1. 为什么类的非静态函数成员都含有隐含的 this 指针？

参考答案

对于一个类的非静态成员函数 f(…)，所有对象都共用 f()的代码（而不是每个对象都有独立的 f()）。因此，在调用 f()时，f()需要知道是谁（哪个对象）调用它，C++编译器用 this 表示这个对象的地址，并作为第一个参数传给 f()。因为任何类的所有非静态成员函数的第一个参数都是 this，所以 C++在设计非静态成员函数时隐含 this 参数。this 指针具有 const 属性。

2. 下面的程序在某些情况下会造成内存泄漏。请在程序中插入一些语句，使得不管在什么情况下，程序都没有内存泄漏的问题。

参考答案

```
struct    A {
      char *p = 0;
      int k = 0;
      A(const char *s): p(new char [strlen(s)+1]) { strcpy(p, s); }
      ~A( ) { if(p) { delete p; p = 0; }
};
char s[] = "12345";
A    x("Hello");
void    main() {
      const char *p = "12345"
      A    y(p);
      short sel = 0;
      scanf("%d", &sel);    //从键盘输入 0,1 或 2
      switch(sel) {
            case    0:
                        return;
            case    1:
                        y.~A( );
                        exit(1);
            default:
                        y.~A( );
                        x.~A( );
                        abort( );
      }
}
```

3. 假设有一个类A和一些语句：

```
struct  A {
  A( ) { … }
  … …
};
```

```
… …
int main( ) {
   A  a1, a2;
   float  x = a1 + 1;
   float  y = 1 + a1;
   float  z = a1 – a2 + 1;
   int  k = a1(“abc”, 1, 1.0);
}
```

分析这个程序缺少哪些函数？写出这些缺失函数的所有可能的原型（函数声明），并将 main( ) 函数翻译成函数调用的形式。

参考答案

缺少函数：operator+( ) 和 operator–( )

它们所有可能的原型：

```
float A::operator+(int);
float operator+(const A &, int);
float operator+(int, const A &);
float A::operator–(const A &);
float operator–(const A &, const A &);
int A::operator()(const char *, int, double);
```

上面的函数原型中，所有的参数 const A & 都可以更换为 const A。

将 mian( )函数翻译成函数调用的形式:

```
int main( ) {
   A a1, a2;
   float x = a1.operator+(1);  或者  float x = operator+(a1,1);
   float y = operator+(1, a1);
   float z = a1.operator–(a2) + 1;  或者  operator–(a1, a2) + 1;
   int  k = a1.operator()(“abc”, 1, 1.0);
}
```

4. 写出下面 main()函数中每条指令的执行结果。

```
struct A {
        int    i;
        A(int v) { i = v; printf("A(%d) ",i); }
        A(const A &a) { i = a.i; printf("A(A&) "); }
        ~A( ) { printf("~A(%d) ",i); }
        operator int() const { printf("int() "); return i; }
        A &operator=(const A &a) {
              printf("=() ");
              i = a.i;    return *this;
        }
};
int main(void)
```

```
{
    A x = 1;
    x = 1;
    A y = x;
    y = x;
    x = 1 + x;
    A z(x+y);
    printf("%d %d", y, (int)y);
}
```

参考答案

```
int main(void)
{
    A x = 1;                    //A(1)
    x = 1;                      //A(1), =(), ~A(1)
    A y = x;                    //A(A&)
    y = x;                      //=()
    x = 1 + x;                  //int(), A(2), =(), ~A(2)
    A z(x+y);                   //int(), int(), A(3)
    printf("%d   %d",y,(int)y); //1(y.i), 1(int())
}                               //~A(3), ~A(1), ~A(2)
```

5. 字符串类的类型声明如下：

```
#include <iostream>
#include <string>
class STRING {
    char *str;
public:
    int strlen( ) const;                        //返回字符串的长度
    int strcmp(const STRING &s) const;          //字符串比较
    STRING &strcpy(const STRING &s);            //深拷贝赋值
    STRING &strcat(const STRING &s);            //将字符串s尾加到当前字符串
    STRING &operator+=(const STRING &s);        //实现 strcat()的功能
    STRING(const char *s);
    ~STRING( );
};
void main(void)
{
    STRING s1("I like apple");
    STRING s2(" and pear");
    STRING s3(" and orange");
    std::cout << "Length of s1=" << s1.strlen( ) << "\n";
    s1.strcat(s2) += s3;
    std::cout << "Length of s1=" << s1.strlen( ) << "\n";
    s3.strcpy(s2).strcpy(s1);
```

```
    std::cout << "Length of s3=" << s3.strlen( ) << "\n";
}
```

试定义字符串复制及连接等函数成员（尽量调用 C 的字符串运算函数）。

参考答案

```
int STRING::strlen( )const{ return ::strlen(str); }
int STRING::strcmp(const STRING &s)const{return ::strcmp(str, s.str); }
STRING &STRING::strcat(const STRING &s) {
    int len = ::strlen(str) + ::strlen(s.str) + 1;
    char *t = str;
    if( str = new char[len] ) {
       ::strcat(::strcpy(str, t), s.str);
     }
     delete t;
    return *this;
}
STRING &STRING::operator+=(const STRING &s)
{
    this->strcat(s);
    return *this;
}
STRING &STRING::strcpy(const STRING &s) {
    int len = ::strlen(s.str) + 1;
    delete str;
    if( str = new char[len] ) ::strcpy(str, s.str);
    return *this;
}
STRING::STRING(char *s) { if(str = new char[::strlen(s)+1]) ::strcpy(str, s); }
STRING::~STRING( ) { if (str) { delete str; str=0; } }
```

6. 完成下面字典类的成员函数。

```
class DICT {
    char **const words;              //存放单词
    const int max;                   //字典可以存放单词的个数
    int pos;                         //当前可以存放单词的空闲位置
public:
    DICT(int max);                   //max 为最大单词个数
    DICT(const DICT &d);             //深拷贝构造
    DICT(DICT &&d) noexcept; //移动构造
    virtual ~DICT( ) noexcept;    //析构
    virtual DICT &operator=(const DICT &d);                 //深拷贝赋值
    virtual DICT &operator=(const DICT &&d) noexcept;    //移动赋值
    virtual int operator( )(const char *word) const;        //查找单词位置,-1 表示没找到
    virtual DICT &operator<<(const char *word);             //若字典中没有该单词则加入
    virtual DICT &operator>>(const char *word);             //删除字典中的这个单词,后面的单词往前移动
                                                            //字典中的单词保持连续存放
```

```
    virtual const char *operator[](int n) const;                    //取出第 n(n>=0)个单词
};
```

参考答案

```
//max 为最大单词个数
DICT::DICT(int max): words(new char *[max]),max(max)
{
    pos = 0;
    for(int k = 0; k < max; k++)
    {
        words[k] = 0;
    }
}

//深拷贝构造
DICT::DICT(const DICT &d): words(0),max(0)
{
    *this = d;
}

//移动构造
DICT::DICT(DICT &&d) noexcept: words(0),max(0)
{
    *this = (DICT &&)d;
}

DICT::~DICT( ) noexcept
{
    if( words )
    {
        for(int k = 0; k < max; k++)
        {
            if( words[k] )
            { delete [] words[k];
              words[k] = 0;
            }
        }
        delete [] words;
        *(char ***)&words = 0;
        *(int *)&max = 0;
    }
}

//深拷贝赋值
DICT &DICT::operator=(const DICT &d)
{
    this->~DICT();
    *(char ***)&words = new char *[d.max];
    *(int *)&max = d.max;
    pos = d.pos;
    for(int k = 0; k < pos; k++)
    {
```

```
            words[k] = new char [strlen(d.words[k])+1];
            strcpy(words[k],d.words[k]);
        }
        for(int k = pos; k < max; k++) words[k] = 0;
        return *this;
}

//移动赋值
DICT &DICT::operator=(const DICT &&d) noexcept
{
        this->~DICT();
        *(char ***)&words = d.words;
        *(int *)&max = d.max;
        pos = d.pos;
        *(char ***)&d.words = 0;
        *(int *)&d.max = 0;
        return *this;
}

//查找单词位置,-1 表示没找到
int DICT::operator( )(const char *word) const
{
        int k = 0;
        while(k<pos && strcmp(word,words[k])!=0) k++;
        return k==pos? -1 : k;
}

//若字典中没有该单词则加入
DICT &DICT::operator<<(const char *word)
{
        if(pos<max-1 && (*this)(word) == -1)
        {
            words[pos] = new char [strlen(word)+1];
            strcpy(words[pos],word);
            pos++;
        }
        return *this;
}

//删除字典中的这个单词,后面的单词往前移动(字典中的单词保持连续存放)
DICT &DICT::operator>>(const char *word)
{
        int k = (*this)(word);
        if( k >= 0 )
        {
            delete [] words[k];
            while(k < pos - 1)
            {
                words[k] = words[k+1];
                k++;
            }
            words[--pos] = 0;
```

```
    }
    return *this;
}

//取出第 n(n>=0)个单词
const char *DICT::operator[](int n) const
{
    return n>=0 && n<pos? words[n] : 0;
}
```

7. 分析 mian( )函数中每条语句的变量 i 的值。

```
int x = 1;
int y = ::x + 1;

struct A {
    int x;
    static int y;
    A &operator+=(A &a) { x += a.x; y += a.y; return *this; }
    operator int() { return x + y; }
    A(int x = ::x+1, int y = ::y + 10): x(x) { this->y = y; }
};
int A::y = 100;

int main() {
    A a(1,2), b(3), c;
    int i, *p = &A::y;
    i = b.x + b.y;
    i = *p;
    i = c;
    i = a + c;
    i = b += c;
    i = ((a+=c)=b)+10;
}
```

参考答案

	i	**::x**	**::y**	**A::y**	**a.x**	**b.x**	**c.x**
a(1, 2)		1	2	2	1		
b(3)				12		3	
c				12			2
i = b.x + b.y	15						
i = *p	12						
i = c	14						
i = a + c	27						
i = b += c	29			24		5	
i = ((a += c) = b) + 10	63			48 48	3 5		