控制组STM32培训1

—— C语言基础

Struct 结构体

```
Struct person {
  int height;
  int weight;
};

struct person a = {100, 100};
a.height = 100;
```

宏定义 define & typedef

define

```
# define HEIGHT 100

int main(void) {
  int a = HEIGHT;
  return 0;
}
```

BUT

```
#include <stdio.h>
#define ADD(a,b) a+b
int main(void) {
  printf("%d\n", ADD(1, 2));
  printf("%d\n", 2 * ADD(1, 2));
}
```

结果如何?

typedef

```
typedef unsigned char u8;

int main(void) {
  u8 a = 1;
  return 0;
}
```

函数指针

```
typedef char (*PTRFUN)(int);
char fun(int a){ return 'a';}
void main(void)   {
  PTRFUN pFun = fun;
  (*pFun)(2);
}
```

Struct 结构体

```
typedef struct person {
  int height;
  int weight;
}Person, *PersonP;

Person a;
a.height = 100;

PersonP b = &a;
b->height = 100;
```

头文件 header

头文件是什么、为什么要有头文件?

```c
// main.c

int add (int a, int b) {
  /* ... */
}
int minus (int a, int b) {
  /* ... */
}
int pow (int a, int b) {
  /* ... */
}

int main(void) {
  add(1, 2);
  minus(1, 2);
  pow(1, 2);
  return 0;
}
```

```c
// my_add.h
#ifndef _MY_ADD_H_
#define _MY_ADD_H_
int add (int a, int b);
int minus (int a, int b);
#endif // _MY_ADD_H_
```

```c
// my_add.c
#include "my_add.h"
int add (int a, int b) {
  /* ... */
}
int minus (int a, int b) {
  /* ... */
}
```

```c
// main.c
#include "my_add.h"
int main(void) {
  add(1, 2);
  minus(1, 2);
  return 0;
}
```

什么应该放在头文件?

NOTICE

需要暴露给其它源文件使用的东西才放到头文件里

```
#ifndef _MY_ADD_H_
#define _MY_ADD_H_

int add (int a, int b); // 函数声明
#define HEIGHT 100  // 宏定义
typedef unsigned char u8;
/* ! 不要将函数体放在头文件内 */

#endif // _MY_ADD_H_
```

编译 & 链接

预处理

```
gcc -E main.c -o main.i
```

头文件内容拷贝进文本中，做预处理操作(#define, #if-else-endif ...)

汇编

```
gcc -S main.i -o main.s
```

将预处理好的源文件汇编生成汇编代码

编译

```
gcc -c main.s -o main.o
```

多文件编译

```
gcc main.c my_add.c -o main
```

为什么不要将函数体放在头文件内?

C 指针 pointer

为什么要使用指针?

传值引用 & 传参引用

```
void edit(int a) {
  a = 1;
}
int main(void) {
  int a = 0;
  edit(a);
  printf("%d\n", a);
}
```

输出的值是?

```c
void edit(int* a) {
  *a = 1;
}
int main(void) {
  int a = 0;
  edit(&a);
  printf("%d\n", a);
}
```

```c
typedef struct person {
  int height;
  int num[1000];
}*PersonP, Person;
void edit(PersonP p) {
  p->height = 100;
}
void print(PersonP p) {
  /* 打印 p->num 的内容 */
  /* ... */
}
void main(void) {
  Person p;
  edit(&p);
}
```

* 和 & 和 ->

- `*`: 在声明时表示指针，在变量前表示间访

```
int a;
int *p = &a;
*p = 1;
```

- `&`: 取地址
- `->`: struct 中使用，`p->a = 1` 相当于 `(*p).a = 1`

3. 有如下代码

```
typedef struct sample_struct {
    struct sample_struct* next;
} *Sample;
```

其中，`Sample` 与 `next` 变量的类型分别是（　　）

A. 指向`sample_struct`的结构体指针，指向`sample_sruct`的结构体指针

B. `sample_struct`结构体，指向`sample_struct`的结构体指针

C. 指向`sample_struct`的结构体指针，`sample_struct`结构体

D. `sample_struct`结构体，`sample_struct`结构体

4. 对题3中的代码，若想对`sample_stuct`中的`next`变量赋值，代码缺省处填写内容正确的是（　　）

```
Sample sample1;
struct sample_struct sample2;
_____________________________
```

① `sample1->next = sample2;`

② `sample1.next = &sample2;`

③ `sample1->next = &sample2;`

④ `sample1->next = *sample2;`

⑤ `*(sample1).next = &sample2;`

⑥ `*(sample1)->next = sample2;`

A. ①③　　B. ②⑤　　C. ③⑤　　D. ②⑥

嵌入式中的C

多用u8、u16、u32代替unsigned char, u. short, u.int

嵌入式中位运算使用较多，内存较小，使用这样的typedef能够更直观显示变量大小，内存占用。

注意编译器优化的坑

```
/* PORTA.Bits0为PA0 IO口 */
PORTA.Bits0 = 0;
while (PORTA.Bits0 != 0) {
  /* do something */
}
```

解决办法：

- 降低MDK编译优化等级
- 变量声明前加`volatile`：`volate u8 a = 0;`

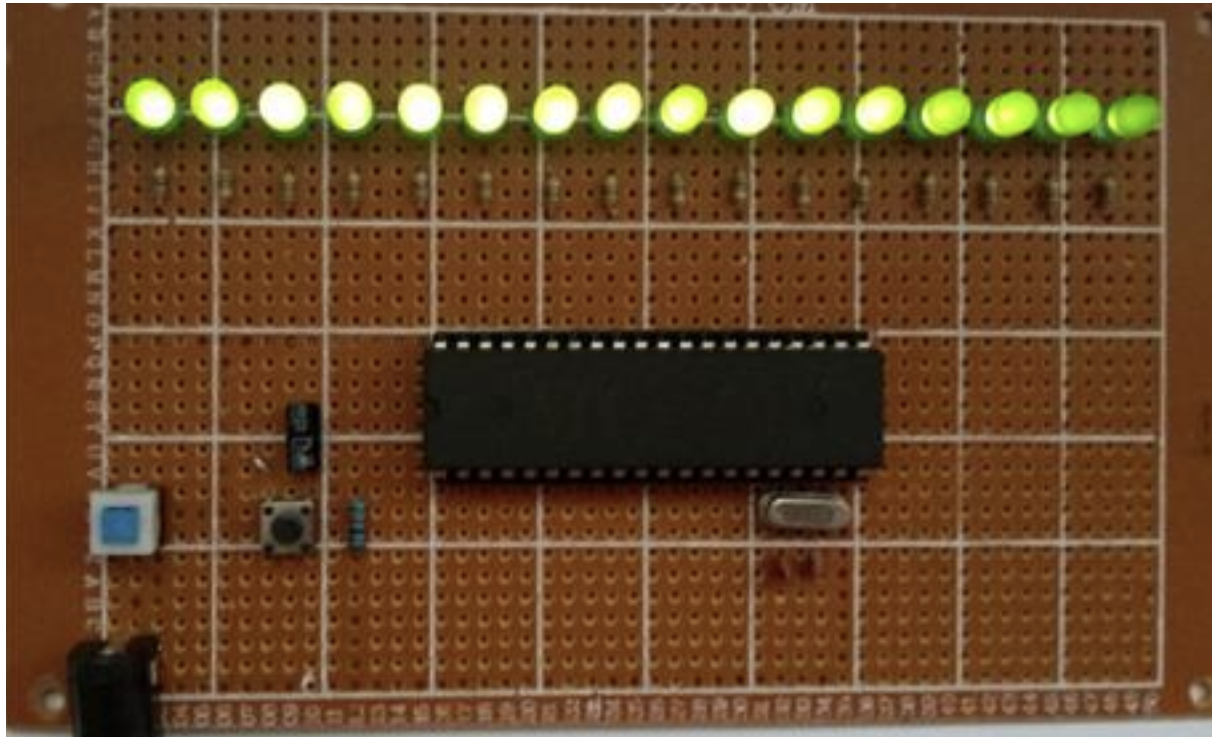
C语言与电路时延

```
/* PORTA.Bits0为PA0 IO口 */
PORTA.Bits0 = 0;
if (PORTA.Bits0 == 0) {
  /* do something */
}
```

位运算与寄存器

重要吗?

Example: 跑马灯实验



库函数方式

```
/* 初始化GPIO */
/* ... */
while (1) {
  int last_i = 2;
  for (int i = 0; i < 8; i++) {
    /* IO置低电平 */
    GPIO_ResetBits(last_i);
    /* IO置高电平 */
    GPIO_SetBits(i);
    last_i = i;
  }
}
```

寄存器位运算方式

```
/* 初始化GPIO */
/* ... */
while (1) {
  u8 temp = PORTA << 7;
  PORTA = PORTA >> 1;
  PORTA = PORTA | temp;
}
```

代码规范

规范的代码非常重要！

- 缩进： 2字符或4字符，不要混用
- 空格：运算符间加空格使变量更直观

```
c = add(num_a, num_b, num_c); // c=add(num_a,num_b,num_c)
if (a == 1 && b != 2); // if(a==1&&b==2)
int add(int a, int b);
```

- 注释规范： 行间注释用`/**/`，行内注释用`//`，注释内容与注释符号有空格，内容清晰简洁

```
/*
 hello world!
 */
/* hello world */
int height; // hight of a person
```

- 变量命名规范：驼峰 or 下划线， 不要用拼音，不要混用驼峰与下划线，变量首字母一般小写，类/结构体名/宏定义一般大写

```
int person_a;
int personB;
typedef struct p {int a;}Person;
#define HEIGHT 100
```

```
int dianji_1;
int chuanganqiB;
```

- 文件名规范：使用多文件，将不同类型函数区分开，头文件与.c文件命名一致，源文件过多时分文件夹放

... ...

多思考怎样写代码结构更清晰、更易读