1. 若给出声明：

```
char c, *pc;
const char cc = 'a';
const char *pcc;
char *const cpc = &c;
const char * const cpcc = &cc;
char *const *pcpc;
```

则下面的赋值哪些是合法的？哪些是非法的？为什么？

```
(1) c = cc;              (10) *pc = "ABCD"[2];
(2) cc = c;              (11) cc = 'a';
(3) pcc = &c;            (12) *cpc = *pc;
(4) pcc = &cc;           (13) pc = *pcpc;
(5) pc = &c;             (14) **pcpc = *pc;
(6) pc = &cc;            (15) *pc = **pcpc;
(7) pc = pcc;            (16) *pcc = 'b';
(8) pc = cpcc;           (17) *pcpc = 'c';
(9) cpc = pc;            (18) *cpcc = 'd';
```

参考答案：

(1) 合法，c不是const类型的变量，可以赋值。

(2) 非法，cc是const类型的变量，不能赋值。

(3) 合法，pcc不是const类型的指针变量，可以指向非const类型的字符变量。

(4) 合法，pcc不是const类型的变量，赋值后指向的cc的类型为const char，同其要求指向的类型一致。

(5) 合法，pc 不是 const 类型的指针变量，赋值后指向的 c 的类型为 char，同其要求指向的类型一致。

(6) 非法，pc要求指向char类型的变量，不能用const char*类型的&c赋值。

(7) 非法，pc要求指向char类型的变量，不能用指向const char*类型的pcc赋值。

(8) 非法，pc要求指向char类型的变量，不能用指向const char*类型的cpcc赋值。

(9) 非法，cpc是const类型的变量，不能赋值。

(10) 合法，pc指向的是非const类型的变量，可以赋值，等价于*pc='C'。

(11) 非法，cc是const类型的变量，不能赋值。

(12) 合法，cpc指向的是非const类型的变量，可以赋值。

(13) 合法，pc是非const类型的指针变量，可以用char *类型的值*pcpc赋值。

(14) 合法，**pcpc代表的是非const类型的字符变量，可以任何字符类型的值赋值。

(15) 合法，*pc代表的是非const类型的字符变量，可以任何字符类型的值赋值。

(16) 非法，*pcc代表的字符是const类型的字符变量，不能赋值。

(17) 非法，*pcpc代表的是const类型的指针变量，不能赋值。

(18) 非法，*cpcc 代表的是 const 类型的只读变量，不能赋值。

2. C按优先级和结合性解释类型，下述声明是什么意思?

(1) typedef void VF_PC_RI(char*, int &)

(2) typedef VF_PC_RI* P_VF_PC_RI;

(3) typedef int &RIFFII(int, int);

(4) extern VF_PC_RI funca;

(5) extern P_VF_PC_RI ptra;

(6) extern void func1 (P_VF_PC_RI *);

(7) extern P_VF_PC_RI func2 (int c);

(8) P_VF_PC_RI func3 (P_VF_PC_RI a);

(9) typedef void (*(**VF_PA_P_PF_V(void))[ ]) (const int);

参考答案：

(1) 定义一个名为VF_PC_RI的类型，该类型定义了一个参数为(char*, int &)没有返回值的函数。

(2) 定义一个名为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) 定义一个名为RIFFII的类型，该类型定义了一个参数为(int ,int)返回一个引用的函数，该引用引用一个整型量。

(4) 说明一个类型为 VF_PC_RI 的函数 funca。

(5) 说明一个类型为P_VF_PC_RI的指针变量ptra。

(6) 说明一个没有返回值的函数func1，该函数的参数是一个指向P_VF_PC_RI类型的指针。

(7) 说明一个参数为int类型的函数func2，该函数的返回值是一个P_VF_PC_RI类型的指针。

(8) 说明一个参数为P_VF_PC_RI类型的函数func3，该函数的返回值是一个P_VF_PC_RI类型的指针。

(9) 定义一个名为 VF_PA_P_PF_V 的类型，该类型定义了一个没有参数的函数，该函数返回一个指针，该指针又指向另一个指针，被指向的指针指向一个数组，数组的每个元素存放一个函数指针，该函数指针指向一个参数为 const int 类型没有返回值的函数。

3. 下面g()函数的重载声明和定义是否会导致编译错误？

float g(int);

int g(int);

int g(int, int y=3);

int g(int, …);

int i = g(8);

参考答案：

(1) 不能定义返回类型仅和 float g(int)不同的函数 int g(int) 。

(2) g(8)在调用时出现二义性，无法确定是调用 int g(int, int y=3)还是 int g(int, …)。

4. 定义函数求 n（n>=1）个 double 类型的数的最大值 double max1(int n, …)。

参考答案：

```
double max1(int n, …) {
   double *p = (double *)(&n + 1);
   double v = p[0];
   for(int k = 1; k < n; k++) {
      if( v < p[k] ) v = p[k];
   }
   return v;
}
```

5. 编写函数 char *numConvert(int desBase, int srcBase, int num)，将 srcBase (2 <= srcBase <= 16) 进制数 num 转换为 desBase (2 <= desBase <= 16) 进制的字符串，返回字符串的首地址。

调用语句示例： char *p = numConvert(2, 8, 365); //将 8 进制数 365 转换为 2 进制字符串
printf(“%s \n”, p);

参考答案：

```
char *numConvert(int desBase, int srcBase, int num)
{
    char buf[100];
    int k = 0;
    /**/
    //将 srcNum 进制数 num 转换为 10 进制数 => num
    while(num != 0) {
        buf[k++] = num % 10;
        num /= 10;
    }
    while(--k >= 0) {
        num = num * srcBase + buf[k];
    }
    /**/
    //将 10 进制数 num 转换为 desNum 进制字符串
    k = 0;
    while(num != 0) {
        buf[k++] = num % desBase;
        num /= desBase;
    }
    char *p = new char [k+1];
    p[k--] = 0;
    for(int i = 0; k >= 0; i++, k--) {
        p[i] = buf[k];
        if(p[i] < 10) p[i] += '0';
        else p[i] = p[i] - 10 + 'A';
    }
    return p;
}
```

6. 集合类的头文件 set.h 如下，请定义其中的函数成员。

```
class SET {
    int *set;              //set 用于存放集合元素
    int card;              //card 为能够存放的元素个数
    int used;              //used 为已经存放的元素个数
public:
    SET(int card);         //card 为能够存放的元素个数
    ~SET( );
    int size( );           //返回集合已经存放的元素个数
    int insert(int v);     //插入 v 成功时返回 1，否则返回 0
    int remove(int v);     //删除 v 成功时返回 1，否则返回 0
    int has(int v);        //元素 v 存在时返回 1，否则返回 0
};
```

参考答案：

```
SET::SET(int card) {
    if (set = new int[card]) this->card = card;
    used = 0;
}

SET::~SET( ) {
    if( set ) {
       delete set;
       set = 0;
       card = used = 0;
    }
}
int SET::size( ) { return used; }
int SET::insert(int v) {
    if(used<card) { set[used++] = v; return 1;}
    return 0;
}
int SET::remove(int v) {
    int x;
    if( used > 0 ) {
       for(x = 0; x < used; x++) {
          if(set[x] == v) {
             used--;
             for(; x < used; x++) set[x] = set[x+1];
             return 1;
          }
          return 0;
       }
    }
    return 0;
}
```

```
int SET::has(int v) {
  for(int x = 0; x < used; x++) if(set[x] == v) return 1;
  return 0;
}
```

7. 二叉树类的头文件 node.h 如下，请定义其中的函数成员。

```
class NODE {
    char      *data;
    NODE   *left, *right;
public:
    NODE(char *data);
    NODE(char *data, NODE *left, NODE *right);
    ~NODE( );
};
```

参考答案：

```
NODE::NODE(char *data) {
  if( this->data = new char[strlen(data)+1] ) {
      strcpy(this->data, data);
      left = right = 0;
  }
}

NODE::NODE(char *data, NODE *left, NODE *right) {
    if( this->data = new char[strlen(data)+1]) {
        strcpy(this->data, data);
        this->left = left;
        this->right = right;
    }
}

NODE::~NODE( ) {
    if(left) left->~NODE( );
    if(right) right->~NODE( );
    if(data) { delete data; data = 0; }
}
```

8. 线性表通常提供元素查找、插入和删除等功能。以下线性表是一个整型线性表，表元素存放在动态申请的内存中，请编程定义整型线性表的函数成员。

```
class INTLIST {
    int   *list;          //动态申请的内存的指针
    int   size;           //线性表能够存放的元素个数
    int   used;           //线性表已经存放的元素个数
public:
```

```
    INTLIST(int s);        //s 为线性表能够存放的元素个数
    int insert(int v);     //插入元素 v 成功时返回 1，否则返回 0
    int remove(int v);     //删除元素 v 成功时返回 1，否则返回 0
    int find(int v);       //查找元素 v 成功时返回 1，否则返回 0
    int get(int k);        //取表的第 k 个元素的值作为返回值
    ~INTLIST( );
};
```

参考答案：

```
INTLIST::INTLIST(int s) {
    if( list = new int[s] ) {
        size = s;
        used = 0;
    }
}

INTLIST::~INTLIST(void) {
    if( list ) { delete list; list=0; size = used = 0; }
}

int INTLIST::insert(int v) {
    if( used < size) {
       list[used++] = v;
       return 1;
    }
    return 0;
}

int INTLIST::remove(int v) {
    for(int i = 0; i < used; i++) {
       if( list[i] == v ) {
          used--;
          for (; i < used; i++) list[i] = list[i+1];
          return 1;
       }
    }
    return 0;
}

int INTLIST::find(int v) {
    for(int i = 0; i < used; i++) {
        if(list[i] == v) return 1;
    }
    return 0;
}
```