```
#ifndef POCKETFLOW_H
#define POCKETFLOW_H

#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>

/* — Forward Declarations — */
typedef struct Params Params;
typedef struct Context Context;
typedef struct IBaseNode IBaseNode;

/* — The Interface (VTable) — */
typedef struct IBaseNodeVTable {
    void (*dtor)(void* self);
    void (*setParamsInternal)(void* self, const Params* params);
    char* (*internalRun)(void* self, Context* sharedContext);
    IBaseNode* (*getNextNode)(const void* self, const char* action);
    bool (*hasSuccessors)(const void* self);
    const char* (*getClassName)(const void* self);
    const Params* (*getParams)(const void* self);
} IBaseNodeVTable;

struct IBaseNode {
    const IBaseNodeVTable* vtable;
};

/* — Node Successor Map Replacement — */
typedef struct NodeSuccessor {
    char* action;
    IBaseNode* node;
    struct NodeSuccessor* next;
} NodeSuccessor;

/* — BaseNode — */
typedef struct BaseNode {
    union {
        IBaseNode interface;
        const IBaseNodeVTable* vtable; /* For direct access */
    };
    Params* params;
    NodeSuccessor* successors;
    const char* className;

    /* Logic Hooks (Prep/Exec/Post) */
    void* (*prep)(Context* ctx);
    void* (*exec)(void* self, void* prepRes);
    char* (*post)(Context* ctx, void* prepRes, void* execRes);
} BaseNode;

/* — Node (Retries & Fallback) — */
typedef struct Node {
    union {
        BaseNode base;
        const IBaseNodeVTable* vtable;
    };
    int maxRetries;
    long waitMillis;
    int currentRetry;
    void* (*execFallback)(void* prepResult, const char* lastError);
} Node;

/* — BatchNode — */
typedef struct {
    void** items;
    size_t count;
} Batch;

typedef struct BatchNode {
    union {
        Node node;
        const IBaseNodeVTable* vtable;
    };
    void* (*execItem)(void* item);
    void* (*execItemFallback)(void* item, const char* error);
} BatchNode;

/* — Flow (Orchestrator) — */
typedef struct Flow {
    union {
        BaseNode base;
        const IBaseNodeVTable* vtable;
    };
    IBaseNode* startNode;
} Flow;

/* — BatchFlow — */
typedef struct {
    Params** items;
    size_t count;
} ParamsBatch;

typedef struct BatchFlow {
    union {
        Flow flow;
        const IBaseNodeVTable* vtable;
    };
    ParamsBatch (*prepBatch)(void* self, Context* ctx);
    char* (*postBatch)(void* self, Context* ctx, ParamsBatch batchPrepResult);
} BatchFlow;

/* — Common Implementation Functions — */

static inline void IBaseNode_Destroy(IBaseNode* self) {
    if (self && self->vtable->dtor) self->vtable->dtor(self);
}

/* Helper to add transitions between nodes */
static inline void BaseNode_AddNext(BaseNode* self, IBaseNode* nextNode, const char* action) {
    NodeSuccessor* entry = malloc(sizeof(NodeSuccessor));
    entry->action = action ? strdup(action) : strdup("");
    entry->node = nextNode;
    entry->next = self->successors;
    self->successors = entry;
}

/* Core Orchestration Loop used by Flow and BatchFlow */
static inline char* Flow_Orchestrate(Flow* self, Context* ctx, Params* runParams) {
    IBaseNode* current = self->startNode;
    char* lastAction = NULL;
    while (current) {
        if (current->vtable->setParamsInternal)
            current->vtable->setParamsInternal(current, runParams ? runParams : self->base.params);

        char* action = current->vtable->internalRun(current, ctx);
        IBaseNode* next = current->vtable->getNextNode(current, action);

        if (lastAction) free(lastAction);
        lastAction = action;
        current = next;
    }
    return lastAction;
}

#endif /* POCKETFLOW_H */
```