自定义类型：结构体、枚举、联合

1 结构体类型

1.1 结构体定义与声明

结构是一组值得集合，而结构体成员可以是不同类型的数据，常用来描述复杂对象。

（1）定义与声明：1.关键字加标号得到结构体类型名2.大括号中定义成员信息，注意分号，这样自定义了结构体类型名3.结构体类型名定义结构体变量。

（2）结构体的自引用

结构体的自引用不是结构体中定义结构体类型的变量，无线递归

struct Note

{

int data;

struct Note n;

};

**而是在结构体中定义变量的地址**，类似链表的数据域和指针域。

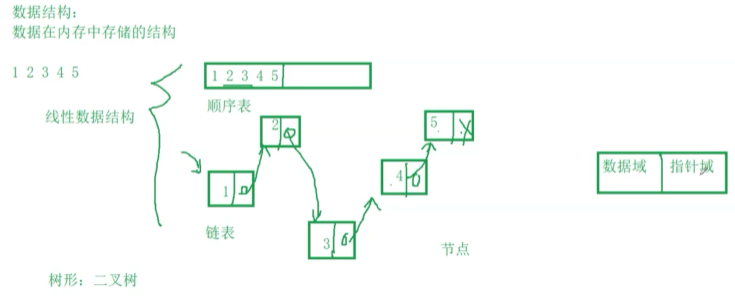
struct Note

{

int data;

struct Note\* next;//指针

};



结构体变量定义，初始化，引用，已经讲过了。

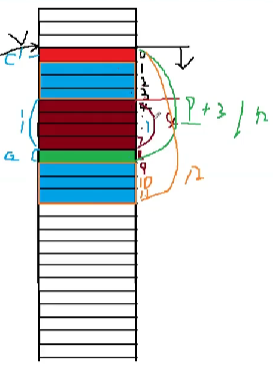
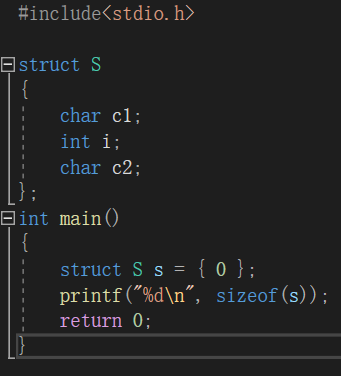
1.2 结构体内存对齐

结构体内存对齐规则，或者说存放规则：

（1）结构体第一个成员放在结构体变量在内存中存储位置0偏移量开始。

（2）从第2个成员往后的所有成员，都放在一个对齐数（成员所占字节与默认对齐数VS为8的较小值）的整数倍的地址处。Linux没有默认对齐数的说法。

（3）结构体的总大小是所有成员的最大对齐数的整数倍。



（4）如果嵌套了结构体的情况，嵌套的结构体对齐到自己的最大对齐数的整数倍处，结构体的整体大小就是所有对齐数（含嵌套结构体对齐数）的整数倍。

#include<stdio.h>

struct S1

{

double a1;

char b1;

int c1;

};

struct S2

{

char a2;

struct S1 s1;

double c2;

};

int main()

{

struct S2 s2 = { 0 };

printf("%d\n", sizeof(s2));

//按照结构体存放规则，s2变量：1.成员a2 0偏移 2.成员s1 嵌套结构体，取他的最大对齐数对齐，得到对齐数为8

//则从偏移量8-23 3.成员c2 所占内存大小为8/8，对齐数为8，则从偏移量24-31 4.结构体总大小应为1/8/8 的倍数

//0-31 32个字节，满足条件，则结构体大小为32.

return 0;

}

为什么存在内存对齐呢？

（1）平台原因（移植原因）：不是所有硬件平台都能访问任意地址上的任意数据的；某些硬件平台只能在某些地址处取某些特定类型的数据，否则会报出硬件异常。

（2）性能原因：数据结构（尤其是栈）应该尽可能的在自然边界上对齐。原因在于，为了访问为对齐的内存，可能处理器需要进行多次访问内存；而对齐的内存则访问次数更少。

总的来说，结构体的内存对齐是一种牺牲内存空间来换取运行效率缩短时间的做法。我们也可以采用一些方法得到一举两得的效果，如将占有内存小的变量放到一起。

修改默认对齐数：2的次方

#include<stdio.h>

//VS默认对齐数为8

//假设修改默认对齐数为2

#pragma pack(2)

struct S1

{

double a1;//8/2 2

char b1;//1/2 1

int c1;//4/2 2

};

#pragma pack()

int main()

{

struct S1 s1 = { 0 };

printf("%d\n", sizeof(s1));//0-7 8 10-13 总大小14

return 0;

}

查看成员相对于结构体起始地址的偏移量：宏**size\_t** **offsetof(** *structName***,** *memberName* **);//结构体类型名，成员名，返回成员的偏移量**

#include<stdio.h>

#include<stddef.h>

//VS默认对齐数为8

struct S1

{

double a1;//8/8 8

char b1;//1/8 1

int c1;//4/8 4

};

int main()

{

struct S1 s1 = { 0 };

printf("%d\n", sizeof(s1));//0-7 8 12-15 总大小16

printf("%d\n", offsetof(struct S1, a1));

printf("%d\n", offsetof(struct S1, b1));

printf("%d\n", offsetof(struct S1, c1));

return 0;

}

1.3 结构体传参

结构体传参，可以采用传值调用或传址调用，而由于结构体空间都比较大，在传值调用时又必须创建一个同样大小的空间，则造成了空间和时间的浪费，同时，传值调用还不能对实参进行改变，具有局限性。而传址调用则很好的克服了这些问题，由此，**结构体传参或者数组传参时尽量采用传址调用，通过指针来接收，如不需要改变内容时可以使用const修饰指针**。

#include<stdio.h>

struct S

{

int data[100];

int num;

};

void print1(struct S s)

{

printf("%d\n", s.num);

}

void print2(struct S\* p)

{

printf("%d\n", p->num);

}

int main()

{

struct S s = { {1,2,3,4}, 10 };

print1(s);

print2(&s);

return 0;

}

1.4 位段

位段是一种类似结构体的自定义类型，它是按照我们的需要来分配内存空间的，按bit位来分配。

1.4.1 位段的定义

（1）结构与结构体相似，但是成员的类型为int或unsigned int或signed int或char型等。

（2）变量后跟冒号和数字组成，表示变量所占的bit位数。

#include<stdio.h>

struct A

{

int \_a : 3;//\_a占3个bit位

int \_b : 5;//\_b占5个bit位

int \_c : 10;//\_c占10个bit位

int \_d : 25;//\_d占25个bit位

};

int main()

{

printf("%d\n", sizeof(struct A));//8

return 0;

}

1.4.2 位段的存储

（1）位段成员的类型为int或unsigned int或signed int或char型等。

（2）位段的空间是按照每次开辟4个字节（int）或1个字节（char）的方式来开辟的，当前面开辟的空间不足时，直接开辟下一段空间。如上：首先开辟4个字节32位，可以看到a,b,c用掉了18个bit位，而当d需要25个bit位时，空间不够，所以直接开辟下一个4字节比特位，由此该位段的大小是8字节。

（3）位段涉及**很多不确定因素**，位段是**不跨平台的**，注重可移植性的程序应避免使用位段。

注：位段对内存空间进行了更为细致的划分，使得空间得到更有效的利用。

**VS中位段的数据存入：**

（1）先使用低地址再使用高地址。

（2）当前空间段不够时，开辟新的空间并直接使用，之前剩余的空间不存。

（3）位段成员的位数不能超过类型大小。

#include<stdio.h>

struct S

{

char a : 3;

char b : 4;

char c : 5;

char d : 4;

};//3个字节 62 03 04

int main()

{

struct S s = { 0 };

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

s.a = 10;//先使用低地址再使用高地址

s.b = 12;//当前空间段不够时，开辟新的空间并直接使用

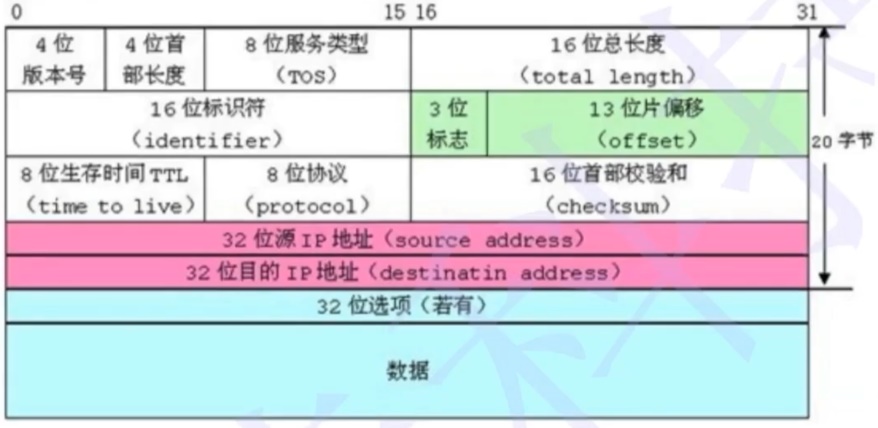
s.c = 3;//注意变量所占的bit数，进行必要的补充或截断

s.d = 4;

return 0;

}

**位段的应用：如网络数据包，避免空间的浪费。**



2 枚举类型

能够一一列举的数据类型，如性别、三原色、手指等，有限次确定的列举，且每次只能为其中之一，成员为常量，不能被改变。系统默认列举成员由0开始表示。

#include<stdio.h>

enum Gender

{

MAN = 1,

WOMAN,

NO

};

int main()

{

enum Gender G1 = NO;

//NO = 1;//常量不能被改变

printf("%d\n", G1);

printf("%d\n", MAN);

printf("%d\n", WOMAN);

printf("%d\n", NO);

printf("%d\n", sizeof(G1));//只能是其中之一，一个int就可以了

printf("%d\n", sizeof(enum Gender));

return 0;

}

应用：常用在情况能够确定且可以一一列举的情况，用数字代表含义，如四则运算。与宏有点类似，但枚举是一种类型，宏则不是。

3 联合体类型

联合体又称共用体，即他的成员共用一段空间，同一时间只能对一个成员进行操作，且由于空间相同，存取操作都会互相影响，注意内存分析。

#include<stdio.h>

union Un

{

char c;

int i;

};

int main()

{

union Un u1 = { 0 };

u1.i = 1;//对联合体四个字节赋值为1

if (u1.c == 1)//由于内存共用，取出一个字节的内容判断大小端

{

printf("小端\n");

}

else

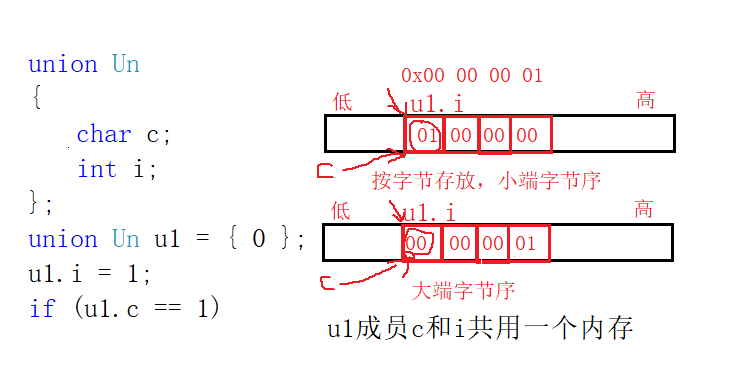
{

printf("大端\n");

}

return 0;

}



联合体的大小至少为成员的最大字节，且存在对齐，联合体的总大小为最大对齐数的整数倍。

#include<stdio.h>

union Un

{

char c;

char c2[9];

int i;

double b;

};

int main()

{

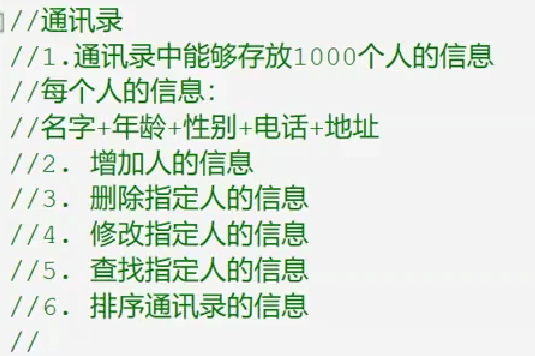
printf("%d\n", sizeof(union Un));

return 0;

}

4 通讯录

通讯录的思想：



软件组成：

