实验环境

C语言实验环境：Code::Blocks 13.12、GNU GCC Compiler

Java语言实验环境：

java version "1.8.0\_60"

Java(TM) SE Runtime Environment (build 1.8.0\_60-b27)

Java HotSpot(TM) 64-Bit Server VM (build 25.60-b23, mixed mode)

文本语言编写：Notepad++

5.7 哪些是声明，哪些是定义？

a) char \*name;

b) type int \* IntPtr;

c) struct rec;

d) int gcd(int, int);

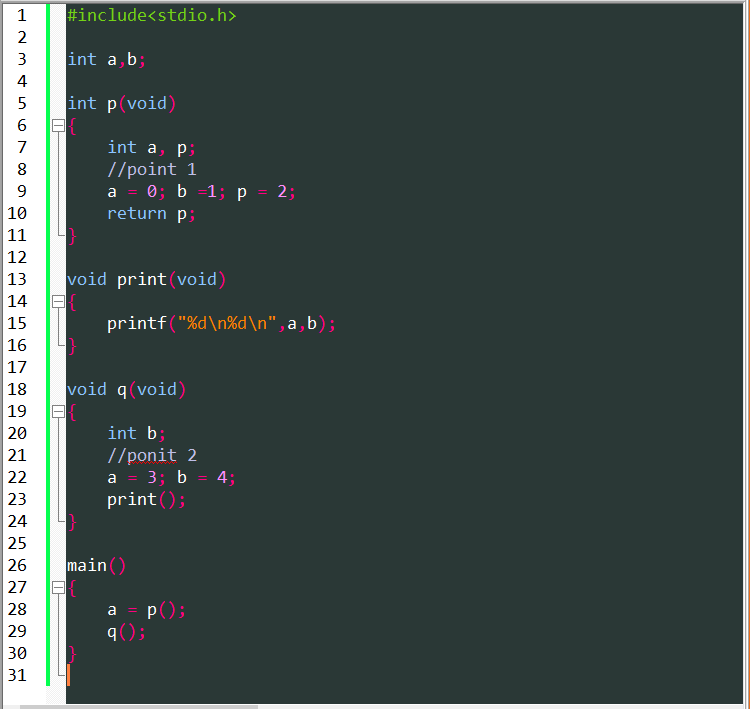
e) extern int z;

解答：

定义：a）、也可以视为既是定义，也是声明。它提供了一个实体在程序中的唯一描述。

声明：b)、c)、d)、e)。将一个名称引入程序。

5.8 下列C程序的符号表：a) 使用静态作用域规则；b) 使用动态作用域规则；各自范围作用下打印什么？

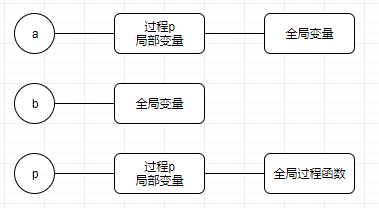


解答：

程序运行结果



1. 静态作用域规则，在编译的时候已经确定了。



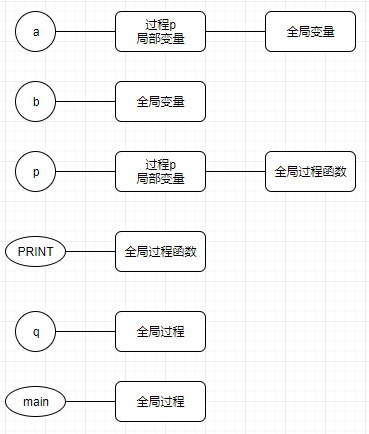
结果输出：

3

1

本次实验中使用的，默认为静态。

1. 动态作用域规则，通过运行过程中活动来确定名字声明。

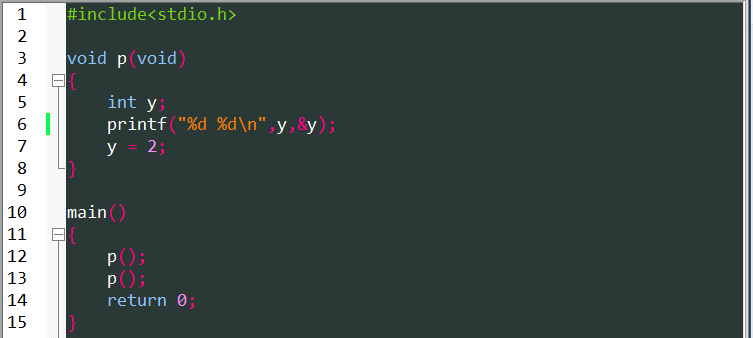


结果输出：

3

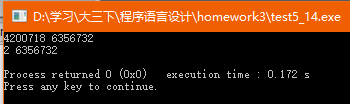
4

5.14 以下程序，第一个值为无用单元（或是0）；第二个值将是2？解释原因。

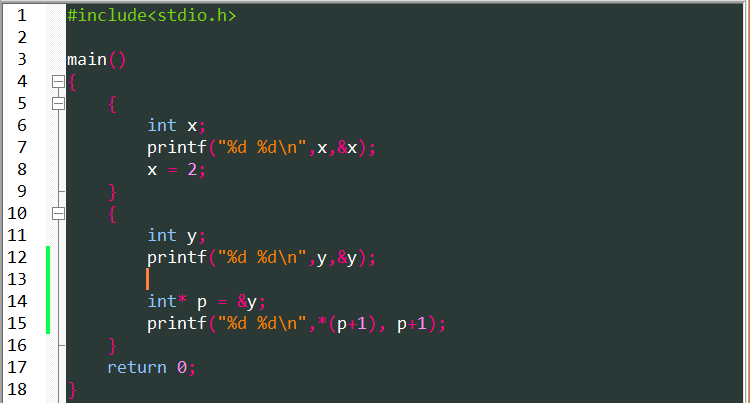


解答：

第一个值，为一个位置地址的内容值，事先并不知道y的地址与其内容，所以第一次输出其值是无效的。当再次调用过程p，y的地址仍取之前相同地址，而在上一个过程中，该地址已经被赋值为2，视为有效的。实验结果如下，可知晓程序运行细节，局部变量的地址。



5.15 以下程序，第一个值为无用单元（或是0）；第二个值将是2？解释原因。

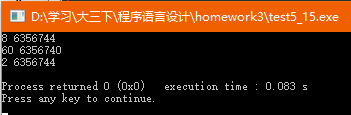


解答：

当编译器，将局部变量x与y的地址取同一个地址时，第一次打印时此地址的内容值是未知的，所以是无用单元，然后经过复制操作，此地址内容值被赋为2，则，打印y时，仍是输出之前x地址的值，即为2。所以第一个值为无用单元，第2个值将是2。这是理论分析！！但是，

经过在codeblock中编写程序，并运行，可以知道实验中，x与y并不是取统一个地址，而y是连续的下一个地址，与x间隔为一个整型长度4byte。所以这种情况下，得出实验结果：

第一个x地址为6356744，值为8(无效值)；y地址6356740，值为60（无效值）；使用指针，指向x地址6356744，其值为2（有效赋值）。



5.17 下面函数声明可用:

1) int pow(int, int);

2) double pow(double, double);

3) double pow(int, double);

以下代码调用pow函数：

int x;

double y;

x = pow(2,3);

y = pow(2,3);

x = pow(2,3.2);

y = pow(2,3.2);

x = pow(2.1,3);

y = pow(2.1,3);

x = pow(2.1,3.2);

y = pow(2.1,3.2);

使用C++写出8次调用pow函数得到的值。

解答：

Code::Block13.12中编辑如下代码

#include<iostream>

**using** **namespace** std**;**

int pow**(**int**,**int**);**

double pow**(**double**,**double**);**

double pow**(**int**,**double**);**

int main**()**

**{**

int x**;**

double y**;**

x **=** pow**(**2**,**3**);**

cout**<<**"x="**<<**x**<<**endl**;**

y **=** pow**(**2**,**3**);**

cout**<<**"y="**<<**y**<<**endl**;**

x **=** pow**(**2**,**3.2**);**

cout**<<**"x="**<<**x**<<**endl**;**

y **=** pow**(**2**,**3.2**);**

cout**<<**"y="**<<**y**<<**endl**;**

x **=** pow**(**2.1**,**3**);**

y **=** pow**(**2.1**,**3**);**

x **=** pow**(**2.1**,**3.2**);**

cout**<<**"x="**<<**x**<<**endl**;**

y **=** pow**(**2.1**,**3.2**);**

cout**<<**"y="**<<**y**<<**endl**;**

**return** 0**;**

**}**

int pow**(**int a**,**int b**)**

**{**

cout**<<**"int pow(int,int):"**;**

**return** 10**\***a**\***b**;**

**}**

double pow**(**double a**,** double b**)**

**{**

cout**<<**"double pow(double,double):"**;**

**return** 10.0**\***a**\***b**;**

**}**

double pow**(**int a**,**double b**)**

**{**

cout**<<**"double pow(int,double):"**;**

**return** 10.0**\***a**\***b**;**

**}**

运行结果：

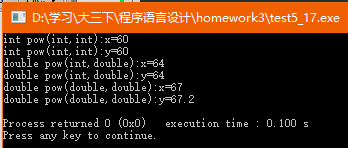






18,19行发生错误，数据类型转化无法完成，是非法的

将此两行注释，后运行。得结果：



Pow函数定义如上代码所示，结果：

x = pow(2,3); 60

y = pow(2,3); 60

x = pow(2,3.2); 64

y = pow(2,3.2); 64

x = pow(2.1,3); 非法

y = pow(2.1,3); 非法

x = pow(2.1,3.2); 67

y = pow(2.1,3.2); 67.2

5.24

（a）下面哪些C语言表达式是l-value，哪些不是，为什么（假定x是int变量，y是int\* 变量）:

1) x+2

2) &x

3) \*&x

4) &x+2

5) \*(&x+2)

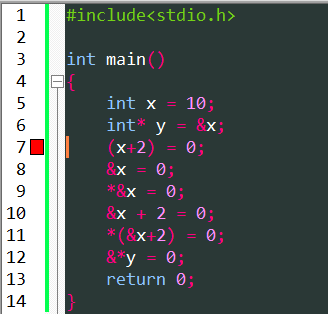
6) &\*y

（b）一个C语言表达式可否虽然是l-value,但不是r-value？请解释。

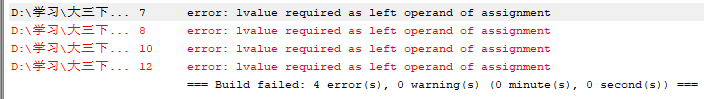
（c）在C语言中，&(&z)永远合法吗，请解释。

解答：

1. 程序代码如下，编写运行后：



编译运行结果如下，7、8、10、12行对应表达式，发生报错：



所以有：

1) x+2 不是l-value，+号计算是(r-value)

2) &x 不是l-value，返回一个指向x地址的指针，是(r-value)

3) \*&x 是l-value，并且等效于x本身

4) &x+2 不是l-value，&x与+号计算都是(r-value)

5) \*(&x+2) 是l-value，\*运算符作用于指针，产生l-value

6) &\*y 不是l-value,返回一个指针是(r-value)，所以(\*&x等效x，但&\*y不等效于y)

1. 答案是否定的，一个l-value一定总是有对应的r-value。
2. &(&x)不一定合法

#include<stdio.h>

int main()

{

int x = 10;

printf("%d\n",&(&x));

return 0;

}

运行报错：



l-value required as unary ‘&’ operand