1. 类中的构造函数和析构函数可以重载吗？说明原因。

参考答案：

C++规定：类的所有实例成员函数的第 1 个参数为 this。

构造函数可以重载：构造函数除了第 1 个 this 参数外，还可以包含任意多个其他参数，因此构造函数可以重载。

析构函数不能重载：析构函数除了第 1 个 this 参数外，不能有其他任何参数，所以析构函数不能重载。

2. 分析下面程序的运行结果。

```
class A {
    int a;
public:
    A(int a): a(a) { printf("A%d",a); }
    A(): a(0) { printf("A%d",a); }
    ~A() { printf("~A%d",a); }
} a(1);

class B {
    int b;
    A a1,a2;
public:
    B(int b): a1(2),b(b) { printf("B%d",b); }
    ~B() { printf("~B%d",b); }
} b(9);

int main() { }
```

参考答案：

(1) 首先需要构造对象 a：调用构造函数 A(1)，显示“A1”；

(2) 其次需要构造对象 b：为了构造 b，首先需要构造成员对象 a1 和 a2，a1 使用 A(2)去构造，输出“A2”；a2 使用无参构造 A()，输出“A0”。构造完 a1 和 a2 后，再构造对象 b，b 使用 B(9)去构造，输出“B9”。

(3) 这样，构造完对象 a 和 b 后，输出“A1A2A0B9”。程序结束时，a 和 b 失去作用域，需要析构，析构是构造的逆顺序，即：析构 b => 析构 a2 => 析构 a1=> 析构 a，输出“~B9~A0~A2~A1”。

总的输出为：A1A2A0B9~B9~A0~A2~A1

3. 下面的类将 N 进制整数字符串 s 转换为 M 进制的字符串，请完成各个成员函数。测试程序为：int main() { NumString s("A4",16); printf("%s %s", s.toString(10), s.toString(2)); }

```
class NumString {
    const char *s;     //s[]为 N 进制整数字符串(如果其中有字母则必须大写),如"9AB6"(N=16)
    const int N;       //2 <= N <= 16
    int    decNum;     //在构造函数中,将 s[]转换为 10 进制数保存到该变量
public:
    NumString(const char *s, int N=10);   //构造函数, s[]中可能含有小写字母
    ~NumString( );          //析构函数
    char *toString(int M); //将 s[]中的 N 进制整数字符串转换为 M 进制数字符串,返回 M 进制数字符串的首地址
private:
    void toNum( );    //将 s[]中的 N 进制整数字符串转换为 10 进制数并保存到 decNum
};
```

参考答案：

```
NumString::NumString(const char *s,int N):s(new char[strlen(s)+1]),N(N),decNum(0)
{
    int k = 0;
    char *p = (char *)this->s;
    do{  //将 s[]中的字符拷贝到 this->s,同时将其中的小写字符转换为大写字母
        p[k] = (s[k]>='a'? s[k]-'a'+'A' : s[k]);
    } while(s[k++] != 0);
    toNum(); //将 this->s 中的 N 进制整数字符串转换为 10 进制数并保存到 decNum
}

NumString::~NumString()
{
    if(s) { delete [] s; s=NULL; }
}

//将 s 为 N 进制整数字符串转换为 10 进制数并返回该数
void NumString::toNum()
{
    int num = 0;
    for(int k = 0; s[k] != 0; k++)
    {
        int m = (s[k]>='A'? s[k]-'A'+10 : s[k]-'0');
        num = num * N + m;
    }
    decNum = num;
}

//将 s[]中的 N 进制整数转换为 M 进制的字符串,返回 M 进制的字符串的首地址
char *NumString::toString(int M)
{
    char buf[100];
```

```
    int k = 0;
    int kk = decNum;
    while(kk != 0)
    {
        buf[k++] = kk % M;
        kk /= M;
    }
    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;
}

int main( )
{
    NumString s("A4",16);
    printf("%s %s", s.toString(10), s.toString(2));
}
```

4. 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;
    }
```

5. 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;
}
```

6. 下面是二叉树类的定义，请完成各个成员函数。

```
class TREE {
    int    item; //节点的值
    TREE   *left, *right; //左、右子树
public:
    TREE(int item); //构造树：item 为根节点、左右子树为 null
    TREE(int item, TREE *left, TREE *right); //构造树：item 为根节点、左右子树为 left、right
    ~TREE( );
    int    getNodeNum( );  //返回节点总数
    int    getNodes(int items[ ]); //将所有的节点的值保存到items[ ]中
};
```

参考答案：

```
TREE::TREE(int item) {
      this->item = item;
      left = right = 0;
}

TREE::TREE(int item, TREE *left, TREE *right) {
      this->item = item;
      this->left = left;
      this->right = right;
}

TREE::~TREE( ) {
    if(left) left->~TREE( );
    if(right) right->~TREE( );
}

int TREE::getNodeNum( ) {
      int L = 0, R = 0;
      if(left) L = left->getNodeNum( );
      if(right) R = right->getNodeNum( );
      return L + R + 1;
}

int TREE::getNodes(int items[ ]) {
      int n = 0;
      if(left) n = left->getNodes(items);
      items[n++] = this->item;
      if(right) n += right->getNodes(items);
      return n;
}
```