链滴

extern "C

作者：localvar

原文链接：https://ld246.com/article/1522225483160

来源网站：链滴

原文

本以为很简单，仔细阅读了一下 C++ 标准，发现内容还不少。总结了一下。

要点：函数类型，函数名，变量名具有语言链接性，language linkage。

- 语言链接性可能会影响到名字以及调用约定等，由实现决定。
- C++ 默认的语言连接性是 C++ 语言链接性。
- 语言链接性仅作用于函数类型，函数名，变量名。
- 不同语言链接性的函数类型是不同的类型，即便其余的地方都相同。
- 语言链接性用链接性规格（linkage-specification）来声明，分有无大括号两种形式。

```
extern string-literal { declaration-seq opt }
extern string-literal declaration;
```

- 所有的实现都必须支持 "C" 和 "C++" 链接性。
- 链接性规格允许嵌套，此时最内层的那个起作用，但是并不建立作用域。因为不是作用域，B 可以到 A

```
extern "C" { extern "C++" { class A{}; } }
extern "C" { class B:A{};}
```

- 如果C链接性施加到C++类成员和成员函数的类型，忽略C链接性，但是其余的地方依然有效，比成员函数的参数。比如extern "C" { class A{ void f(void (*p)()){} }; }
  - A 不是函数和变量，没有语言链接性
  - f 是 C++ 成员函数，忽略所指定的 C 链接性，如果在外层没指定别的，就是
  - C++链接性。
  - p 具有 C 链接性。
- 除了C++链接性的函数外，同一个函数不带链接性规格的声明的函数不能早于带的。如果前面声明带链接性规格的形式，后面又出现了不带的形式，不受影响。
  - 比如 extern "C" void foo(); void foo(); 是可以的，反过来不行。
  - 但是 extern void foo(); extern "C++" void foo(); 可以。
- 特定名字的 C 语言链接性的函数最多只能有一个，即便放在不同的 namespace 里，也是同一个函，变量也是如此。这样的函数或变量也不能重复定义。两个 f 是同一个

```
namespace A { extern "C" void f(); }
namespace B { extern "C" void f(); }
```

- 有链接性规格的函数具有外部链接性。因此下面的代码是错误的。

```
extern "C" void f();
static void f();
```

- 链接性规格，带大括号的形式，不影响里面的声明是声明还是定义。单个声明的形式，视为 extern 限定符。

```
extern "C" int i;       // 是声明;
extern "C" { int i; }   // 是定义;
extern "C" { extern int i; } // 又是声明。
```

● 单个声明形式的连接性规格，不能再指定存储类别。

```
extern "C" static void f(); // error
```

● 因为语言链接性只跟链接性有关，因此 C 语言链接性的函数也可以有默认参数，函数的接口可以有用。

● 标准 C++ 库中的符号默认是 C++ 链接性。C++ 用的标准 C 库中的外部链接性符号可以是 C 或者 ++ 语言链接性，推荐后者。任何带有连续两个下划线的名字保留给实现用作同时具有 C 和 C++ 链性的名字。

● 标准 C 库中的任何函数签名都保留给实现用来做同时具有 C 和 C++ 链接性的名字。

● 因为不同语言连接性的相同签名的函数类型算作不同类型，所以可以相互重载，不算同一个函数。似的有两个原型的函数还有 bsearch，qsort 等带函数指针类型的参数的函数。

```
// commented by ctrlz
// 这里应该是ill-formed.
extern "C" int atexit(void (*f)(void))   // 这里的 f 是 C 链接性
extern "C++" int atexit(void (*f)(void)) // 这里的 f 是 C++ 链接性
```

● 以上都是 C++ 标准里的理论，落实到实践上，为了实现方便，很多编译器都把 C 和 C++ 链接性函数的类型视为相同的类型，此时的链接性仅仅影响生成的名字，或者还可能影响异常规格，比如把 xtern "C" 的函数默认视为 throw() 的等，并不影响调用约定。同样也为了实现的方便，C++ 所用的 C 库中的符号也都是 C 语言链接性的，因此这样的情况下，atexit, bsearch, qsort 等在这些编译器搭的库中也就只有一种原型了。这种情况下，同样签名的 C 和 C++ 链接性的函数类型就算做同种类型，下边的程序编译就会失败（VC 和 gcc 在这一点上都不符合标准）：

```
//commented by ctrlz.
//ill-formed
void f(void (*p)())
{
}
extern "C" void f(void (*q)())
{
}
```

● 链接性不应该影响函数的类型，就像普通函数的返回类型不影响函数的类型一样，从重载解析的角看，如果接受以上的行为，势必影响重载解析的过程。