实验三：信号序列操作 C 语言实现

计算机学院 郭嘉睿 2413174

一、实验平台

编译器：gcc.exe (Rev8, Built by MSYS2 project) 15.2.0

版本控制：git version 2.48.1.windows.1

构建工具：GNU Make 4.4.1 built for Windows32

托管平台：GitHub

二、实验目的

本实验旨在以 C 语言实现一个通用的离散时间信号（Discrete-Time Signal）处理工具，采用命令行交互的形式，支持有限序列与无限流式输入。通过本实验，学生应能：

1. 验证各基本信号操作（补零、延迟、上采样、差分、累加等）的数学正确性；
2. 分析哪些信号操作可在线（Streaming）实现，哪些需离线全序列信息；
3. 设计模块化的 C 语言信号处理系统，理解函数抽象、结构体状态管理；
4. 观察与比照不同操作在有限与无限输入下的行为差异；
5. 掌握 GNU Make 构建体系与 Doxygen 注释规范，提升工程化能力。

三、实验核心问题

1. 核心操作定义

设输入序列（$x[n]$），输出序列（$y[n]$）：

操作	数学定义	说明
前补零 (pad-front)	$y[n]=\begin{cases}0, & n<k\\ x[n-k], & n\geq k\end{cases}$	在开头补 k 个零
延迟 (delay)	$y[n] = x[n-d]$	因果系统，可流式实现
提前 (advance)	$y[n] = x[n+a]$	需未来信息，非因果

操作	数学定义	说明
上采样 (upsample)	$y[n]=\begin{cases} x\left[\frac{m}{N}\right], m\ mod\ M=0 \\ 0, \quad others \end{cases}$	插入零
下采样 (downsample)	$y[n]=x[nM]$	抽取每 M 个样本
差分 (diff)	$y[n]=x[n]-x[n-1]$	一阶差分，流式可实现
累加 (cumsum)	$y[n]=\sum_{k=0}^{n} x[k]$	前缀和，可流式实现

2. 核心问题

- 如何在 有限内存 下实现随来随处理？
- 对于无限长序列，哪些操作可因果实现（不依赖未来样本）？
- C 程序如何设计统一的接口，既支持离线处理，也支持流式状态机？

四、实验设计

1. 输入输出设计

输入：命令行参数指定操作名与参数；数据由 stdin 输入。

输出：结果序列打印至 stdout；错误信息统一输出到 stderr。

2. 程序结构

采用五文件结构：

文件	功能
sequence.h	定义序列结构 seq_t 与流式状态 seq_stream_t
sequence.c	实现所有操作函数及流式接口
cli.h	声明命令行入口函数 cli_main()
cli.c	参数解析、输入读取、操作调用、错误提示
main.c	程序入口，调用 cli_main()

3. 核心设计思路

- 模块化实现：每个算子一个函数；离线与在线接口统一。
- 状态抽象：通过 seq_stream_t 保存操作类型、参数与缓冲区。
- 错误控制：所有错误以英文输出到 stderr，返回标准错误码。
- Doxygen 规范：每个函数、结构体双语注释。

五、代码与结果展示及分析

1. 核心代码示例：流式累加

```
seq_err_t seq_stream_step(seq_stream_t *st, int has_input, double x, double *y, int
*has_output)
{
    if (!st || !has_output) return SEQ_ERR_ARG;
    if (!st->active) return SEQ_ERR_STATE;

    *has_output = 0;

    switch (st->op) {
    case SEQ_OP_CUMSUM:
        if (!has_input) return SEQ_OK;
        if (!y) return SEQ_ERR_ARG;
        st->acc += x;
        *y = st->acc;
        *has_output = 1;
        return SEQ_OK;
    default:
        seq_log_error("unsupported op");
        return SEQ_ERR_UNSUPPORTED;
    }
}
```

说明：

- 本代码为流式累加核心逻辑。
- 每次输入样本（x），累积求和保存在 st->acc 中。
- 该操作满足因果性，可在线实时输出，无需未来样本。

2. 修复问题与调试分析

(1) CLI 参数解析错误

- 原实现错误地计算 param_count = argc - 2。
- 导致命令 seqops cumsum stream 报错 *too many parameters*。
- 修正为 param_count = argc - 3。

(2) 流式初始化崩溃问题

- 初始调用 seq_stream_init() 时，seq_stream_reset() 在未初始化变量上执行 free()。
- 导致 RtlFreeHeap SIGSEGV。
- 修复：seq_stream_reset() 仅清零字段，不执行 free()。

3. 实验结果截图（简要示例）

```
PS D:\Code\Digital-Signal-Processing\2> make
gcc -std=c11 -Og -g -c main.c -o main.o
gcc -std=c11 -Og -g -c cli.c -o cli.o
gcc -std=c11 -Og -g -c sequence.c -o sequence.o
gcc -std=c11 -Og -g -o seqops.exe main.o cli.o sequence.o
Build complete. Cleaning up object files...
make[1]: Entering directory 'D:/Code/Digital-Signal-Processing/2'
make[1]: Leaving directory 'D:/Code/Digital-Signal-Processing/2'
PS D:\Code\Digital-Signal-Processing\2> echo 1 2 3 4 END | ./seqops.exe cumsum stream
ONLINE:YES
1 3 6 10
```

所有结果符合数学定义与流式判定逻辑。

六、结果分析与讨论

1. 在线/离线操作区分明确： delay、diff、cumsum 等可实现实时计算；advance、reverse 需全局信息。
2. 代码稳定性： 修复后程序无内存错误，Windows 与 Linux 均可稳定运行。
3. 工程化实现：
    - 模块划分合理（≤5 文件）；
    - Makefile 自动清理；
    - Doxygen 可生成完整文档。
4. 边界情况：
    - 参数错误与空输入能正确报错；
    - 非数字 token 自动检测。

七、实验总结

本实验通过从零设计一个 C 语言信号处理 CLI 工具，验证了序列操作在有限与流式环境下的可行性与工程实现方法。最终系统实现了：

- 9 类基本操作（含数学与工程意义一致性验证）；
- 完整的错误检测与 ONLINE 判定机制；
- 平台无关的构建与执行。

本项目展示了从数学模型到可复用软件组件的完整工程路径，是数字信号处理与系统软件结合的典型案例。