1. 合法：（1）（3）（4）（5）（10）（12）（13）（14）（15）

不合法：（2）（6）（7）（8）（9）（11）（16）（17）（18）

**原因如下：**

（1）合法的，cc是常量左值，可以赋给左值c

（2）不合法的，cc是const类型，无法修改其值

（3）合法的，可以修改pcc的值，但是pcc指向的const char实体无法修改

（4）合法的

（5）合法的

（6）不合法的，通过pc可以修改cc的值，但实际上cc无法修改

（7）不合法的，pcc是所指单元值只读的指针

（8）不合法的，同（7）

（9）不合法的，cpc为常量指针，其值无法修改

（10）合法的，可以将右值赋给左值pc

（11）不合法的，cc是const类型，无法修改其值

（12）合法的，cpc所指向的实体可以修改

（13）合法的，pcpc为常量二级指针，其指向的一级指针可以赋给pc

（14）合法的，pcpc所指向的一级指针所指向的实体可以被赋为pc所指向实体的值

（15）合法的

（16）不合法的，无法通过引用来修改pcc的值

（17）不合法的，\*pcpc无法修改

（18）不合法的，cpcc的值无法修改

1. （1）定义了一个名为VF\_PC\_RI的类型，该类型定义了一个参数是（char\*,int&）且无返回值的函数

（2）定义了一个名为P\_VF\_PC\_RI的类型，该类型定义了一个指向VF\_PC\_RI类型的指针

（3）定义了一个名为RIFFII的类型，该类型定义了一个参数是（int，int）且返回一个引用整型量的引用的函数

（4）说明一个类型为VF\_PC\_RI的外部函数funca

（5）说明一个类型为P\_VF\_PC\_RI的外部函数ptra

（6）说明一个参数为一个指向P\_VF\_PC\_RI类型的指针的无返回值的外部函数fun1

（7）说明一个参数为int类型的返回一个P\_\_VF\_PC\_RI类型指针的外部函数fun2

（8）说明一个参数为P\_\_VF\_PC\_RI类型的返回一个P\_\_VF\_PC\_RI类型指针的函数fun3

（9）定义了一个名为VF\_PA\_P\_PF\_V的类型，该类型定义了一个无参函数，该函数返回一个指向另一个指针的指针，被指指针指向一个每个元素都存放一个函数指针的数组，其中存放的函数指针指向一个参数是const int类型且无返回值的函数

1. 会导致

（1）不能定义只与float g(int)有返回值不同的函数int g(int)

（2）g(8)调用时有歧义，无法确定是int g(int, int y=3)还是int g(int, …);

double max1(int *c*, ...)

{

    double maxn = -1;

    va\_list ap;

    va\_start(ap, *c*);

    maxn = va\_arg(ap, double);

    for (int i = 1; i < *c*; i++)

    {

        float temp = va\_arg(ap, double);

        if (maxn < temp)

            maxn = temp;

    }

    return maxn;

}

char \*numConvert(int *desBase*, int *srcBase*, int *num*)

{

    char \*s = (char \*)malloc(sizeof(char));

    int dec = 0, len = 0;

    for (int i = 0; num != 0; i++)

    {

        dec += (num % 10) \* pow(srcBase, i);

        num /= 10;

    }

    for (int left = 1; dec != 0;)

    {

        left = dec % desBase;

        dec /= desBase;

        s = (char \*)realloc(s, sizeof(char) \* (len + 1));

        s[len++] = (left < 9 ? left + '0' : left - 10 + 'A');

    }

    s[len] = '\0';

    strrev(s);

    return s;

}

6.

SET::SET(int *card*)

{

    if (set = new int[*card*])

        SET::card = *card*;

    uesd = 0;

}

SET::~SET()

{

    if (set)

    {

        delete set;

        set = nullptr;

        card = 0;

        used = 0;

    }

}

int SET::size()

{

    return used;

}

int SET::insert(int *v*)

{

    if (used == card)

        return 0;

    else

    {

        set[used++] = *v*;

        return 1;

    }

}

int SET::remove(int *v*)

{

    if (used > 0)

    {

        for (int i = 0; i < used; i++)

        {

            if (set[i] == *v*)

            {

                used--;

                for (; i < used; i++)

                    set[i] = set[i + 1];

                return 1;

            }

        }

        return 0;

    }

    return 0;

}

int SET::has(int *v*)

{

    for (int i = 0; i < used; i++)

    {

        if (set[i] == *v*)

        {

            return 1;

        }

    }

    return 0;

}

7.

NODE::NODE(char \*)

{

    if (NODE::data = new char[strlen(data) + 1])

    {

        strcpy(NODE::data, data);

        left = right = nullptr;

    }

}

NODE::NODE(char \**data*, NODE \**left*, NODE \**right*)

{

    if (NODE::data = new char[strlen(*data*) + 1])

    {

        strcpy(NODE::data, *data*);

        NODE::left = *left*;

        NODE::right == *right*;

    }

}

NODE::~NODE(){

    if(left)

        left->~NODE();

    if(right)

        right->~NODE();

    if(data){

        delete data;

        data = nullptr;

    }

}

8.

INTLIST::INTLIST(int *s*)

{

    if (list == new int[*s*])

    {

        size = *s*;

        used = 0;

    }

}

INTLIST::~INTLIST(void)

{

    if (list)

    {

        delete list;

        list = nullptr;

        size = 0;

        used = 0;

    }

}

int INTLIST::insert(int *v*)

{

    if (used < size)

    {

        list[used++] = *v*;

        return 1;

    }

    return 1;

}

int INTLIST::remove(int *v*)

{

    if (used)

    {

        for (int i = 0; i < used; i++)

        {

            if (*v* == list[i])

            {

                used--;

                for (; i < used; i++)

                    list[i] = list[i + 1];

                return 1;

            }

        }

        return 0;

    }

    return 0;

}

int INTLIST::find(int *v*)

{

    for (int i = 0; i < used; i++)

    {

        if (*v* == list[i])

        {

            return 1;

        }

    }

}

int INTLIST::get(int *k*)

{

    if (used >= *k*)

        return list[*k* - 1];

    return 0;

}