名称粉碎规律

测试代码

测试平台为VS2017

```
// ?foo@@YAXXZ
void foo();

// ?foo@@YAXH@Z
void foo(int a);

// ?foo@@YAXHH@Z
void foo(int a, int b);

// ?foo@@YAXHN@Z
void foo(int a, double b);

// ?foo@@YAXHND@Z
void foo(int a, double b, char c);

// ?foo@@YAXHNDJ@Z
void foo(int a, double b, char c, long d)

// ?foo@@YAHN@Z
int foo(double a);

// ?foo@@YANNH@Z
double foo(double a, int b);

// ?foo@@YAFD@Z
short foo(char c);

// ?bar@@YGXXZ
void __stdcall bar();

// ?fc@@YIXXZ
void __fastcall fc();
```

名称粉碎

C++函数名称粉碎以`?函数名@@`开头，其后接着字母，分别是：

- 前三个是调用约定
  - `YA`是`__cdecl`
  - `YG`是`__stdcall`
  - `YI`是`__fastcall`

- 后面是返回值类型（后面的参数类型标识跟这里用的一样）：
  - `X`是`void`
  - `N`是`double`
  - `F`是`short`
  - `H`是`short`
- 而后是参数列表，分别是：
  - `H`是`int
  - `N`是`double`
  - `D`是`char`
  - `J`是`long`
- 最后是结尾标志

type、type *与type &

```
// ?foo@@YAHH@Z
int foo(int a);

// ?foo@@YAHAAH@Z
int foo(int &a);

// ?foo@@YAHPAH@Z
int foo(int *a);
```

在上述例子中，引用与指针在经过名称粉碎之后呈现不同的名字，所以可以构成重载。但是当调用时编译器不知道时该调用**int**类型的还是**int &**类型的函数（即具有二义性），所以编译不通过
若想调用**int &**类型的函数，则需要强转`static_cast<int(*)(int &)>(foo)(a);`

type *与const type *

```
// ?foo@@YAHPAH@Z
int foo(int *a);

// ?foo@@YAHPBH@Z
int foo(const int *a);
```

type ***与const type ***经名称粉碎后呈现不同的名字，所以可以构成重载

type与typedef type MYTYPE

```
// ?foo@@YAHH@Z
int foo(int a);

// ?foo@@YAHH@Z
typedef int MYINT;
MYINT foo(MYINT a);
```

type与**MYTYPE**经名称粉碎后呈现的名字一样，不能构成重载