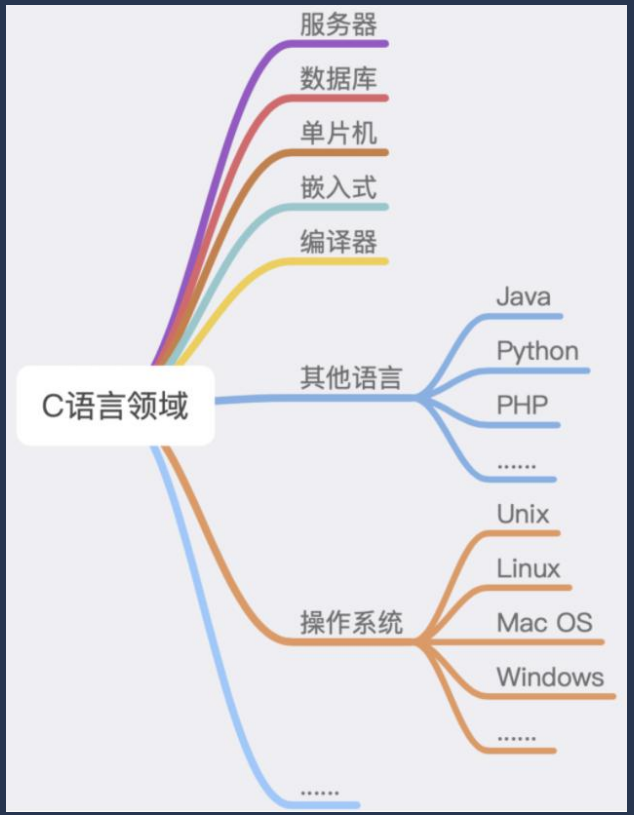


C语言应用领域

Speech/Training Topics

如果你问我，C语言有多伟大？那么，我可能会想一下说：多伟大我不知道，但是我知道很伟大：凡是带电的地方，可能都会有她的影子！

任何比C语言更低级的语言，都不足以完整地抽象一个计算机系统；任何比C高级的语言，都可以用C来实现！



为什么选择C程序

Speech/Training Topics

在大学里学习编程很多都是选择C语言作为入门语言，也有很多人在刚开始学习C语言觉得入门很简单，但是深入一点特别是到指针就觉得力不从心了。很多人的C语言学习也只是从入门到放弃，最后转投其他语言，更有甚者因此打击了自己的学习编程的学习信心，就直接放弃自己的相关专业了！

有那么多编程语言，为什么我们还要选择学习C语言呢？

- C语言除了能让你了解编程的相关概念，还能让你明白程序的运行原理，比如，计算机的各个部件是如何交互的，程序在内存中是一种怎样的状态，操作系统和用户程序之间有着怎样的“爱恨情仇”，这些底层知识决定了你今后的职业发展高度；

- C语言在高级编程语言史上足够年长，它是开发其他高级编程语言的参考，许多高级编程语言都借鉴了C语言的架构，或者干脆就是C语言编写的。所以学习了C语言不仅仅能够编写C语言程序，对学习其他编程语言也是帮助极大的。可以说，如果把学习编程比作小说里的学习武功，那学会了C语言就相当于学会了“九阳神功”，再学习其他编程语言时就如有神助，事半功倍。

- C语言是操作系统及硬件开发开发的首选原生（native）编程语言，这使得在这些系统中运行的C程序具有极大灵活性和可移植性。正是如此，经过这么多年的洗礼，C语言依然没有被淘汰，相反却稳步发展，并且C语言在未来很长一段时间里，是不可能被淘汰的。如果你是物联网、电信、电子、自动化控制等弱电相关专业的同学，那恭喜你C语言就是你的不二选择了；

- 由于C语言的指针比较晦涩难懂，所以很多人从入门到放弃。另外其它的一些编程语言学习比C语言简单多了，这就导致大家都一窝蜂地去学习Python、Java等其它火热的编程语言，在职场上形成极大的“内卷”。相反由于物联网行业的爆发式增长，现在企业对C程序员的招聘却越来越难了。

- 如果大家今后打算从事C语言的研发工作，从职业发展上来看的话，“墙裂”建议大家务必还要深入学习一下C++这门面向对象的编程语言。从某种角度看来，C语言与C++这两个编程语言在一起就是系统级开发的倚天剑与屠龙刀；

C程序开发

Speech/Training Topics

1

C程序编写

C程序是程序员使用编辑器编写的C语言程序代码，它是程序员按照C语言的语法规则对计算机控制的逻辑表达，但计算机并不能直接执行。

```c
#include <stdio.h>

#define PRINT_TIMES     3   // 宏定义: 需要打印多少次

int main(void)
{
    int             i;          // 定义了一个整形类型的变量 i , 占4个字节存储空间
    char            ch = 0x36;  // 定义了一个char类型的变量 ch, 占1个字节存储空间

    /* for 循环打印字符, 具体打印多少次由 PRINT_TIMES 决定*/
    for(i=0; i<PRINT_TIMES; i++)
    {
        printf("%c", ch);
    }

    printf("\n");

    return 0;
}
```

2

C程序编译

C程序必须要由编译器编译生成CPU所能识别的二进制可执行文件才能运行。C程序编译成功也只能说明C程序符合C语言的语法规则，程序的逻辑错误编译器并不能检测出来。

```
guowenxue@ubuntu-master:~$
guowenxue@ubuntu-master:~$ gcc hello.c -o hello
guowenxue@ubuntu-master:~$
guowenxue@ubuntu-master:~$ ls hello*
hello  hello.c
guowenxue@ubuntu-master:~$
guowenxue@ubuntu-master:~$ file hello
hello: ELF 64-bit LSB  executable, x86-64, version 1 (SYSV), dynamically
linked (uses shared libs), for GNU/Linux 2.6.24, BuildID[sha1]=184d030c5f
98504d24d366d851d3abe266a4733c, not stripped
guowenxue@ubuntu-master:~$
```

3

C程序运行

编译生成的程序可以运行，程序运行时出错或没有实现程序设计的目标，这是代码的逻辑错误，与C程序编译无关。换而言之，程序编译正常但未必能正常执行！

```
guowenxue@ubuntu-master:~$
guowenxue@ubuntu-master:~$ ./hello
666
guowenxue@ubuntu-master:~$
guowenxue@ubuntu-master:~$
```

C程序结构

Speech/Training Topics

1

预处理指令

所有以 # 开头的行表示预处理指令，在编译阶段由预处理器处理。

2

代码注释

C语言有两种注释方式，即代码块注释 /* */ 和 单行注释 // 两种；也可以使用预处理条件编译指令 #if 0 ... #endif 实现代码块注释；

3

函数

函数是一组一起实现某个具体功能的语句。每个 C 程序都至少有一个函数，即主函数 main() ，在编程时我们应该把每个基本功能封装成一个函数。

4

语句与表达式

语句和表达式C程序的基本单位，每条语句以； 结束。

```c
#include <stdio.h>

#define PRINT_TIMES      3   // 宏定义: 需要打印多少次

int main(void)
{
    int             i;          // 定义了一个整形类型的变量 i , 占4个字节存储空间
    char            ch = 0x36;  // 定义了一个char类型的变量 ch, 占1个字节存储空间

    /* for 循环打印字符, 具体打印多少次由 PRINT_TIMES 决定*/
    for(i=0; i<PRINT_TIMES; i++)
    {
        printf("%c", ch);
    }

    printf("\n");

    return 0;
}
```

软件开发流程

Speech/Training Topics

1

需求分析

要解决的问题进行详细的分析，弄清楚问题的要求。对应岗位产品经理。

2

概要设计

根据用户交互过程和用户需求来确定交互框架和模块。对应岗位系统架构师或项目经理。

3

详细设计

选择一种编程语言，实现每个模块的算法和流程图(visio或yed)。对应岗位系统架构师或项目经理。

4

程序编写

使用相应编程语言实现相应的功能函数。对应岗位码农。

5

编译运行

编译程序并运行测试、调试。对应岗位码农。

6

运行测试

程序运行、功能/压力测试。对应岗位软件测试工程师。

7

编译运行

程序长时间工作并运营维护。对应岗位运维工程师。

02

变量与基本数据类型

变量

Speech/Training Topics

C程序里的变量其实是程序可操作的存储区的名称，C 中每个变量都要有特定的类型，而类型决定了变量的存储空间大小和内存布局；

C程序里变量的名称由字母、数字和下划线字符组成，但必须以字母或下划线开头，此外C程序对大小写敏感。

在变量或函数命名时我们应该遵循“见其名，知其义”的原则，如体重用变量名 weight 比 var 更容易理解；

对于多个单词组成的变量命名一般有两种组合方法：

- 驼峰式命名法(Windows程序员)： printEmployeePaychecks();
- 下划线命名法(Linux程序员)： print_employee_paychecks();

```c
#include <stdio.h>

#define PRINT_TIMES      3   // 宏定义: 需要打印多少次

int main(void)
{
    int             i;          // 定义了一个整形类型的变量 i , 占4个字节存储空间
    char            ch = 0x36;  // 定义了一个char类型的变量 ch, 占1个字节存储空间

    /* for 循环打印字符, 具体打印多少次由 PRINT_TIMES 决定*/
    for(i=0; i<PRINT_TIMES; i++)
    {
        printf("%c", ch);
    }

    printf("\n");

    return 0;
}
```

基本数据类型

Speech/Training Topics

C语言并没有明确规定每种基本数据类型的大小，而只是保证short不会比int长, long不会比int短，具体他们的大小由系统和编译器决定。

类型	存储字节	取值范围
char / unsigned char	1字节	-128~127 / 0~255
short / unsigned short	2字节	-32768~32767 / 0~65535
int / unsigned int	2字节或4字节	-32,768 ~ 32,767 或 -2,147,483,648 ~ 2,147,483,647
long / unsigned long	4字节	
long long / unsigned long long	8字节	
float	4字节	精度：6 位小数
double	8字节	精度：15 位小数
long double	16字节	精度：19 位小数

typedef

Speech/Training Topics

typedef 为C语言的关键字，作用是为一种数据类型定义一个新名字，这里的数据类型包括内部数据类型（int，char等）和自定义的数据类型（struct等）。

typedef 的一个重要用途是定义机器无关的类型。在8位单片机如STM8上，其int类型为2个字节；而在32位单片机STM32上，其int类型为4个字节。如果我们写的代码希望能够在两个不同的处理器上保持可移植性，则可以使用typedef来重新定义新的类型。

STM8单片机：

```
typedef   signed char     int8_t;
typedef   signed short    int16_t;
typedef   signed long     int32_t;

typedef unsigned char     uint8_t;
typedef unsigned short    uint16_t;
typedef unsigned long     uint32_t;
```

STM32单片机：

```
typedef   signed char     int8_t;
typedef   signed short    int16_t;
typedef   signed int      int32_t;

typedef unsigned char     uint8_t;
typedef unsigned short    uint16_t;
typedef unsigned int      uint32_t;
```

这时我们如果想使用4个字节的有符号类型的存储空间就使用 int32_t，无符号类型则用uint32_t，这样可以保证不同的处理器间的代码可移植性。

03

构造数据类型

构造数据类型

Speech/Training Topics

1

数组(array)

数组是用来存储一组具有相同类型数据的顺序集合，所有的数组都是由连续的内存位置组成。最低的地址对应第一个元素，最高的地址对应最后一个元素。

2

枚举(enum)

枚举其实是 C 语言中的一种基本数据类(int类型)，它可以让数据更简洁，更易读。
枚举第一个成员如果不赋值的话默认为 0，后续成员的值是前一个成员上加 1。

```
enum
{
    MON=1, TUE, WED, THU, FRI, SAT, SUN,
};
```

3

联合(union)

联合(共用体)是一种特殊的数据类型，它允许您在相同的内存位置存储不同类型的数据。这样可以定义一个带有多成员的共用体，通过不同的成员访问不同的存储空间。

```
union
{
    unsigned char     var1;
    unsigned short    var2;
    unsigned int      var3;
};
```

4

结构体(struct)

结构体是 C 语言中另一种用户自定义的可用的数据类型，它允许您存储不同类型的数据，结构体里的成员按定义的先后顺序存放。

```
typedef struct student_s
{
    char      name[32];
    int       age;
    float     score;
} student_t;

struct  student_s    student1;
student_t            student2;
```

大小端字节序(笔试必考)

Speech/Training Topics

计算机硬件有两种储存数据的方式：小端字节序（little endian，LSB）和 大端字节序（big endian，MSB）

- 小端字节序：低位字节在前，高位字节在后；绝大部分处理器和系统都是小端字节序。
- 大端字节序：高位字节在前，低位字节在后；网络通信采用大端字节序通信。

int var = 0x12345678;

内存地址	LSB	MSB
…. ….		
0x3000 0003	0x12	0x78
0x3000 0002	0x34	0x56
0x3000 0001	0x56	0x34
0x3000 0000	0x78	0x12
…. ….		

判断大端还是小端字节序代码

```c
#include <stdio.h>

#define LSB             1
#define MSB             0

int is_lsb(void)
{
    union
    {
        unsigned char       a;
        unsigned int        b;
    } var;

    var.b = 0x12345678;

    if( var.a == 0x78 )
        return LSB;
    else
        return MSB;
}

int main(void)
{
   printf("System endianness is %s\n", is_lsb() ? "LSB" : "MSB" );
   return 0;
}

/*+-------------------------------------------------------------------+

 $ gcc lsb.c -o lsb
 $ ./lsb
 System endianness is LSB

 +-------------------------------------------------------------------+*/
```

结构体数组

Speech/Training Topics

计算机硬件有两种储存数据的方式：小端字节序（little endian，LSB）和 大端字节序（big endian，MSB）

- 小端字节序：低位字节在前，高位字节在后；绝大部分处理器和系统都是小端字节序。
- 大端字节序：高位字节在前，低位字节在后；网络通信采用大端字节序通信。

int var = 0x12345678;

内存地址	LSB	MSB
…. ….		
0x3000 0003	0x12	0x78
0x3000 0002	0x34	0x56
0x3000 0001	0x56	0x34
0x3000 0000	0x78	0x12
…. ….		

判断大端还是小端字节序代码

```c
#include <stdio.h>

#define LSB             1
#define MSB             0

int is_lsb(void)
{
    union
    {
        unsigned char       a;
        unsigned int        b;
    } var;

    var.b = 0x12345678;

    if( var.a == 0x78 )
        return LSB;
    else
        return MSB;
}

int main(void)
{
   printf("System endianness is %s\n", is_lsb() ? "LSB" : "MSB" );
   return 0;
}

/*+-------------------------------------------------------------------+

 $ gcc lsb.c -o lsb
 $ ./lsb
 System endianness is LSB

 +-------------------------------------------------------------------+*/
```

04

字符串与字符串函数

字符串

Speech/Training Topics

在C语言中并没有C++或Java中的string这个类型来表示字符串，在C中字符串实际上是使用 \0 结尾的一维字符数组。下面的声明和初始化创建了一个 LingYun 字符串。由于在数组的末尾存储了空字符，所以字符数组的大小比单词 LingYun 的字符数多一个。

char str[8] = {'L', 'i', 'n', 'g', 'Y','u','n', '\0'};　　或　　char str[]=”LingYun”;

而我们知道，字符在计算机中都是以ASCII值的形式来存储(因为计算机只能存储并识别0、1二进制数)，所以他们在计算机中的存储为：

数组索引	0	1	2	3	4	5	6	7
字符变量	'L'	'i'	'n'	'g'	'Y'	'u'	'n'	'\0'
存储值	0x4c	0x69	0x6e	0x67	0x59	0x75	0x6e	0x00

```
guowenxue@ubuntu-master:~$
guowenxue@ubuntu-master:~$ gcc str.c -o str
guowenxue@ubuntu-master:~$ ./str
String Mode output     : LingYun
Character Mode output : L i n g Y u n
Hex Mode output        : 0x4c 0x69 0x6e 0x67 0x59 0x75 0x6e 0x00
int Mode output        : 76 105 110 103 89 117 110 0
guowenxue@ubuntu-master:~$
```

```c
#include <stdio.h>

int main(int argc, char **argv)
{
    int         i;
    char        str[8]="LingYun";

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

    printf("Character Mode output : ");
    for(i=0; i<8; i++)
    {
        printf("%c ", str[i]);
    }
    printf("\n");


    printf("Hex Mode output        : ");
    for(i=0; i<8; i++)
    {
        printf("0x%02x ", str[i]);
    }
    printf("\n");

    printf("int Mode output        : ");
    for(i=0; i<8; i++)
    {
        printf("%d ", str[i]);
    }
    printf("\n");

    return 0;
}
```

字符串函数

Speech/Training Topics

C语言基础库里提供了很多字符串的操作函数，并且在笔试面试中经常会考到这些函数。以char str[]=”LingYun”为例讲解这些函数：

sizeof(str)： sizeof其实是一个关键字(并不是一个函数)，它是用来求一个存储空间的大小。sizeof(str)求str存储空间的大小其值为8(包含'\0'这个存储空间)；
strlen(str)： strlen()是用来求字符串长度的函数，在求长度时以碰到'\0'就结束，所以strlen(str)其值为7(不包含\0)。如果char str[7]=”LingYun”则长度不确定了，想想为什么!

strcpy(str1, str2)： 如果想把一个字符串赋值给另外一个字符串，不可以使用 str1=str2; 而需要使用字符串拷贝函数strcpy()，它用来将一个字符串拷贝到另外一个字符串中去。如strcpy(buf, str)把str拷贝到buf中去，这时候要注意buf要有足够的空间来容纳str，否则buf会溢出导致整个程序终止。所以这个函数不是安全的函数，一般建议调用strncpy，如strncpy(buf, str, sizeof(buf))

strcat(str1, str2)： 字符串连接函数，用来将str2连接到str1后面去。对于该函数使用同样要注意 str1 要有足够的空间容纳这两个字符串的长度，否则会溢出；一般建议使用strncat()；

strcmp()、strncmp()、strcasecmp()、strncasecmp()； 比较两个字符串的内容是否相同并不能使用 str1 == str2 来判断，而是使用这几个函数来判断。其中strcmp()和strncmp() 区分大小写，而strcasecmp()和strncasecmp()不区分大小写。他们在工作时只是简单地拿第一个字符串的每个字符与第二个字符串的相应字符做减法，如果不为0 则返回差值。所以如果返回值为0，则表示两个字符串相同!

strchr(s1, ch); 查找并返回字符ch在s1中第一次出现的位置，如果没找到则返回NULL； strrchr(s1, ch) 查找并返回字符ch在s1中最后一次出现的位置；

strstr(s1, s2); 查找并返回字符串s1在字符串s1中第一次出现的位置，如果没有找到则返回NULL； 如果不分大小写查找字符串则使用 strcasestr()函数；

sprintf()、snprintf() 用来格式化生成一个字符串，它的工作方式与printf()相同，只是printf把字符串打印到标准输出上，而sprintf()则输出到buf中。如我们想生成产品序列号：
snprintf(sn_buf, sizeof(sn_buf), “LingYun-%02d-%04d”, week, serial);

数组索引	0	1	2	3	4	5	6	7
字符变量	'L'	'i'	'n'	'g'	'Y'	'u'	'n'	'\0'
存储值	0x4c	0x69	0x6e	0x67	0x59	0x75	0x6e	0x00

05

运算符

算术运算符

Speech/Training Topics

运算符是一种告诉编译器执行特定的数学或逻辑操作的符号。C 语言内置了丰富的运算符，并提供了以下类型的运算符：

运算符
算术运算符
比较运算符
逻辑运算符
位运算符
赋值运算符
杂项运算符

下表显示了 C 语言的所有算术运算符。假设A 的值为 20， B 的值为13，则：

运算符	描述	实例
+	把两个操作数相加	A + B 将得到 33
-	第一个操作数中减去第二个操作数	A - B 将得到 7
*	把两个操作数相乘	A * B 将得到 260
/	分子除以分母	A / B 将得到 1
%	取模运算符，整除后的余数	A % B 将得到 7
++	自增运算符，整数值增加 1	A++ 将得到 21
--	自减运算符，整数值减少 1	A-- 将得到 19

比较与逻辑运算符

Speech/Training Topics

下表显示了 C 语言的所有比较运算符。假设A 的值为 20， B 的值为13，则：

运算符	描述	实例
==	检查两个操作数的值是否相等，如果相等则条件为真。	(A == B) 为假。
!=	检查两个操作数的值是否相等，如果不相等则条件为真。	(A != B) 为真。
>	检查左操作数的值是否大于右操作数的值，如果是则条件为真。	(A > B) 为真。
<	检查左操作数的值是否小于右操作数的值，如果是则条件为真。	(A < B) 为假。
>=	检查左操作数的值是否大于或等于右操作数的值，如果是则条件为真。	(A >= B) 为真。
<=	检查左操作数的值是否小于或等于右操作数的值，如果是则条件为真。	(A <= B) 为假。

下表显示了 C 语言支持的所有关系逻辑运算符。假设 A 的值为 1， B 的值为 0，则：

运算符	描述	实例
&&	称为逻辑与运算符。如果两个操作数都非零，则条件为真。	(A && B) 为假。
||	称为逻辑或运算符。如果两个操作数中有任意一个非零，则条件为真。	(A || B) 为真。
!	称为逻辑非运算符。用来逆转操作数的逻辑状态。如果条件为真则逻辑非运算符将使其为假。	!(A && B) 为真。

赋值运算符

Speech/Training Topics

下表列出了 C 语言支持的赋值运算符：

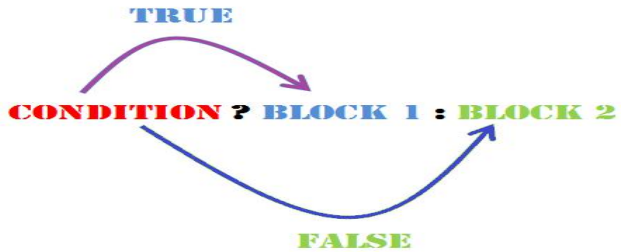
运算符	描述	实例
=	简单的赋值运算符，把右边操作数的值赋给左边操作数。	C = A + B 将把 A + B 的值赋给 C。
+=	加且赋值运算符，把右边操作数加上左边操作数的结果赋值给左边操作数。	C += A 相当于 C = C + A
-=	减且赋值运算符，把左边操作数减去右边操作数的结果赋值给左边操作数。	C -= A 相当于 C = C - A
*=	乘且赋值运算符，把右边操作数乘以左边操作数的结果赋值给左边操作数。	C *= A 相当于 C = C * A
/=	除且赋值运算符，把左边操作数除以右边操作数的结果赋值给左边操作数。	C /= A 相当于 C = C / A
%=	求模且赋值运算符，求两个操作数的模赋值给左边操作数。	C %= A 相当于 C = C % A
<<=	左移且赋值运算符	C <<= 2 等同于 C = C << 2
>>=	右移且赋值运算符	C >>= 2 等同于 C = C >> 2
&=	按位与且赋值运算符	C &= 2 等同于 C = C & 2
^=	按位异或且赋值运算符	C ^= 2 等同于 C = C ^ 2
|=	按位或且赋值运算符	C |= 2 等同于 C = C | 2

其它运算符

Speech/Training Topics

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()	它是一个关键字运算符，并不是一个函数，其返回所求对象的存储空间大小。	sizeof(a) 将返回 4，其中 a 是整数。
&	取地址运算符，返回变量的地址，指针中经常用到。	&a 获取变量a的地址，可以赋值给一个指针。
*	取值运算符，表示取指针指向内容的值，在定义时表示定义一个指针。	int a=10; int *p = &a; int b=*p;
? :	C语言中唯一的一个三目运算符。	max = a > b ? a : b ;



位运算符

Speech/Training Topics

位运算符作用于位，并逐位执行操作。&、 | 和 ^ 的真值表如下所示：

&	0	1
0	0	0
1	0	1

|	0	1
0	0	1
1	1	1

^	0	1
0	0	1
1	1	0

位运算符作用于位，并逐位执行操作。~、<< 和 >> 的说明如下，假设unsigned char A=0x04：

运算符	描述	实例
~	取反运算符，按二进制位进行"取反"运算。	~A = 0xFB
<<	二进制左移运算符。将一个运算对象的各二进制位全部左移若干位（左边的二进制位丢弃，右边补0）。	A<<2 = 0x10。
>>	二进制右移运算符。将一个数的各二进制位全部右移若干位，正数左补0，负数左补1，右边丢弃。	A>>2 = 0x1

与、或、亦或的应用：

- 0跟任何数(0或1)做与运算(&)其值为0， 而1则不变 => 通常用与运算用来清除某个位;
- 1跟任何数(0或1)做或运算(|)其值为1， 而0则不变 => 通常用或运算用来设置某个位;
- 1跟任何数(0或1)做亦或运算(^)其值为1， 而0则不变 => 通常用亦或运算用来反转某个位;

移位运算和取反运算符的应用：

- 某个数左移N位其实就是 让这个数乘以2^N ;
- 某个数右移N位其实就是 让这个数除以2^N;
- 取反运算符经常配合 或、亦或运算符使用，用来将指定位设置为1或反转

优先级与结合律

Speech/Training Topics

类别	运算符	结合律
后缀	() [] -> . ++ --	从左到右
一元	+ - ! ~ ++ -- (type) * & sizeof	从右到左
乘除	* / %	从左到右
加减	+ -	从左到右
移位	<< >>	从左到右
比较	< <= > >=	从左到右
相等	== !=	从左到右
位与	&	从左到右
位异或	^	从左到右
位或	|	从左到右
逻辑与	&&	从左到右
逻辑或	||	从左到右
条件	? :	从右到左
赋值	= += -= *= /= %= >>= <<= &= ^= |=	从右到左
逗号	,	从左到右

06

控制语句

条件判断

Speech/Training Topics

C语言默认并没有布尔类型（true/fasle），它把零或 null 认为假，而所有其它任何非零和非空的值认为真。我们可以使用if和else关键字对条件进行判断，当判断条件成真时执行{ } 里的代码。在编写代码时注意编程规范，最好每一个条件判断都应该加上{ }，即使执行语句只有一行也应如此。

{ } 一定要缩进(一般用 TAB 或 4 个空格来缩进)、对齐，另外国内程序员一般喜欢 { 另起一行。

```
.... ....

if( bool_expression )
{
      do_something;
}


if( bool_expression ) {

      do_something;
}

.... ....
```

```
.... ....

if( bool_expression1 )
{
      do_something1;
}
else if( bool_expression2 );
{
      do_something2;
}
else
{
      do_something3;
}

.... ....
```

```
.... ....
if( bool_expression1 )
{

    do_something1;

    if( bool_expression2 )
    {
       do_something2;
    }

    do_something3;
}

.... ....
```

```
if( var = 10 )
   do_something;
```

```
if( a != b )
if( a> b )
do_something1
else
do_something2
```

```
if( a > b );
{
    printf("a>b\n");
}
```

switch/case语句

Speech/Training Topics

一个 switch 语句允许测试一个变量等于多个值时的情况，每个值称为一个 case，且被测试的变量会对每个 switch case 进行检查;

switch 语句中的 expression 是一个常量表达式，必须是一个整型类型数据(char 或 enum 本质上也是整形类型)；

当被测试的变量等于 case 中的常量时，case 后跟的语句将被执行，直到遇到 break 语句为止；

不是每一个 case 都需要包含 break。如果 case 语句不包含 break，控制流将会 继续 后续的 case，直到遇到 break 为止。

switch 语句可以有一个可选的 default (也可以没有)，出现在 switch 的结尾，default 可用于在上面所有 case 都不为真时执行的任务；

```
char  grade = 'B'

switch( grade )
{
    case 'A':
       printf("Great!");
       break;

    case 'B':
    case 'C':
       printf("Good!");

    case 'D':
       printf("Passed!");
       break;

    default:
       printf("Lost!");
       break;
}
```

循环语句

Speech/Training Topics

C语言中的循环语句允许我们多次执行一个语句或代码块，其有三种循环方式：

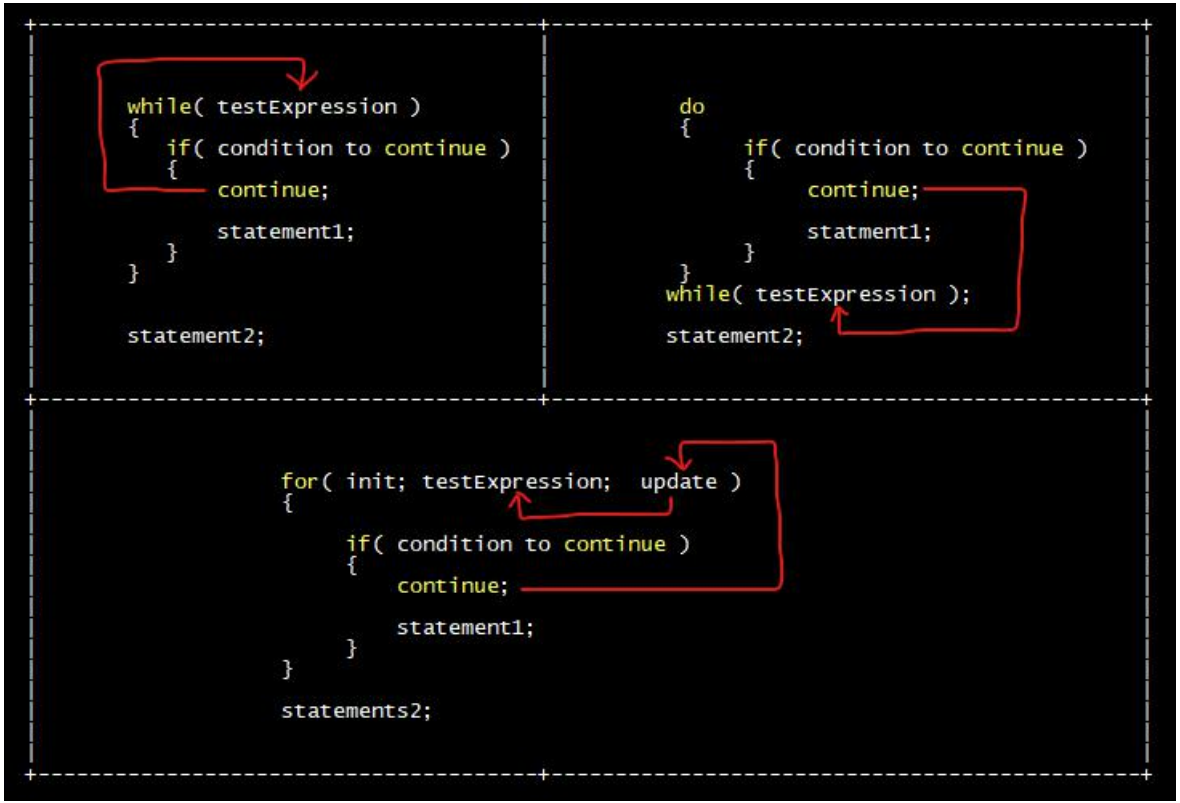
```
.... ....
while( testExpression )
{
        statements;
        update;
}
.... ....
```

```
.... ....
do
{
        statments;
        update;
}
while( testExpression );
.... ....
```

```
.... ....
for( init; testExpression;  update )
{
        statements;
}
.... ....
```

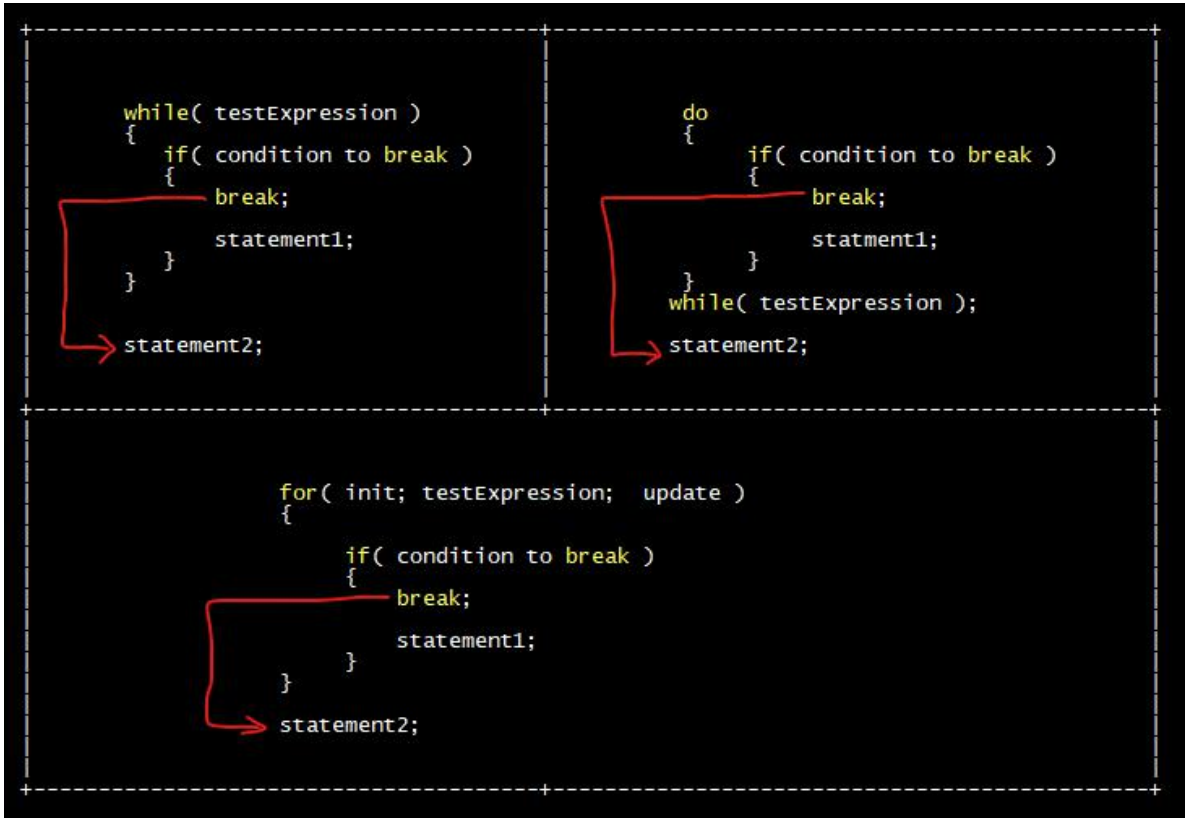
continue语句

Speech/Training Topics



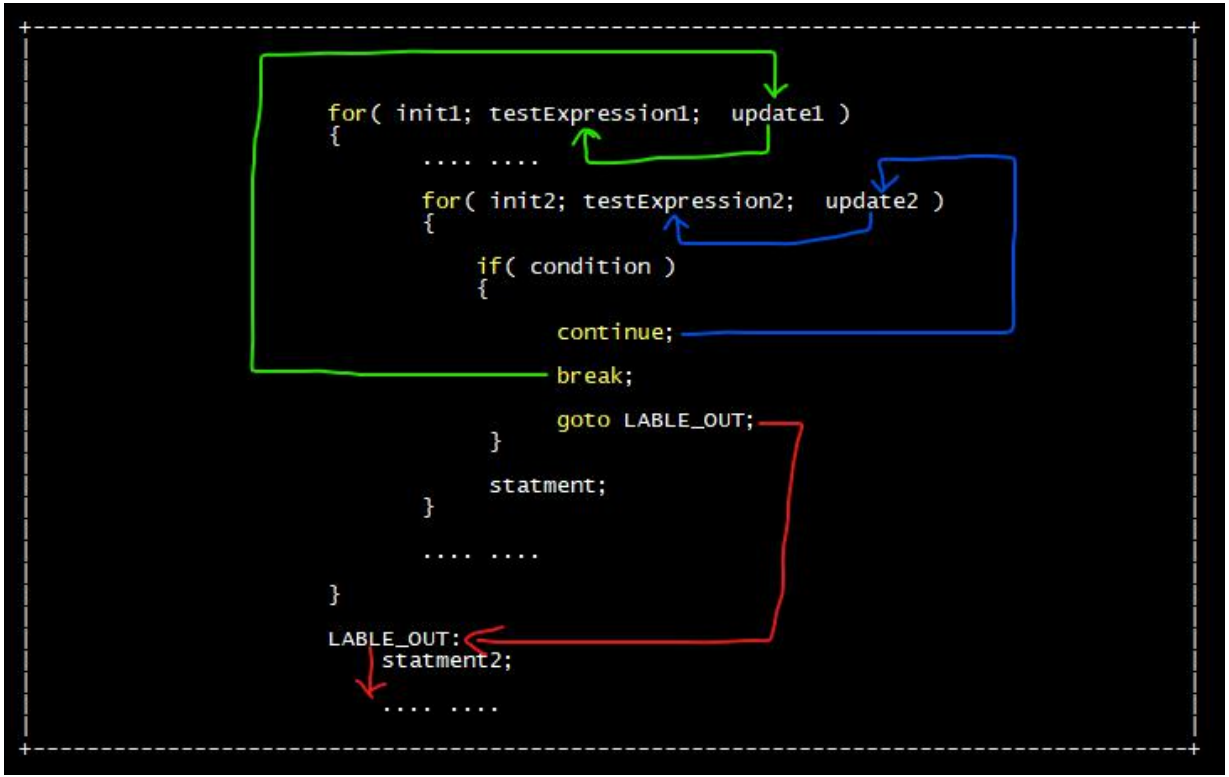
break语句

Speech/Training Topics



goto语句

Speech/Training Topics



07

函数与源码管理

函数

Speech/Training Topics

main函数作为C程序的入口函数，其他的函数在main函数调用后才会执行，例如之前使用的printf()函数等都是C语言提供的库函数，在开发中除了使用C语言提供的库函数外，还需要针对业务功能开发对应的函数。

函数是C程序的基本模块，用于完成特定任务的代码单元。封装函数就能实现代码的重复利用，让程序更加模块化，结构清晰，从而增强程序的可读性和降低后期维护成本。函数应该先定义，再使用；如果函数调用发生在函数定义之前，那应该先事先声明函数。

函数定义：

```
返回值类型  函数名(形参1，形参2, ...，形参N)
{
    变量定义部分;

    执行语句部分;

    函数返回语句;
}
```

函数声明：

```
返回值类型  函数名(形参1，形参2, ...，实参N);
```

函数调用:

```
函数名(实参1，实参2，  ...，实参N);
```

函数使用

Speech/Training Topics

```
76  void turn_led(int which, int status)
77  {
78      GPIO_PinState        level;
79
80      if( which >= LedMax )
81          return ;
82
83      level = status==OFF ? GPIO_PIN_SET : GPIO_PIN_RESET;
84
85      HAL_GPIO_WritePin(leds[which].group, leds[which].pin, level);
86  }
87
88  void blink_led(int which, uint32_t interval)
89  {
90      turn_led(which, ON);
91      HAL_Delay(interval);
92
93      turn_led(which, OFF);
94      HAL_Delay(interval);
95  }
```

函数

Speech/Training Topics

main函数作为C程序的入口函数，其他的函数在main函数调用后才会执行，例如之前使用的printf()函数等都是C语言提供的库函数，在开发中除了使用C语言提供的库函数外，还需要针对业务功能开发对应的函数。

函数是C程序的基本模块，用于完成特定任务的代码单元。封装函数就能实现代码的重复利用，让程序更加模块化，结构清晰，从而增强程序的可读性和降低后期维护成本。函数应该先定义，再使用；如果函数调用发生在函数定义之前，那应该先事先声明函数。

函数定义：

```
返回值类型  函数名(形参1，形参2, ...，形参N)
{
    变量定义部分;

    执行语句部分;

    函数返回语句;
}
```

函数声明：

```
返回值类型  函数名(形参1，形参2, ...，实参N);
```

函数调用:

```
函数名(实参1，实参2，  ...，实参N);
```

08

预处理与宏

源文件管理

Speech/Training Topics

09

指针入门

10

深入理解指针

11

C程序内存布局

12

再谈结构体

位字段

Speech/Training Topics

有些信息在存储时，并不需要占用一个完整的字节， 而只需占用1个或几个位(bit)。例如在存放一个开关量时，只有0和1 两种状态， 用一位二进位即可。为了节省存储空间，并使处理简便， C语言又提供了一种数据结构，称为“位域”或“位段”。所谓“位域”是把一个字节中的二进位划分为几个不同的区域，并说明每个区域的位数。每个域有一个域名，允许在程序中按域名进行操作，这样就可以把几个不同的对象用一个字节的二进制位域来表示。位段成员必须声明为int、unsigned int或signed int类型（short char long），如果声明的总数超过了一个unsigned int大小，那么会使用下一个unsigned int的存储位置，这时不允许一个字段跨越两个unsigned int之间的边界。

8.4.1 GPIO port mode register (GPIOx_MODER) (x =A to E and H)

Address offset:0x00

Reset value:

- 0xABFF FFFF (for port A)
- 0xFFFF FEBF (for port B)
- 0xFFFF FFFF for ports C..E
- 0x0000 000F (for port H)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODE15[1:0]		MODE14[1:0]		MODE13[1:0]		MODE12[1:0]		MODE11[1:0]		MODE10[1:0]		MODE9[1:0]		MODE8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODE7[1:0]		MODE6[1:0]		MODE5[1:0]		MODE4[1:0]		MODE3[1:0]		MODE2[1:0]		MODE1[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **MODE[15:0][1:0]:** Port x configuration I/O pin y (y = 15 to 0)
These bits are written by software to configure the I/O mode.
00: Input mode
01: General purpose output mode
10: Alternate function mode
11: Analog mode (reset state)

```
typedef struct gpio_modreg_s
{

    unsigned int  mode0:2;
    unsigned int  mode1:2;
    unsigned int  mode2:3;
    unsigned int  mode3:2;
    ... ... ... ... ...
    unsigned int  mode12:2;
    unsigned int  mode13:2;
    unsigned int  mode14:3;
    unsigned int  mode15:2;

} gpio_modreg_t;
```

结构体指针

Speech/Training Topics

有些信息在存储时，并不需要占用一个完整的字节， 而只需占用1个或几个位(bit)。例如在存放一个开关量时，只有0和1 两种状态， 用一位二进位即可。为了节省存储空间，并使处理简便， C语言又提供了一种数据结构，称为“位域”或“位段”。所谓“位域”是把一个字节中的二进位划分为几个不同的区域，并说明每个区域的位数。每个域有一个域名，允许在程序中按域名进行操作，这样就可以把几个不同的对象用一个字节的二进制位域来表示。位段成员必须声明为int、unsigned int或signed int类型（short char long），如果声明的总数超过了一个unsigned int大小，那么会使用下一个unsigned int的存储位置，这时不允许一个字段跨越两个unsigned int之间的边界。

8.4.1 GPIO port mode register (GPIOx_MODER) (x =A to E and H)

Address offset:0x00

Reset value:

- 0xABFF FFFF (for port A)
- 0xFFFF FEBF (for port B)
- 0xFFFF FFFF for ports C..E
- 0x0000 000F (for port H)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODE15[1:0]		MODE14[1:0]		MODE13[1:0]		MODE12[1:0]		MODE11[1:0]		MODE10[1:0]		MODE9[1:0]		MODE8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODE7[1:0]		MODE6[1:0]		MODE5[1:0]		MODE4[1:0]		MODE3[1:0]		MODE2[1:0]		MODE1[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **MODE[15:0][1:0]:** Port x configuration I/O pin y (y = 15 to 0)
These bits are written by software to configure the I/O mode.
00: Input mode
01: General purpose output mode
10: Alternate function mode
11: Analog mode (reset state)

```
typedef struct gpio_modreg_s
{

    unsigned int  mode0:2;
    unsigned int  mode1:2;
    unsigned int  mode2:3;
    unsigned int  mode3:2;
    ... ... ... ... ...
    unsigned int  mode12:2;
    unsigned int  mode13:2;
    unsigned int  mode14:3;
    unsigned int  mode15:2;

} gpio_modreg_t;
```

结构体对齐

Speech/Training Topics

有些信息在存储时，并不需要占用一个完整的字节， 而只需占用1个或几个位(bit)。例如在存放一个开关量时，只有0和1 两种状态， 用一位二进位即可。为了节省存储空间，并使处理简便， C语言又提供了一种数据结构，称为“位域”或“位段”。所谓“位域”是把一个字节中的二进位划分为几个不同的区域，并说明每个区域的位数。每个域有一个域名，允许在程序中按域名进行操作，这样就可以把几个不同的对象用一个字节的二进制位域来表示。位段成员必须声明为int、unsigned int或signed int类型（short char long），如果声明的总数超过了一个unsigned int大小，那么会使用下一个unsigned int的存储位置，这时不允许一个字段跨越两个unsigned int之间的边界。

13

链表

14

栈

15

队列