GNUv2 mangler grammar.

Source

- See PDF, section 8.4 (Gnu v2 name mangling)
- See rizinorg/rz-libdemangle issue#8

Attachments

PDF **LinkersAndLoaders.pdf** 3.68 MB

PDF **calling_conventions.pdf** 1.07 MB

The following content is copied and reformatted from calling_conventions.pdf above.

Notes

- While writing demangler for GNU v2, I realized that compilers following GNU v2 name mangling grammar, have some extra rules that are not mentioned in the calling_conventions.pdf file. These are special naming rules and were a bit confusing for the lack of grammar, but I ultimately made it work by taking reference from old demangler code and with the test cases.

GNU v2 Name Mangling

This mangling scheme is used in Gnu C++ version 2.x.x under several operating systems (Linux, BSD, Windows). Later versions of Gnu C++ use a different scheme described in the next section. The Gnu2 mangling scheme is a dialect of the scheme used by `cfront`, one of the oldest C++ tools. Variants of this scheme are widely used in UNIX systems (See: J. R. Levine: Linkers and Loaders. Morgan Kaufmann Publishers, 2000).

Global

The type of a global object is not coded, only class or namespace qualifiers, if any:

$$\langle public\ name \rangle ::= \left[\_\langle qualifiers\ list \rangle \langle list\ term \rangle\right] \langle name \rangle$$
$$\langle qualifiers\ list \rangle ::= [\langle qualifiers\ count \rangle] \left[\langle name\ length \rangle \langle class\ name \rangle\right]_1^\infty$$
$$\langle list\ term \rangle ::= \cdot \mid \$$$

- `<qualifiers count>` is the number of class or namespace qualifiers.
  - It is omitted if the count is `1`.
  - It is `Q<number>` if the number is `2 - 9`.
  - It is `Q_<number>_` if the number is more than `9`.
- All numbers are decimal.
- Some versions use `.` as list terminator (Red Hat), other versions use `$` (FreeBSD, Cygwin, Mingw32).
- The namespace `std` is ignored.

Example

- `char beta[6] = "Hello";` is coded as `beta`
- `char Namespace1::beta[6];` is coded as `_10Namespace1.beta` or `_10Namespace1$beta`

Functions & Member Functions

Functions and member functions are coded as follows

$$\langle public\ name \rangle ::= \langle name \rangle \_\_ \left[\langle qualifiers\ list \rangle \mid F\right] \left[\langle parameter\ type \rangle\right]_1^\infty$$

- The `<qualifiers list>` is replaced with an `F` if there are no class or namespace qualifiers.
- `<parameter type>` is the type of each function parameter, as defined by Table 9.
- The return type and function modifiers are not included.
- Types that occur more than once in the parameter list can be repeated according to the following rules :
  - Each parameter is assigned a number, beginning with `0`.
  - All parameters are numbered, regardless of whether they are identical to a previous parameter.
  - A repeated occurrence of a parameter is replaced by a reference to the first occurrence if it is a `pointer`, `reference`, `array` or other non-simple type.
  - `bool` is also treated as a non-simple type, while `long double` and `unsigned __int64` are treated as simple types.
  - A repeated occurrence of a non-simple parameter is replaced by `T <first occur>` where `<first occur>` is the number assigned to the first occurrence.
  - If the number is bigger than 9 then `<first occur>` is followed by an `_`.
  - A sequence of identical types can be replaced by `N <count> <first occur>` where `<count>` is the number of identical parameter types to replace and `<first occur>` is the number assigned to the first occurrence.
  - Both `<count>` and `<first occur>` are followed by an `_` if bigger than `9`.
  - For obscure reasons, the compiler uses the `T` replacement rather than the `N` replacement for the first parameter in a sequence of identical parameters if the preceding parameter is not the first occurrence of the same type.
  - There is no method for abbreviating repeated names that are not part of identical parameter types.

Example

`bool ExampleFunction (int* a, int b, int c, int* d, bool e, bool f, bool* g);` is coded as `ExampleFunction__FPiiiT0bT4Pb`

Constructors, Destructors & Operators

Constructors, destructors and operators are coded in the same way as functions with `<name>_` replaced by an operator name from Table 17.

Template Functions

Template functions are coded as follows :

$$\langle public\ name \rangle ::= \langle name \rangle \_\_H \langle numtp \rangle \left[Z\langle type\ parameter \rangle \mid \langle type \rangle \langle value \rangle\right]_1^{\langle numtp \rangle} \_ \left[\langle qualifiers\ list \rangle\right] \left[\langle parameter\ type \rangle \mid X\langle temp.par.num. \rangle\ 1\right]_1^\infty \_ \langle return\ type \rangle$$

where :

- `<numtp>` is the number of template parameters,
- `<type parameter>` is a template type parameter,
- `<value>` is a template constant parameter of type `<type>`.
- `<parameter type>` in the list of function parameters is replaced by `X <temp.par.num.> 1` if, and only if, it is explicitly declared as the same type as a template type parameter.
- `<temp.par.num.>` is the number of the template parameter referred to, starting at 0.
- The return type is included only for template functions.

Template Class

Template classes are coded by replacing `<class name length> <class name>` by

$$t\langle name\ length\rangle\langle name\rangle\langle numtp\rangle\,[Z\langle type\ parameter\rangle \mid \langle type\rangle\langle value\rangle]_{\langle numtp\rangle}^{\langle numtp\rangle}$$

Under Windows, all public names get an additional underscore prefix, for example `_ExampleFunction__FPiiiT0bT4Pb`. The `_` prefix is used only under Windows. It is omitted in Linux and FreeBSD, except possibly if the old `a.out` object file format is used.