C/C++位域知识小结

Posted on 2013-04-22 21:35 扬名 阅读(1130) 评论(0) 编辑 收藏

几篇较全面的位域相关的文章：

http://www.uplook.cn/blog/9/93362/

C/C++位域(Bit-fields)之我见

C中的位域与大小端问题

内存对齐全攻略–涉及位域的内存对齐原则

本文主要对位域相关知识进行了一下梳理，参考如下：

C语言中的位域

史上最全的C位域总结2

C结构体之位域（位段）

C/C++中以一定区域内的位(bit)为单位来表示的数据成为位域，位域必须指明具体的数目。

位域的作用主要是节省内存资源，使数据结构更紧凑。

1. 一个位域必须存储在同一个字节中，不能跨两个字节，故位域的长度不能大于一个字节的长度。

如一个字节所剩空间不够存放另一位域时，应从下一单元起存放该位域。也可以有意使某位域从下一单元开始。例如：

```
struct BitField
{
   unsigned int a:4;  //占用4个二进制位;
   unsigned int  :0;  //空位域,自动置0;
   unsigned int b:4;  //占用4个二进制位,从下一个存储单元开始存放;
   unsigned int c:4;  //占用4个二进制位;
   unsigned int d:5;  //占用5个二进制位,剩余的4个bit不够存储4个bit的数据,从下一个存储单元开始存放;
   unsigned int  :0;  //空位域,自动置0;
   unsigned int e:4;  //占用4个二进制位,从这个存储单元开始存放;
};
```

2. 取地址操作符**&**不能应用在位域字段上**;**

3. 位域字段不能是类的静态成员**;**

4. 位域字段在内存中的位置是按照从低位向高位的顺序放置的**;**

一周好文(23)
C++(16)
cocos2d(15)
objective-C(5)
算法(4)
随笔(3)
Linux(3)
工作(1)
自制小工具(1)
更多

随笔档案(139)
2013年4月 (5)
2013年3月 (6)
2013年2月 (7)
2012年12月 (1)
2012年11月 (5)
2012年9月 (8)
2012年8月 (7)
2012年7月 (14)
2012年6月 (11)
2012年5月 (13)
2012年4月 (25)
2012年3月 (3)
2012年2月 (7)
2012年1月 (1)
2011年9月 (2)
2011年7月 (5)
2011年3月 (1)
2011年2月 (1)
2011年1月 (2)
2010年12月 (1)
2010年10月 (6)
2010年9月 (6)
2010年8月 (2)


```
struct BitField
  {
    unsigned char a:2;  //最低位;
    unsigned char b:3;
    unsigned char c:3;  //最高位;
  };
  union Union
  {
    struct BitField bf;
    unsigned int n;
  };
  union Union ubf;
  ubf.n = 0;    //初始化;
  ubf.bf.a = 0; //二进制为: 000
  ubf.bf.b = 0; //二进制为: 000
  ubf.bf.c = 1; //二进制为: 001
  printf("ubf.bf.n = %u\n", ubf.n);
```

位域中的位域字段按照从低位向高位顺序方式的顺序来看,那么,a、b、c这三个位域字段在内存中的放置情况是:

最高位是c:001,中间位是b:000,最低位是a:000;所以,这个位域结构中的8二进制内容就是: 00100000,总共8个位,其十进制格式就是32;

实际上打印出来的ubf.n值就是32;

ubf.n = 100; //二进制为: 01100100

printf("ubf.bf.a = %d, ubf.bf.b = %d, ubf.bf.c = %d\n", ubf.bf.a, ubf.bf.b, ubf.bf.c);

此时,对于位域ubf.bf来说,其位于字段仍然按照从低位向高位顺序方式的顺序放置,则,最高位是c:011,中间位是b:001,最低位是a:00;

所以,ubf.bf.a = 0; ubf.bf.b = 1; ubf.bf.c = 3;

实际上打印出来的结果也的确如此;不够存储下一个位域的4位,故设为空位域,不使用,自动置0;e从第四个字节处开始存放,占用4位;

5. 位域的对齐

1. 如果相邻位域字段的类型相同，且其位宽之和小于类型的sizeof大小，则后面的字段将紧邻前一个字段存储，直到不能容纳为止；

2. 如果相邻位域字段的类型相同，但其位宽之和大于类型的sizeof大小，则后面的字段将从新的存储单元开始，其偏移量为其类型大小的整数倍；

3.如果相邻的两个位域字段的类型不同,则各个编译器的具体实现有差异,VC6采取不压缩方式,GCC和Dev-C++都采用压缩方式;

4. 整个结构体的总大小为最宽基本类型成员大小的整数倍。

5. 如果位域字段之间穿插着非位域字段，则不进行压缩；（不针对所有的编译器）

```
  struct BFA
  {
    unsigned char a:2;
    unsigned char b:3;
    unsigned char c:3;
  };
  struct BFB
  {
    unsigned char a:2;
```

并行队列的执行顺序与其加入队列的顺序相同。并行队列不是并发执行的吗？执行顺序怎么会和加入队列的顺序相同呢？

--Weiki

4. Re:ACE小记

博主有没有在windows环境下用codeblocks编译过ace的示例代码啊？

--yyrdl

5. Re:iOS多线程GCD

@蝗虫的大腿&表示取地址符，这个是定义一个静态变量，然后在dispatch_once函数第一次运行时写入数据，之后就不会再次写入，可以保证后面block函数内部的代码只被执行一次...

--蜗牛不会跑

```
    unsigned char b:3;
    unsigned char c:3;
    unsigned int  d:4;   //多出来这个位域字段;
};
```

sizeof(BFA)=1, sizeof(BFB)=8;

这也说明了第三点中"相邻两个位于字段类型不相同时,VC6采取不压缩的方式"

6. 当要把某个成员说明成位域时,其类型只能是**int,unsigned int**与**signed int**三者之一**(**说明**:int**类型通常代表特定机器中整数的自然长度。**short**类型通常为**16**位**,long**类型通常为**32**位**,int**类型可以为**16**位或**32**位**.**各编译器可以根据硬件特性自主选择合适的类型长度**.**见**The C Programming Language**中文 **P32)**。

尽管使用位域可以节省内存空间,但却增加了处理时间,在为当访问各个位域成员时需要把位域从它所在的字中分解出来或反过来把一值压缩存到位域所在的字位中.

```
#include <iostream>
 #include <memory.h>
 using namespace std;
 struct A
 {
     int a:5;
     int b:3;
 };
 int main(void)
 {
     char str[100] = "0134324324afsadfsdlfjlsdjfl";
         struct A d;
     memcpy(&d, str, sizeof(A));
     cout << d.a << endl;
     cout << d.b << endl;
     return 0;
 }
```

在32位x86机器上输出：

```
高位 00110100 00110011   00110001    00110000 低位
       '4'       '3'       '1'          '0'
其中d.a和d.b占用d低位一个字节（00110000）,d.a : 10000, d.b : 001
```

解析：在默认情况下，为了方便对结构体内元素的访问和管理，当结构体内的元素长度都小于处理器的位数的时候，便以结构体里面最长的元素为对其单位，即结构体的长度一定是最长的数据元素的整数倍；如果有结构体内存长度大于处理器位数的元素，那么就以处理器的位数为对齐单元。由于是32位处理器，而且结构体中a和b元素类型均为int（也是4个字节），所以结构体的A占用内存为4个字节。

上例程序中定义了位域结构A，两个个位域为a（占用5位），b（占用3位），所以a和b总共占用了结构A一个字节（低位的一个字节）。

当程序运行到14行时，d内存分配情况：

```
高位 00110100 00110011   00110001    00110000 低位
       '4'       '3'       '1'          '0'
其中d.a和d.b占用d低位一个字节（00110000）,d.a : 10000, d.b : 001
```

d.a内存中二进制表示为10000，由于d.a为有符号的整型变量，输出时要对符号位进行扩展，所以结果为-16（二进制为11111111111111111111111111110000）

d.b内存中二进制表示为001，由于d.b为有符号的整型变量，输出时要对符号位进行扩展，所以结果为1（二进制为00000000000000000000000000000001）

另一个例子，来自http://blog.chinaunix.net/uid-28697486-id-3511598.htm

```
#include "stdio.h"

void main(int argn ,char *argv)
{
    struct     test {
        unsigned a:10;
        unsigned b:10;
        unsigned c:6;
        unsigned :2;//this two bytes can't use
        unsigned d:4;
        }data,*pData;
    data.a=0x177;
    data.b=0x111;
    data.c=0x7;
    data.d=0x8;

    pData=&data;
    printf("data.a=%x data.b= %x data.c=%x data.d=%xn",pData->a,pData->b,pData->c,pData->d);//位域可以使用指针

    printf("sizeof(data)=%dn",sizeof(data));    //4 bytes ，最常用的情况

    struct testLen{
    char a:5;
    char b:5;
    char c:5;
    char d:5;
    char e:5;
    }len;

    printf("sizeof(len)=%dn",sizeof(len));      //5bytes 规则2

    struct testLen1{
        char a:5;
        char b:2;
        char d:3;
        char c:2;
        char e:7;
        }len1;
    printf("sizeof(len1) =%dn",sizeof(len1));     //3bytes 规则1

    struct testLen2{
        char a:2;
        char :3;
        char b:7;
        long d:20; //4bytes
        char e:4;
        }len2;
    printf("sizeof(len2)=%dn",sizeof(len2));  //12 规则3，4,5，总长为4的整数倍，2+3 占1byte，b占1bye 由于与long对其，2+3+7 占4字节，
后面 d 与 e进行了优化 占一个4字节
```

```
    struct testLen3{
        char a:2;
        char :3;
        char b:7;
        long d:30;
        char e:4;
        }len3;
    printf("sizeof(len3)=%dn",sizeof(len3));//12 规则3，4，5，总长为4的整数倍，2+3 占1byte，b占1bye 由于与long对其，2+3+7 占4字节，后面 d占一个4字节，为了保证与long对其e独占一个4字节
}
```

另：C++标准库提供了一个bitset 类模板，它可以辅助操纵位的集合。在可能的情况下应尽可能使用它来取代位域。

标签: iphone, C++

绿色通道： 好文要顶 关注我 收藏该文 与我联系

扬名
关注 - 8
粉丝 - 50
+加关注

0 0

(请您对文章做出评价)

« 上一篇：Big-Endian & Little-Endian

刷新评论 刷新页面 返回顶部

发表评论

昵称： 纯蓝犹豫

评论内容：

提交评论　注销　订阅评论

[使用Ctrl+Enter键快速提交]

最新**IT**新闻:
· 甲骨文、微软、IBM和谷歌都盯上了一家市值400亿美元的公司
· 微软专利获批，有了会情绪识别的可穿戴设备，人人都是福尔摩斯
· 不能只靠电商：马云想用5万人服务20亿人 创10万亿交易额
· Android Chrome的分裂现状
· 硅谷风投大佬Paul Graham：成为好的天使投资人的12条原则
» 更多新闻...

最新知识库文章:
· 携程App的网络性能优化实践
· 技术领导力：作为技术团队领导经常为人所忽略的技能和职责
· 在LinkedIn做面试官的故事

· 架构之重构的12条军规（上）
· 序列化和反序列化
» 更多知识库文章...