西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

C程序设计

Programming in C

1011014

主讲：姜学锋，计算机学院

设计函数 - 函数间的数据传递（2）

3、进程内存分配原理

4、对象的生命期

4.6.5 生命期

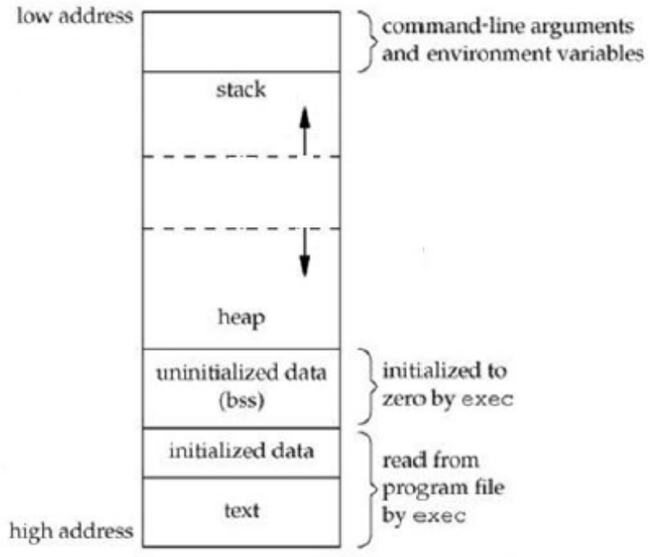
- 1．进程内存分配原理
- 变量是程序运行期间其值可改变的量，变量提供了程序可以操作的有名字的存储区。
- 实际上，C语言中的存储区有些是没有名字的，例如函数返回值，引入一个新的概念——对象（object）。

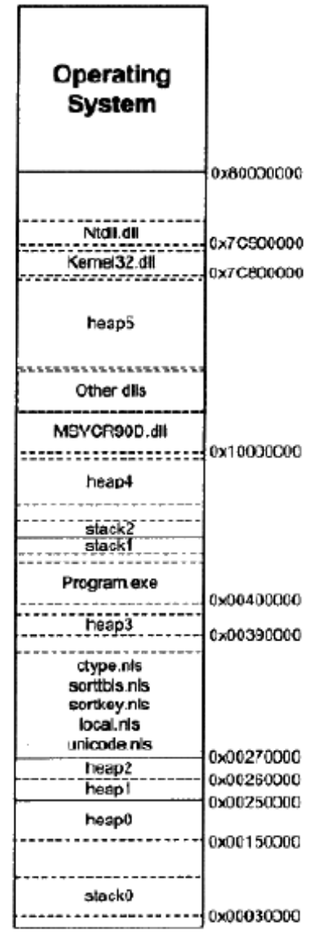
4.6.5 生命期

- 对象是内存中具有类型的存储区，可以使用对象来描述程序中可操作的大部分数据，而不管这些数据是基本类型还是构造类型，是有名字的还是没名字的，是可读写的还是只读的。一般而言，变量是有名字的对象。

4.6.5 生命期

- 进程内存分配原理
  - 从低地址到高地址分别为：
    - 代码段
    - 数据段（初始化）
    - 数据段（未初始化）（BSS）
    - 堆
    - 栈
    - 命令行参数和环境变量
  - 其中，堆向高内存地址生长。
  - 栈向低内存地址生长。





Windows Process Virtual Space

4.6.5 生命期

- 进程内存分配原理
  - 在进程的地址空间中：
    - 代码段中存放：全局常量（const）、字符串常量、函数以及编译时可决定的某些东西
    - 数据段（初始化）中存放：初始化的全局变量、初始化的静态变量（全局的和局部的）
    - 数据段（未初始化）（BSS）中存放：未初始化的全局变量、未初始化的静态变量（全局的和局部的）
    - 堆中存放：动态分配的区域（malloc、new等）
    - 栈中存放：局部变量（初始化以及未初始化的，但不包含静态变量）、局部常量（const）
    - 命令行参数和环境变量顾名思义存放命令行参数和环境变量

4.6.5 生命期

- C语言中，每个名字都有作用域，即可以使用名字的区域，而每个对象都有生命期（lifetimes），即在程序执行过程中对象存在的时间。

4.6.5 生命期

- 2．动态存储
- 动态存储（dynamic storage duration）是指在程序运行期间，系统为对象动态地分配存储空间。动态存储的特点是存储空间的分配和释放是动态的，要么由函数调用来自动分配释放，要么由程序指令来人工分配释放，在分配至释放之间就是对象的生命期，这个生命期是整个程序运行期的一部分。从前一节可知动态存储是在内存布局中的栈和堆两个段实现的。

4.6.5 生命期

- 动态存储的优点是对象不持久地占有存储空间，释放后让出空闲空间给其他对象的分配。程序中多数对象应该是动态存储方式的，因为程序的执行在一段时期往往局限于 部分对象，而不会是所有对象，这样未起作用的对象持久占有内存空间，只会导致内存越来越少，多进程、多任务无从谈起。

4.6.5 生命期

- 动态存储在分配和释放的形式有两种，一种是由函数调用来自动完成的，称为自动存储（automatic storage），一种是由程序员通过指令的方式来人工完成的，称为自由存储（free storage）。

4.6.5 生命期

- 自动存储的优点是程序员不用理会对象何时分配、分配多大、何时释放，其生命期完全与函数调用相同，自由存储的优点是生命期由程序员的意愿来决定，因此即使函数已经运行完也可以继续存在，直到释放指令发生。自由存储的缺点是程序员需要操心释放时机，多数情况下程序员会忘记释放，或者忘记已经释放。

- 自由存储只能通过指针来实现。

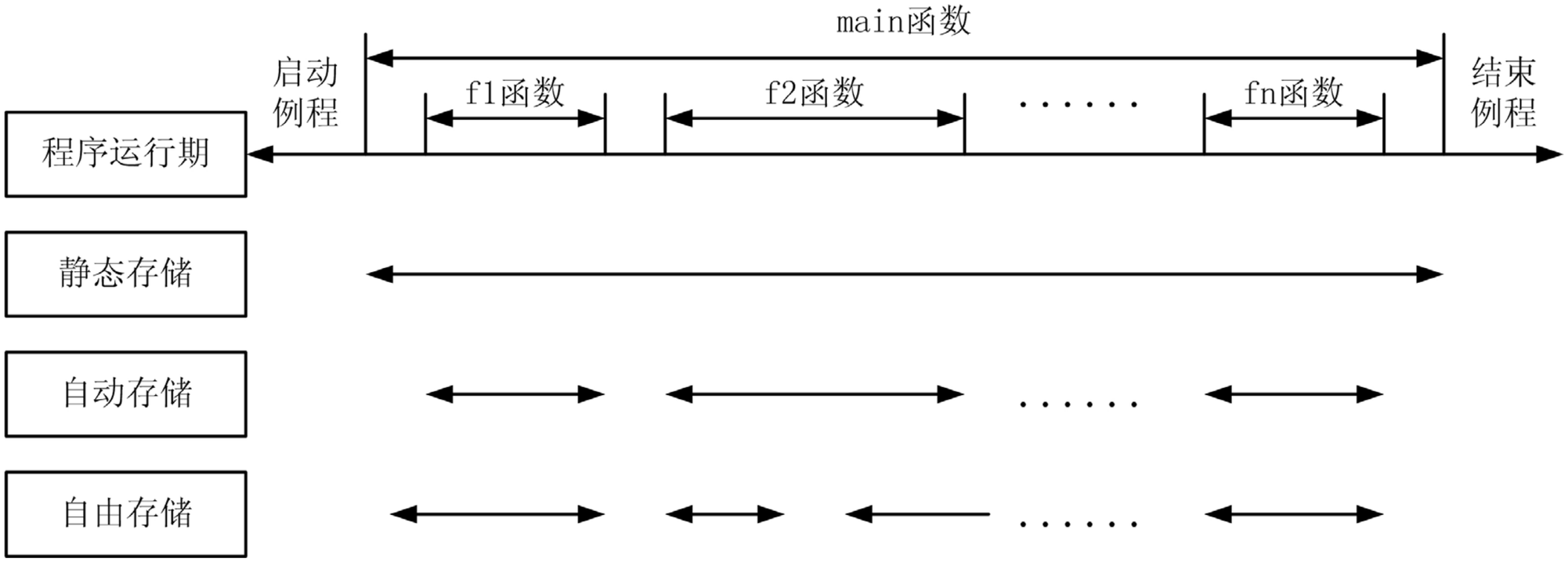
4.6.5 生命期

- 3．静态存储
- 静态存储（static storage duration）是指对象在整个程序运行期持久占有存储空间，其生命期与程序运行期相同。静态存储的特点是对象的数据可以在程序运行期始终保持直到修改为止，或者程序结束为止，静态存储的分配和释放在编译完成时就决定好了。从前一节可知静态存储是在内存布局中的数据段实现的。

- 现代程序设计的观点是，除非有必要尽量少地使用静态存储。

4.6.5 生命期

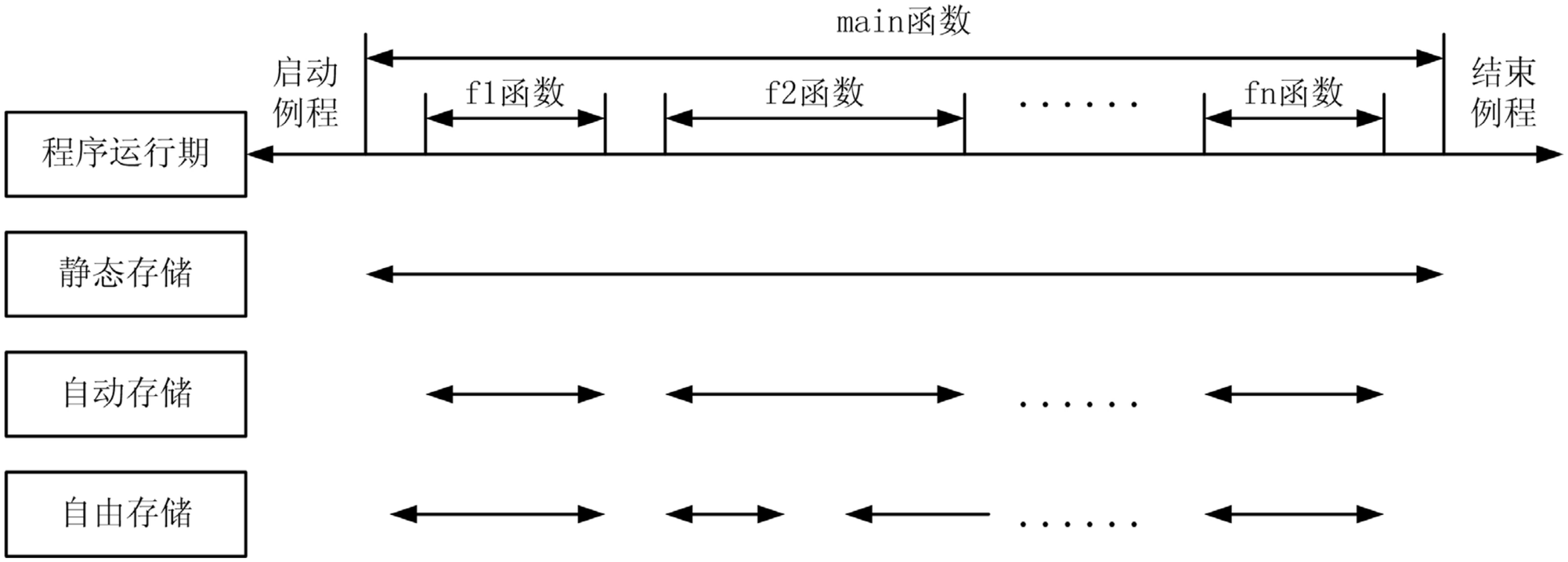
图4.10 存储类别的生命期



程序运行前由系统自动执行启动例程，做必要的初始化工作，程序运行后由系统自动执行结束例程，做必要的清理工作。

4.6.5 生命期

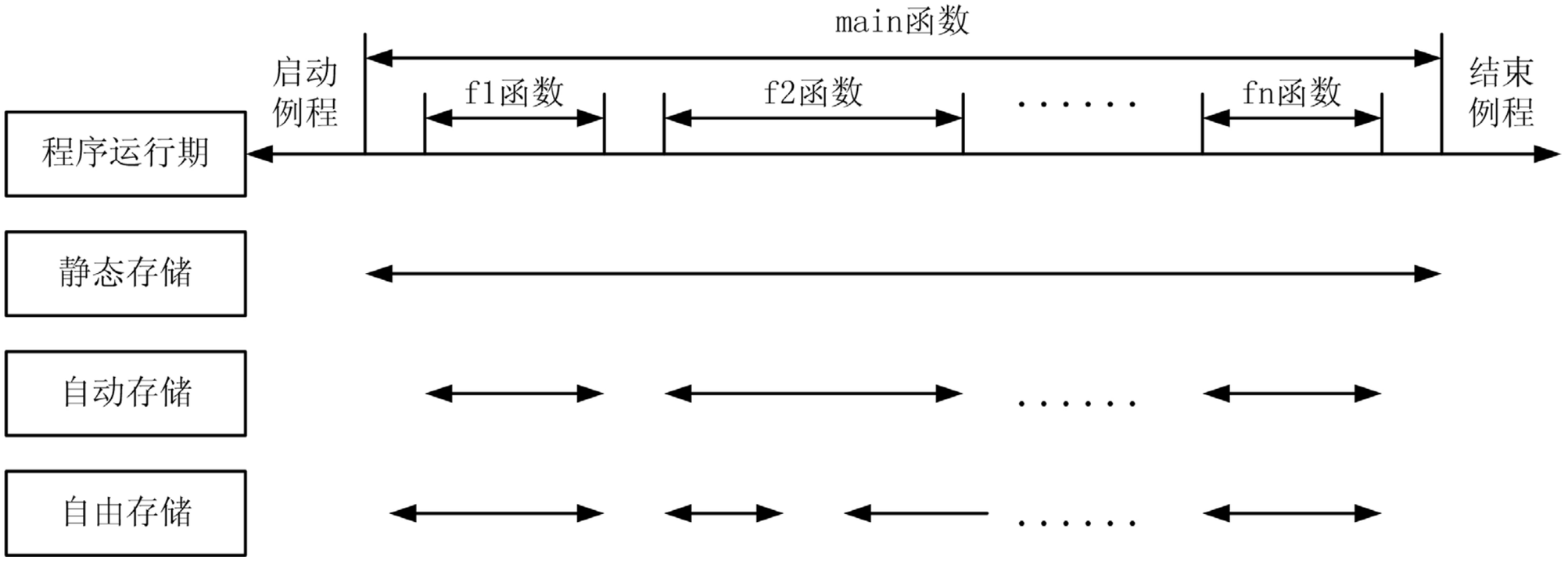
图4.10 存储类别的生命期



从图中可以看到静态存储生命期最长，自动存储随函数而定，自由存储既可以比函数调用期小，又可以跨多个函数甚至整个程序。

4.6.5 生命期

图4.10 存储类别的生命期



特殊的是main函数，它与程序运行期相同。

程序中所有的全局对象和静态局部对象都是静态存储的。

4.6.5 生命期

- 3．自动对象
- 默认情况下，函数或复合语句中的对象（包含形参）称为自动对象（automatic objects），其存储方式是自动存储，程序中大多数对象是自动存储。
- 自动对象进入函数时分配空间，结束函数时释放空间。

4.6.5 生命期

- 可以在对象的前面加上auto存储类别修饰来说明对象是自动对象，例如自动变量的语法形式为：

```
auto 类型 变量名[=初值], ......
```

- 它与不写存储类型修饰的结果是一样，即都是自动对象，换言之默认的存储类型修饰就是auto，例如：

```
①int a,b,c;
②auto int a,b,c;
```

- 是等价的。

4.6.5 生命期

- 4．寄存器变量
- C语言允许用CPU的寄存器来存放局部变量，称为寄存器变量。在局部变量前加上register存储类别修饰来定义的，其形式为：

```
register 类型 变量名[=初值], ......
```

4.6.5 生命期

- 寄存器变量不按前述的存储方式来描述，但可以将它理解为动态存储。由于CPU的寄存器数目是有限的，因而register并不总是每次定义就一定用寄存器来存放局部变量，当编译器不能使用寄存器时，则它会自动地将register修饰转换到auto修饰。

- 寄存器变量是早期计算机硬件限制下的产物，其目的是优化变量的存取速度；它在嵌入式计算、单片机、数字信号处理（DSP）等微处理环境中还有应用，但目前主流编译器都有变量优化处理，无需特别指明寄存器变量。

4.6.5 生命期

- 5．静态局部对象
- 在局部对象的前面加上static存储类别修饰用来指明对象是静态局部对象（static local object），一般形式为：

```
static 返回类型 函数名(形式参数列表)
{
  static 类型 变量名[=初值], ......
}
```

- 静态局部对象与全局对象一样是按静态存储处理的，即它的生命期与程序运行期相同，所以静态局部对象可以将其值一致保持到程序结束，或者下次修改时。

4.6.5 生命期

- 静态局部对象与全局对象有区别，它的作用域是函数作用域或者块作用域，即它只能在局部区域内使用。

4.6.5 生命期

```
1    int f1(int x,int y) // f1函数
2    {
3        int a,b=20;
4        static int m=100;
5        if (x>y) {
6            int a;
7            static int t;
                ⋮
10       }
          ⋮
20   }
```

x,y有效

a,b,m有效

a,t有效

4.6.5 生命期

- 静态局部对象在还未调用函数前，甚至是程序运行就进行初始化了。静态局部对象如果定义时未赋初值，那么编译器会自动将它的存储单元用0填充，对于数值型来说就是0这个值。

4.6.5 生命期

- 当一个函数会多次调用又希望将它的某些对象的值保持住时，就应该使用静态局部对象，但静态局部对象属于持久占有存储空间，所以应持谨慎和适度使用的原则。
- 需要注意，如果在全局对象定义前加上static关键字，不是静态的意思（全局对象本身已经是静态的），而是私有的意思，即此时的全局对象的作用域限定在全局对象所处的源文件中，其他文件不可见。

4.6.5 生命期

【例4.10】

计算函数调用次数。

4.6.5 生命期

例4.10

```
1  #include <stdio.h>
2  int fun()
3  {
4    static int cnt=0; //静态局部变量会保持其值
5    cnt++;
6    return cnt;
7  }
8  int main()
9  {
10   int i,c;
11   for (i=1;i<=10;i++) c=fun();
12   printf("%d\n",c);
13   return 0;
14 }
```

在main函数开始执行前，第4行的cnt赋初值0，其后不用再赋；

每次调用fun函数就执行第5行的cnt累加，所以cnt实际记录的是函数调用的次数。


程序设计