Name Mangling

- The process of turning source program names of symbols into their object file names is known as name mangling (or name decoration).
- Reasons for name mangling
  - Avoid name collisions.
  - Overloading names.
  - Assisting link-time type checking.
  - Passing information from compiler to linker.
  - Enabling cross-language interoperability.
- Used in many popular languages, such as C++, Python, Java, and Rust.

Name Mangling in C++

- No standard name mangling scheme. Different compiler/OS combinations use different conventions.
  - This is actually encouraged by the Annotated C++ Reference Manual, so that incompatible libraries may be detected at link-time rather than at run-time.

Name Mangling in C++

- No standard name mangling scheme. Different compiler/OS combinations use different conventions.
  - This is actually encouraged by the Annotated C++ Reference Manual, so that incompatible libraries may be detected at link-time rather than at run-time.
- The function

```
int testfunc(char*, int, double, int, char, int*, float)
{ return 1; }
```

  may have its name mangled to `_Z8testfuncPcidicPif`.
  - `_Z` is a reserved identifier by C++ rules.
  - `8` is the number of characters in the function's name.
  - `testfunc` is the name of the function.
  - The argument types are indicated in order.

  On a different system, its mangled name may be `?testfunc@@YAHPADHNHDPAHM@Z`.

Name Mangling in C++

- No standard name mangling scheme. Different compiler/OS combinations use different conventions.
  - This is actually encouraged by the Annotated C++ Reference Manual, so that incompatible libraries may be detected at link-time rather than at run-time.
- The function

  ```
  int testfunc(char*, int, double, int, char, int*, float)
  { return 1; }
  ```

  may have its name mangled to `_Z8testfuncPcidicPif`.
  - `_Z` is a reserved identifier by C++ rules.
  - `8` is the number of characters in the function's name.
  - `testfunc` is the name of the function.
  - The argument types are indicated in order.

  On a different system, its mangled name may be `?testfunc@@YAHPADHNHDPAHM@Z`.
- Name mangling is disabled inside `extern "C"` blocks.

Name Mangling in Rust

- [Ref: The Rust Symbol Name Mangling Scheme, RFC #2603. https://rust-lang.github.io/rfcs/2603-rust-symbol-name-mangling-v0.html]
- Essential goal
  - The scheme must provide an unambiguous string encoding for everything that can end up in a binary's symbol table.
- Desirable properties
  - A mangled symbol should be *decodable* to some degree.
  - A mangling scheme should be platform-independent.
  - The scheme should be time- and space-efficient.
  - When used as part of a stable ABI, it should be possible to predict the symbol name for a given source-level construct.
- Non-goals
  - The mangling scheme does not try to be compatible with an existing (e.g. C++) mangling scheme.
  - The RFC does not try to define a standardized *demangled* form for symbol names.

An Example of Name Mangling in Rust

Given the declaration

```
mod foo {
    fn bar() {...}
}
```

located in a crate named `mycrate` with version `1234`, name mangling would produce the following unambiguous name.

```
_RNvNtCs1234_7mycrate3foo3bar
<>^^^^^<----><------><--><-->
 ||||||    |       |     |   |
 ||||||    |       |     |   +--- "bar" identifier
 ||||||    |       |     +------- "foo" identifier
 ||||||    |       +------------- "mycrate" identifier
 ||||||    +--------------------- disambiguator for "mycrate"
 |||||+---------------------------- start-tag for "mycrate"
 ||||+----------------------------- namespace tag for "foo"
 |||+------------------------------ start-tag for "foo"
 ||+------------------------------- namespace tag for "bar"
 |+-------------------------------- start-tag for "bar"
 +---------------------------------- common Rust symbol prefix
```