1. 均占4个字节的内存单元
2. 1为int类型，1.0为double类型，“1”为char\*类型
3. 在win64模式下,sizeof(a)=20，sizeof(p)=8;前者为数组a所占空间的大小，后者为指针所占内存单元大小
4. 构造函数可以被重载，析构函数不可以被重载。因为构造函数可以有多个且可以带参数，而析构函数只能有一个且不能带参数
5. 执行A x = 1;后输出A(1)

执行x = 1;后输出A(1) =() ~A(1)

执行A y = x;后输出A(A&)

执行y = x;后输出=()

执行x = 1 + x;后输出int() A(2) =() ~A(2)

执行A z(x+y);后输出int() int() A(3)

执行printf("%d %d", y, (int)y);后输出int() A(A&) 6422028 1~A(1) ~A(3) ~A(1) ~A(2)

STRING::STRING(char \**s*)

{

    if (str = new char[::strlen(*s*) + 1])

        ::strcpy(str, *s*);

}

STRING::~STRING()

{

    if (str)

    {

        delete str;

        str = nullptr;

    }

}

int STRING::strlen() const

{

    return ::strlen(str);

}

int STRING::strcmp(const STRING &*s*) const

{

    return ::strcmp(str, *s*.str);

}

STRING &STRING::strcpy(const STRING &*s*)

{

    if (str)

    {

        delete str;

        str = nullptr;

    }

    if (str = new char[*s*.strlen() + 1])

        ::strcpy(str, *s*.str);

    return (\***this**);

}

STRING &STRING::strcat(const STRING &*s*)

{

    str = (char \*)realloc(str, sizeof(char) \* (::strlen(str) + ::strlen(*s*.str) + 1));

    ::strcat(str, *s*.str);

    return (\***this**);

}

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 order = 0;

    if (left)

        order = left->getNodes(*items*);

*items*[order++] = TREE::item;

    if (right)

        order = order + right->getNodes(*items*);

    return order;

}

SEQUENCE::SEQUENCE(TREE &*t*)

{

    items=new int[*t*.getNodeNum()];

*t*.getNodes(items);

}

DICT::DICT(int *max*) : words(new char \*[*max*]), max(*max*), pos(0) {}

DICT::DICT(const DICT &*d*) : words(new char \*[max]), max(max), pos(0)

{

    \*(const\_cast<int \*>(&max)) = *d*.max;

    \*(const\_cast<int \*>(&pos)) = *d*.pos;

    char \*\*\*temp = const\_cast<char \*\*\*>(&words);

    for (int i = 0; i < *d*.pos; i++)

        \*(\*temp + i) = *d*.words[i];

}

DICT::DICT(DICT &&*d*) noexcept : words(new char \*[max]), max(max), pos(0)

{

    pos = *d*.pos;

    \*(const\_cast<int \*>(&max)) = *d*.max;

    \*(const\_cast<char \*\*\*>(&words)) = *d*.words;

    \*(const\_cast<char \*\*\*>(&*d*.words)) = NULL;

    \*(const\_cast<int \*>(&*d*.max)) = 0;

*d*.pos = 0;

}

DICT::~DICT()

{

    if (words)

    {

        for (int i = 0; i < pos; i++)

            delete words[i];

        delete words;

        \*(const\_cast<char \*\*\*>(&words)) = 0;

    }

}

DICT &DICT::operator=(const DICT &*d*)

{

    \*(const\_cast<int \*>(&max)) = *d*.max;

    \*(const\_cast<int \*>(&pos)) = *d*.pos;

    char \*\*\*temp = const\_cast<char \*\*\*>(&words);

    for (int i = 0; i < *d*.pos; i++)

        \*(\*temp + i) = *d*.words[i];

}

DICT &DICT::operator=(const DICT &&*d*) noexcept

{

    pos = *d*.pos;

    \*(const\_cast<int \*>(&max)) = *d*.max;

    \*(const\_cast<char \*\*\*>(&words)) = *d*.words;

    \*(const\_cast<char \*\*\*>(&*d*.words)) = NULL;

    \*(const\_cast<int \*>(&*d*.max)) = 0;

    \*(const\_cast<int \*>(&*d*.pos)) = 0;

}

int DICT::operator()(const char \**word*) const

{

    for (int i = 0; i < pos; i++)

    {

        if (strcmp(\*(words + i), *word*) == 0)

            return i;

    }

    return -1;

}

DICT &DICT::operator<<(const char \**word*)

{

    if (**this**->operator()(*word*) == -1)

    {

        if (pos <= max)

        {

            char \*\*\*temp = const\_cast<char \*\*\*>(&words);

            \*(\*temp + pos) = (char \*)*word*;

            pos++;

        }

    }

}

DICT &DICT::operator<<(const char \**word*)

{

    int flag = **this**->operator()(*word*);

    if (flag != -1)

    {

        char \*\*\*temp = const\_cast<char \*\*\*>(&words);

        for (int i = flag; i < pos; i++)

        {

            \*(\*temp + i) = \*(\*temp + i + 1);

        }

        pos--;

    }

}

const char \*DICT::operator[](int *n*) const

{

    if (*n* < pos)

        return \*(words + *n*);

}

    i = b.x + b.y;

    //此时i=15

    i = \*p;

//此时i=12

i = c;

    //此时i=14

    i = a + c;

    //此时i=27

    i = b += c;

    //此时i=29

    i = ((a += c) = b) + 10;

    //此时i=63