

***Package na***

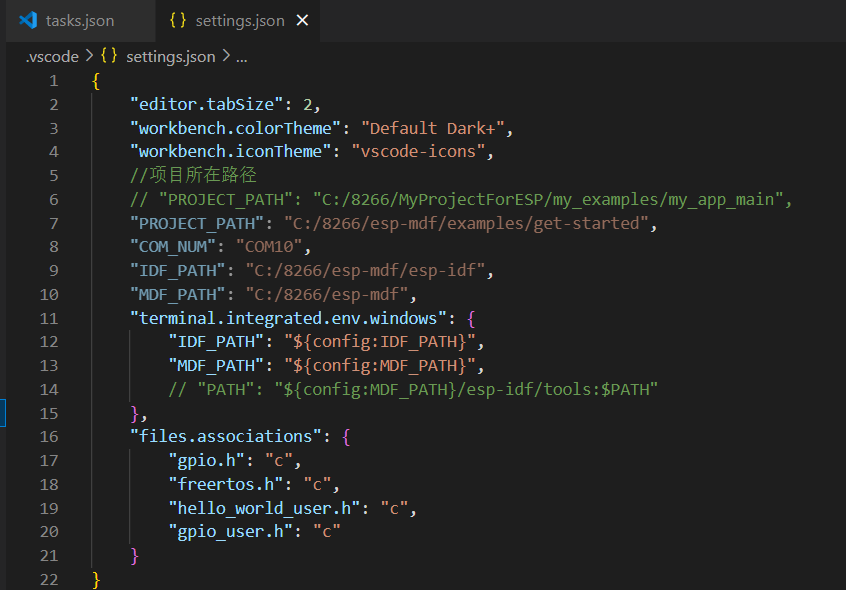
|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

# powershell

### 获取指定环境变量



|  |
| --- |
| PS C:\8266\esp-mdf> echo $env:IDF\_PATH  C:/8266/esp-mdf/esp-idf |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

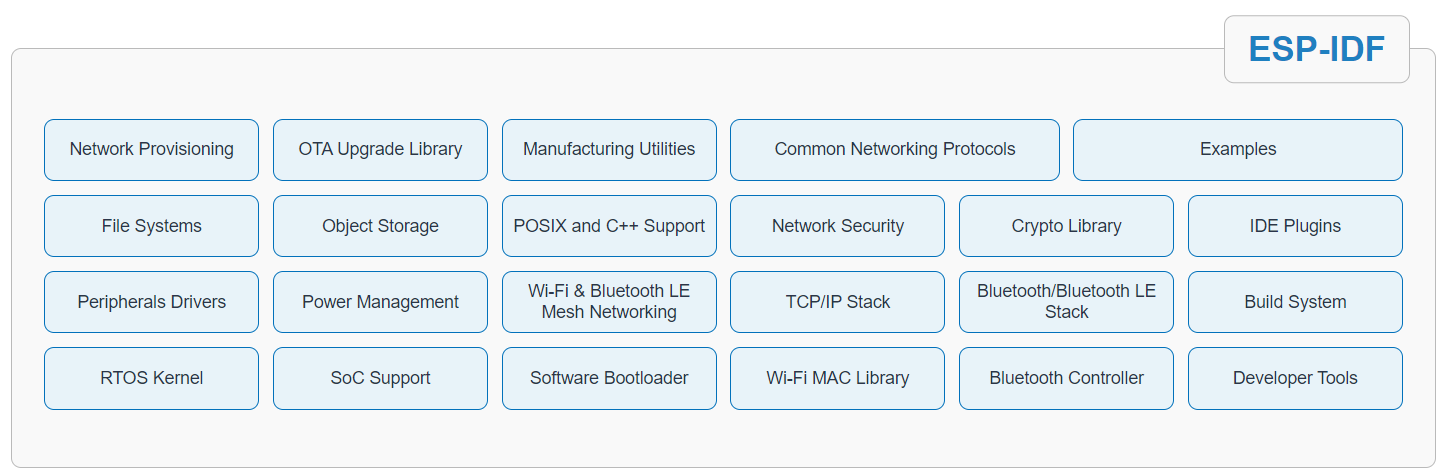
|  |
| --- |
|  |

|  |
| --- |
|  |

# 乐鑫官方的SDK理解

## esp-idf

ESP-IDF 是乐鑫官方的物联网开发框架，适用于 ESP32、ESP32-S、ESP32-C 和 ESP32-H 系列 SoC。它基于 C/C++ 语言提供了一个自给自足的 SDK，方便用户在这些平台上开发通用应用程序。ESP-IDF 目前已服务支持数以亿计的物联网设备，并已开发构建了多种物联网产品，例如照明、消费电子大小家电、支付终端、工控等各类物联网设备。







|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

# C语言10大基础算法，学C语言必会源码（珍藏版）

# 冒泡排序法

冒泡排序算法的原理如下：

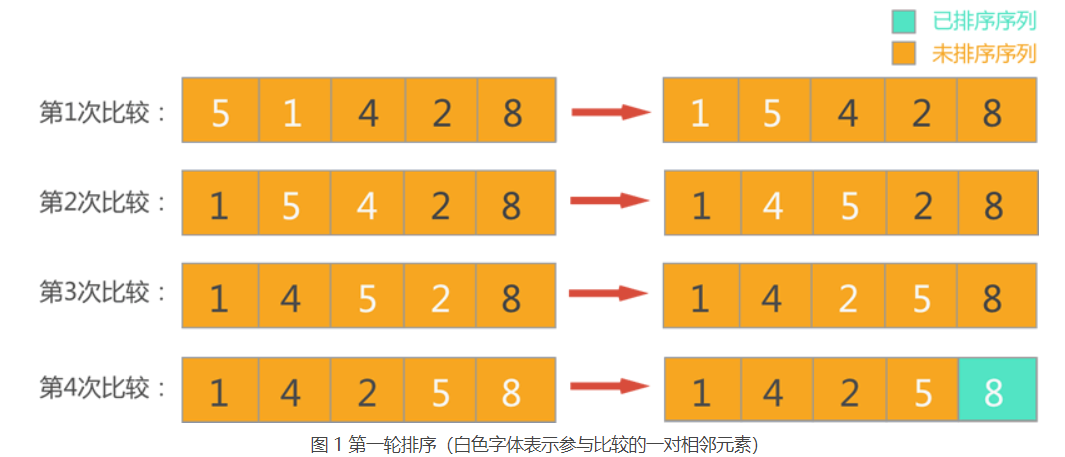
|  |
| --- |
| 1、比较相邻的元素。如果第一个比第二个大，就交换他们两个。 [1]  2、对每一对相邻元素做同样的工作，从开始第一对到结尾的最后一对。在这一点，最后的元素应该会是最大的数。 [1]  3、针对所有的元素重复以上的步骤，除了最后一个。 [1]  4、持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较。 |

对于从小到大排序，每次找到一个最大，然后将最大放在末尾，这样就减少一次循环次数，下次找到倒数第二大即可。

## 举例说明

假设待排序序列为 (5,1,4,2,8)，如果采用冒泡排序对其进行升序（由小到大）排序，则整个排序过程如下所示：

1) 第一轮排序，此时整个序列中的元素都位于待排序序列，依次扫描每对相邻的元素，并对顺序不正确的元素对交换位置，整个过程如图 1 所示。



# 中位值平均滤波法

如下截取我的小车的ADC电压采样的滤波方法，这是对中位值平均滤波法进行了改造

|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* 函数名称：static uint16\_t Filter(uint16\_t val)  \*\* 功能说明：中位值平均滤波法（又称防脉冲干扰平均滤波法）  \*\* 入口参数：采样值  \*\* 出口参数：中位值  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #define FILTER\_N 64  static uint16\_t Filter(uint16\_t index) {  uint32\_t i, j;  uint32\_t filter\_temp, filter\_sum = 0,posF=1;  uint16\_t buf[FILTER\_N];  for(i=0;i<FILTER\_N;i++)  buf[i]=ADCConvertedValue[index+i\*13];  // 采样值从小到大排列（冒泡法）  for(j = 0; j < (FILTER\_N - 1); j++) {  for(i = 0; i < (FILTER\_N - 1 - j); i++) {  if(buf[i] > buf[i + 1]) {  filter\_temp = buf[i];  buf[i] = buf[i + 1];  buf[i + 1] = filter\_temp;  }  }  }  filter\_sum = buf[0];  // 提取有效的采样值  for(i = 1; i < FILTER\_N; i++) {  float percent = (buf[i]-buf[i-1])\*1.f/(buf[i-1]\*1.f+0.001f); //计算相邻采样值的偏差量，后面的0.001f防止为0  if(percent > 0.1f){  if(posF > 30) break;  posF = 1;  filter\_sum = buf[i];  }else{  filter\_sum += buf[i];  posF += 1;  }  }  // 去除最大最小极值后求平均 //正常的中位值平均滤波法  // for(i = 1; i < (FILTER\_N - 1); i++) filter\_sum += buf[i];  // return filter\_sum / (FILTER\_N - 2);  return filter\_sum / posF;  } |