КОНСТРУИРОВАНИЕ ПРОГРАММ

Лекция № 06.3 Внешние определения

Преподаватель: Поденок Леонид Петрович, 505а-5

+375 17 293 8039 (505а-5)

+375 17 320 7402 (ОИПИ НАНБ)

prep@lsi.bas-net.by

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

Кафедра ЭВМ, 2021

2021.10.13

Оглавление

Семантика

Объявление идентификатора определяет интерпретацию и атрибуты набора идентификаторов.

Определение идентификатора — это объявление этого же идентификатора, которое:

- для объекта выделяет для него область памяти;
- для функции включает тело функции;
- для константы перечисления является (единственным) объявлением идентификатора

```
enum color {
    red;
    green;
    blue;
};
```

- для имени типа (typedef-имени) является первым (или единственным) объявлением идентификатора

```
typedef unsigned int uint32_t;                                  // <stdint.h>

typedef void (*sighandler_t)(int);                              // <signal.h>
sighandler_t signal(int signum, sighandler_t handler);
```

Внешние определения

Единица программного текста (исходный файл) после обработки препроцессором является единицей трансляции, которая состоит из последовательности внешних объявлений.

Они характеризуются как «внешние», потому что они появляются вне какой-либо функции и, следовательно, имеют область видимости файла.

Объявление, которое к тому же вызывает резервирование памяти для объекта или функции, поименованной идентификатором, является определением.

Внешнее определение — это внешнее объявление, которое *одновременно* является определением либо функции, либо определением объекта.

Если идентификатор, объявленный с внешним типом связи, используется в выражении[1], где-то во всей программе должно быть ровно одно внешнее определение для идентификатора.

Если же такой идентификатор в выражениях не используется, внешних определений должно быть не больше одного (одно или может не быть). Иными словами, если идентификатор, объявленный с внешним связыванием, в выражениях не используется, для него не требуется никакого внешнего определения.

В спецификаторах внешних объявлений не должны появляться спецификаторы класса памяти **`auto`** и **`register`**.

В единице трансляции для каждого идентификатора, объявленного с внутренним связыванием, должно быть не более одного внешнего определения. Более того, если идентификатор, объявленный с внутренним связыванием, используется в выражении, в этой единице трансляции должно быть ровно одно внешнее определение для данного идентификатора.

---

1 кроме как часть операнда оператора sizeof или _Alignof, результатом которого является целочисленная константа

Определения функций

Синтаксис

определение-функции:

спецификации-объявления декларатор [список-объявлений] составной-оператор

список-объявлений:

декларатор

список-объявлений декларатор

декларатор:

идентификатор

декларатор **(** *список-типов-параметров* **)** // описан в объявлениях ф-ции

декларатор **(** *[список-идентификаторов]* **)** // описан в объявлениях ф-ции

составной-оператор: // описан в операторах

{ }

{ *список-членов-блока* **}**

Связка «*спецификации-объявления декларатор*» — это объявление функции.

Примеры

```
static int foo() {
    foo_body
}
```

```
static int foo(p1, p2)
int    p1;
double p2;
{
    foo_body
}
```

```
static int
foo(int p1, double p2) {
    foo_body
}
```

`static int` — спецификации-объявления (специф-ции класса памяти и типа);

`foo(...)` — декларатор

`int p1; double p2;` — список-объявлений для параметров

`{ foo_body }` — составной-оператор

Декларатор в определении функции указывает имя определяемой функции и идентификаторы ее параметров.

Если декларатор включает в себя список типов параметров, этот список должен указывать типы всех параметров. Такой декларатор также служит прототипом функции для последующих вызовов той же функции в том же модуле трансляции.

Если декларатор включает список идентификаторов, типы параметров должны быть объявлены в следующем за декларатором списке объявлений. В любом случае тип каждого параметра должен быть полным типом объекта.

Если функция, принимающая переменное число аргументов, определена без списка типов параметров, который заканчивается многоточием, поведение не определено.

Каждый параметр имеет автоматическую продолжительность хранения; его идентификатор является **lvalue**, т.е. его значение можно изменять.

Повторное объявление идентификатора параметра в теле функции допускается только во вложенном блоке.

Расположение параметров в памяти стандартом не определяется[2].

При входе в функцию вычисляются выражения размера каждого изменяемого параметра, а значение каждого выражения аргумента преобразуется в тип соответствующего параметра, как если бы он был присвоен оператором присваивания.

Выражения массивов и обозначения функций в качестве аргументов перед вызовом преобразуются в указатели.

После того, как всем параметрам присвоены значения, выполняется составной оператор, который составляет тело определения функции.

2 Расположение параметров в памяти определяется ABI

Ограничения

Идентификатор, объявленный в определении функции и который является именем функции, должен иметь тип функции, указанный в области деклараторов определения функции.

```
typedef int F(void);          // тип F -- ‘‘функция без параметров,
                              // возвращающая int’’
F f, g;                       // f и g имеют тип, совместимый с F
F f { /* ... */ }             // НЕВЕРНО: ошибка синтаксиса/ограничений
F g() { /* ... */ }           // НЕВЕРНО: объявляет, что g возвращает функцию
int f(void) { /* ... */ }     // ВЕРНО: f имеет тип, совместимый с F
int g() { /* ... */ }         // ВЕРНО: g имеет тип, совместимый с F
F *e(void) { /* ... */ }      // e возвращает указатель на функцию
F *((e))(void) { /* ... */ } // то же самое: скобки не имеют значения
int (*fp)(void);              // fp указывает на функцию, имеющую тип F
F *Fp;                        // Fp указывает на функцию, имеющую тип F
```

Тип, возвращаемый функцией, должен быть **void** или полным типом объекта, отличным от типа массива.

Спецификатор класса памяти, если таковой имеется, в спецификаторах объявления функции должен быть либо **extern**, либо **static**.

Идентификатор, объявленный как имя определения типа (typedef-имя), не должен быть повторно объявляться в качестве параметра.

Объявления в списке объявлений не должны содержать спецификатора класса памяти, кроме **register**, и никаких инициализаций.

Пример 1

```
extern int max(int a, int b) {
    return a > b ? a : b;
}
```

extern — спецификатор класса памяти;
int — спецификатор типа;
max(int a, int b) - декларатор функции;
{ return a > b ? a : b; } — это тело функции.

Следующее определение той же самой функции, но использующей для объявлений параметров форму со списком идентификаторов, имеет вид:

```
extern int max(a, b)
int a, b;
{
    return a > b ? a : b;
}
```

Здесь **int a, b;** — список объявлений для параметров.

Разница между этими двумя определениями заключается в том, что первая форма действует как объявление прототипа[3], которое вызывает преобразование аргументов последующих вызовов функции, тогда как вторая форма — нет.

3 Прототип функции - это объявление функции, которое объявляет типы ее параметров.

Пример 2

Чтобы передать одну функцию другой, можно сказать

```
int f(void);
/* ... */
g(f);
```

В этом случае определение **g** могло бы быть следующим:

```
void g(int (*funcp)(void)) {
    ...
    (*funcp)(); // или просто funcp(); ...
    ...
}
```

или, что эквивалентно,

```
void g(int func(void)) {
    ...
     func(); // or (*func)(); ...
    ...
}
```

Определение внешних объектов

Семантика

Если объявление идентификатора для объекта имеет область действия файла и инициализатор, объявление является внешним **определением** для идентификатора.

Объявление идентификатора для объекта, имеющего область видимости/действия файла без инициализатора и без спецификатора класса памяти, или со спецификатором класса памяти **`static`**, составляет *слабое определение*[4].

Если единица трансляции содержит одно или несколько *слабых* определений для идентификатора и эта единица трансляции не содержит для него внешнего определения, то поведение будет точно такое, как если бы единица трансляции содержала для этого идентификатора объявление с составным типом, с областью видимости файла, с инициализатором, равным 0.

Если объявление идентификатора для объекта является слабым определением и имеет внутренний тип связи, объявляемый тип не должен быть неполным типом.

4 tentative definition

Пример 1

```
int i1 = 1;         // определение, внешнее связывание
static int i2 = 2; // определение, внутреннее связывание
extern int i3 = 3; // определение, внешнее связывание
int i4;             // слабое определение, внешнее связывание
static int i5;     // слабое определение, внутреннее связывание

int i1; // допустимое слабое определение, ссылается на предыдущее
int i2; // 6.2.2 renders undefined, несогласованность связывания
int i3; // допустимое слабое определение, ссылается на предыдущее
int i4; // допустимое слабое определение, ссылается на предыдущее
int i5; // 6.2.2 renders undefined, несогласованность связывания

extern int i1; // ссылается на предыдущее, чей тип связывания внешний
extern int i2; // ссылается на предыдущее, чей тип связывания внутренний
extern int i3; // ссылается на предыдущее, чей тип связывания внешний
extern int i4; // ссылается на предыдущее, чей тип связывания внешний
extern int i5; // ссылается на предыдущее, чей тип связывания внутренний
```

Пример 2
Если в конце единицы трансляции, содержащей

```
int i[];
```

массив **i** все еще имеет неполный тип, неявный инициализатор заставляет его иметь один элемент, который устанавливается в ноль при запуске программы.