作者：X Y K

4.C语言编译常见错误举例

1.预处理错误 -i

`#include "name"` 从当前目录找name （也包括环境变量）（自定义）

`#include <name>` 从环境变量找name （系统库）

常见错误 not find

解决方案 gcc -I 跟查找文件的目录

或者 `#include "./path/name"`

2.编译错误 -c

语法错误 ; { }

3.链接错误 -o

原材料不够，

undefined reference to 'name '

寻找标签是否实现，链接时加入

or 多了

multiple definition of 'name'

多次实现了标签，只保留一个标签实现

5.C语言预处理介绍

`#include` 包含头文件

`#define` 宏 （替换） 不进行语法检查

`#define 宏名 宏体` 注意加括号(保证安全)

```
#define ABC (5+3)

#define ABC(x) (5+(x))
```

`#ifdef #else #endif` 条件编译

预定义宏（系统定义，方便调试开发）

`__FUNCTION__` :函数名

`__LINE__` : 行号（定位在哪一行执行该代码）

`__FILE__` : 文件名

```
printf("the %s, %s, %d \n",__FUNCTION__,__LINE__,__FILE__);
```

6.条件预处理举例

1.调试版本

```
#ifdef TEST

  printf("-------%s-------\n",__FILE__);

#endif
```

2.发行版本

`gcc -D :`

`gcc -DTEST === #define TEST`

从而控制debug信息的输出

7.宏展开下的#、##

`# 字符串化`

`## 连接符号`

```
#define ABC(x) #x                    #define ABC(x)   day##x

printf(ABC(ab\n));  // ab  printf("the day is %d",ABC(1))   //day1
```

2-2 C语言常用关键字及运算符操作

1.关键字

When　How　Why

关键字 ——>编译器预先定义一定意义的字符串

32个关键字　`sizeof` -->关键字

`sizeof()` 是编译器给我们查看内存空间容量一个工具，可以在任何环境中实现，是关键字（printf需要一定环境）

1.数据类型

C 操作对象： 资源/内存（内存类型的资源，LCD缓存，IO设备）

资源属性【大小】---->由编译器决定的（例如char 1个字节，int 4个字节，long 4个或者8个字节，short 2个字节）

限制内存，关键字　int a;　 sizeof(a) = 4;

char

硬件芯片操作的最小单位：bit 1 0 （位）

软件操作最小单位： char a;　（一组bit） 1B = 8bit

8 bit == 256

char a =300; // a溢出

int

大小：根据编译器来决定

编译器最优的处理大小：

系统一个周期，所能接受最大处理单位，

32bit　　4B　　int

16bit (max 65535)　　2B　　int

整型常量

```
char a = 300;   //300L
```

```
int a = 66535;  //对于2B系统，编译能过，但结果不可控
```

浮点

大小： float 4B double 8B

浮点型常量： 1.0 double　（可用1.0f , 强制使用float型）

2.自定义数据类型

C编译器默认定义的内存分配不符合实际资源形式。 自定义 = 基本元素的集合

struct

元素之间的和

```
struct myabc{
```

```
unsigned int a;

unsigned int b;

unsigned int c;}

int struct myabc mybuf
```

---------(变量声明的顺序有要求)

union

共用起始地址的一段内存 （技巧型代码）

```
union myabc{

char a;

int b;}

  union myabc abc;
```

enum

被命名的整型常数集合

```
#define MON 0

#define TUE 1

#define WED 2

enum abc{MOD ,TUE , WED};

enum 枚举名称{常量列表};
```

typedef

数据类型的别名

3.逻辑结构

分支，循环

`if、else、switch、case、default、do、while、for、continue、break、goto`

4.类型修饰符

对内存资源存放位置的限定

资源属性中位置的限定

`auto`

`auto char a` -> 可读可写

`{`

`auto char a ;}` -> a放在栈中

默认情况----->分配的内存可读可写的区域

区域如果在`{}`，栈空间

`register`

限制变量定义在寄存器上的修饰符，定义一些快速访问的变量（编译器会尽量安排CPU的寄存器取去存放这个变量，如果寄存器不足，变量还是存放在存储器中），&这个符号对register不起作用

内存（存储器） 寄存器

`static`

应用场景：修饰3种数据：

1. 函数内部的变量
2. 函数外部的变量
3. 函数

extern

const

常量的定义，只读的变量

volatile

告知编译器编译方法：不优化编译

修饰变量的值的修改，不仅仅可以通过软件，也可以通过其他方式

5.常用运算符

算术操作运算

+ - / % *

逻辑运算

|| && > >= < <=

`A || B != B||A`

! ?:

真 假

返回结果就是 1 0

位运算

<< >>

A & 0 ------> 0

& 屏蔽

int a = 0x1234;

a & 0xff00; 屏蔽低八位，取出高八位

A & 1 ------> A

&取出

| 设置，保留(设置为高电平)

A | 0 =====A

A | 1 =====1

对bit5置1：

`a = (a | (0x1 << 5));` ----> a |(0x1 << n)

对bit5清0：

`a = a & ~(0x1 << 5);` -----> a = a & (~(0x1 << n)) ;

^、~

算法

赋值运算符

a | (0x1 << 5)　　~a

内存访问符号

`()` 限制符、函数访问

`[]` 数组，内存访问的ID符号

`{}` 函数体的限制符

`->　.` 对自定义空间不同成员的访问

`&`取地址　`*`取内容

2-3 C语言内存空间的使用

指针

指针概述

内存类型资源 地址、门牌号的代名词　`* &`

指针（就是一个地址）

指针变量：存放指针这个概念的盒子

C语言编译器对指针这个特殊概念：有俩个疑问

1. 分配一个盒子，盒子要多大（在**32bit**系统中，指针就**4**个字节）

2. 盒子里存放的地址、所指向内存的读取方法是什么（`char *p;(告诉指针一次读一个字节)  int *p;(一次读4个字节)` ）

指针指向内存空间，一定要保证合法性

```
int a = 0x12345678
```

```
char *p1;
```

```
p1 = &a;
```

```
printf("the p1 is %x \n",*p1);
```

上述程序可以通过字节的方式去读int类型，此时p1返回78

指针+修饰符

`const` 常量、只读【不能变】（内存属性：1.内存操作大小。2.内存的变化性，可读可写）

`char const *p;` == `const char *p` 【recommend】

字符串（指向随意地址，地址内容不可变）

`char *const p` 【recommend】== `char *p const`

硬件资源 （指向固定地址，地址内容可以改变）

`const char *const p` 指向ROM（指向固定地址，地址内容不可变）

`""`字符串是整型常量,不可随意改

`char *p = "hello world! \n";` (const 类型) = `const char *p`(限制更加深)

```
*p ='a';
```

segmentation fault 报错

执行正确如下：

```
char buff[] =  {"hello , world !\n"};

char *p2 = buff;

*p2 ='a';
```

```
voliatile
```

防止优化指向内存地址

```
volatile char *p

*p = 0x10
```

`while(*p == 0x10)` 如果编译器优化就是死循环。（假设*p指向键盘传来的数据）

typedef

什么类型 变量名称；

`char *name_t` `name_t`是一个指针，指向了一个char类型的内存

`typedef char *name_t` `name_t`是一个指针类型的名称，指向了一个char类型的内存

指针+运算符

++、--、+、-

```
int *p = xxx [0x12]
```

`p+1    [0x12 + 1*(sizeof(*p))]` p不变

指针的加法运算，实际上加的是一个单位，单位的大小可以使用sizeof(p[0])

`p++  p--` 更新地址 `p`

[]

变量名[n]

n:ID 标签

地址内容的标签访问方式,取出标签里的值

`p+n  == p[n]`

变量分配从高往低

```
int a = 0x12345678;

int b = 0x99999199;

int *p1 = &b;

char *p2 = (char *)&b;

printf("the p1+1 is %x,%x,%x \n",*(p1+1),p1[1],*p1+1); // 结果 12345678 12345678  9999919a

printf("the p2+1 is %x \n",p2[1]);  结果 ： 91
```

```
const int a = 0x12345678;

int b = 0x11111111;

int *p = &b;

p[1]=0x100;
```

此时a变为0x100(越界修改)

逻辑操作符

..... == !=

1.跟一个特殊值比较

0x0:地址的无效值，结束标志

NULL

2.指针必须是同类型比较才有意义

多级指针

本质还是盒子，盒子里内容可能又是地址(存放地址的容器)

多级指针能让之前非线性排列（字符串）的结构成为线性

int **p;

char **p;

argc (参数个数) **argv（参数内容）

当 argv[i]==NULL说明字符串读完 --->结束

数组

数组空间

空间赋值

按照标签逐一处理

`int a[10] ={空间};` 第一次赋值

c语言本身，一般不支持空间和空间的拷贝

数组空间的初始化和变量的初始化本质不同

对于第二次内存初始化赋值，只能逐一处理。

一块空降，当成字符空间，提供一套字符拷贝函数（原则：内存空间和内存空间的逐一赋值的功能的一个封装体，一旦空间中出现了0这个特殊值，函数即将结束）

`strcpy()` 容易内存泄漏,因为停止条件就是读到“0”

字符空间及地址

字符空间： ASCII码编码来解码的空间 --->给人看（%s \0作为结束标志）

非字符空间：数据采集 0x00 0xff (8bit) 开辟一个存储这些数据盒子

`char ---> string`

`unsigned char ---> data` (推荐逐一拷贝，结束只能通过定义拷贝个数实现)

拷贝三要素

1. src
2. dest
3. num

```
int buf[10];

int sensor_buf[100];

memcpy(buf,sensor_buf,10*sizeof(int));

unsigned char buf1[10];

unsigned char sensor_buf1[100];

memcpy(buf,sensor_buf,10*sizeof(unsigned char));
```

指针与数组

`char *a[100];//* 告诉 a[100] 存放的属性`

`sizeof(a) = 100 * 4;` (* 占4个字节，而char 仅仅代表读取方式)

`char **a;` //跟多级指针一样

多维数组

定义一个指针，指向 `int a[10]`的首地址

`int *p1 = a;`

定义一个指针，指向 `int b[5][6]` 的首地址

`int *p[5]` p有五个空间，每个空间都有* （从右往左翻译）

`int (*p)[5]` p是一个地址，然后地址读取方式是5个int读

所以 `int (*p2)[6] = b;`

如果 `int b[2][3][4];`

`int (*p)[3][4];`就可以指向上面的b

结构体、共用体

定义、字节对齐

字节对齐可以提高效率，牺牲一定空间换取时间的效率

最终结构体的大小一定是4的倍数，结构体里成员变量的顺序不一致，也会影响其大小

位域

内存分布图

内存的属性：

1. 大小
2. 在哪里

`int a;` 默认方式（栈）但前面加了`static`会放在全局数据段

编译 ---》汇编---》链接

内核空间 （应用程序不许访问）

栈空间 （局部变量） RW

运行时堆空间 （malloc）

全局的数据空间 （初始化的、未初始化的）static RW data _bss

只读数据段 ""（""代表常量空间） R TEXT

代码段 （code）R TEXT

0x0

`size`可以看到各段的内容(全局数据空间)

`strings` （只读数据段）

`nm` 看静态代码段， 查看可执行程序的标签

栈空间

运行时，函数内部使用的变量，函数一旦返回就释放，生存周期是函数内。

堆空间

运行时，可以自我管理的分配和释放的空间，生存周期是程序员决定

分配：

`malloc()` 一旦成功，返回分配好的地址给我们，只需接受，对于新地址读法，由程序员灵活把握，输入参数指定分配大小，单位就是B

```
char *p

p = (char *)malloc()

if(p==NULL){ error;}
```

释放：

```
free(p);
```

只读空间

静态空间、整个程序结束时释放内存、生存周期最长

段错误分析

2-4 C语言函数的使用

函数概述

一堆代码的集合，用一个标签去描述它（复用化）

标签----函数名

函数具备3要素：

1. 函数名 （地址）
2. 输入参数
3. 返回值

在定义函数时，必须将3要素告知编译器

如何用指针保存函数？

```
int fun(int,int,char)
```

```
int (*p)(int,int,char);
```

定义函数，调用函数

输入参数

承上启下的功能

调用者：

函数名（要传递的数据） //实参

被调者：

函数的具体实现

函数的返回值，函数名（接受的数据） //形参

{

xxxxx

}

实参传递给形参

传递到形式：拷贝（1.逐位拷贝）

```
void myswap(int buf) //预留4个字节接受
```

值传递

上层调用者保护自己空间值不被修改的能力

地址传递

上层调用者让下层子函数修改自己空间值的方式

连续空间的传递(地址传递)

1、数组

数组名 -- 标签

```
int abc[10];
```

```
fun(abc)
```

形参：

```
void fun(int *p)  === void fun(int p[10])
```

2、结构体

结构体变量

`struct abc{int a;int b;int c};`

`struct abc buf;`

实参

`fun(buf);` `fun(&buf)`

形参

`void fun(struct abc a1)` `void fun(struct abc *a2)`

连续空间只读性

`const char *p` 只读空间

`char *p` 该空间可能修改

`void fun(const char *p)` ==>避免出现 `p[n] = '1'`

字符空间（`char`）和非字符空间区别：结束的标志

空间2要素：1.空间首地址 2.结束标志

结束标志：内存里面存放了0x00(1b),字符空间

非字符空间 0x00，不能当作结束标志

字符空间最小模板：

`void fun(char *p)`

```
{

int i = 0;

while(p[i] == 0){ //或者其他条件

p[i]操作  p[i]=x;a=p[i]

i++;

}

}

""---->初始化 const char *

char buf[10] --->初始化 char *
```

非字符空间

unsigned char * (非字符)

short *

struct *

int *

读内存的方法不是一个字节一个字节读

这些空间0不能作为结束标志

```
1.需要加长度，**限制空间大小**

2.  void  * (代表任意指针，数据空间（非字符空间）的标志符)（内存）
```

```
void *
```

1.修改　int * short* long*　　（是对某一个值修改 例如：单一变量）

2.空间传递

　2.1 子函数看看空间里的情况　const*

　2.2 子函数反向修改上层空间内容 char* void*

注意！　void* 是对某一个内存的修改（比如数组、结构体等）

返回值

提供启下功能的一种表现形式

基本语法

返回类型 函数名称 (输入列表)

{

　return

}

调用者：

```
fun()
```

被调用者：

```
int fun()
{
  return num;
}
```

拷贝

返回类型

基本数据（变量）、空间（指针）

返回连续空间类型

指针作为空间返回的唯一数据类型

`int *fun()` 地址：指针的合法性

作为函数的设计者，必须保证函数返回的地址所指向的空间是合法（不是局部变量(比如char)）

使用者： `int *p =fun();`

函数内部实现

1.静态区 static

2.只读区 （字符串常量）

3.堆区 （(char *)malloc ,free）

基本数据类型 fun(void)

{

基本数据类型 ret;

xxxxx

ret = xxx;

return ret;

}

2-5常见面试题目

嵌入式0x10道题目

对于宏定义

1.括号 2.数值大小定义(L)

数据声明

用变量a给出下面的定义
a)一个整型数（An integer)

```
int a;
```

b)一个指向整型数的指针（A pointer to an integer)

```
int *a;
```

C)一个指向指针的的指针，它指向的指针是指向一个整型数（A pointer to a pointer toan integer)

```
int * *a;
```

d)一个有10个整型数的数组（An array of 10 integers)

```
int a[10];
```

e)一个有10个指针的数组，该指针是指向一个整型教的。(An array of 10 pointers tointegers)

```
int *a[10];
```

f)一个指向有10个整型数数组的指针（A pointer to an array of 10 integers)

```
int [10] *a;
```

```
int (*a)[10];
```

g)一个指向函数的指针，该函数有一个整型参数并返回一个整型数(A pointer to afunction that takes an integer as an argument and returns an integer)

```
int (*a)( int );
```

以指针`*a`为中心，其右边说面读写方式

h)一个有10个指针的数组，该指针指向一个函数，该函数有一个整型参数并返回一个整型数(An array of ten pointers to functions that take an integer argument and return aninteger)

```
int  (*a[10]) (int);
```

修饰符使用总结

关键字static的作用是什么？

1、修饰局部变量

默认局部变量，在栈空间存在，生存期比较段

局部静态化，局部变量在静态数据段保存，生存期比较长

2、修饰全局变量

防止重命名，限制变量名只在本文件内起作用

3、修饰全局函数

防止重命名，限制该函数只在本文件内起作用

关键字const有什么含义？

C：只读，建议性，不具备强制性 ！=常量（可以通过数组越界去修改）

C++:常量

关键字volatile有什么含义？并给出三个不同的例子

防止C语言编译器优化。

1.修饰的变量，该变量的修改可能通过第三方来修改

位操作

取固定内存，比如0x67a9

```
int *p =(int *)0x67a9;
```

```
*((int *)0x67a9) = 0x1111;
```

```
((void (*)(void))0x67a9)();
```