{The Problem

```
inline std::string plain_demangle(char const *name){
   if (!name) return "unknown";
   char const *toBeFreed = abi::__cxa_demangle(name,0,0,0);
   std::string result(toBeFreed?toBeFreed:name);
   ::free(const_cast<char*>(toBeFreed));
   return result;
}
```

Leaks on Exception!

see also:
http://stackoverflow.com/questions/27440953/stdunique-ptr-for-c-functions-that-need-free/

1st Solution Attempt

```
inline std::string plain_demangle(char const *name){
   if (!name) return "unknown";
   std::unique_ptr<char const, decltype(&std::free)>
   toBeFreed { abi::__cxa_demangle(name,0,0,0), &std::free};
   std::string result(toBeFreed?toBeFreed:name);
   return result;
}
```

Doesn’t Compile
requires const_cast!

1st Solution Attempt

```
inline std::string plain_demangle(char const *name){
   if (!name) return "unknown";
   std::unique_ptr<char const, decltype(&std::free)>
   toBeFreed { abi::__cxa_demangle(name,0,0,0), &std::free};
   std::string result(toBeFreed?toBeFreed.get():name);
   return result;
}
```

```
/usr/local/include/c++/7.0.1/bits/unique_ptr.h:268:17: error: invalid conversion from 'const void*' to 'void*' [-fpermissive]
```

Doesn't Compile requires const_cast!

1st Solution Attempt

Size Overhead!

```
static_assert(sizeof(std::unique_ptr<char const,decltype(&std::free)>)==sizeof(char*),"");
// compile error!
```

extra Pointer!

Better Solution

```
struct free_deleter{
   template <typename T>
   void operator()(T *p) const {
      std::free(const_cast<std::remove_const_t<T>*>(p));
   }
};
template <typename T>
using unique_C_ptr=std::unique_ptr<T,free_deleter>;

inline std::string plain_demangle(char const *name){
   if (!name) return "unknown";
   unique_C_ptr<char const>
      toBeFreed {abi::__cxa_demangle(name,0,0,0)};
   std::string result(toBeFreed?toBeFreed.get():name);
   return result;
}
```

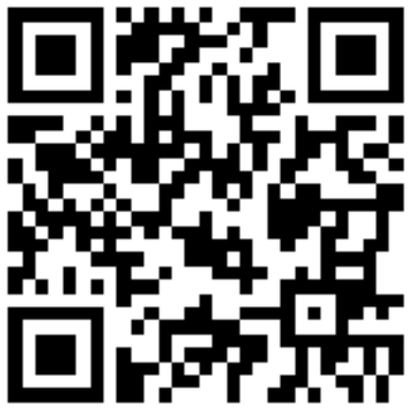
Better Solution

```
struct free_deleter{
   template <typename T>
   void operator()(T *p) const {
      std::free(const_cast<std::remove_const_t<T>*>(p));
   }
};
template <typename T>
using unique_C_ptr=std::unique_ptr<T,free_deleter>;

static_assert(sizeof(char *)==sizeof(unique_C_ptr<char>),"");
// compiles!
```

Space Efficient!

Wrap all your to-be-freed* C Pointers with unique_C_ptr for RAII



http://stackoverflow.com/questions/27440953/

}