# 程序生命周期阶段基础

在前一章中，读者应该已经对现代多任务操作系统在程序执行中所起的作用有了大致的了解。那么开发人员会很自然地想到下一个问题：若想让程序能够执行，需要做什么，如何做以及为何这样做？

就像一只蝴蝶的生命周期是从毛毛虫阶段开始的一样，程序的生命周期极大地依赖于二进制文件的内部构造。操作系统装载器会加载、解包这些数据，并将这些数据用于程序的执行阶段。本章随后即将讨论的内容将围绕准备数据和嵌入数据到二进制可执行文件结构体的方法展开，读者应该不会对此感到意外。我们假定程序是用C/C++编写的。

为了完全理解整个生命周期过程，我们会对程序生命周期的剩余细节进行详细的分析，包括程序的加载与执行阶段。本书将会重点围绕以下几个程序生命周期进行深入的讨论：

1. 编写源代码。
2. 编译。
3. 链接。
4. 装载。
5. 执行。

实际上，相比链接、装载和执行阶段来说，本章的内容侧重于讲解编译阶段的有关细节。

本章也会稍后提及其余一些阶段（特别是链接阶段）的内容，但是你仅仅会了解到这些内容的冰山一角。在介绍完大多数和链接相关的基础知识之后，本书的剩余章节将会针对装载和执行阶段进行详细的阐述。

## 2.1 基本假设

尽管大多数读者很可能属于高级或专家级开发人员，但我们在一开始依然会使用一些相当基础和简单的例子。在本章中，主要讨论一个非常简单且易于理解的例子。这个演示项目由两个简单的源代码文件组成，我们首先分别编译这两个文件，然后将其链接到一起。该项目的代码可以确保编译和链接的复杂度是最简单合理的。

比较特别的是，在本例中不会链接外部库，尤其是不会进行动态链接。唯一的例外是会链接C运行时库（对于绝大多数用C编写的程序来说都会链接这个库）。正因为在C程序执行的生命周期中使用C运行时库是如此普遍，因此为了简单起见，我们会有目的性地忽略链接C运行时库的一些特定细节，并假定所有C运行时库代码都被“自动”地嵌入程序内存映射结构体中。

通过这种方法，我们就能用最为简单且清晰的方式，对程序构建中的一些主要问题进行详细的阐述。

## 2.2 编写代码

考虑到本书讨论的主题是程序的构建过程（也就是编写完代码之后的过程），所以我们不准备在编写源代码上花费太长的时间。

除了在极少数情况下使用脚本来生成源代码的情况外，本书均假定用户使用编辑器输入代码，这些代码是一些使用ASCII字符编写的满足所选编程语言语法规则的语句（我们这里用的是C和C++）。你可以选择多种不同的编辑器，从最简单的ASCII文本编辑器到高级的集成开发环境（IDE）工具。考虑到本书绝大多数读者都是富有经验的开发人员，所以在选择编辑器的问题上也没有必要花费那么多时间去一一讲解了。

但是，这里有一项编程实践会对接下来的构建过程中产生重要影响，因此需要我们额外关注。为了更好地对源代码进行组织，开发人员通常会遵循以下编程实践方法：将具有不同功能性的代码置于不同的文件中，这使得一个项目通常包含许多不同的源代码和头文件。

人们从早期微处理器的开发环境出现时就已经开始采用这项编程实践。从那时起，开发人员就把这个编程实践作为一项开发过程中的固定设计方案，事实证明这样可以很好地组织代码，而且使得代码维护工作变得轻松许多。

这项实用的编程实践会对开发过程产生深远影响。在接下来的构建过程阶段中，你很快就会了解到这项实践会导致一些不可预测性，我们需要花些心思来解决这些问题。

**概念解释：基于演示项目**

为了更好地说明编译过程的复杂性，同时也让读者能够快速拥有一次动手实践的机会，本书提供了一个简单的演示项目。该项目的代码非常简单，只包含了一个头文件和两个源代码文件。虽说如此，但为了让读者理解整个过程，这个经过精心设计的演示项目也足以说明程序生命周期中的各个重点。

该项目由以下几个文件组成：

* 源代码文件main.c
* 头文件function.h，其中声明了main()函数中所调用的函数和使用的数据。
* 源代码文件function.c，其中包含的代码实现main()函数引用的函数，并初始化

main()函数引用的数据。

演示项目是在Linux环境中使用gcc编译器构建的。代码清单2-1到代码清单2-3列出了演示项目中用到的代码。

代码清单2-1 function.h

#pragma once

#define FIRST\_OPTION

#ifdef FIRST\_OPTION

#define MULTIPLIER(3.0)

#else

#define MULTIPLIER(2.0)#endif

float add\_and\_multiply(float a, float y);

代码清单2-2 function.c

int nCompletionStatus = 0;

float add(float x, float y)

{

float z = x + y;

}

float add\_and\_multiply(float x, float y)

{

float z = add(x,y);

z \*= MULTIPLIER;

return z;

}

代码清单2-3 main.c

#include “function.h”