C语言加强

变量

变量，可以变，可读可写，都存在内存中，指针（地址）在32位处理器中都是4字节

只读的常量放在flash中，加const放于ram中，节省内存

sizeof和关键字

- sizeof

  `Char *p` `Int *p2` `Sizeof(*p)  = sizeof(int)` `Sizeof(p) = 32`

- Volatile

  防止编译器优化（只在cpu中改变），必须要把值写入内存中用于必须要读的寄存器

- Extern

  可以在文件中直接写，也可以在包含的头文件里面写，Extern建议放在.h文件中声明

  不建议使用全局变量，要的话用static，可以做一个函数接口 get_x()让别人获得变量的值

- vTypedef （类比 #define,define是宏定义，直接替换），typedef是类型定义 可以创建类型别名，可以让使用更方便。

```
typedef int A;
typedef int * B;
typedef struct lcd_operation C;
typedef struct lcd_operation * D;
typedef int (*add_type)(int, int);
```

A、B、C、D都是类型别名，add_type同理

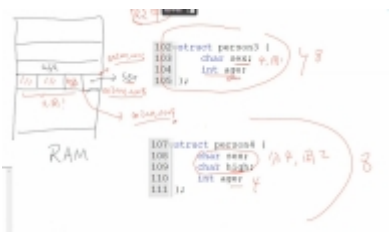
```
add_type f1;
add_type f2;
```

结构体

struct类型声明不占空间，声明变量占空间

Struct 里面定义的变量类型和非4的倍数，在内存中会优化到占4个字节

对齐效率高



变量赋值

变量的类型是多大空间，一次赋值就会赋值多大空间

通过指针赋值

```
Struct person {
    char a;
    char b
    int c;
}
struct person wei = {“1”, ”2”, 1};
struct person *pt;
*pt = wei // 把wei的值全部赋给\*pt
            //*pt 也是一个person结构体类型；(故并非传地址)
            //pt是指针类型
```

结构体指针

结构体声明中，不可以放自己的结构体，但可以放自己的结构体指针，结构体用.取成员，指针用 ->取成员

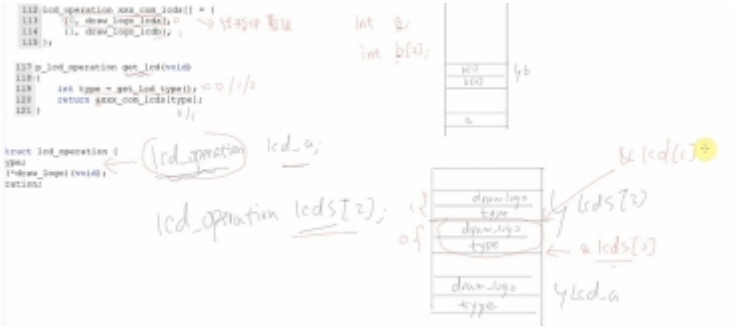
Int (*function)(void)

{

}

定义函数指针，function是变量，占4字节，function和&function都一样 。

get_lcb 取出结构体指针进行传递 4字节，避免了传递结构体，省空间



链表

- 链表的实质就是指针，可以存放下一个元素的地址,最原始的链表就是只存下一个的地址，有实际意义的链表成员里面会有别的信息

```
struct list{
    char *name;
    struct person *next;
}

struct person{
    char name;
    char age;
    struct person *next;
}p1
void InitList(struct list *pList, char *name)
{
```

```
    pList->name = name;
    pList->next = NULL;
}
void AddItemTolist(struct list *pList,struct person *new_persion)
{
    struct person *last
    if(pList->next = NULL)
    {
        last->next= new_persion;
        new_persion->next= NULL;
        return;
    }
    last = pList->next;
    while(last!= NULL)
    {
        last =a_list->next;
    }
    last->next= new_persion;
    new_persion->next= NULL;
}
void DelItemFromList(struct list *pList, struct person *person)  //链表的删除
{
    struct person *pre = NULL;
    struct person *p = pList->next;
    /* 找到person */
    while (p != NULL && p!=person)
    {
        pre = p;
        p=p->next;
    }
    /*退出条件 p== NULL,p==person*/
    if(p == NULL)
    {
        printf("cannot find.\r\n");
        return;
    }
    if(pre == NULL)
    {
        pList->next = p->next;
    }
    else
    {
        pre->next= p->next;
    }


}
int main()
{
    struct list a_list;
    int i;
    InitList(&a_list, "A_class"); //链表的创建
    i = 0;
    while(p[i].name != NULL)
    {
        AddItemTolist(&a_list,&p1); //添加链表
        i++;
    }
```

```
    PrintList(a_list);
}

/****************修改*****************/
struct list{
    char *name;
    struct person head;
}

struct person{
    char name;
    char age;
    struct person *next;
}p1
void InitList(struct list *pList, char *name)
{
    pList->name = name;
    pList->head->next = NULL;
}
void AddItemTolist(struct list *pList,struct person *new_persion)
{
    struct person *last= &pList->head;
    while(last->next!= NULL)
    {
        last =a_list->next;
    }
    last->next= new_persion;
    new_persion->next= NULL;
}
void DelItemFromList(struct list *pList, struct person *person)  //链表的删除
{
    struct person *pre = pList->head;
    /* 找到person */
    while (pre!= NULL && p->next!=person)
    {
        pre=p->next;
    }
    /*退出条件 p== NULL,p==person*/
    if(pre == NULL)
    {
        printf("connot find .\r\n");
        return
    }
    else
        pre->next= person-> next;
}
int main()
{
    struct list a_list;
    int i;
    InitList(&a_list, "A_class"); //链表的创建
    i = 0;
    while(p[i].name != NULL)
    {
        AddItemTolist(&a_list,&p1); //添加链表
        i++;
    }
```

```
    PrintList(a_list);
}
```

普通链表的缺点：不同的链表，结构体指针类型不同，得重新写一份

改进： 定义指针成员为struct node *next，指向链表下一个成员结构体中的node结构体指针（**统一性高**）

架构与汇编简明

全局变量的初始化和空间分配

编译后的程序下载到flash上，程序开始之后，把有初始值全局变量统一复制到ram上变量所处于的地址上，静态变量和全局变量一样被复制过去，只是别的函数或文件无法访问无初始值的被统一初始化为0，

初始值为0/无初始值的全局变量或静态变量 被类似`memset`(将指针变量 s 所指向的前 n 字节的内存单元用一个“整数” c 替换)统一置0

栈和堆

栈是c分配的空闲内存

- 向下增长
- 估计栈大小：寻找使用局部变量最多的调用链
- 选出空闲空间

堆是栈之外程序员自己或者他人分配的空闲空间，可以自己控制（栈无法