第5章

C语言的函数

本章主要内容：

（1）函数。

（2）局部变量和全局变量。

（3）动态存储和静态存储。

（4）跑马灯实验。

5.1 函　　数

函数

函数是C语言支持程序模块化设计的基本单元。程序员可以直接调用系统提供的库函数，从而简化程序的开发；也可以根据需要自己定义函数，从而实现一个独立的功能模块。函数的使用使程序结构层次清晰，便于阅读和调试。函数也是代码重用的一种重要手段。

【例5-1】　从键盘任意输入一个正整数n，计算1到n之间所有奇数的和并输出。

程序代码：

```
/*****************************************************************************
源程序名:D:\C_Example\5_Function\oddSum.c
功能:计算1到正整数n之间所有奇数的和并输出
输入数据:正整数n的值
输出数据:1到正整数n之间所有奇数的和
*****************************************************************************/
#include<stdio.h>
int oddSum(int m)                  //函数定义
{
  int i, s;
  s=0;                             //存放奇数和的变量初始化
  for(i=1;i<=m;i++)                //循环n次,判断1到n之间的每个整数是否是奇数
    if((i%2)!=0)                   //判断是否是奇数
      s=s+i;                       //奇数累加
  return s;
}
int main()
```

```
{
  long int sum;                          //定义存放奇数和的变量 sum
  int n;
  printf("请输入正整数 n=");
  scanf("%d",&n);                        //输入正整数 n
  sum = oddSum(n);                       //调用求奇数和的函数
  printf("奇数和 sum=%d",sum);           //输出奇数和
  return 1;
}
```

编辑、编译和运行例 5-1 的程序,运行结果如图 5-1 所示。

1. 函数的定义

函数定义的一般形式如图 5-2 所示。

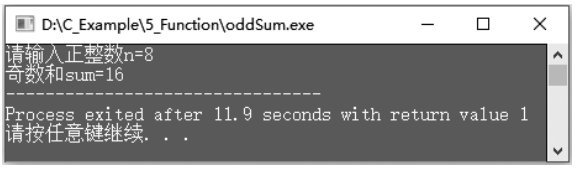

图 5-1 oddSum.exe 程序运行结果

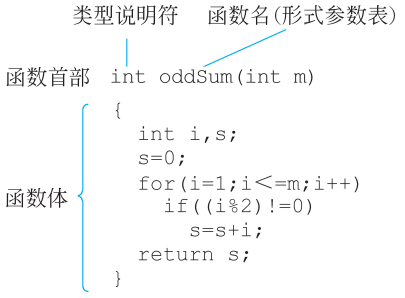

图 5-2 函数定义的一般形式

类型说明符是函数返回值的类型。当函数有返回值时,要通过 return 语句返回,因此 return 语句中的表达式类型或者返回值的类型应该与类型说明符保持一致;当函数无返回值时,类型说明符为 void。函数名是用户自定义的函数名称。形式参数表是用逗号分隔的参数列表,参数是实现函数与外部传递数据的通道。若不需要传递数据则参数为空,称为无参函数;否则为带参函数。

2. 函数的调用

函数调用的一般形式为

```
函数名(实际参数表);                    //无参函数省略实际参数表
```

例 5-1 中的 int oddSum(int m)语句是 oddSum 函数的定义,其中参数 m 称为形式参数,简称形参,形参在内存中没有分配存储空间;sum = oddSum(n)语句是对 oddSum 函数的调用,其中参数 n 称为实际参数,简称实参,是在 main 函数中定义的变量。

参数是调用函数与被调用函数之间交换数据的通道。当函数被调用时,系统才为形参在内存中分配存储空间,并将实参 n 的值传递给形参 m,完成数据传递。函数执行结束时,为形参分配的存储空间将被释放。

注意: 函数调用时提供的实参个数、类型、顺序应与定义的形参一致。

函数的参数可以是一般变量、指针或引用形式,分别传递变量值、地址值。当形参是变量值时,函数调用时实参向形参单向传递值,它们各自占用不同的存储空间,因而形参值的改变不会影响实参值。当形参是指针时,传递的是地址值,具体参见 6.1 节。

3. 函数原型

函数只有通过调用才能被执行，与函数的定义位置无关。在例 5-1 的程序中，是将被调用函数 oddSum(int m)定义在主调函数 main 之前，现在将 oddSum(int m)函数定义在 main 函数之后，但在编译时函数调用语句 sum = oddSum(n);将会出现编译错误："' oddSum ' : undeclared identifier"，这是因为函数必须在使用之前进行声明或者定义。

函数原型是对函数的声明，其作用是告诉编译器有关函数的接口信息，包括：函数的名称，函数的参数个数、类型和顺序，函数的返回值类型。编译器需要根据这些信息检查函数调用是否正确。

函数原型声明的一般形式为

```
返回值类型 函数名(形参类型　形参名[,…]);
```

如果函数定义在被调用函数之前，则可省略函数原型的声明。因为，函数定义中就给出了函数的原型信息。在例 5-1 的程序中添加函数声明语句，并将 oddSum 函数定义在 main 函数之后，代码如下：

```
/*********************************************************************************
源程序名:D:\C_Example\5_Function\oddSum01.c
功能:计算 1 到正整数 n 之间所有奇数的和并输出
输入数据:正整数 n 的值
输出数据:1 到正整数 n 之间所有奇数的和
*********************************************************************************/
#include<stdio.h>
int oddSum(int m);                          //函数声明
int main()
{
  long int sum;
  int n;
  printf("请输入正整数 n=");
  scanf("%d",&n);
  sum = oddSum(n);
  printf("奇数和 sum=%d",sum);
  return 1;
}
int oddSum(int m)
{
  int i, s;
  s=0;
  for(i=1;i<=m;i++)
    if((i%2)!=0)
      s=s+i;
  return s;
}
```


C 语言的局部变量和全局变量

5.2 C 语言的局部变量和全局变量

在前面的程序中,使用变量时要考虑的基本特性包括变量名、变量的数据类型和变量的值。除此之外,C 语言的变量还分为局部变量和全局变量,这两种变量在 C 语言程序中的作用域是不同的。

变量的作用域是指变量在程序中的有效作用范围,即被声明的变量在程序中的有效代码区域,也称变量在该区域是可见的或可用的。下面首先通过例 5-2 展示局部变量和全局变量及其作用域问题。

【例 5-2】 从键盘任意输入两个整数,将其中较大的数输出。

程序代码:

```
/*****************************************************************************************
源程序名:D:\C_Example\5_Function\maxData.c
功能:展示变量作用域,局部变量和全局变量
输入数据:任意输入两个整数
输出数据:两个整数中较大的数以及变量的地址
*****************************************************************************************/
#include<stdio.h>
void maxdata(int a,int b);                          //函数参数中定义局部变量 a 和 b
int max;                                            //定义全局变量 max
int main()
{
  int a,b;                                          //定义局部变量 a 和 b
  max = 0;
  printf("请输入两个整数 a=");
  scanf("%d",&a);
  printf("请输入两个整数 b=");
  scanf("%d",&b);
  printf("\n\n");
  printf("main 函数中复合语句前面变量 a 的地址:%x\n",&a);
  printf("main 函数中复合语句前面变量 a 的值 a=%d\n",a);
  printf("main 函数中复合语句前面变量 b 的地址:%x\n",&b);
  printf("main 函数中复合语句前面变量 b 的值 b=%d\n",b);
  printf("main 函数中复合语句前面变量 max 的地址:%x\n",&max);
  printf("main 函数中复合语句前面变量 max 的值 max=%d\n",max);
  printf("\n\n");
  {                                                 //复合语句开始
    int a,b,max;
    a=3; b=5;max=3;
    printf("复合语句中变量 a 的地址:%x\n",&a);
    printf("复合语句中变量 a 的值 a=%d\n",a);
    printf("复合语句中变量 b 的地址:%x\n",&b);
    printf("复合语句中变量 b 的值 b=%d\n",b);
    printf("复合语句中变量 max 的地址:%x\n",&max);
    printf("复合语句中变量 max 的值 max=%d\n",max);
    printf("\n\n");
```

```
    }                                                    //复合语句结束
    printf("main 函数中复合语句后面变量 a 的地址:%x\n",&a);
    printf("main 函数中复合语句后面变量 a 的值 x=%d\n",a);
    printf("main 函数中复合语句后面变量 b 的地址:%x\n",&b);
    printf("main 函数中复合语句后面变量 b 的值 b=%d\n",b);
    printf("\n\n");
    maxdata(a,b);                                        //调用 maxdata()为全局变量 max 赋值
    printf("maxdata 函数执行后 main 函数中变量 max 的地址:%x\n",&max);
    printf("maxdata 函数执行后 main 函数中变量 max 的值 max=%d\n",max);
    printf("\n\n");
    printf("%d,%d 中的最大的数是:%d",a,b,max); //引用全局变量
    return 1;
}
void maxdata(int a,int b)                                //为全局变量赋值,函数不需要返回值
{
    if(a>=b)
      max=a;
    else
      max=b;                                             //引用全局变量
    printf("maxdata 函数中变量 a 的地址:%x\n",&a);
    printf("maxdata 函数中变量 a 的值 a=%d\n",a);
    printf("maxdata 函数中变量 b 的地址:%x\n",&b);
    printf("maxdata 函数中变量 b 的值 b=%d\n",b);
    printf("maxdata 函数中变量 max 的地址:%x\n",&max);
    printf("maxdata 函数中变量 max 的值 max=%d\n",max);
}
```

在例 5-2 的程序中分别定义了如下变量：

(1) 在所有函数的外部、main 函数的前面用 int max;定义了整型变量 max。

(2) 在 main 函数内部第一行用 int a,b;定义了整型变量 a 和 b。

(3) 在 main 函数内部用一对花括号括起来的复合语句中用 int a,b,max;定义了整型变量 a、b 和 max。

(4) 在 void maxdata(int a,int b)函数用圆括号括起来的形参表中用 int a 和 int b 定义了整型变量 a 和 b。

可以看出,在例 5-2 的程序中存在 2 个重名的 max 变量、3 个重名的 a 变量和 3 个重名的 b 变量。那么,这些变量中同名的变量是同一个变量吗？它们之间存在什么关系？它们之间又有什么差别呢？

要想搞清楚这些问题,首先编辑、编译和运行例 5-2 的程序,观察运行结果,如图 5-3 所示。需要强调的是,由于计算机内存大小不同,正在运行的操作系统以及正在执行的程序也不同,不同的计算机运行例 5-2 的程序,得到的各个变量的地址会不一样。也就是不同的计算机执行例 5-2 的程序的结果会跟图 5-3 类似,但可能有区别。

图 5-3 显示了程序中定义的这些变量的地址和存储的值,图 5-4 是根据这些变量的地址以及这些变量所存储的值绘制的变量占用内存的示意图。从图 5-3 和图 5-4 可以得出变量、变量地址、变量存储的值的关系,如表 5-1 所示。

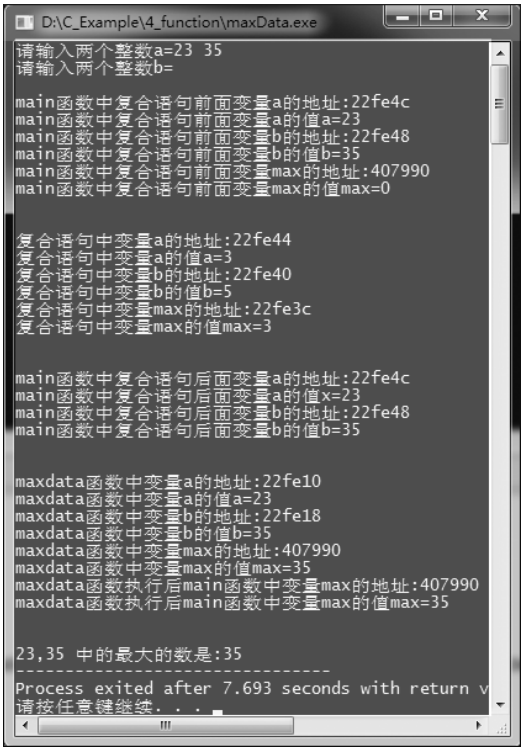

图 5-3　maxData.c 程序执行结果

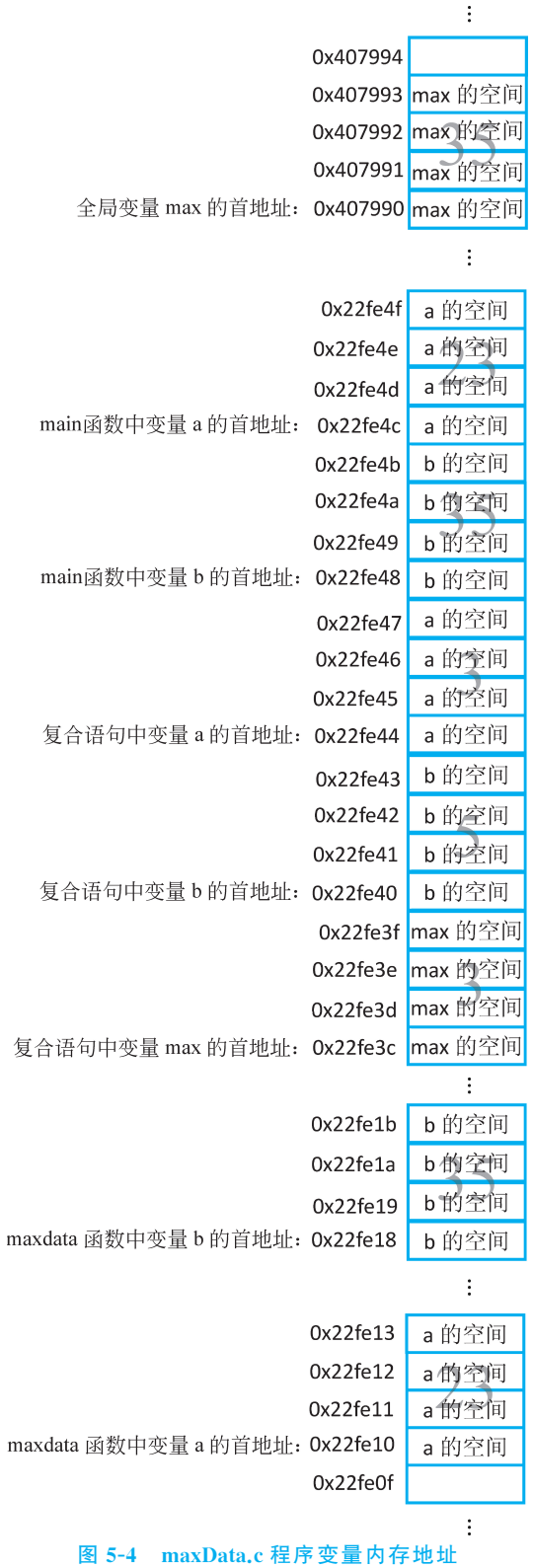

图 5-4　maxData.c 程序变量内存地址

表 5-1　程序 maxdata.c 中的变量、变量地址、变量存储的值的关系

序号	变量所属函数		变量类型和名称	变 量 地 址	变量存储的值	变量类别
1	无		int max	0x407990	0→35	全局变量
2	main 函数		int a	0x22fe4c	23	局部变量
3			int b	0x22fe48	35	局部变量
4	main 函数	复合语句	int max	0x22fe3c	3	局部变量
5		复合语句	int a	0x22fe44	3	局部变量
6		复合语句	int b	0x22fe40	5	局部变量
7	maxdata 函数		int a	0x22fe10	23	局部变量
8			int b	0x22fe18	35	局部变量
9			max	0x407990	35	全局变量

从表 5-1 中可以清晰地看出：

(1) 前两个 max 变量(序号 1 和 4)不但地址不同，存储的数据也不同，显然是两个不同的变量。其中，第一个 max 变量不属于任何函数，被称为全局变量；第二个 max 变量是在 main 函数内部的复合语句中定义的变量，被称为局部变量。

(2) 第一个和第三个 max 变量(序号 1 和 9)地址相同(都是 0x407990)，存储的数据也相同(都是 35)，实际上它们是同一个变量，也就是程序唯一的全局变量，通过这个全局变量实现了变量 max 在函数 main 和 maxdata 之间的共享应用，起到在不同函数之间传递数值 35 的作用。

(3) 3 个 a 变量(序号为 2、5、7)虽然同名，但地址不相同，就像 3 个不同的人尽管取了相同的名字，仍然是 3 个不同的人一样，它们是 3 个不同的变量，属于不同的函数或者复合语句，被称为局部变量。

(4) 3 个 b 变量(序号为 3、6、8)和上面讲的 a 变量类似，它们的地址也不相同，属于不同的函数或者复合语句，也是局部变量。

全局变量(global variable)是指定义在函数外，不属于任何函数的变量，其作用域是从定义位置开始到程序所在文件结束。全局变量的定义格式与局部变量完全相同，只是定义位置不同，它可以定义在程序的开始、中间等任何位置。全局变量对定义位置之后的所有函数都有效。因此，全局变量成为多个函数共享的变量，可以用于函数之间的数据传递。例如，在例 5-2 中，main 函数前面定义的全局变量 max 既在 main 函数中得到使用，也在 maxdata(int a,int b)函数中得到使用，起到了传递数值 35 的作用，将 maxdata(int a,int b)函数中的变量 a 和 b 中较大的值通过 max 传递给 main 函数。

局部变量(local variable)通常是指在函数的形参表、函数内部、复合语句内部定义的变量，例如在 maxdata 函数中定义的 3 个局部变量。局部变量的作用域局限于函数内部或者复合语句内部，从定义位置开始至本函数或者本复合语句结束。使用局部变量可以避免不同函数之间同名变量的互相干扰，也就是说不同函数内部可以出现同名变量，它们有各自的存储空间和作用范围，不会产生冲突。图 5-4 直观地展示了局部变量的这种情况。

在复合语句内定义的局部变量，其作用域只限于复合语句内；在函数原型声明语句内的局部变量，其作用域只限于函数原型。具体可参见例 5-2 中此种变量的作用情况。

注意：函数原型声明语句 void maxdata(int a,int b);等价于 void maxdata(int,int);。

函数原型声明语句中的形参作用域只在声明语句内。因此,声明语句中的函数形参名是可以省略的。即使函数原型中带有形参名,编译器在编译时也将忽略它。

注意：使用全局变量会破坏函数的独立性,容易使函数之间相互干扰,因此要谨慎使用。

由于作用域的不同,程序中可能出现全局变量与局部变量同名,例如,例 5-2 在 main 函数外部定义了全局变量 max,在 main 函数内部复合语句中定义了局部变量 max,在这种情况下,复合语句内部的局部变量 max 会屏蔽全局变量 max,也就是只有局部变量 max 有效。


C 语言变量的静态存储和动态存储

5.3 C 语言变量的静态存储和动态存储

C 语言变量不但分为全局变量和局部变量,在变量存储类别上还分为静态存储和动态存储。变量存储类别决定了变量的生命周期,也就是变量在内存中的生存时间,对应变量从开始获得存储空间到最后释放存储空间的整个过程。为了更好地理解变量的静态存储和动态存储,首先看一个变量静态存储和动态存储的 C 语言程序。

【例 5-3】 变量静态存储和动态存储示例。

程序代码:

```
/*****************************************************************************************
源程序名:D:\C_Example\5_Function\variableStorage.c
功能:展示变量静态存储和动态存储
输入数据:无
输出数据:见图 5-5
*****************************************************************************************/
#include<stdio.h>
int Fun();
int main()
{
  int f,i;                                  //自动变量
  printf("f 和 i 未初始化时的值:f = %d, i= %d\n\n",f,i);
  for(i = 1;i<= 5;i++)
  {
    f = Fun();
    printf("  自动变量:f = %d\n",f);
  }
  return 0;
}
int Fun()
{
  int m = 0;                                //自动变量
  static int n=0;                           //静态局部变量
  m++;
  printf("  自动变量:m = %d ",m);
  n++;
  printf("  静态局部变量:n = %d ",n);
  return n;
}
```

编辑、编译和运行例 5-3 的程序,运行结果如图 5-5 所示。从程序运行结果可以看出,在函

图 5-5　variableStorage.exe 程序运行结果

数 Fun 中定义的变量 m 和 n 差异比较大，比较一下这两个变量在函数 Fun 中的定义、初始化和运算：两个变量的初始化分别是 int m = 0;和 static int n=0;，两个变量都进行自增运算。两者的差别在于变量 n 的定义前面加了 static，就是因为这一差别，导致两个变量运算的值差异巨大：变量 m 的值每次调用函数 Fun 时一直是 1；变量 n 的值每次调用函数 fun 时都会增加 1，一直到 5。原因就在于 m 变量是动态存储的，变量 n 因为定义时前面加了 static，所以就成为静态存储的变量。其实，例 5-3 程序的 main 函数中定义的变量 f 和 i 也都是动态存储的。

1. 动态存储

采用动态存储方式时，由系统自动完成内存的分配和释放。动态存储由系统的堆栈实现，系统自动根据函数的执行情况为变量分配和回收堆栈空间。动态存储分配的数据区就是动态存储区，存放程序中函数内部的局部变量，例如例 5-3 的 main 函数中定义的变量 f、i 以及例 5-2 的 maxdata(int a,int b)函数形参中定义的变量 a、b 等。

局部变量和形参都默认为 auto 存储类别。auto 类型的变量也称自动变量，都采用动态存储方式。一般自动变量在定义时都省略 auto。

因此，int a,b;等价于 auto int a,b;，这两种定义的效果是一样的。

自动变量只有在函数被调用时才由系统分配相应的存储单元；到函数调用结束时，存储单元被自动释放。这一切由系统自动完成。在例 5-3 的 Fun 函数中，m 被定义为自动变量。在运行程序的过程中，Fun 函数被循环调用 5 次。每次被调用时，m 变量均被分配存储空间，并被初始化为 0；然后通过 m++运算自增其值，变为 1，并输出 m=1；最后在函数结束时 m 变量的存储空间被释放，进入下一轮循环。这样就导致每次循环执行时都是输出 m=1，一共输出 5 次。因此，局部变量、形参在函数被调用之前并不占有存储单元。

注意：自动变量如果未被赋初值，则其值是随机的，因此，程序中一定要注意自动变量的初始化。例如，例 5-3 的 main 函数中定义的自动变量 f 和 i 在未被初始化时输出的随机值分别是 1 和 0。

2. 静态存储

与动态存储不同的是，静态存储方式是在变量定义时就由系统分配了存储空间并保持不变，直至整个程序结束。全局变量和 static 类型的局部变量都是静态存储方式，被分配在静态存储区。

全局变量在整个程序运行期间都占用存储空间，其生命周期贯穿于整个程序。全局变量在定义时如果未被初始化，系统将自动为其赋初值 0，例如例 5-2 的 main 函数前定义的全局变量 max。

除了全局变量,还有一种特殊的 static 类型的局部变量,称为静态局部变量。当局部变量在定义时加上 static 就成为静态存储方式的局部变量。

例如,在例 5-3 的 Fun 函数中,static int n; 将 n 定义为静态局部变量。在运行程序过程中,Fun 函数被循环调用 5 次。第一次被调用时 n 变量被分配存储空间并被初始化为 0,然后通过 n++运算自增其值变为 1,输出 n=1,在 Fun 函数结束时 n 变量的空间依然保留;随后进入第二次调用,此时 n 变量的空间不但存在,而且 n=1,在此基础上通过 n++运算自增其值变为 2,Fun 函数执行结束;随后进入第三次调用,使 n=3;随后进入第四次调用,使 n=4;最后进入第五次调用,使 n=5。

因此,静态局部变量在函数第一次被调用时获得存储单元,其后一直保持该存储单元,直到程序运行结束,即使所在的函数被调用结束,存储空间也不会被回收,只有整个程序执行结束时,静态变量占据的存储空间才被释放和回收。静态局部变量只在第一次使用时完成初始化。如果变量未被赋值,系统自动将其初始化为 0。其后变量的值一直保持到下一次函数被调用时。

如果要指定变量的存储类别,其一般定义形式为

```
存储类别 类型标识符 变量名 1[,…];
```

例如:

```
auto int x,y;                          //auto 可以省略
static float f1,f2;                    //static 不能省略
```

虽然静态局部变量与全局变量的生命周期都贯穿了整个程序的运行过程,但静态局部变量只能在其定义的函数内使用,不能用于其他函数,也就是其作用域始终不变。


C 语言程序在内存中的存储结构

5.4 C 语言程序在内存中的存储结构

C 语言程序在内存中的存储结构主要由代码区、常量区、静态区、堆区和栈区组成。

(1) 代码区。顾名思义,用于存放编写的 C 语言程序的代码(编译后的二进制代码)。

(2) 常量区。常量(包括数值常量、字符常量和字符串常量)存储在该区域,不允许修改。

(3) 静态区。存储全局变量和静态变量的区域,初始化的全局变量和静态变量在一块区域,未初始化的全局变量和静态变量在相邻的另一块区域。在程序结束后,该区域内的变量由系统释放。

(4) 堆区。在程序运行过程中,通过 new、malloc、realloc 函数申请分配的内存块称为堆区。编译器不会负责该区域的释放工作,需要由程序员编写执行释放内存的程序代码予以释放。堆区的分配方式类似于数据结构中的链表。内存泄漏通常说的就是堆区。堆区往地址增大方向增长。

(5) 栈区。用于存放函数的参数值、函数内部的局部变量等,由编译器自动分配和释放,通常在函数执行完后就释放了,其操作方式类似于数据结构中的栈。栈的内存分配运算内置于 CPU 的指令集,效率很高,但是分配的内存量有限,例如 iOS 中栈区的大小是 2MB。栈区往地址减小方向增长。