# 实验 1 实验环境的使用

**实验难度：**★★☆☆☆
**建议学时：**2 学时

## 一、实验目的

- 熟悉实验环境的基本使用方法。
- 掌握正则表达式和 NFA（非确定有穷自动机）的含义。
- 实现正则表达式到 NFA 的转换。

## 二、预备知识

- 在这个实验中 NFA 状态结构体使用了类似于二叉树的数据结构，还包括了单链表插入操作以及栈的一些基本操作。如果读者对这一部分知识有遗忘，可以复习一下数据结构中的相关内容。
- 实验中需要把正则表达式转换为 NFA（非确定有穷自动机），所以要对正则表达式和 NFA 有初步的理解。读者可以参考配套的《编译原理》教材，学习这一部分内容。

## 三、实验内容

由于读者是第一次进行编译原理实验，所以在此文档中会首先介绍实验环境的基本使用方法，然后再引导读者完成正则表达式转换为 NFA 的实验内容。

编译原理实验所使用的实验环境主要包括在线课程管理平台和 VSCode 编程环境。请读者按照下面的步骤完成实验内容，同时，仔细体会在线课程管理平台和 VSCode 编程环境的基本使用方法。在本实验题目中，操作步骤会编写的尽量详细，并会对在线课程管理平台和 VSCode 编程环境的核心功能进行具体说明。但是，在后面的实验题目中会尽量省略这些内容，而将重点放在实验相关的源代码上。如有必要，读者可以回到本实验题目中，参考在线课程管理平台和 VSCode 编程环境的基本使用方法。

### 3.1 VSCode 的基本使用方法

#### 安装和启动 VSCode

本书充分利用了 VSCode 高度可定制的特性，为编译原理实验专门制作了一个 VSCode 编程环境。该 VSCode 编程环境中已经集成了 Make 构建工具、GCC 编译器、GDB 调试器、Flex 词法分析生成工具和 Bison 语法分析生成工具等，同时包含了一些有用的 VSCode 插件，目的就是免去读者手工构建实验环境所带来的学习成本，使读者可以将主要精力放在对编译原理的分析与理解上。

读者首先需要下载与本书配套的 VSCode 压缩包文件，然后将其解压缩到 64 位 Windows 7 或 Windows 10 本地磁盘的一个目录中（例如 D:\vscode，目录路径不要包含中文字符或者空格），无需修改环境变量，也无需进行任何安装操作。在启动 VSCode 之前，读者还需要单独安装 Python 语言解释器（3.8 及以上版本）和 Git 客户端软件（2.18 及以上版本）。

最后，读者就可以双击 D:\vscode\Code.exe 文件启动 VSCode 了。

#### VSCode 的窗口布局

VSCode 的窗口布局由下面的若干元素组成：

- 编辑器：这是主要的代码编辑区域，可以多列或者多行的打开多个编辑器。

- 侧边栏：位于左侧的侧边栏包含了文件资源管理器、文件搜索、源代码版本管理、调试与运行、插件等基本视图。
- 活动栏：位于侧边栏的左侧，可以方便的让用户在不同的视图之间进行切换。
- 状态栏：位于底部的状态栏用于显示当前打开文件的光标位置、编码格式等信息。
- 面板：编辑器的下方可以展示不同的面板，包括显示输出信息的面板、显示调试信息的面板、显示错误信息的面板和集成终端。面板也可以被移动到编辑器的右侧。

**使用浏览器从在线课程平台领取任务**

在线课程平台是用于教师在线布置实验任务，并统一管理学生提交的作业的平台。

1. 读者通过浏览器访问平台，可以打开平台的登录页面。注意，如果读者使用的是校园网或局域网中的平台，需要从平台管理员处获得 URL。
2. 在登录页面中，读者输入用户名和密码，点击“登录”按钮，可以登录平台。
3. 登录成功后，在“课程”列表页面中，可以找到编译原理对应的实验课程。点击此课程的链接，可以进入该课程的详细信息页面。
4. 在课程的详细信息页面中，可以查看课程描述信息，该信息对于完成实验十分重要，建议读者认真阅读。
5. 点击左侧导航中的“任务”链接，可以打开任务列表页面。
6. 在任务列表中找到本次实验对应的任务，点击右侧的“领取任务”按钮，可以进入“领取任务”页面。
7. 在“领取任务”页面填写“新建项目名称”和“新建项目路径”，然后选择“项目所在的群组”，点击“领取任务”按钮后，可以创建个人项目用于完成本次实验，并自动跳转到该项目所在的页面。

**提示：**在“领取任务”页面中，“新建项目名称”和“新建项目路径”这两项的内容可以使用默认值，如果选择的群组中已经包含了名称或者路径相同的项目，就会导致领取任务失败，此时需要修改新建项目名称和新建项目路径的内容。

8. 在新建的个人项目页面中，包括左侧的导航栏、项目信息、文件列表等，如图 1-1 所示。
9. 点击图 1-1 红色方框中的按钮，可以复制个人项目的 URL，在本实验后面的练习中会使用这个 URL 将此项目克隆到读者计算机的本地磁盘中。

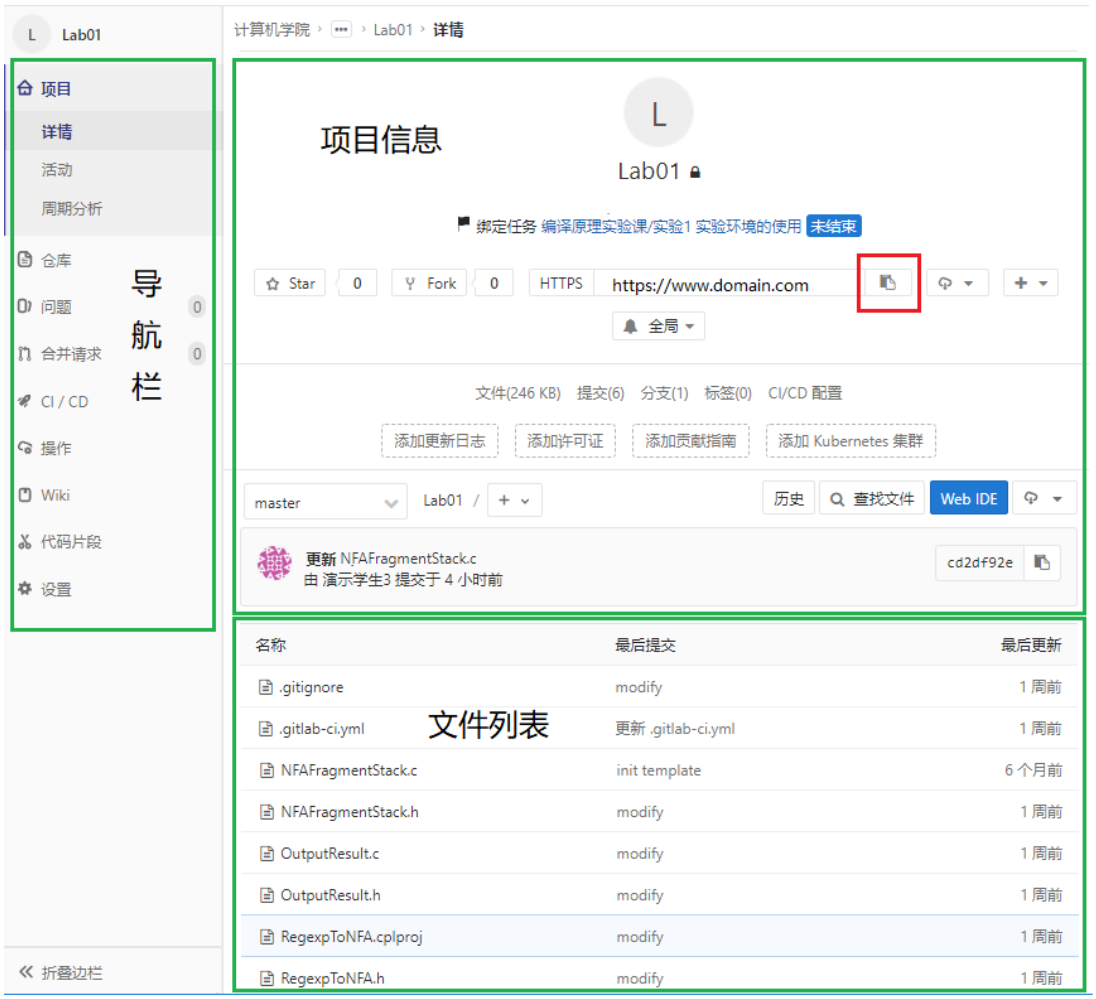


图 1-1：领取任务后得到的个人项目页面

**使用 VSCode 将项目克隆到本地磁盘**

在读者从平台领取了任务之后，可以按照下面的步骤将个人项目克隆到本地磁盘中。

1. 在 VSCode 的“View”菜单中选择“Command Palette...”，会 在 VSCode 的顶部中间位置显示一个用来输入命令的面板。
2. 在 VSCode 的命令面板中输入“Git”后，会在 列表中提示出所有与 Git 相关的命令。选择列表中的“Git: Clone”命令后按回车，会提示输入 Git 远程库的 URL，将之前复制的个人项目的 URL 粘贴到命令面板中并按回车。
3. Git 远程库的 URL 输入成功后，会自动打开“选择文件夹”窗口，提示用户选择一个本地文件夹用来保存项目。此时，读者就可以在本地磁盘中选择一个合适的文件夹（注意，本地文件夹路径中不要包含中文字符和空格），然后点击“Select Repository Location”按钮。

**注意：**如果在选择文件夹后，弹出了 Windows 安全中心的凭据登录窗口，读者可以选择其中的“取消”按钮，跳过此步骤。这里不建议读者使用 Windows 凭据管理 Git 远程库的账号信息，一方面是由于一旦读者把用户名密码输入错误后，需要去 Windows 控制面板中的凭据管理器修改、或者删除凭据，比较麻烦；另外一方面，如果读者是在一台公共电脑上操作的话，使用 Windows 凭据管理器保存登录的用户名和密码是非常危险的。

4. 选择本地文件夹后，VSCode 会在命令面板中提示输入“Username”，输入平台的用户名后按回车。接下来会提示输入“Password”，输入平台的密码后按回车。用户名和密码校验成功后就开始将 Git 远程库克隆到本地了。

5. 克隆成功后，会在 VSCode 的右下角弹出克隆完成提示框，点击其中的“open”按钮会使用 VSCode 打开克隆到本地的项目。

**使用 VSCode 登录平台**

使用 VSCode 打开项目后，VSCode 会自动在顶部中间位置弹出登录平台的提示框，只有登录平台后才能继续使用为编译原理实验定制的功能。

登录平台的步骤如下：

1. 首先，需要在 VSCode 顶部弹出的列表中选择平台 URL。如果是第一次登录平台，URL 列表应该是空的，此时就需要选择列表中手动输入平台 URL 的那一项。
2. 然后，在弹出的编辑框中根据提示信息输入平台 URL 后按回车。平台 URL 的格式类似于 http://www.domain.com，具体使用哪个 URL 可以咨询平台管理员。
3. 最后，按照提示依次输入平台的用户名和密码即可完成登录。

**查看项目中的文件**

将项目克隆到本地磁盘后，在 VSCode 左侧的“文件资源管理器”窗口中可以查看项目包含的所有文件夹和源代码文件。也可以使用 Windows 资源管理器打开项目所在的文件夹，方法是在“文件资源管理器”窗口中的任意一个文件夹或文件节点上点击右键，然后在弹出的快捷菜单中选择“Reveal in File Explorer”。

**3.2 阅读实验源代码**

该实验主要包含了三个头文件 RegexpToNFA.h、RegexpToPost.h、NFAFragmentStack.h 和三个 C 源文件 main.c、RegexpToPost.c、NFAFragmentStack.c。

下面对这些文件的主要内容、结构和作用进行说明：

**main.c**

在“EXPLORER”窗口中双击“main.c”打开此文件。此文件主要包含了以下内容：

1. 在文件的开始位置，使用预处理命令包含了 RegexpToNFA.h、RegexpToPost.h 和 NFAFragmentStack.h 文件。
2. 定义了 main 函数。在其中实现了栈的初始化。然后，调用了 re2post 函数，将正则表达式转换到解析树的后序序列。最后，调用 post2nfa 函数，将解析树的后序序列转换到 NFA。
3. 在 main 函数的后面，定义了函数 CreateNFAState 和 MakeNFAFragment，这两个函数分别是用来创建一个新的 NFA 状态和构造一个新的 Fragment。接着定义了函数 post2nfa，关于此函数的功能、参数和返回值，可以参见其注释。在这个函数中用‘$’表示空转换，此函数的函数体还不完整，留给读者完成。

**RegexpToPost.c**

1. 在文件的开始位置，使用预处理命令包含了 RegexpToPost.h 文件。
2. 定义了 re2post 函数，此函数主要功能是将正则表达式转换成为解析树的后序序列形式。

**NFAFragmentStack.c**

1. 在文件的开始位置，使用预处理命令包含了 NFAFragmentStack.h 文件。
2. 定义了与栈相关的操作函数。在构造 NFA 的过程中，这个栈主要用来放置 NFA 片段。

**RegexpToNFA.h**

1. 包含用到的 C 标准库头文件。目前只包含了标准输入输出头文件“stdio.h”。
2. 包含其他模块的头文件。目前没有其他模块的头文件需要被包含。
3. 定义了与 NFA 相关的数据结构，包括 NFA 状态 NFAState 和 NFA 片段 NFAFragment。具体内容可参见下面的两个表格。

| NFAState 的域 | 说明 |
|---|---|
| Transform | 状态间转换的标识。用 '$' 表示 'ε-转换'。 |
| Next1 和 Next2 | 用于指向下一个状态。由于一个 NFA 状态可以存在多个转换，而在本程序中使用的是类似于二叉树的储存结构，每一个状态最多只有两个转换，所以，这里定义两个指针就足够了。当 NFA 只有一个转换时，优先使用 Next1，Next2 赋值为 NULL。 |
| Name | 状态名称。使用整数表示(从 1 开始)，根据调用 CreateNFAState 函数的顺序依次增加。 |
| AcceptFlag | 是否为接受状态的标志。1 表示是接受状态 0 表示非接受状态。 |

| NFAFragment 的域 | 说明 |
|---|---|
| StartState | NFAFragment 的开始状态。 |
| AcceptState | NFAFragment 的接受状态。在构造 NFA 的过程中总是在 NFA 的开始状态和接受状态上进行操作，所以用开始状态和接受状态表示一个 NFA 片段就足够了。 |

4. 声明函数和全局变量。

RegexpToPost.h

1. 包含其他模块的头文件。目前只包含了头文件“RegexpToNFA.h”。
2. 声明函数。为了使程序模块化，所以将 re2post 函数声明包含在一个头文件中再将此头文件包含到“main.c”中。

NFAFragmentStack.h

1. 包含其他模块的头文件。目前只包含了头文件“RegexpToNFA.h”。
2. 定义重要的数据结构。定义了与栈相关的数据结构。
3. 声明函数。声明了与栈相关的操作函数。

**其它文件**

除了以上这些 C 语言源代码文件外，读者也需要了解一下其它重要文件的内容和作用。

| 文件名 | 说明 |
|---|---|
| makefile | Make 构建工具使用此文件中的脚本调用 GCC 编译器将 C 语言源代码文件编译为可执行文件。 |
| input1.txt~input8.txt | 在每个文件中都保存了一个正则表达式，序号越大，正则表达式越复杂。在运行程序时，会将其中一个文件的内容重定向到标准输入，从而可以在程序中从标准输入读取到一个正则表达式，然后将其转换为 NFA。 |
| output1.txt~output8.txt | 在每个文件中都保存了一个 NFA 的文本描述信息，作为标准答案，并且与相同序号的 input 文件中的正则表达式是对应的。在自动化验证时，程序会将 NFA 的文本描述信息打印到标准输出，而标准输出会重定向到 user_output1.txt~user_output8.txt 文件中，然后与相同序号的 output 文件比较，如果两个文件的内容相同，就说明验证成功，否则，验证失败。 |

### 3.3 演示程序的执行过程

这里介绍的一个重要特性就是，在还没有为 main.c 文件编写源代码的情况下，读者就可以通过演示功能，了解到程序应该具有的功能和执行流程，并通过数据可视化窗口观察到

程序在运行的过程中内存中数据的变化效果，从而帮助读者正确实现 main.c 文件中的代码。

在项目中提供了一个文件 main.demo（只读文件），该文件中的内容与 main.c 文件中的内容类似。但是，main.demo 文件中的 post2nfa 函数并不是空的，而是使用简洁、直观的伪代码进行了描述（一行伪代码可能会对应多行 C 源代码），偶尔也会在读者理解起来比较困难的地方直接提供 C 源代码，目的就是为了让读者可以在 main.c 文件中顺利的编写出正确的源代码。

**演示程序的配置**

在读者观察演示程序的执行过程之前，有必要先了解一下当前项目是如何利用 launch.json 文件配置 VSCode 中的调试功能的，这些内容可以帮助读者更好的理解 VSCode 的调试功能以及演示程序运行的原理。

首先，在 VSCode 左侧的活动栏中点击 Run and Debug 按钮打开 RUN AND DEBUG 窗口，然后，点击其顶部的下拉列表，默认选中的是“Demo input1”这一项，表示把 input1.txt 文件中的内容重定向到标准输入，并调试已经预先构建好的演示程序的可执行文件 demo.exe，并以此类推。而“input1”这一项表示把 input1.txt 文件中的内容重定向到标准输入，并调试读者编写的应用程序可执行文件 app.exe，并以此类推。

为了更好的理解 VSCode 调试功能的配置信息，读者可以在 VSCode 的文件资源管理器中双击打开.vscode/launch.json 文件，该文件中的内容就是用来设定 VSCode 调试功能的。读者可以在此文件中看到有一个顶层的 configurations 数组，其后的中括号中包含了所有的数组元素，而每个数组元素（在大括号中）就对应了之前下拉列表中的一项。在 name 为 Demo input1 的数组元素中，将 program 设置为 demo.exe，表示调试的可执行文件是已经预先构建好的演示程序的可执行文件 demo.exe，将 args 数组设置为两个元素，分别为<和 input1.txt，这样在启动调试时，就会向调试器发送一个命令行“demo.exe < input1.txt”，表示将文件 input1.txt 中的内容重定向到 demo.exe 的标准输入。类似的，在 name 为 input1 的数组元素中，将 program 设置为 app.exe，表示调试的可执行文件是读者编写的应用程序可执行文件 app.exe，将 args 数组设置为两个元素，分别为<和 input1.txt，这样在启动调试时，就会向调试器发送一个命令行“app.exe < input1.txt”，表示将文件 input1.txt 中的内容重定向到 app.exe 的标准输入。

**调试演示程序**

1. 在 VSCode 左侧的活动栏中点击 Run and Debug 按钮，可以打开 RUN AND DEBUG 窗口，在顶部的列表框中选择“Demo input1”这一项。
2. 在 Run 菜单中选择 Start Debugging 菜单项启动调试（快捷键是 F5）。
3. 在专门为展示演示程序执行流程的 main.demo 文件中（不是 main.c 文件）的 main 函数里面中断执行了。黄色箭头指向中断的代码行，中断的代码行表示该行代码还没有执行，也就是说该行代码是下一步将要执行的代码。
4. 在 VSCode 右侧自动打开了“数据可视化”窗口，使用图形化的方式显示出了内存中的数据。此时，由于还没有将正则表达式转换为 NFA，所以在可视化窗口中只是显示了当前程序在 main 函数的开始位置使用 scanf 函数从标准输入读取到的正则表达式“ab”。此正则表达式与 input1.txt 文件中的内容是一样的。

**注意**，在调试演示程序的过程中，main.demo 文件中包含了很多隐藏的断点，所以读者并不需要自己添加断点。在后续的演示过程中，读者只需要按 F5，让演示程序在 main.demo 的隐藏断点依次中断，就可以获得很好的演示效果。但是，由于没有在 main.demo 文件中的所有代码行添加隐藏断点，所以在演示的过程中并不会在每一行代码处都中断执行。

5. 按 F5 继续调试，进入 post2nfa 函数，并自动刷新了可视化窗口的内容。在可视化

窗口中可以使用鼠标左键拖动图形，也可以使用鼠标滚轮缩放图形。

6. 继续按 F5 调试，每执行 main.demo 中的一行伪代码，可视化窗口中的内容就会发生相应的变化，例如构造单字符的 NFA 片段，构造连接 NFA 片段等，读者需要结合伪代码和可视化窗口中的内容理解正则表达式转换为 NFA 的过程。
7. 当演示程序运行完毕后，会在 VSCode 底部的 TERMINAL 窗口中使用文本描述的方式打印输出 NFA。读者可以对照可视化视图中的 NFA 图形和 OutputResult 函数来理解文本描述方式的 NFA。
8. 如果需要结束调试，可以选择“Run”菜单的 “Stop Debugging”（快捷键 Ctrl+F5）。

**切换可视化窗口中的图号**

当可视化窗口中包含多个图形时，在图形的顶部会显示出包含了哪些图形，正在显示哪个图形，以及每个图形的图号（#1 表示 1 号图，#2 表示 2 号图）。可视化窗口默认总是显示 1 号图的内容，当读者需要查看 2 号图时，在可视化窗口顶部的编辑框中输入“#2”后按回车即可。

在本实验中 1 号图显示的是正则表达式和 NFA 相关的内容，2 号图显示的是正则表达式的解析树，由于正则表达式的解析树在调试的过程中并不会发生变化，所以只需在开始调试时查看 2 号图中解析树的内容即可，读者应该重点关注 1 号图的变化。

**让演示程序直接运行到最后的结果**

前面已经带领读者调试了配置名称为“Demo input1”的演示程序，读者也可以在 RUN AND DEBUG 窗口顶部的下拉列表中选择其它的演示程序配置，从而调试不同的正则表达式转换为 NFA 的过程。

后面的算例提供的正则表达式比较复杂（例如 input8.txt 文件中的正则表达式），读者可以按 F5 逐步的调试演示程序执行的过程，详细研究每一个 NFA 片段的构建过程。但是由于逐步调试的步骤比较多，如果读者急于看到最终的 NFA，可以选择“Demo toend”这一项，默认使用 input1.txt 文件的内容作为标准输入，并直接运行到演示程序结束的位置，这样读者就可以很快看到最终的 NFA 了。如果要查看其它正则表达式转换的 NFA，可以在.vscode/lauch.json 文件中找到 name 为 Demo toend 的元素，将其 args 属性值中的 input1.txt 更改为 input2.txt，这样就可以调试算例 2。以此类推，可以调试其它的算例。

### 3.4 编程实现正则表达式转换为 NFA

**生成项目**

项目中提供了一个 makefile 文件。Make 工具就是使用 makefile 文件中的脚本将源代码生成为可执行文件的。由于篇幅的限制，本书没有详细说明 makefile 文件的内容，请读者自行学习 makefile 文件的内容。

生成项目的方法是，在 VSCode 的“Terminal”菜单中选择“Run Build Task...”，会在 VSCode 的顶部中间位置弹出一个可以执行的 Task 列表，选择其中的“生成项目”即可。在项目生成的过程中，位于编辑器下方的“TERMINAL”窗口会实时显示生成的进度和结果。

生成项目的过程，就是将项目中包含的每个 C 源代码文件（.c 文件）编译为一个对象文件（.o 文件），然后再将所有对象文件链接为一个可执行文件（.exe 文件）的过程。以本实验为例，成功生成项目后，默认会在本项目的目录下生成对象文件 main.o、RegexpToPost.o 和 NFAFragmentStack.o，以及可执行文件 app.exe。请读者在 VSCode 左侧的文件资源管理器中找到这些刚刚生成的对象文件和可执行文件。

**解决语法错误**

如果在源代码中存在语法错误，在生成项目的过程中，“TERMINAL”窗口会显示相应的错误信息（包括错误所在文件的路径，错误在文件中的位置，以及错误原因）。此时，在 VSCode 底部的“PROBLEMS”窗口中双击错误信息所在的行，VSCode 会使用源代码编辑器自动打开

错误所在的文件，并定位到错误所在的代码行。

可以按照下面的步骤进行练习解决语法错误：

1. 在源代码文件中故意输入一些错误的代码（例如删除一个代码行结尾的分号）。
2. 生成项目。
3. 在“PROBLEMS”窗口中双击错误信息来定位存在错误的代码行，并将代码修改正确。
4. 重复步骤 2、3，直到项目生成成功。

**自动化验证（失败）**

之前提到了 main.c 文件中的 post2nfa 函数还不完整，是留给读者完成的。而当读者完成此函数后，往往需要使用调试功能、或者执行功能，来判断所完成的程序是否能够达到预期的效果。但是，由于本项目提供了 8 个算例，读者手动验证每个算例的过程就会比较麻烦，所以本项目提供了自动化验证功能（后续的编译原理实验也都提供了自动化验证功能），该功能可以自动化的、精确的完成所有算例的验证过程。

运行自动化验证时，会依次运行所有的算例。首先，使用命令行

```
app.exe < input1.txt > user_output1.txt
```

运行第一个算例（Case1），也就是将 input1.txt 中的内容重定向到标准输入，这样应用程序就会将 input1.txt 中的正则表达式转换为 NFA，并将 NFA 的文本描述打印到标准输出，由于在命令行中将标准输出重定向到了 user_output1.txt 文件，所以 NFA 的文本描述信息就被保存到了 user_output1.txt 文件中。最后，将第一个算例的标准答案文件 output1.txt 与 user_output1.txt 进行文本比较，如果两个文本文件中的内容一致，说明读者编写的应用程序可以将 input1.txt 文件中的正则表达式转换为正确的 NFA，如果两个文本文件中的内容不一致，说明读者编写的应用程序存在错误。验证其它算例的过程也是类似的。

按照下面的步骤运行自动化验证功能：

1. 在 VSCode 的“Terminal”菜单中选择“Run Build Tasks...”（快捷键是 Ctrl+Shift+B），在弹出的下拉列表中选择“测试”。
2. 在验证过程中，VSCode 底部的“TERMINAL”窗口会显示验证的过程和最后的结果。
3. 由于 post2nfa 函数还不完整，所以会导致验证失败。此时，会使用浏览器自动打开 result_comparation.html 文件，在其中显示了标准答案文件与读者编写的应用程序产生的结果文件的不同之处，从而帮助读者准确定位失败的原因。

**实现 post2nfa 函数**

读者已经通过调试演示程序详细了解了正则表达式转换为 NFA 的过程，接下来，打开 main.c 文件，编写 post2nfa 函数的源代码，使之可以正常处理所有算例中的正则表达式。

**提示：**

- 在 post2nfa 函数中已经给出了构造单字符 NFA 片段和连接 NFA 片段的源代码，这两步操作可以完成对 input1.txt 中正则表达式到 NFA 的转换（转换后的 NFA 如图 1-4 所示），读者可以在此基础上完成其他形式 NFA 片段的构造。
- 对于问号的 NFA 片段的构造（如图 1-7 所示），原则上也可以将状态 1 作为开始状态，状态 2 作为接受状态，并通过 ε 转换来表达接受状态为空的情况。但是如果存在一个或多个 NFA 片段与问号 NFA 片段相连接的情况，对于上述的情况就不适用了。因为在本程序中使用的是类似于二叉树的存储结构，一个状态最多只有两个转换指针，所以，必须在原来的基础上再添加两个状态作为开始状态和接受状态。

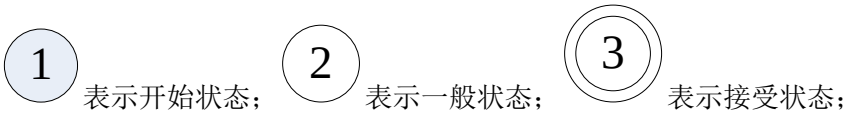


图 1-2：NFA 状态图例。

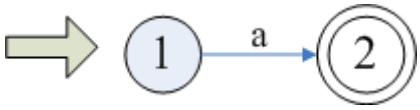


图 1-3：表示单字符的 NFA 片段。

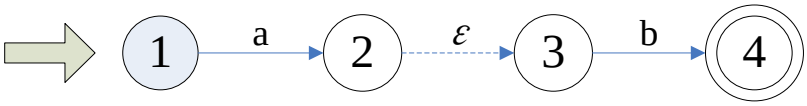


图 1-4：表示连接的 NFA 片段（对应 input1.txt 中的正则表达式）。

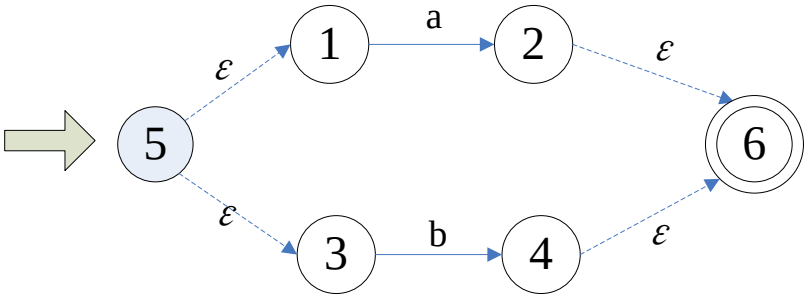


图 1-5：表示选择的 NFA 片段（对应 input2.txt 中的正则表达式）。

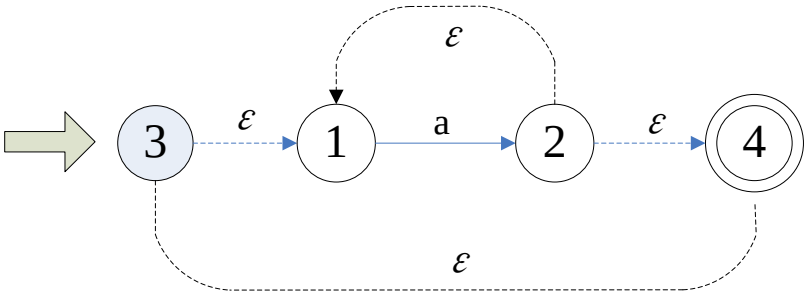


图 1-6：表示星号的 NFA 片段（对应 input3.txt 中的正则表达式）。

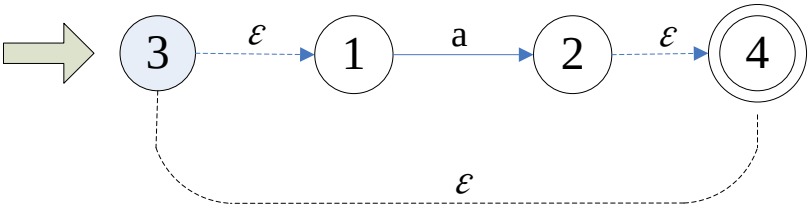


图 1-7：表示问号的 NFA 片段（对应 input4.txt 中的正则表达式）。

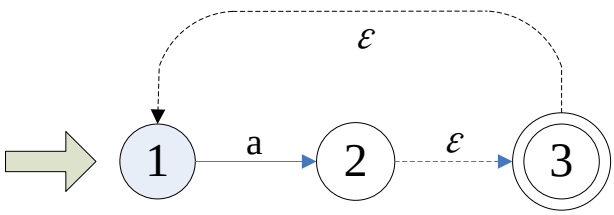


图 1-8：表示加号的 NFA 片段（对应 input5.txt 中的正则表达式）

**运行项目**

当读者为 post2nfa 函数编写完源代码，并且成功生成可执行文件后，就可以按照下面的步骤运行项目，查看得到的 NFA 是否正确：

1. 在 VSCode 左侧的活动栏中点击 Run and Debug 按钮，可以打开 RUN AND DEBUG 窗口，在其顶部的列表框中选择 input1 这一项。这样，在运行程序的时候，就会将 input1.txt 文件中的内容重定向到标准输入，从而将该文件中的正则表达式转换为 NFA。
2. 在“RUN”菜单中选择“Run Without Debugging”（快捷键是 Ctrl+F5），VSCode 就会开始运行 app.exe，并会在 VSCode 底部的“TERMINAL”窗口中打印输出文本描述的 NFA。
3. 如果读者编写的程序可以正常处理 input1.txt 文件中的正则表达式后，还可以在 RUN AND DEBUG 窗口顶部的列表框中选择 input2~input7 这些项，然后运行项目，查看从文件中输入的正则表达式生成的 NFA 是否正确。

**调试项目**

如果读者发现程序运行的结果与预想的结果不一致，或者程序存在异常行为（例如死循环、数组越界、访问野指针导致崩溃等情况），说明读者编写的源代码中存在逻辑错误（不是语法错误）。此时，读者可以使用调试功能查找逻辑错误产生的原因。

VSCode 提供的调试器（这里使用的是 GNU GDB）是一个功能强大的工具，使用此调试器可以观察程序的运行时行为并确定逻辑错误的位置，可以中断程序的执行以检查代码，计算和编辑程序中的变量。为了顺利进行后续的各项实验，读者一定要学会灵活使用这些调试功能。

**使用断点中断执行**

读者之前在使用调试设置 Demo input1 调试演示程序的过程中，由于已经在 main.demo 文件中添加了隐藏断点，所以读者不需要自己手动添加断点就可以让程序在适当的代码处中断执行。但是，当读者需要调试 main.c 文件中的源代码时，就需要自己手动添加了断点了。方法如下：

1. 使用 VSCode 打开 main.c 文件。
2. 在 main 函数中调用 post2nfa 函数所在行的最左侧（行号的左侧）点击鼠标左键，会在该位置显示一个红色圆点，表明已经为此行代码添加了一个断点。
3. 在 VSCode 左侧的活动栏中点击 Run and Debug 按钮，可以打开 RUN AND DEBUG 窗口，在其顶部的列表框中选择 input1 这一项。这样，在调试的时候，就会将 input1.txt 文件中的内容重定向到标准输入，从而将该文件中的正则表达式转换为 NFA。如果读者需要调试其它 input 文件中的正则表达式，就需要将调试配置设置为对应的项。
4. 在“Run”菜单中选择“Start Debugging”（快捷键 F5）开始调试程序，会在刚刚添加断点的代码行左侧空白处显示出一个黄色箭头，表示程序已经在此行代码处中断执行（注意，黄色箭头指向的代码行是下一条将要执行的代码行，此行代码当前还没有执行）。

**单步调试**

接下来，按照下面的步骤练习使用单步调试功能：

1. 在“Run”菜单中选择“Step Into”，调试进入 post2nfa 函数，会在该函数中第一条可以执行的代码行中断执行。
2. 然后使用“Run”菜单中的“Step Over”单步调试 post2nfa 函数中的源代码。每执行一行代码，可视化窗口中的内容就会发生相应的变化，例如构造单字符的 NFA

片段，构造连接 NFA 片段等，读者可以根据程序运行的过程以及可视化窗口中的内容查找逻辑错误的位置。

3. 读者此时可以在左侧的“Run And Debug”视图的“CALL STACK”中查看函数的调用层次，双击 main 函数或子函数所在行，可以切换到指定函数的上下文。
4. 读者也可以根据需要在“Run”菜单中选择“Continue”继续运行，或者选择“Stop Debugging”结束调试。

通过上面的调试过程，读者应该了解到“Run”菜单中的“Step Into”功能用于调试进入一个函数，也就是说，当黄色箭头指向一个函数调用语句时，如果选择“Step Into”，就会继续调试函数内部的语句；否则，如果选择“Step Over”，就不会进入函数内部，而是会跳转到函数运行完毕后的下一条将要执行的代码。此外，还有“Step Out”，顾名思义，就是从函数中直接跳出到函数运行完毕后返回的代码行。

**查看变量的值**

在调试的过程中，VSCode 提供了三种查看变量值的方法，读者需要灵活掌握这些方法，在需要的时候根据变量的值判断程序的行为是否正常。按照下面的步骤练习这些方法：

1. 启动调试，在之前设置的断点处中断。
2. 将鼠标移动到源代码编辑器中某个变量的名称上，会弹出一个窗口显示出变量的值。
3. 选择“View”菜单中的“Run”打开左侧的“Run And Debug”视图，在“VARIABLES”窗口中，可以查看所有变量的值，包括全局变量和当前函数上下文中的局部变量。当读者在“CALL STACK”窗口中双击不同函数所在行，切换函数的上下文时，“VARIABLES”中的局部变量也会随之切换。
4. 在源代码编辑器中某个变量的名称上点击鼠标右键，在弹出的快捷菜单中选择“Add To Watch”，可以在左侧的“WATCH”窗口中持续监视此变量的值。
5. 结束此次调试。

**继续验证项目**

当读者认为自己编写的源代码可以正常处理所有的正则表达式后，可以继续使用自动化验证功能（在前面介绍了自动化验证的使用方法），来自动检测所编写的程序是否能够通过所有 Case 的验证。

**提交作业**

实验结束后需要将本地磁盘上修改后的源代码文件提交到平台的个人项目中，方便教师通过平台查看读者提交的作业。可以按照下面的步骤提交作业：

1. 使用 VSCode 查看文件变更详情，具体方法是在 VSCode 的“View”菜单中选择“SCM”，打开左侧的“Git 源代码版本控制”窗口，在变更列表中会显示最近新建、删除或内容被修改的文件。使用鼠标左键双击变更列表中的文件，会使用编辑器显示当前的文件内容（右侧）与上一个版本的文件内容（左侧）相比较有哪些修改。如果想放弃对文件的修改，可以点击文件所在行右侧的“Discard Changes”按钮。
2. 在确认文件变更详情没有错误的情况下，读者就可以开始提交作业了。最快捷的方法是，在 VSCode 的“Terminal”菜单中选择“Run Build Task...”，会在 VSCode 的顶部中间位置弹出一个可以执行的 Task 列表，选择其中的“提交作业”，会自动完成 Git 库的提交（commit）和推送（push）操作，读者只需要在 VSCode 顶部中间位置弹出的命令窗口中，按照提示依次输入平台的用户名和密码即可。
3. 提交作业成功后，会自动使用浏览器打开平台中个人项目的页面，读者可以在这个页面中浏览项目中的文件和变更记录。
4. 如果在提交作业后发现仍然有文件需要修改，可以在完成修改后重复上面的步骤，再次提交作业。

除了上面步骤中介绍的使用 Task 完成提交作业的方法外，读者也可以使用 VSCode 提供的“Git 源代码版本控制”窗口中的提交(Commit)和推送(Push)功能完成作业的提交，具体方法请读者自行学习，这对于读者今后熟练使用 VSCode 中的 Git 功能也是很有帮助的。

**线上查看流水线的运行结果**

读者提交作业后，默认会自动使用浏览器打开项目的流水线列表页面，读者可以在列表顶部查看最新一条流水线的运行结果。流水线运行的过程与本地自动化验证完全一致，也是在通过所有 Case 的验证后，流水线就成功结束，否则流水线就会失败。如果流水线运行失败，读者可以在流水线输出的信息中查找失败的原因，然后使用 VSCode 继续修改源代码，在通过本地自动化验证后再次提交作业即可。读者可以在平台提供的《学生手册》中查看关于流水线的更多帮助内容。

**线上查看代码缺陷**

线上流水线运行成功后，会对读者的源代码进行静态分析并生成一个代码质量报告。读者可以在任务信息板的“代码缺陷”一栏查看代码质量报告的详情，特别要注意一些严重缺陷（例如内存泄露等）。点击报告中的每一条链接，可以在源代码中定位存在代码质量问题的位置，点击每一条链接右边的图标，可以查看该代码质量问题的解决方法。读者可以在平台提供的《学生手册》中查看关于代码缺陷的更多帮助内容。

## 四、思考与练习

1. 编写一个 FreeNFA 函数，当在 main 函数的最后调用此函数时，可以将整个 NFA 的内存释放掉，从而避免内存泄露。
2. 读者编写完代码之后可以对 input2.txt 到 input5.txt 中的算例进行一一验证，确保程序可以将所有形式的正则表达式转换为正确的 NFA，并验证通过。
3. 对 input6.txt、input7.txt、input8.txt 文件中的正则表达式进行验证，并画出例 7 和例 8 的 NFA 状态图。
4. 详细阅读 re2post 函数中的源代码，并尝试在源代码中添加注释。然后尝试为本实验中所有的例子绘制解析树（类似二叉树）。