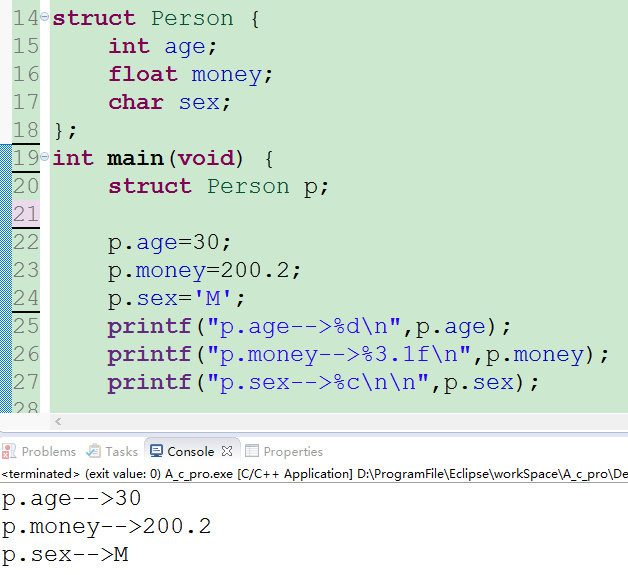
**结构体**

结构体是C语言中自定义的数据类型，是一组变量的集合，有别于数组，数组仅限于同一种数据类型，而结构体可以是任何数据类型，包括数组。结构体里面的每一个变量或者数组都是结构体的成员。

**下面来看看结构体的基本使用**

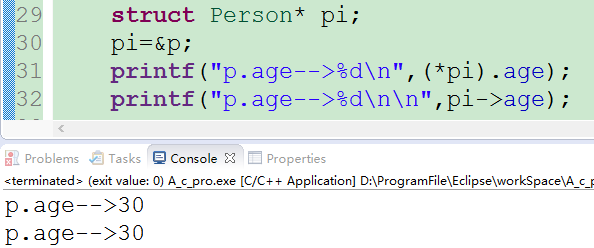


14-18行，定义了一个名称为Person的结构体，结构体里面有 age，money，sex 三种数据类型；

20-24行，是结构体的声明一个名为 P 的 Person 变量，并赋值；

25-27行，是结构体的基本使用，从中可以看出的是，结构体取值的时候和Java中极为相似，都是中间 . 的形式进行取其中某个变量的值。

下面来看一下**结构体指针的使用。**



29-30行，定义了一个结构体指针，名为 pi，并将结构体变量 p 的地址赋值给 pi 。

32行，结构体指针的使用，既然 pi 是结构体 p 的指针，那么，取结构体那就是前面加个 \* 咯，就这么简单，照葫芦画瓢嘛，现在取到了结构体 p，那么去 p 里面的就简单了，直接使用前面的方式中间加个 . 即可；

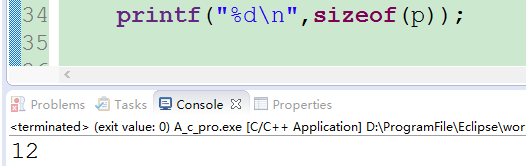
33行，则是指针结构体的另一种使用方式，直接使用 -> 就可以取里面相应的值了，比32行的方式是不是假单很多呢？

**结构体的长度**

结构体和结构体指针都说完了，最后来看一下结构体的长度。

长度简单嘛，直接把里面的累加不就得了，int 和 float 都是占 4 个字节，而 char 占 1 个字节，那么就直接加起来就是 9 不就得了。

到底是不是，验证一下就知道咯。



看，打印出来的结果是12，咦，怎么那么奇怪，难道 char 也占 4 个字节？

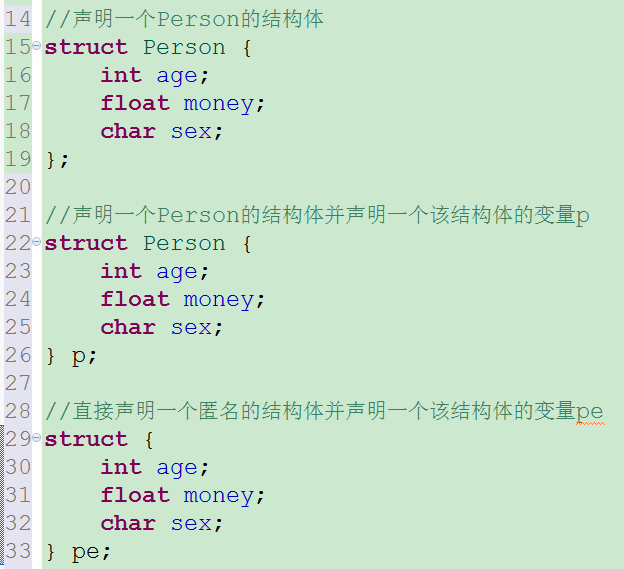
没错，就是这样，char 也是占 4 个字节，为什么呢？因为里面的其他两个变量的长度都是 4，那么把里面的 char 也定义成 4 个字节，这样可以方便的指针偏移。

但这不是绝对的，同样的代码，执行的结果也不一样，这和编译器以及运行环境的不一样，哈哈。

这里需要注意的是，虽然都是4个字节，前面也说了这样是为了方便指针的偏移，那么自然的就想到指针的加减操作，那么我们就这样使用行不行？

如果我要取它的money字段，那么我就这样：printf("%f",\*(p+1)); 行不行呢？不行，自己可以去尝试一下！

最后附上**结构体的三种写法：**



代码里面注释写的也比较清楚了，这里就不再多说了，结构体的介绍到此结束，下面来看联合体

**联合体：**

联合体判断大小端

由于联合体所有数据共享一块地址空间，存放数据的所有成员都是从低地址开始存放，所以我们可以在联合体内定义一个int和一个char类型变量，然后在外部实例化的时候创建int变量，用char变量调用，相当于隐式类型转化，如果结果为1，则低字节存放在低地址，既是小端机器，反之大端机器。

#include <iostream>

using namespace std;

union Test

{

int a;

char b;

};

int main()

{

Test t;

t.a = 1;

if (t.b == 1)

cout << "小端机器" << endl;

else

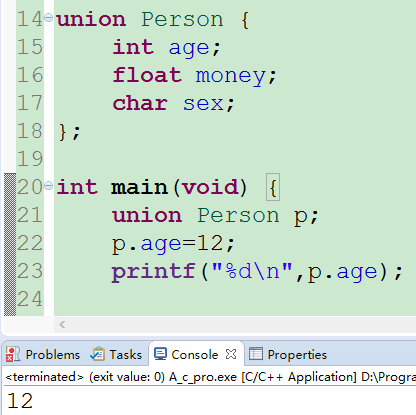
cout << "大端机器" << endl;

return 0;

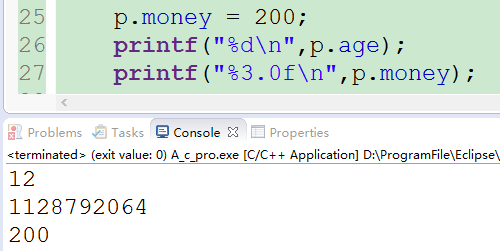
}

结构体搞懂了，联合体就简单了。

联合体和结构体很相似，**结构体的关键字是 struct 而联合体的关键字是 union**  。看代码：



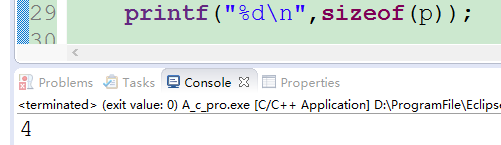
这看起来和结构体没什么差别嘛。没差别就不会出现联合体咯。所以再看下面的代码：



接上面的代码，第一行的打印结果是12，这个不用过多的解释，然后当你去给联合体中的 money 去赋值之后，你再打印一下 age 的值，你就会发现，age 的值已经完全不对了，而你再去打印一下 money 的值，你就会发现 money 的值是对的。

所以，这就是**两者的区别，结构体里面的数据是会不断的覆盖的，而结构体里面的数据是相互独立的。**

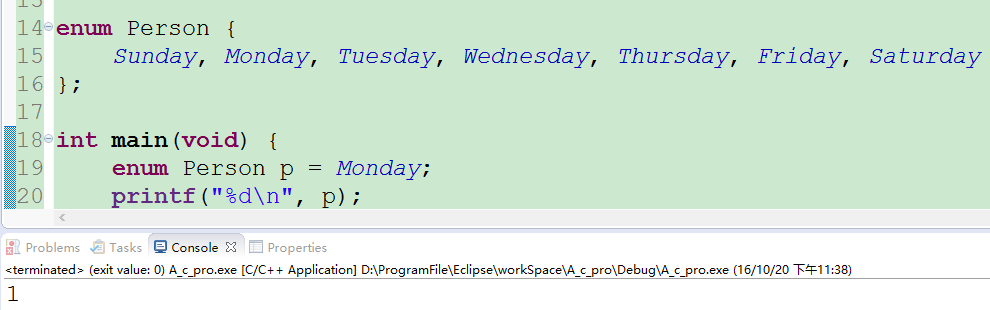
现在问题来了，联合体的长度是多少呢？还是累加？累加就没有意义了，占那么大的内存存那么少的数据是得不偿失的，所以，联合体的长度就是取该联合体中最长的一个变量的长度。



联合体的介绍也就到这里。

结构体和联合体都介绍完了，最后来介绍一个枚举，**枚举就是穷举，也就是说，只能是这里面的值中的某一个，不可能是其他。**

看看代码：



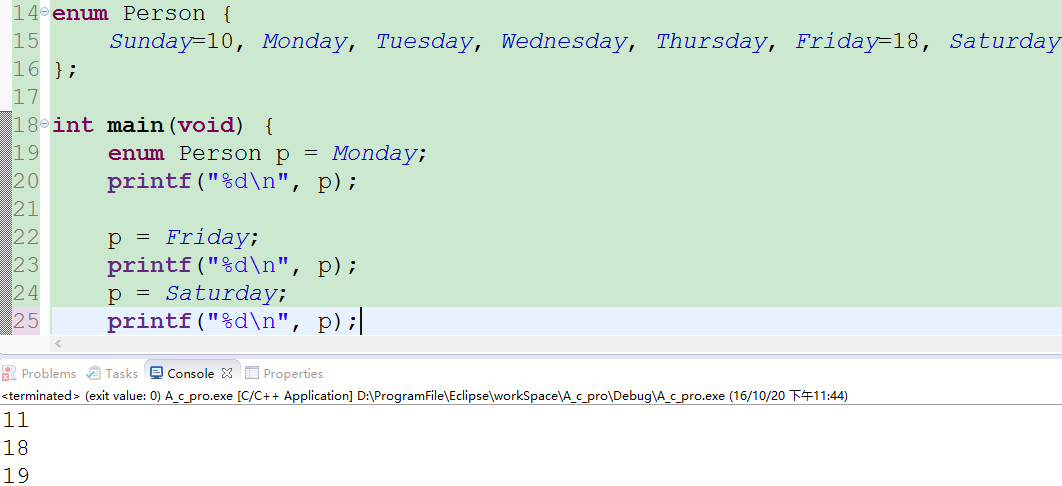
14-17行是枚举类型的声明；

19行是定义并赋值；

20行是打印结果，打印的结果为什么会是int 类型的值呢？又为什么是 1 呢？

这很简单，枚举是穷举，那么就是有限的，而不是无限的，既然是有限的，那么里面就会对枚举的每一个值进行编号，像数组一样，从0开始 编号，这就是为什么打印出来的值是int，为什么是1呢？因为Monday在枚举中的排行是第2啊，就这么简单。

当然，如果你不喜欢从0开始编号，也可以自定义一个初始值：



从代码上可以看到，枚举不仅可以自己自定义起始的值，中间也可以重新自定义，并且后面的也会依次的+1

枚举(enum)

enum相当于int类型,所以sizeof(enum结构名)时总是为4

枚举，就先到这里。

C++定义的结构名、联合名、枚举名 都是类型名，可以直接用于变量的声明或定义。即在C++中定义变量时不必在结构名、联合名、枚举名 前加上前缀struct、union、enum。

enum color {red,blak,white,blue,yellow};

struct student {char name[6]; int age; int num;};

union score {int i\_sc; float f\_sc;};</span></span>

在C中的使用方法:

<span style="font-size:14px;"><span style="font-size:12px;">#include "head.h"

int main(void)

{

enum color en\_col;

struct student st\_stu; union score un\_sc;

union score un\_sc;

//....

return 0;

}

在C++中的使用方法:

<span style="font-size:14px;"><span style="font-size:12px;">#include "head.h"

int main(void)

{

color en\_col;

student st\_stu;

score un\_sc;

//....

return 0;

 2.1 C++中struct 和 class 的区别

在C++中struct也是一种类,他与class具有相同的功能,用法完全相同。唯一的区别就是：在没有指定成员的访问权限时，struct中默认为public权限，class中默认为private权限。

2.2 C++中的 union 和 class 的区别

union支持 public , protected 以及 private 权限。读者看到这可能会问，要是这样的话，union与class还有什么区别吗？区别还是有的

1. union不支持继承

2. union中不能定义虚函数。

3.在没有指定成员的访问权限时,union中默认为public权限

4.union中的成员类型比class少

**设置为4字节对齐  
#pragma  pack(4)**

**struct student  other;**

**//other={3,25,"king",100,80};                             //error 仅在定义时可以用"{}"直接初始化**

3.引用传递   
swap(int \*a,int \*b);   
swap(&a,&b);   
效果等同于：   
swap(int &a,int &b);   
swap(a,b);

2。用指向结构体的指针作为函数参数：

这句话听起来很绕，其实就是指针作为函数参数，（指针类型为指向结构体的指针）

下面举个例子：

void ff(struct student \*p)

main()

{

struct student a;

ff(struct &a);

}

<https://blog.csdn.net/qq_36946274/article/details/80891239>

enum是枚举类型，但说它是一种类型，它和内置类型及类类型都有些许不同。它的用法真的很奇怪，我们一起来看一看：

enum week{monday,tuesday,wednesday,thursday,friday,saturday,sunday};

week date=monday;

1

2

乍一看，enum和类的用法差不多啊，这样想你就错了，它会让你大开眼界：

enum {monday.....sunday};

int date=monday;

1

2

就问你强不强，你类类型能这么用？类对象不带名字，然后里面的成员作用域符号都不用带就能直接用，这个时候，又感觉enum和内置类型差不多了。week这个“中间商”根本就是可有可无，你开心的话，还可以这么用：

enum {monady...sunday} week;

1

或者是这么用

enum{monday...sunday}yesterday=monday,today=tuesday,tommorrow=wednesday;

1

总的来说，我们用enum时用的都是enum里的成员，它有没有“实例化”对象可以跟我们没有半毛钱关系，下面是我从别人那里“拿”来的总结：

(1) 枚举型是一个集合，集合中的元素(枚举成员)是一些命名的整型常量，元素之间用逗号,隔开。

(2) week是一个标识符，可以看成这个集合的名字，是一个可选项，即是可有可无的项。

(3) 第一个枚举成员的默认值为整型的0，后续枚举成员的值在前一个成员上加1。

(4) 可以人为设定枚举成员的值，从而自定义某个范围内的整数。

(5) 枚举型是预处理指令#define的替代。

(6) 类型定义以分号;结束。

对了，忘了介绍一下C++11的新标准，enum也可以像class一样带有作用域：

enum class {monday...sunday};//错误

enum class week{monday...sunday};//正确

int today=monday;//错误

int today=week::monday;//正确

1

2

3

4

我们可以发现，加上class修饰符后，enum这个“坏孩子”就开始从良了，不能再任性地不带对象名，也不能顽皮地不用::修饰符了，它更像是一个类。但是坏孩子的天性是抹杀不掉的，它还是保留了enum的特点，即(3) 第一个枚举成员的默认值为整型的0，后续枚举成员的值在前一个成员上加1。

---------------------