1. 在 win32 x86 模式下，int *p; int **pp; double *q; 请说明 p、pp、q 个占几个字节的内存单元。

参考答案

不管是什么类型的指针，也不管是几重指针，它们存放的是一个物理地址。所以，p、pp、q 都是占 4 个字节的内存单元。

2. 常量 1、1.0、“1”的数据类型是什么？

参考答案

1	int
1.0	double
“1”	const char *

3. 语句：short int a[10]; short int *p = a; sizeof(a)等于 sizeof(p)吗？为什么？

参考答案

a 表示包含 10 个 short int 的缓冲区，p 变量存贮的是 a 的地址。

sizeof(a) = 20

sizeof(p) = 4 (win32 x86)

4. 类的构造函数和析构函数可以重载吗？为什么？

参考答案

构造函数：可以重载，因为构造函数的参数可以任意定义（除了 this）。

析构函数：不能重载，因为析构函数只能由 1 个 this 参数。

5. 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)
```

6. 字符串类的类型声明如下：

```
#include <string.h>
#include <iostream.h>
    class STRING {
    char *str;
public:
    int strlen( ) const;
    int strcmp(const STRING &s) const;
    STRING &strcpy(const STRING &s);
    STRING &strcat(const STRING &s);
    STRING(char *s);
    ~STRING( );
};
void main(void)
{
    STRING s1("I like apple");
    STRING s2(" and pear");
    STRING s3(" and orange");
    cout << "Length of s1=" << s1.strlen( ) << "\n";
    s1.strcat(s2).strcat(s3);
    cout << "Length of s1=" << s1.strlen( ) << "\n";
    s3.strcpy(s2).strcpy(s1);
```

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

试定义字符串复制及连接等函数成员，这些函数成员调用 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::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; } }
```

7. 完成下面类的成员函数。

```
class SEQUENCE;
class TREE {
    int    item;   //节点的值
    TREE *left, *right;
    friend SEQUENCE;
public:
    int getNodeNum( );          //返回节点总数
    int getNodes(int items[ ]);   //将所有的节点保存到items[ ]中
};
class SEQUENCE {
    int    *items;     //用于保存1个TREE中的所有节点
    int    size;       //items中元素的个数
 public:
    SEQUENCE(TREE &t);   //将t中的所有节点保存到items所指的缓冲区
};
```

参考答案

```
int TREE::getNodeNum( ) {
    int l = 0, r = 0;
    if(left) l = left->getnodes( );
```

```
        if(right) r = right->getnodes( );
        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;
    }
    SEQUENCE::SEQUENCE(TREE &t) {
        int m;
        size = t.getNodeNum( )
        items = new int[size];
        t.getNodes(items);
    }
```

8. 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;
}
```

9. 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		