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）

1、程序映像

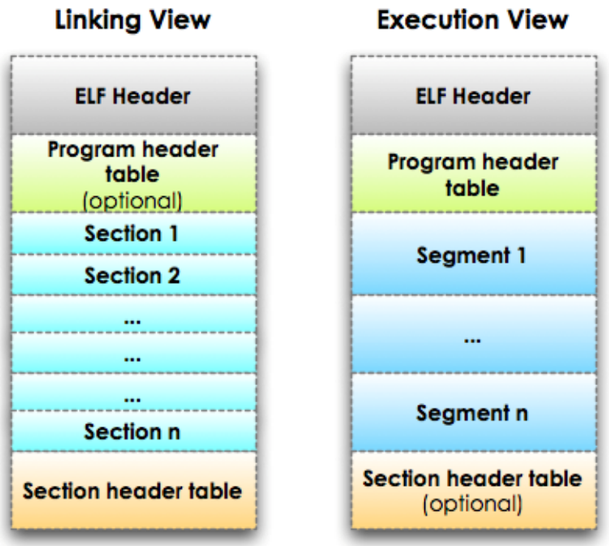
2、程序加载

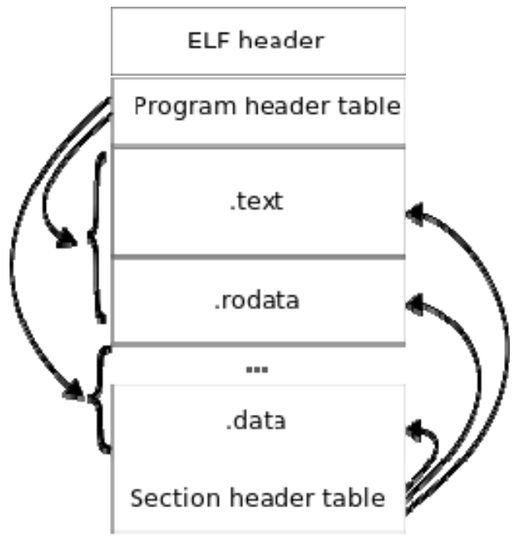
4.6.4 程序映像和内存布局

- 1．程序映像
- C源程序经过编译、连接后，成为二进制形式的可执行文件，称为程序映像。可执行文件采用ELF格式（可执行连接格式）存储。

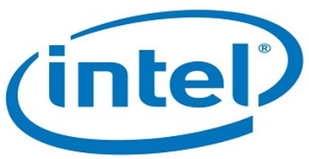
4.6.4 程序映像和内存布局

- 可执行连接格式（ELF，Executable and Linkable Format）。





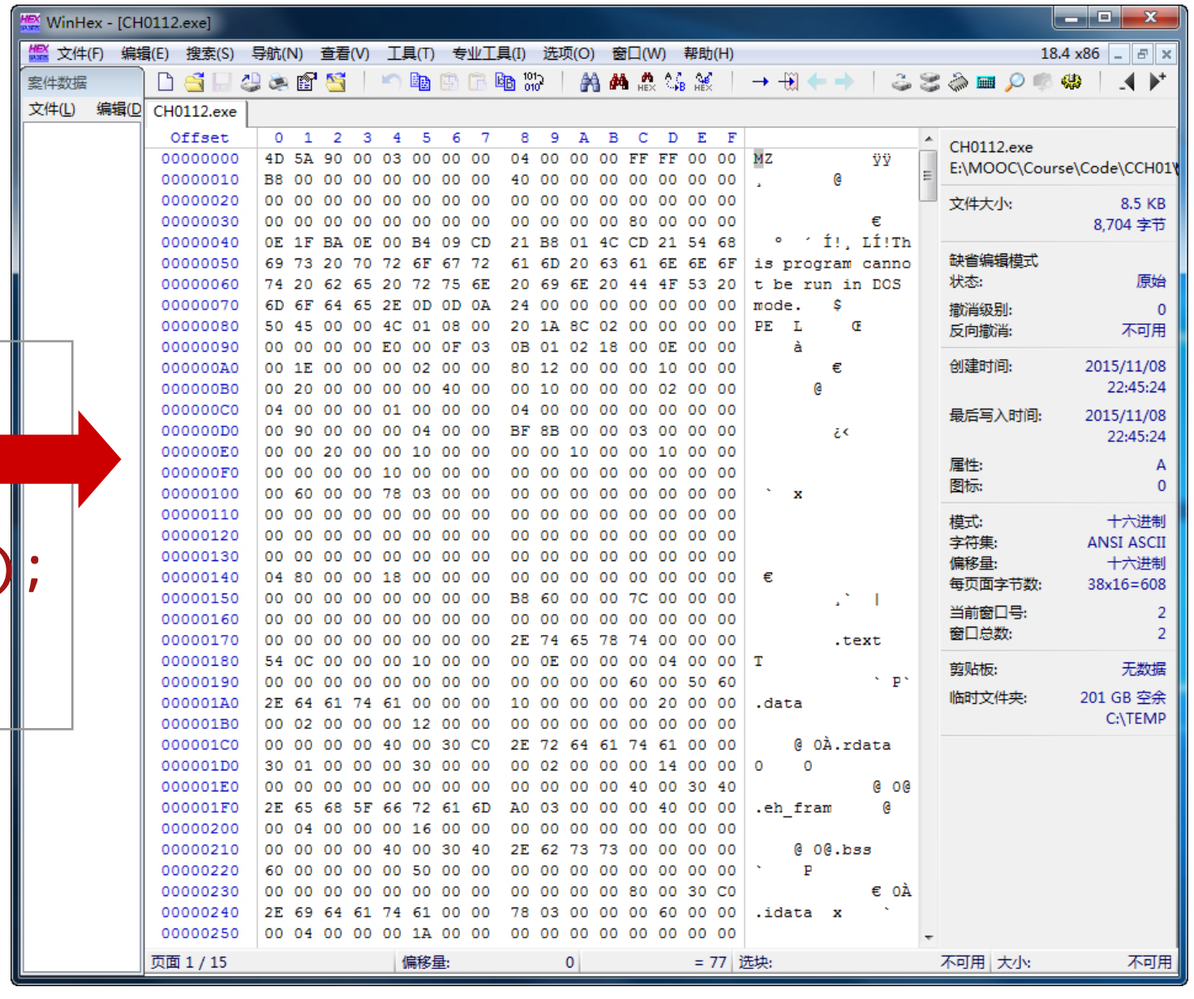
Linking View		Execution View
ELF Header		ELF Header
Program Header Table (Optional)		Program Header Table
Section 1		Segment 1
....		
Section n		Segment 2
....		
....		....
Section Header Table		Section Header Table (Optional)



4.6.4 程序映像和内存布局

- Helloworld
- 程序映像

```
#include <stdio.h>
int main()
{
  printf("hello,world\n");
  return 0;
}
```

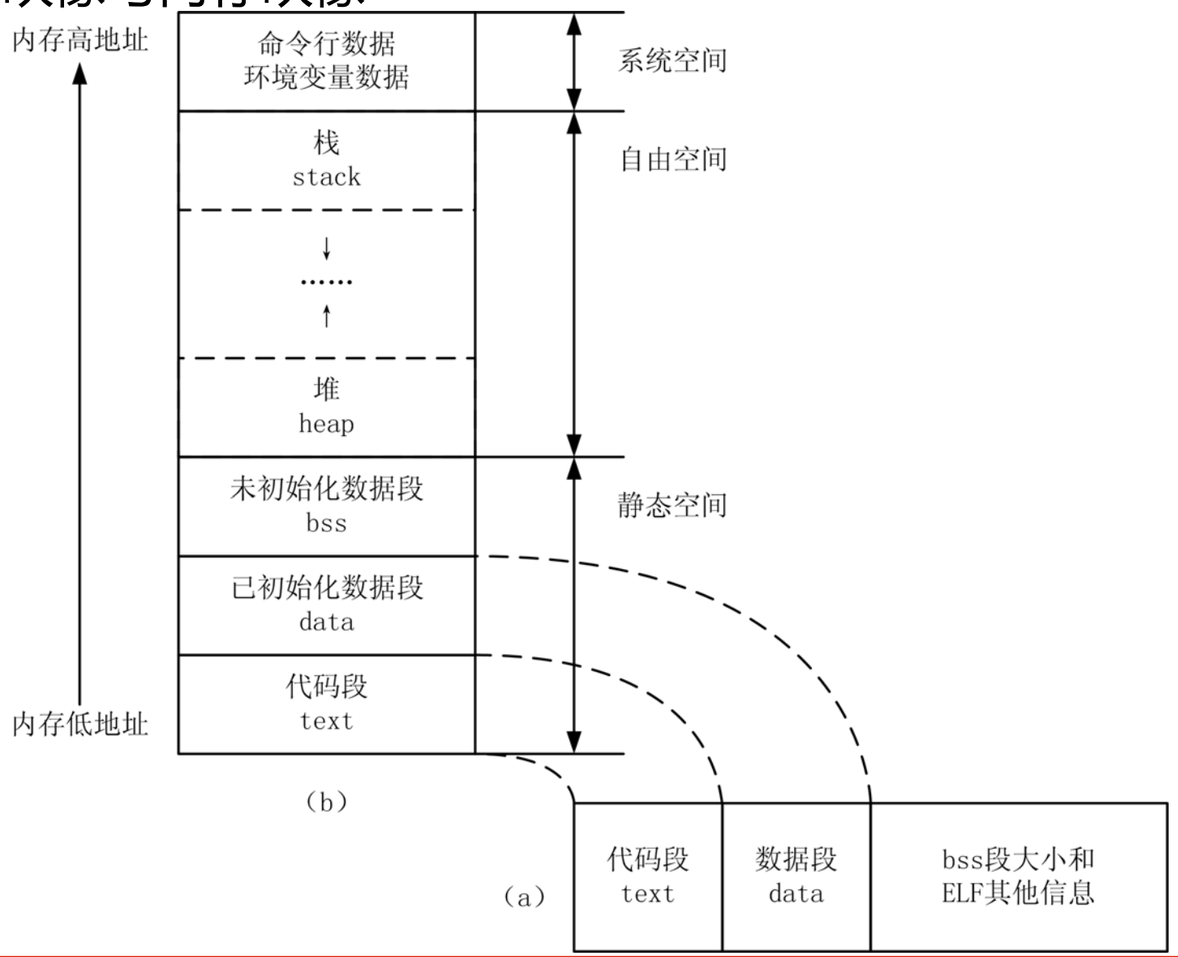


4.6.4 程序映像和内存布局

- 可执行连接格式ELF内容包含程序指令、已初始化的静态数据和其他一些重要信息，例如未初始化的静态数据空间大小、符号表（symbol table），调试信息（debugging information）、动态共享库的链接表（linkage tables for dynamic shared libraries）等。

4.6.4 程序映像和内存布局

图4.9 可执行文件映像与内存映像



4.6.4 程序映像和内存布局

- 2．程序加载
- 运行程序时，由操作系统将可执行文件载入到计算机内存中，成为一个进程（process）。程序在内存中的布局由5个段（segment）组成。

4.6.4 程序映像和内存布局

- 1．代码段
- 代码段（text segment）存放程序执行的机器指令（machine instructions）。通常情况下，text段是可共享的，使其可共享的目的是对于频繁被执行的程序，只需要在内存中有一份副本即可。text段通常也是只读的，使其只读的原因是防止一个程序意外地修改了它的指令。

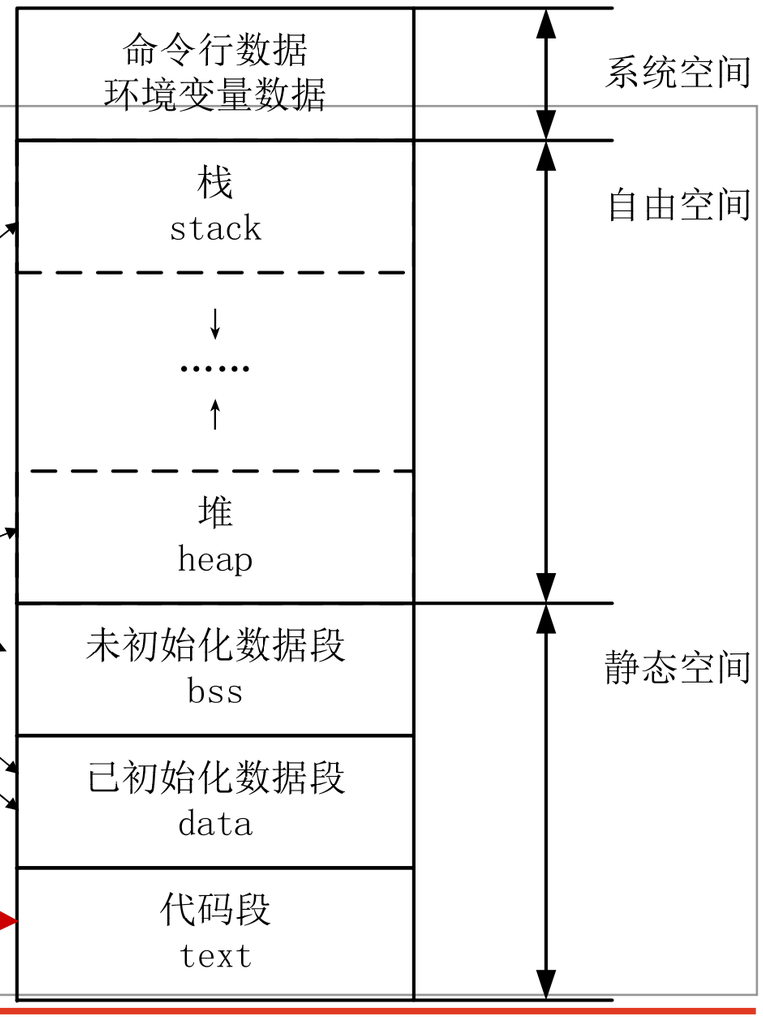
4.6.4 程序映像和内存布局

- C程序的表达式、语句、函数等编译成机器指令就存放于text段。程序运行时由操作系统从程序映像中取出text段，布局在程序内存地址最低的区域，然后跳转到text段的main函数开始运行程序，程序结束后由操作系统收回这段内存区域。

4.6.4 程序映像和内存布局

程序加载后的内存映像

```
#include <stdio.h>
#include <stdlib.h>
int K=10;//存放data段
float f;//存放bss段
int main()
{
  static int i=5;//存放data段
  double d=5;//存放stack段
  char *p;
  p=malloc(1);//存放heap段
  //...
  free(p);//释放heap段
  return 0;
}
```



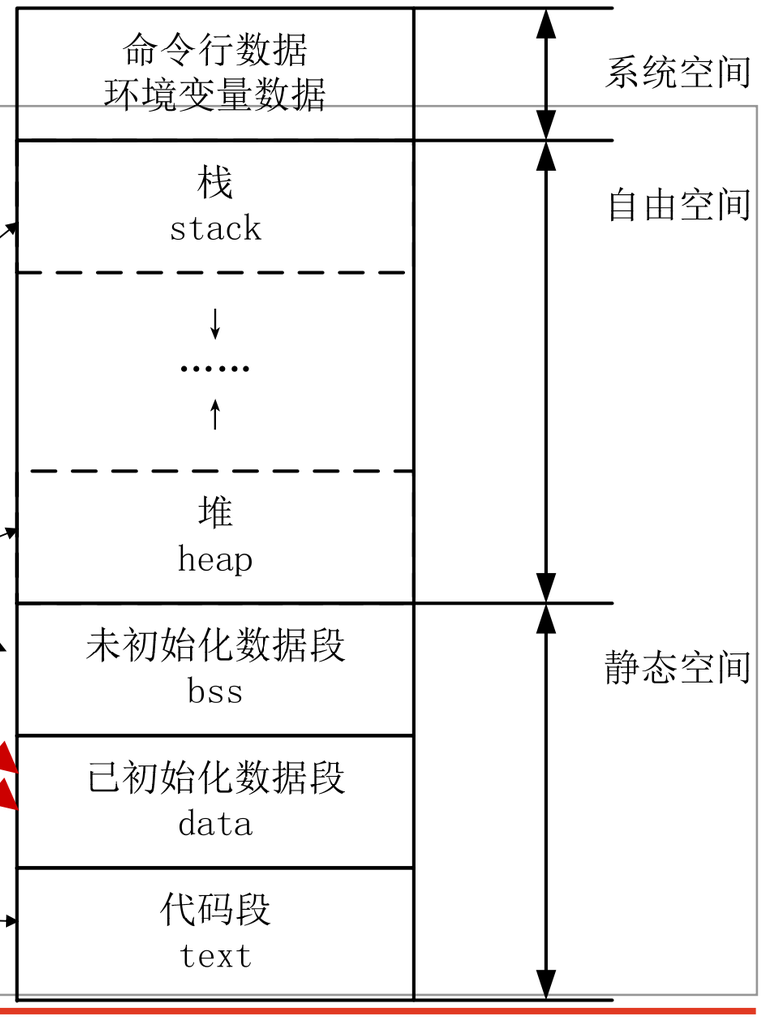
4.6.4 程序映像和内存布局

- 2．已初始化数据段
- 已初始化数据段（data segment）用来存放C程序中所有已赋初值的全局和静态变量、对象，也包括字符串、数组等常量，但基本类型的常量不包含其中，因为这些常量被编译成指令的一部分存放于text段。

4.6.4 程序映像和内存布局

程序加载后的内存映像

```
#include <stdio.h>
#include <stdlib.h>
int K=10;//存放data段
float f;//存放bss段
int main()
{
  static int i=5;//存放data段
  double d=5;//存放stack段
  char *p;
  p=malloc(1);//存放heap段
  //...
  free(p);//释放heap段
  return 0;
}
```



4.6.4 程序映像和内存布局

- 程序运行时由操作系统从程序映像中取出data段，布局在程序内存地址较低的区域。程序结束后由操作系统收回这段内存区域，即释放data段。
- 显然，data段的存储单元有与程序代码相同的生命期，它们的初始值实际在编译时就已经确定了。即使程序没有运行，这些存储单元的初始值也固定下来了，当程序开始运行时，这些存储单元是没有初始化的动作。

- 在程序运行中，data段的存储单元数据会一直保持到改变为止，或保持到程序结束为止。

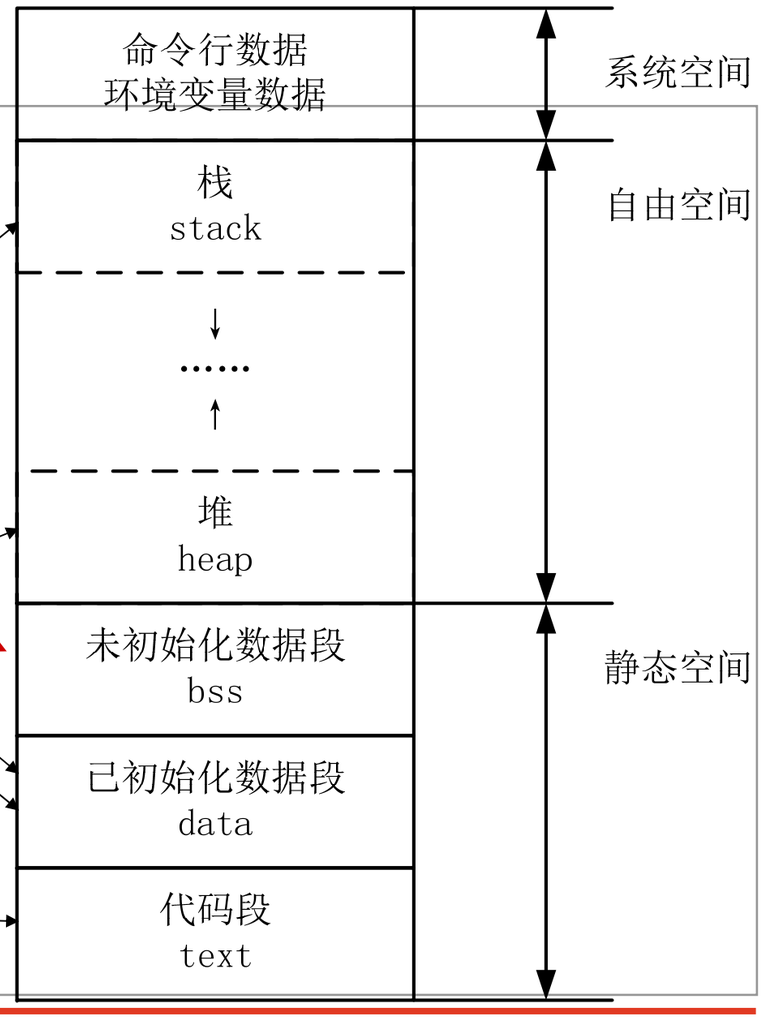
4.6.4 程序映像和内存布局

- 3．未初始化数据段
- 未初始化数据段（bss segment）用来存放C程序中所有未赋初值的全局和静态变量。
- 在程序映像中没有存储bss段，只有它的空间大小信息；程序运行前由操作系统根据这个大小信息分配bss段，且数据值全都初始化为0，布局在与data段相邻的区域。程序结束后由操作系统收回这段内存区域，即释放bss段。

4.6.4 程序映像和内存布局

程序加载后的内存映像

```
#include <stdio.h>
#include <stdlib.h>
int K=10;//存放data段
float f;//存放bss段
int main()
{
  static int i=5;//存放data段
  double d=5;//存放stack段
  char *p;
  p=malloc(1);//存放heap段
  //...
  free(p);//释放heap段
  return 0;
}
```



4.6.4 程序映像和内存布局

- 显然，bss段的存储单元也有与程序代码相同的生命期，但与data段不同的是如果程序没有运行，bss段的存储空间是不存在的，因而也就不会有初始值。在程序运行前，这些存储单元会初始化为0。此后，bss段的存储单元的性质与data段完全相同。

4.6.4 程序映像和内存布局

- data段和bss段的存储特点，决定了C程序中所有全局和静态变量、对象的存储空间在main函数运行前就已经存在，就有了初始值。程序运行到这些变量和对象的定义处时，是不会再有初始化动作的。在程序运行中这些变量和对象的存储空间不会被释放，一直保持到程序运行结束。期间如果数据被修改，则修改会一直保持。

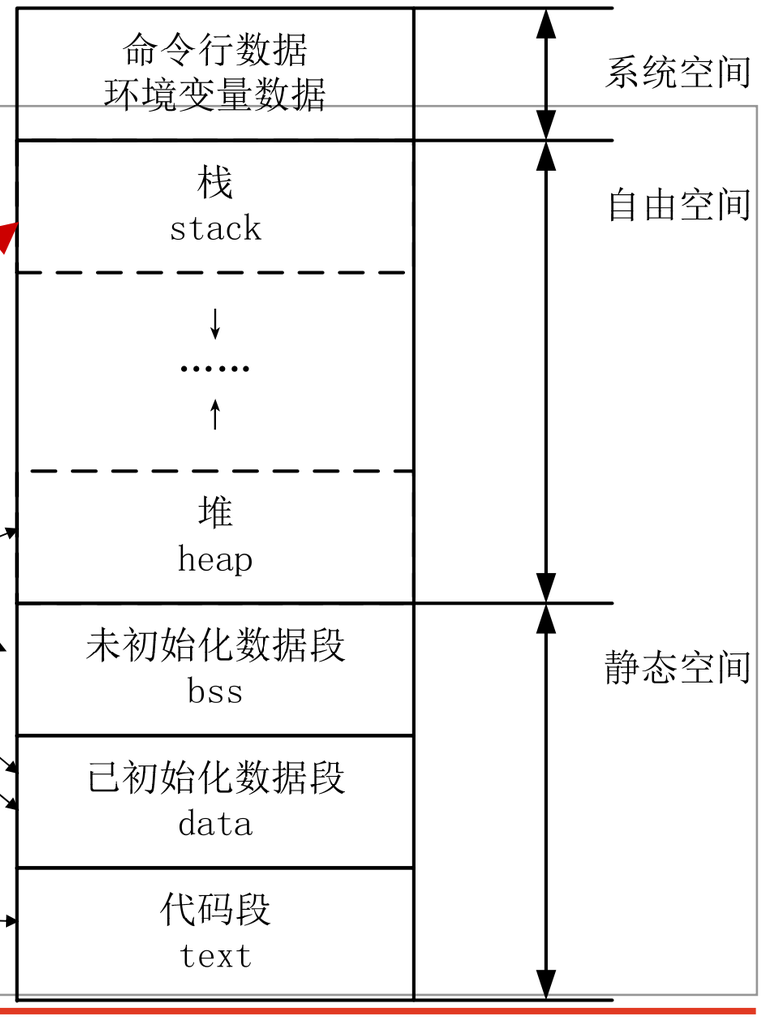
4.6.4 程序映像和内存布局

- 4．栈
- 栈（stack）用来存放C程序中所有局部的非静态型变量、临时变量，包含函数形参和函数返回值。
- 程序映像中没有栈，在程序开始运行时也不会分配栈。每当一个函数被调用，程序在栈段中按函数栈框架入栈，就分配了局部变量存储空间。如果这些变量有初始化，就会有赋值指令给这些变量送初值，否则变量的值就呈现随机性。当函数调用结束时，函数栈框架出栈，函数局部变量释放存储空间。

4.6.4 程序映像和内存布局

程序加载后的内存映像

```
#include <stdio.h>
#include <stdlib.h>
int K=10;//存放data段
float f;//存放bss段
int main()
{
  static int i=5;//存放data段
  double d=5;//存放stack段
  char *p;
  p=malloc(1);//存放heap段
  //...
  free(p);//释放heap段
  return 0;
}
```



4.6.4 程序映像和内存布局

- 栈的存储特点，决定了C程序中所有局部的非静态型变量，其存储方式是动态的。函数调用开始时得到分配，赋予初值，函数调用结束时释放空间，变量不存在。下次函数调用时再重复。

4.6.4 程序映像和内存布局

- 5．堆
- 堆（heap）用来存放C程序中动态分配的存储空间。

- 程序映像中没有堆，在程序开始运行时不会分配堆，函数调用时也不会分配堆。堆的存储空间分配和释放是通过指定的程序方式来进行的，即由程序员使用指令分配和释放，若程序员不释放，程序结束可能由操作系统回收。

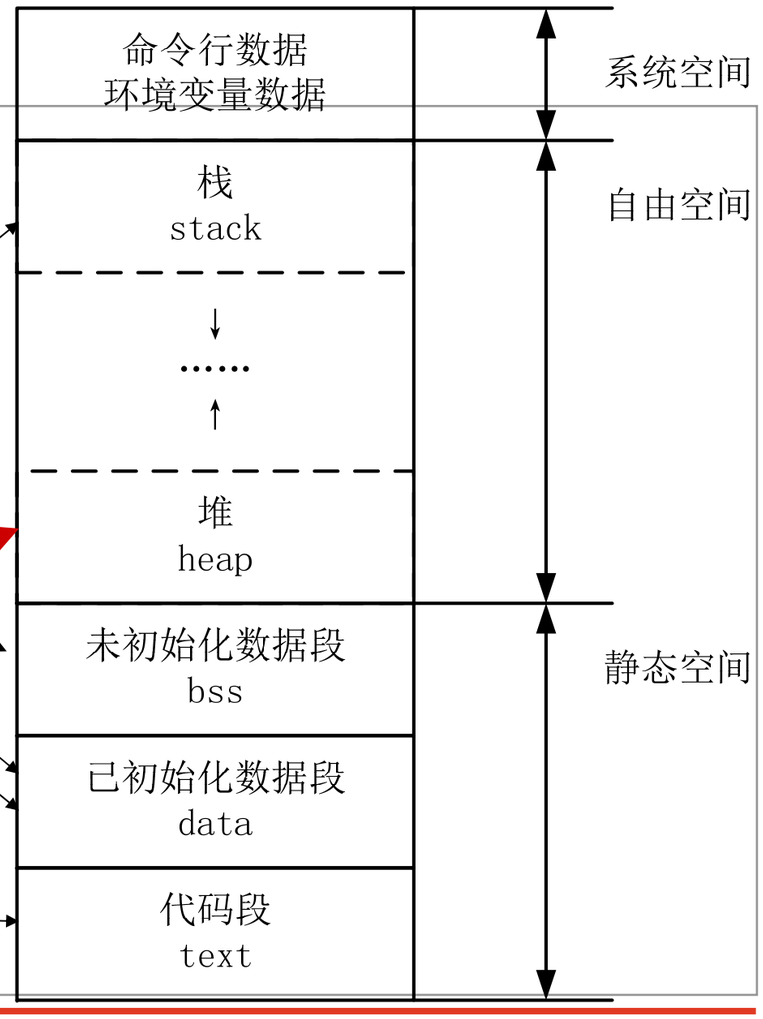
4.6.4 程序映像和内存布局

- C语言中可以通过使用指针、动态内存分配和释放函数来实现堆的分配和释放。程序可以通过动态内存分配和释放来使用堆区，堆区有比栈更大的存储空间、更自由的使用方式。

4.6.4 程序映像和内存布局

程序加载后的内存映像

```
#include <stdio.h>
#include <stdlib.h>
int K=10;//存放data段
float f;//存放bss段
int main()
{
  static int i=5;//存放data段
  double d=5;//存放stack段
  char *p;
  p=malloc(1);//存放heap段
  //...
  free(p);//释放heap段
  return 0;
}
```



4.6.4 程序映像和内存布局

- 堆和栈的共同点是动态存储，处于这两个区域的存储单元可以随时分配和释放，所以这些存储单元的使用特点呈现临时性的特点。data段的特点是静态存储，处于这个区域的存储单元随程序运行而存在，随程序结束才释放，相对程序生命期，data段存储单元的使用特点呈现持久性的特点。data段由于持久占有存储空间，因此大小会被操作系统限定，而堆可以达到空闲空间的最大值。

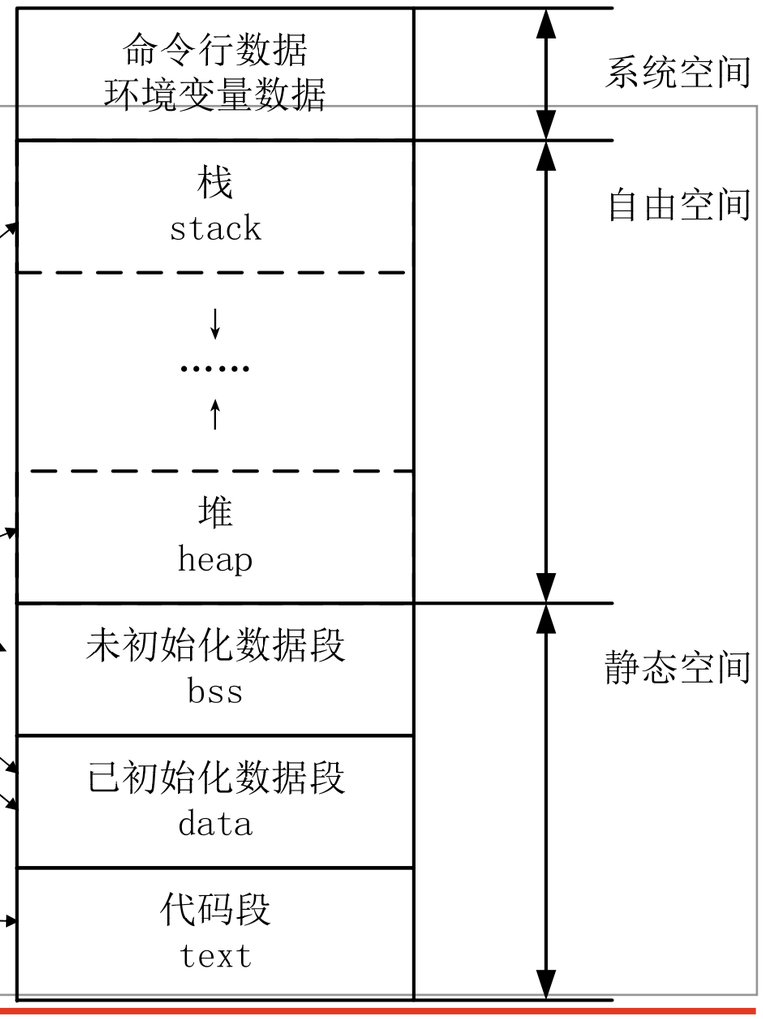
4.6.4 程序映像和内存布局

- 堆和栈的区别是分配方式的不同，栈是编译器根据程序代码自动确定大小，到函数调用时有指令自动完成分配和释放的；堆则完全由程序员指定分配大小、何时分配、何时释放。堆的优点是分配和释放是自由的，缺点是需要程序员自行掌握分配和释放时机，特别是释放时机，假如已经释放了还要使用堆会产生引用错误，或者始终没有释放产生内存泄漏（memory leak）。

4.6.4 程序映像和内存布局

程序加载后的内存映像

```
#include <stdio.h>
#include <stdlib.h>
int K=10;//存放data段
float f;//存放bss段
int main()
{
  static int i=5;//存放data段
  double d=5;//存放stack段
  char *p;
  p=malloc(1);//存放heap段
  //...
  free(p);//释放heap段
  return 0;
}
```



4.6.4 程序映像和内存布局

例4.23

```
1  //程序内存布局举例
2  #include <stdlib.h>
3  int a=0; //a存储在已初始化数据区data
4  char *p1; //p1存储在bss区（未初始化全局变量）
5  int main()
6  { int b; //b存储在stack
7    char s[]="abc"; //s为数组变量，存储在stack
8    //"abc"为字符串常量，存储在data
9    char *p1,*p2,*p3="123456"; //p1-p3在栈区 123456\0在data
10   static int c =0; //c为全局（静态）数据，存储在data
11   //静态数据会自动初始化
12   p1=(char *)malloc(10);//分配得来的10个字节的区域在heap
13   p2=(char *)malloc(20);//分配得来的20个字节的区域在heap
14   free(p1); free(p2);
15   return 0;
```


程序设计