第 3 部分

工 程 案 例

引　言

学习程序设计的目标之一是能进行软件开发，而实际的软件开发与普通的编程练习有不小的区别。简单的编程练习通常集中于某一种算法的解决、某一种数据结构的实现，而实际的软件开发则着眼于整个设计目标的实现。软件工程就是指导软件开发和维护的工程学科，它采用工程的原理、概念、技术和方法来开发和维护软件，把经过时间检验证明正确的管理技术和当前能采取的最好的技术方法结合起来，以经济地开发出高质量的软件并有效地维护它。

在这一部分，将通过 5 个不同的案例练习 C 语言在多种场景下的软件应用开发，读者可根据自己的兴趣选做其中的一个或多个案例，以提高自己的软件开发能力。这些案例虽然各自应用领域不同，但是它们的设计开发都要遵循软件工程设计的思想。只有遵循这些科学的设计方法和原则，才能高效地开发出适应性好的软件。本节介绍一些 C 语言工程开发的基本原则和方法，这些原则和方法也将在后续的案例中得到具体的应用和体现。

1. 工程开发基础

概括地说，软件工程是指导计算机软件开发和维护的一门工程学科。下面按照软件开发过程中的各阶段介绍其基本任务和常用方法。

1）需求分析

为了开发出真正满足用户需求的软件产品，首先必须知道用户的需求。

需求分析包含了功能需求、性能需求、可用性需求等各方面的综合要求。它的最基本的任务是准确地回答“系统必须做什么”这一问题，但不是确定系统怎样完成这项工作，而仅仅是确定系统必须完成哪些工作，也就是通过与用户的反复交流，对目标系统建立完整、准确、清晰、具体的需求。

2）总体设计

总体设计的基本目的是回答“概括地说，系统应该如何实现”这一问题，因此总体设计又称为概要设计或初步设计。通过这个阶段的工作将划分出组成系统的物理元素——程序、文件、数据库、人工过程和文档等。总体设计的另一项重要任务是设计软件的结构，也就是要确定系统中每个程序由哪些模块组成的以及这些模块相互之间的关系。

总体设计过程通常由两个主要阶段组成：一是系统设计阶段，要确定系统的具体实现方案；二是结构设计阶段，需要确定软件结构。软件结构通常情况下使用图形工具来描述，如层次图和结构图。

3）详细设计

详细设计阶段的根本目标是确定应该怎样实现所要求的系统，也就是说，经过这个阶段

的设计工作,应该得出对目标系统的精确描述,从而在编码阶段可以把这个描述直接翻译成某种程序设计语言书写的程序。

详细设计阶段的任务还不是具体地编写程序,而是要设计出程序的“蓝图”,之后程序员将根据这个蓝图写出实际的程序代码。因此,详细设计的结果基本上决定了最终的程序代码的质量。详细设计的目标不仅是逻辑上正确地实现每个模块的功能,更重要的是设计出的处理过程应该尽可能简明易懂。结构化程序设计技术是实现上述目标的关键技术,因此是详细设计的逻辑基础。

4) 编码

这个阶段的关键任务是写出正确的、容易理解、容易维护的程序模块。程序员应该根据目标系统的性质和实际环境,选取一种适合的程序设计语言,把详细设计的结果翻译成程序,并仔细调试编写出的每个模块。

编码时要注重编码规范,确保各模块的代码风格统一,并提高代码的可读性。本书主教材中已总结了标准的编码规范,请读者查看。

5) 测试

什么是测试? 它的目标是什么? 以下规则可以看作测试的目标或定义。

(1) 测试是为了发现程序中的错误而执行程序的过程。

(2) 好的测试方案是极可能发现迄今为止尚未发现的错误的测试方案。

(3) 成功的测试是发现了迄今为止尚未发现的错误的测试。

从上述规则可以看出,测试的正确性定义是“为了发现程序中的错误而执行程序的过程”。这和某些人通常想象的“测试是为了表明程序是正确的”和“成功的测试是没有发现错误的测试”等是完全相反的。正确认识测试的目标十分重要,测试目标决定了测试方案的设计。如果为了表明程序是正确的而进行测试就会设计一些不易暴露错误的测试方案;相反,如果测试是为了发现程序中的错误,就会力求设计出最能暴露错误的测试方案。

测试用例(test case)是为某个特殊目标而编写的一组测试输入、执行条件以及预期结果,以便测试某个程序路径或核实是否满足某个特定需求。测试用例设计和执行是测试工作的核心,也是工作量最大的任务之一。

在编写测试用例前,要详细了解需求并且准确理解软件所实现的功能,然后着手制定测试用例。测试数据应该选用少量高效的测试数据进行尽可能完备的测试。测试用例通常包括如下几方面的内容。

(1) 编号(测试用例的编号);

(2) 测试项(欲测试的功能);

(3) 测试输入(应输入的数据和相应的操作处理);

(4) 预期结果(预期的输出结果或其他响应效果);

(5) 测试结果(测试结论为“通过”或“不通过”)。

2. 多文件结构组织

在软件开发中,通常采用自顶向下的设计模式。所谓自顶向下就是从总问题开始,将其分解为一个一个小问题的解决方案。在设计之前,首先要考虑的是文件结构。什么是文件结构? 就是程序怎么通过文件组织起来。如果是简单的程序练习,一般只需要一个源文件

就可以实现，也就谈不上什么文件组织的问题。但如果是稍大型的程序，代码量达到几千行甚至上万行，就有必要通过文件对其进行组织了。想象一下，如果数千行的程序写在一个源文件中，会给程序的阅读和调试带来很大的麻烦，当程序员需要对一个问题进行定位的时候就需要上下翻动查找。而如果将程序分成多个文件，例如每个模块一个文件，程序员就可以很快地定位，同时给该程序的后续拓展带来极大的方便。

C 程序可以包括两种文件，一种用于保存程序的定义，即以“.c”作为后缀的源文件；另一种用于保存程序的声明，即以“.h”作为后缀的头文件。C 程序的每个模块都可以分别由一个源文件来实现，该文件里包括这个模块中所有功能函数的定义，同时每个源文件都可以配置一个对应的头文件，该文件里包含这个源文件所需的预处理内容以及函数声明。这样，当 A 模块需要调用 B 模块所定义的函数时，只需要在 A 模块的源文件中引用 B 模块的头文件即可。

1）头文件结构

头文件由如下 3 部分内容组成。

（1）头文件开头处的版权和版本声明；

（2）预处理；

（3）函数声明等。

版权和版本的声明一般位于头文件和源文件的开始处，必须以注释的形式呈现，其主要内容如下。

（1）版权信息；

（2）文件名称，标识符，摘要；

（3）当前版本号，作者/修改者，完成日期；

（4）版本历史信息。

下面是一个典型的版权和版本声明示例。

```
/*
* Copyright (C) xxx公司
* All rights reserved.
*
* 文件名称:filename.h
* 摘要:简要描述本文件的内容
*
* 当前版本:1.1
* 作者:作者(或修改者)名字
* 完成日期:2023年1月20日
*
* 取代版本:1.0
* 原作者:原作者(或修改者)名字
* 完成日期:2022年1月20日
*/
```

实际应用中，版权和版本声明可根据需要自行增减，这部分内容实际上也是为了给程序的后续修改者提供方便。

接下来，头文件的真正有效内容包含预处理部分和数据及函数声明部分，这部分内容有

几个编写原则需要遵守,列举如下。

(1) 为了防止头文件被重复引用,应当用 ifndef/define/endif 结构产生预处理结构。

(2) 用#include <filename.h> 格式来引用标准库的头文件(编译器将从标准库目录开始搜索)。

(3) 用#include "filename.h" 格式来引用非标准库的头文件(编译器将从用户的工作目录开始搜索)。

(4) 头文件中只存放"声明"而不存放"定义"。

上述原则中,(2)和(3)很好理解,即如果引用的是类似于"stdio.h"这样的标准库头文件,就采用尖括号的形式,如果是自定义的头文件,就采用双引号的形式,这样做是为了加快编译的搜索速度。原则(4)的意思是不应该在头文件里具体定义一个变量或是函数,换句话说,头文件里不应该产生真正占用内存的数据和程序,而应该仅仅是声明。下面通过示例解释原则(1)。

```
#ifndef TYPESET_H                          //防止 typeset.h 被重复引用
#define TYPESET_H

#include<math.h>                           //引用标准库的头文件
…
#include "myheader.h"                      //引用非标准库的头文件
…
void Function1(…);                         //函数声明
…

#endif
```

原则(1)的最大作用就是防止一个头文件被重复引用,如 A.c 引用了 B.h 和 C.h,而 B.h 同样引用了 C.h,如果没有防护机制,就会造成 C.h 的重复引用。如示例所示,采用条件编译机制即可防止该问题的出现,这里 TYPESET_H 是该头文件的一个唯一标识,当然这个标识的名字是自定义的,约定俗成采用和头文件一样的名字,不过把小写字符全部替换成大写字符,点换成下画线。条件编译的作用是,如果该文件在编译器第一次编译时未定义这个唯一的标识,就会定义它,同时编译这个头文件,当编译器再次遇到该头文件时,因为已经定义过该标识,就不会再次参与编译了。这就保证了每个头文件最多只能编译一次。

2) 源文件结构

源文件的内容包含如下 3 部分。

(1) 文件开头处的版权和版本声明;

(2) 头文件引用;

(3) 程序实现(包含定义数据和函数)。

源文件结构的示例如下。

```
//版权和版本声明
#include<stdio.h>
#include "typeset.h"                       //引用头文件
…
```

```
//全局变量定义
int value;
…
//函数定义
void Function1(…)
{
…
}
```

需要说明的是，在编写程序时的一个原则是：不要定义不必要的全局变量。但有时候，可能产生这样一种情形，即多个模块的函数都需要使用同一个数据，这时采用全局变量可以极大地提高程序的效率。全局变量理论上可以定义在任意一个源文件中，但通常的做法是：把所有的全局变量都定义在主函数所在的文件，当其他模块的文件需要使用该全局变量时，在其源文件中使用 extern 关键字对该全局变量进行声明。

3）文件组织

下面以一个小例子说明 C 语言的多文件组织结构。假设要开发一个包含输入、计算、输出功能的小系统，就可以将这 3 个功能分别组织成一对 C 语言的源文件和头文件，如图 3-0-1 所示。

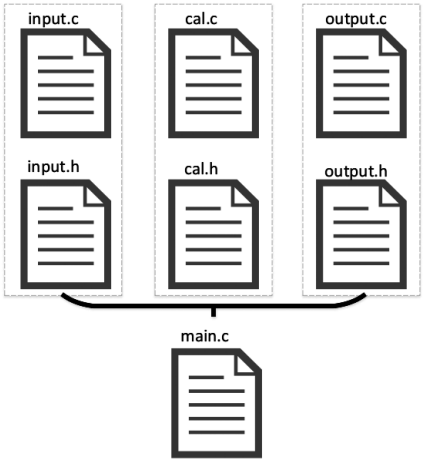

图 3-0-1　多文件组织示例

在命名文件的时候要注意，和命名一个变量或一个函数相似，文件的命名也要做到见名知义，这样软件的读者在浏览源代码文件结构时就基本可以看出软件设计者的模块划分结构。在实际的软件开发中，文件的组织与模块的划分息息相关。模块划分合理、文件组织清晰会给后续的开发和调试带来极大的益处。

3. 案例学习指南

本节介绍在案例学习中需要注意的一些问题。本部分的全部案例都将按照案例介绍、详细设计、系统测试与总结 3 部分展开。

在案例介绍部分，会针对案例的应用背景、设计目的、需求分析以及总体设计进行介绍。在阅读完案例介绍部分后，则可以进入案例的详细设计，展开具体的案例实验。

考虑到一个完整的工程案例通常会使用包括函数、指针、结构体、文件在内的语法,为了使读者在学习 C 语言的过程中尽早接触工程案例设计,第二部分详细设计对每个案例都划分为 3 个阶段。读者可以随着学习 C 语言的过程,由浅入深、由易到难地分阶段进行案例学习。详细设计中的 3 个阶段对应的语法要求通常按学习顺序展开如下。

阶段一:通常要求学习完循环结构和数组;

阶段二:通常要求学习完函数;

阶段三:通常要求学习完结构体和文件。

其中,每个阶段都是一个完整的程序。每个阶段都是对前一个阶段的完善和深化,到第三个阶段则完整实现了工程案例的全部功能。每个阶段的末尾都附有该阶段所对应的完整程序电子链接,读者可以扫描二维码下载查看。

最后一部分是系统测试与总结,读者可按照提示对编写好的案例程序进行测试,而后可进一步对程序进行拓展,使其功能更为完善和丰富。

案例1　工程入门实例——扫雷

1.1　案例介绍

扫雷游戏，是微软公司于1992年，伴随着当时Windows系统一同发布的一款大众益智类游戏，这个基于数字逻辑的谜题游戏多年来一直深受用户喜爱，经久不衰。下面就让我们利用所学的程序设计的知识，自己动手设计、编写一款控制台界面下的扫雷游戏。

1.1.1　设计目的

扫雷游戏设计的初衷是基于控制台界面，在一个9×9(初级)大小的方块矩阵中随机布置一定数量的地雷(地雷个数为10个)，然后由玩家逐个翻开方块，以找出所有地雷为最终游戏目标。如果玩家翻开的方块处是地雷，则地雷爆炸，游戏结束。

1.1.2　需求分析

扫雷游戏是一款众所周知的游戏，这个案例的题目看似简单，实际上在编写程序之前同样需要考虑方方面面的需求，即这个程序都要实现哪些功能。本案例的目标是在控制台下(即命令行窗口)实现一个简略版的扫雷游戏，可以参考Windows系统自带的扫雷桌面游戏来给出此案例的功能需求。

(1) 在控制台下实现经典的扫雷游戏；

(2) 游戏的开始和结束通过菜单完成；

(3) 游戏初始界面为一个9×9的正方形；

(4) 游戏中地雷的个数以及初始位置设置；

(5) 通过输入坐标的方式进行挖雷操作。

以上是根据设计要求初步想到的基本需求，从需求分析的角度出发，开发人员在进行软件开发前，需要经过深入、细致的分析和调研，以便准确地理解用户和项目的功能、性能、可靠性等具体要求，从而将用户非形式的需求表述转换为完整的需求定义，也就是需要确定系统必须做什么的过程。在实际的软件开发过程中，需求有可能是动态变化的，可能会加入新的需求或者原有的需求会发生改变，这必然会给软件开发带来一定的困难，同时也要求设计者在设计的时候要尽量遵循设计规范，使软件更具适应性。

1.1.3 总体设计

首先,在编写程序之前,要从整体上将程序进行模块化,即把软件拆分成若干功能模块。这些功能模块功能相对独立,彼此之间联系较少,耦合在一起之后能够实现软件的全部需求。针对扫雷游戏的具体功能,将其划分为 3 个模块,如图 3-1-1 所示。

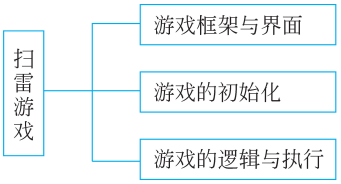

图 3-1-1 扫雷游戏功能模块

(1) 游戏框架与界面模块。此模块的基本功能是实现游戏的基本框架。对于一个游戏而言,当游戏启动时,需要有一个初始化界面(即游戏的初始化菜单),用户可根据菜单中提供的选项,选择开始游戏或者退出游戏。因此,该功能模块理应在主函数内实现,可在主函数内调用相关函数,完成具体的功能。

(2) 游戏的初始化模块。该模块的基本功能是对游戏数据进行初始化。在扫雷游戏中,在每次游戏开始时,需要提前初始化的设置有很多,例如,需要给定一个游戏的界面,又或者需要在游戏界面的范围内进行布雷等。对于一个控制台程序而言,可以使用一些符号来进行图形界面的初始化,同时通过一些函数的编写来实现布雷等具体功能。

(3) 游戏的逻辑与执行模块。对于一个简化版的扫雷游戏而言,其最主要的功能就是雷的排查,以及以当前区域为中心点周围所存在的雷的数量的计算,这些具体的功能是本游戏的核心功能,需要用户自定义函数将功能进行独立封装,以方便调用。

1.2 详细设计

为适应学习者的学习进度,让学习者尽早地进入 C 语言实际应用案例的开发与设计,本节将此案例的开发过程拆分成多个阶段,当学习者处于不同的学习阶段时,可以选择案例的对应部分进行学习。

1.2.1 游戏初始化界面

编程能力要求:熟悉输入输出、分支结构、循环结构和数组。

本阶段的主要目的是游戏初始界面的设计。在整体的界面设计过程中,包含了 3 个具体步骤。步骤 1 为游戏菜单界面的实现;步骤 2 为开始或者退出游戏选项的实现;步骤 3 为游戏棋盘的初始化设计。这 3 个步骤以基本输入输出为核心,学习者在掌握了输入输出的基本知识后则可开展步骤 1 的练习,在学习完成分支与循环结构的知识后可开展步骤 2 的练习,在掌握了数组的知识后,则可开展步骤 3 的练习。各步骤的具体实现如下。

步骤 1,能力要求:基本运算与输入输出。

步骤 1 的目的是实现游戏初始界面的设计,学习者只需完成“输入输出”部分的学习,就可以开展本步骤的练习。在控制台下的扫雷游戏中,需要一个菜单界面来确定是否开始游戏。菜单的具体界面可以由 * 组成,辅之以简单的选项介绍。在没有学习自定义函数知识