知识点1【强制类型对齐】（了解）0-1

指定对齐原则：

1. 使用#pragma pack改变默认对其原则

格式：

#pragma pack (value)时的指定对齐值value。

注意：

1.value只能是：1 2 4 8等

2.指定对齐值与数据类型对齐值相比取较小值

如：如果指定对齐值：

设为1：则short、int、float等均为1

设为2：则char仍为1，short为2，int 变为2

步骤：

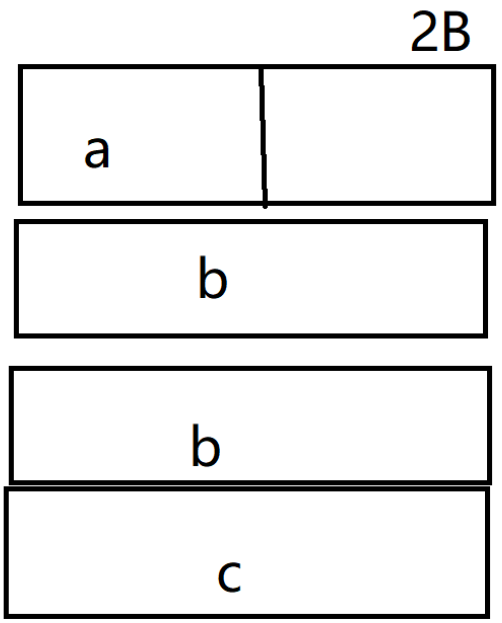
1、确定分单位:每一行应该分配的字节数,min(value,默认分配单位)。

2、成员偏移量 = 成员自身类型的整数(0~n)倍

3、收尾工作 = 分配单位的整数（0~n）倍。

案例1：

```
#include<stdio.h>
//指定对齐规则
#pragma pack(2)
typedef struct
{
 char a;
 int b;
 short c;
}DATA1;
void test01()
{
 printf("%d\n", sizeof(DATA1));//8
 return;
}
```



知识点2【位段 位域】

➢C语言允许在一个结构体中以位为单位来指定其成员所占内存长度，以位为单位的成员称为“位段”或称“位域”

```
struct  packed_data{
        unsigned int a:2;
        unsigned int b:6;
        unsigned int c:4;
        unsigned int d:4;
        unsigned int i;
} data;
```

其中a,b,c,d分别占2位，6位，4位，4位，i为整型，占4个字节


```
/位段 一般只考虑unsigned int类型 也可以考虑unsigned char
typedef struct
{
 unsigned char a:2;//a只占一个字节中的两位二进制位
 unsigned char b:2;//b只占一个字节中的两位二进制位

 //相邻位域 可以压缩 （压缩的位数 不能超过 成员自身大小）
 unsigned char c:5;

}DATA2;

void test02()
```

```
{
 printf("%d\n", sizeof(DATA2));//2
}
```

位段的使用：

```
/位段 一般只考虑unsigned int类型 也可以考虑unsigned char
typedef struct
{
 unsigned char a:2;//a只占一个字节中的两位二进制位
 unsigned char b:2;//b只占一个字节中的两位二进制位

 //相邻位域 可以压缩 （压缩的位数 不能超过 成员自身大小）
 unsigned char c:5;

}DATA2;

void test02()
{
 DATA2 data;

 printf("%d\n", sizeof(DATA2));//2

 //位段 不能取地址
 //printf("%p\n", &data.a);

 //位段的赋值 不要超过 位段的大小 a:2
 data.a = 6;//0110
 printf("%u\n", data.a);//2
}
```

无意义的位段：

```
typedef struct
{
 unsigned char a:2;//00
 unsigned char :4;//无意义的位段（占有4位）
 unsigned char b:2;//11
}DATA3;

void test03()
```

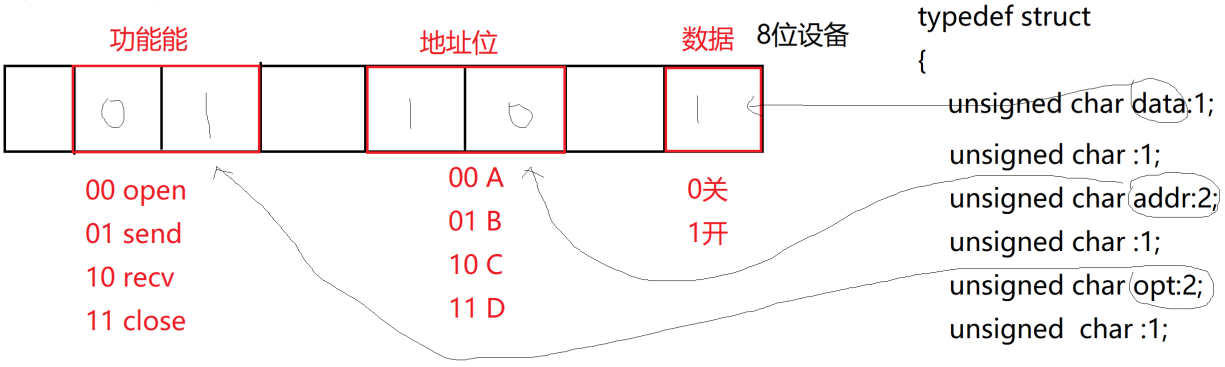
```
{
  DATA3 data;
  memset(&data, 0,1);

  data.a = 0;//00
  data.b = 3;//11

  printf("%d\n",sizeof(DATA3));

  printf("%#x\n", *(unsigned char *)&data);//1100 0000 //0xc0
}
```

说明：a是低位 b是高位

```
typedef struct
{
    unsigned char a:2;
    unsigned char :2;//无意义的位段（占有两位）
    unsigned char b:2;
}DATA3;
```

应用场景：0-2

单片机的寄存器8b



需求：给C设备 发送 1

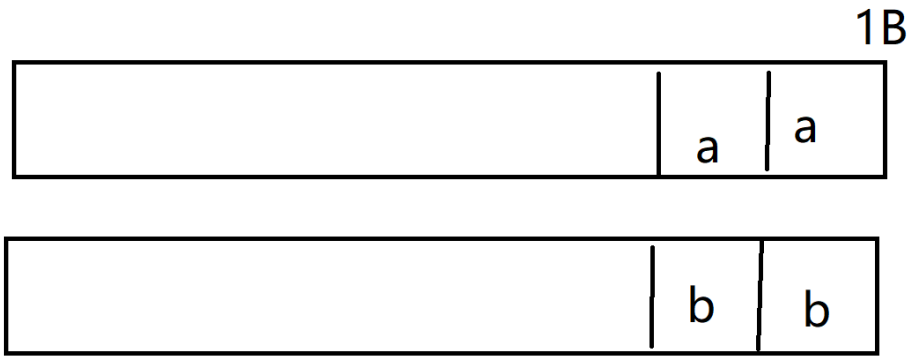
DATA msg;

```
msg.data = 1;
msg.addr = 2;
msg.opt = 1;
```

另起一个位段：(了解)

```
typedef struct
{
 unsigned char a:2;//00
 unsigned char :0;//另起一个位段
```

```
 unsigned char b:2;//11
}DATA4;
void test04()
{
 printf("%d\n", sizeof(DATA4));
 }
```

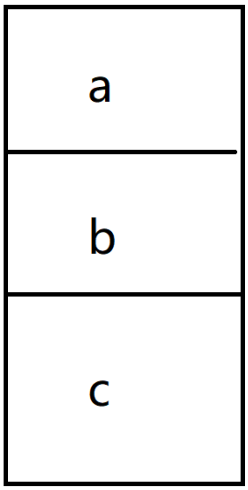


知识点3【结构体 与 共用体 的区别】

结构体：struct

所有的成员拥有独立的空间

```
struct stu
{
 char a;
 short b;
 int c;
};// a b c成员有用独立的空间
```



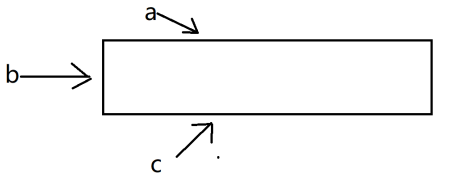
共用体（联合体）： union

所有的成员 共享 同一份空间。

```
union stu
{
 char a;
 short b;
 int c;
};// a b c成员共享同一份空间
```

union stu
{
  char a;
  short b;
  int c;
};// a b c成员共享同一份空间

共用体的空间大小 由最大的成员决定



知识点3【共用体】

共用体的空间用 最大的成员 决定

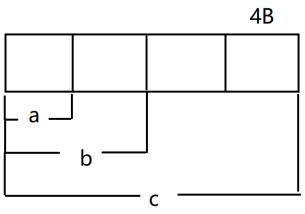
```
union data
{
 char a;
 short b;
 int c;
};
void test05()
{
```

```
 union data A;
  printf("%d\n",sizeof(union data));//4

  A.a = 10;
  A.b = 20;
  A.c = 30;
  //共用体是最后一次 赋值有效（不严谨）

  printf("%d\n", A.a+A.b+A.c);//90
 }
```

共用体 虽然 共有同一份空间 但是从空间读取的字节数 是有成员自身类型决定。

```
union stu
{
   char a;
   short b;
   int c;
};// a  b  c成员共享同一份空间
```

共用体的空间大小 由最大的成员决定

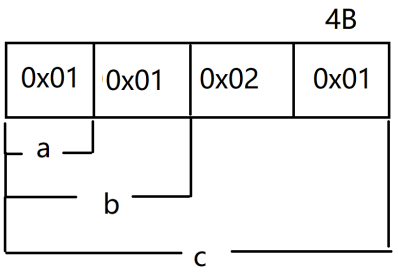


案例：

```
union data
{
 char a;
 short b;
 int c;
};
void test05()
{
 union data A;
  printf("%d\n",sizeof(union data));//4

  A.c = 0x01020304;
  A.b = 0x0102;
  A.a = 0x01;

  printf("%#x\n", A.a+A.b+A.c);//0x01020203
 }
```

```
union stu
{
    char a;
    short b;
    int c;
};// a  b  c成员共享同一份空间
```

共用体的空间大小 由最大的成员决定



A.a+A.b+A.c

```
A.c = 0x01020304;
A.b = 0x0102;
A.a = 0x01;
```

A.a:0x01 + A.b:0x0101 + A.c:0x01020101

A.c:0x01020101

A.b: 0x0101

A.c: 0x01

0x01020203

知识点4【枚举】(了解)1-1

➢枚举

将变量的值一一列举出来，变量的值只限于列举出来的值的范围内

➢枚举类型定义：

```
enum   枚举名
{
    枚举值表
};
```

在枚举值表中应列出所有可用值，也称为枚举元素

枚举变量仅能取枚举值所列元素

```
1  //枚举列表的值 默认是从0开始
2  enum POKER{ HONGTAO, HEITAO=30, MEIHUA=40, FANGKUAI };
3  void test06()
4  {
5   //poker_color 的取值为HONGTAO, HEITAO, MEIHUA, FANGKUAI中某一个
6   enum POKER poker_color = HEITAO;
7
8   printf("poker_color = %d\n", poker_color);//30
9   printf("HONGTAO = %d\n", HONGTAO);//0
10   printf("HEITAO = %d\n", HEITAO);//30
11   printf("MEIHUA = %d\n", MEIHUA);//40
12   printf("FANGKUAI = %d\n", FANGKUAI);//41
```

```
13
14 }
```

知识点5【链表】

数组的分类：便于 便来

静态数组：int arr[10] 数据过多造成 空间溢出 数据过小 空间浪费

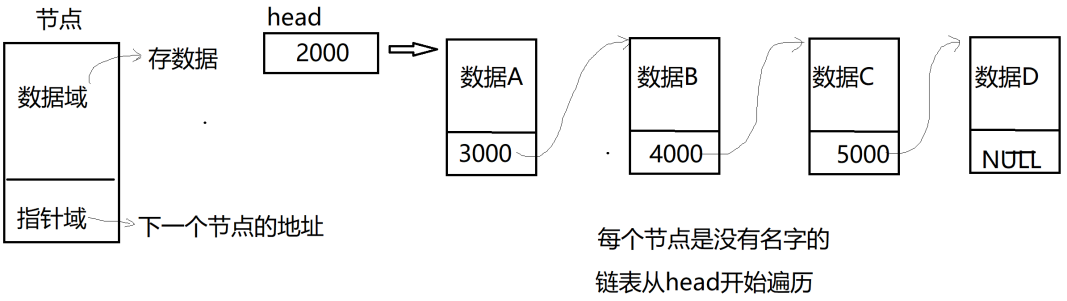
动态数组：malloc calloc realloc 合理利用空间 不能快捷的 插入或删除数据（会涉及到大量的数据移动）

链表是一种物理存储上非连续，数据元素的逻辑顺序通过链表中的指针链接次序，实 现的一种线性存储结构。

链表由一系列节点（链表中每一个元素称为节点）组成，节点在运行时动态生成（malloc），每个节点包括两个部分：

一个是存储数据元素的数据域

另一个是存储下一个节点地址的指针域



链表节点的定义：（结构体实现）

```
1  typedef struct stu
2  {
3   //数据域（自定义）
4   int num;
5   char name[32];
6   float score;
7
8   //指针域
9   struct stu *next;//保存下一个节点的地址
10  }STU;
```

```
11
```

知识点6【静态链表】

```
typedef struct stu
{
 //数据域（自定义）
 int num;
 char name[32];
 float score;

 //指针域 保存下一个节点的地址
 struct stu *next;
}STU;

void test07()
{
 //链表头
 STU *head = NULL;
 //遍历链表
 STU *pb = NULL;
 STU data1 = {100,"德玛", 59};
 STU data2 = {101,"小炮", 89};
 STU data3 = {102,"小法", 79};
 STU data4 = {103,"盲僧", 99};
 STU data5 = {104,"快乐风男", 39};

 head = &data1;
 data1.next = &data2;
 data2.next = &data3;
 data3.next = &data4;
 data4.next = &data5;
 data5.next = NULL;

 pb = head;
 while(pb != NULL)
 {
 printf("%d %s %f\n", pb->num,pb->name,pb->score);
 pb=pb->next;//pb指向下一个节点
 }
```

```
}
```

知识点7【动态链表操作】1-2

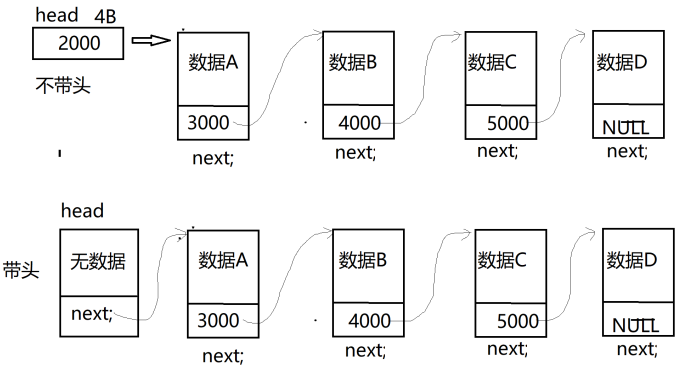
1、布局整个程序框架

main.c

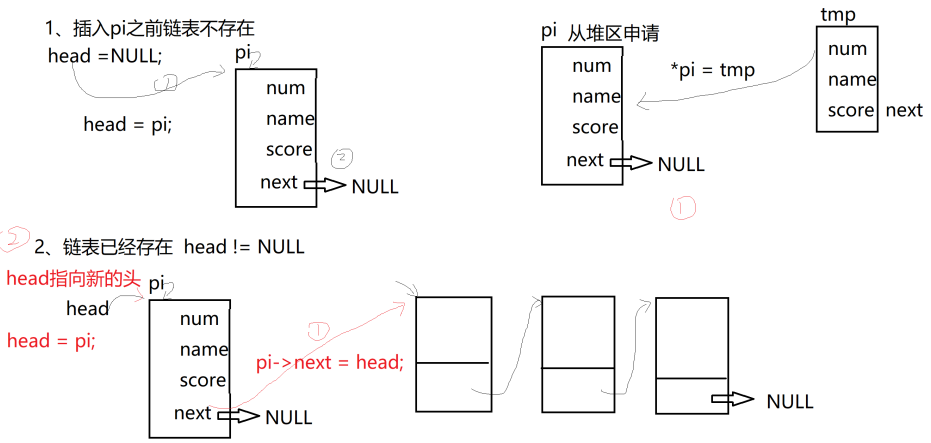
```c
#include<stdio.h>

void stu_help(void);
int main(int argc,char *argv[])
{
 stu_help();

 while(1)
 {
  char cmd[32]="";
  printf("请输入操作指令:");

  scanf("%s",cmd);
  if(strcmp(cmd,"help") == 0)
  {
  stu_help();
  }
  else if(strcmp(cmd,"insert") == 0)
  {
  printf("-----insert------\n");
  }
  else if(strcmp(cmd,"print") == 0)
  {
  printf("-----print------\n");
  }
  else if(strcmp(cmd,"search") == 0)
  {
  printf("-----search------\n");
  }
  else if(strcmp(cmd,"delete") == 0)
  {
  printf("-----delete------\n");
  }
  else if(strcmp(cmd,"free") == 0)
  {
```

```
36   printf("-----free------\n");
37   }
38   else if(strcmp(cmd,"quit") == 0)
39   {
40   break;
41   }
42
43   }
44   return 0;
45  }
46
47  void stu_help(void)
48  {
49   printf("#################################\n");
50   printf("#help:打印帮助信息 #\n");
51   printf("#insert:插入链表节点 #\n");
52   printf("#print:遍历链表节点信息 #\n");
53   printf("#search:查询链表节点 #\n");
54   printf("#delete:删除链表节点 #\n");
55   printf("#free:释放链表 #\n");
56   printf("#quit:退出 #\n");
57   printf("#################################\n");
58   return;
59  }
```

2、链表插入节点 之 头部之前插入。



原理分析图：



案例：

main.c

```c
#include<stdio.h>
#include<string.h>
#include "link.h"
void stu_help(void);
int main(int argc,char *argv[])
{
 //定义一个链表头 注意 一定要赋值为NULL
 STU *head=NULL;

 stu_help();

 while(1)
 {
 char cmd[32]="";
 printf("请输入操作指令:");

 scanf("%s",cmd);
 if(strcmp(cmd,"help") == 0)
 {
 stu_help();
 }
 else if(strcmp(cmd,"insert") == 0)
 {
 STU tmp;
 printf("请输入需要插入的数据:");
 scanf("%d %s %f",&tmp.num, tmp.name, &tmp.score);

 //将tmp数据 插入到head所指向的链表中
 head = insert_link(head, tmp);
```

```
 }
 else if(strcmp(cmd,"print") == 0)
 {
 print_link(head);
 }
 else if(strcmp(cmd,"search") == 0)
 {
 printf("-----search------\n");
 }
 else if(strcmp(cmd,"delete") == 0)
 {
 printf("-----delete------\n");
 }
 else if(strcmp(cmd,"free") == 0)
 {
 printf("-----free------\n");
 }
 else if(strcmp(cmd,"quit") == 0)
 {
 break;
 }

 }
 return 0;
}

void stu_help(void)
{
 printf("################################\n");
 printf("#help:打印帮助信息 #\n");
 printf("#insert:插入链表节点 #\n");
 printf("#print:遍历链表节点信息 #\n");
 printf("#search:查询链表节点 #\n");
 printf("#delete:删除链表节点 #\n");
 printf("#free:释放链表 #\n");
 printf("#quit:退出 #\n");
 printf("################################\n");
 return;
}
```

link.c

```c
#include<stdio.h>
#include<stdlib.h>//calloc
#include"link.h"
STU* insert_link(STU *head, STU tmp)
{

 //1、从堆区申请一个待插入的节点空间
 STU *pi = (STU *)calloc(1,sizeof(STU));
 if(pi == NULL)
  {
  perror("calloc");
  return head;
  }

  //2、将tmp的值 赋值 给*pi
  *pi = tmp;
  pi->next = NULL;//注意

  //3、将pi插入到链表中
  if(head == NULL)//链表不存在
  {
  head = pi;
  //return head;
  }
  else//链表存在（头部之前插入）
  {
  //1、让pi 指向就的头
  pi->next = head;

  //2、head指向新的头节点
  head = pi;

  //return head;
  }

  return head;
}

void print_link(STU *head)
{
 if(head == NULL)//链表不存在
```

```
 {
 printf("link not find\n");
 return;
 }
 else
 {
 STU *pb = head;
 while(pb != NULL)
 {
 printf("%d %s %f\n", pb->num, pb->name,pb->score);
 //pb指向下一个节点
 pb = pb->next;
 }

 }

 return;
}
```

link.h

```
//防止头文件重复包含
#ifndef __LINK_H__
#define __LINK_H__
//链表节点类型 定义
typedef struct stu
{
 //数据域
 int num;
 char name[32];
 float score;

 //指针域
 struct stu *next;
}STU;

extern STU* insert_link(STU *head, STU tmp);
extern void print_link(STU *head);
#endif

```