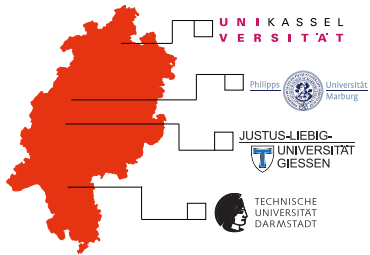
Excursion 2: Name Transformations

Hessisches Kompetenzzentrum für Hochleistungsrechnen (HKHLR)

Dr. Christian Iwainsky

V1.0



HKHLR is funded by the Hessian Ministry of Sciences and Arts

HESSEN

This segment's contents:

- C++ name mangling,
- C and Fortran name transformations,
- basic techniques for inspecting a programs symbols, and
- basic demangling process for C++ mangling.

- Function-, method, and subroutine names referred as symbols in compilation,
- For linking (and for dynamic libraries) each used symbol, be it function, method or subroutine, a matching implementation is required.
- Command line tool `nm` exploring function symbols in a binary or shared library.
- Different languages and compilers transform symbols during "compilation".
  - C++ compilers (GNU, Intel and other) heavily transform most symbols,
  - GCC comatible compilers canonize Fortran symbols.

C++ code example

```
1  #include <iostream>
2  using namespace std;
3  class myClass {
4    public:
5    double toDouble(int k){
6      return (double)k;
7    }
8  };
9  void aFunction(int arg){
10   cout << "Output" << arg
11     << endl;
12 }
13 int main(int c,char ** v){
14   aFunction(10);
15   return 0;
16 }
```

Symbol-list excerpt

```
1  _GLOBAL__sub_I__Z9aFunctioni
2  _Z41__static_initialization_and_destruction_0ii
3  _Z9aFunctioni
4  _ZN7myClass8toDoubleEi
5  _ZNSolsEPFRSoS_E
6  _ZNSolsEi
7  _ZNSt8ios_base4InitC1Ev
8  _ZNSt8ios_base4InitD1Ev
9  _ZSt4cout
10 _ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
11 _ZStL19piecewise_construct
12 _ZStL8__ioinit
13 _ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc
14 __cxa_atexit
15 __dso_handle
16 main
```

The symbol `main` is not transformed; called by program loader.

Demangling mangled C++ symbols:

- "`_Z9aFunctioni`":
  - `_Z`: A function or method,
  - `9aFunction`: Length of identifier and identifier, and
  - `i`: One integer argument.
- "`_ZN4demo10demoMethodEif`":
  - `_Z`: A function or method,
  - `N .. E`: Begin (`N`) and end (`E`) of fully qualified identifier,
  - `4demo`: Length and class name, namespace or identifier (demo),
  - `10demoMethod`: Length and class name, namespace or identifier (demoMethod), and
  - `if`: Two arguments: an integer and a float.

Command line tool `c++filt` decodes mangled symbols:

`c++filt _ZN4demo10demoMethodEif` $\longrightarrow$ `demo::demoMethod(int, float).`

"Fortran code example"

```
1  module arithmetic
2      public :: add
3      contains
4      subroutine add( a, b, result )
5          integer, intent( in )  :: a, b
6          integer, intent( out ) :: result
7          result = a + b
8      end subroutine add
9  end module arithmetic
10 subroutine aFunction(n)
11   integer, intent(in) :: n
12   print *, 'Some Output: ', n
13 end subroutine aFunction
14 program demo
15   call aFunction(10)
16 end program demo
```

"Symbol excerpt"

```
1  MAIN__
2  __arithmetic_MOD_add
3  _gfortran_set_args
4  _gfortran_set_options
5  _gfortran_st_write
6  _gfortran_st_write_done
7  _gfortran_transfer_character_write
8  _gfortran_transfer_integer_write
9  afunction_
10 main
11 options.1.3785
```

"C code example"

```c
#include <stdio.h>
void aFunction(int argument1)
{
  printf("Output %i\n",\
  argument1);
}

int main(int argc, char ** argv)
{
  aFunction(10);
  return 0;
}
```

"Symbol excerpt"

```
__libc_start_main@@GLIBC_2.2.5
_fini
_init
_start
aFunction
main
printf@@GLIBC_2.2.5
```

shell

```
nm demo.exe
```

Output in 3 columns:

1. Address of entity within file.
2. Symbol location / status:
   `U`: symbol without implementation; typically function from shared library.
   `T` or `t`: Symbol defined and implemented in file.
3. (Mangled) symbols: Symbols as stored in the file.

"Symbol excerpt"

```
1  0...04008e1 t \
     _Z41__static_initialization_and_destruction_0ii
2  0...0400876 T _Z9aFunctioni
3  0...0400934 W _ZN7myClass8toDoubleEi
4              U _ZNSt8ios_base4InitC1Ev@@GLIBCXX_3.4
5              U _ZNSt8ios_base4InitD1Ev@@GLIBCXX_3.4
6  0...0601060 B _ZSt4cout@@GLIBCXX_3.4
7  0...04009e8 r _ZStL19piecewise_construct
8  0...0601171 b _ZStL8__ioinit
9  0...0400790 T _start
10 0...0601170 b completed.7295
11 0...0601050 W data_start
12 0...04007d0 t deregister_tm_clones
13 0...0400870 t frame_dummy
14 0...04008b0 T main
```

shell

```
nm demo.exe | c++filt
```

"Symbol excerpt"

```
00000000004008e1 t __static_initialization_and_destruction_0(int, int)
0000000000400876 T aFunction(int)
0000000000400934 W myClass::toDouble(int)
                 U std::ios_base::Init::Init()@@GLIBCXX_3.4
                 U std::ios_base::Init::~Init()@@GLIBCXX_3.4
0000000000601060 B std::cout@@GLIBCXX_3.400000000004009e8 r std::piecewise_construct
0000000000601171 b std::__ioinit0000000000400790 T _start
0000000000601170 b completed.7295
0000000000601050 W data_start
00000000004007d0 t deregister_tm_clones
0000000000400870 t frame_dummy
00000000004008b0 T main
```

This segments contents:

- C++ name mangling,
- C and Fortran name transformations,
- basic techniques for inspecting a programs symbols, and
- basic demangling process for C++ mangling.
- Tools:
  - `nm`
  - `c++filt`