

单片机自学指南

序文

为了同学们更好学习城市轨道交通传感器技术，做到除了掌握课程以外，还要兼具创新意识，使所学知识可以作为一项真正的技能在生活与工作中应用，故作此书。

所有目前所学知识都是从现有的资料整理而得出，所以此书是在前人的基础上而作成，故遵循本网站倡导的“海盗原则”，即此书版权属于所有人，任何人可以下载、引用、传阅、教学、编辑等.....

严禁此书用于商业用途或以个人名义出版，若经发现后果自负！

2025.1.23

简介&概论

由于此书面向专科学生（我也是☺），故章节的排序与一本传统意义上的“教材”有所区别，主张先掌握技能，再了解理论。所以要是你也追求实用的原则，可以按照默认顺序阅读。

如果你很厉害，认为此顺序不合理，想先学习理论，那可以试试以下顺序：**第六章、第五章、第一章~第四章**

此外，本书欢迎所有人进行编辑，若有什么疑问和改进建议，欢迎联系网站站长！！

接下来是我自己的见解，要记住，本书的理论和思想部分，**没看明白的话，那只要先照做就行，学着学着就会悟了！！**

我认为开发单片机无非就三个步骤：**编程 编译 烧录**

- **编码**：就是写代码，定义逻辑、功能和各种实现。
- **编译**（此书基本上是生成hex文件）：把写好的代码通过编译软件（比如Keil）转换成单片机可以理解的机器语言，生成.hex文件，这个文件包含了可以烧录到单片机里的指令和数据。
- **烧录**：将生成的.hex文件通过烧录软件（比如STC-ISP等）将程序写入单片机中，完成硬件上的软件安装。

只要了解这三个步骤的思想，基本上可以在任何我们常用的智能设备上实现单片机开发，通常是电脑，现在手机进行开发的方法我不知道，但肯定也是有的，本书讲述用电脑进行操作。在后续的理论学习章节中，我会详细阐述这三个步骤。

第一章 电脑的使用及技巧

1. 把你的疑问用简洁明了的文字整理好。
2. 打开任意一个浏览器（此处使用Firefox为例）。



3. 打开浏览器，在能输入文本的搜索栏内输入你的疑问。



4. 如果这样没解决的话，欢迎各位进行留言和评论，或者直接联系我们！

第二章 工具的安装和使用

还记得你刚看的简介吗？接下来我们便要为我们的第一步——**编码**作准备，就是装一个写代码的软件（它同时拥有**编译**功能）。在此之后，我们要安装Proteus仿真软件，来实现在电脑上仿真（或者说模拟、虚拟）开发单片机来实现**烧录**功能。

本章节将带大家安装必备工具，以及在最后对安装的一个个步骤进行解释。

第一节 Keil5的安装

首先打开浏览器，将该网址复制到地址栏，按回车

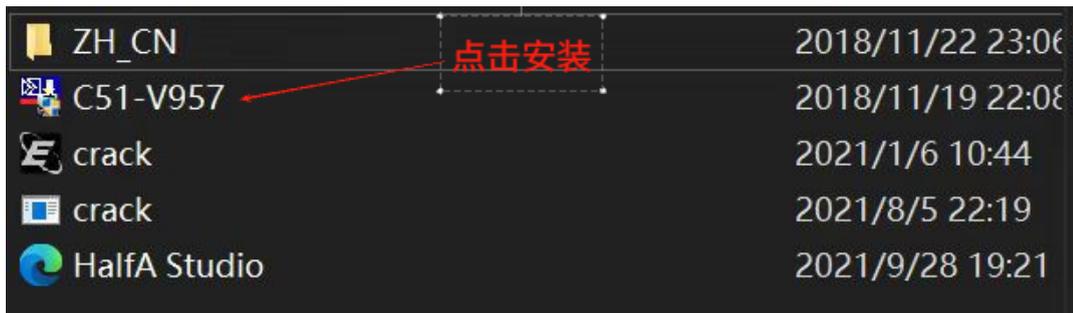
<http://snow2.online/?p=163>



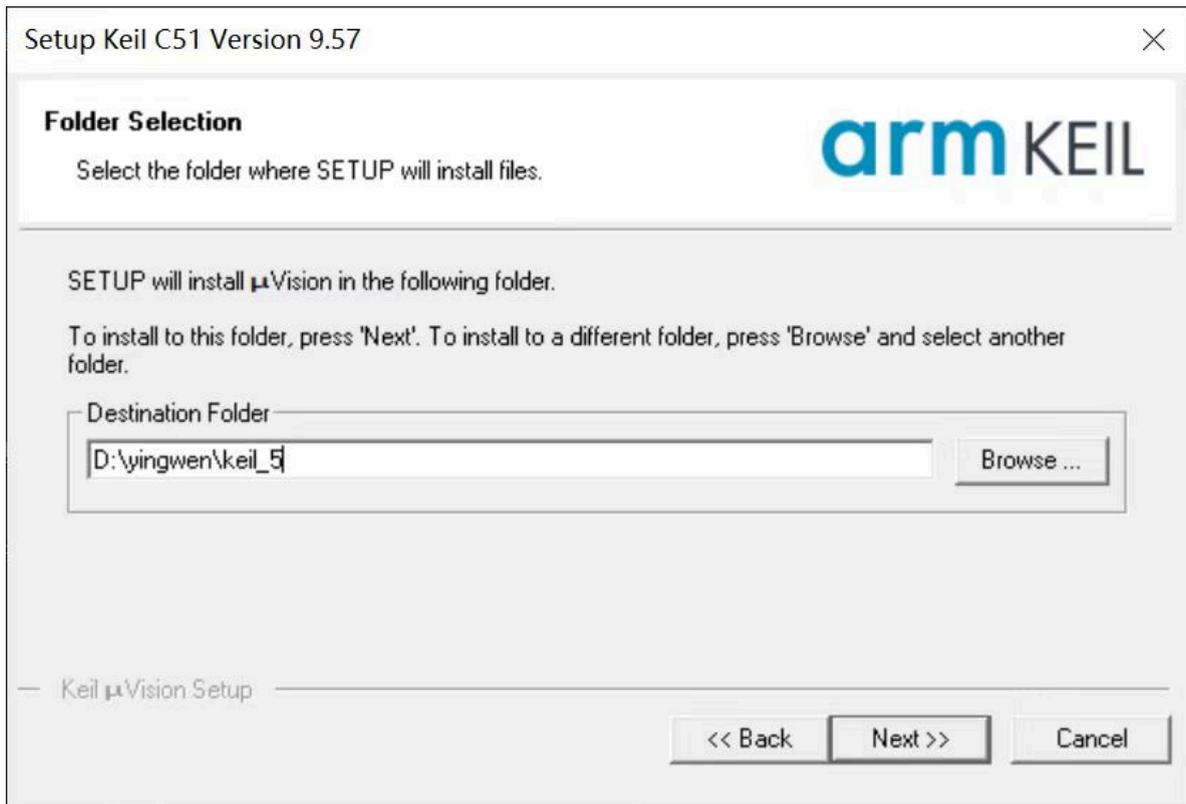
之后点击此处下载



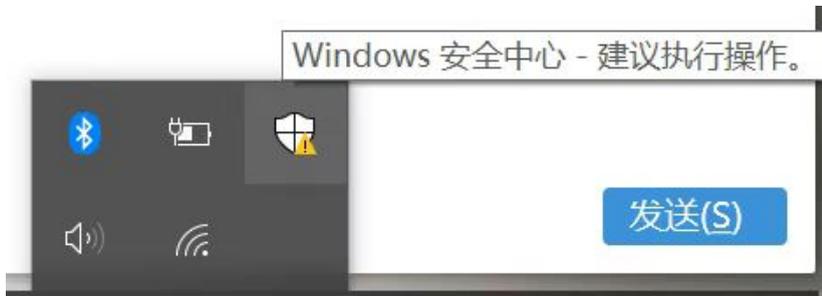
右键解压文件后如下，点击开始安装步骤



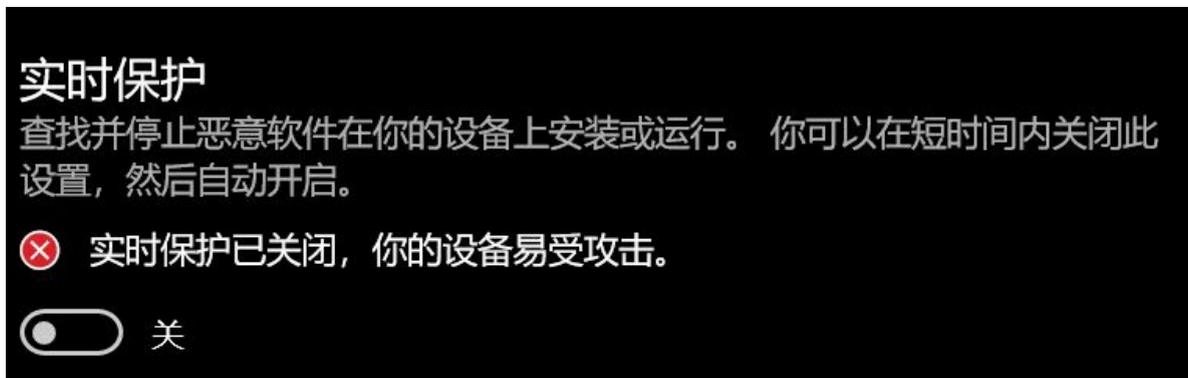
然后开始设置该软件的位置，**文件目录的每一级必须为英文**，以免之后使用时出现错误



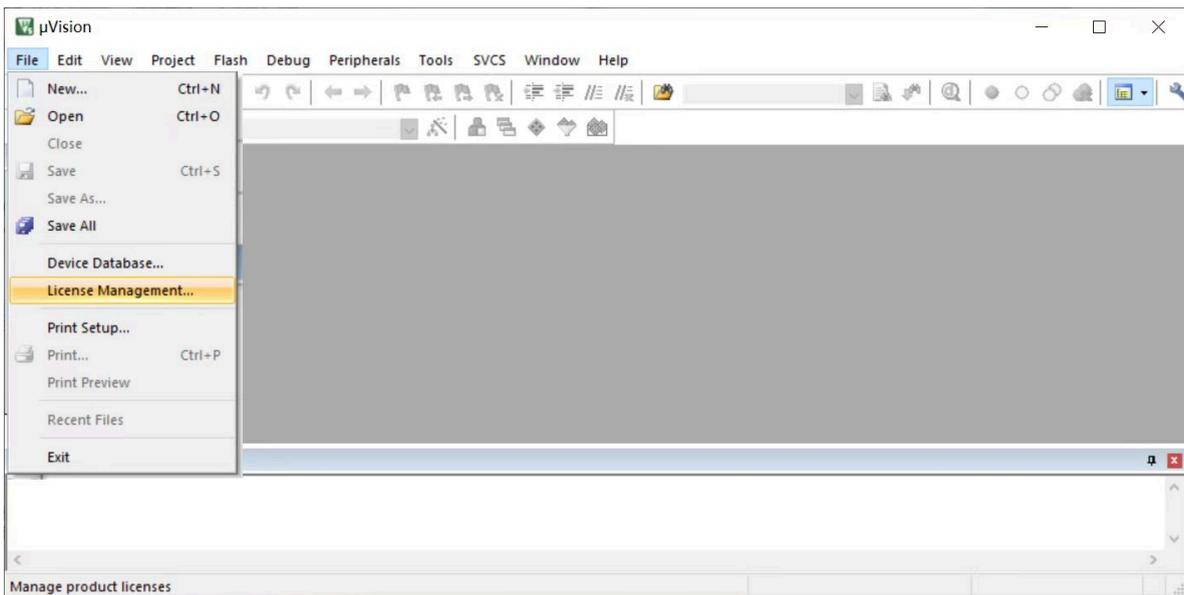
安装好后，在电脑右下角状态栏找到盾牌标志，基本上都如下：



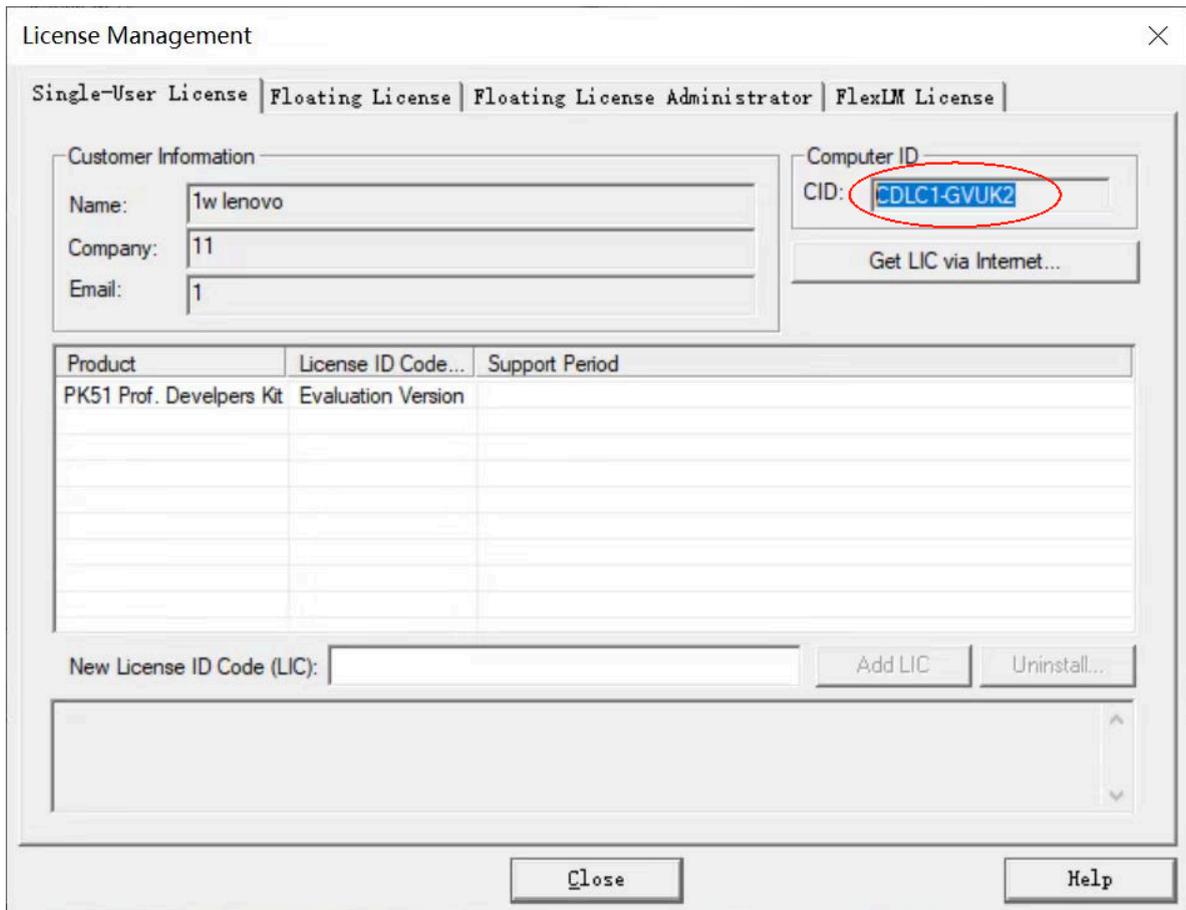
点击后找到“**病毒和威胁防护设置**”选项，点击管理设置，关闭实时保护



我们之后在桌面上找到该软件，右键以管理员身份运行，在上方工具栏点击File后，打开License Management...选项



复制右上角CID代码



之后我们打开该软件，对Keil5进行破解（若双击无法打开请尝试右键点击以管理员身份运行），进行以下步骤



Keil Generic Keygen - EDGE



把复制CID代码在此粘贴

Keil Embedded Workbench

1

Keygen

License Details

CID: CDLC1-GVUK2 Target: C51

Prof. Developers Kit (Plus)

3

复制此处的代码

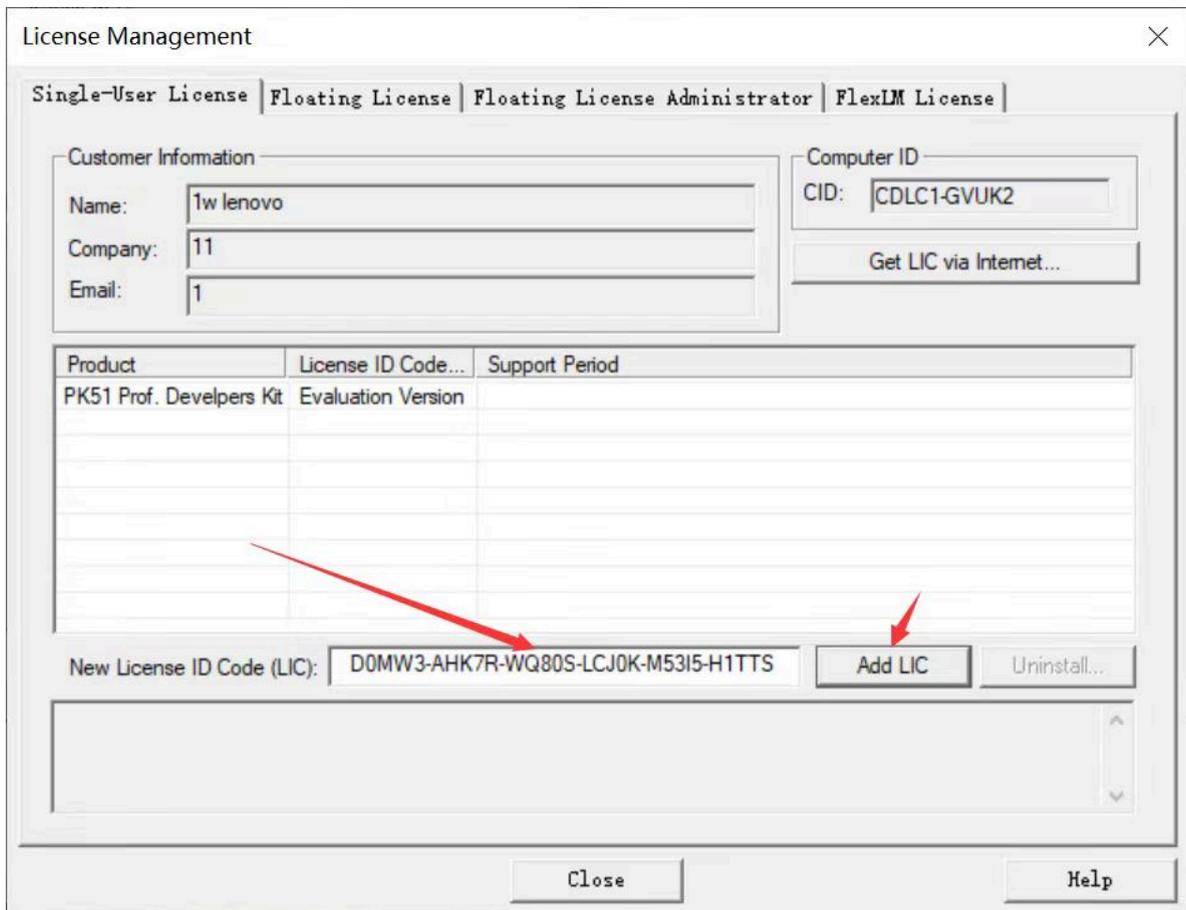
DOMW3-AHK7R-WQ80S-LCJ0K-M53I5-H1TTS

2

Generate 点击此按钮

Exit

回到Keil5软件，将复制的代码粘贴至框中，之后点击Add LIC



到此 恭喜你完成了Keil5的安装!!! 🌈

第二节 Proteus的安装

让我们再次打开浏览器，打开该网址

<http://snow2.online/?p=145>

这次要点两个，下载好后，把两个都给解压了

Proteus 安装

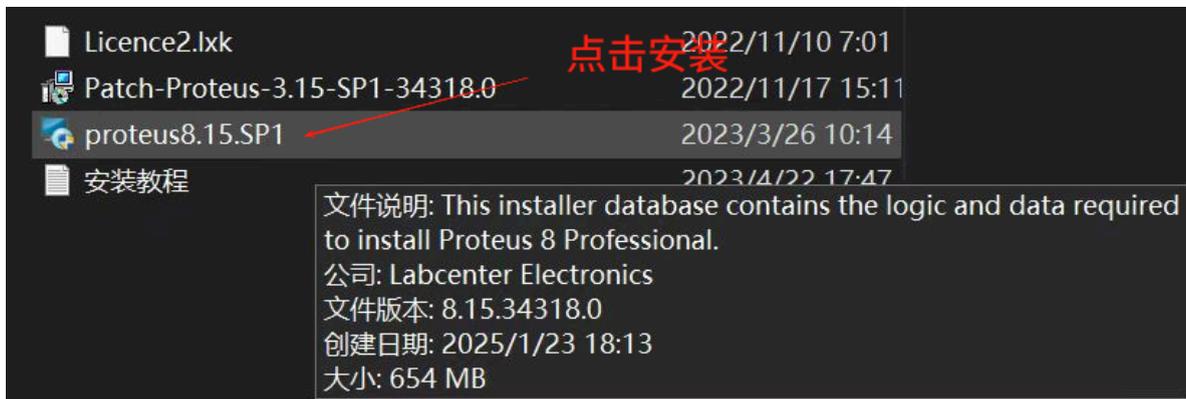
60次阅读 2条评论 编辑

proteus8.15汉化

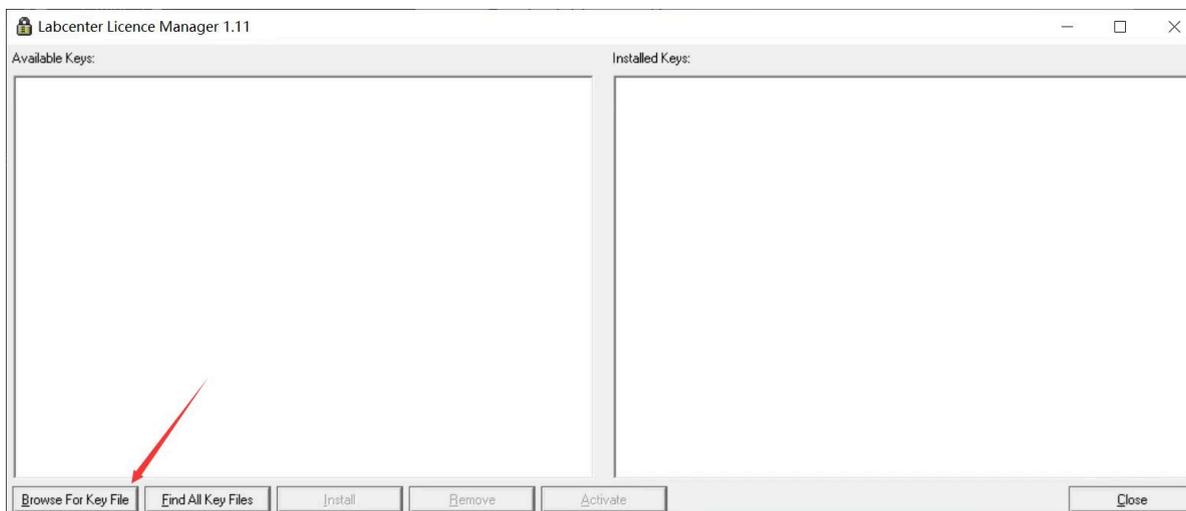
proteus8.15.SP1

proteus8.15.SP1_	2025/1/23 18:12	压缩(zip)文件夹	657,636 KB
proteus8.15汉化	2025/1/23 18:01	压缩(zip)文件夹	1,287 KB

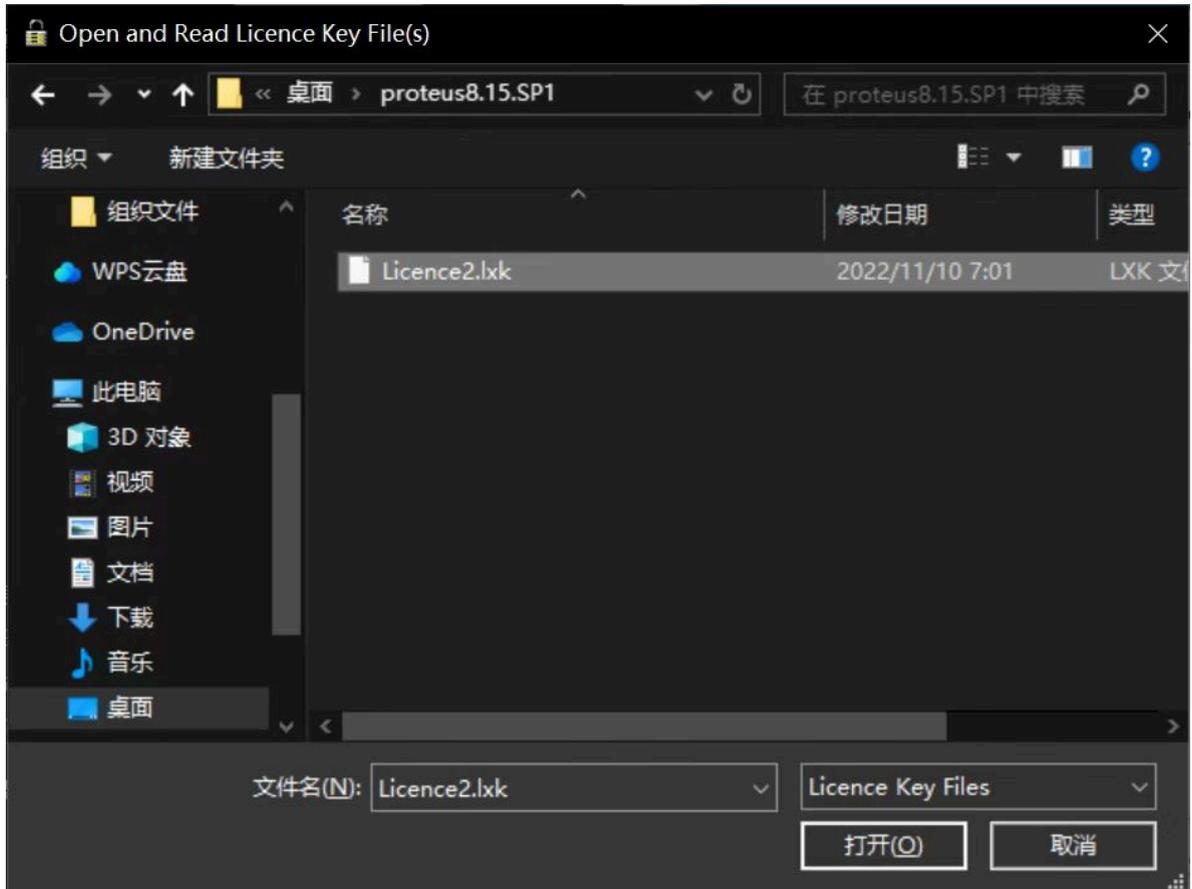
首先点它进行安装（此后没有说明的步骤基本上是一直点击下一步‘Next’）



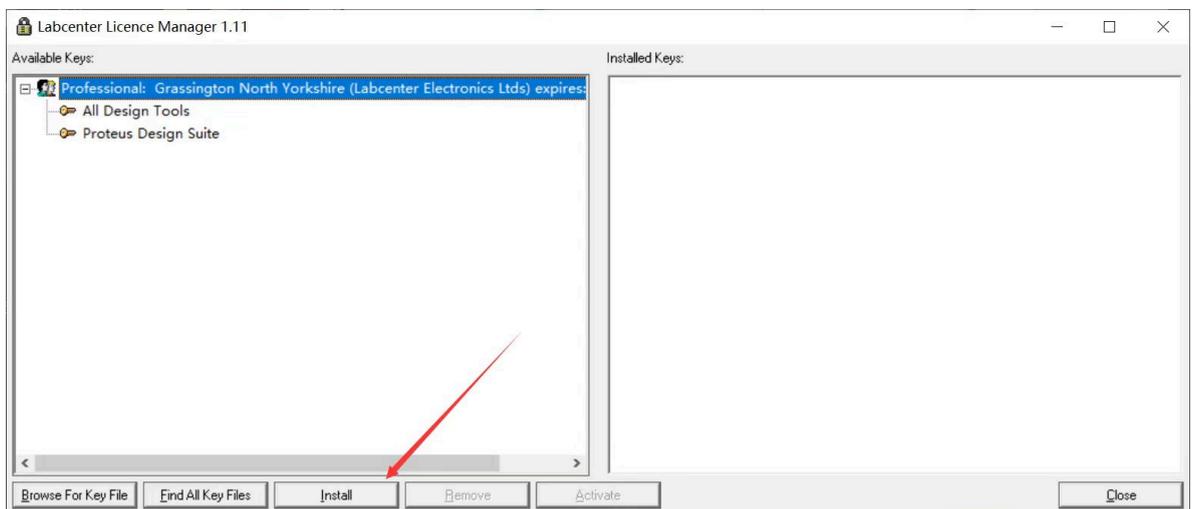
一直点下一步（next），直到这一步，点击最左边的键

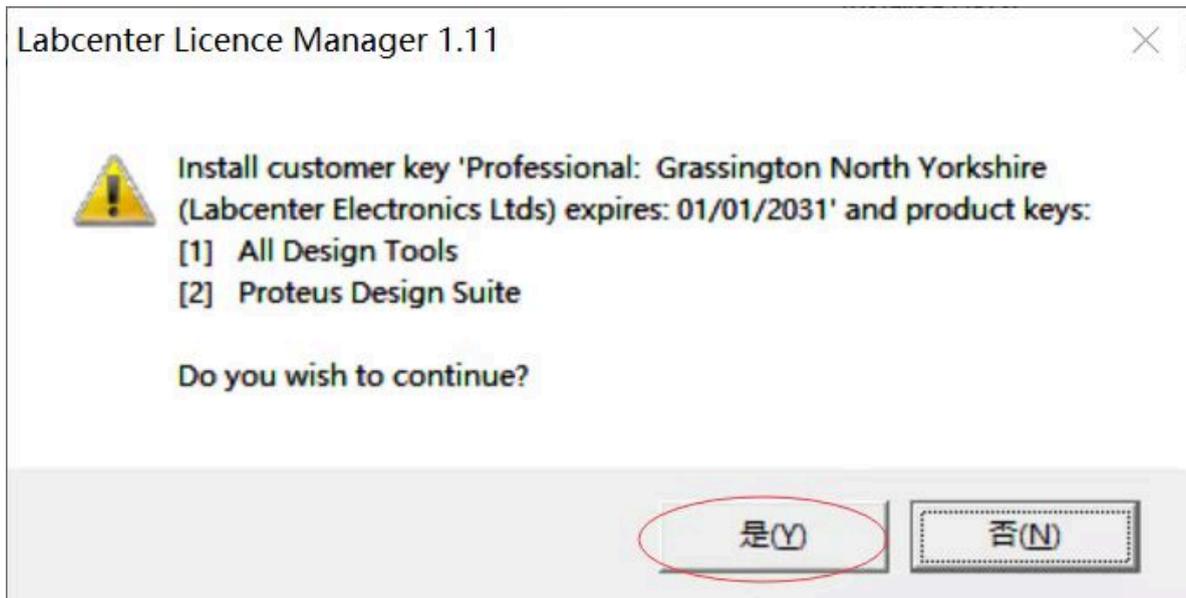


找到你解压的位置，选中这个文件，点击打开

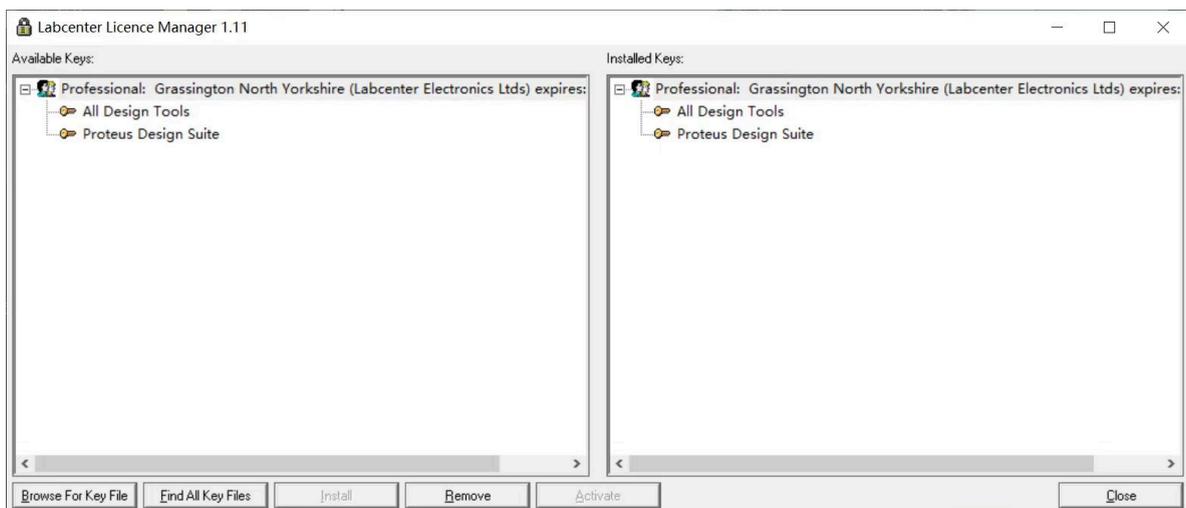


之后就会变成这样，然后我们点击最左边的键，若跳出弹框就
点是





变成这样就成功了，我们点击最右边的Close即可



之后进行安装配置，选择第二项

Proteus 8 Professional Setup

Choose the installation you want

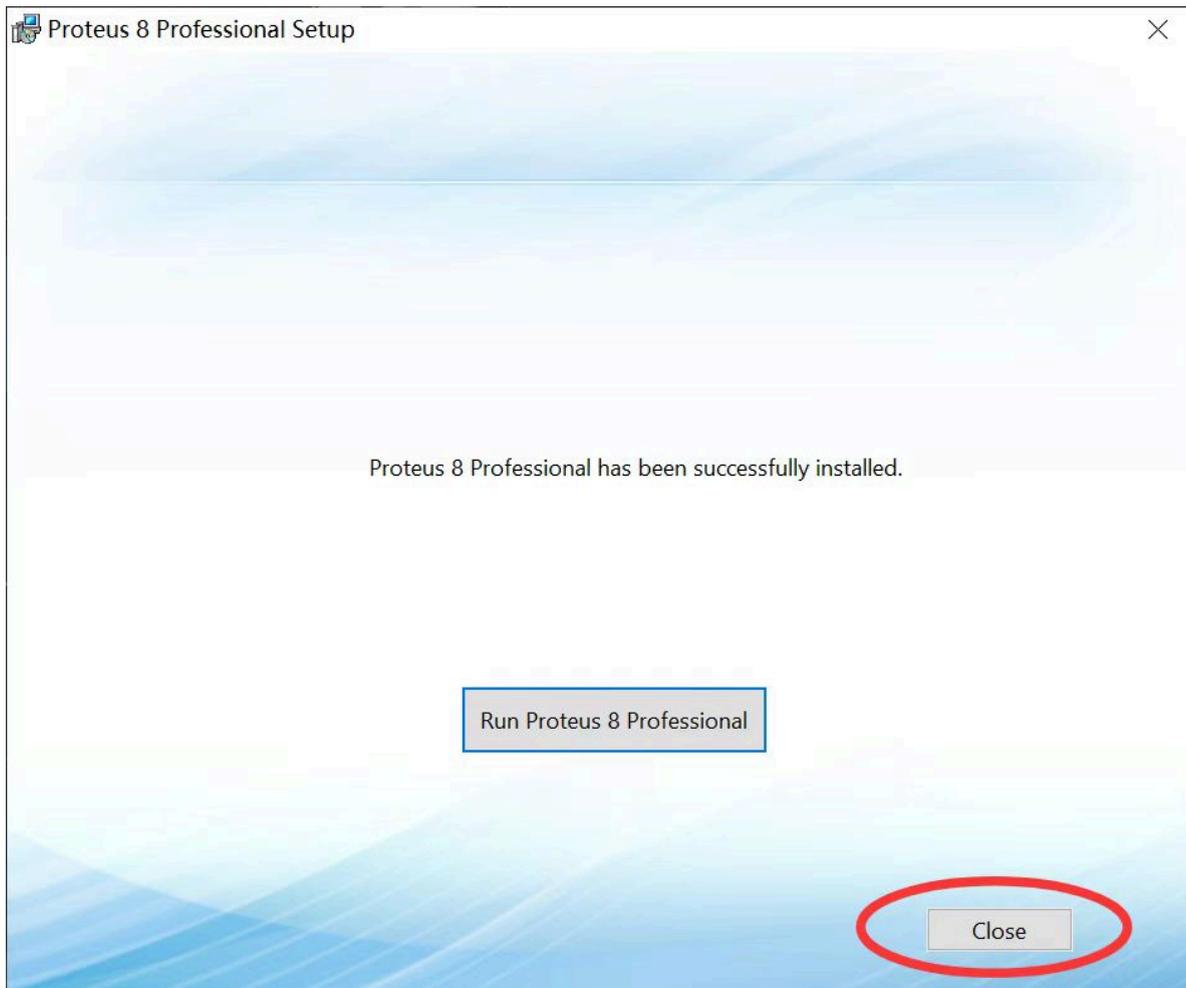
 **Typical**
Installs the most common program features. Recommended for most users.

 **Custom**
Allows users to choose which program features will be installed and where they will be installed. Recommended for advanced users.

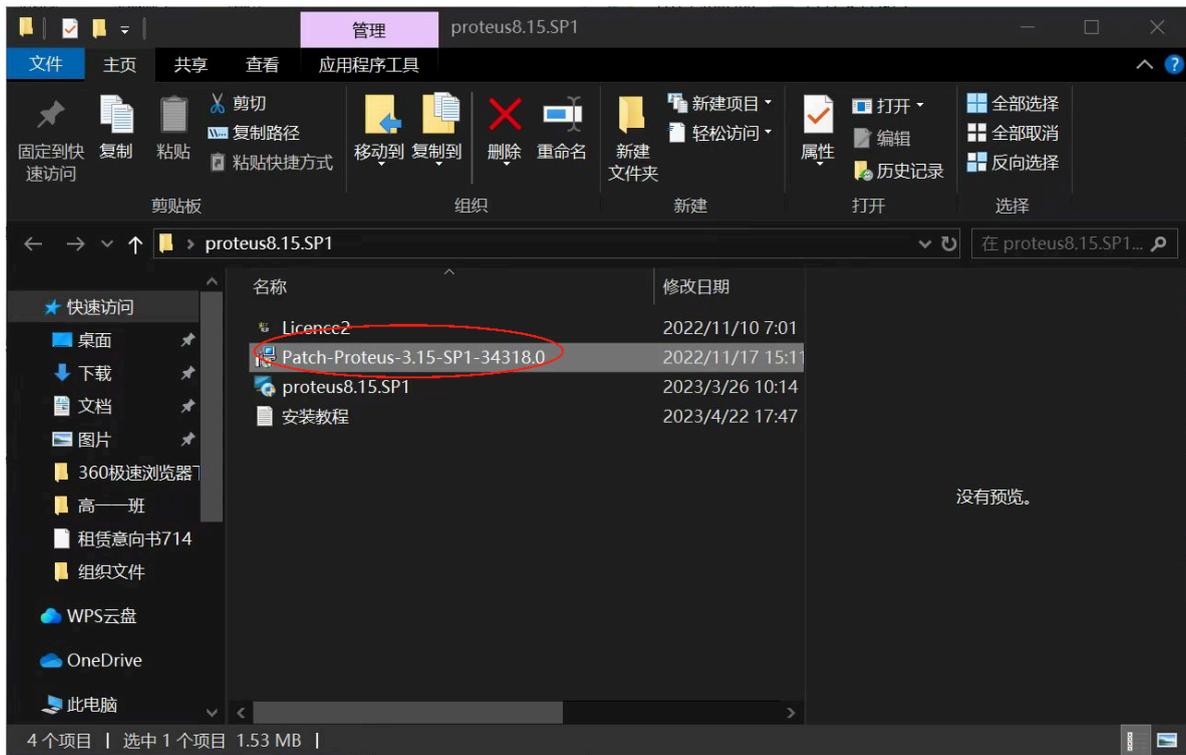
< Back



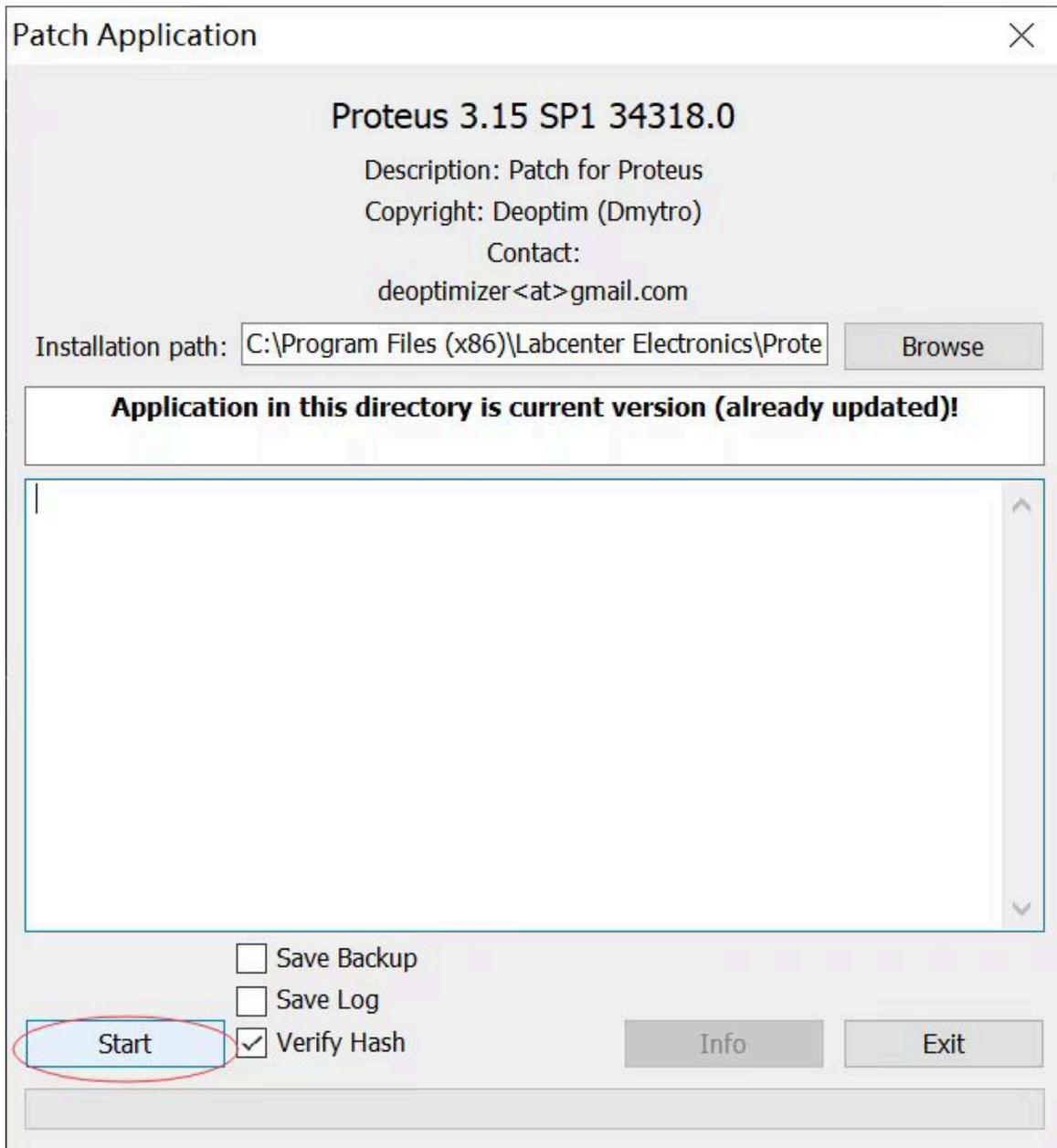
安装完成就如下，我们先点击关闭（Close）



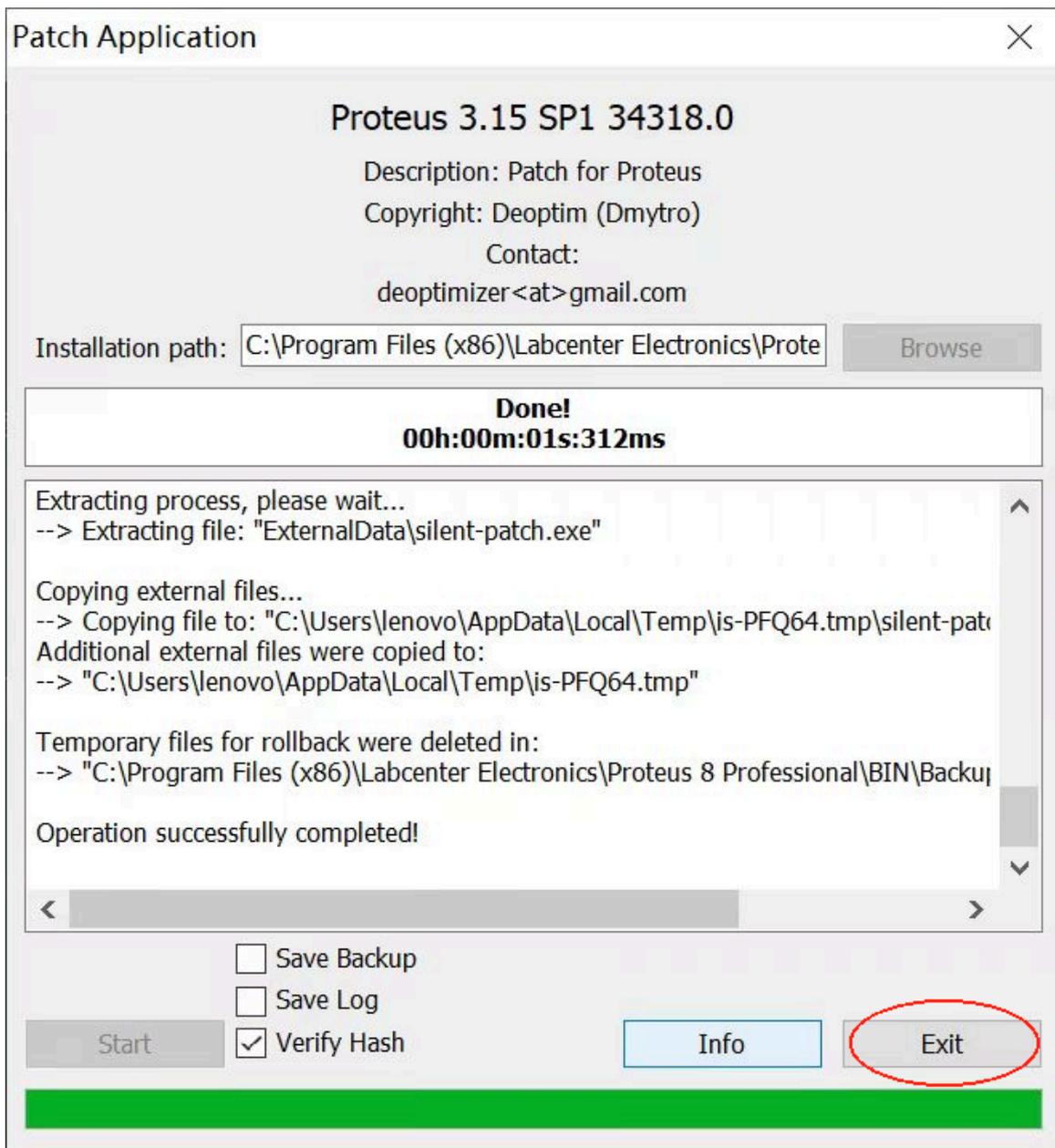
我们回到最开始解压的文件夹里，找到这个软件



打开后点击Start

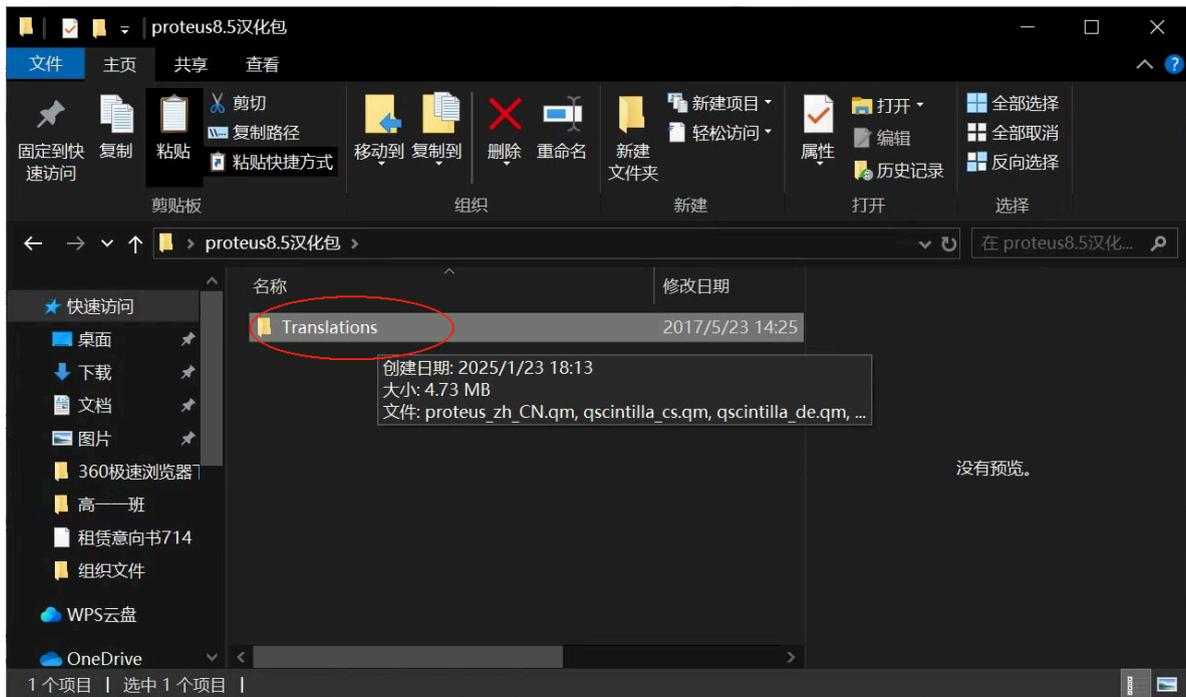


变成这样就完成了 点击退出 (Exit)



接下来是最后一步，**安装汉化**。

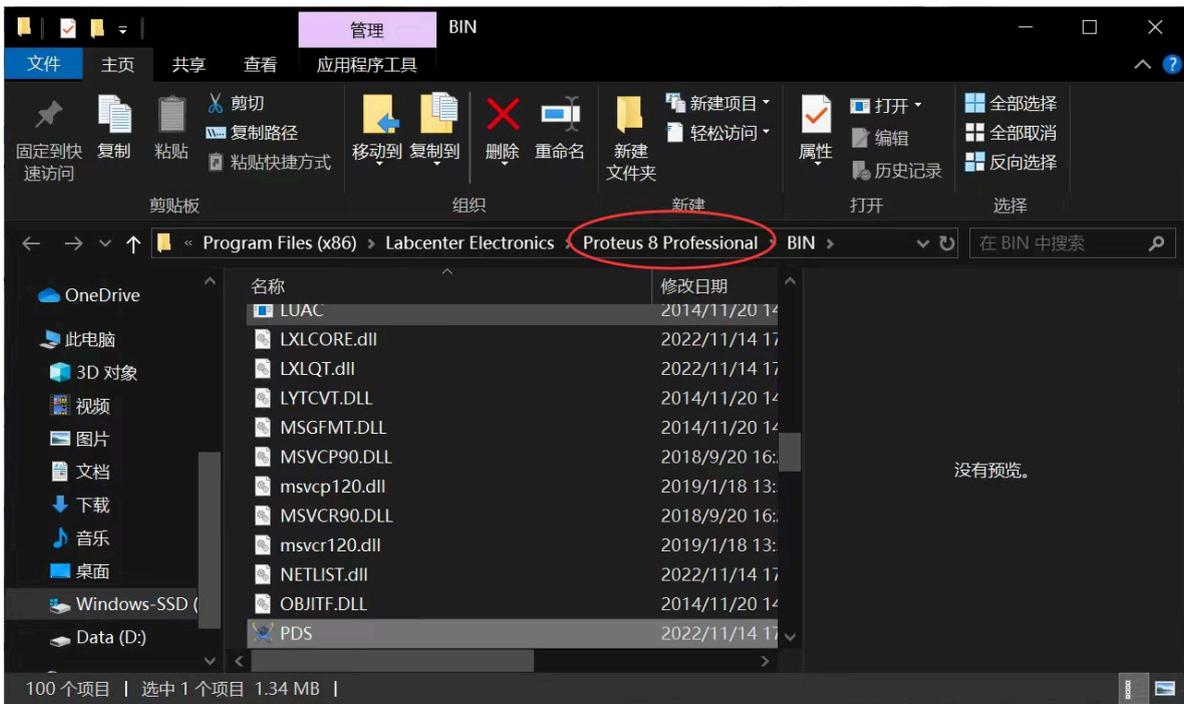
首先找到解压的另一个文件夹，复制Translations文件夹



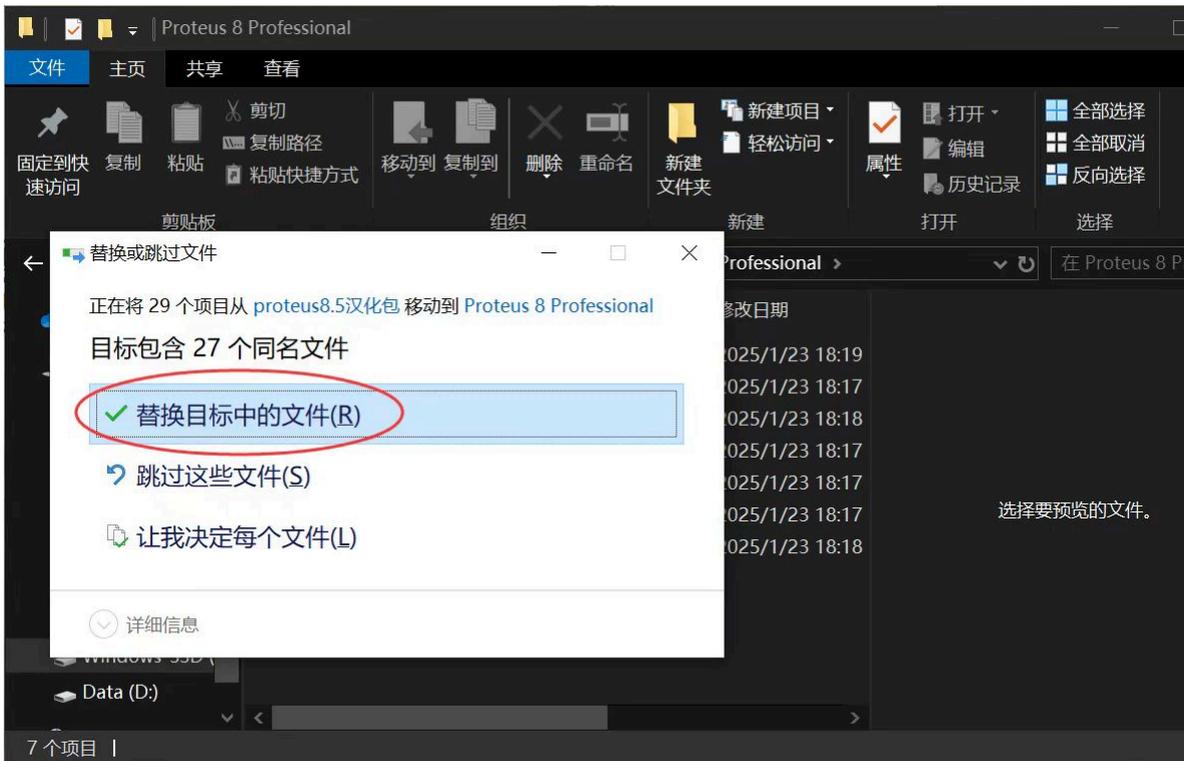
之后在桌面上找到该软件的图标，右键点击选择打开文件所在位置（或者你记得装在哪里也可以直接打开）



找到该位置（可以直接点击画圈处），右键空白处选择粘贴（或者使用Ctrl+c快捷键）



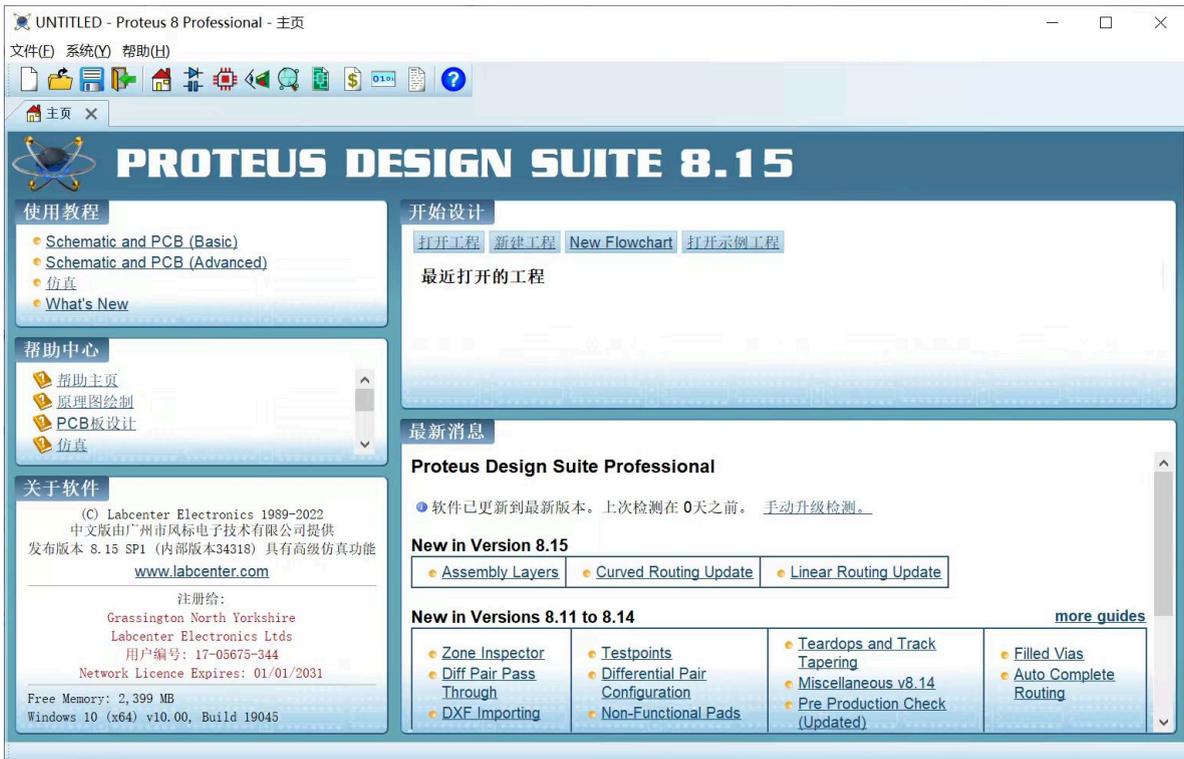
选择此项



跳出对话框就按顺序这样点



之后再点击打开软件 页面是这样的，那就没问题了，我们就可以开始学习了



恭喜装完必备工具🎉

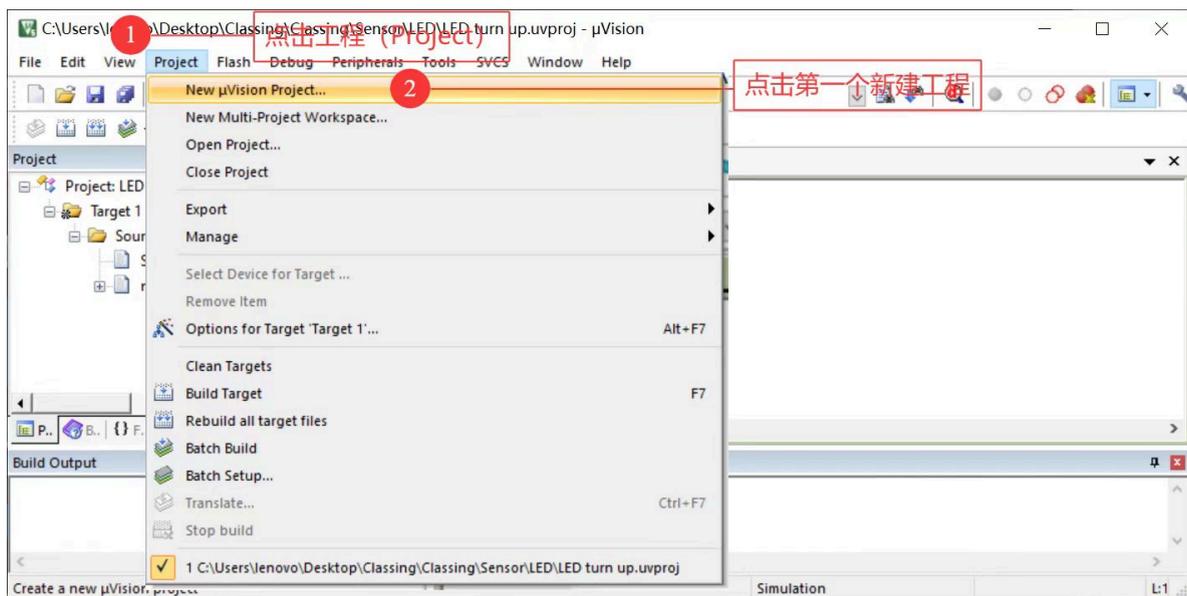
第三节 使用工具进行仿真开发

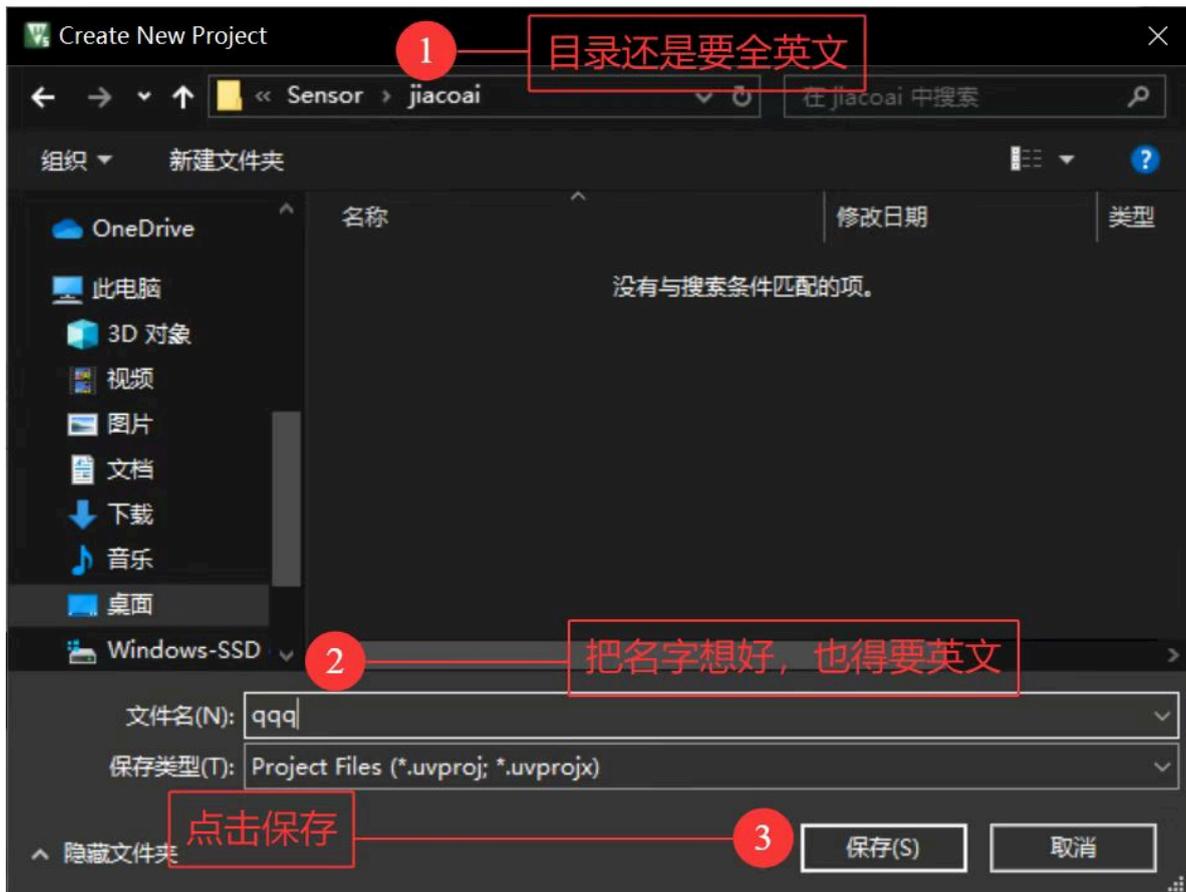
现在我们已经装好了工具，就可以在电脑上模拟单片机开发的整个流程了。该小节将给出简单的例子——点亮一个LED灯，先告诉你工具的使用步骤，原理目前会很少提到。

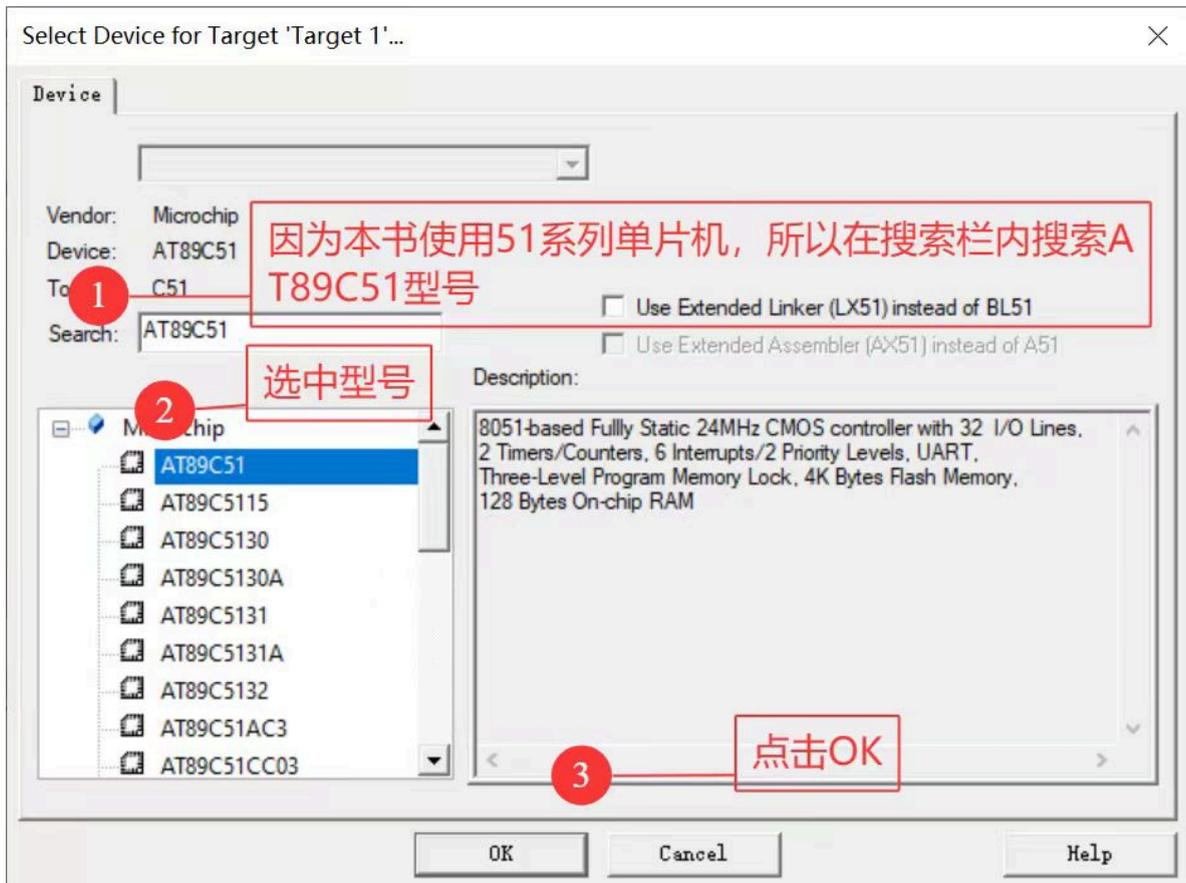
（正所谓“**行动是最好的老师**”）

一、使用Keil5创建工程

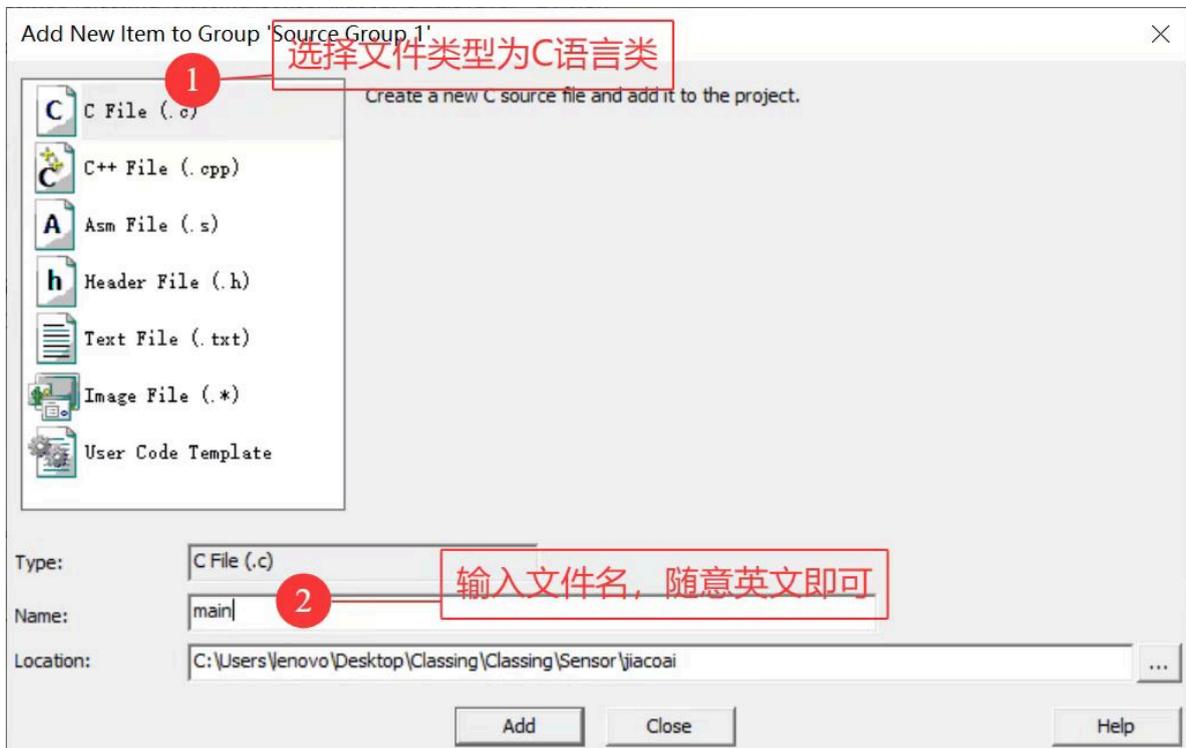
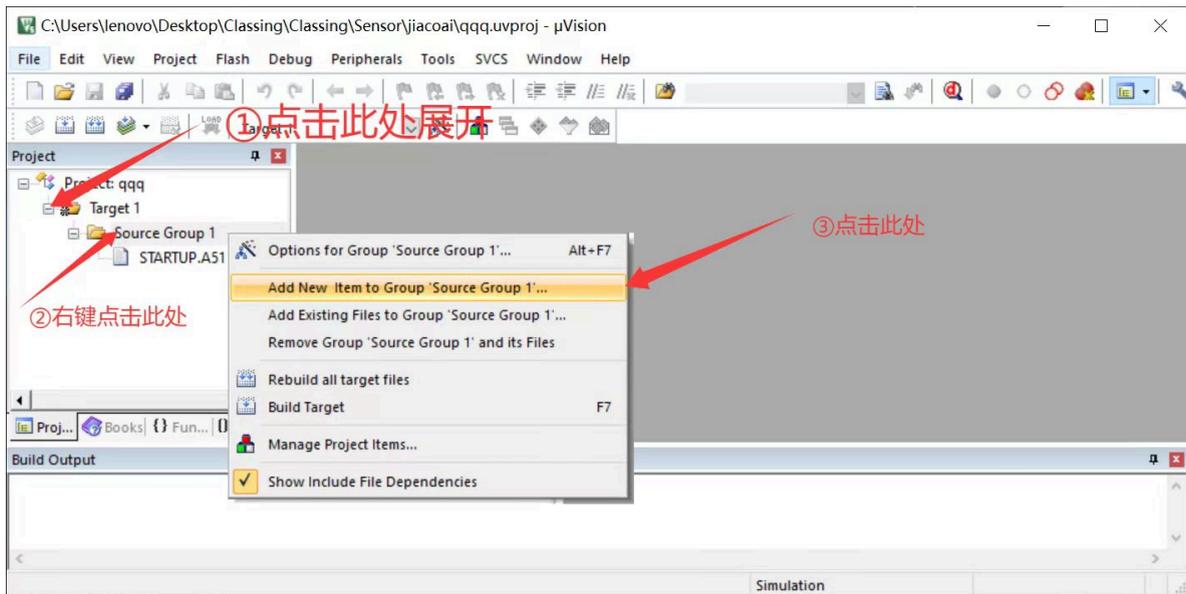
1.首先肯定要先打开Keil5,之后按如下步骤新建工程



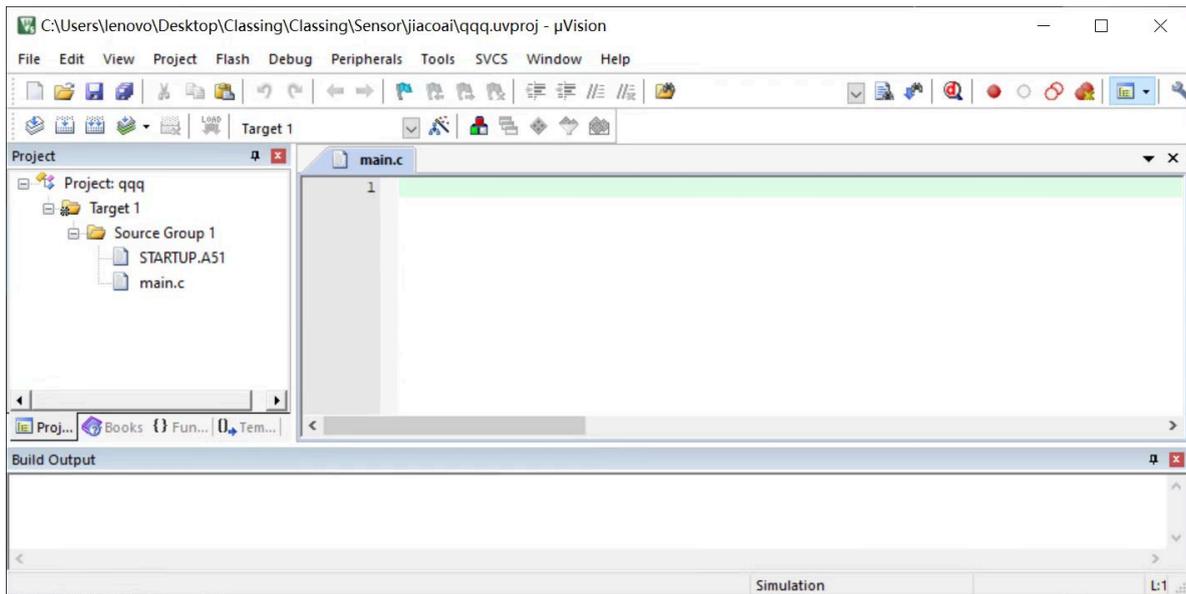




之后，新工程就建立好了，随后我们就要开始写代码，我们要添加一个C语言的文件

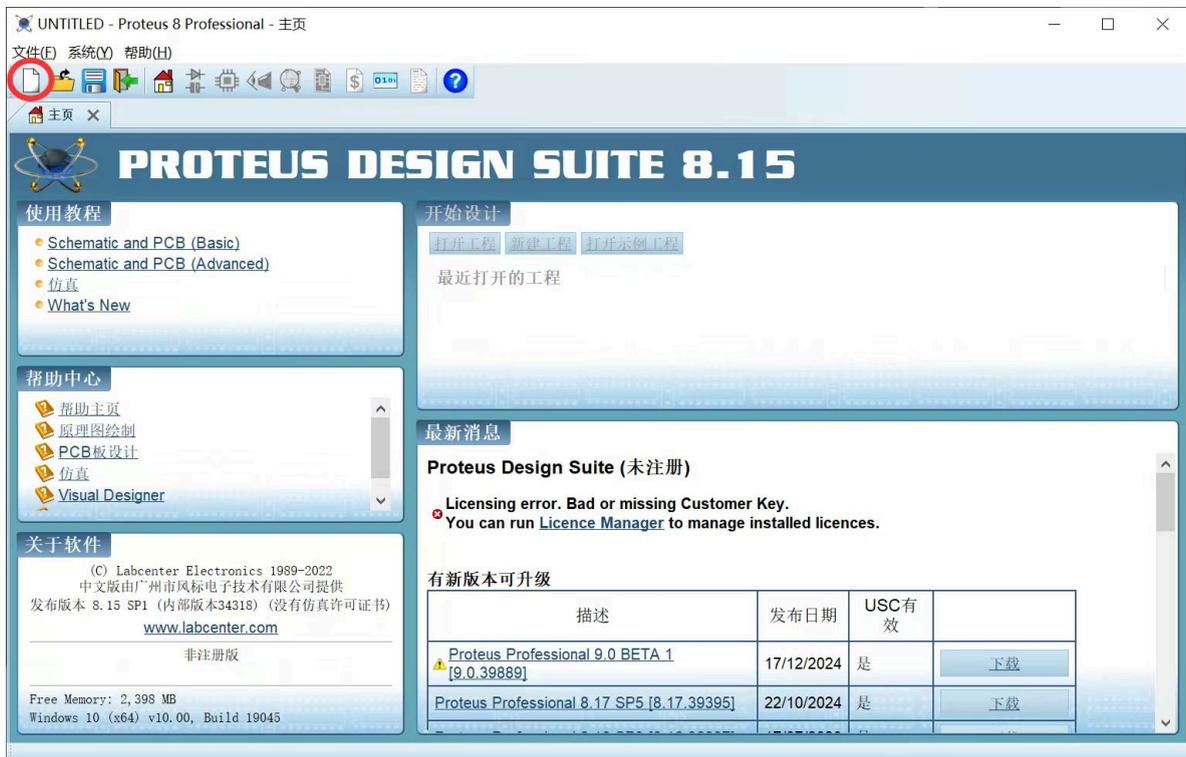


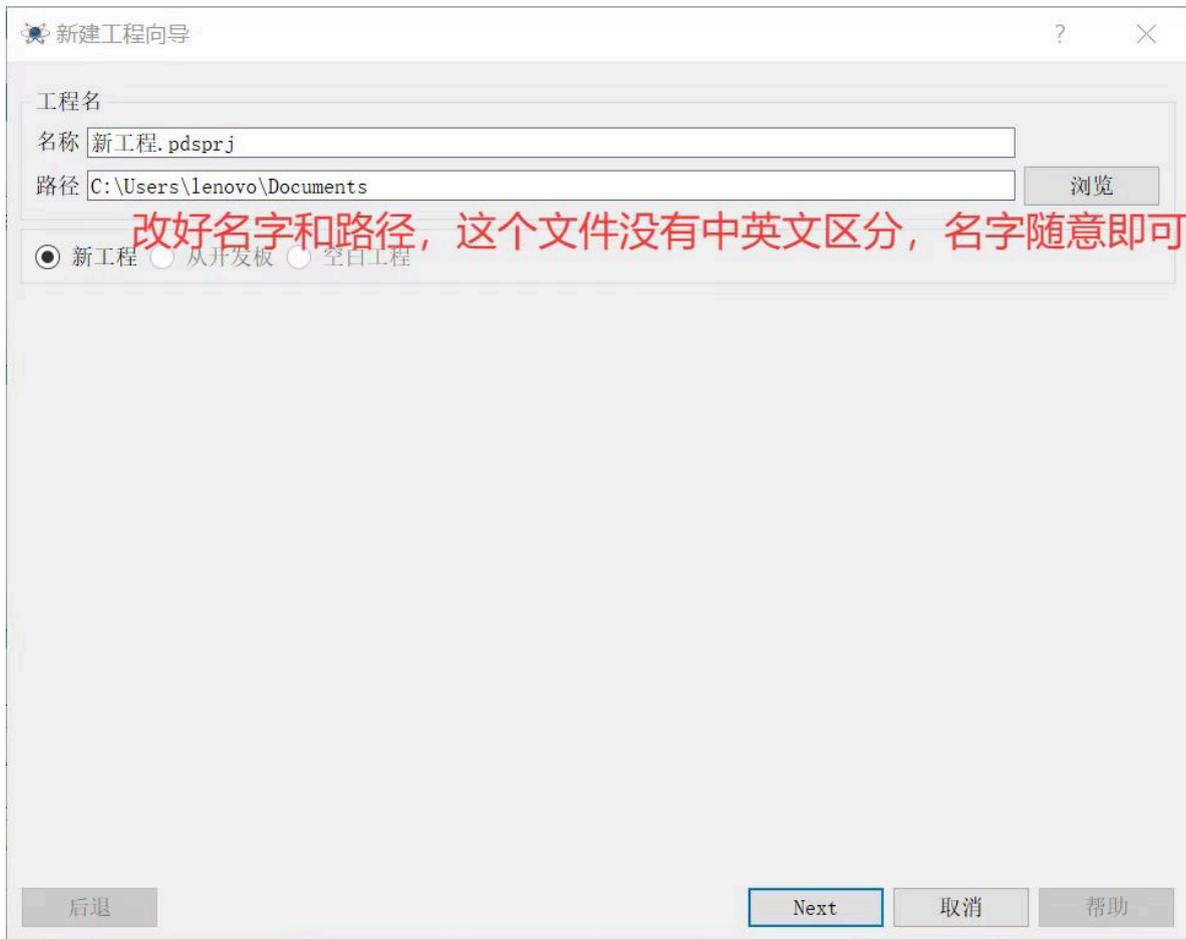
完成后是这个样子，之后我们就可以写代码了



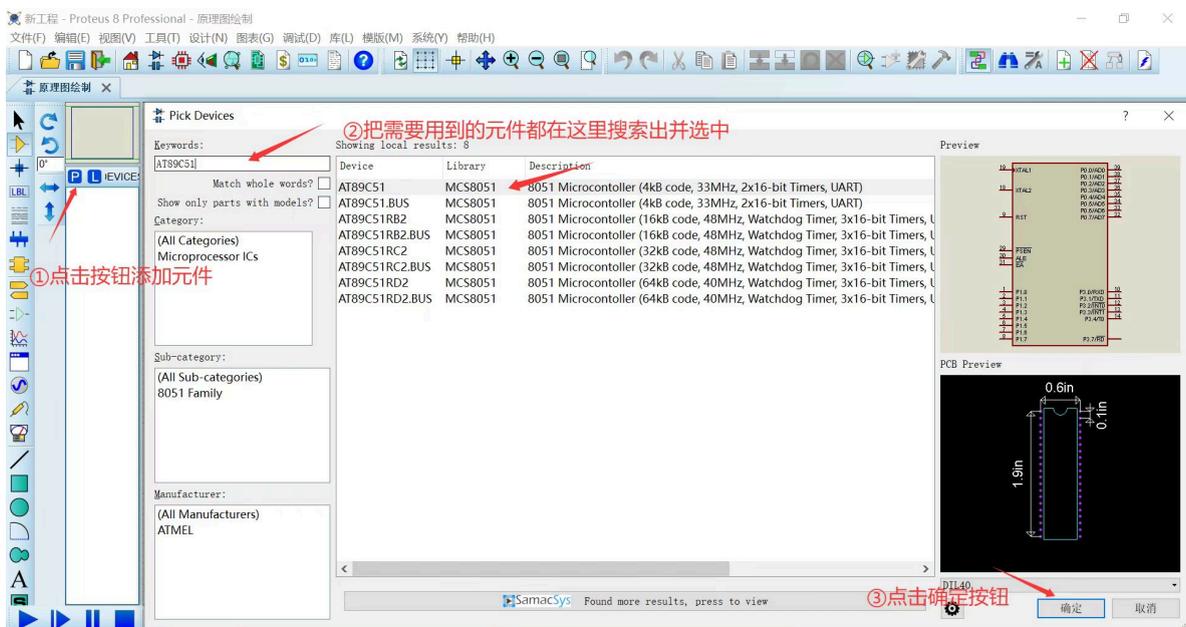
之后我会给出一段简单的代码，并作简要说明，以及让它在仿真软件中展示代码在单片机里运行的效果。不过我们先不写代码，我们打开Proteus软件，来练练手，熟悉一下操作。

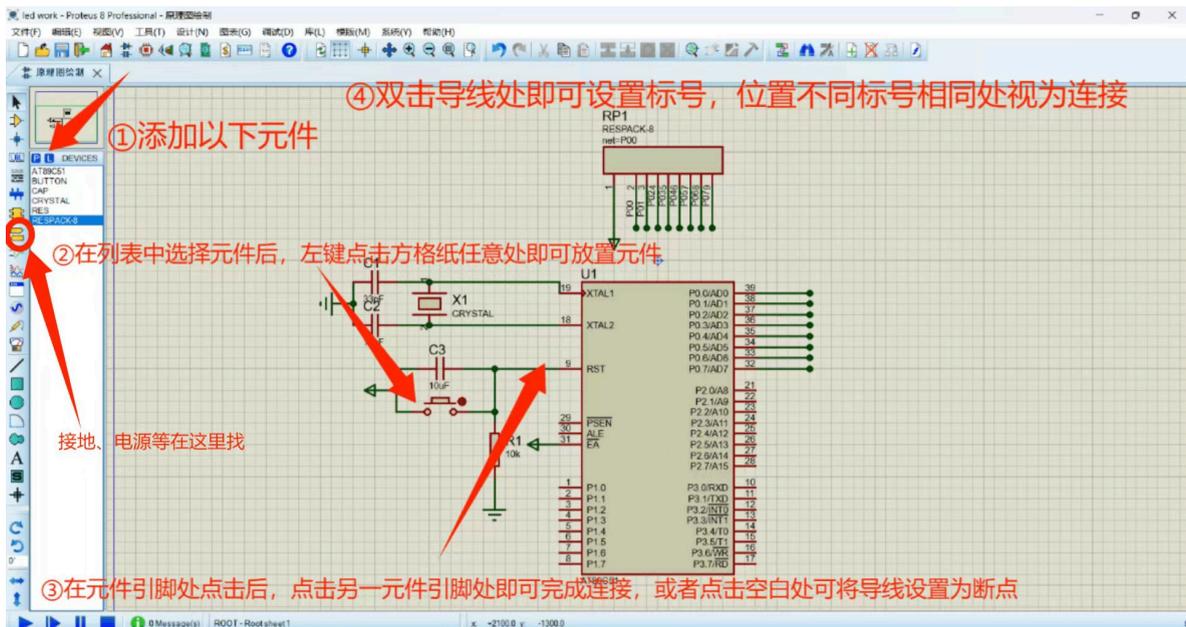
首先还是熟悉的新建





进入后我们先添加元件，以下为添加元件的步骤（这一步开始先不用开干，让你开干的时候会事先说明的）





上图即为51单片机的最小系统图，即基本上所有的51单片机仿真都需要拼出这个框架（其实这只是为了严谨，实际上仿真时这些花里胡哨不加上也没事，只需在EA处加上电源即可），元件放置好后，双击数值（如10uF）即可更改元件自身数值大小。

接下来对此最小系统作出简略的原理解释，粗略了解即可（我自己没整明白的地方用红色标记表示出来了，可以点击进入别人写的文章查看）

1. 时钟与复位模块

- **功能**：提供单片机的时钟信号和初始复位信号。
- **组成**：
 - **晶振 (X1)**：提供时钟信号，常见的是12MHz或11.0592Mhz。晶振会发出规律的电信号，帮助单片机保持在一个工作节奏上（设置晶振频率就像规定心跳跳动快慢一样）

- **电容 (C1, C3)** : 电容C1用于晶振的稳定工作 , C3用于滤波 , 避免干扰影响单片机的工作。
- **复位电路 (R1,C3) 、 复位引脚 (RST)** : 复位分为**开机复位**和**按键复位** , 我按照日常使用电脑的简单思维理一下 :
原理当然是电容、电阻和开关之间的组成的短路、通路关系 , 我之后再补充

开机复位 : 即单片机一通电就进行“重启” , 因为单片机在工作之前 , 可能有上一次工作结束后留下的混乱状态 (烂摊子) , 所以开机就要重启一下先恢复到最干净、初始的状态。

按键复位 : 就像电脑一样 , 单片机在运行时也会有程序跑飞、死机的情况 , 按下按键即可重启。

2. 单片机核心模块

- **功能** : 这是系统的核心部分 , 负责执行程序和控制其他模块。
- **组成** :
 - **AT89C51单片机 (U1)** : 这是整个系统的大脑 , 负责执行存储在程序内存中的指令。它具有多个I/O端口 (P0、P1、P2、P3) , 用于与外部设备进行交互。

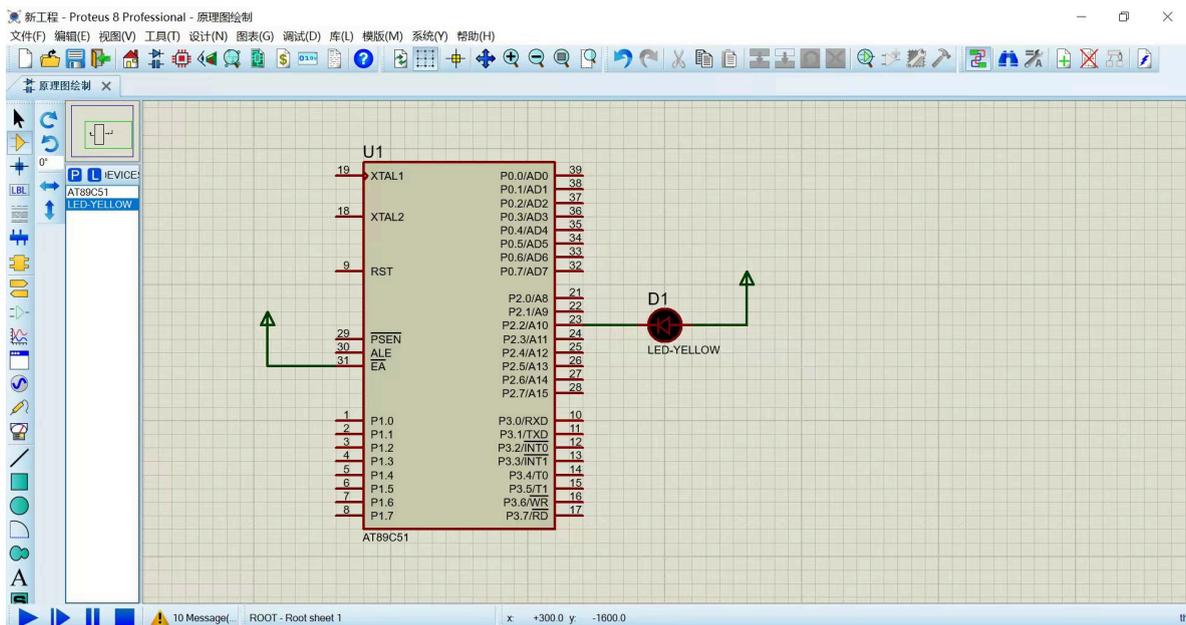
3. 外部I/O接口模块

- **功能** : 与外部设备 (如传感器、按钮、显示器等) 进行交互。
- **组成** :

- **RESPACK-8模块**：这是一个8位的开关阵列模块，通过P0口与单片机连接。可以是多个按钮或者开关的集合，单片机可以读取这些输入信号，用于控制其他操作或输出。
- **P0、P1、P2、P3口**：可以连接到LED、显示器、继电器、传感器等外设。不同端口的用途取决于具体的应用需求。

简单了解原理后，我们开始制作我们的第一个单片机程序——点亮一个LED灯

首先要把图纸做出来



好了之后，我们打开Keil5，新建工程，创建好main.c文件后，开始输入代码（灰色注释部分不用输入）

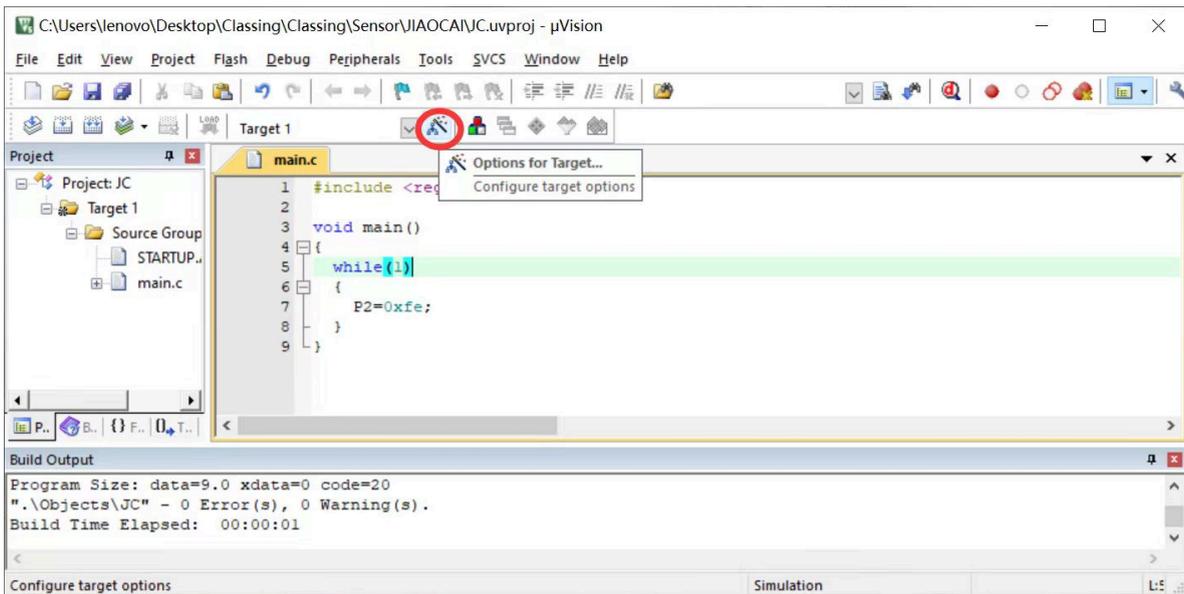
```

#include <reg52.h> //51头文件，为了让

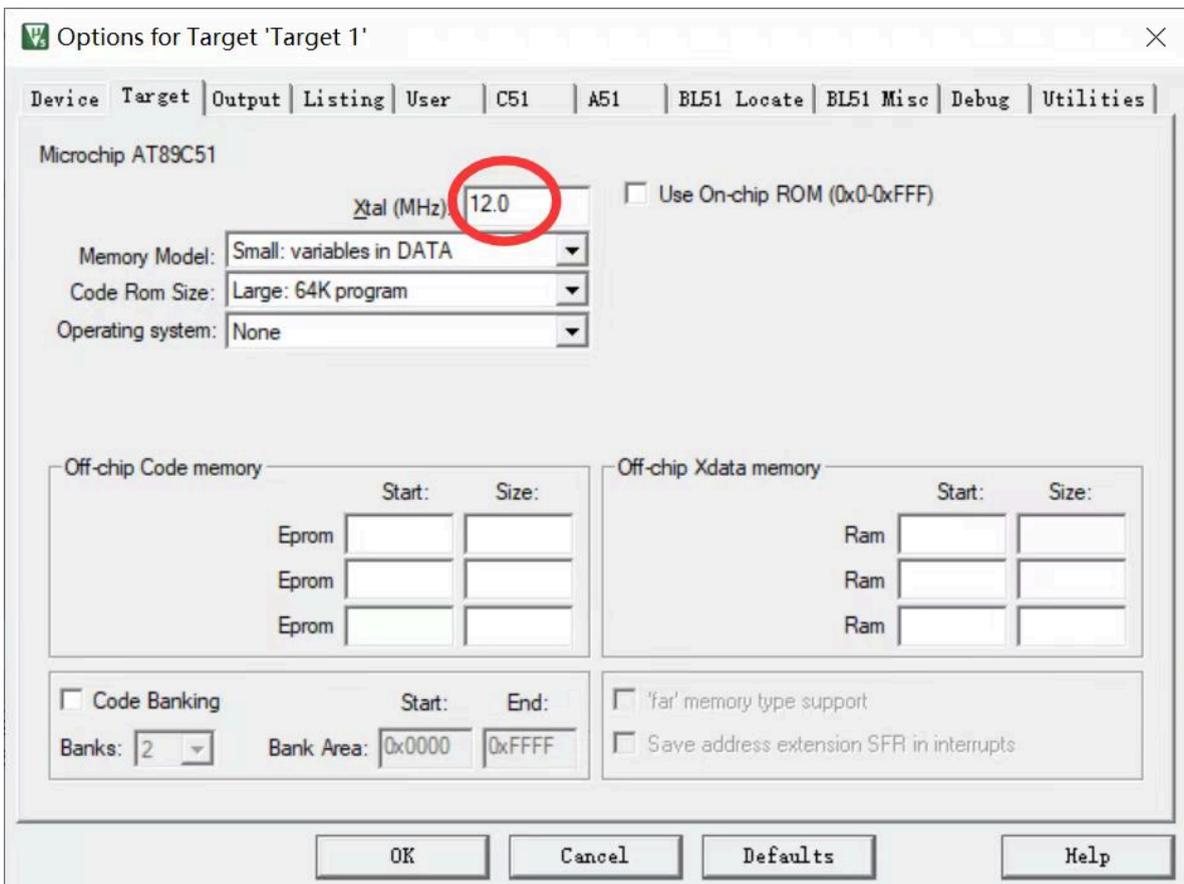
void main() //主函数，程序的入口，所有执行的操作
都从这里开始。
{
    while(1) //写出一个主循环，在单片机的应用中，
一般会在主循环中执行需要反复运行的任务。
    {
        P2=0xFB; //第三颗led灯点亮。FB为十六进制，转
化为二进制为11111011，LED的接法是正极接电源（共阳极）
    } //
}
/*P2.0（第1个引脚）输出高电平，LED灭。
P2.1（第2个引脚）输出高电平，LED灭。
P2.2（第3个引脚）输出低电平，LED亮。给低电平时，LED的负极
电信号便为0，正极因接电源所以电信号为1，故二极管导通发光
P2.3（第4个引脚）输出高电平，LED灭。其他二极管因两端都是
1，没有高低之分，故没有点亮
P2.4（第5个引脚）输出高电平，LED灭。
P2.5（第6个引脚）输出高电平，LED灭。
P2.6（第7个引脚）输出高电平，LED灭。
P2.7（第8个引脚）输出高电平，LED灭。
*/

```

输入代码，然后完成下列步骤，便完成了三大步的第一步——**编码**（下面图片中的代码不对，请抄上面的）

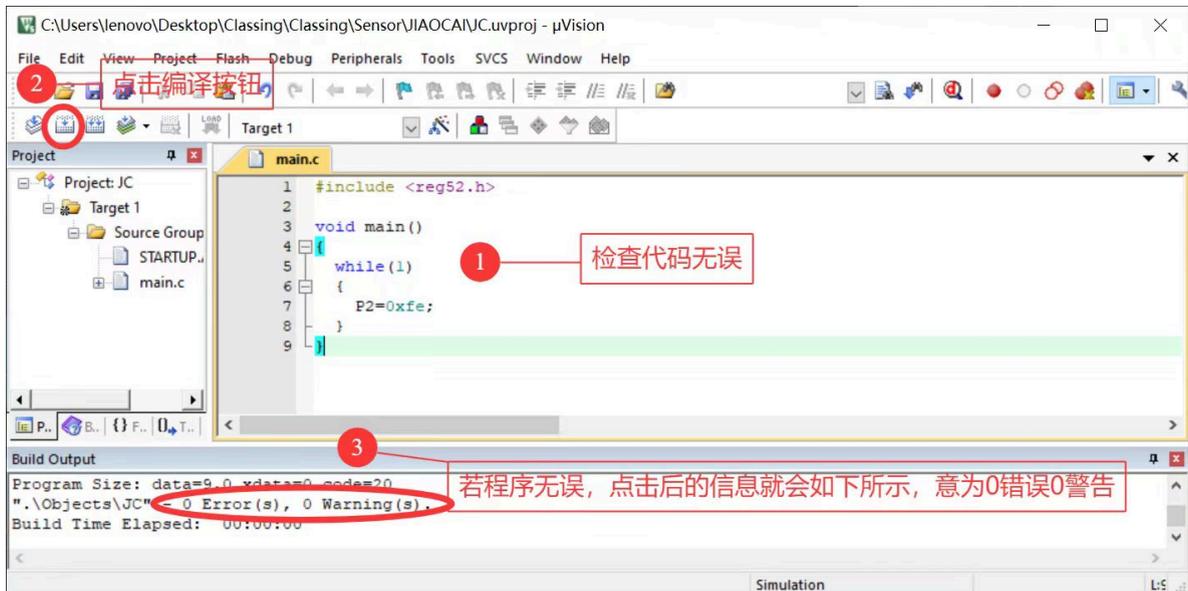


改成12.0，为我们常用的晶振频率

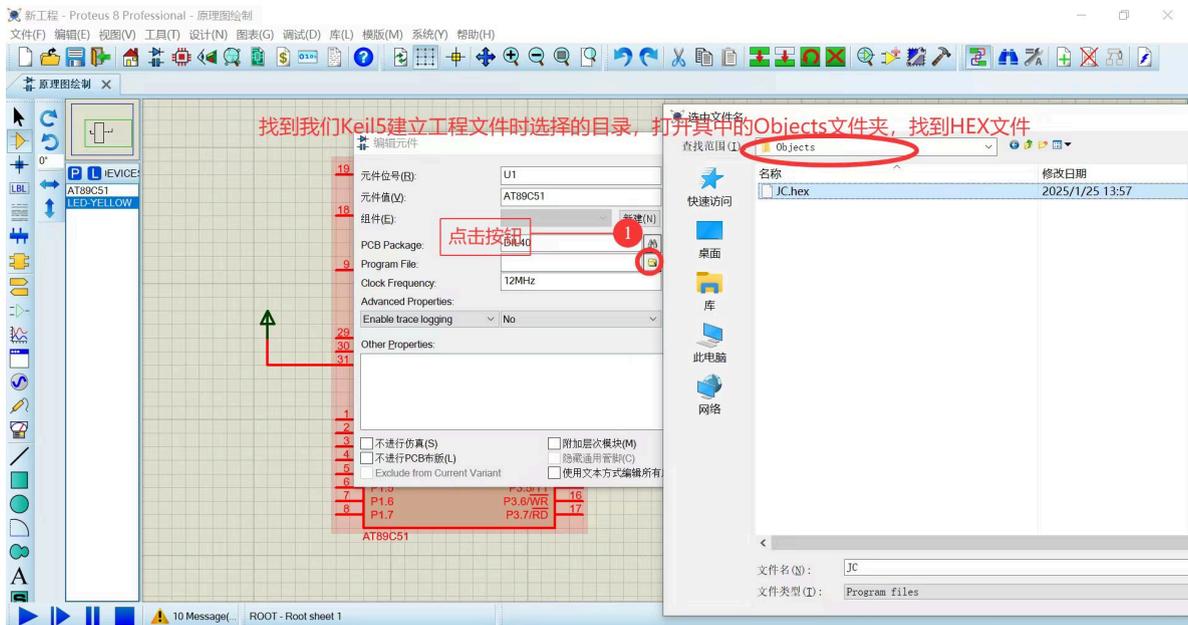


把勾打上，以便让代码由C语言文件编译为芯片可识别的HEX文件（一种十六进制文件）

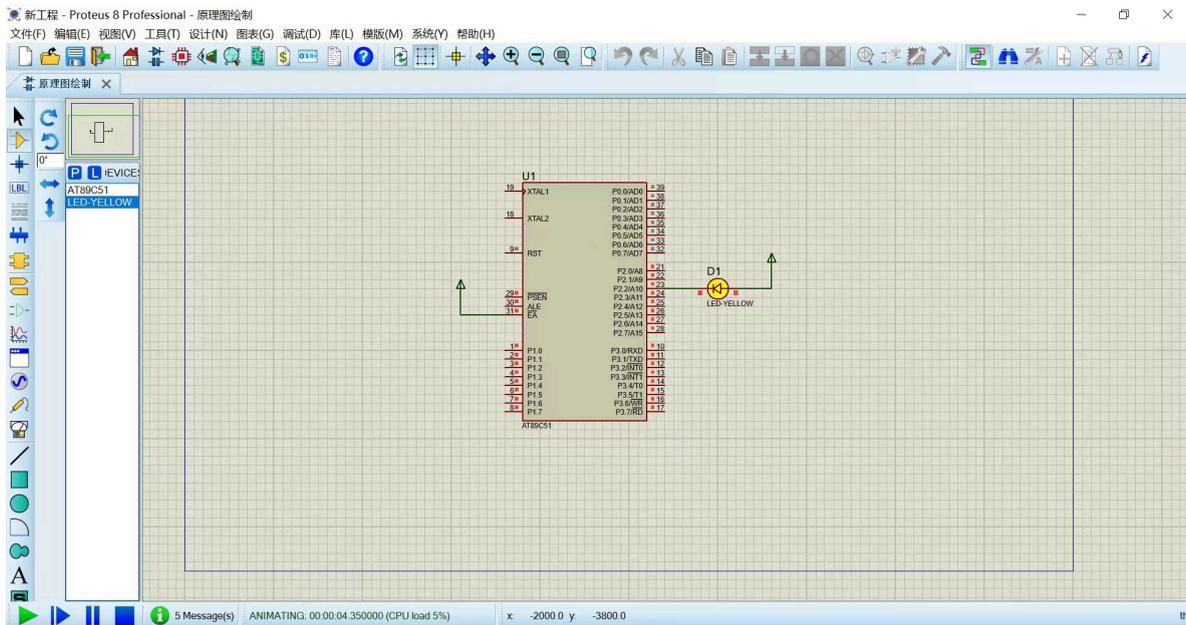
检查代码无误后，我们点击编译按钮即完成第二步——编译，此时HEX文件已经生成



之后我们打开Proteus，回到刚刚的图纸上，右键单片机点击编辑属性，并将刚刚的HEX文件导入单片机



完成之后，我们点击左下角的播放键即可运行



恭喜你点亮人生中第一颗虚拟灯泡🎉

***安装中各操作的解释**

第三章 实例应用与自主设计

第四章 连接烧录单片机

第五章 C语言基础及编译

C语言是一种高级编程语言，它具有结构化、简洁、高效等特点，广泛应用于嵌入式系统、操作系统以及各类硬件开发中。对于51单片机编程，C语言具有较高的可读性和可维护性，而且它可以通过汇编指令与硬件进行直接交互，充分利用51单片机的硬件资源。

第一节 C语言程序基本结构

我们先以简单的C语言程序来学习基本结构

本章大部分参考于：

[ChatGPT](#)

所有操作都为键盘切换至英文状态时的操作

写代码时注意格式规范和整洁，使用Tab键可以一次实现较大的空格

```
#include <stdio.h>           //定义头文件，即函数的仓库，告  
诉电脑我们需要用到哪一种函数，之前单片机也有它自己的仓库。  
  
int main()                   //定义主函数，主函数有多种定义  
（比如还有之前单片机的void main）  
{                             //紧贴着main此括号开始为程序的  
主体，程序从上到下依次运行  
    printf("hello world!");   //printf为函数名  
    return 0;                //固定格式，写就行了  
}
```

牢记框架，以后下面来介绍我们遇到的第一个函数：

`printf`：用于输出一个变量，(" ")中为我们输出的内容

因为我们没有装C语言的编译器，所以我们使用在线网站来检验结果

在线运行C语言

接下来我们把代码在网站中试试是否能成功



```
hello.c
1 #include <stdio.h> //定义头文件，即函数的仓库，告诉电脑我们需要用到哪一种函数，之前单片机也有它自己的仓库。
2
3 int main() //定义主函数，主函数有多种定
4 { //紧贴着main此括号开始为程序的主体，程序从上到下依次运行
5     printf("hello world!"); //printf为函数名
6     return 0; //固定格式，写就行了
7 } //程序主体的结束
```

1.在框中输入好代码

2.点击运行

Output Empty

标准输出: 3.查看结果，如果是这样那就对了

```
hello world!
```

第二节 数据类型与变量

常见的数据类型：

整数型 (int)

用途：用来存储整数，比如 1, 42, -99。

大小：通常占 4 个字节，能存储的范围很大，比如 -2147483648 到 2147483647（具体范围和系统有关）。

例子：

```
int age = 18; // 表示年龄的变量，值是 18。
```

浮点型 (float)

用途：用来存储带小数的数值，比如 3.14, -0.618。

大小：通常占 4 个字节，能存储小数的精确值。

例子：

```
float price \= 19.99; // 商品价格的变量，值是 19.99。
```

字符型 (char)

用途：用来存储单个字符，比如字母 'A'、符号 '#' 或数字 '5'（注意这里是字符形式）。

大小：通常占 1 个字节。

例子：

```
char grade \= 'A'; // 表示成绩等级的变量，值是 'A'。
```

如何定义变量？

在使用变量前，我们需要先定义它，也就是告诉程序“我要一个什么类型的容器，名字叫什么”。

需要我们这样表示

```
变量名 = 赋的值
```

(这里要区分一下 “=”并不是等于的意思，而是赋值的意思 而 “==”才是等于)

例子如下：

```
int age;    // 定义一个整数型变量，名字叫 age。  
float pi;   // 定义一个浮点型变量，名字叫 pi。  
char grade; // 定义一个字符型变量，名字叫 grade。
```

如何给变量赋值？

赋值语法：

```
变量名 = 值;
```

例子：

```
age = 18;    // 给 age 这个变量赋值 18。  
pi = 3.14;   // 给 pi 这个变量赋值 3.14。  
grade = 'A'; // 给 grade 这个变量赋值 'A'。
```

还有一种表示方法为：

变量类型 变量名 = 要赋的值

(即把定义变量和赋值这两步合并为一步)

```
int age = 18;  
float pi = 3.14;  
char grade = 'A';
```

变量命名的规则

1. 只能包含字母、数字和下划线，不能包含空格或其他符号。
2. 不能以数字开头。
3. 不能使用关键字 (比如 `int`、`return`)。

第三节 运算符与表达式

C语言中的运算符主要分为以下几类：

算术运算符

赋值运算符

比较运算符

1. 算术运算符

用于基本的数学运算：加减乘除以及取余数。先乘除再加减，可以用括号提示哪一步先进行，不分中括号、小括号

运算符	含义	示例	结果
+	加法	5 + 3	8

运算符	含义	示例	结果
-	减法	10 - 6	4
*	乘法	4 * 3	12
/	除法	8 / 2	4
%	取余数	10 % 3	1

例子：

```
int a = 15, b = 4;
int sum = a + b;    // 加法, sum = 19
int diff = a - b;   // 减法, diff = 11
int prod = a * b;   // 乘法, prod = 60
int quo = a / b;    // 除法, quo = 3
int rem = a % b;    // 取余, rem = 3

int b;
b = ((1 + 2) + 2) / 4;
//或者写作👉
int b;
b = 1 + 2;
b = b + 2;
b = b / 4;
//或者
int b = ((1 + 2) + 2) / 4;
//总之格式只要合规, 都是正确的
```

2. 赋值运算符

赋值运算符用来给变量赋值。

运算符	含义	示例
=	直接赋值	<code>x = 10</code>
+=	加后赋值，相当于 <code>x = x + 5</code>	<code>x += 5</code>
-=	减后赋值，相当于 <code>x = x - 3</code>	<code>x -= 3</code>
*=	乘后赋值，相当于 <code>x = x * 2</code>	<code>x *= 2</code>
/=	除后赋值，相当于 <code>x = x / 4</code>	<code>x /= 4</code>
%=	取余后赋值，相当于 <code>x = x % 2</code>	<code>x %= 2</code>

例子：

```
int x = 10;  
x += 5; // 相当于 x = x + 5, 结果 x = 15。  
x *= 2; // 相当于 x = x * 2, 结果 x = 30。
```

3. 比较运算符

比较两个值的大小，返回结果是真（1）或假（0）。

运算符	含义	示例	结果
==	等于	<code>5 == 5</code>	1
!=	不等于	<code>5 != 3</code>	1
>	大于	<code>10 > 5</code>	1
<	小于	<code>5 < 3</code>	0

运算符	含义	示例	结果
>=	大于等于	6 >= 6	1
<=	小于等于	7 <= 10	1

例子：

```
int a = 5, b = 10;
int result1 = (a < b); // 5 < 10, 结果是 1。
int result2 = (a == b); // 5 == 10, 结果是 0。
int result3 = (b >= 10); // 10 >= 10, 结果是 1。
```

4. 逻辑运算符

用于判断多个条件是否同时成立，返回结果是**真（1）或假（0）**。

与：只有当两个条件都为真时，结果才为真。否则结果为假。

或：只要其中一个条件为真，结果就是**真**。

非：将条件的真假取反，真变假，假变真。（若有多个只对**最终结果**的真假进行取反）

运算符	含义	示例	结果
&&	与（并且）	(3 > 2) && (5 > 3)	1
	或（或者）	(3 \> 2) (5 \< 3)	1
!	非（取反）	!(3 > 2)	0

代码中例子：

```
int a = 5, b = 10, c = 15;
if ((a < b) && (b < c)) {
    // 条件成立, 因为 a < b 并且 b < c。
}
if ((a > b) || (b < c)) {
    // 条件成立, 因为 b < c。
}
if (!(a > b)) {
    // 条件成立, 因为 a > b 是假, 取反后为真。
}
```

5. 自增与自减运算符

用于让变量的值增加或减少 1。因符号位置不同而分前后

运算符	含义	示例	结果
++	自增, 增加 1	x++ 或 ++x	x = x + 1
--	自减, 减少 1	x-- 或 --x	x = x - 1

区别：

- `x++` : 先使用 `x` 的值, 再让 `x` 加 1。
- `++x` : 先让 `x` 加 1, 再使用 `x` 的值。

例子：

```
int a = 5;
int b = ++a; // 前缀自增：a 先加 1，再赋给 b，a 变成 6，
b 变成 6。
```

```
int x = 5;
int y = x++; // 后缀自增：y 先等于 x 的原值，再将 x 加
1，x 变成 6，y 变成 5。
```

6.运算符优先级

当一个表达式中包含多个运算符时，优先级决定了运算的先后顺序。

优先级从高到低：

1. `()`：括号优先级最高，强制指定运算顺序。
2. `++`、`--`：自增、自减。
3. `*`、`/`、`%`：乘法、除法、取余。
4. `+`、`-`：加法、减法。
5. `==`、`!=`、`>`、`<`：比较运算符。
6. `&&`、`||`：逻辑运算符。

例子：

```
int result = 2 + 3 * 4; // 先计算 3 * 4, 结果是 12, 然后加 2, 结果是 14。  
int result2 = (2 + 3) * 4; // 先计算 2 + 3, 结果是 5, 然后乘 4, 结果是 20。
```

第四节 控制语句

控制语句是程序中用于控制执行流程的语句。通过控制语句，我们可以实现**条件判断**、**循环执行**以及**跳转操作**。

1. 条件语句

1.1 if 语句

`if` 语句根据条件判断是否执行某段代码。

语法：

```
if (条件) {  
    // 条件为真时执行的代码  
}
```

示例：

```
int a = 10;
if (a > 5) {
    printf("a 大于 5");
}
```

//可以直接从字面意思读，如果a>5，那么输出a 大于 5。在开头给a赋值为5，所以经过判断，输出a 大于 5

1.2 if-else 语句

`if-else` 用于在条件为假时执行另一段代码。

语法：

```
if (条件) {
    // 条件为真时执行的代码
} else {
    // 条件为假时执行的代码
}
```

示例：

```
int a = 3;
if (a > 5) {
    printf("a 大于 5\n");
} else {
```

```
printf("a 小于或等于 5\n");  
}  
/*
```

也可以按照字面意思看，如果 $a > 5$ ，输出a 大于 5，否则输出a 小于等于 5。

\n 告诉程序在输出时换到下一行

原本的结果为： a 大于 5 a 小于或等于 5

变为：

a 大于 5

a 小于或等于 5

```
*/
```

1.3 else-if 语句

多个条件判断可以用 `else if`。

示例：

```
int score = 85;  
if (score >= 90) {  
    printf("优秀\n");  
} else if (score >= 60) {  
    printf("及格\n");  
} else {  
    printf("不及格\n");  
}
```

2.循环语句

2.1 for 循环

`for` 循环常用于执行固定次数的循环。

语法：

```
for (初始化; 条件; 更新) {  
    // 循环执行的代码  
}
```

示例：

```
for (int i = 0; i < 5; i++) {  
    printf("i = %d\n", i);  
}
```

/*

1 先定义变量*i* 初始值为0

2 检查*i*是否小于5 如果条件为真，则执行循环 `printf("i = %d\n", i);`

3 如果条件为假，结束循环。

`%d`表示输出为一个十进制的整数

每次循环的值：

循环次数	<i>i</i> 的值	输出
第 1 次	0	<code>i = 0</code>
第 2 次	1	<code>i = 1</code>

```
第 3 次      2      i = 2
第 4 次      3      i = 3
第 5 次      4      i = 4
```

最终输出结果为：

```
i = 0
i = 1
i = 2
i = 3
i = 4
*/
```

2.2 while 循环

`while` 循环在条件为真时持续执行。

语法：

```
while (条件) {
    // 条件为真时执行的代码
}
```

示例：

```
int i = 0;
while (i < 5) {
```

```
printf("i = %d\n", i);  
i++;  
}  
/*
```

1 定义变量*i* 初始值为0

2 检查 $i < 5$ 是否成立，如果条件为真，执行循环 `printf("i = %d\n", i);` 和 `i++`。

3 如果条件为假，退出循环。

每次循环的值：

循环次数	<i>i</i> 的值 (进入循环时)	输出
第 1 次	0	<i>i</i> = 0
1		
第 2 次	1	<i>i</i> = 1
2		
第 3 次	2	<i>i</i> = 2
3		
第 4 次	3	<i>i</i> = 3
4		
第 5 次	4	<i>i</i> = 4
5		

最终输出结果：

```
i = 0  
i = 1  
i = 2  
i = 3  
i = 4  
/*
```

虽然 `for` 和 `while` 都能实现循环效果，但是通常在我们已知循环的次数或范围时，会选择用 `for` 循环

在当循环的次数未知，只依赖某个条件为真时继续执行，使用 `while` 循环更合适。

可以看出`for`循环的条件检查和变量更新部分都现在了循环头部
(`i < 5; i++`)

而`while`循环的这两部分写在循环中，更分散，逻辑灵活，适合不确定循环次数的情况。

2.3 do-while 循环

`do-while` 循环至少会执行一次。

示例：

```
int i = 0;
do {
    printf("i = %d\n", i);
    i++;
} while (i < 5);
//条件真时它就是一个普通的while循环，请看下面👉

int i = 9;
do {
    printf("i = %d\n", i); // 输出 "i = 9"
} while (i < 5);
```

```
//先执行循环体，再检查条件。即使条件一开始是假的，循环也会执行至少一次。此时条件一开始即为假的，所以直接输出i = 9
```

3. 跳转语句

3.1 break

`break` 用于提前终止循环。

示例：

```
for (int i = 0; i < 10; i++) {  
    if (i == 5) {  
        break;  
    }  
    printf("i = %d\n", i);  
}
```

```
/*
```

当 `i == 5` 时，`break` 会立刻终止整个循环，无论后续条件是否成立。

```
i = 0
```

```
i = 1
```

```
i = 2
```

```
i = 3
```

```
i = 4
```

```
*/
```

3.2 continue

`continue` 用于跳过当前循环，直接进入下一次循环。

示例：

```
for (int i = 0; i < 10; i++) {  
    if (i % 2 == 0) {  
        continue;  
    }  
    printf("i = %d\n", i);  
}
```

/*

`continue`可以直接跳过当前循环后续代码，进入下一次循环。
该代码旨在实现输出奇数、跳过偶数。

结果为：

i = 1

i = 3

i = 5

i = 7

i = 9

*/

思考#1

前五题：难度低 如果你能答对，看代码之类的不成问题，之后开发也可以从实例中见微知著，不至于一头雾水。

后五题：难度适中 可以培养自己的思维，在以后开发中能有自己的见解。

1. 判断题

以下代码的输出是什么？

```
int x = 8;
if (x > 10) {
    printf("x 大于 10\n");
} else {
    printf("x 小于或等于 10\n");
}
```

- A. x大于10
- B. x小于或等于10
- C. 没有输出

2. 填空题

补全以下代码，使它输出：

```
a = 15
```

```
#include <stdio.h>

int main() {
```

```
int a = ___;  
printf("a = ___\n");  
return 0;  
}
```

3. 多选题

以下哪些是合法的变量名？

- A. `int num1;`
- B. `float 2value;`
- C. `char _temp;`
- D. `double x+y;`

4. 简答题

分析以下代码：

```
int a = 5, b = 10;  
int result = (a + b) * 2;  
printf("结果是 %d\n", result);
```

输出结果是什么？

5. 编程题

写一个程序，要求：

1 定义一个变量 num , 初始化为 7。

2 如果 num 是偶数 , 输出 num是偶数 ; 否则输出 num是奇数 。

6. 判断题

以下代码的输出是什么 ?

```
int a = 10, b = 20;
if (a < b && b > 15) {
    printf("条件成立\n");
}
```

- A. 条件成立
- B. 无输出

7. 填空题

补全以下代码 , 使其输出 :

结果是: 28

```
#include <stdio.h>

int main() {
```

```
int a = 4, b = 5;
int result = ___;
printf("结果是:%d\n", result);
return 0;
}
```

8. 多选题

以下哪些语句会输出 10 ?

```
int x = 10, y = 20;
```

- A. `printf("%d\n", x);`
- B. `printf("%d\n", y);`
- C. `printf("%d\n", x + y - 20);`
- D. `printf("%d\n", x * 2);`

9. 编程题

写一个程序，要求：

定义两个变量 x 和 y 。

如果 x 大于 y ，输出 x 比 y 大。

否则输出 y 比 x 大。（赋值随意即可）

10. 简答题

分析以下代码：

```
int x = 3, y = 5;
x += y;
y *= 2;
printf("x = %d, y = %d\n", x, y);
```

程序会输出什么？

答案：

1. B
2. 15 ; %d
3. A C
4. 30
- 5.

```
#include <stdio.h>

int main() {
    int num = 7;

    if (num % 2 == 0) {
        printf("%d 是偶数\n", num);
    } else {
        printf("%d 是奇数\n", num);
    }
}
```

```
    return 0;
}
```

6. A

7.

```
#include <stdio.h>

int main() {
    int a = 4, b = 5;
    int result = a * b + a * 2; // 4*5 + 4*2 =
20 + 8 = 28
    printf("结果是 :%d\n", result);
    return 0;
}
```

8. AC

9.

```
#include <stdio.h>

int main() {
    int x = 10, y = 5; // 定义并初始化变量 x 和 y

    if (x > y) {
        printf("x 比 y 大\n");
    } else {
        printf("y 比 x 大\n");
    }
}
```

```
    return 0;  
}
```

10. $x = 8, y = 10$

第五节 数组

数组是一种数据结构，用来存储多个相同类型的数据。数组的每个元素可以通过索引来访问，索引从 0 开始。数组的大小（即元素的数量）在创建时确定，并且不可更改。

数组的定义：

数组的定义格式是：

```
类型 数组名[数组大小];
```

示例：

```
int arr[5]; // 定义一个包含 5 个整数的数组
```

数组初始化：

可以在定义数组时直接初始化它的值：

```
int arr[5] = {1, 2, 3, 4, 5}; // 初始化一个包含 5 个整数的数组
```

//如果没有给所有元素赋初值，未赋值的元素会被默认初始化为 0。



```
int arr[5] = {1, 2}; // 剩余元素将被默认初始化为 0，结果为 {1, 2, 0, 0, 0}
```

访问数组元素：

数组通过索引访问元素，索引从 0 开始：

```
int arr[5] = {10, 20, 30, 40, 50};  
printf("%d\n", arr[0]); // 输出 10  
printf("%d\n", arr[2]); // 输出 30
```

遍历数组：

可以通过循环访问数组的所有元素：

```
#include <stdio.h>  
  
int main() {  
    int arr[5] = {1, 2, 3, 4, 5}; //定义了一个整数类型的数组，名字叫 arr，包含 5 个元素。
```

```
    for (int i = 0; i < 5; i++) { //i初始值为0,只要i小
于5,继续循环,每次循环i+1
        printf("arr[%d] = %d\n", i, arr[i]);
    }
/*
%d 是占位符,用来输出整数。
arr[%d]:表示数组的第 i 个元素。
arr[i]:获取索引为 i 的数组元素。
*/
    return 0;
}
/*
最终输出为
arr[0] = 1
arr[1] = 2
arr[2] = 3
arr[3] = 4
arr[4] = 5
*/
```

数组的特点：

固定大小：数组的大小在定义时确定，不能更改。

类型一致：数组中的所有元素类型相同。

索引访问：通过索引快速访问元素。

第六节 函数

之前我们在第一节已经对函数略知一二，接下来再深入学习一些。

什么是函数？

函数是一段代码的集合，用来完成一个特定的任务。之前我们看过了，我们可以自己定义函数，然后在这个函数中让他实现怎样的运算。

函数可以接收输入（参数），执行操作后返回结果。

函数的基本结构

语法：

```
返回值类型 函数名(参数列表) {  
    // 函数体：实现具体的功能  
    return 返回值; // 如果返回值类型是 void，可以省略  
}  
//如果你不明白 返回值类型 这个名字的意义 那就看看它的全称：函数  
//执行完毕后所返回数据的类型
```

示例：

```
#include <stdio.h>  
  
// 定义一个函数  
int add(int a, int b) { //返回值类型int 函数名add (参数  
列表)(int a,int b)  
    return a + b; // 返回两个数的和  
}
```

```
int main() {
    int result = add(3, 5); // 调用函数并传入参数
    printf("结果是:%d\n", result);
    return 0;
}
//输出结果为： 结果是：8
```

函数的组成部分

返回值类型：函数返回的数据类型（如 int、float、void 等）。

函数名：函数的名字，用于调用它。

参数列表：传递给函数的数据，多个参数用逗号分隔。（比如 add(3,5)）

函数体：函数执行的代码块（基本上就是括号里的内容）。

return 语句：返回函数的结果（void类型的函数可以没有）。

函数嵌套调用：

函数可以在另一个函数中调用：

```
#include <stdio.h>

int square(int x) {
    return x * x;
}

int sumOfSquares(int a, int b) {
    return square(a) + square(b); // 调用 square 函
```

```
数  
}
```

```
int main() {  
    int result = sumOfSquares(3, 4); // 调用  
    sumOfSquares 函数  
    printf("结果是 : %d\n", result);  
    return 0;  
}
```

接下来我要题型你，之后几章的内容我自己也学起来很蛋疼，但是多看看还是能看懂。

第七节 指针

指针是 C 语言中一个非常重要但略显复杂的概念。它能让程序更灵活高效，但对于初学者来说，理解起来可能需要点耐心。让我们用简单易懂的方式来学习指针！

什么是指针？

指针是一个变量，它**存储了另一个变量的地址**。

每个变量在内存中都有一个唯一的地址。

指针的作用就是记录这些地址。

那么什么是内存？我们之后再说.....

举个例子：

假设有一个变量 a，它的值是 10，存储在内存地址 0x100。

指针可以用来保存这个地址 0x100，并通过这个地址访问或修改 a 的值。

指针的定义

```
类型名 *指针变量名;
```

类型名：指针指向的数据类型，比如 int 表示指针指向一个整数。

星号 *：表示这是一个指针变量。

指针变量名：指针的名称。

示例：

```
int *p; // 定义一个指向整数的指针 p
```

获取变量地址：

C 语言中可以使用 **取地址符** & 获取变量的内存地址。

```
int a = 10;  
int *p = &a; // 将变量 a 的地址赋给指针 p  
/*  
&a 表示变量 a 的地址。*/
```

```
p 保存了这个地址。  
*/
```

通过指针访问变量的值:

可以使用 **解引用符** * 访问指针所指向的变量值。

```
int a = 10;  
int *p = &a;    // p 保存 a 的地址  
  
printf("a 的值是 : %d\n", *p); // 解引用指针, 输出 a 的值  
*p = 20; // 修改指针指向的值, 相当于修改 a 的值  
printf("a 修改后的值是 : %d\n", a);  
/*  
运行结果为 :  
a 的值是 : 10  
a 修改后的值是 : 20  
*/
```

NULL指针

指针在使用之前, 必须指向一个合法的地址。如果一个指针没有被初始化, 可以将它设置为 NULL, 表示它不指向任何地址。

示例 :

```
int *p = NULL; // 定义一个空指针
```

//使用空指针时需要特别小心，访问或解引用空指针会导致程序崩溃▲

指针的简单示例：

1：打印变量地址

```
#include <stdio.h>

int main() {
    int a = 10;
    printf("a 的地址是：%p\n", &a); // 使用 %p 打印地址
    return 0; /*打印变量 a 的地址。
&a：取地址运算符，获取变量 a 的内存地址。
%p：格式化输出地址。内存地址通常是十六进制格式，例如
0x7ffc1234abcd
*/
}

//输出结果：a 的地址是：0x7ffeefbfff5c8
```

2.指针和变量

```
#include <stdio.h>

int main() {
    int a = 10;
    int *p = &a; // 定义一个指针 p，指向 a 的地址
```

```

printf("a 的值是 :%d\n", a);
printf("通过指针访问 a 的值是 :%d\n", *p);

*p = 20; // 修改指针指向的值
printf("a 修改后的值是 :%d\n", a);

return 0;
}
/*运行结果： a 的值是 :10
通过指针访问 a 的值是 :10
a 修改后的值是 :20
*/

```

3.指针和数组

```

#include <stdio.h>

int main() {
    int arr[3] = {1, 2, 3};
    int *p = arr; // 指针指向数组的第一个元素

    for (int i = 0; i < 3; i++) {
        printf("arr[%d] = %d\n", i, *(p + i)); //
用指针访问数组元素
    }

    return 0;
}
/*arr[0] = 1

```

```
arr[1] = 2
arr[2] = 3
*/
```

第八节 结构体与联合体

结构体

结构体是一种聚合数据类型，可以将多个变量组合在一起，这些变量可以是不同的类型。

先看代码

```
#include <stdio.h>

// 定义结构体类型
struct Person { //Person 是结构体的名字。
    char name[50]; //结构体包含三个成员：name（字符串）、age（整数）和 height（浮点数）。
    int age;
    float height;
};

int main() {
    // 定义结构体变量
    struct Person person1 = {"Alice", 25, 1.68}; //定义 person1 变量，并初始化。

    // 访问结构体成员
```

```

    printf("姓名: %s\n", person1.name); //使用点运算符
    (. ) 访问和修改结构体的成员。
    printf("年龄: %d\n", person1.age);
    printf("身高: %.2f米\n", person1.height);

    // 修改成员的值
    person1.age = 26;
    printf("更新后的年龄: %d\n", person1.age);

    return 0;
}

/*运行结果为：姓名: Alice
年龄: 25
身高: 1.68米
更新后的年龄: 26
*/

```

联合体

联合体是一种特殊的数据类型，它的所有成员共享同一块内存，因此每次只能存储一个成员的值。

再看代码：

```

#include <stdio.h>

// 定义联合体
union Data {

```

```

    int i;
    float f;
    char str[20];
};
/*使用 union 关键字定义联合体类型。
Data 是联合体的名字。
联合体包含三个成员：i（整数）、f（浮点数）和 str（字符串）。
*/
int main() {
    // 定义联合体变量data
    union Data data;

    // 为成员赋值
    data.i = 10;
    printf("i = %d\n", data.i);

    data.f = 3.14;
    printf("f = %.2f\n", data.f);
/*赋值 data.i 后再赋值 data.f，会覆盖前一个成员的值。
联合体的大小等于最大成员的大小。
*/
    sprintf(data.str, "Hello");
    printf("str = %s\n", data.str);

    return 0;
}

/*最终结果：i = 10
            f = 3.14
            str = Hello
*/

```

结构体和联合体的区别

特性	结构体 (struct)	联合体 (union)
内存分配	每个成员有自己的内存，结构体的大小是所有成员大小的总和。	所有成员共享同一块内存，大小等于最大成员的大小。
数据存储	每个成员可以同时存储不同的数据。	每次只能存储一个成员的数据，后面的值会覆盖前面的值。
用途	用于存储多个数据，每个成员都有独立的意义。	用于节省内存，需要在不同时间存储不同类型的数据。

思考#2

1. 数组

题目：

定义一个长度为 3 的整型数组，初始化为 {10,20,30}，输出数组的第一个元素。

要求：

直接输出数组的第一个元素。

2. 函数

题目：

编写一个函数 `int square(int x)`，返回整数 `x` 的平方。在主函数中调用该函数并输出结果。

要求：

定义一个整数变量，调用 `square` 函数，并输出结果。

3. 指针

题目：

定义一个整数变量 num，将它的地址传递给指针 ptr，通过指针修改 num 的值为 50，并输出修改后的值。

要求：

- 使用指针访问并修改变量 num 的值。

4. 结构体

题目：

定义一个结构体 Person，包含姓名（char[10]）和年龄（int）。在主函数中创建一个 Person 结构体变量，并赋值后输出姓名和年龄。

要求：

定义结构体并赋值。

输出结构体的成员。

5. 联合体

题目：

定义一个联合体 Data，包含一个整数 i 和一个浮点数 f。在主函数中为联合体的 i 赋值并输出它。

要求：

定义联合体并赋值给 i。

输出 i 的值。

这些题目更简单，适合刚开始接触相关概念的同学。

答案

1.

```
#include <stdio.h>

int main() {
    int arr[3] = {10, 20, 30};
    printf("数组第一个元素是: %d\n", arr[0]);
    return 0;
}
```

2.

```
#include <stdio.h>

int square(int x) {
    return x * x;
}

int main() {
    int result = square(5);
    printf("5 的平方是: %d\n", result);
    return 0;
}
```

3.

```
#include <stdio.h>

int main() {
    int num = 30;
    int *ptr = &num;

    *ptr = 50; // 通过指针修改 num 的值
    printf("修改后的 num 值是: %d\n", num);
    return 0;
}
```

4.

```
#include <stdio.h>

struct Person {
    char name[10];
    int age;
};

int main() {
    struct Person p = {"Tom", 20};
    printf("姓名: %s, 年龄: %d\n", p.name, p.age);
    return 0;
}
```

```
}
```

5.

```
#include <stdio.h>

union Data {
    int i;
    float f;
};

int main() {
    union Data d;
    d.i = 100;
    printf("联合体中整数 i 的值是: %d\n", d.i);
    return 0;
}
```

第六章 理论学习