КОНСТРУИРОВАНИЕ ПРОГРАММ

Лекция № 14 Преобразование имен

+375 17 293 8039 (505a-5)

+375 17 320 7402 (ОИПИ НАНБ)

prep@lsi.bas-net.by

ftp://student:2ok*uK2@Rwox@lsi.bas-net.by/

Кафедра ЭВМ, 2022

2022.04.27

Оглавление

Преобразование имен (Name mangling)....3
Типичные стратегии....6
Спецификатор внешнего связывания (external linkage) "C"....11

Преобразование имен (Name mangling)

C++ поддерживает перегрузку функций (function overloading), то есть в одном контексте позволительно иметь несколько функций с одинаковыми именами и различиями в параметрах.

Чтобы отличить такие функции друг от друга при генерации объектного кода компилятор изменяет их имена, добавляя информацию об аргументах.

Такой метод добавления дополнительной информации к именам функций называется **преобразованием** или **декорированием имени** (Name Mangling). Пример:

```
// mangling.c++
class ABC {
public:
    void fun(long a, long b) {}
    void fun(float a, float b) {}
    void fun(int a, float b) {}
};

int main( ) {
   ABC obj;
   obj.fun(1l,2l);
   obj.fun(1,2.3f);
   obj.fun(3.2f,4.2f);
   return 0;
}
```

Скомпилируем программу

```
$ g++ mangling.c++ -o mangling
```

Данная программа имеет три функции с именем **fun**, отличающиеся количеством аргументов и их типами. Названия этих функций компилятором gcc 11.2.1 преобразуются, как показано ниже:

```
$ nm mangling |grep fun
0000000000401178 W _ZN3ABC3funEff
000000000040118e W _ZN3ABC3funEif
0000000000401164 W _ZN3ABC3funEll
```

ABC — общая строка для имени класса;

fun — общая строка для имени функции;

ff — два аргумента типа **float**;

ll — два аргумента типа **long**;

if — первый аргумент типа **int**, второй **float**.

```
$ nm --demangle mangling | grep fun #
0000000000401178 W ABC::fun(float, float)
000000000040118e W ABC::fun(int, float)
0000000000401164 W ABC::fun(long, long)
```

Декорирование имен решает проблему перегруженных идентификаторов в языках программирования. (Идентификатор называется «перегруженым», если одно и то же имя используется в нескольких контекстах или в нескольких значениях.)

В Ada или Modula-3 функция может появляться в нескольких модулях.

Если идентификатор экспортируется из отдельно скомпилированного модуля, ему необходимо иметь имя, под которым этот идентификатор будет известен компоновщику.

В связи с этим следует различать имена в исходном коде и имена времени компоновки.

Почему именно так?

В те времена, когда создавались редакторы связей (компоновщики), такие языки, как C, FORTAN и COBOL, не имели пространств имен, классов, членов классов и других подобных вещей. Поэтому в преобразовании имен не было необходимости.

Преобразование имен стало необходимым для поддержки объектно-ориентированных функциональных особенностей, которые редакторы связей (компоновщики) не в состоянии адекватно учесть и реализовать.

Из-за большого количества требований, которым вынуждены удовлетворять различные языки программирования, не существует простого решения на уровне редактора связей. Редакторы связей предназначены для работы с выходными данными (объектными модулями), генерируемыми различными компиляторами, и поэтому должны иметь универсальный способ поддержки имен

Стандарт языка C++ не определяет какой-либо конкретный метод декорирования имен, поэтому различные компиляторы к именам функций добавляют неодинаковую информацию.

Системная утилита **`nm`** печатает на **`stdout`** список символов, содержащихся в объектном файле (или исполняемом модуле). Опция **`--demangle`** печатает декорированные имена в том виде, как они выглядят в исходном файле. Это облегчает отладку.

Типичные стратегии

Каждый тип отображается на строку, которая объединяется с идентификатором и представляет имя во время компоновки.

Обычно в C++ перегрузка разрешена только для функций и только для типов аргументов.

В Ada аналогичным образом можно перегружать и типы результатов (строгая типизация с ограничением неявных преобразования типов — только целые и вещественные числового типа к совместимому числовому типу).

Если идентификатор используется более чем в одном модуле или пространстве имен, имя модуля объединяется с именем идентификатора, например, **`List_get`** вместо **`List.get`**.

В зависимости от того, какие символы являются допустимыми в именах времени компоновки, возможно, необходимо выполнить дополнительное декорирование. Например, может быть необходимо использовать подчеркивание в качестве символа «escape», чтобы можно было отличать

```
List_my.get -> List__my_get
```

от

```
List.my_get -> List_my__get
```

Вся причина и цель декорирования имен состоит в необходимости гарантировать, что различные идентификаторы в исходном коде будут отображены на различные имена времени компоновки.

Преобразование/декорирование имен — это процесс, с помощью которого компиляторы изменяют имена идентификаторов в исходном коде, чтобы помочь компоновщику в устранении неоднозначности между этими идентификаторами.

Это позволяет языку программирования гибко предоставлять одно и то же имя нескольким скомпилированным объектам и иметь согласованный способ поиска соответствующего объекта.

Например, это позволяет нескольким классам с одинаковыми именами существовать в разных пространствах имен, что обычно осуществляется путем добавления имени пространства имен к имени класса.

Перегрузка функций, операторов и методов во многих языках реализуется добавлением «декора», который зависит от количества и типов параметров, а в ряде случаев также и типа возврата — это позволяет в скомпилированной библиотеке существовать нескольким методам одного типа с одинаковым именем.

Нужно также отметить, что помимо количества параметров и их типов в процессе преобразования имен могут также участвовать соглашения о вызовах.

Преобразование имен — это процесс, используемый компиляторами C++, которые дают каждой функции в программе уникальное имя. В C++, как правило, программы имеют, как минимум, несколько функций с одинаковыми именами.

Пример. Обычно имена членов генерируются уникальным образом путем объединения имени члена с именем класса, например, с учетом декларации:

```
class Class1  {
        public:
            int val;
            ...
};
```

val становится примерно таким:

```
val__11Class1
```

В Фортране преобразование имени необходимо, потому что этот язык нечувствителен к регистру — **Foo**, **F00**, **f0o**, **foo** и прочие имена такого рода будут преобразованы в один и тот же символ, имя которого должно быть каким-то образом нормализовано.

Различные компиляторы реализуют преобразование по-разному, и это создает большие проблемы при взаимодействии с языком C или с двоичными объектами, скомпилированными с другим компилятором. Например, GNU g77/ g95 всегда добавляет завершающее подчеркивание к имени в нижнем регистре, но только если только это имя не содержит одно или несколько подчеркиваний. В этом случае добавляются два подчеркивания.

```
program test
end program

subroutine foo()
end subroutine
subroutine b_ar()
end subroutine
subroutine b_a_r()
end subroutine
```

Создает следующие декорированные символы:

```
0000000000400806 g     F .text  0000000000000006        b_ar__
0000000000400800 g     F .text  0000000000000006        foo_
000000000040080c g     F .text  0000000000000006        b_a_r__
```

Чтобы вызвать код на Фортране из кода на языке C, должно использоваться имя подпрограммы, преобразованное с учетом возможных различных стратегий декорирования.

Чтобы вызвать C-код из Фортрана, интерфейс, написанный на C, должен экспортировать правильно преобразованные имена и перенаправить вызов из фортрана в C-подпрограмму.

Спецификатор внешнего связывания (external linkage) "C"

Изменение имени нежелательно при связывании модулей C с библиотеками или объектными файлами, скомпилированными с помощью компилятора C++. Чтобы компилятор C++ не искажал имя функции, можно применить к объявлению или объявлениям спецификатор внешнего связывания (external linkage) **"C"**, как показано в следующем примере:

```
extern "C" {
   int f1(int);
   int f2(int);
   int f3(int);
};
```

Это объявление сообщает компилятору C++, что ссылки на функции **f1**, **f2** и **f3** не должны преобразовываться.

Спецификатор внешнего связывания **"C"** также можно использовать для предотвращения преобразования имен функций, определенных в C++, с тем чтобы их можно было вызывать из C. Например,

```
extern "C" {
   void p(int) {
      /* not mangled */
   }
};
```

В нескольких уровнях вложенности объявлений типа внешнего связывания преобладает внутренняя внешняя спецификация.

```
extern "C" {
    extern "C++" {
        void func();
    }
}
```

В этом примере **func** будет иметь тип связывания **C++**.

Спецификатор внешнего связывания **"C"** Используется во включаемых препроцессором файлах, которые включаются как в C, так и в C++.

```
#ifdef __cplusplus
    extern "C" {
#indif

// Здесь идут объявления, которые в C++ не должны декорироваться

#ifdef __cplusplus
}
#indif
```