C程序存储布局

一个由 C/C++编译的程序占用的内存（ memory ）分为以下几个部分：

1. 程序代码区（ .text ）

（即代码段-text ）存放函数体的二进制代码，代码段由程序中执行的机器代码组成。在C语言中，程序语句进行编译后，形成机器代码。在执行程序的过程中，CPU的程序计数器指向代码段的每一条机器代码，并由处理器依次运行。

2. 文字常量区（ .rodata ）

常量字符串就是放在这里的（即只读数据段），程序结束后由系统释放(rodata—read only data)。只读数据段是程序使用的一些不会被更改的数据，使用这些数据的方式类似查表式的操作，由于这些变量不需要更改，因此只需要放置在只读存储器中即可。

3. 全局区/静态区（ static ）

全局变量和静态变量的存储是放在一块的。初始化的全局变量和静态变量在一块区域(.rwdata or .data)（即已初始化读写数据段），未初始化的全局变量和未初始化的静态变量在相邻的另一块区域(.bss)（即未初始化数据段），程序结束后由系统释放。已初始化数据是在程序中声明，并且具有初值的变量，这些变量需要占用存储器的空间，在程序执行时它们需要位于可读写的内存区域内，并具有初值，以供程序运行时读写。未初始化数据是在程序中声明，但是没有初始化的变量，这些变量在程序运行之前不需要占用存储器的空间。（注意：在 C++中，已经不再严格区分bss和 data了，它们共享一块内存区域。）

4. 堆区（ heap ）

一般由程序员分配释放(new/malloc/calloc delete/free)，若程序员不释放，程序结束时可能由操作系统回收。（注意：它与数据结构中的堆是两回事，但分配方式倒类似于链表。）

5. 栈区（ stack ）

由编译器自动分配释放，存放函数的参数值，局部变量的值等。其操作方式类似于数据结构中的栈。

静态区与动态区、映像与运行时、节与段：

1. 代码段（ Text ）、只读数据段（ RO data ）、读写数据段（ RW data ）、未初始化数据段（ BSS ）属于静态区域，而堆（ heap ）和栈（ stack ）属于动态区域。

2. 代码段（程序代码区）、只读数据段和（文字常量区）、读写数据段（全局区/静态区中已初始化的）将在链接之后产生，未初始化数据段（全局区/静态区中未初始化的）将在程序初始化的时候开辟，而堆和栈将在程序的运行中分配和释放。

3. C语言程序分为映像和运行时两种状态。在编译-连接后形成的映像中，将只包含代码段（ Text ）、只读数据段（ RO Data ）和读写数据段（ RW Data ）。在程序运行之前，将动态生成未初始化数据段（ BSS ），在程序的运行时还将动态形成堆（ Heap ）区域和栈（ Stack ）区域。一般来说，在静态的映像文件中，各个部分称之为节（ Section ），而在运行时的各个部分称之为段（ Segment ）。如果不详细区分，可以统称为段。

在C语言的程序中，对变量的使用还有以下几点需要注意：

1. 函数体中定义的变量通常是在栈上，不需要在程序中进行管理，由编绎器处理。

2. 用malloc,calloc,realloc等分配内存的函数所分配的内存空间在堆上，程序必须保证在使用free释放，否则会发生内存泄漏。

3. 所有函数体外定义的是全局变量，加了static后的变量不管是在函数内部或外部都放在全局区。

4. 使用const定义的变量将放于程序的只读数据区。

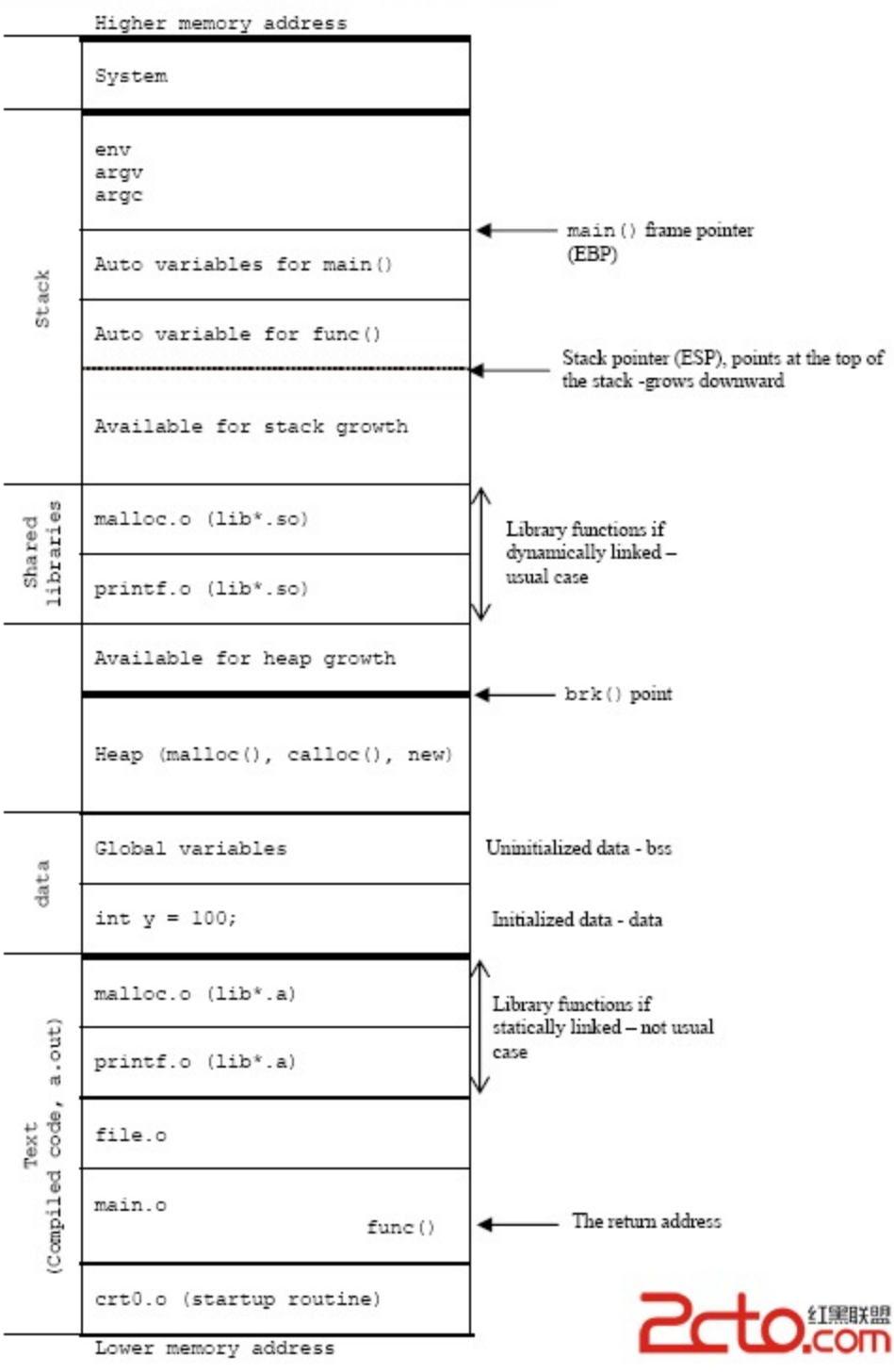
栈空间主要用于以下3种数据的存储：

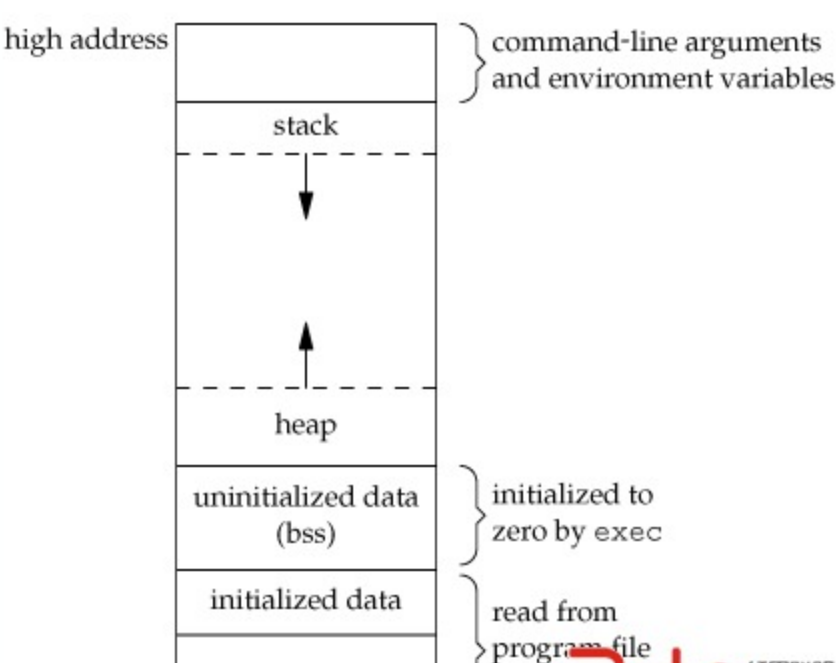
1.函数内部的动态变量

2.函数的参数

3.函数的返回值

栈空间是动态开辟与回收的。在函数调用过程中，如果函数调用的层次比较多，所需要的栈空间也逐渐加大，对于参数的传递和返回值，如果使用较大的结构体，在使用的栈空间也会比较大。





例子程序：

```
//main.cpp
int a = 0;                                  // 全局初始化区(data)
char *p1;                                   // 全局未初始化区(bss)
int main()
{
    int b;                                  // 栈区(stack)
    char s[] = "abc";                       // 栈区(stack)
    char *p2;                               // 栈区(stack)
    char *p3 = "123456";                    // p3 在栈区(stack);   "123456\0" 在常量区(rodata)
    static int c =0;                        // 全局/静态 初始化区   (data)
    p1 = (char *)malloc(10);
    p2 = (char *)malloc(20);                // 分配得来的 10 和 20 字节的区域就在堆区 (heap)
    strcpy(p1, "123456");                   // "123456\0" 放在常量区(rodata). 编译器可能会将它与 p3 所指向的"123456\0"优化成一个地方。
    return 0;
}
```

堆和栈的区别：

1.管理方式

对于栈来讲，是由编译器自动管理；对于堆来说，释放工作由程序员控制，容易产生 memory leak。

2. 空间大小

一般来讲在 32 位系统下，堆内存可以达到接近 4G 的空间，从这个角度来看堆内存几乎是没有什么限制的。但是对于栈来讲，一般都是有一定的空间大小的，例如，在 VC6 下面，默认的栈空间大小大约是 1M。

3. 碎片问题

对于堆来讲，频繁的new/delete 势必会造成内存空间的不连续，从而造成大量碎片，使程序效率降低；对于栈来讲，则不会存在这个问题，因为栈是先进后出的队列，永远都不可能有一个内存块从栈中间弹出。

4. 生长方向

对于堆来讲，生长方向是向上的，也就是向着内存地址增加的方向；对于栈来讲，它的生长方向是向下的，是向着内存地址减小的方向增长。

5. 分配方式

堆都是动态分配的，没有静态分配的堆；栈有 2 种分配方式：静态分配和动态分配。静态分配是编译器完成的，比如局部变量的分配，动态分配由 alloca 函数进行分配，但是栈的动态分配和堆是不同的，它的动态分配是由编译器进行释放，不需要我们手工实现。

6. 分配效率

栈是机器系统提供的数据结构，计算机会在底层分配专门的寄存器存放栈的地址，压栈出栈都有专门的指令执行，这就决定了栈的效率比较高；堆则是 C/C++函数库提供的，它的机制是很复杂的，例如为了分配一块内存，库函数会按照一定的算法（具体的算法可以参考数据结构/操作系统）在堆内存中搜索可用的足够大小的空间，如果没有足够大小的空间（可能是由于内存碎片太多），就有可能调用系统功能去增加程序数据段的内存空间，然后进行返回。显然，堆的效率比栈要低得多。

无论是堆还是栈，都要防止越界现象的发生。

关于 Global 和 Static 类型的一点讨论

1. static 全局变量与普通的全局变量有什么区别？

全局变量(外部变量)的定义之前再冠以 static 就构成了静态的全局变量。全局变量本身就是静态存储方式， 静态全局变量当然也是静态存储方式。 这两者在存储方式上并无不同。

这两者的区别在于非静态全局变量的作用域是整个源程序， 当一个源程序由多个源文件组成时，非静态的全局变量在各个源文件中都是有效的。而静态全局变量则限制了其作用域，即只在定义该变量的源文件内有效，在同一源程序的其它源文件中不能使用它。

由于静态全局变量的作用域局限于一个源文件内，只能为该源文件内的函数公用，因此可以避免在其它源文件中引起错误。static 全局变量只初使化一次，防止在其他文件单元中被引用。

2. static 局部变量和普通局部变量有什么区别 ？

把局部变量改变为静态变量后是改变了它的存储方式即改变了它的生存期。把全局变量改变为静态变量后是改变了它的作用域，限制了它的使用范围。static 局部变量只被初始化一次，下一次依据上一次结果值。

3. static 函数与普通函数有什么区别？

static 函数与普通函数作用域不同,仅在本文件。只在当前源文件中使用的函数应该说明为内部函数(static)，内部函数应该在当前源文件中说明和定义。对于可在当前源文件以外使用的函数，应该在一个头文件中说明，要使用这些函数的源文件要包含这个头文件.static 函数在内存中只有一份（ .data ），普通函数在每个被调用中维持一份拷贝。