Многофайловые Си-программы

Лекция №3

Часть 2

Структура Си-программы

Программа на Си состоит из одного или нескольких файлов (текстовых). Содержание файлов программы:

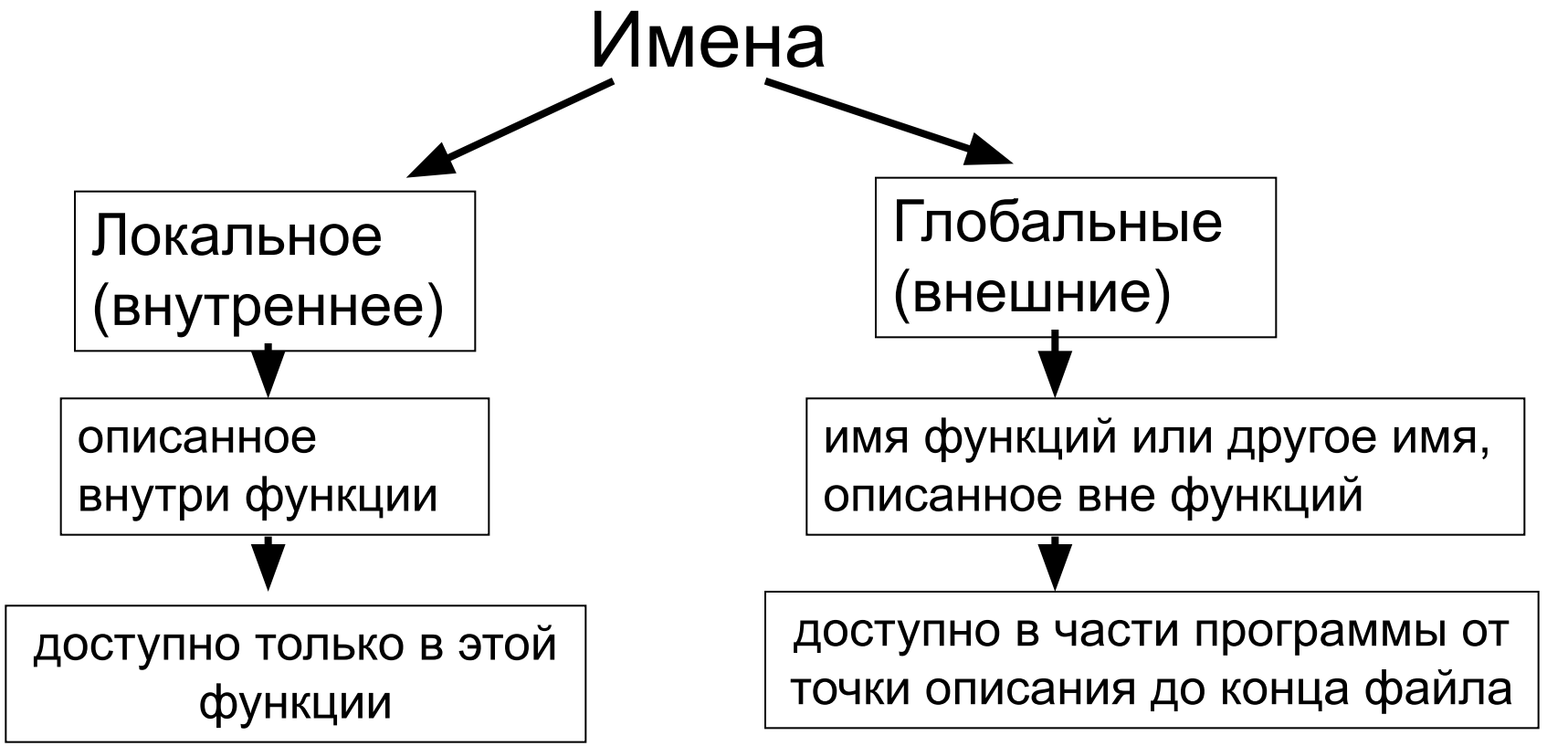
- описания функций, одна из которых обязательно **main**;
- прототипы (шаблоны) функций;
- директивы препроцессора;
- описания глобальных переменных, констант, типов (т. е. описания глобальных имен, отличных от функций).

Структура Си-программы

Язык "C" разрабатывался со стремлением сделать функции эффективными и удобными для использования; "C"-программы обычно состоят из большого числа маленьких функций, а не из нескольких больших. Программа может размещаться в одном или нескольких исходных файлах любым удобным образом; исходные файлы могут компилироваться отдельно и загружаться вместе наряду со скомпилированными ранее функциями из библиотек. Мы здесь не будем вдаваться в детали этого процесса, поскольку они зависят от используемой системы.

K&R

Область действия имен



Область действия имени – часть программы, в которой это имя может быть использовано

Глобальные (внешние) имена

1. **Имена функций.** Эти имена видимы из всех файлов программы. Однако шаблон функции действует только в пределе одного файла. Поэтому приходится помещать *в каждый файл* программы прототипы используемых в этом файле функций. Удобно это делать с помощью директивы **include** препроцессора.
2. **Имена переменных, констант, типов, объявленные вне функций.** Если требуется воспользоваться таким глобальным именем вне области его действия (в области от начала файла до объявления имени или в другом файле), то нужно повторить объявление имени, предварив его описателем **extern**. Объявление **extern** не предусматривает распределение памяти; оно лишь делает нужное имя доступным.
3. **Обратите внимание:**
    1. В инструкции **extern** невозможна инициализация.
    2. Компилятор Си по описаниям глобальных переменных не только дает им место в памяти, но и обнуляет их. Локальные переменные при описании не обнуляются; их значения считаются неопределенными (если, конечно, они не заданы инициализацией, вводом или присваиванием).
    3. Инструкции **extern** удобно вставлять с помощью **include**.

Пример: область действия имен

файл 1

```
 float a;
 int i;
 main ()
{ int i;
   extern float b;
   a=...; b=...;
}
float f1()
{int i; a=...
 }
float b;
a=...;
float f2()
{int i;a=...; b=...;
 }
```

файл 2

```
extern float a;
f3()
{a=...;
 }
extern int c;
f4()
{ a=...
}
f5()
{
}
int c; . . .
```

Пояснения к примеру

Глобальная переменная **a** может использоваться во всех функциях файла 1, т. к. она описана в самом начале файла 1; она также доступна всем функциям файла 2, потому что объявление ***extern float a*** стоит в начале файла 2.

Глобальная переменная **i** файла 1 недоступна ни одной функции этого файла, так как каждая функция имеет локальную переменную **i**. Описание ***int i*** приводит к выделению ячейки памяти под переменную i каждый раз при входе в блок {...},где стоит это описание; при выходе из блока эта ячейка освобождается. Локальные **i** доступны только в блоке своей функции, а глобальная **i** - во всем файле 1, за исключением этих функций.

Глобальная переменная **b** файла 1 может использоваться в функции f2, т. к. объявлена до описания f2. В функциях, описанных выше объявления **b**, эта переменная недоступна. Объявление ***extern float b*** в блоке функции **main**, позволяет этой функции использовать **b**; тем не менее, для f1 переменная **b** остается недоступной.

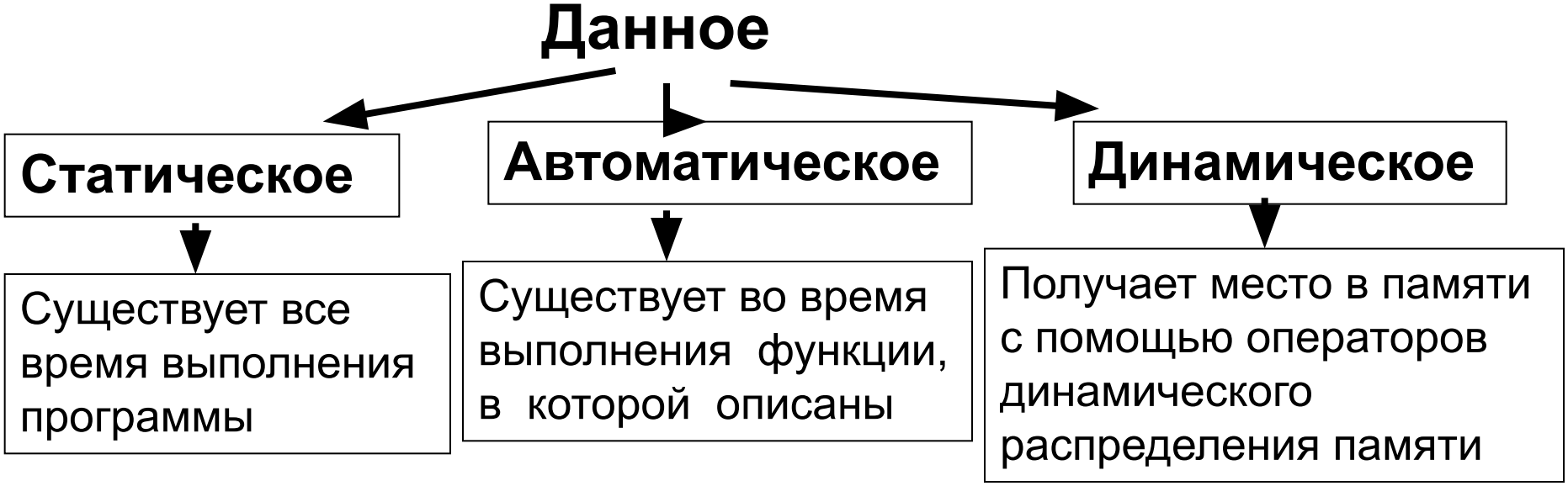
В файле 2 вместо двух объявлений ***int c*** и ***extern int c*** можно было бы оставить только ***int c***, поместив его на место ***extern int c***.

Рекомендации по использованию локальных и глобальных переменных

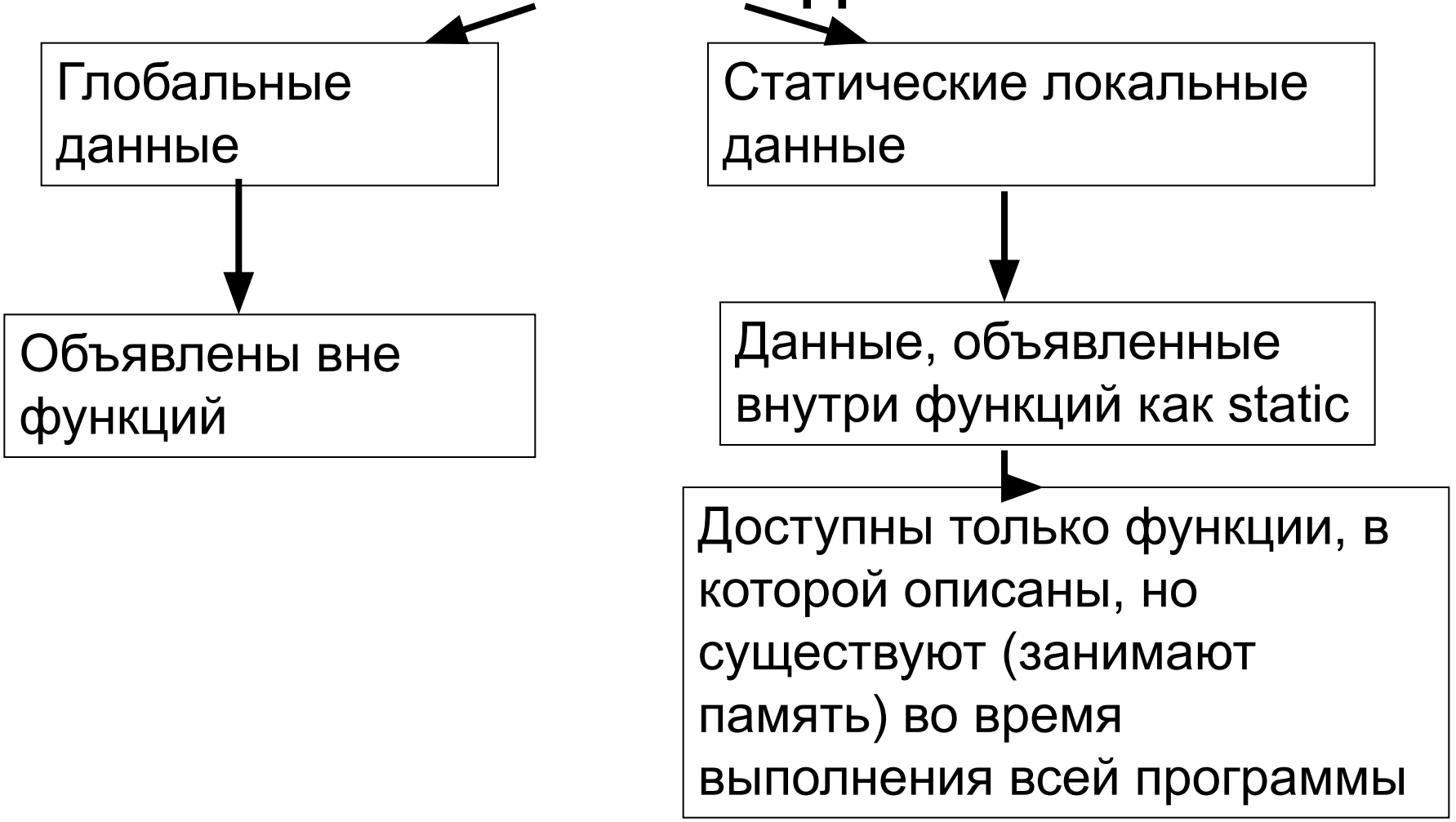
1. По возможности описания глобальных переменных, в том числе и ***extern***, следует ставить в начало файла.
2. Не стоит делать глобальными промежуточные переменные (например, **i** в программе, приведенной ниже), потому что:
   1) это делает подпрограмму менее универсальной, так как приводит к появлению непонятного пользователю "стыка" - промежуточной глобальной переменной;
   2) приводит к неэкономному расходованию памяти, так как глобальные переменные занимают память в течение всего времени работы программы.

Время жизни (существования) имени переменной или константы

Время жизни – время, в течение которого под данное распределена память.



Статические данные



Статические данные распределены в специальном ***статическом сегменте памяти*** *программы*

Автоматические данные

- Это, прежде всего, локальные данные функции, не объявленные как **static**.
- Под локальные данные, не объявленные как **static**, память распределяется в **стеке функций**.
- К автоматическим данным также относятся переменные типа **register**, которые хранятся во внутренних регистрах процессора. В нашем курсе эти переменные не рассматриваются.
- Память под автоматические данные распределяется при вызове функции и освобождается при завершении ее работы (передаче управления функции, вызвавшей данную) – они существуют пока работает функция.

Применение статических локальных данных

Статические локальные переменные часто используются программами управления ресурсами - например, для подсчета числа обращений к программе.

Пример использования локальной статической переменной:

```
f(); /*шаблон*/
main()
{...f();f();...;f();/*переменная i будет накапливать число обращений*/
}                    /* к f*/
f()
{static int i=0;
/*инициализация работает один раз при первом входе в блок */
 ... i=i+1;
}
```

Возможное расположение данных в памяти

- Статический сегмент программы
- Стек функций
- Динамическая память
- Регистры процессора

Класс памяти

Класс памяти характеризует область действия, время жизни и расположение в памяти переменных.

Существуют следующие **описатели класса памяти:**
auto - для переменных, действующих в пределах блока; обычно принимается по умолчанию.
register - тоже, что и auto, но для регистров процессора.
static - для описания статических переменных (имеет смысл для локальных переменных).
extern - делает доступными глобальные переменные, расширяя их область действия.

Характеристики классов памяти

Класс памяти	Время жизни	Область действия	Место в памяти
auto (обычно используется по умолчанию)	временно	функция	стек функций
register	временно	функция	регистры процессора
static	постоянно	функция	статический сегмент
extern	постоянно	файл, программа	статический сегмент