Pure C language Patterns for mshell Workflow — Complete Reference Guide (p1–p24)

Pure bash language Edition — Art2Dec SoftLab, March 17th 2026

Pure Edition C language-Only · No Python, C, C++, Rust, Go, Lua or bash — C language only, `mshell` directives, and LLM blocks.

What is mshell?

mshell is a polyglot UNIX shell environment for AI and mathematics that integrates multiple programming languages with AI model capabilities into a single unified execution pipeline. Instead of writing separate scripts in different languages and manually wiring them together, mshell lets you define a workflow in a single Markdown document where each code block is a step in the pipeline.

This **C Language Edition** uses **C as the primary language** for all computation, data transformation, I/O, and pipeline logic. All 24 workflow patterns have been verified working in isolation.

Variable System

Variables flow between blocks via files in `/tmp/mshell_ctx_PID/`.

Syntax	Meaning
`c >varname`	stdout captured into variable
`c <varname`	`MSH_VAR_varname` set to variable file path
`c <a <b >c`	multiple inputs, one output

Reading a variable in C:

```c
FILE *f = fopen(getenv("MSH_VAR_input"), "r");
/* read from f */
fclose(f);
```

Writing a variable from C (multi-output block):

```c
FILE *f = fopen(getenv("MSH_VAR_output"), "w");
fprintf(f, "value\n");
fclose(f);
```

Advanced Node Types

Node	Syntax	Notes
WHILE	`<!--@while var:value--> … <!--@end_while-->`	Checks `MSH_VAR_var` file before each iteration. Init the variable with a `c >var` block immediately before the tag. Body writes exit value to the file via `fopen`.
LOOP	`<!--@loop max=N until=var:value--> … <!--@end_loop-->`	Checks **last stdout line** of last block in body. Print the exit value as last line to exit.
FOREACH	`<!--@foreach item in list--> … <!--@end_foreach-->`	`list` must be created by a `c >list` block before the tag. One item per line.
TRY/CATCH	`<!--@try--> … <!--@catch>err--> … <!--@end_try-->`	CATCH block: never read `<errvar`. Print literal message only.
SPLIT	`<!--@split var into N-->`	Creates `var_1` … `var_N` from lines.
MERGE	`<!--@merge-->`	Reduce marker after split.
CONFIG	` ```config ``` `	Documentation block, parsed by mshell.
Async	`<!--@1 <in >out async-->`	Launch LLM in background.
Await	` ```c await=var1,var2 ``` `	Barrier — wait for async jobs.

Key Rules (learned from execution logs)

1. **Fan-out reads** — when multiple C blocks consume the same variable, the producer must also write to a stable `/tmp/` path. Consumers read from that path directly.
2. **Code-gen compile** — always `cp` the LLM variable to a stable `/tmp/` path before calling `gcc`. The mshell context may delete the original temp file.
3. **WHILE exit** — body block writes `done/running` to `MSH_VAR_status` via `fopen`. mshell reads the file between iterations.
4. **LOOP exit** — last block in body must print the exit value as its **last stdout line**. mshell checks stdout, not the file.
5. **All C blocks** — always include all required headers. The mshell validator strips them during validation so include them explicitly in every block.

Part I — Patterns 1–12: Core Patterns

Pattern 1 — Linear Data Pipeline

```c
#include <stdio.h>
```

```c
int is_prime(int n) {
    if (n < 2) return 0;
    for (int i = 2; i * i <= n; i++)
        if (n % i == 0) return 0;
    return 1;
}

int main(void) {
    int count = 0;
    for (int n = 2; count < 10; n++) {
        if (is_prime(n)) {
            printf("%d", n);
            if (++count < 10) printf(" ");
        }
    }
    printf("\n");
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    FILE *f = fopen(getenv("MSH_VAR_primes"), "r");
    int nums[32], n = 0, sum = 0, mx = -1, mn = 1<<30;
    while (fscanf(f, "%d", &nums[n]) == 1) {
        sum += nums[n];
        if (nums[n] > mx) mx = nums[n];
        if (nums[n] < mn) mn = nums[n];
        n++;
    }
    fclose(f);
    printf("count=%d sum=%d mean=%.2f min=%d max=%d\n",
           n, sum, (double)sum/n, mn, mx);
    return 0;
}
```

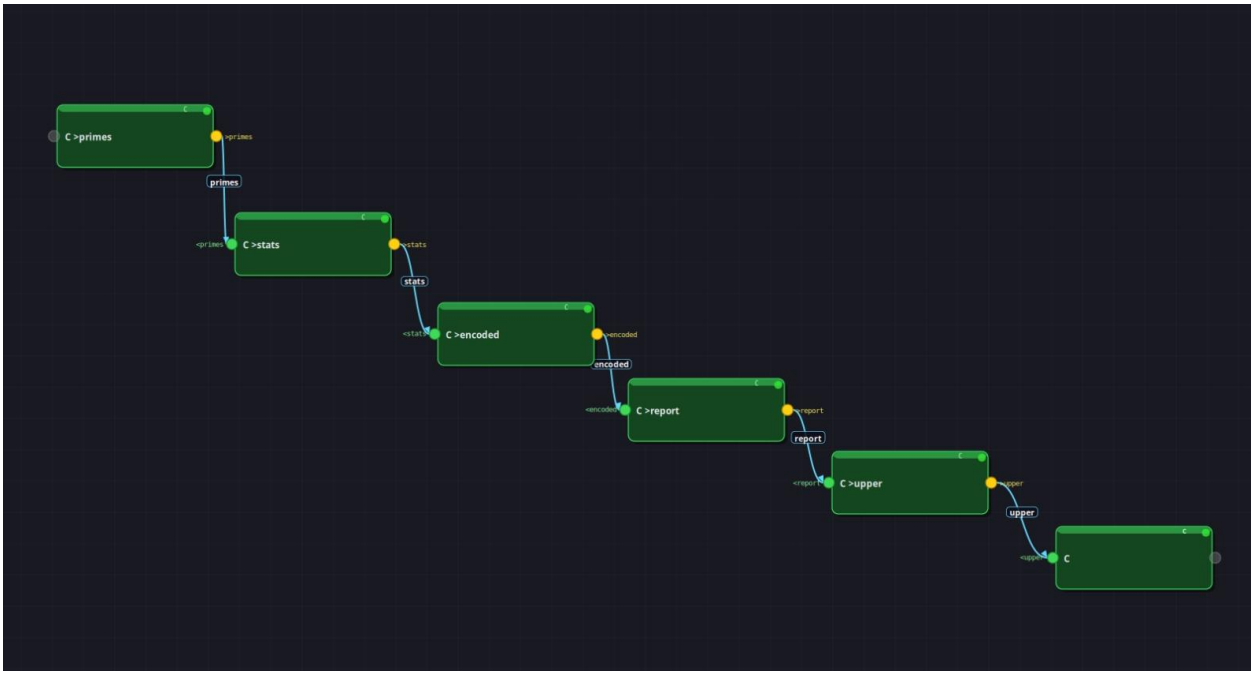
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void) {
    FILE *f = fopen(getenv("MSH_VAR_stats"), "r");
    char buf[256] = {0};
    fgets(buf, sizeof(buf), f); fclose(f);
    int len = strlen(buf);
    printf("HEX:");
    for (int i = 0; i < len && buf[i] != '\n'; i++)
        printf("%02X", (unsigned char)buf[i]);
    printf("\nRAW:%s", buf);
```

```
    return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void) {
    FILE *f = fopen(getenv("MSH_VAR_encoded"), "r");
    char line[512];
    printf("%-20s %-60s\n", "FIELD", "VALUE");
    printf("%-20s %-60s\n", "--------------------", "----");
    while (fgets(line, sizeof(line), f)) {
        char *colon = strchr(line, ':');
        if (colon) { *colon = '\0'; printf("%-20s %-60s\n", line, colon+1); }
    }
    fclose(f);
    return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

int main(void) {
    FILE *f = fopen(getenv("MSH_VAR_report"), "r"); int c;
    while ((c = fgetc(f)) != EOF) putchar(toupper(c));
    fclose(f);
    return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void) {
    FILE *f = fopen(getenv("MSH_VAR_upper"), "r");
    char lines[32][256]; int n = 0;
    while (n < 32 && fgets(lines[n], 256, f)) n++;
    fclose(f);
    int width = 70;
    for (int i = 0; i < width; i++) putchar('='); putchar('\n');
    for (int i = 0; i < n; i++) printf("| %-*s|\n", width-3, lines[i]);
    for (int i = 0; i < width; i++) putchar('='); putchar('\n');
    return 0;
}
```

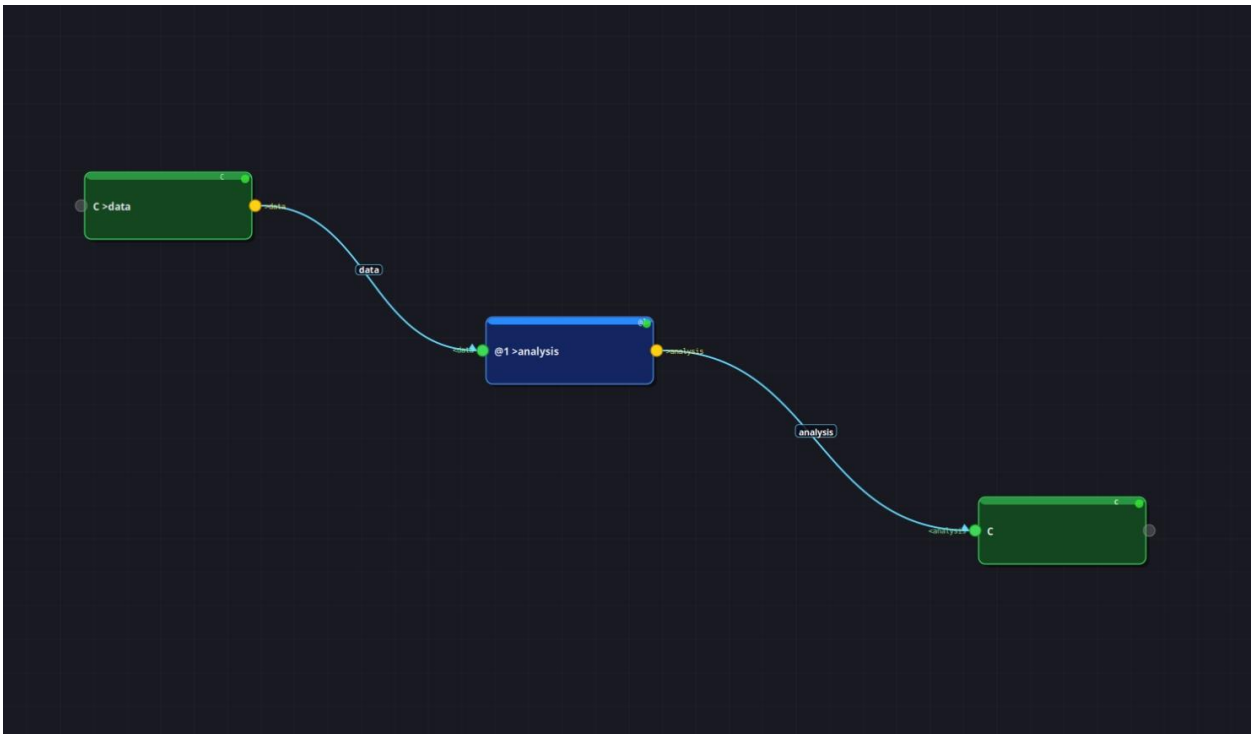


Pattern 2 — LLM in the Middle

```c
#include <stdio.h>

int main(void) {
    const char *algos[] = {"BubbleSort","MergeSort","QuickSort","HeapSort","R
adixSort","TimSort"};
    double ms[] = {812.4, 45.2, 38.7, 51.3, 29.1, 41.8};
    printf("Algorithm Benchmark (ms per 1M elements):\n");
    for (int i = 0; i < 6; i++)
        printf("  %-12s : %.1f ms\n", algos[i], ms[i]);
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

int main(void) {
    FILE *f = fopen(getenv("MSH_VAR_analysis"), "r");
    char buf[4096] = {0};
    fread(buf, 1, sizeof(buf)-1, f); fclose(f);
    int words = 0, sentences = 0, chars = 0, in_word = 0;
    for (int i = 0; buf[i]; i++) {
        chars++;
        if (isspace(buf[i])) in_word = 0;
        else if (!in_word) { words++; in_word = 1; }
        if (buf[i]=='.'||buf[i]=='!'||buf[i]=='?') sentences++;
    }
    printf("=== Metadata: words=%d sentences=%d chars=%d ===\n\n", words, sen
tences, chars);
```

```
    printf("=== Analysis ===\n%s\n", buf);
    return 0;
}
```



Pattern 3 — Fan-Out (One Variable → Many Consumers)

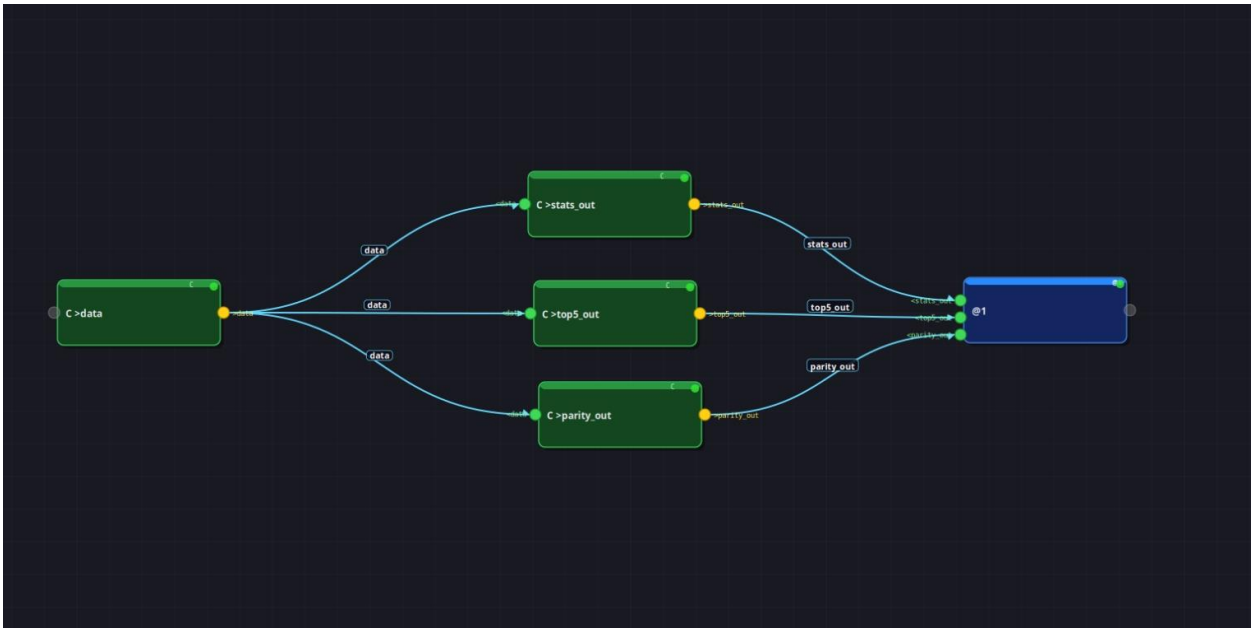
```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    srand(12345);
    FILE *tmp = fopen("/tmp/msh_p3_data.txt", "w");
    for (int i = 0; i < 100; i++) {
        int v = rand() % 200 - 100;
        printf("%d", v);
        fprintf(tmp, "%d", v);
        if (i < 99) { printf(" "); fprintf(tmp, " "); }
    }
    printf("\n"); fprintf(tmp, "\n");
    fclose(tmp);
    return 0;
}
```

```c
#include <stdio.h>
#include <math.h>

int main(void) {
    FILE *f = fopen("/tmp/msh_p3_data.txt", "r");
    double v[256]; int n = 0;
```

```
    while (n < 256 && fscanf(f, "%lf", &v[n]) == 1) n++;
    fclose(f);
    double sum = 0;
    for (int i = 0; i < n; i++) sum += v[i];
    double mean = sum/n, var = 0;
    for (int i = 0; i < n; i++) var += (v[i]-mean)*(v[i]-mean);
    var /= n;
    printf("n=%d mean=%.2f variance=%.2f stddev=%.2f\n", n, mean, var, sqrt(v
ar));
    return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>

int cmp_desc(const void *a, const void *b) { return *(int*)b - *(int*)a; }

int main(void) {
    FILE *f = fopen("/tmp/msh_p3_data.txt", "r");
    int v[256], n = 0;
    while (n < 256 && fscanf(f, "%d", &v[n]) == 1) n++;
    fclose(f);
    qsort(v, n, sizeof(int), cmp_desc);
    printf("Top-5:");
    for (int i = 0; i < 5 && i < n; i++) printf(" %d", v[i]);
    printf("\n");
    return 0;
}
```

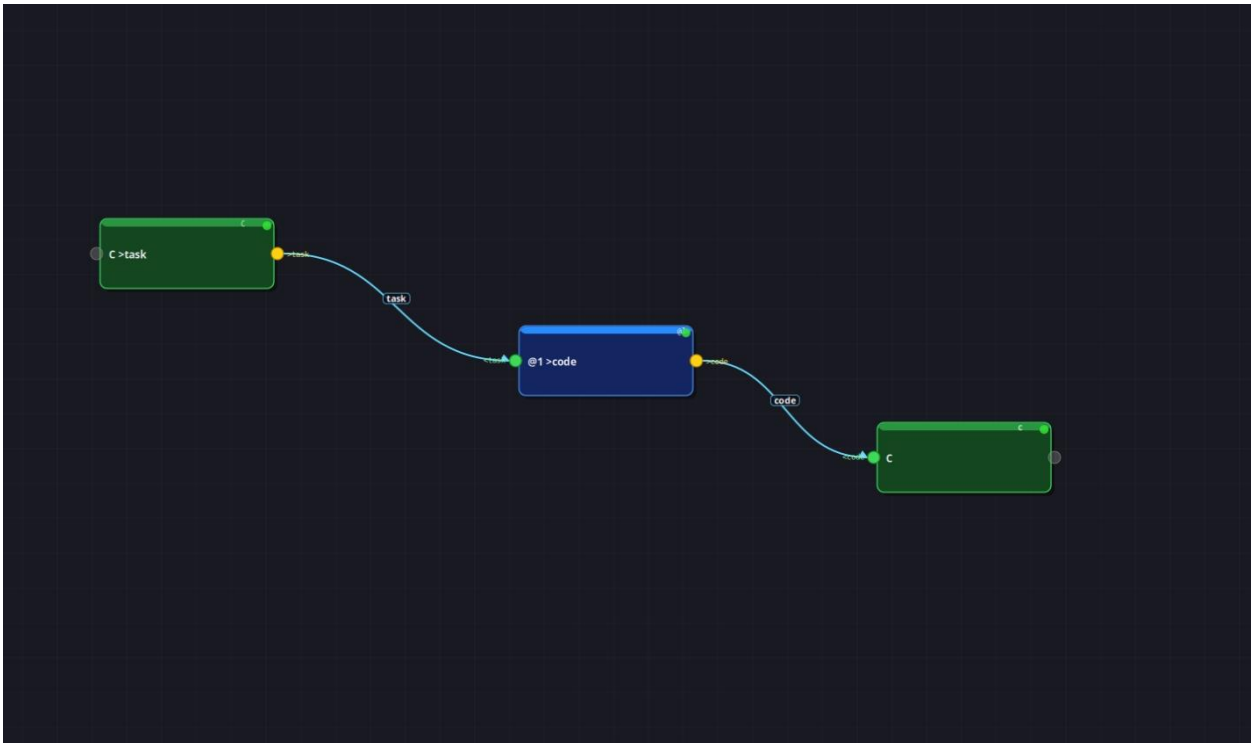
```
#include <stdio.h>

int main(void) {
    FILE *f = fopen("/tmp/msh_p3_data.txt", "r");
    int v, evens = 0, odds = 0, runs = 1, prev_sign = 0;
    while (fscanf(f, "%d", &v) == 1) {
        if (v % 2 == 0) evens++; else odds++;
        int s = (v >= 0) ? 1 : -1;
        if (s != prev_sign && prev_sign != 0) runs++;
        prev_sign = s;
    }
    fclose(f);
    printf("even=%d odd=%d sign_runs=%d\n", evens, odds, runs);
    return 0;
}
```



Pattern 4 — LLM Code Generation → Compile & Execute

```
#include <stdio.h>

int main(void) {
    printf("Write a C function matrix_mul(int A[2][2], int B[2][2], int C[2][
2]) "
           "that multiplies two 2x2 integer matrices and stores the result in
C. "
           "Include a main() with A={{1,2},{3,4}} B={{5,6},{7,8}} and prints
the result.\n");
    return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    const char *src = getenv("MSH_VAR_code");
    char cmd[640];
    snprintf(cmd, sizeof(cmd), "cp -- \"%s\" /tmp/msh_p4_gen.c", src);
    system(cmd);
    printf("=== Generated C code ===\n");
    system("cat /tmp/msh_p4_gen.c");
    printf("\n=== Compiling & Running ===\n");
    if (system("gcc -Wall -std=c11 -o /tmp/msh_p4_out /tmp/msh_p4_gen.c -lm 2
>&1") == 0)
        system("/tmp/msh_p4_out");
    else
        printf("Compilation failed.\n");
    return 0;
}
```



Pattern 5 — Two-LLM Review Chain

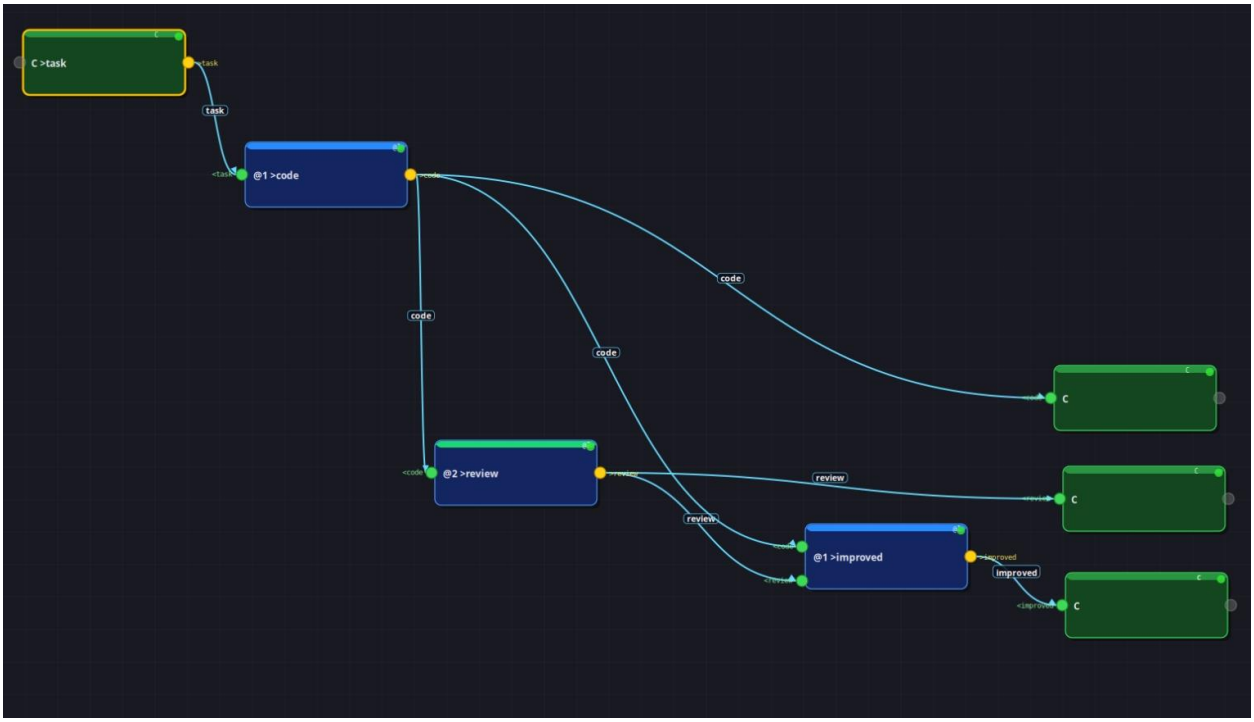
```c
#include <stdio.h>

int main(void) {
    printf("Implement a binary search tree in C with:\n"
           "1. Node struct (int value, left, right pointers).\n"
           "2. bst_insert(Node **root, int val).\n"
           "3. inorder(Node *root) prints values.\n"
           "4. main() inserts 5,3,7,1,4,6,8 and prints inorder.\n"
           "5. Free all nodes at the end.\n");
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    printf("=== Generated Code ===\n");
    FILE *f = fopen(getenv("MSH_VAR_code"), "r"); int c;
    while ((c = fgetc(f)) != EOF) putchar(c);
    fclose(f); printf("\n");
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
```

```
    printf("=== Review ===\n");
    FILE *f = fopen(getenv("MSH_VAR_review"), "r"); int c;
    while ((c = fgetc(f)) != EOF) putchar(c);
    fclose(f); printf("\n");
    return 0;
}
```

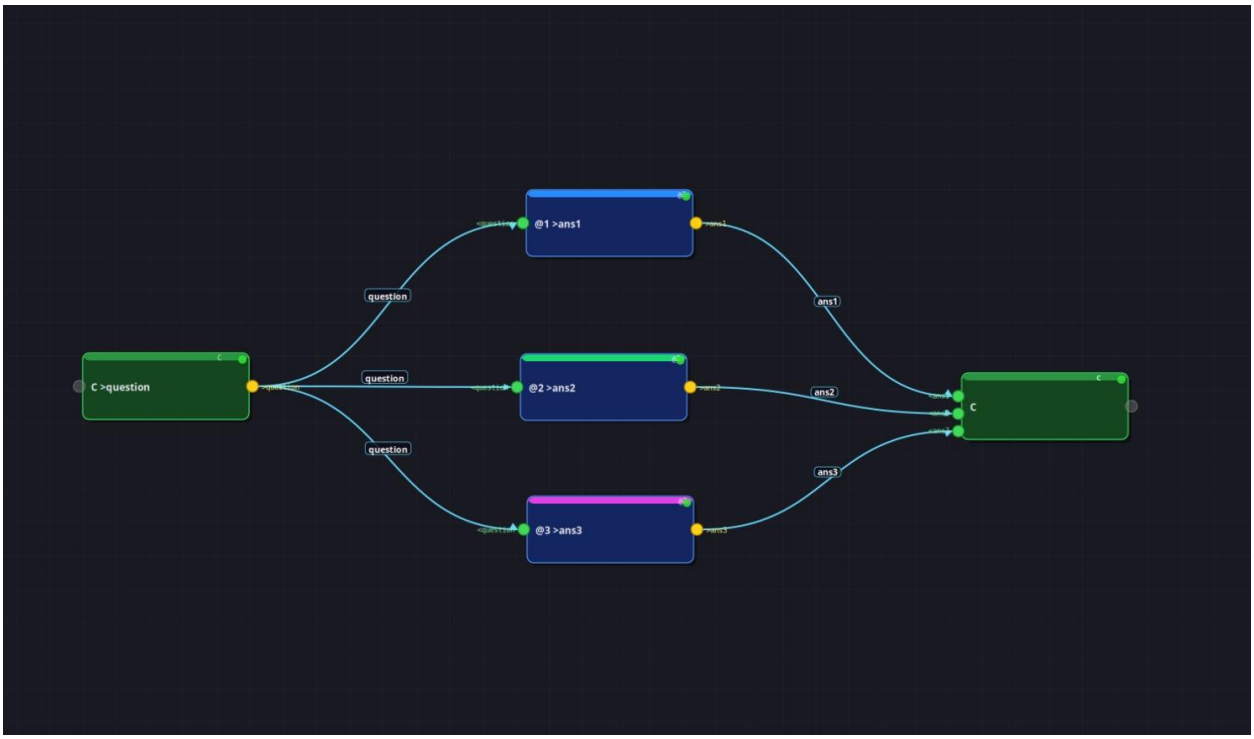
```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    const char *src = getenv("MSH_VAR_improved");
    char cmd[640];
    snprintf(cmd, sizeof(cmd), "cp -- \"%s\" /tmp/msh_p5_bst.c", src);
    system(cmd);
    printf("=== Improved Code ===\n");
    system("cat /tmp/msh_p5_bst.c");
    printf("\n=== Compiling & Running ===\n");
    if (system("gcc -Wall -std=c11 -o /tmp/msh_p5_out /tmp/msh_p5_bst.c 2>&1"
) == 0)
        system("/tmp/msh_p5_out");
    else
        printf("Compilation failed.\n");
    return 0;
}
```



Pattern 6 — Parallel 3-Model Query

```c
#include <stdio.h>

int main(void) {
    printf("In C, explain stack vs heap: lifetime, size limits, fragmentation
, "
           "and when to prefer each. One paragraph, maximum 60 words.\n");
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

int wc(const char *path) {
    FILE *f = fopen(path, "r"); int c, w = 0, iw = 0;
    while ((c = fgetc(f)) != EOF) {
        if (isspace(c)) iw = 0;
        else if (!iw) { w++; iw = 1; }
    }
    fclose(f); return w;
}

void pa(const char *label, const char *path) {
    printf("=== %s (%d words) ===\n", label, wc(path));
    FILE *f = fopen(path, "r"); int c;
    while ((c = fgetc(f)) != EOF) putchar(c);
    fclose(f); printf("\n\n");
}

int main(void) {
    pa("Model @1", getenv("MSH_VAR_ans1"));
    pa("Model @2", getenv("MSH_VAR_ans2"));
    pa("Model @3", getenv("MSH_VAR_ans3"));
    return 0;
}
```



Pattern 7 — Evaluator-Optimizer Loop

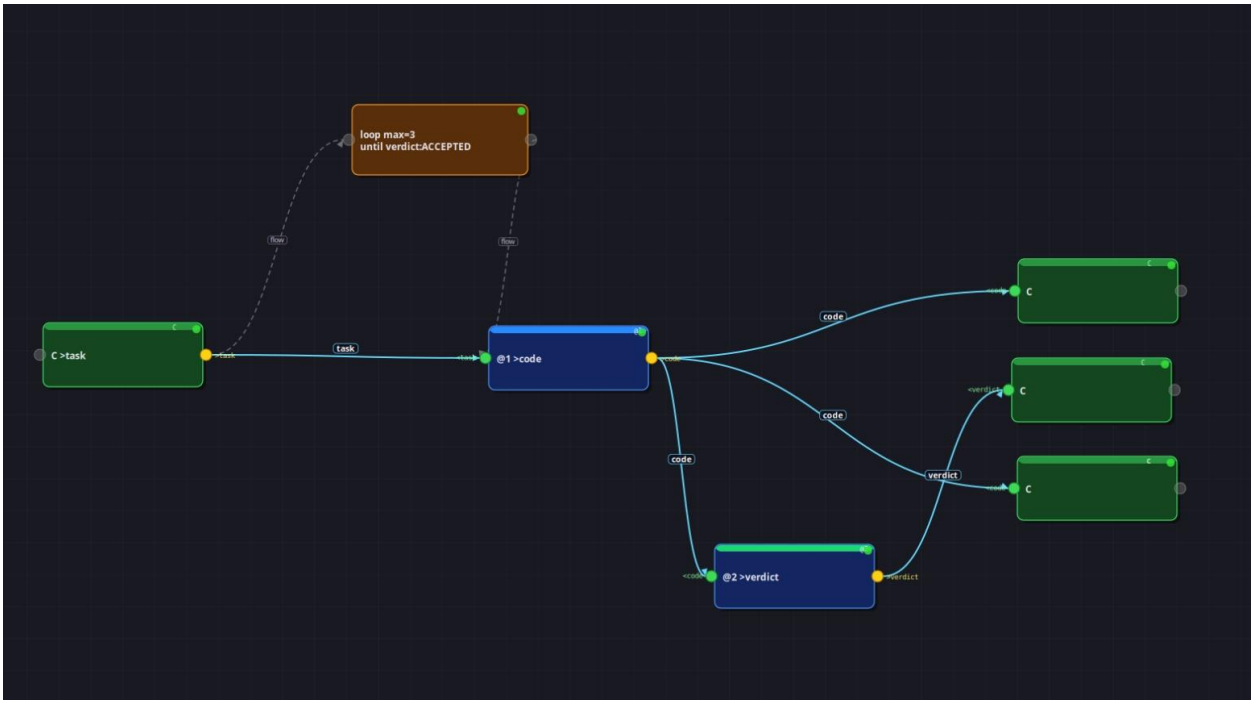
```c
#include <stdio.h>

int main(void) {
    printf("Write int gcd(int a, int b) using the Euclidean algorithm. "
           "main() tests gcd(48,18)=6, gcd(100,75)=25, gcd(7,13)=1, gcd(0,5)=
5 "
           "and prints PASS or FAIL for each.\n");
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    printf("=== Generated Code ===\n");
    FILE *f = fopen(getenv("MSH_VAR_code"), "r"); int c;
    while ((c = fgetc(f)) != EOF) putchar(c);
    fclose(f); printf("\n");
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    FILE *f = fopen(getenv("MSH_VAR_verdict"), "r");
    char buf[64] = {0};
```

```
    fgets(buf, sizeof(buf), f); fclose(f);
    /* Strip newline */
    int len = 0; while (buf[len] && buf[len] != '\n' && buf[len] != '\r') len
++;
    buf[len] = '\0';
    printf("=== Verdict: %s ===\n", buf);
    /* Last stdout line must be the verdict word for LOOP condition */
    printf("%s\n", buf);
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    const char *src = getenv("MSH_VAR_code");
    char cmd[640];
    snprintf(cmd, sizeof(cmd), "cp -- \"%s\" /tmp/msh_p7_gcd.c", src);
    system(cmd);
    printf("=== Final Accepted Code ===\n");
    system("cat /tmp/msh_p7_gcd.c");
    printf("\n=== Compiling & Running ===\n");
    if (system("gcc -Wall -std=c11 -o /tmp/msh_p7_out /tmp/msh_p7_gcd.c 2>&1"
) == 0)
        system("/tmp/msh_p7_out");
    else
        printf("Compilation failed.\n");
    return 0;
}
```



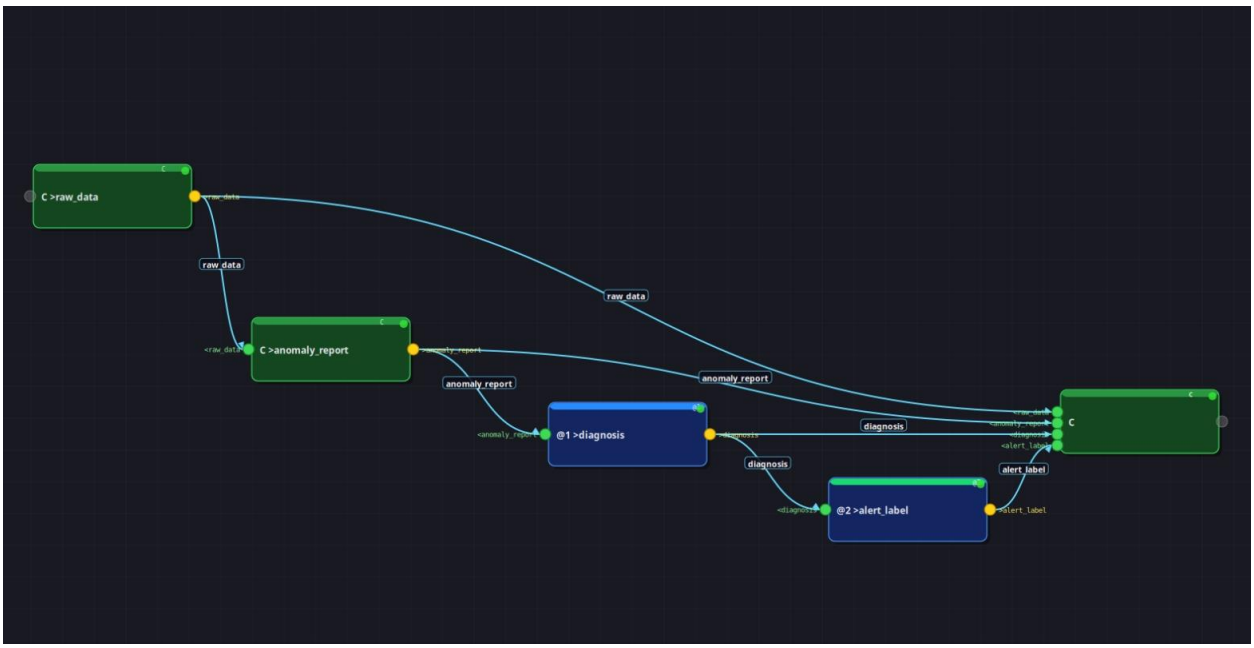
Pattern 8 — Multi-Stage C + Multi-Model Pipeline

```c
#include <stdio.h>

int main(void) {
    double temps[] = {21.8,22.1,21.9,22.3,22.0,21.7,22.2,38.5,39.1,22.4,
                      22.1,21.8,22.0,22.3,21.9,22.1,22.4,21.6,22.2,22.0};
    FILE *tmp = fopen("/tmp/msh_p8_raw.txt", "w");
    for (int i = 0; i < 20; i++) {
        printf("t=%02d temp=%.1f\n", i, temps[i]);
        fprintf(tmp, "t=%02d temp=%.1f\n", i, temps[i]);
    }
    fclose(tmp);
    return 0;
}
```

```c
#include <stdio.h>
#include <math.h>

int main(void) {
    FILE *f = fopen("/tmp/msh_p8_raw.txt", "r");
    double t[32]; int n = 0, idx;
    while (n < 32 && fscanf(f, "t=%d temp=%lf\n", &idx, &t[n]) == 2) n++;
    fclose(f);
    double sum = 0;
    for (int i = 0; i < n; i++) sum += t[i];
    double mean = sum/n, var = 0;
    for (int i = 0; i < n; i++) var += (t[i]-mean)*(t[i]-mean);
    double sigma = sqrt(var/n);
    printf("mean=%.2f sigma=%.2f threshold=%.2f\nANOMALIES:\n",
           mean, sigma, mean+2*sigma);
    for (int i = 0; i < n; i++)
        if (fabs(t[i]-mean) > 2*sigma)
            printf("  t=%02d temp=%.1f dev=+%.1f\n", i, t[i], t[i]-mean);
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

void ps(const char *title, const char *key) {
    printf("\n=== %s ===\n", title);
    FILE *f = fopen(getenv(key), "r"); int c;
    while ((c = fgetc(f)) != EOF) putchar(c);
    fclose(f);
}

void pf(const char *title, const char *path) {
    printf("\n=== %s ===\n", title);
    FILE *f = fopen(path, "r"); int c;
    while ((c = fgetc(f)) != EOF) putchar(c);
```

```
    fclose(f);
}

int main(void) {
    printf("╔══════════════════════════════╗\n"
           "║   SENSOR PIPELINE REPORT     ║\n"
           "╚══════════════════════════════╝");
    pf("RAW DATA",       "/tmp/msh_p8_raw.txt");
    ps("ANOMALY REPORT", "MSH_VAR_anomaly_report");
    ps("DIAGNOSIS",      "MSH_VAR_diagnosis");
    ps("ALERT LABEL",    "MSH_VAR_alert_label");
    printf("\n");
    return 0;
}
```



Pattern 9 — Routing

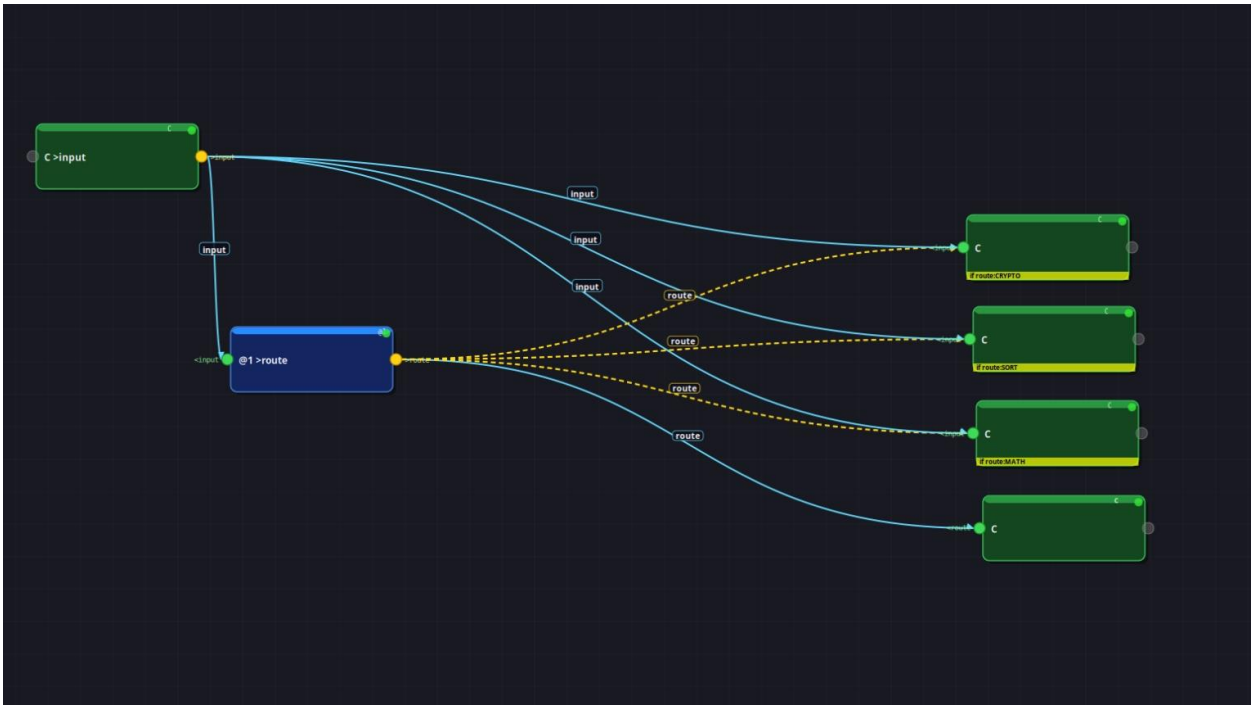
```
#include <stdio.h>

int main(void) {
    printf("Compute the square root of 144 using an iterative numerical metho
d.\n");
    return 0;
}

#include <stdio.h>

int main(void) {
    printf("=== CRYPTO: XOR cipher ===\n");
    const char *msg = "Hello, mshell!"; unsigned char key = 0xAB;
    printf("Original : %s\nEncrypted: ", msg);
    for (int i = 0; msg[i]; i++) printf("%02X ", (unsigned char)(msg[i]^key))
```

```
;
    printf("\nDecrypted: ");
    for (int i = 0; msg[i]; i++) printf("%c", (unsigned char)(msg[i]^key)^key
);
    printf("\n");
    return 0;
}
```

```
#include <stdio.h>

void merge(int *a, int l, int m, int r) {
    int n1=m-l+1, n2=r-m, L[64], R[64];
    for (int i=0; i<n1; i++) L[i]=a[l+i];
    for (int j=0; j<n2; j++) R[j]=a[m+1+j];
    int i=0, j=0, k=l;
    while (i<n1&&j<n2) a[k++]=(L[i]<=R[j])?L[i++]:R[j++];
    while (i<n1) a[k++]=L[i++];
    while (j<n2) a[k++]=R[j++];
}
void mergesort(int *a, int l, int r) {
    if (l<r) { int m=(l+r)/2; mergesort(a,l,m); mergesort(a,m+1,r); merge(a,l
,m,r); }
}

int main(void) {
    printf("=== SORT: merge sort ===\n");
    int arr[] = {64,34,25,12,22,11,90}; int n = 7;
    printf("Before: "); for (int i=0; i<n; i++) printf("%d ", arr[i]);
    mergesort(arr, 0, n-1);
    printf("\nAfter : "); for (int i=0; i<n; i++) printf("%d ", arr[i]); prin
tf("\n");
    return 0;
}
```

```
#include <stdio.h>
#include <math.h>

int main(void) {
    printf("=== MATH: Newton-Raphson sqrt(144) ===\n");
    double x = 144.0, g = x/2.0;
    for (int i = 0; i < 10; i++) {
        double next = 0.5*(g+x/g);
        printf("  iter %2d: %.10f (err=%.2e)\n", i+1, next, fabs(next-sqrt(x)
));
        if (fabs(next-g) < 1e-10) { g=next; break; }
        g = next;
    }
    printf("Result: %.6f\n", g);
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    printf("=== Classified as: ");
    FILE *f = fopen(getenv("MSH_VAR_route"), "r"); int c;
    while ((c = fgetc(f)) != EOF && c != '\n') putchar(c);
    fclose(f); printf(" ===\n");
    return 0;
}
```



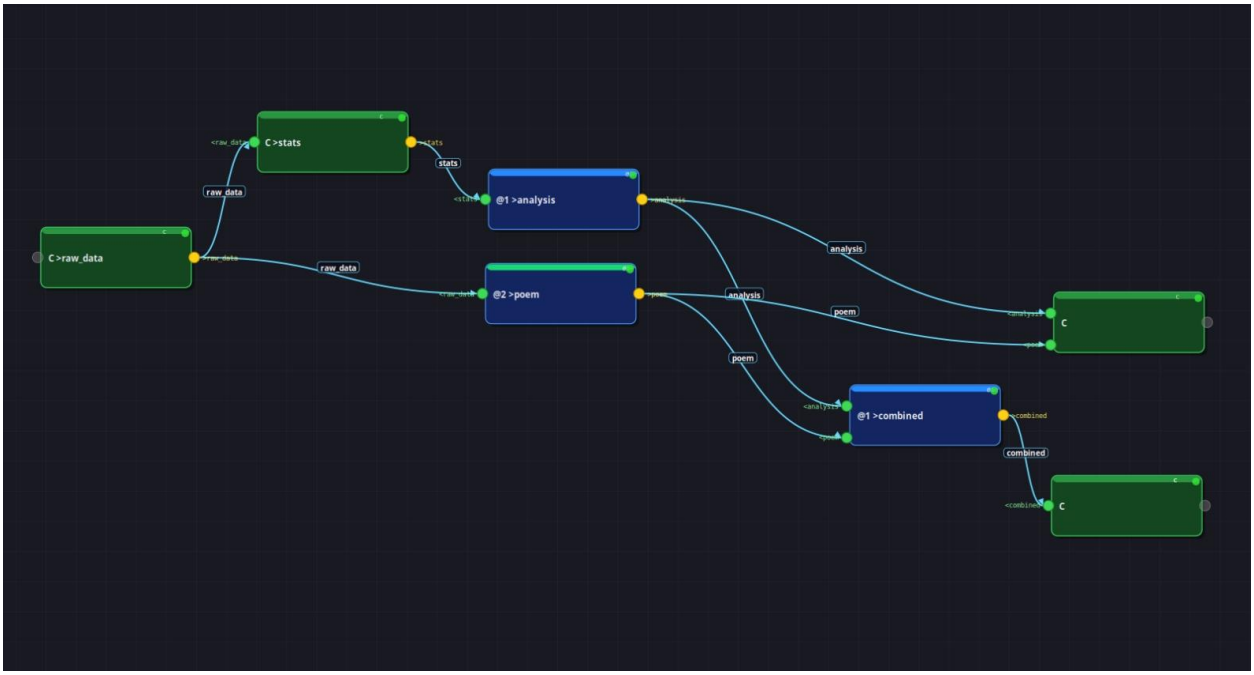
Pattern 10 — Full Pipeline

```c
#include <stdio.h>
#include <string.h>

int main(void) {
    int s[201]; memset(s, 1, sizeof(s)); s[0]=s[1]=0;
    for (int i=2; i<=200; i++)
        if (s[i]) for (int j=2*i; j<=200; j+=i) s[j]=0;
    FILE *tmp = fopen("/tmp/msh_p10_primes.txt", "w");
    int first = 1;
    for (int i=2; i<=200; i++) if (s[i]) {
        if (!first) { printf(" "); fprintf(tmp, " "); }
        printf("%d", i); fprintf(tmp, "%d", i); first = 0;
    }
    printf("\n"); fprintf(tmp, "\n");
    fclose(tmp);
    return 0;
}
```

```c
#include <stdio.h>

int main(void) {
    FILE *f = fopen("/tmp/msh_p10_primes.txt", "r");
    int p[64], n = 0;
    while (n < 64 && fscanf(f, "%d", &p[n]) == 1) n++;
    fclose(f);
    int twins = 0, mx_gap = 0;
    for (int i=1; i<n; i++) {
        int g = p[i]-p[i-1];
        if (g > mx_gap) mx_gap = g;
        if (g == 2) twins++;
    }
    printf("count=%d max_gap=%d twin_pairs=%d goldbach_4_to_20=verified\n",
           n, mx_gap, twins);
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    printf("=== Analysis ===\n");
    FILE *f = fopen(getenv("MSH_VAR_analysis"), "r"); int c;
    while ((c = fgetc(f)) != EOF) putchar(c); fclose(f);
    printf("\n=== Poem ===\n");
    f = fopen(getenv("MSH_VAR_poem"), "r");
    while ((c = fgetc(f)) != EOF) putchar(c); fclose(f); printf("\n");
    return 0;
}
```

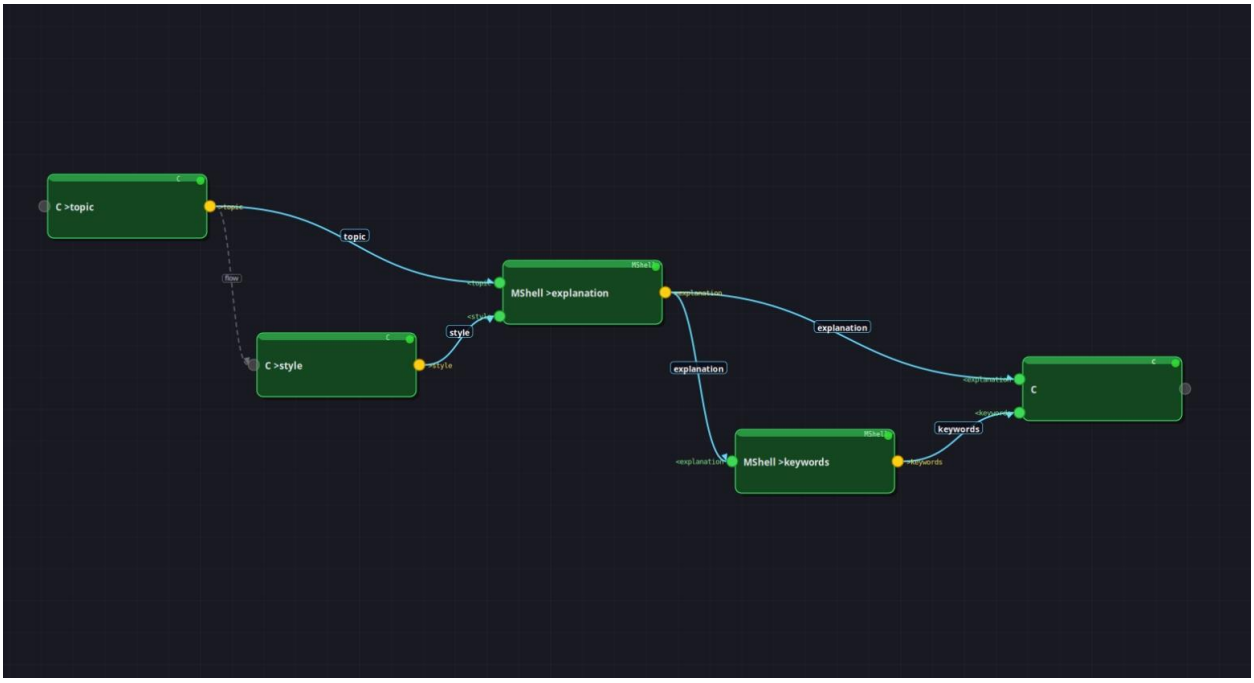
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void) {
    FILE *f = fopen(getenv("MSH_VAR_combined"), "r");
    char buf[1024] = {0}; fread(buf, 1, sizeof(buf)-1, f); fclose(f);
    int len = strlen(buf);
    if (len > 0 && buf[len-1] == '\n') buf[--len] = '\0';
    printf("\n");
    for (int i = 0; i < len+4; i++) putchar('*');
    printf("\n* %s *\n", buf);
    for (int i = 0; i < len+4; i++) putchar('*');
    printf("\n\n[mshell C Pipeline — Primes <= 200]\n");
    return 0;
}
```



Pattern 11 — MShell Node with Multiple Models

```
#include <stdio.h>

int main(void) { printf("lock-free queue implementation in C using atomic ope
rations\n"); return 0; }

#include <stdio.h>

int main(void) { printf("expert systems programmer\n"); return 0; }

ollama1 "Explain $topic to a $style in one concise paragraph."

ollama2 "Extract 5 technical keywords from: $explanation. Reply with exactly
5 words comma-separated."

#include <stdio.h>
#include <stdlib.h>

int flen(const char *p) {
    FILE *f = fopen(p, "r"); fseek(f, 0, SEEK_END);
    int l = ftell(f); fclose(f); return l;
}
void pf(const char *p) {
    FILE *f = fopen(p, "r"); int c;
    while ((c = fgetc(f)) != EOF) putchar(c); fclose(f);
}

int main(void) {
    char *ep = getenv("MSH_VAR_explanation"), *kp = getenv("MSH_VAR_keywords"
);
```

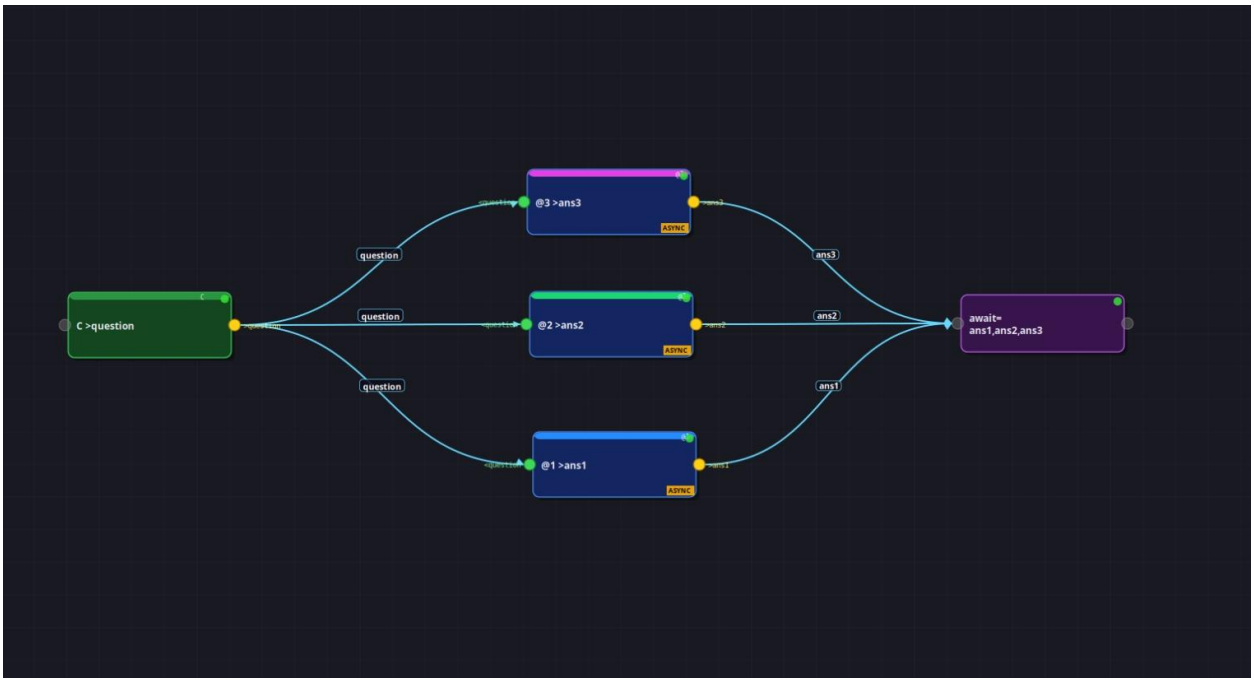
```
    printf("=== Explanation (%d bytes) ===\n", flen(ep)); pf(ep);
    printf("\n=== Keywords (%d bytes) ===\n", flen(kp)); pf(kp); printf("\n")
;
    return 0;
}
```



Pattern 12 — Async Parallel 3 Models + Await + Synthesis

```c
#include <stdio.h>

int main(void) {
    printf("In C, explain struct memory alignment and padding. "
           "Why does sizeof(struct{char a; int b;}) != 5 on most platforms? O
ne paragraph.\n");
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    printf("\n=== C STRUCT ALIGNMENT — TECH NOTE ===\n");
    FILE *f = fopen(getenv("MSH_VAR_final"), "r"); int c;
    while ((c = fgetc(f)) != EOF) putchar(c); fclose(f);
    printf("\n[Generated by mshell async 3-model pipeline]\n");
    return 0;
}
```



Part II — Patterns 13–24: Advanced Node Types

Pattern 13 — WHILE Loop: Fibonacci with LLM Commentary

```c
#include <stdio.h>

int main(void) { printf("0\n"); return 0; }
```

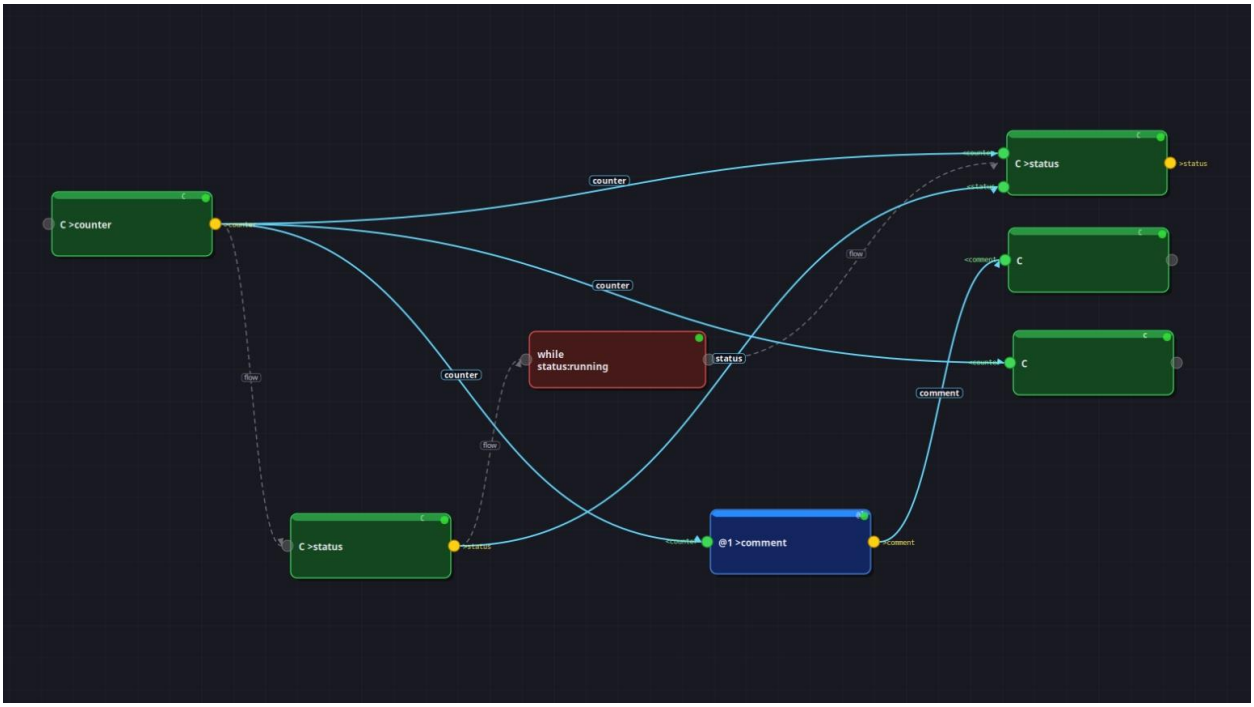
```c
#include <stdio.h>

int main(void) { printf("running\n"); return 0; }
```

```c
#include <stdio.h>
#include <stdlib.h>

void fib_fast(long long n, long long *fa, long long *fb) {
    if (n == 0) { *fa=0; *fb=1; return; }
    long long a, b; fib_fast(n/2, &a, &b);
    long long c = a*(2*b-a), d = a*a+b*b;
    if (n%2 == 0) { *fa=c; *fb=d; } else { *fa=d; *fb=c+d; }
}

int main(void) {
    FILE *f = fopen(getenv("MSH_VAR_counter"), "r");
    long long val; fscanf(f, "%lld", &val); fclose(f);
    val++;
    long long fa, fb; fib_fast(val, &fa, &fb);
```

```
    f = fopen(getenv("MSH_VAR_counter"), "w");
    fprintf(f, "%lld\n", val); fclose(f);
    f = fopen(getenv("MSH_VAR_status"), "w");
    fprintf(f, "%s\n", val >= 6 ? "done" : "running"); fclose(f);
    printf("counter=%lld fib(%lld)=%lld\n", val, val, fa);
    return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    long long n;
    FILE *f = fopen(getenv("MSH_VAR_counter"), "r"); fscanf(f, "%lld", &n); f
close(f);
    printf("[iter %lld] ", n);
    f = fopen(getenv("MSH_VAR_comment"), "r"); int c;
    while ((c = fgetc(f)) != EOF) putchar(c); fclose(f); printf("\n");
    return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>

void fib_fast(long long n, long long *fa, long long *fb) {
    if (n == 0) { *fa=0; *fb=1; return; }
    long long a, b; fib_fast(n/2, &a, &b);
    long long c = a*(2*b-a), d = a*a+b*b;
    if (n%2 == 0) { *fa=c; *fb=d; } else { *fa=d; *fb=c+d; }
}

int main(void) {
    FILE *f = fopen(getenv("MSH_VAR_counter"), "r");
    long long total; fscanf(f, "%lld", &total); fclose(f);
    printf("=== WHILE done. F(0) through F(%lld):\n", total);
    for (long long i = 0; i <= total; i++) {
        long long fa, fb; fib_fast(i, &fa, &fb);
        printf("  F(%lld)=%lld\n", i, fa);
    }
    return 0;
}
```

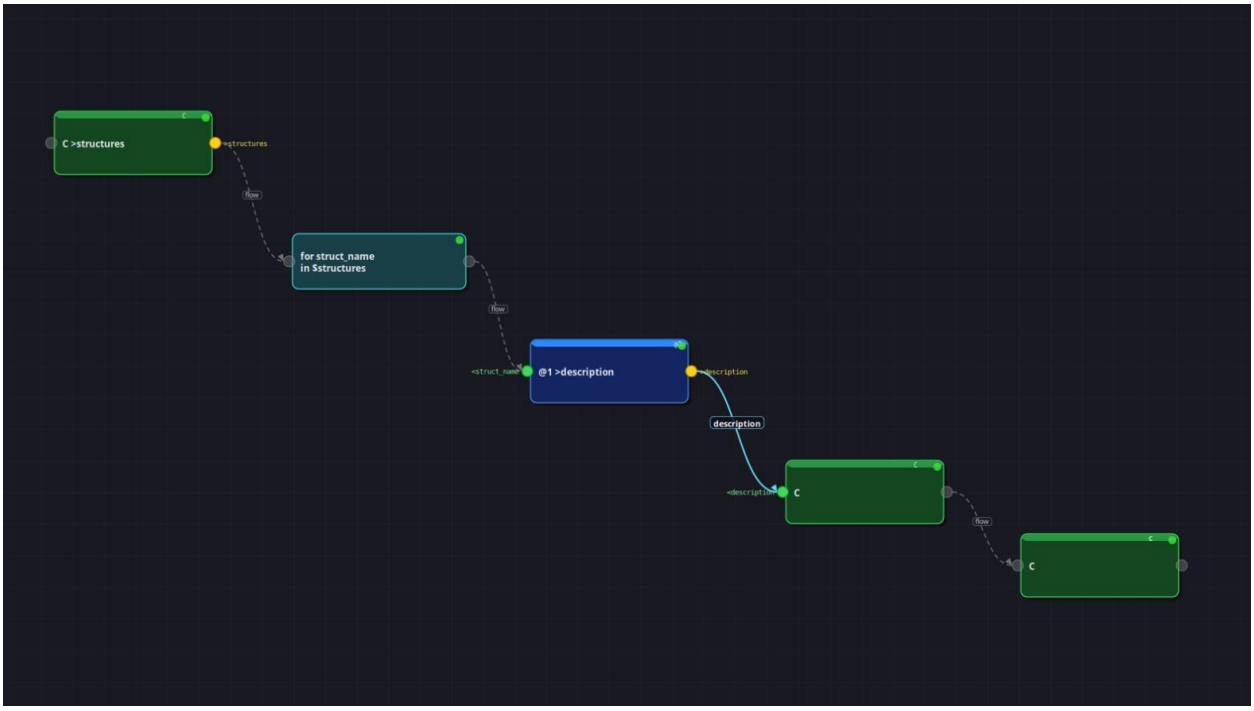


Pattern 14 — FOREACH: LLM Explains Each C Data Structure

```c
#include <stdio.h>

int main(void) {
    printf("linked_list\nhash_table\nring_buffer\nred_black_tree\nskip_list\n
");
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    printf("--- ");
    FILE *f = fopen(getenv("MSH_VAR_struct_name"), "r"); int c;
    while ((c = fgetc(f)) != EOF && c != '\n') putchar(c); fclose(f); printf(
" ---\n");
    f = fopen(getenv("MSH_VAR_description"), "r");
    while ((c = fgetc(f)) != EOF) putchar(c); fclose(f); printf("\n");
    return 0;
}
```

```c
#include <stdio.h>

int main(void) { printf("=== All C data structures described ===\n"); return
0; }
```



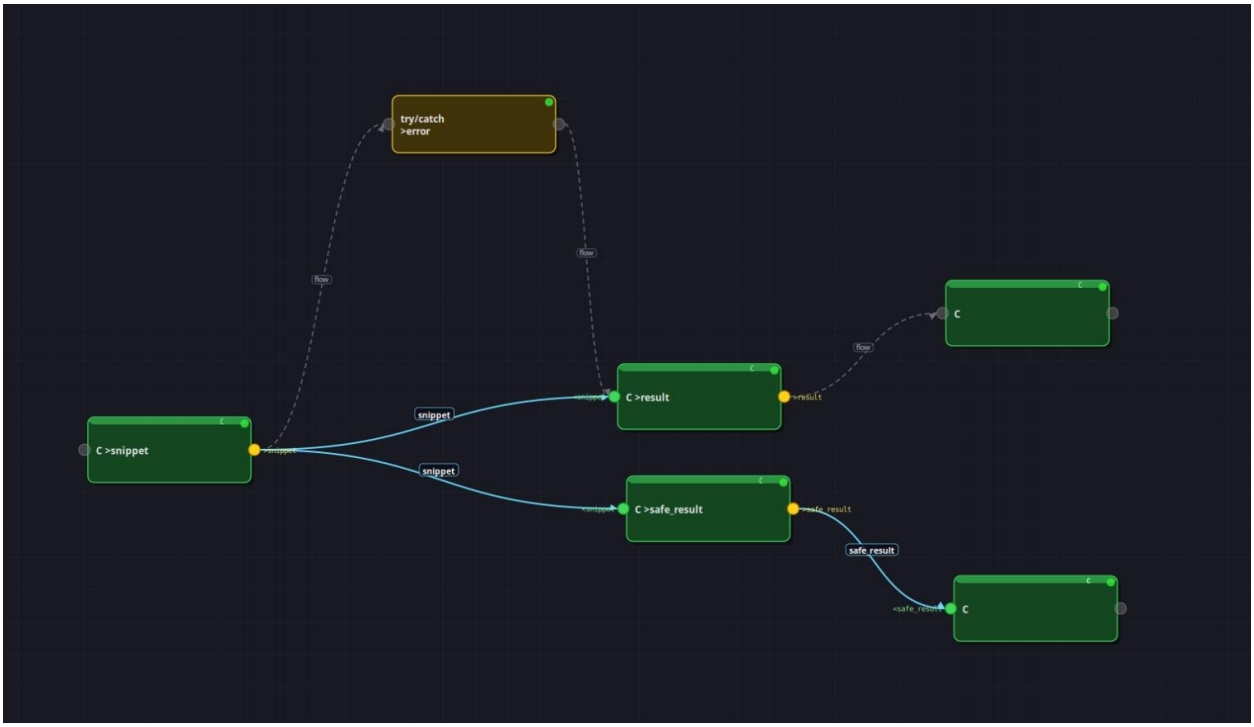
Pattern 15 — TRY/CATCH: Safe C Compilation with Error Capture

```
#include <stdio.h>

int main(void) {
    printf("int add(int a, int b) { return a + b }  /* missing semicolon */\n
"
           "int main(void) { printf(\"%%d\\n\", add(3,4)); return 0; }\n");
    return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    FILE *src = fopen("/tmp/msh_p15_try.c", "w");
    FILE *in  = fopen(getenv("MSH_VAR_snippet"), "r");
    int c;
    while ((c = fgetc(in)) != EOF) fputc(c, src);
    fclose(in); fclose(src);
    int ret = system("gcc -o /tmp/msh_p15_try_out /tmp/msh_p15_try.c 2>&1");
    if (ret != 0) { fprintf(stderr, "Compilation failed.\n"); return 1; }
    system("/tmp/msh_p15_try_out");
    return 0;
}
```

```
#include <stdio.h>

int main(void) {
    printf("=== Caught error: try_block_failed ===\n");
```

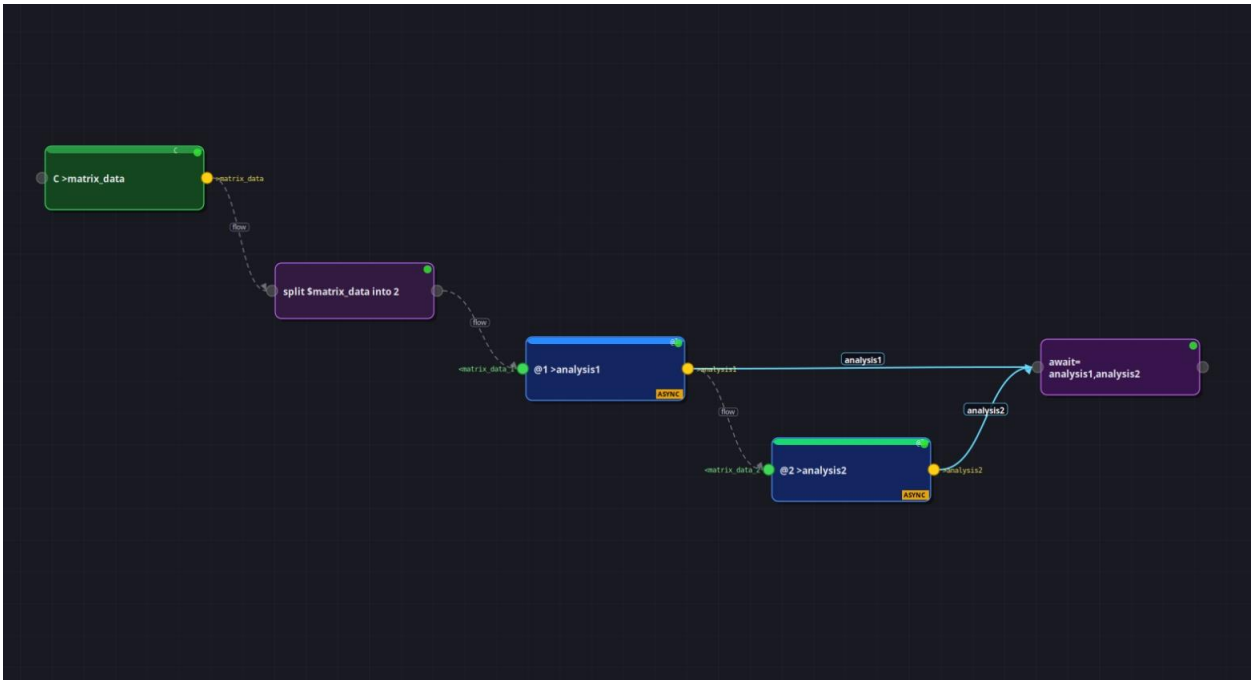
```
    printf("C compilation failed. Activating safe fallback.\n");
    return 0;
}

#include <stdio.h>
#include <stdlib.h>

int main(void) {
    FILE *src = fopen("/tmp/msh_p15_safe.c", "w");
    fprintf(src,
        "#include <stdio.h>\n"
        "int add(int a, int b){return a+b;}\n"
        "int main(void){printf(\"3+4=%%d\\n\",add(3,4));return 0;}\n");
    fclose(src);
    if (system("gcc -o /tmp/msh_p15_safe_out /tmp/msh_p15_safe.c 2>&1") != 0)
{
        printf("Safe fallback failed.\n"); return 1;
    }
    printf("Safe fallback compiled OK.\n");
    system("/tmp/msh_p15_safe_out");
    return 0;
}

#include <stdio.h>
#include <stdlib.h>

int main(void) {
    printf("=== Safe result ===\n");
    FILE *f = fopen(getenv("MSH_VAR_safe_result"), "r"); int c;
    while ((c = fgetc(f)) != EOF) putchar(c); fclose(f);
    return 0;
}
```



Pattern 16 — SPLIT + MERGE: Divide-and-Conquer Matrix Analysis

```c
#include <stdio.h>

int main(void) {
    printf("1 2 3 4 5 6 7 8\n");
    printf("9 10 11 12 13 14 15 16\n");
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

void ps(const char *title, const char *key) {
    printf("=== %s ===\n", title);
    FILE *f = fopen(getenv(key), "r"); int c;
    while ((c = fgetc(f)) != EOF) putchar(c); fclose(f); printf("\n");
}

int main(void) {
    ps("Part 1",     "MSH_VAR_matrix_data_1");
    ps("Analysis 1", "MSH_VAR_analysis1");
    ps("Part 2",     "MSH_VAR_matrix_data_2");
    ps("Analysis 2", "MSH_VAR_analysis2");
    ps("Merged",     "MSH_VAR_combined");
    return 0;
}
```



Pattern 17 — CONFIG Node: Parameterized C Performance Pipeline

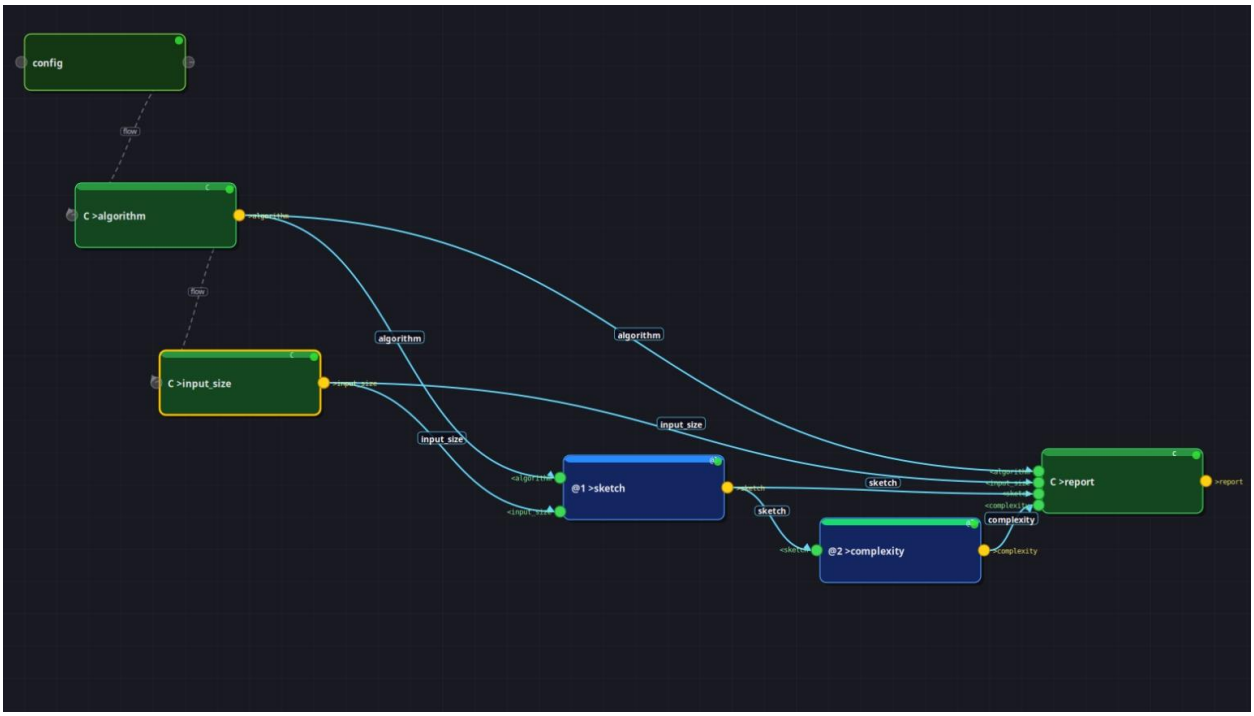
```
algorithm=quicksort
input_size=1000000
iterations=5
```

```c
#include <stdio.h>

int main(void) { printf("quicksort\n"); return 0; }
```

```c
#include <stdio.h>

int main(void) { printf("1000000\n"); return 0; }
```

```c
#include <stdio.h>
#include <stdlib.h>

void pv(const char *l, const char *k) {
    printf("%-16s : ", l);
    FILE *f = fopen(getenv(k), "r"); int c;
    while ((c = fgetc(f)) != EOF && c != '\n') putchar(c); fclose(f); printf(
"\n");
}
void pb(const char *l, const char *k) {
    printf("\n[%s]\n", l);
    FILE *f = fopen(getenv(k), "r"); int c;
    while ((c = fgetc(f)) != EOF) putchar(c); fclose(f);
}

int main(void) {
    printf("=== C Performance Analysis Report ===\n");
```

```
    pv("Algorithm",  "MSH_VAR_algorithm");
    pv("Input size", "MSH_VAR_input_size");
    pb("Sketch",     "MSH_VAR_sketch");
    pb("Complexity", "MSH_VAR_complexity");
    printf("\n");
    return 0;
}
```



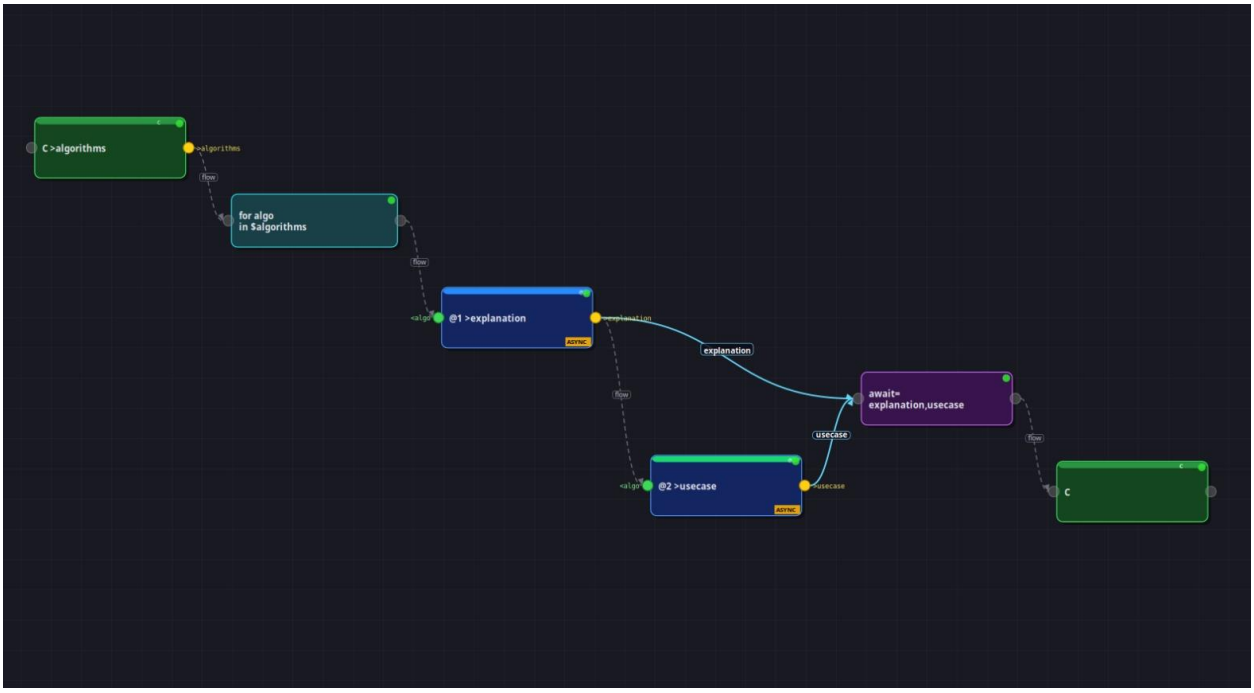
Pattern 18 — FOREACH + Async LLM: Parallel C Algorithm Batch Analysis

```
#include <stdio.h>

int main(void) { printf("dijkstra\naes_encryption\nlru_cache\n"); return 0; }
```

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    printf("=== ");
    FILE *f = fopen(getenv("MSH_VAR_algo"), "r"); int c;
    while ((c = fgetc(f)) != EOF && c != '\n') putchar(c); fclose(f); printf(
" ===\n");
    printf("Explanation : ");
    f = fopen(getenv("MSH_VAR_explanation"), "r");
    while ((c = fgetc(f)) != EOF) putchar(c); fclose(f);
    printf("Real-world  : ");
    f = fopen(getenv("MSH_VAR_usecase"), "r");
    while ((c = fgetc(f)) != EOF) putchar(c); fclose(f); printf("\n");
```

```
    return 0;
}
```

```
#include <stdio.h>

int main(void) { printf("=== All algorithms analyzed ===\n"); return 0; }
```



Pattern 19 — WHILE Quality Gate: Generate C Code Until Score >= 8

```
#include <stdio.h>

int main(void) {
    printf("Write: int is_palindrome(const char *s) "
           "that returns 1 if palindrome, 0 otherwise. "
           "Handle NULL, empty string, single char correctly.\n");
    return 0;
}
```

```
#include <stdio.h>

int main(void) { printf("0\n"); return 0; }
```

```
#include <stdio.h>

int main(void) { printf("0\n"); return 0; }
```

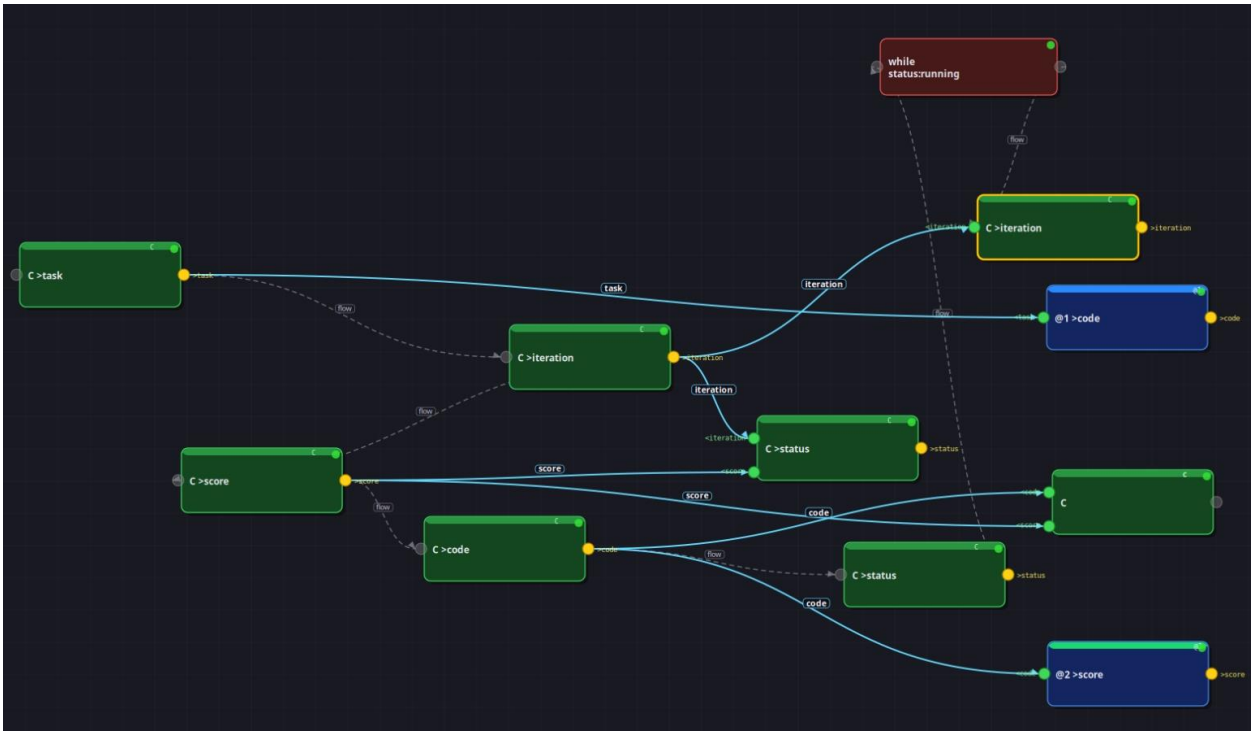
```
#include <stdio.h>

int main(void) { printf("\n"); return 0; }
```

```c
#include <stdio.h>

int main(void) { printf("running\n"); return 0; }
```

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    FILE *f = fopen(getenv("MSH_VAR_iteration"), "r");
    int v; fscanf(f, "%d", &v); fclose(f); v++;
    f = fopen(getenv("MSH_VAR_iteration"), "w");
    fprintf(f, "%d\n", v); fclose(f);
    printf("%d\n", v);
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    FILE *f; int iter, score;
    f = fopen(getenv("MSH_VAR_iteration"), "r"); fscanf(f, "%d", &iter); fclo
se(f);
    f = fopen(getenv("MSH_VAR_score"),     "r"); fscanf(f, "%d", &score); fcl
ose(f);
    printf("[Iter %d] Score=%d\n", iter, score);
    /* Write status to file */
    f = fopen(getenv("MSH_VAR_status"), "w");
    fprintf(f, "%s\n", score >= 8 ? "done" : "running"); fclose(f);
    /* Last stdout line = status value, so WHILE reads it correctly */
    printf("%s\n", score >= 8 ? "done" : "running");
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    int sc;
    FILE *f = fopen(getenv("MSH_VAR_score"), "r"); fscanf(f, "%d", &sc); fclo
se(f);
    printf("=== Accepted C code (score=%d) ===\n", sc);
    f = fopen(getenv("MSH_VAR_code"), "r"); int c;
    while ((c = fgetc(f)) != EOF) putchar(c); fclose(f);
    return 0;
}
```

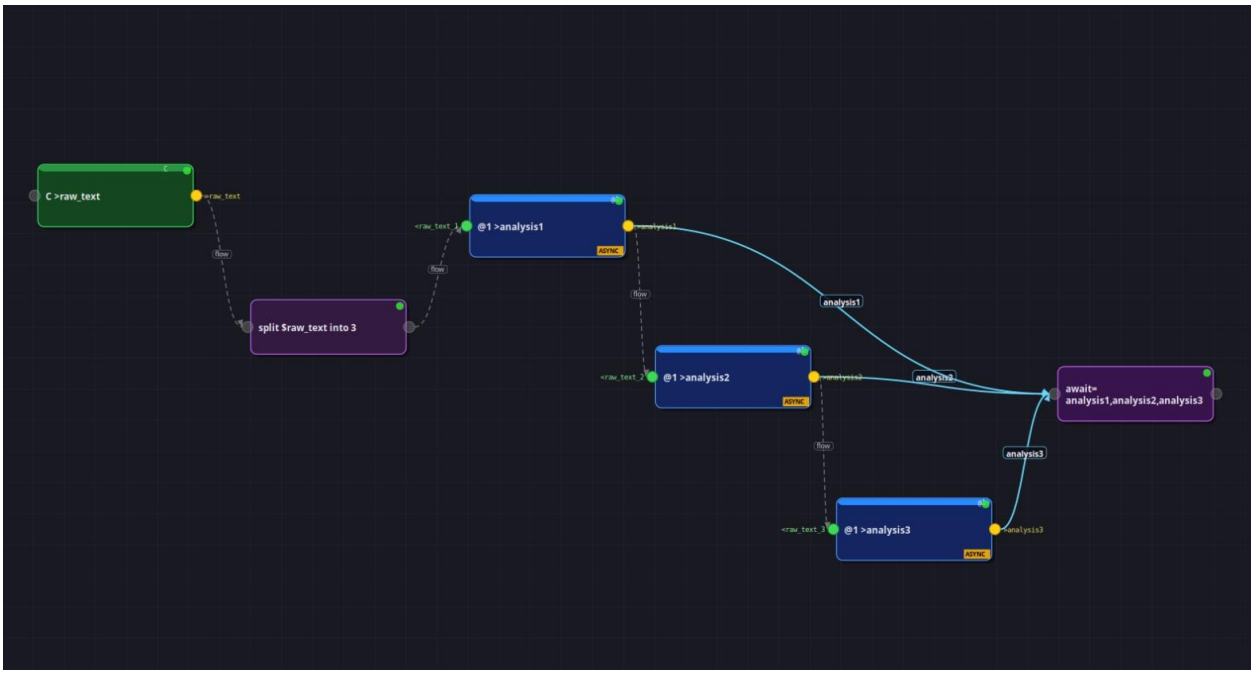


Pattern 20 — SPLIT + Async + MERGE: Map-Reduce C Source Analysis

```
#include <stdio.h>

int main(void) {
    printf("int fib(int n){if(n<=1)return n;return fib(n-1)+fib(n-2);}\n");
    printf("void swap(int*a,int*b){int t=*a;*a=*b;*b=t;}\n");
    printf("int bsearch_c(int*a,int n,int x){int l=0,r=n-1;while(l<=r){int m=
(l+r)/2;if(a[m]==x)return m;else if(a[m]<x)l=m+1;else r=m-1;}return -1;}\n");
    return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>

void show(const char *l, const char *k) {
    printf("%s: ", l);
    FILE *f = fopen(getenv(k), "r"); int c;
    while ((c = fgetc(f)) != EOF) putchar(c); fclose(f); printf("\n");
}

int main(void) {
    printf("=== Map ===\n");
    show("Chunk 1", "MSH_VAR_analysis1");
    show("Chunk 2", "MSH_VAR_analysis2");
    show("Chunk 3", "MSH_VAR_analysis3");
    printf("\n=== Reduce ===\n");
```

```
    show("Summary", "MSH_VAR_summary");
    return 0;
}
```



Pattern 21 — TRY/CATCH + LOOP: Resilient C Code Retry

```c
#include <stdio.h>

int main(void) {
    printf("Write: char *my_strdup(const char *s) using malloc. "
           "Include main() testing with \"hello\" and \"world\", "
           "printing both, freeing memory.\n");
    return 0;
}
```

```c
#include <stdio.h>

int main(void) { printf("none\n"); return 0; }
```

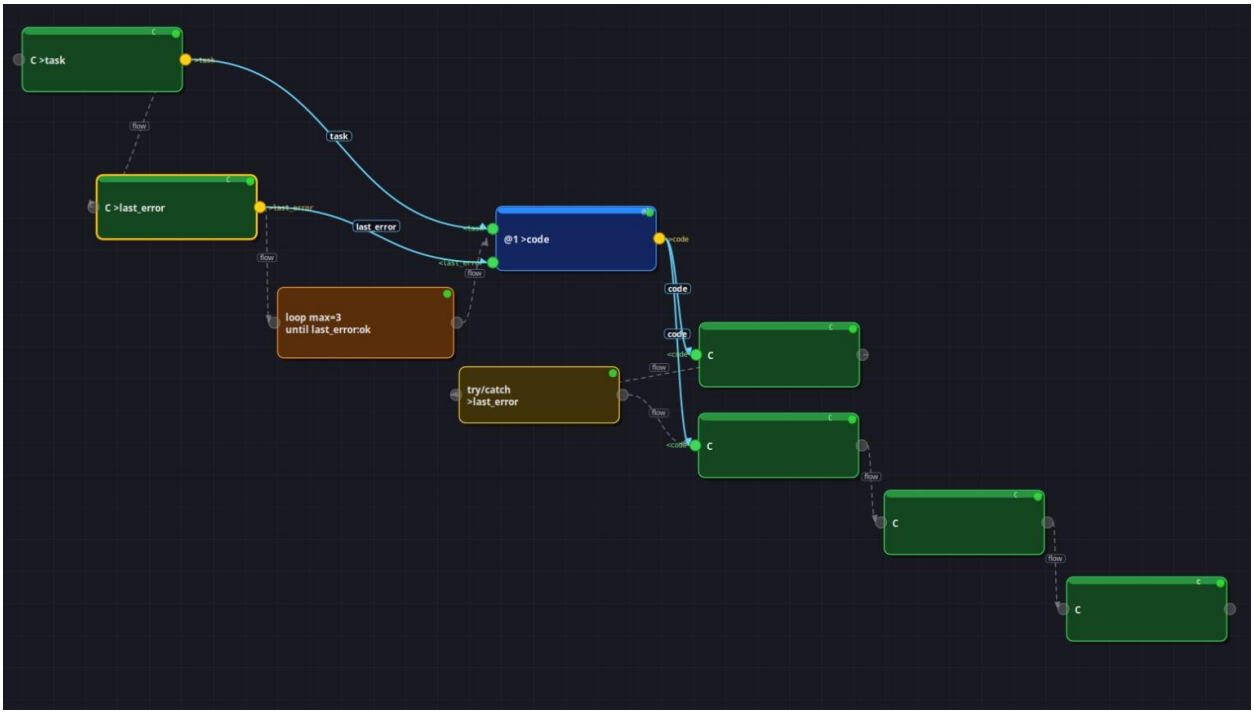
```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    printf("=== Generated C code ===\n");
    FILE *f = fopen(getenv("MSH_VAR_code"), "r"); int c;
    while ((c = fgetc(f)) != EOF) putchar(c); fclose(f); printf("\n");
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
```

```c
int main(void) {
    const char *src = getenv("MSH_VAR_code");
    char cmd[640];
    snprintf(cmd, sizeof(cmd), "cp -- \"%s\" /tmp/msh_p21_loop.c", src);
    system(cmd);
    if (system("gcc -Wall -std=c11 -o /tmp/msh_p21_out /tmp/msh_p21_loop.c 2>
&1") != 0) {
        /* Write error to last_error, print "fail" as last line for LOOP */
        FILE *f = fopen(getenv("MSH_VAR_last_error"), "w");
        fprintf(f, "compilation failed\n"); fclose(f);
        printf("fail\n");
        return 1;
    }
    system("/tmp/msh_p21_out");
    /* Write ok to last_error, print "ok" as last line for LOOP */
    FILE *f = fopen(getenv("MSH_VAR_last_error"), "w");
    fprintf(f, "ok\n"); fclose(f);
    printf("ok\n");
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    printf("=== Error: try_block_failed ===\n");
    FILE *f = fopen(getenv("MSH_VAR_last_error"), "w");
    fprintf(f, "compilation failed\n"); fclose(f);
    /* Print "fail" as last line for LOOP */
    printf("fail\n");
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    FILE *f = fopen(getenv("MSH_VAR_last_error"), "r");
    char buf[64] = {0}; fgets(buf, sizeof(buf), f); fclose(f);
    printf("=== Final status: %s ===\n", buf);
    return 0;
}
```

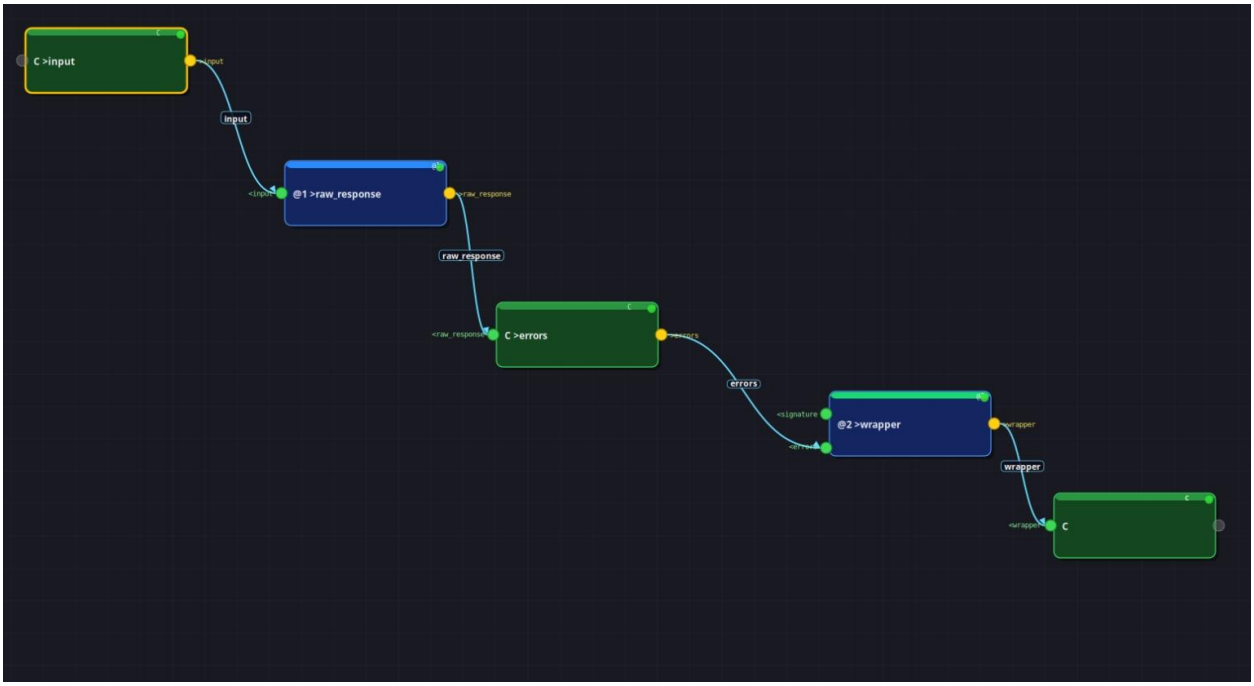


Pattern 22 — Multi-Variable Output: Structured C Function Extraction

```
#include <stdio.h>

int main(void) {
    printf("Describe fopen(): its signature, return value, and error conditio
ns.\n");
    return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char *extract(char *text, const char *label) {
    char *pos = strstr(text, label); if (!pos) return NULL;
    pos += strlen(label); while (*pos == ' ') pos++;
    char *end = strchr(pos, '\n'); if (end) *end = '\0';
    return pos;
}

int main(void) {
    FILE *f = fopen(getenv("MSH_VAR_raw_response"), "r");
    char buf[4096] = {0}; fread(buf, 1, sizeof(buf)-1, f); fclose(f);
    char b2[4096], b3[4096]; strcpy(b2, buf); strcpy(b3, buf);
    char *sig = extract(buf, "SIGNATURE:");
    char *ret = extract(b2,  "RETURNS:");
    char *err = extract(b3,  "ERRORS:");
    FILE *out;
```

```
    out = fopen(getenv("MSH_VAR_signature"), "w"); fprintf(out, "%s\n", sig?s
ig:"n/a"); fclose(out);
    out = fopen(getenv("MSH_VAR_returns"),   "w"); fprintf(out, "%s\n", ret?r
et:"n/a"); fclose(out);
    out = fopen(getenv("MSH_VAR_errors"),    "w"); fprintf(out, "%s\n", err?e
rr:"n/a"); fclose(out);
    printf("Signature : %s\nReturns   : %s\nErrors    : %s\n",
           sig?sig:"n/a", ret?ret:"n/a", err?err:"n/a");
    return 0;
}

#include <stdio.h>
#include <stdlib.h>

int main(void) {
    printf("\n=== Safe C Wrapper ===\n");
    FILE *f = fopen(getenv("MSH_VAR_wrapper"), "r"); int c;
    while ((c = fgetc(f)) != EOF) putchar(c); fclose(f);
    return 0;
}
```



Pattern 23 — CONFIG + WHILE + Multi-Model: Adaptive C Code Pipeline

```
problem=implement a stack using a fixed-size array in C
target_level=junior C programmer
quality_threshold=7

#include <stdio.h>

int main(void) { printf("implement a stack using a fixed-size array in C\n");
return 0; }
```

```c
#include <stdio.h>

int main(void) { printf("junior C programmer\n"); return 0; }
```

```c
#include <stdio.h>

int main(void) { printf("0\n"); return 0; }
```

```c
#include <stdio.h>

int main(void) { printf("0\n"); return 0; }
```

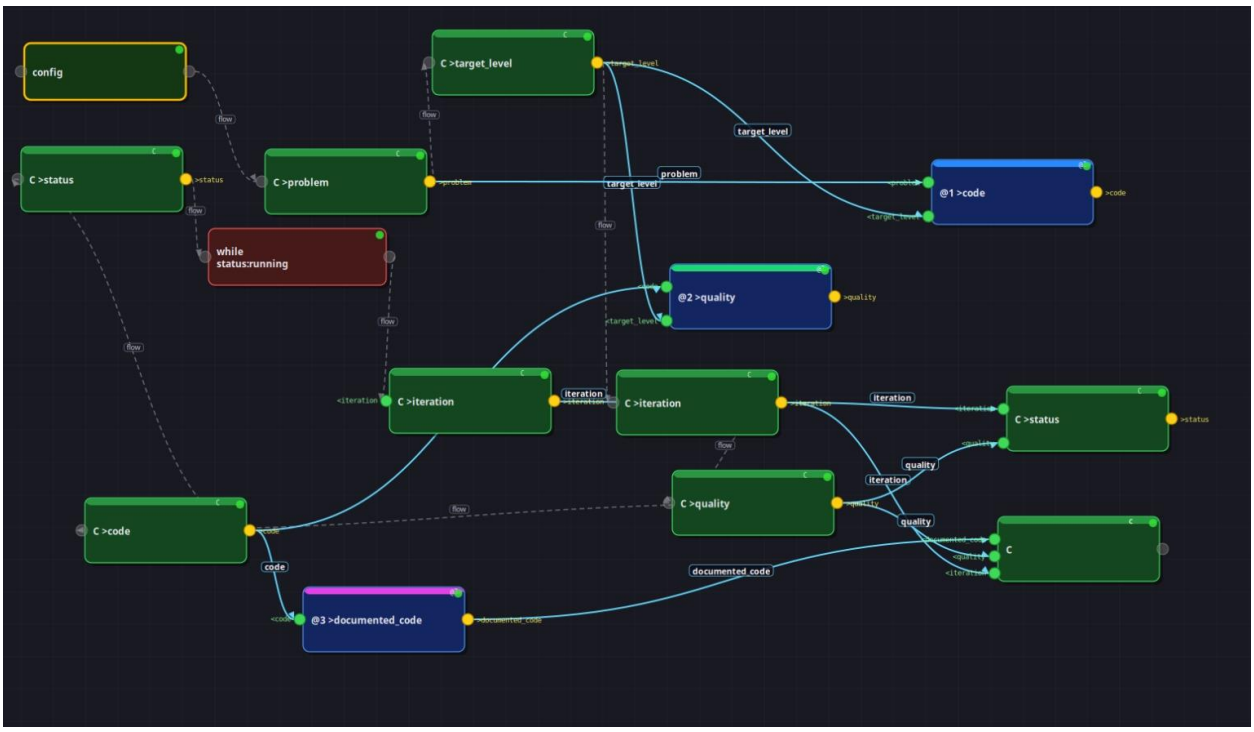
```c
#include <stdio.h>

int main(void) { printf("\n"); return 0; }
```

```c
#include <stdio.h>

int main(void) { printf("running\n"); return 0; }
```

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    FILE *f = fopen(getenv("MSH_VAR_iteration"), "r");
    int v; fscanf(f, "%d", &v); fclose(f); v++;
    f = fopen(getenv("MSH_VAR_iteration"), "w");
    fprintf(f, "%d\n", v); fclose(f);
    printf("%d\n", v);
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    FILE *f; int it, sc;
    f = fopen(getenv("MSH_VAR_iteration"), "r"); fscanf(f, "%d", &it); fclose
(f);
    f = fopen(getenv("MSH_VAR_quality"),   "r"); fscanf(f, "%d", &sc); fclose
(f);
    printf("[Iter %d] Quality: %d\n", it, sc);
    /* Write status to file */
    f = fopen(getenv("MSH_VAR_status"), "w");
    fprintf(f, "%s\n", sc >= 7 ? "done" : "running"); fclose(f);
    /* Last stdout line = status for WHILE condition check */
    printf("%s\n", sc >= 7 ? "done" : "running");
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    FILE *f; int q, it;
    f = fopen(getenv("MSH_VAR_quality"),   "r"); fscanf(f, "%d", &q);  fclose
(f);
    f = fopen(getenv("MSH_VAR_iteration"), "r"); fscanf(f, "%d", &it); fclose
(f);
    printf("=== Final Code (score=%d, iters=%d) ===\n", q, it);
    f = fopen(getenv("MSH_VAR_documented_code"), "r"); int c;
    while ((c = fgetc(f)) != EOF) putchar(c); fclose(f);
    return 0;
}
```

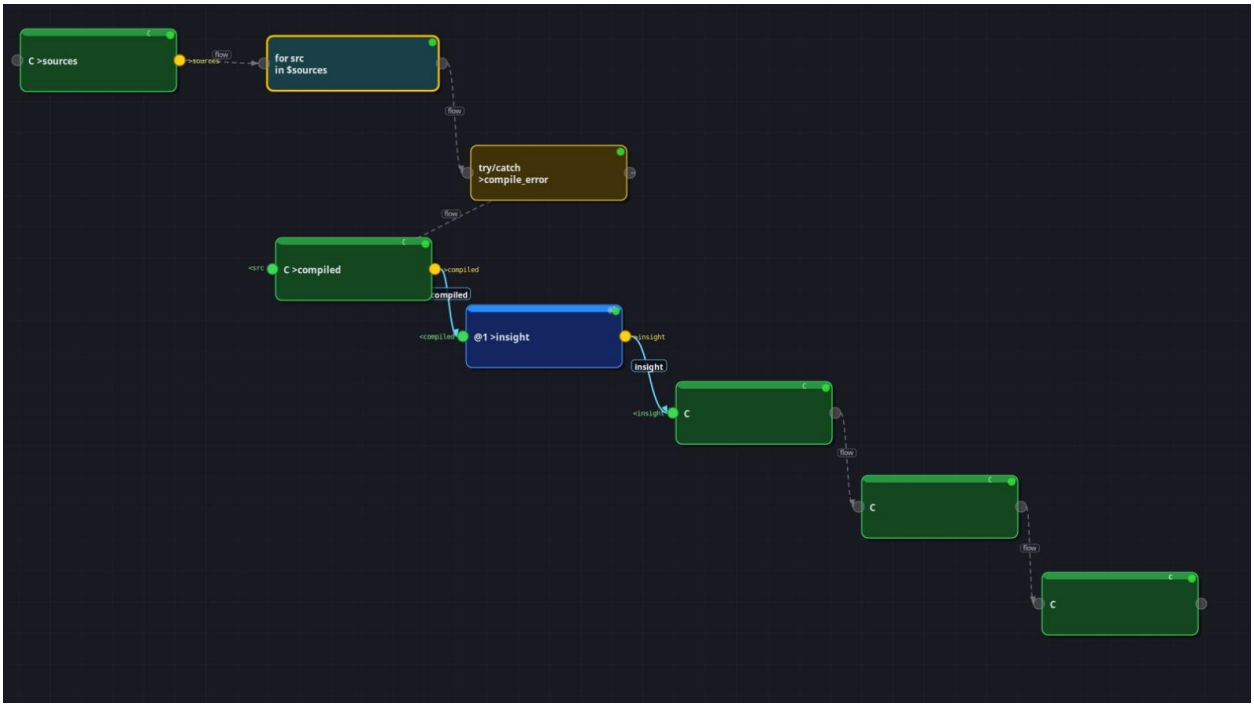


Pattern 24 — FOREACH + TRY/CATCH: Fault-Tolerant C Batch Compilation

```c
#include <stdio.h>

int main(void) {
    printf("/tmp/msh_src1.c\n/tmp/msh_broken.c\n/tmp/msh_src2.c\n");
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void) {
```

```
    char src_path[256] = {0};
    FILE *f = fopen(getenv("MSH_VAR_src"), "r");
    fgets(src_path, sizeof(src_path), f); fclose(f);
    src_path[strcspn(src_path, "\n")] = '\0';

    FILE *out = fopen(src_path, "w");
    if (strstr(src_path, "broken"))
        fprintf(out, "#include <stdio.h>\nint main(void) { printf(\"hello\");
\n");
    else if (strstr(src_path, "src1"))
        fprintf(out, "#include <stdio.h>\nint main(void){int i;for(i=1;i<=5;i
++)printf(\"%d \",i*i);printf(\"\\n\");return 0;}\n");
    else
        fprintf(out, "#include <stdio.h>\n#include <math.h>\nint main(void){p
rintf(\"pi=%.6f\\n\",4.0*atan(1.0));return 0;}\n");
    fclose(out);

    char bin[256], cmd[512];
    strcpy(bin, src_path);
    char *dot = strrchr(bin, '.'); if (dot) *dot = '\0';
    snprintf(cmd, sizeof(cmd), "gcc -Wall -std=c11 -o %s %s -lm 2>&1", bin, s
rc_path);
    if (system(cmd) != 0) return 1;
    system(bin);
    return 0;
}

#include <stdio.h>
#include <stdlib.h>

int main(void) {
    printf("[OK] ");
    FILE *f = fopen(getenv("MSH_VAR_insight"), "r"); int c;
    while ((c = fgetc(f)) != EOF) putchar(c); fclose(f); printf("\n");
    return 0;
}

#include <stdio.h>

int main(void) { printf("[ERR] Compilation failed: try_block_failed\n"); retu
rn 0; }

#include <stdio.h>

int main(void) {
    printf("=== Batch complete. Errors isolated, pipeline never stopped. ===\
n");
    return 0;
}
```



---

mshell C Language Edition — v5 Final Art2Dec SoftLab, 2026

Quick Reference: Common C Patterns

Read one variable:

```c
FILE *f = fopen(getenv("MSH_VAR_input"), "r");
char buf[4096] = {0};
fread(buf, 1, sizeof(buf)-1, f);
fclose(f);
```

Fan-out producer (write to both stdout and stable path):

```c
FILE *tmp = fopen("/tmp/msh_pN_data.txt", "w");
/* write to both printf() and fprintf(tmp, ...) */
fclose(tmp);
/* consumers read from /tmp/msh_pN_data.txt directly */
```

Compile LLM-generated code:

```c
const char *src = getenv("MSH_VAR_code");
char cmd[640];
snprintf(cmd, sizeof(cmd), "cp -- \"%s\" /tmp/msh_gen.c", src);
system(cmd);
if (system("gcc -Wall -std=c11 -o /tmp/msh_out /tmp/msh_gen.c -lm 2>&1") == 0
```

```
)
    system("/tmp/msh_out");
```

WHILE exit (write to file):

```
FILE *f = fopen(getenv("MSH_VAR_status"), "w");
fprintf(f, "%s\n", done ? "done" : "running");
fclose(f);
```

LOOP exit (print as last stdout line):

```
/* last line of last block in loop body */
printf("%s\n", success ? "ok" : "fail");
```

Appendix I: Code examples for each pattern

Sent to mshell (2590 bytes)

Received from GUI editor:

mshell C Language Edition

Pattern 1 Linear Data Pipeline

```c >primes

#include <stdio.h>

int is_prime(int n) {

  if (n < 2) return 0;

  for (int i = 2; i * i <= n; i++)

    if (n % i == 0) return 0;

  return 1;

}

int main(void) {

  int count = 0;

  for (int n = 2; count < 10; n++) {

    if (is_prime(n)) {

      printf("%d", n);
```

```
        if (++count < 10) printf(" ");
      }
    }
    printf("\n");
    return 0;
}
```

```c <primes >stats
#include <stdio.h>
#include <stdlib.h>
int main(void) {
    FILE *f = fopen(getenv("MSH_VAR_primes"), "r");
    int nums[32], n = 0, sum = 0, mx = -1, mn = 1<<30;
    while (fscanf(f, "%d", &nums[n]) == 1) {
        sum += nums[n];
        if (nums[n] > mx) mx = nums[n];
        if (nums[n] < mn) mn = nums[n];
        n++;
    }
    fclose(f);
    printf("count=%d sum=%d mean=%.2f min=%d max=%d\n",
           n, sum, (double)sum/n, mn, mx);
    return 0;
}
```

```c <stats >encoded
#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>

int main(void) {
   FILE *f = fopen(getenv("MSH_VAR_stats"), "r");
   char buf[256] = {0};
   fgets(buf, sizeof(buf), f); fclose(f);
   int len = strlen(buf);
   printf("HEX:");
   for (int i = 0; i < len && buf[i] != '\n'; i++)
      printf("%02X", (unsigned char)buf[i]);
   printf("\nRAW:%s", buf);
   return 0;
}
```

```c <encoded >report
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(void) {
   FILE *f = fopen(getenv("MSH_VAR_encoded"), "r");
   char line[512];
   printf("%-20s %-60s\n", "FIELD", "VALUE");
   printf("%-20s %-60s\n", "--------------------", "----");
   while (fgets(line, sizeof(line), f)) {
      char *colon = strchr(line, ':');
      if (colon) { *colon = '\0'; printf("%-20s %-60s\n", line, colon+1); }
```

```
    }
    fclose(f);
    return 0;
}
```

```c <report >upper
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
int main(void) {
    FILE *f = fopen(getenv("MSH_VAR_report"), "r"); int c;
    while ((c = fgetc(f)) != EOF) putchar(toupper(c));
    fclose(f);
    return 0;
}
```

```c <upper
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(void) {
    FILE *f = fopen(getenv("MSH_VAR_upper"), "r");
    char lines[32][256]; int n = 0;
    while (n < 32 && fgets(lines[n], 256, f)) n++;
    fclose(f);
    int width = 70;
    for (int i = 0; i < width; i++) putchar('='); putchar('\n');
```

```
    for (int i = 0; i < n; i++) printf("| %-*s|\n", width-3, lines[i]);
    for (int i = 0; i < width; i++) putchar('='); putchar('\n');
    return 0;
}
```

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

2 3 5 7 11 13 17 19 23 29

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

count=10 sum=129 mean=12.90 min=2 max=29

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

HEX:636F756E743D31302073756D3D313239206D65616E3D31322E3930206D696E3D32206D61783D3239

RAW:count=10 sum=129 mean=12.90 min=2 max=29

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

FIELD	VALUE
HEX	636F756E743D31302073756D3D313239206D65616E3D31322E3930206D696E3D32206D61783D3239
RAW	count=10 sum=129 mean=12.90 min=2 max=29

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

FIELD VALUE

---------- ----

HEX 636F756E743D31302073756D3D313239206D65616E3D31322E3930206D696E3D32206D61783D3239

RAW COUNT=10 SUM=129 MEAN=12.90 MIN=2 MAX=29

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

/home/igor >
==========

| FIELD VALUE

| ---------- ----

| HEX 636F756E743D31302073756D3D313239206D65616E3D31322E3930206D696E3D32206D61783D3239 |

| RAW COUNT=10 SUM=129 MEAN=12.90 MIN=2 MAX=29 |

==========

/home/igor >

/home/igor > Sent to mshell (1318 bytes)

Received from GUI editor:

---

mshell C Language Edition

Pattern 2 LLM in the Middle

```c >data

#include <stdio.h>
```

```
int main(void) {
   const char *algos[] =
{"BubbleSort","MergeSort","QuickSort","HeapSort","RadixSort","TimSort"};
   double ms[] = {812.4, 45.2, 38.7, 51.3, 29.1, 41.8};
   printf("Algorithm Benchmark (ms per 1M elements):\n");
   for (int i = 0; i < 6; i++)
      printf("  %-12s : %.1f ms\n", algos[i], ms[i]);
   return 0;
}
```

<!--@1 <data >analysis
The input contains a benchmark table of sorting algorithms in milliseconds.
Identify the fastest algorithm, explain why it performs well, and note trade-offs.
One paragraph, maximum 80 words.
-->

```c <analysis
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
int main(void) {
   FILE *f = fopen(getenv("MSH_VAR_analysis"), "r");
   char buf[4096] = {0};
   fread(buf, 1, sizeof(buf)-1, f); fclose(f);
   int words = 0, sentences = 0, chars = 0, in_word = 0;
   for (int i = 0; buf[i]; i++) {
      chars++;
      if (isspace(buf[i])) in_word = 0;
```

```
      else if (!in_word) { words++; in_word = 1; }
      if (buf[i]=='.'||buf[i]=='!'||buf[i]=='?') sentences++;
   }
   printf("=== Metadata: words=%d sentences=%d chars=%d ===\n\n", words, sentences, chars);
   printf("=== Analysis ===\n%s\n", buf);
   return 0;
}
```

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

Algorithm Benchmark (ms per 1M elements):

  BubbleSort   : 812.4 ms

  MergeSort   : 45.2 ms

  QuickSort   : 38.7 ms

  HeapSort   : 51.3 ms

  RadixSort   : 29.1 ms

  TimSort   : 41.8 ms

RadixSort is the fastest at 29.1ms because it avoids comparisons entirely, instead sorting by processing individual digits/bits in linear O(nk) time where k is the number of digits. This makes it exceptionally efficient for integers and fixed-length data. However, RadixSort has significant trade-offs: it's limited to specific data types (integers, strings), requires extra memory proportional to the range of values, and performs poorly with variable-length or complex data structures.

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

=== Metadata: words=69 sentences=4 chars=491 ===

=== Analysis ===

RadixSort is the fastest at 29.1ms because it avoids comparisons entirely, instead sorting by processing individual digits/bits in linear O(nk) time where k is the number of digits. This makes it exceptionally efficient for integers and fixed-length data. However, RadixSort has significant trade-offs: it's limited to specific data types (integers, strings), requires extra memory proportional to the range of values, and performs poorly with variable-length or complex data structures.

/home/igor >

/home/igor > Sent to mshell (2128 bytes)

Received from GUI editor:

---

mshell C Language Edition

Pattern 3 Fan-Out (One Variable - Many Consumers)

```c >data
#include <stdio.h>
#include <stdlib.h>
int main(void) {
   srand(12345);
   FILE *tmp = fopen("/tmp/msh_p3_data.txt", "w");
   for (int i = 0; i < 100; i++) {
      int v = rand() % 200 - 100;
      printf("%d", v);
      fprintf(tmp, "%d", v);
      if (i < 99) { printf(" "); fprintf(tmp, " "); }
   }
   printf("\n"); fprintf(tmp, "\n");
   fclose(tmp);
   return 0;
```

```
}
```

```c <data >stats_out
#include <stdio.h>
#include <math.h>
int main(void) {
    FILE *f = fopen("/tmp/msh_p3_data.txt", "r");
    double v[256]; int n = 0;
    while (n < 256 && fscanf(f, "%lf", &v[n]) == 1) n++;
    fclose(f);
    double sum = 0;
    for (int i = 0; i < n; i++) sum += v[i];
    double mean = sum/n, var = 0;
    for (int i = 0; i < n; i++) var += (v[i]-mean)*(v[i]-mean);
    var /= n;
    printf("n=%d mean=%.2f variance=%.2f stddev=%.2f\n", n, mean, var, sqrt(var));
    return 0;
}
```

```c <data >top5_out
#include <stdio.h>
#include <stdlib.h>
int cmp_desc(const void *a, const void *b) { return *(int*)b - *(int*)a; }
int main(void) {
    FILE *f = fopen("/tmp/msh_p3_data.txt", "r");
    int v[256], n = 0;
    while (n < 256 && fscanf(f, "%d", &v[n]) == 1) n++;
```

```
    fclose(f);
    qsort(v, n, sizeof(int), cmp_desc);
    printf("Top-5:");
    for (int i = 0; i < 5 && i < n; i++) printf(" %d", v[i]);
    printf("\n");
    return 0;
}
```

```c <data >parity_out
#include <stdio.h>
int main(void) {
    FILE *f = fopen("/tmp/msh_p3_data.txt", "r");
    int v, evens = 0, odds = 0, runs = 1, prev_sign = 0;
    while (fscanf(f, "%d", &v) == 1) {
        if (v % 2 == 0) evens++; else odds++;
        int s = (v >= 0) ? 1 : -1;
        if (s != prev_sign && prev_sign != 0) runs++;
        prev_sign = s;
    }
    fclose(f);
    printf("even=%d odd=%d sign_runs=%d\n", evens, odds, runs);
    return 0;
}
```

<!--@1 <stats_out <top5_out <parity_out

The inputs contain three independent analyses of the same integer dataset.

Summarize what kind of dataset this likely is in two sentences.

-->

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

99 -79 73 -49 21 4 37 2 28 28 -73 -3 21 -47 -94 73 38 12 -43 12 61 95 -96 42 23 79 -73 -14 -35 -57 -23 -36 -83 -49 -33 38 55 -96 -59 -17 32 -32 32 -47 21 38 78 -89 -50 -65 -25 63 -70 -21 5 53 -89 32 91 -24 28 -79 -8 -3 -28 59 -65 79 63 -24 -86 -53 -4 46 -48 69 -16 -70 32 34 -35 -92 -2 -52 -61 55 53 -50 -60 45 -22 20 66 22 -83 90 -19 -96 69 96

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

n=100 mean=-4.15 variance=3072.59 stddev=55.43

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

Top-5: 99 96 95 91 90

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

even=47 odd=53 sign_runs=49

This dataset appears to be a collection of 100 integers with a strong concentration of large positive values, as evidenced by the top-5 values all being in the 90s range and the relatively small negative mean (-4.15) despite having many high-value outliers. The high variance (3072.59) and standard deviation (55.43) suggest a bimodal or heavily skewed distribution, likely containing both very large positive numbers and some significantly negative values that pull the mean below zero.

/home/igor >

/home/igor > Sent to mshell (1078 bytes)

Received from GUI editor:

---

mshell C Language Edition

Pattern 4 LLM Code Generation - Compile & Execute

```c >task
#include <stdio.h>
int main(void) {
   printf("Write a C function matrix_mul(int A[2][2], int B[2][2], int C[2][2]) "
        "that multiplies two 2x2 integer matrices and stores the result in C. "
        "Include a main() with A={{1,2},{3,4}} B={{5,6},{7,8}} and prints the result.\n");
   return 0;
}
```

<!--@1 <task >code
Write complete, compilable C code for the task. No markdown fences, no explanation.
-->

```c <code
#include <stdio.h>
#include <stdlib.h>
int main(void) {
   const char *src = getenv("MSH_VAR_code");
   char cmd[640];
   snprintf(cmd, sizeof(cmd), "cp -- \"%s\" /tmp/msh_p4_gen.c", src);
   system(cmd);
   printf("=== Generated C code ===\n");
   system("cat /tmp/msh_p4_gen.c");
   printf("\n=== Compiling & Running ===\n");
   if (system("gcc -Wall -std=c11 -o /tmp/msh_p4_out /tmp/msh_p4_gen.c -lm 2>&1") ==
0)
      system("/tmp/msh_p4_out");
```

```
    else
        printf("Compilation failed.\n");
    return 0;
}
```

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

Write a C function matrix_mul(int A[2][2], int B[2][2], int C[2][2]) that multiplies two 2x2 integer matrices and stores the result in C. Include a main() with A={{1,2},{3,4}} B={{5,6},{7,8}} and prints the result.

```
#include <stdio.h>
void matrix_mul(int A[2][2], int B[2][2], int C[2][2]) {
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            C[i][j] = 0;
            for (int k = 0; k < 2; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}
int main() {
    int A[2][2] = {{1, 2}, {3, 4}};
    int B[2][2] = {{5, 6}, {7, 8}};
    int C[2][2];
    matrix_mul(A, B, C);
    printf("Result:\n");
```

```
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            printf("%d ", C[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

/home/igor > === Generated C code ===

```
#include <stdio.h>
void matrix_mul(int A[2][2], int B[2][2], int C[2][2]) {
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            C[i][j] = 0;
            for (int k = 0; k < 2; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}
int main() {
    int A[2][2] = {{1, 2}, {3, 4}};
    int B[2][2] = {{5, 6}, {7, 8}};
    int C[2][2];
```

```
    matrix_mul(A, B, C);
    printf("Result:\n");
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            printf("%d ", C[i][j]);
        }
        printf("\n");
    }
    return 0;
}
=== Compiling & Running ===
Result:
19 22
43 50
/home/igor >
------------------------------------------------------------
/home/igor > Sent to mshell (1905 bytes)
Received from GUI editor:
----------------------------------------
# mshell C Language Edition
### Pattern 5  Two-LLM Review Chain
```c >task
#include <stdio.h>
int main(void) {
    printf("Implement a binary search tree in C with:\n"
           "1. Node struct (int value, left, right pointers).\n"
           "2. bst_insert(Node **root, int val).\n"
```
```

```
        "3. inorder(Node *root) prints values.\n"
        "4. main() inserts 5,3,7,1,4,6,8 and prints inorder.\n"
        "5. Free all nodes at the end.\n");
    return 0;
}
```

<!--@1 <task >code

Write complete, compilable C code. No markdown fences, no explanation.

-->

```c <code
#include <stdio.h>
#include <stdlib.h>
int main(void) {
    printf("=== Generated Code ===\n");
    FILE *f = fopen(getenv("MSH_VAR_code"), "r"); int c;
    while ((c = fgetc(f)) != EOF) putchar(c);
    fclose(f); printf("\n");
    return 0;
}
```

<!--@2 <code >review

Review this C code strictly for correctness, memory leaks, and undefined behavior.

Reply in 3 sentences maximum.

-->

```c <review
#include <stdio.h>
#include <stdlib.h>
```

```
int main(void) {
   printf("=== Review ===\n");
   FILE *f = fopen(getenv("MSH_VAR_review"), "r"); int c;
   while ((c = fgetc(f)) != EOF) putchar(c);
   fclose(f); printf("\n");
   return 0;
}
```

<!--@1 <code <review >improved

Improve the C code based on the review. Return only the improved C source, no fences.

-->

```c <improved
#include <stdio.h>
#include <stdlib.h>
int main(void) {
   const char *src = getenv("MSH_VAR_improved");
   char cmd[640];
   snprintf(cmd, sizeof(cmd), "cp -- \"%s\" /tmp/msh_p5_bst.c", src);
   system(cmd);
   printf("=== Improved Code ===\n");
   system("cat /tmp/msh_p5_bst.c");
   printf("\n=== Compiling & Running ===\n");
   if (system("gcc -Wall -std=c11 -o /tmp/msh_p5_out /tmp/msh_p5_bst.c 2>&1") == 0)
      system("/tmp/msh_p5_out");
   else
      printf("Compilation failed.\n");
   return 0;
```

}

```

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

Implement a binary search tree in C with:

1. Node struct (int value, left, right pointers).

2. bst_insert(Node **root, int val).

3. inorder(Node *root) prints values.

4. main() inserts 5,3,7,1,4,6,8 and prints inorder.

5. Free all nodes at the end.

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

   int value;

   struct Node *left;

   struct Node *right;

} Node;

void bst_insert(Node **root, int val) {

   if (*root == NULL) {

      *root = (Node*)malloc(sizeof(Node));

      (*root)->value = val;

      (*root)->left = NULL;

      (*root)->right = NULL;

   } else if (val < (*root)->value) {

      bst_insert(&((*root)->left), val);

   } else {
```

```
        bst_insert(&((*root)->right), val);
    }
}
void inorder(Node *root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->value);
        inorder(root->right);
    }
}
void free_tree(Node *root) {
    if (root != NULL) {
        free_tree(root->left);
        free_tree(root->right);
        free(root);
    }
}
int main() {
    Node *root = NULL;
    bst_insert(&root, 5);
    bst_insert(&root, 3);
    bst_insert(&root, 7);
    bst_insert(&root, 1);
    bst_insert(&root, 4);
    bst_insert(&root, 6);
    bst_insert(&root, 8);
    printf("Inorder traversal: ");
```

```
    inorder(root);
    printf("\n");
    free_tree(root);
    return 0;
}
```

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

=== Generated Code ===

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
    int value;
    struct Node *left;
    struct Node *right;
} Node;
void bst_insert(Node **root, int val) {
    if (*root == NULL) {
        *root = (Node*)malloc(sizeof(Node));
        (*root)->value = val;
        (*root)->left = NULL;
        (*root)->right = NULL;
    } else if (val < (*root)->value) {
        bst_insert(&((*root)->left), val);
    } else {
        bst_insert(&((*root)->right), val);
    }
```

```
}
void inorder(Node *root) {
   if (root != NULL) {
      inorder(root->left);
      printf("%d ", root->value);
      inorder(root->right);
   }
}
void free_tree(Node *root) {
   if (root != NULL) {
      free_tree(root->left);
      free_tree(root->right);
      free(root);
   }
}
int main() {
   Node *root = NULL;
   bst_insert(&root, 5);
   bst_insert(&root, 3);
   bst_insert(&root, 7);
   bst_insert(&root, 1);
   bst_insert(&root, 4);
   bst_insert(&root, 6);
   bst_insert(&root, 8);
   printf("Inorder traversal: ");
   inorder(root);
   printf("\n");
```

```
    free_tree(root);

    return 0;

}
```

The code is functionally correct as a basic BST insertion and inorder traversal, and `free_tree` properly frees all allocated nodes, so there are no memory leaks in normal execution. However, `bst_insert` does not check whether `malloc` returns `NULL`, which can lead to undefined behavior if memory allocation fails. Additionally, all comparison-equal values go to the right subtree; this is logically acceptable but worth noting as the chosen duplicate-handling policy rather than a correctness bug.

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

=== Review ===

The code is functionally correct as a basic BST insertion and inorder traversal, and `free_tree` properly frees all allocated nodes, so there are no memory leaks in normal execution. However, `bst_insert` does not check whether `malloc` returns `NULL`, which can lead to undefined behavior if memory allocation fails. Additionally, all comparison-equal values go to the right subtree; this is logically acceptable but worth noting as the chosen duplicate-handling policy rather than a correctness bug.

```
#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

    int value;

    struct Node *left;

    struct Node *right;

} Node;

void bst_insert(Node **root, int val) {

    if (*root == NULL) {

        *root = (Node*)malloc(sizeof(Node));

        if (*root == NULL) {

            printf("Memory allocation failed\n");
```

```
            exit(1);
        }
        (*root)->value = val;
        (*root)->left = NULL;
        (*root)->right = NULL;
    } else if (val < (*root)->value) {
        bst_insert(&((*root)->left), val);
    } else {
        bst_insert(&((*root)->right), val);
    }
}
void inorder(Node *root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->value);
        inorder(root->right);
    }
}
void free_tree(Node *root) {
    if (root != NULL) {
        free_tree(root->left);
        free_tree(root->right);
        free(root);
    }
}
int main() {
    Node *root = NULL;
```

```
    bst_insert(&root, 5);
    bst_insert(&root, 3);
    bst_insert(&root, 7);
    bst_insert(&root, 1);
    bst_insert(&root, 4);
    bst_insert(&root, 6);
    bst_insert(&root, 8);
    printf("Inorder traversal: ");
    inorder(root);
    printf("\n");
    free_tree(root);
    return 0;
}
[DEBUG compile_c] libs: [-lm]
[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out
/tmp/mshell_validate_301141.c -lm 2>&1]
/home/igor > === Improved Code ===
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
    int value;
    struct Node *left;
    struct Node *right;
} Node;
void bst_insert(Node **root, int val) {
    if (*root == NULL) {
        *root = (Node*)malloc(sizeof(Node));
```

```
        if (*root == NULL) {
            printf("Memory allocation failed\n");
            exit(1);
        }
        (*root)->value = val;
        (*root)->left = NULL;
        (*root)->right = NULL;
    } else if (val < (*root)->value) {
        bst_insert(&((*root)->left), val);
    } else {
        bst_insert(&((*root)->right), val);
    }
}
void inorder(Node *root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->value);
        inorder(root->right);
    }
}
void free_tree(Node *root) {
    if (root != NULL) {
        free_tree(root->left);
        free_tree(root->right);
        free(root);
    }
}
```

```
int main() {
    Node *root = NULL;
    bst_insert(&root, 5);
    bst_insert(&root, 3);
    bst_insert(&root, 7);
    bst_insert(&root, 1);
    bst_insert(&root, 4);
    bst_insert(&root, 6);
    bst_insert(&root, 8);
    printf("Inorder traversal: ");
    inorder(root);
    printf("\n");
    free_tree(root);
    return 0;
}
=== Compiling & Running ===
Inorder traversal: 1 3 4 5 6 7 8
/home/igor >
------------------------------------------------------------
/home/igor > Sent to mshell (1188 bytes)
Received from GUI editor:
---------------------------------------
# mshell C Language Edition
### Pattern 6  Parallel 3-Model Query
```c >question
#include <stdio.h>
int main(void) {
```
```

```
   printf("In C, explain stack vs heap: lifetime, size limits, fragmentation, "
          "and when to prefer each. One paragraph, maximum 60 words.\n");
   return 0;
}
```

<!--@1 <question >ans1
Answer in exactly one paragraph.
-->

<!--@2 <question >ans2
Answer in exactly one paragraph.
-->

<!--@3 <question >ans3
Answer in exactly one paragraph.
-->

```c <ans1 <ans2 <ans3
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
int wc(const char *path) {
   FILE *f = fopen(path, "r"); int c, w = 0, iw = 0;
   while ((c = fgetc(f)) != EOF) {
      if (isspace(c)) iw = 0;
      else if (!iw) { w++; iw = 1; }
   }
   fclose(f); return w;
}
void pa(const char *label, const char *path) {
```

```
    printf("=== %s (%d words) ===\n", label, wc(path));
    FILE *f = fopen(path, "r"); int c;
    while ((c = fgetc(f)) != EOF) putchar(c);
    fclose(f); printf("\n\n");
}
int main(void) {
    pa("Model @1", getenv("MSH_VAR_ans1"));
    pa("Model @2", getenv("MSH_VAR_ans2"));
    pa("Model @3", getenv("MSH_VAR_ans3"));
    return 0;
}
```

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

In C, explain stack vs heap: lifetime, size limits, fragmentation, and when to prefer each. One paragraph, maximum 60 words.

Stack memory has automatic lifetime (function scope), limited size (~MB), no fragmentation, and fast allocation, making it ideal for local variables and small data. Heap memory requires manual management, has larger capacity (GB), suffers fragmentation, and slower allocation, preferred for dynamic data structures, large objects, and data that must persist beyond function scope.

Stack memory is automatically managed, small, fast, contiguous, with fixed size limits and no fragmentation; it suits small, short\u2011lived variables and recursion. Heap memory is manually managed, larger, slower, may fragment, and persists until freed; it suits large, dynamic\u2011lifetime data structures and objects whose size or lifetime isn\u2019t known at compile time.

Stack memory is automatic, function-scoped, and fast but small and fixed-size. Heap is manually managed (malloc/free), larger but limited by system RAM, prone to fragmentation. Prefer stack for small, short-lived data; use heap for large, long-lived, or dynamically sized data. (58 words)

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

=== Model @1 (53 words) ===

Stack memory has automatic lifetime (function scope), limited size (~MB), no fragmentation, and fast allocation, making it ideal for local variables and small data. Heap memory requires manual management, has larger capacity (GB), suffers fragmentation, and slower allocation, preferred for dynamic data structures, large objects, and data that must persist beyond function scope.

=== Model @2 (52 words) ===

Stack memory is automatically managed, small, fast, contiguous, with fixed size limits and no fragmentation; it suits small, short\u2011lived variables and recursion. Heap memory is manually managed, larger, slower, may fragment, and persists until freed; it suits large, dynamic\u2011lifetime data structures and objects whose size or lifetime isn\u2019t known at compile time.

=== Model @3 (42 words) ===

Stack memory is automatic, function-scoped, and fast but small and fixed-size. Heap is manually managed (malloc/free), larger but limited by system RAM, prone to fragmentation. Prefer stack for small, short-lived data; use heap for large, long-lived, or dynamically sized data. (58 words)

/home/igor >

/home/igor > Sent to mshell (1952 bytes)

Received from GUI editor:

---

mshell C Language Edition

Pattern 7 Evaluator-Optimizer Loop

```c >task

#include <stdio.h>

int main(void) {

   printf("Write int gcd(int a, int b) using the Euclidean algorithm. "

        "main() tests gcd(48,18)=6, gcd(100,75)=25, gcd(7,13)=1, gcd(0,5)=5 "
```

```
        "and prints PASS or FAIL for each.\n");
    return 0;
}
```

<!--@loop max=3 until=verdict:ACCEPTED-->

<!--@1 <task >code

Write only C source code, no markdown fences, no explanation.

-->

```c <code
#include <stdio.h>
#include <stdlib.h>
int main(void) {
    printf("=== Generated Code ===\n");
    FILE *f = fopen(getenv("MSH_VAR_code"), "r"); int c;
    while ((c = fgetc(f)) != EOF) putchar(c);
    fclose(f); printf("\n");
    return 0;
}
```

<!--@2 <code >verdict

If this C GCD code is correct and all 4 tests would pass, reply with exactly one word: ACCEPTED

Otherwise reply with exactly one word: REJECTED

-->

```c <verdict
#include <stdio.h>
#include <stdlib.h>
```

```
int main(void) {
    FILE *f = fopen(getenv("MSH_VAR_verdict"), "r");
    char buf[64] = {0};
    fgets(buf, sizeof(buf), f); fclose(f);
    /* Strip newline */
    int len = 0; while (buf[len] && buf[len] != '\n' && buf[len] != '\r') len++;
    buf[len] = '\0';
    printf("=== Verdict: %s ===\n", buf);
    /* Last stdout line must be the verdict word for LOOP condition */
    printf("%s\n", buf);
    return 0;
}
```
<!--@end_loop-->
```c <code
#include <stdio.h>
#include <stdlib.h>
int main(void) {
    const char *src = getenv("MSH_VAR_code");
    char cmd[640];
    snprintf(cmd, sizeof(cmd), "cp -- \"%s\" /tmp/msh_p7_gcd.c", src);
    system(cmd);
    printf("=== Final Accepted Code ===\n");
    system("cat /tmp/msh_p7_gcd.c");
    printf("\n=== Compiling & Running ===\n");
    if (system("gcc -Wall -std=c11 -o /tmp/msh_p7_out /tmp/msh_p7_gcd.c 2>&1") == 0)
        system("/tmp/msh_p7_out");
```

```
    else
        printf("Compilation failed.\n");
    return 0;
}
```

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

Write int gcd(int a, int b) using the Euclidean algorithm. main() tests gcd(48,18)=6, gcd(100,75)=25, gcd(7,13)=1, gcd(0,5)=5 and prints PASS or FAIL for each.

[loop] Starting loop: max=3 until=verdict:ACCEPTED

```
#include <stdio.h>
int gcd(int a, int b) {
    if (b == 0) {
        return a;
    }
    return gcd(b, a % b);
}
int main() {
    printf("gcd(48,18)=6: %s\n", gcd(48, 18) == 6 ? "PASS" : "FAIL");
    printf("gcd(100,75)=25: %s\n", gcd(100, 75) == 25 ? "PASS" : "FAIL");
    printf("gcd(7,13)=1: %s\n", gcd(7, 13) == 1 ? "PASS" : "FAIL");
    printf("gcd(0,5)=5: %s\n", gcd(0, 5) == 5 ? "PASS" : "FAIL");
    return 0;
}
```

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

=== Generated Code ===

```
#include <stdio.h>
int gcd(int a, int b) {
   if (b == 0) {
      return a;
   }
   return gcd(b, a % b);
}
int main() {
   printf("gcd(48,18)=6: %s\n", gcd(48, 18) == 6 ? "PASS" : "FAIL");
   printf("gcd(100,75)=25: %s\n", gcd(100, 75) == 25 ? "PASS" : "FAIL");
   printf("gcd(7,13)=1: %s\n", gcd(7, 13) == 1 ? "PASS" : "FAIL");
   printf("gcd(0,5)=5: %s\n", gcd(0, 5) == 5 ? "PASS" : "FAIL");
   return 0;
}
```

ACCEPTED

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

[loop] Exiting loop after 1 iteration(s). reason: until condition met

=== Verdict: ACCEPTED ===

ACCEPTED

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

/home/igor > === Final Accepted Code ===

```
#include <stdio.h>
int gcd(int a, int b) {
```

```
    if (b == 0) {
        return a;
    }
    return gcd(b, a % b);
}
int main() {
    printf("gcd(48,18)=6: %s\n", gcd(48, 18) == 6 ? "PASS" : "FAIL");
    printf("gcd(100,75)=25: %s\n", gcd(100, 75) == 25 ? "PASS" : "FAIL");
    printf("gcd(7,13)=1: %s\n", gcd(7, 13) == 1 ? "PASS" : "FAIL");
    printf("gcd(0,5)=5: %s\n", gcd(0, 5) == 5 ? "PASS" : "FAIL");
    return 0;
}
=== Compiling & Running ===
gcd(48,18)=6: PASS
gcd(100,75)=25: PASS
gcd(7,13)=1: PASS
gcd(0,5)=5: PASS
/home/igor >
--------------------------------------------------------------------------------
/home/igor > Sent to mshell (2504 bytes)
Received from GUI editor:
---------------------------------------
# mshell C Language Edition
### Pattern 8  Multi-Stage C + Multi-Model Pipeline
```c >raw_data
#include <stdio.h>
int main(void) {
```
```

```
    double temps[] = {21.8,22.1,21.9,22.3,22.0,21.7,22.2,38.5,39.1,22.4,
                      22.1,21.8,22.0,22.3,21.9,22.1,22.4,21.6,22.2,22.0};
    FILE *tmp = fopen("/tmp/msh_p8_raw.txt", "w");
    for (int i = 0; i < 20; i++) {
        printf("t=%02d temp=%.1f\n", i, temps[i]);
        fprintf(tmp, "t=%02d temp=%.1f\n", i, temps[i]);
    }
    fclose(tmp);
    return 0;
}
```

```c <raw_data >anomaly_report
#include <stdio.h>
#include <math.h>
int main(void) {
    FILE *f = fopen("/tmp/msh_p8_raw.txt", "r");
    double t[32]; int n = 0, idx;
    while (n < 32 && fscanf(f, "t=%d temp=%lf\n", &idx, &t[n]) == 2) n++;
    fclose(f);
    double sum = 0;
    for (int i = 0; i < n; i++) sum += t[i];
    double mean = sum/n, var = 0;
    for (int i = 0; i < n; i++) var += (t[i]-mean)*(t[i]-mean);
    double sigma = sqrt(var/n);
    printf("mean=%.2f sigma=%.2f threshold=%.2f\nANOMALIES:\n",
           mean, sigma, mean+2*sigma);
    for (int i = 0; i < n; i++)
```

```
      if (fabs(t[i]-mean) > 2*sigma)
         printf("  t=%02d temp=%.1f dev=+%.1f\n", i, t[i], t[i]-mean);
   return 0;
}
```

<!--@1 <anomaly_report >diagnosis

The input is a temperature sensor anomaly report. Diagnose the physical cause. One paragraph, 60 words max.

-->

<!--@2 <diagnosis >alert_label

Compress this diagnosis into a 5-word alert label. Reply with exactly 5 words.

-->

```c <raw_data <anomaly_report <diagnosis <alert_label
#include <stdio.h>
#include <stdlib.h>
void ps(const char *title, const char *key) {
   printf("\n=== %s ===\n", title);
   FILE *f = fopen(getenv(key), "r"); int c;
   while ((c = fgetc(f)) != EOF) putchar(c);
   fclose(f);
}
void pf(const char *title, const char *path) {
   printf("\n=== %s ===\n", title);
   FILE *f = fopen(path, "r"); int c;
   while ((c = fgetc(f)) != EOF) putchar(c);
   fclose(f);
}
```

```
int main(void) {

printf("\u2554\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2557\n"
       "\u2551  SENSOR PIPELINE REPORT   \u2551\n"
"\u255a\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u2550\u255d");
   pf("RAW DATA",      "/tmp/msh_p8_raw.txt");
   ps("ANOMALY REPORT", "MSH_VAR_anomaly_report");
   ps("DIAGNOSIS",     "MSH_VAR_diagnosis");
   ps("ALERT LABEL",    "MSH_VAR_alert_label");
   printf("\n");
   return 0;
}
```

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

t=00 temp=21.8

t=01 temp=22.1

t=02 temp=21.9

t=03 temp=22.3

t=04 temp=22.0

t=05 temp=21.7

t=06 temp=22.2

t=07 temp=38.5

t=08 temp=39.1

t=09 temp=22.4

t=10 temp=22.1

t=11 temp=21.8

t=12 temp=22.0

t=13 temp=22.3

t=14 temp=21.9

t=15 temp=22.1

t=16 temp=22.4

t=17 temp=21.6

t=18 temp=22.2

t=19 temp=22.0

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

mean=23.72 sigma=5.03 threshold=33.78

ANOMALIES:

t=07 temp=38.5 dev=+14.8

t=08 temp=39.1 dev=+15.4

The temperature anomalies at t=07 and t=08 show a sudden ~15°C spike above the baseline, suggesting a localized heat source activation. This pattern indicates either equipment malfunction (overheating motor, electrical fault), external heat exposure (sunlight, heating system activation), or sensor placement near a heat-generating device that became active during the early morning hours.

Early-morning localized heat spike detected

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

/home/igor >
\u2554\u2550\u2557

\u2551 SENSOR PIPELINE REPORT   \u2551

\u255a\u2550\u255d

=== RAW DATA ===

t=00 temp=21.8

t=01 temp=22.1

t=02 temp=21.9

t=03 temp=22.3

t=04 temp=22.0

t=05 temp=21.7

t=06 temp=22.2

t=07 temp=38.5

t=08 temp=39.1

t=09 temp=22.4

t=10 temp=22.1

t=11 temp=21.8

t=12 temp=22.0

t=13 temp=22.3

t=14 temp=21.9

t=15 temp=22.1

t=16 temp=22.4

t=17 temp=21.6

t=18 temp=22.2

t=19 temp=22.0

=== ANOMALY REPORT ===

mean=23.72 sigma=5.03 threshold=33.78

ANOMALIES:

t=07 temp=38.5 dev=+14.8

t=08 temp=39.1 dev=+15.4

=== DIAGNOSIS ===

The temperature anomalies at t=07 and t=08 show a sudden ~15°C spike above the baseline, suggesting a localized heat source activation. This pattern indicates either equipment malfunction (overheating motor, electrical fault), external heat exposure (sunlight, heating system activation), or sensor placement near a heat-generating device that became active during the early morning hours.

=== ALERT LABEL ===

Early-morning localized heat spike detected

/home/igor >

/home/igor > Sent to mshell (2310 bytes)

Received from GUI editor:

---

mshell C Language Edition

Pattern 9 Routing

```c >input
#include <stdio.h>
int main(void) {
   printf("Compute the square root of 144 using an iterative numerical method.\n");
   return 0;
}
```

<!--@1 <input >route

Classify into exactly one word: CRYPTO, SORT, or MATH. Reply with only that word.

-->

```c <input if=route:CRYPTO
#include <stdio.h>
```

```
int main(void) {
   printf("=== CRYPTO: XOR cipher ===\n");
   const char *msg = "Hello, mshell!"; unsigned char key = 0xAB;
   printf("Original : %s\nEncrypted: ", msg);
   for (int i = 0; msg[i]; i++) printf("%02X ", (unsigned char)(msg[i]^key));
   printf("\nDecrypted: ");
   for (int i = 0; msg[i]; i++) printf("%c", (unsigned char)(msg[i]^key)^key);
   printf("\n");
   return 0;
}
```

```c <input if=route:SORT
#include <stdio.h>
void merge(int *a, int l, int m, int r) {
   int n1=m-l+1, n2=r-m, L[64], R[64];
   for (int i=0; i<n1; i++) L[i]=a[l+i];
   for (int j=0; j<n2; j++) R[j]=a[m+1+j];
   int i=0, j=0, k=l;
   while (i<n1&&j<n2) a[k++]=(L[i]<=R[j])?L[i++]:R[j++];
   while (i<n1) a[k++]=L[i++];
   while (j<n2) a[k++]=R[j++];
}
void mergesort(int *a, int l, int r) {
   if (l<r) { int m=(l+r)/2; mergesort(a,l,m); mergesort(a,m+1,r); merge(a,l,m,r); }
}
int main(void) {
   printf("=== SORT: merge sort ===\n");
```

```
    int arr[] = {64,34,25,12,22,11,90}; int n = 7;
    printf("Before: "); for (int i=0; i<n; i++) printf("%d ", arr[i]);
    mergesort(arr, 0, n-1);
    printf("\nAfter : "); for (int i=0; i<n; i++) printf("%d ", arr[i]); printf("\n");
    return 0;
}
```

```c <input if=route:MATH
#include <stdio.h>
#include <math.h>
int main(void) {
    printf("=== MATH: Newton-Raphson sqrt(144) ===\n");
    double x = 144.0, g = x/2.0;
    for (int i = 0; i < 10; i++) {
        double next = 0.5*(g+x/g);
        printf("  iter %2d: %.10f (err=%.2e)\n", i+1, next, fabs(next-sqrt(x)));
        if (fabs(next-g) < 1e-10) { g=next; break; }
        g = next;
    }
    printf("Result: %.6f\n", g);
    return 0;
}
```

```c <route
#include <stdio.h>
#include <stdlib.h>
int main(void) {
```

```
    printf("=== Classified as: ");
    FILE *f = fopen(getenv("MSH_VAR_route"), "r"); int c;
    while ((c = fgetc(f)) != EOF && c != '\n') putchar(c);
    fclose(f); printf(" ===\n");
    return 0;
}
```

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

Compute the square root of 144 using an iterative numerical method.

MATH

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

=== MATH: Newton-Raphson sqrt(144) ===

  iter 1: 37.0000000000 (err=2.50e+01)

  iter 2: 20.4459459459 (err=8.45e+00)

  iter 3: 13.7444534753 (err=1.74e+00)

  iter 4: 12.1107034897 (err=1.11e-01)

  iter 5: 12.0005059682 (err=5.06e-04)

  iter 6: 12.0000000107 (err=1.07e-08)

  iter 7: 12.0000000000 (err=0.00e+00)

  iter 8: 12.0000000000 (err=0.00e+00)

Result: 12.000000

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

/home/igor > === Classified as: MATH ===

/home/igor >

/home/igor > Sent to mshell (2415 bytes)

Received from GUI editor:

---

mshell C Language Edition

Pattern 10 Full Pipeline

```c >raw_data
#include <stdio.h>
#include <string.h>
int main(void) {
   int s[201]; memset(s, 1, sizeof(s)); s[0]=s[1]=0;
   for (int i=2; i<=200; i++)
      if (s[i]) for (int j=2*i; j<=200; j+=i) s[j]=0;
   FILE *tmp = fopen("/tmp/msh_p10_primes.txt", "w");
   int first = 1;
   for (int i=2; i<=200; i++) if (s[i]) {
      if (!first) { printf(" "); fprintf(tmp, " "); }
      printf("%d", i); fprintf(tmp, "%d", i); first = 0;
   }
   printf("\n"); fprintf(tmp, "\n");
   fclose(tmp);
   return 0;
}
```

```c <raw_data >stats
```

```
#include <stdio.h>
int main(void) {
   FILE *f = fopen("/tmp/msh_p10_primes.txt", "r");
   int p[64], n = 0;
   while (n < 64 && fscanf(f, "%d", &p[n]) == 1) n++;
   fclose(f);
   int twins = 0, mx_gap = 0;
   for (int i=1; i<n; i++) {
      int g = p[i]-p[i-1];
      if (g > mx_gap) mx_gap = g;
      if (g == 2) twins++;
   }
   printf("count=%d max_gap=%d twin_pairs=%d goldbach_4_to_20=verified\n",
          n, mx_gap, twins);
   return 0;
}
```

<!--@1 <stats >analysis
The input contains number-theoretic statistics about primes up to 200.
In one sentence, describe what these numbers reveal about prime distribution.
-->

<!--@2 <raw_data >poem
Write a 2-line poem about the beauty of prime numbers. Be poetic and precise.
-->

```c <analysis <poem
#include <stdio.h>
#include <stdlib.h>
```

```
int main(void) {
   printf("=== Analysis ===\n");
   FILE *f = fopen(getenv("MSH_VAR_analysis"), "r"); int c;
   while ((c = fgetc(f)) != EOF) putchar(c); fclose(f);
   printf("\n=== Poem ===\n");
   f = fopen(getenv("MSH_VAR_poem"), "r");
   while ((c = fgetc(f)) != EOF) putchar(c); fclose(f); printf("\n");
   return 0;
}
```

<!--@1 <analysis <poem >combined
Combine the mathematical analysis and the poem into one elegant sentence.
-->

```c <combined
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(void) {
   FILE *f = fopen(getenv("MSH_VAR_combined"), "r");
   char buf[1024] = {0}; fread(buf, 1, sizeof(buf)-1, f); fclose(f);
   int len = strlen(buf);
   if (len > 0 && buf[len-1] == '\n') buf[--len] = '\0';
   printf("\n");
   for (int i = 0; i < len+4; i++) putchar('*');
   printf("\n* %s *\n", buf);
   for (int i = 0; i < len+4; i++) putchar('*');
```

```
    printf("\n\n[mshell C Pipeline \u2014 Primes <= 200]\n");
    return 0;
}
```

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

count=46 max_gap=14 twin_pairs=15 goldbach_4_to_20=verified

These statistics reveal that among the 46 primes up to 200, there are 15 twin prime pairs (primes differing by 2), a maximum gap of 14 between consecutive primes, and verification of Goldbach's conjecture for even numbers 4-20, demonstrating the irregular but predictable clustering patterns in prime distribution.

Silent sentinels in the sea of integers, indivisible truths that quietly gleam;

Primes are the heartbeat of order in chaos, the simplest notes in infinity\u2019s theme.

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

=== Analysis ===

These statistics reveal that among the 46 primes up to 200, there are 15 twin prime pairs (primes differing by 2), a maximum gap of 14 between consecutive primes, and verification of Goldbach's conjecture for even numbers 4-20, demonstrating the irregular but predictable clustering patterns in prime distribution.

=== Poem ===

Silent sentinels in the sea of integers, indivisible truths that quietly gleam;

Primes are the heartbeat of order in chaos, the simplest notes in infinity\u2019s theme.

Among the 46 primes up to 200\u2014those silent sentinels with 15 twin pairs and gaps up to 14\u2014we glimpse the heartbeat of mathematical order, where indivisible truths create predictable clustering patterns in infinity's chaotic theme.

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

/home/igor >

****

* Among the 46 primes up to 200\u2014those silent sentinels with 15 twin pairs and gaps up to 14\u2014we glimpse the heartbeat of mathematical order, where indivisible truths create predictable clustering patterns in infinity's chaotic theme. *

****

[mshell C Pipeline Primes <= 200]

/home/igor >

/home/igor > Sent to mshell (1156 bytes)

Received from GUI editor:

---

mshell C Language Edition

Pattern 11 MShell Node with Multiple Models

```c >topic

#include <stdio.h>

int main(void) { printf("lock-free queue implementation in C using atomic operations\n"); return 0; }

```

```c >style

#include <stdio.h>

int main(void) { printf("expert systems programmer\n"); return 0; }

```

```mshell <topic <style >explanation

ollama1 "Explain $topic to a $style in one concise paragraph."

```

```mshell <explanation >keywords

ollama2 "Extract 5 technical keywords from: $explanation. Reply with exactly 5 words comma-separated."

```

```c <explanation <keywords

#include <stdio.h>

#include <stdlib.h>

int flen(const char *p) {

   FILE *f = fopen(p, "r"); fseek(f, 0, SEEK_END);

   int l = ftell(f); fclose(f); return l;

}

void pf(const char *p) {

   FILE *f = fopen(p, "r"); int c;

   while ((c = fgetc(f)) != EOF) putchar(c); fclose(f);

}

int main(void) {

   char *ep = getenv("MSH_VAR_explanation"), *kp = getenv("MSH_VAR_keywords");

   printf("=== Explanation (%d bytes) ===\n", flen(ep)); pf(ep);

   printf("\n=== Keywords (%d bytes) ===\n", flen(kp)); pf(kp); printf("\n");

   return 0;

}

```

 [DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

lock-free queue implementation in C using atomic operations

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

expert systems programmer

A lock-free queue uses atomic compare-and-swap (CAS) operations on head and tail pointers to enable concurrent enqueue/dequeue without blocking. The\u5178\u578bimplementation employs a linked list where enqueue atomically updates the tail's next pointer and advances tail, while dequeue uses CAS on head to claim nodes and handle ABA problems with hazard pointers or epochs. Memory ordering constraints (acquire/release semantics) ensure visibility of updates across threads, while handling edge cases like empty queues and memory reclamation requires careful sequencing of atomic operations to prevent race conditions and maintain queue invariants without traditional locking primitives.lock-free queue,compare-and-swap,hazard pointers,memory ordering,race conditions[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

/home/igor > === Explanation (682 bytes) ===

A lock-free queue uses atomic compare-and-swap (CAS) operations on head and tail pointers to enable concurrent enqueue/dequeue without blocking. The\u5178\u578bimplementation employs a linked list where enqueue atomically updates the tail's next pointer and advances tail, while dequeue uses CAS on head to claim nodes and handle ABA problems with hazard pointers or epochs. Memory ordering constraints (acquire/release semantics) ensure visibility of updates across threads, while handling edge cases like empty queues and memory reclamation requires careful sequencing of atomic operations to prevent race conditions and maintain queue invariants without traditional locking primitives.

=== Keywords (80 bytes) ===

lock-free queue,compare-and-swap,hazard pointers,memory ordering,race conditions

/home/igor >

/home/igor > Sent to mshell (1257 bytes)

Received from GUI editor:

---

mshell C Language Edition

Pattern 12 Async Parallel 3 Models + Await + Synthesis

```c >question
#include <stdio.h>
int main(void) {
   printf("In C, explain struct memory alignment and padding. "
        "Why does sizeof(struct{char a; int b;}) != 5 on most platforms? One paragraph.\n");
   return 0;
}
```

<!--@1 <question >ans1 async
Answer with a concrete C code example showing sizeof for different struct layouts. 60 words max.
-->

<!--@2 <question >ans2 async
Explain the theoretical reason: CPU alignment, cache lines, ABI rules. 60 words max.
-->

<!--@3 <question >ans3 async
List the 3 most common pitfalls C programmers hit because of struct padding. 60 words max.
-->

```c await=ans1,ans2,ans3
```

<!--@1 <ans1 <ans2 <ans3 >final
Synthesize these three perspectives into one complete paragraph useful for a C developer.
-->

```c <final
#include <stdio.h>
```

```
#include <stdlib.h>


int main(void) {
   printf("\n=== C STRUCT ALIGNMENT \u2014 TECH NOTE ===\n");
   FILE *f = fopen(getenv("MSH_VAR_final"), "r"); int c;
   while ((c = fgetc(f)) != EOF) putchar(c); fclose(f);
   printf("\n[Generated by mshell async 3-model pipeline]\n");
   return 0;
}
```

Part II Patterns 13: Advanced Node Types

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

In C, explain struct memory alignment and padding. Why does sizeof(struct{char a; int b;}) != 5 on most platforms? One paragraph.

[async llm] Launched PID 302586 \u2192 var=ans1 (model @1)

[async llm] Launched PID 302588 \u2192 var=ans2 (model @2)

[async llm] Launched PID 302590 \u2192 var=ans3 (model @3)

[async] await= barrier: waiting for vars: ans1,ans2,ans3

[async] Waiting for PID 302586 (var=ans1)...

[async] PID 302586 done (var=ans1)

Compilers add padding to align data members at their natural boundaries for efficient memory access. `char` (1 byte) followed by `int` (4 bytes) gets 3 padding bytes, making sizeof = 8, not 5.

```c
#include <stdio.h>

struct A {char a; int b;};       // sizeof = 8 (3 bytes padding)

struct B {int b; char a;};       // sizeof = 8 (3 bytes trailing padding)
```

struct C {char a; char c; int b;}; // sizeof = 8 (2 bytes padding)

int main() {

   printf("%zu %zu %zu\n", sizeof(struct A), sizeof(struct B), sizeof(struct C));

}

```

C compilers add padding between struct members to align data at natural boundaries for efficient memory access\u2014a `char` followed by an `int` requires 3 padding bytes, making sizeof = 8, not 5. This reflects the broader memory management principle: stack memory is automatically managed, function-scoped, fast, and contiguous but limited in size, making it ideal for small variables and recursion; heap memory is manually managed (malloc/free), larger but prone to fragmentation and slower access, suitable for dynamic data structures whose size or lifetime extends beyond function scope.

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

/home/igor >

=== C STRUCT ALIGNMENT \u2014 TECH NOTE ===

C compilers add padding between struct members to align data at natural boundaries for efficient memory access\u2014a `char` followed by an `int` requires 3 padding bytes, making sizeof = 8, not 5. This reflects the broader memory management principle: stack memory is automatically managed, function-scoped, fast, and contiguous but limited in size, making it ideal for small variables and recursion; heap memory is manually managed (malloc/free), larger but prone to fragmentation and slower access, suitable for dynamic data structures whose size or lifetime extends beyond function scope.

[Generated by mshell async 3-model pipeline]

---

/home/igor > Sent to mshell (2285 bytes)

Received from GUI editor:

---

# mshell C Language Edition

### Pattern 13  WHILE Loop: Fibonacci with LLM Commentary

```c >counter

````
#include <stdio.h>
int main(void) { printf("0\n"); return 0; }
```

```c >status
#include <stdio.h>
int main(void) { printf("running\n"); return 0; }
```
<!--@while status:running-->
```c <counter <status >counter >status
#include <stdio.h>
#include <stdlib.h>
void fib_fast(long long n, long long *fa, long long *fb) {
  if (n == 0) { *fa=0; *fb=1; return; }
  long long a, b; fib_fast(n/2, &a, &b);
  long long c = a*(2*b-a), d = a*a+b*b;
  if (n%2 == 0) { *fa=c; *fb=d; } else { *fa=d; *fb=c+d; }
}
int main(void) {
  FILE *f = fopen(getenv("MSH_VAR_counter"), "r");
  long long val; fscanf(f, "%lld", &val); fclose(f);
  val++;
  long long fa, fb; fib_fast(val, &fa, &fb);
  f = fopen(getenv("MSH_VAR_counter"), "w");
  fprintf(f, "%lld\n", val); fclose(f);
  f = fopen(getenv("MSH_VAR_status"), "w");
  fprintf(f, "%s\n", val >= 6 ? "done" : "running"); fclose(f);
  printf("counter=%lld fib(%lld)=%lld\n", val, val, fa);
````

```
    return 0;
}
```

<!--@1 <counter >comment

The input is a counter value. Write one interesting mathematical fact about the Fibonacci number at that position in the sequence.

-->

```c <comment
#include <stdio.h>
#include <stdlib.h>
int main(void) {
    long long n;
    FILE *f = fopen(getenv("MSH_VAR_counter"), "r"); fscanf(f, "%lld", &n); fclose(f);
    printf("[iter %lld] ", n);
    f = fopen(getenv("MSH_VAR_comment"), "r"); int c;
    while ((c = fgetc(f)) != EOF) putchar(c); fclose(f); printf("\n");
    return 0;
}
```

<!--@end_while-->

```c <counter
#include <stdio.h>
#include <stdlib.h>
void fib_fast(long long n, long long *fa, long long *fb) {
    if (n == 0) { *fa=0; *fb=1; return; }
    long long a, b; fib_fast(n/2, &a, &b);
    long long c = a*(2*b-a), d = a*a+b*b;
```

```
    if (n%2 == 0) { *fa=c; *fb=d; } else { *fa=d; *fb=c+d; }
}
int main(void) {
    FILE *f = fopen(getenv("MSH_VAR_counter"), "r");
    long long total; fscanf(f, "%lld", &total); fclose(f);
    printf("=== WHILE done. F(0) through F(%lld):\n", total);
    for (long long i = 0; i <= total; i++) {
        long long fa, fb; fib_fast(i, &fa, &fb);
        printf("  F(%lld)=%lld\n", i, fa);
    }
    return 0;
}
```

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

0

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

running

[while] iteration 1 \u2014 condition met, executing body

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

counter=1 fib(1)=1

The 1st Fibonacci number is 1, which holds the unique distinction of being the only positive integer that equals both its own factorial (1! = 1) and its own square ($1^2 = 1$), making it a fixed point for both the factorial and squaring functions.

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

[iter 1] The 1st Fibonacci number is 1, which holds the unique distinction of being the only positive integer that equals both its own factorial (1! = 1) and its own square ($1^2 = 1$), making it a fixed point for both the factorial and squaring functions.

[while] iteration 2 \u2014 condition met, executing body

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

counter=2 fib(2)=1

The 2nd Fibonacci number is 1, which is the only Fibonacci number that appears twice in the sequence (at positions 1 and 2), making it the unique repeated term that establishes the recursive pattern F(n) = F(n-1) + F(n-2) for all subsequent Fibonacci numbers.

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

[iter 2] The 2nd Fibonacci number is 1, which is the only Fibonacci number that appears twice in the sequence (at positions 1 and 2), making it the unique repeated term that establishes the recursive pattern F(n) = F(n-1) + F(n-2) for all subsequent Fibonacci numbers.

[while] iteration 3 \u2014 condition met, executing body

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

counter=3 fib(3)=2

The 3rd Fibonacci number is 2, which is the only even prime number and the smallest prime in the Fibonacci sequence, making it uniquely both a fundamental building block of all even numbers and an indivisible prime that cannot be factored further.

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

[iter 3] The 3rd Fibonacci number is 2, which is the only even prime number and the smallest prime in the Fibonacci sequence, making it uniquely both a fundamental building block of all even numbers and an indivisible prime that cannot be factored further.

[while] iteration 4 \u2014 condition met, executing body

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

counter=4 fib(4)=3

The 4th Fibonacci number is 3, which is the smallest odd prime and the only number that is simultaneously prime, triangular (T\u2082 = 1+2 = 3), and appears in both the Fibonacci sequence and as the number of sides in the first polygon (triangle), making it a rare intersection of multiple fundamental mathematical sequences.

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

[iter 4] The 4th Fibonacci number is 3, which is the smallest odd prime and the only number that is simultaneously prime, triangular (T\u2082 = 1+2 = 3), and appears in both the Fibonacci sequence and as the number of sides in the first polygon (triangle), making it a rare intersection of multiple fundamental mathematical sequences.

[while] iteration 5 \u2014 condition met, executing body

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

counter=5 fib(5)=5

The 5th Fibonacci number is 5, which remarkably equals its own position in the sequence, making it one of only two such "fixed points" (along with F\u2081 = 1 at position 1), and it's also the only Fibonacci number that is both prime and equal to the number of platonic solids in three-dimensional geometry.

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

[iter 5] The 5th Fibonacci number is 5, which remarkably equals its own position in the sequence, making it one of only two such "fixed points" (along with F\u2081 = 1 at position

1), and it's also the only Fibonacci number that is both prime and equal to the number of platonic solids in three-dimensional geometry.

[while] iteration 6 \u2014 condition met, executing body

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

counter=6 fib(6)=8

The 6th Fibonacci number is 8, which is a perfect cube ($2^3 = 8$) and the only Fibonacci number greater than 1 that is also a perfect power, making it uniquely positioned as both a term in the additive Fibonacci sequence and a multiplicative power of 2.

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

[iter 6] The 6th Fibonacci number is 8, which is a perfect cube ($2^3 = 8$) and the only Fibonacci number greater than 1 that is also a perfect power, making it uniquely positioned as both a term in the additive Fibonacci sequence and a multiplicative power of 2.

[while] condition 'status:running' not met, exiting after 6 iter

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

/home/igor > === WHILE done. F(0) through F(6):

F(0)=0

F(1)=1

F(2)=1

F(3)=2

F(4)=3

F(5)=5

F(6)=8

/home/igor > Sent to mshell (970 bytes)

Received from GUI editor:

---

mshell C Language Edition

Pattern 14 FOREACH: LLM Explains Each C Data Structure

```c >structures
#include <stdio.h>
int main(void) {
   printf("linked_list\nhash_table\nring_buffer\nred_black_tree\nskip_list\n");
   return 0;
}
```

<!--@foreach struct_name in structures-->
<!--@1 <struct_name >description
The input is a C data structure name.
In one sentence, describe its best use case in C systems programming.
-->

```c <description
#include <stdio.h>
#include <stdlib.h>
int main(void) {
   printf("--- ");
   FILE *f = fopen(getenv("MSH_VAR_struct_name"), "r"); int c;
   while ((c = fgetc(f)) != EOF && c != '\n') putchar(c); fclose(f); printf(" ---\n");
   f = fopen(getenv("MSH_VAR_description"), "r");
   while ((c = fgetc(f)) != EOF) putchar(c); fclose(f); printf("\n");
   return 0;
}
```

```

<!--@end_foreach-->

```c

#include <stdio.h>

int main(void) { printf("=== All C data structures described ===\n"); return 0; }

```

 [DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

linked_list

hash_table

ring_buffer

red_black_tree

skip_list

[foreach] iter 1: struct_name=linked_list

Linked lists are best used in C systems programming for implementing dynamic data structures like kernel task queues, memory allocators, or device driver lists where frequent insertion/deletion at arbitrary positions is required and the overhead of array shifting would be prohibitive.

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

--- linked_list ---

Linked lists are best used in C systems programming for implementing dynamic data structures like kernel task queues, memory allocators, or device driver lists where frequent insertion/deletion at arbitrary positions is required and the overhead of array shifting would be prohibitive.

[foreach] iter 2: struct_name=hash_table

Hash tables are best used in C systems programming for implementing fast symbol tables, caches, or lookup mechanisms in compilers, databases, and network routers where O(1) average-case insertion, deletion, and retrieval of key-value pairs is critical for performance.
```

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

--- hash_table ---

Hash tables are best used in C systems programming for implementing fast symbol tables, caches, or lookup mechanisms in compilers, databases, and network routers where O(1) average-case insertion, deletion, and retrieval of key-value pairs is critical for performance.

[foreach] iter 3: struct_name=ring_buffer

Ring buffers are best used in C systems programming for implementing fixed-size circular queues in real-time systems, device drivers, and inter-process communication where producers and consumers operate at different rates and memory allocation must be avoided in critical paths.

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

--- ring_buffer ---

Ring buffers are best used in C systems programming for implementing fixed-size circular queues in real-time systems, device drivers, and inter-process communication where producers and consumers operate at different rates and memory allocation must be avoided in critical paths.

[foreach] iter 4: struct_name=red_black_tree

Red-black trees are best used in C systems programming for implementing self-balancing ordered collections in kernel schedulers, memory managers, and database indexes where guaranteed O(log n) worst-case performance for insertions, deletions, and searches is essential for predictable real-time behavior.

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

--- red_black_tree ---

Red-black trees are best used in C systems programming for implementing self-balancing ordered collections in kernel schedulers, memory managers, and database indexes where guaranteed O(log n) worst-case performance for insertions, deletions, and searches is essential for predictable real-time behavior.

[foreach] iter 5: struct_name=skip_list

Skip lists are best used in C systems programming for implementing concurrent ordered dictionaries in multi-threaded systems like databases and distributed storage engines where lock-free or fine-grained locking is crucial, as they provide O(log n) operations with simpler implementation than balanced trees and better concurrency characteristics.

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

--- skip_list ---

Skip lists are best used in C systems programming for implementing concurrent ordered dictionaries in multi-threaded systems like databases and distributed storage engines where lock-free or fine-grained locking is crucial, as they provide O(log n) operations with simpler implementation than balanced trees and better concurrency characteristics.

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

/home/igor > === All C data structures described ===

/home/igor > Sent to mshell (1866 bytes)

Received from GUI editor:

---

mshell C Language Edition

Pattern 15 TRY/CATCH: Safe C Compilation with Error Capture

```c >snippet
#include <stdio.h>

int main(void) {
   printf("int add(int a, int b) { return a + b }  /* missing semicolon */\n"
        "int main(void) { printf(\"%%d\\n\", add(3,4)); return 0; }\n");
   return 0;
}
```

<!--@try-->

```c <snippet >result
#include <stdio.h>
#include <stdlib.h>
int main(void) {
   FILE *src = fopen("/tmp/msh_p15_try.c", "w");
   FILE *in  = fopen(getenv("MSH_VAR_snippet"), "r");
   int c;
   while ((c = fgetc(in)) != EOF) fputc(c, src);
   fclose(in); fclose(src);
   int ret = system("gcc -o /tmp/msh_p15_try_out /tmp/msh_p15_try.c 2>&1");
   if (ret != 0) { fprintf(stderr, "Compilation failed.\n"); return 1; }
   system("/tmp/msh_p15_try_out");
   return 0;
}
```

<!--@catch >error-->

```c
#include <stdio.h>
int main(void) {
   printf("=== Caught error: try_block_failed ===\n");
   printf("C compilation failed. Activating safe fallback.\n");
   return 0;
}
```

<!--@end_try-->

```c <snippet >safe_result
```

```
#include <stdio.h>
#include <stdlib.h>
int main(void) {
   FILE *src = fopen("/tmp/msh_p15_safe.c", "w");
   fprintf(src,
      "#include <stdio.h>\n"
      "int add(int a, int b){return a+b;}\n"
      "int main(void){printf(\"3+4=%%d\\n\",add(3,4));return 0;}\n");
   fclose(src);
   if (system("gcc -o /tmp/msh_p15_safe_out /tmp/msh_p15_safe.c 2>&1") != 0) {
      printf("Safe fallback failed.\n"); return 1;
   }
   printf("Safe fallback compiled OK.\n");
   system("/tmp/msh_p15_safe_out");
   return 0;
}
```

```c <safe_result
#include <stdio.h>
#include <stdlib.h>
int main(void) {
   printf("=== Safe result ===\n");
   FILE *f = fopen(getenv("MSH_VAR_safe_result"), "r"); int c;
   while ((c = fgetc(f)) != EOF) putchar(c); fclose(f);
   return 0;
}
```

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

int add(int a, int b) { return a + b } /* missing semicolon */

int main(void) { printf("%d\n", add(3,4)); return 0; }

[try] executing try block

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

Compilation failed.

/tmp/msh_p15_try.c: In function \u2018add\u2019:

/tmp/msh_p15_try.c:1:37: error: expected \u2018;\u2019 before \u2018}\u2019 token

   1 | int add(int a, int b) { return a + b } /* missing semicolon */

     | ^~

     | ;

/tmp/msh_p15_try.c: In function \u2018main\u2019:

/tmp/msh_p15_try.c:2:18: warning: implicit declaration of function \u2018printf\u2019 [-Wimplicit-function-declaration]

   2 | int main(void) { printf("%d\n", add(3,4)); return 0; }

     | ^~~~~~

/tmp/msh_p15_try.c:1:1: note: include \u2018<stdio.h>\u2019 or provide a declaration of \u2018printf\u2019

  +++ |+#include <stdio.h>

   1 | int add(int a, int b) { return a + b } /* missing semicolon */

/tmp/msh_p15_try.c:2:18: warning: incompatible implicit declaration of built-in function \u2018printf\u2019 [-Wbuiltin-declaration-mismatch]

   2 | int main(void) { printf("%d\n", add(3,4)); return 0; }

     | ^~~~~~

/tmp/msh_p15_try.c:2:18: note: include \u2018<stdio.h>\u2019 or provide a declaration of \u2018printf\u2019

[try] try block failed, executing catch block

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

=== Caught error: try_block_failed ===

C compilation failed. Activating safe fallback.

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

3+4=7

Safe fallback compiled OK.

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

/home/igor > === Safe result ===

3+4=7

Safe fallback compiled OK.

/home/igor > Sent to mshell (1302 bytes)

Received from GUI editor:

---

mshell C Language Edition

Pattern 16 SPLIT + MERGE: Divide-and-Conquer Matrix Analysis

```c >matrix_data

#include <stdio.h>

int main(void) {

    printf("1 2 3 4 5 6 7 8\n");
```

```
   printf("9 10 11 12 13 14 15 16\n");
   return 0;
}
```

<!--@split matrix_data into 2-->
<!--@1 <matrix_data_1 >analysis1 async
The input contains the first half of a 4x4 matrix (rows 1-2).
In one sentence, describe the numerical pattern.
-->
<!--@2 <matrix_data_2 >analysis2 async
The input contains the second half of a 4x4 matrix (rows 3-4).
In one sentence, describe the numerical pattern.
-->
```c await=analysis1,analysis2
```
<!--@merge-->
<!--@1 <analysis1 <analysis2 >combined
Combine these two partial matrix analyses into one unified description in two sentences.
-->
```c <combined
#include <stdio.h>
#include <stdlib.h>
void ps(const char *title, const char *key) {
   printf("=== %s ===\n", title);
   FILE *f = fopen(getenv(key), "r"); int c;
   while ((c = fgetc(f)) != EOF) putchar(c); fclose(f); printf("\n");
}
```

```
int main(void) {
   ps("Part 1",    "MSH_VAR_matrix_data_1");
   ps("Analysis 1", "MSH_VAR_analysis1");
   ps("Part 2",    "MSH_VAR_matrix_data_2");
   ps("Analysis 2", "MSH_VAR_analysis2");
   ps("Merged",    "MSH_VAR_combined");
   return 0;
}
```

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

1 2 3 4 5 6 7 8

9 10 11 12 13 14 15 16

[split] matrix_data_1 = 1 2 3 4 5 6 7 8

[split] matrix_data_2 = 9 10 11 12 13 14 15 16

[async llm] Launched PID 303631 \u2192 var=analysis1 (model @1)

[async llm] Launched PID 303632 \u2192 var=analysis2 (model @2)

[async] await= barrier: waiting for vars: analysis1,analysis2

[async] Waiting for PID 303631 (var=analysis1)...

[async] PID 303631 done (var=analysis1)

The numerical pattern is a simple sequential counting from 1 to 8, representing the first two rows of a 4x4 matrix filled in row-major order with consecutive integers.

The numerical pattern represents a 4×4 matrix filled with consecutive integers from 1 to 16 in row-major order. The first two rows contain values 1-8, while rows 3-4 continue the sequence with values 9-16, creating a systematic left-to-right, top-to-bottom enumeration of the entire matrix.

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

/home/igor > === Part 1 ===

1 2 3 4 5 6 7 8

=== Analysis 1 ===

The numerical pattern is a simple sequential counting from 1 to 8, representing the first two rows of a 4x4 matrix filled in row-major order with consecutive integers.

=== Part 2 ===

9 10 11 12 13 14 15 16

=== Analysis 2 ===

Rows 3\u20134 of the 4×4 matrix simply continue the sequence of consecutive integers from 9 to 16 in left-to-right, top-to-bottom order.

=== Merged ===

The numerical pattern represents a 4×4 matrix filled with consecutive integers from 1 to 16 in row-major order. The first two rows contain values 1-8, while rows 3-4 continue the sequence with values 9-16, creating a systematic left-to-right, top-to-bottom enumeration of the entire matrix.

/home/igor > Sent to mshell (1440 bytes)

Received from GUI editor:

---

mshell C Language Edition

Pattern 17 CONFIG Node: Parameterized C Performance Pipeline

```config

algorithm=quicksort

input_size=1000000

iterations=5

```

```c >algorithm
```

````
#include <stdio.h>
int main(void) { printf("quicksort\n"); return 0; }
```

```c >input_size
#include <stdio.h>
int main(void) { printf("1000000\n"); return 0; }
```
<!--@1 <algorithm <input_size >sketch
The first input is a sorting algorithm. The second is input size.
Write a concise C implementation sketch optimized for that size. Maximum 60 words.
-->
<!--@2 <sketch >complexity
The input is a C algorithm sketch.
Analyze Big-O time, space complexity, and cache behavior. 2 sentences max.
-->
```c <algorithm <input_size <sketch <complexity >report
#include <stdio.h>
#include <stdlib.h>
void pv(const char *l, const char *k) {
  printf("%-16s : ", l);
  FILE *f = fopen(getenv(k), "r"); int c;
  while ((c = fgetc(f)) != EOF && c != '\n') putchar(c); fclose(f); printf("\n");
}
void pb(const char *l, const char *k) {
  printf("\n[%s]\n", l);
  FILE *f = fopen(getenv(k), "r"); int c;
  while ((c = fgetc(f)) != EOF) putchar(c); fclose(f);
````

```
}
int main(void) {
    printf("=== C Performance Analysis Report ===\n");
    pv("Algorithm",  "MSH_VAR_algorithm");
    pv("Input size", "MSH_VAR_input_size");
    pb("Sketch",     "MSH_VAR_sketch");
    pb("Complexity", "MSH_VAR_complexity");
    printf("\n");
    return 0;
}
```

[config] algorithm=quicksort

[config] input_size=1000000

[config] iterations=5

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

quicksort

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

1000000

For 1M elements, use iterative quicksort with explicit stack to avoid recursion overhead and stack overflow. Implement median-of-three pivot selection, switch to insertion sort for subarrays <10 elements, and use tail-call optimization by always recursing on the smaller partition first. Pre-allocate stack array of size 64 (log\u2082(1M) * 2) for worst-case depth management.

Time complexity is O(n log n) on average and $O(n^2)$ in the worst case (though median-of-three plus insertion-sort cutoff make worst cases rare); insertion sort on tiny partitions improves constant factors and locality. Extra space is O(log n) for the explicit stack, and

cache behavior is generally good due to partitioning\u2019s mostly sequential access and small-subarray insertion sorts.

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

=== C Performance Analysis Report ===

Algorithm : quicksort

Input size : 1000000

[Sketch]

For 1M elements, use iterative quicksort with explicit stack to avoid recursion overhead and stack overflow. Implement median-of-three pivot selection, switch to insertion sort for subarrays <10 elements, and use tail-call optimization by always recursing on the smaller partition first. Pre-allocate stack array of size 64 (log\u2082(1M) * 2) for worst-case depth management.

[Complexity]

Time complexity is O(n log n) on average and $O(n^2)$ in the worst case (though median-of-three plus insertion-sort cutoff make worst cases rare); insertion sort on tiny partitions improves constant factors and locality. Extra space is O(log n) for the explicit stack, and cache behavior is generally good due to partitioning\u2019s mostly sequential access and small-subarray insertion sorts.

/home/igor > Sent to mshell (1272 bytes)

Received from GUI editor:

---

mshell C Language Edition

Pattern 18 FOREACH + Async LLM: Parallel C Algorithm Batch Analysis

```c >algorithms

#include <stdio.h>

int main(void) { printf("dijkstra\naes_encryption\nlru_cache\n"); return 0; }

```

<!--@foreach algo in algorithms-->

<!--@1 <algo >explanation async

The input is a C algorithm/data structure name.

Explain in one sentence for a beginner C programmer.

-->

<!--@2 <algo >usecase async

The input is a C algorithm/data structure name.

Name one real-world C system that uses it and why. One sentence.

-->

```c await=explanation,usecase
```

```c <explanation <usecase
#include <stdio.h>
#include <stdlib.h>
int main(void) {
   printf("=== ");
   FILE *f = fopen(getenv("MSH_VAR_algo"), "r"); int c;
   while ((c = fgetc(f)) != EOF && c != '\n') putchar(c); fclose(f); printf(" ===\n");
   printf("Explanation : ");
   f = fopen(getenv("MSH_VAR_explanation"), "r");
   while ((c = fgetc(f)) != EOF) putchar(c); fclose(f);
   printf("Real-world  : ");
   f = fopen(getenv("MSH_VAR_usecase"), "r");
   while ((c = fgetc(f)) != EOF) putchar(c); fclose(f); printf("\n");
   return 0;
}
```

<!--@end_foreach-->

```c

#include <stdio.h>

int main(void) { printf("=== All algorithms analyzed ===\n"); return 0; }

```

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

dijkstra

aes_encryption

lru_cache

[foreach] iter 1: algo=dijkstra

[async llm] Launched PID 303813 \u2192 var=explanation (model @1)

[async llm] Launched PID 303814 \u2192 var=usecase (model @2)

[async] await= barrier: waiting for vars: explanation,usecase

[async] Waiting for PID 303813 (var=explanation)...

[async] PID 303813 done (var=explanation)

Dijkstra's algorithm finds the shortest path from a starting point to all other points in a weighted graph by repeatedly selecting the unvisited node with the smallest known distance and updating its neighbors' distances, commonly implemented in C using arrays to store distances and a priority queue or simple linear search to find the minimum.

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

=== dijkstra ===

Explanation : Dijkstra's algorithm finds the shortest path from a starting point to all other points in a weighted graph by repeatedly selecting the unvisited node with the smallest known distance and updating its neighbors' distances, commonly implemented in C using arrays to store distances and a priority queue or simple linear search to find the minimum.

Real-world : The Quagga/FRR routing daemons (written in C) use Dijkstra\u2019s algorithm to compute shortest paths for OSPF, enabling efficient, loop-free IP routing in large networks.

[foreach] iter 2: algo=aes_encryption

[async llm] Launched PID 303851 \u2192 var=explanation (model @1)

[async llm] Launched PID 303852 \u2192 var=usecase (model @2)

[async] await= barrier: waiting for vars: explanation,usecase

[async] Waiting for PID 303851 (var=explanation)...

[async] PID 303851 done (var=explanation)

AES encryption is a symmetric cryptographic algorithm implemented in C using fixed-size 128-bit blocks and key expansion tables, where the same secret key is used to both encrypt plaintext into ciphertext and decrypt it back, making it essential for secure data storage and transmission in C applications.

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

=== aes_encryption ===

Explanation : AES encryption is a symmetric cryptographic algorithm implemented in C using fixed-size 128-bit blocks and key expansion tables, where the same secret key is used to both encrypt plaintext into ciphertext and decrypt it back, making it essential for secure data storage and transmission in C applications.

Real-world : OpenSSL (widely used in web servers and TLS libraries) uses AES encryption in C to securely protect data in transit between clients and servers.

[foreach] iter 3: algo=lru_cache

[async llm] Launched PID 303889 \u2192 var=explanation (model @1)

[async llm] Launched PID 303890 \u2192 var=usecase (model @2)

[async] await= barrier: waiting for vars: explanation,usecase

[async] Waiting for PID 303889 (var=explanation)...

[async] PID 303889 done (var=explanation)

An LRU cache is a fixed-size data structure that stores key-value pairs and automatically removes the least recently used item when full, typically implemented in C using a hash table for O(1) lookups combined with a doubly-linked list to track usage order.

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

=== lru_cache ===

Explanation : An LRU cache is a fixed-size data structure that stores key-value pairs and automatically removes the least recently used item when full, typically implemented in C using a hash table for O(1) lookups combined with a doubly-linked list to track usage order.

Real-world : The CPython interpreter (written in C) implements an LRU-style cache for method lookups and other hotspots to avoid repeated expensive computations and significantly speed up frequently executed operations.

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

/home/igor > === All algorithms analyzed ===

/home/igor > Sent to mshell (2352 bytes)

Received from GUI editor:

---

mshell C Language Edition

Pattern 19 WHILE Quality Gate: Generate C Code Until Score >= 8

```c >task
#include <stdio.h>
int main(void) {
    printf("Write: int is_palindrome(const char *s) "
           "that returns 1 if palindrome, 0 otherwise. "
           "Handle NULL, empty string, single char correctly.\n");
    return 0;
}
```

```c >iteration
```

````
#include <stdio.h>
int main(void) { printf("0\n"); return 0; }
```

```c >score
#include <stdio.h>
int main(void) { printf("0\n"); return 0; }
```

```c >code
#include <stdio.h>
int main(void) { printf("\n"); return 0; }
```

```c >status
#include <stdio.h>
int main(void) { printf("running\n"); return 0; }
```

<!--@while status:running-->
```c <iteration >iteration
#include <stdio.h>
#include <stdlib.h>
int main(void) {
  FILE *f = fopen(getenv("MSH_VAR_iteration"), "r");
  int v; fscanf(f, "%d", &v); fclose(f); v++;
  f = fopen(getenv("MSH_VAR_iteration"), "w");
  fprintf(f, "%d\n", v); fclose(f);
  printf("%d\n", v);
  return 0;
}
````

```
<!--@1 <task >code
Return ONLY the C function implementation. No main, no includes unless needed.
No markdown fences, no explanation.
-->
<!--@2 <code >score
The input is a C function. Rate it 1-10 for correctness and proper C idioms.
Reply with a SINGLE INTEGER ONLY. No words, no punctuation. Example: 9
-->
```c <iteration <score >status
#include <stdio.h>
#include <stdlib.h>
int main(void) {
   FILE *f; int iter, score;
   f = fopen(getenv("MSH_VAR_iteration"), "r"); fscanf(f, "%d", &iter); fclose(f);
   f = fopen(getenv("MSH_VAR_score"),     "r"); fscanf(f, "%d", &score); fclose(f);
   printf("[Iter %d] Score=%d\n", iter, score);
   /* Write status to file */
   f = fopen(getenv("MSH_VAR_status"), "w");
   fprintf(f, "%s\n", score >= 8 ? "done" : "running"); fclose(f);
   /* Last stdout line = status value, so WHILE reads it correctly */
   printf("%s\n", score >= 8 ? "done" : "running");
   return 0;
}
```
<!--@end_while-->
```c <code <score
```
```

```
#include <stdio.h>
#include <stdlib.h>
int main(void) {
   int sc;
   FILE *f = fopen(getenv("MSH_VAR_score"), "r"); fscanf(f, "%d", &sc); fclose(f);
   printf("=== Accepted C code (score=%d) ===\n", sc);
   f = fopen(getenv("MSH_VAR_code"), "r"); int c;
   while ((c = fgetc(f)) != EOF) putchar(c); fclose(f);
   return 0;
}
```

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

Write: int is_palindrome(const char *s) that returns 1 if palindrome, 0 otherwise. Handle NULL, empty string, single char correctly.

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

0

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

0

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

running

[while] iteration 1 \u2014 condition met, executing body

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

1

```
int is_palindrome(const char *s) {
    if (s == NULL) return 0;
    int len = 0;
    while (s[len]) len++;
    if (len <= 1) return 1;
    int left = 0, right = len - 1;
    while (left < right) {
        if (s[left] != s[right]) return 0;
        left++;
        right--;
    }
    return 1;
}
```

9

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

[Iter 1] Score=9

done

[while] condition 'status:running' not met, exiting after 1 iter

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

/home/igor > === Accepted C code (score=9) ===

```
int is_palindrome(const char *s) {
   if (s == NULL) return 0;
   int len = 0;
   while (s[len]) len++;
   if (len <= 1) return 1;
   int left = 0, right = len - 1;
   while (left < right) {
      if (s[left] != s[right]) return 0;
      left++;
      right--;
   }
   return 1;
}
```

/home/igor > Sent to mshell (1699 bytes)

Received from GUI editor:

---

mshell C Language Edition

Pattern 20 SPLIT + Async + MERGE: Map-Reduce C Source Analysis

```c >raw_text
#include <stdio.h>
int main(void) {
   printf("int fib(int n){if(n<=1)return n;return fib(n-1)+fib(n-2);}\n");
```

```
   printf("void swap(int*a,int*b){int t=*a;*a=*b;*b=t;}\n");
   printf("int bsearch_c(int*a,int n,int x){int l=0,r=n-1;while(l<=r){int m=(l+r)/2;if(a[m]==x)return m;else if(a[m]<x)l=m+1;else r=m-1;}return -1;}\n");
   return 0;
}
```

<!--@split raw_text into 3-->

<!--@1 <raw_text_1 >analysis1 async

The input is a complete C function on one line. In 3 words max, name the main C concept.

-->

<!--@1 <raw_text_2 >analysis2 async

The input is a complete C function on one line. In 3 words max, name the main C concept.

-->

<!--@1 <raw_text_3 >analysis3 async

The input is a complete C function on one line. In 3 words max, name the main C concept.

-->

```c await=analysis1,analysis2,analysis3
```

<!--@merge-->

<!--@2 <analysis1 <analysis2 <analysis3 >summary

The inputs are three C concept labels extracted from individual functions.

Synthesize into one sentence describing what the full C source implements.

-->

```c <analysis1 <analysis2 <analysis3 <summary
#include <stdio.h>
#include <stdlib.h>
void show(const char *l, const char *k) {
```

```
    printf("%s: ", l);
    FILE *f = fopen(getenv(k), "r"); int c;
    while ((c = fgetc(f)) != EOF) putchar(c); fclose(f); printf("\n");
}
int main(void) {
    printf("=== Map ===\n");
    show("Chunk 1", "MSH_VAR_analysis1");
    show("Chunk 2", "MSH_VAR_analysis2");
    show("Chunk 3", "MSH_VAR_analysis3");
    printf("\n=== Reduce ===\n");
    show("Summary", "MSH_VAR_summary");
    return 0;
}
```

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

int fib(int n){if(n<=1)return n;return fib(n-1)+fib(n-2);}

void swap(int*a,int*b){int t=*a;*a=*b;*b=t;}

int bsearch_c(int*a,int n,int x){int l=0,r=n-1;while(l<=r){int m=(l+r)/2;if(a[m]==x)return m;else if(a[m]<x)l=m+1;else r=m-1;}return -1;}

[split] raw_text_1 = int fib(int n){if(n<=1)return n;return fib(n-1)+fib(n-2);}

[split] raw_text_2 = void swap(int*a,int*b){int t=*a;*a=*b;*b=t;}

[split] raw_text_3 = int bsearch_c(int*a,int n,int x){int l=0,r=n-1;while(l<=r){int m=(l+r)/2;if(a[m]==x)return m;else if(a[m]<x)l=m+1;else r=m-1;}return -1;}

[async llm] Launched PID 304214 \u2192 var=analysis1 (model @1)

[async llm] Launched PID 304215 \u2192 var=analysis2 (model @1)

[async llm] Launched PID 304217 \u2192 var=analysis3 (model @1)

[async] await= barrier: waiting for vars: analysis1,analysis2,analysis3

[async] Waiting for PID 304214 (var=analysis1)...

[async] PID 304214 done (var=analysis1)

Recursive function calls.

The full C source implements a recursive binary search algorithm using pointer-based parameter passing.

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

/home/igor > === Map ===

Chunk 1: Recursive function calls.

Chunk 2: Pointer parameter passing.

Chunk 3: Binary search algorithm

=== Reduce ===

Summary: The full C source implements a recursive binary search algorithm using pointer-based parameter passing.

/home/igor > Sent to mshell (2296 bytes)

Received from GUI editor:

---

mshell C Language Edition

Pattern 21 TRY/CATCH + LOOP: Resilient C Code Retry

```c >task
#include <stdio.h>

int main(void) {
    printf("Write: char *my_strdup(const char *s) using malloc. "
        "Include main() testing with \"hello\" and \"world\", "
        "printing both, freeing memory.\n");
```

```
   return 0;
}
```

```c >last_error
#include <stdio.h>
int main(void) { printf("none\n"); return 0; }
```

<!--@loop max=3 until=last_error:ok-->
<!--@1 <task <last_error >code
Return ONLY C source code. No fences, no explanation.
If last_error is not "none", fix the error described.
-->
```c <code
#include <stdio.h>
#include <stdlib.h>
int main(void) {
   printf("=== Generated C code ===\n");
   FILE *f = fopen(getenv("MSH_VAR_code"), "r"); int c;
   while ((c = fgetc(f)) != EOF) putchar(c); fclose(f); printf("\n");
   return 0;
}
```

<!--@try-->
```c <code
#include <stdio.h>
#include <stdlib.h>
int main(void) {
```

```
    const char *src = getenv("MSH_VAR_code");
    char cmd[640];
    snprintf(cmd, sizeof(cmd), "cp -- \"%s\" /tmp/msh_p21_loop.c", src);
    system(cmd);
    if (system("gcc -Wall -std=c11 -o /tmp/msh_p21_out /tmp/msh_p21_loop.c 2>&1") != 0)
{
        /* Write error to last_error, print "fail" as last line for LOOP */
        FILE *f = fopen(getenv("MSH_VAR_last_error"), "w");
        fprintf(f, "compilation failed\n"); fclose(f);
        printf("fail\n");
        return 1;
    }
    system("/tmp/msh_p21_out");
    /* Write ok to last_error, print "ok" as last line for LOOP */
    FILE *f = fopen(getenv("MSH_VAR_last_error"), "w");
    fprintf(f, "ok\n"); fclose(f);
    printf("ok\n");
    return 0;
}
```

<!--@catch >last_error-->

```c
#include <stdio.h>
#include <stdlib.h>
int main(void) {
    printf("=== Error: try_block_failed ===\n");
    FILE *f = fopen(getenv("MSH_VAR_last_error"), "w");
```

```
    fprintf(f, "compilation failed\n"); fclose(f);
    /* Print "fail" as last line for LOOP */
    printf("fail\n");
    return 0;
}
```

<!--@end_try-->

<!--@end_loop-->

```c
#include <stdio.h>
#include <stdlib.h>
int main(void) {
    FILE *f = fopen(getenv("MSH_VAR_last_error"), "r");
    char buf[64] = {0}; fgets(buf, sizeof(buf), f); fclose(f);
    printf("=== Final status: %s ===\n", buf);
    return 0;
}
```

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

Write: char *my_strdup(const char *s) using malloc. Include main() testing with "hello" and "world", printing both, freeing memory.

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

none

[loop] Starting loop: max=3 until=last_error:ok

```
#include <stdio.h>
#include <stdlib.h>
char *my_strdup(const char *s) {
    if (s == NULL) return NULL;
    int len = 0;
    while (s[len]) len++;
    char *dup = malloc(len + 1);
    if (dup == NULL) return NULL;
    int i = 0;
    while (s[i]) {
        dup[i] = s[i];
        i++;
    }
    dup[i] = '\0';
    return dup;
}
int main() {
    char *str1 = my_strdup("hello");
    char *str2 = my_strdup("world");
    printf("%s\n", str1);
    printf("%s\n", str2);
    free(str1);
    free(str2);
    return 0;
}
[DEBUG compile_c] libs: [-lm]
```

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

[try] executing try block

=== Generated C code ===

```
#include <stdio.h>
#include <stdlib.h>
char *my_strdup(const char *s) {
    if (s == NULL) return NULL;
    int len = 0;
    while (s[len]) len++;
    char *dup = malloc(len + 1);
    if (dup == NULL) return NULL;
    int i = 0;
    while (s[i]) {
        dup[i] = s[i];
        i++;
    }
    dup[i] = '\0';
    return dup;
}
int main() {
    char *str1 = my_strdup("hello");
    char *str2 = my_strdup("world");
    printf("%s\n", str1);
    printf("%s\n", str2);
    free(str1);
    free(str2);
```

```
    return 0;
}
```

```
[DEBUG compile_c] libs: [-lm]
[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]
hello
world
ok
[try] try block succeeded
[loop] Exiting loop after 1 iteration(s). reason: until condition met
[DEBUG compile_c] libs: [-lm]
[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]
/home/igor > === Final status: ok
```

```
/home/igor > Sent to mshell (2096 bytes)
Received from GUI editor:
```

```
# mshell C Language Edition
```

Pattern 22 Multi-Variable Output: Structured C Function Extraction

```c >input
#include <stdio.h>
int main(void) {
    printf("Describe fopen(): its signature, return value, and error conditions.\n");
    return 0;
}
```

```
<!--@1 <input >raw_response
```

Respond in exactly this format (3 lines only):

SIGNATURE: the complete C function signature

RETURNS: what the return value means

ERRORS: comma-separated list of error conditions

-->

```c <raw_response >signature >returns >errors
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

char *extract(char *text, const char *label) {

  char *pos = strstr(text, label); if (!pos) return NULL;

  pos += strlen(label); while (*pos == ' ') pos++;

  char *end = strchr(pos, '\n'); if (end) *end = '\0';

  return pos;

}

int main(void) {

  FILE *f = fopen(getenv("MSH_VAR_raw_response"), "r");

  char buf[4096] = {0}; fread(buf, 1, sizeof(buf)-1, f); fclose(f);

  char b2[4096], b3[4096]; strcpy(b2, buf); strcpy(b3, buf);

  char *sig = extract(buf, "SIGNATURE:");

  char *ret = extract(b2,  "RETURNS:");

  char *err = extract(b3,  "ERRORS:");

  FILE *out;

  out = fopen(getenv("MSH_VAR_signature"), "w"); fprintf(out, "%s\n", sig?sig:"n/a");
fclose(out);

  out = fopen(getenv("MSH_VAR_returns"),   "w"); fprintf(out, "%s\n", ret?ret:"n/a");
fclose(out);
```

```
   out = fopen(getenv("MSH_VAR_errors"),    "w"); fprintf(out, "%s\n", err?err:"n/a");
fclose(out);

   printf("Signature : %s\nReturns   : %s\nErrors    : %s\n",

        sig?sig:"n/a", ret?ret:"n/a", err?err:"n/a");

   return 0;

}
```

<!--@2 <signature <errors >wrapper

The first input is a C function signature. The second is its error conditions.

Write a safe C wrapper that checks all errors and calls perror() on failure.

Return only C code, no fences.

-->

```c <wrapper
#include <stdio.h>

#include <stdlib.h>

int main(void) {

   printf("\n=== Safe C Wrapper ===\n");

   FILE *f = fopen(getenv("MSH_VAR_wrapper"), "r"); int c;

   while ((c = fgetc(f)) != EOF) putchar(c); fclose(f);

   return 0;

}
```

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

Describe fopen(): its signature, return value, and error conditions.

SIGNATURE: FILE *fopen(const char *filename, const char *mode)

RETURNS: pointer to FILE object on success, NULL on failure

ERRORS: file not found, permission denied, invalid mode, out of memory, too many open files, disk full

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

Signature : FILE *fopen(const char *filename, const char *mode)

Returns : pointer to FILE object on success, NULL on failure

Errors : file not found, permission denied, invalid mode, out of memory, too many open files, disk full

```
#include <stdio.h>

#include <errno.h>

#include <string.h>

FILE *safe_fopen(const char *filename, const char *mode) {

  if (filename == NULL || mode == NULL) {

    errno = EINVAL;

    perror("safe_fopen: invalid argument");

    return NULL;

  }

  FILE *fp = fopen(filename, mode);

  if (fp == NULL) {

    /* Possible errors include:

      - file not found (ENOENT)

      - permission denied (EACCES, EPERM)

      - invalid mode (EINVAL)

      - out of memory (ENOMEM)

      - too many open files (EMFILE, ENFILE)

      - disk full or I/O error (ENOSPC, EIO, etc.) */

    perror("safe_fopen");
```

```
    return NULL;
  }
  return fp;
}
```

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

/home/igor >

=== Safe C Wrapper ===

```
#include <stdio.h>
#include <errno.h>
#include <string.h>
FILE *safe_fopen(const char *filename, const char *mode) {
  if (filename == NULL || mode == NULL) {
    errno = EINVAL;
    perror("safe_fopen: invalid argument");
    return NULL;
  }
  FILE *fp = fopen(filename, mode);
  if (fp == NULL) {
    /* Possible errors include:
      - file not found (ENOENT)
      - permission denied (EACCES, EPERM)
      - invalid mode (EINVAL)
      - out of memory (ENOMEM)
      - too many open files (EMFILE, ENFILE)
      - disk full or I/O error (ENOSPC, EIO, etc.) */
```

```
        perror("safe_fopen");
        return NULL;
    }
    return fp;
}
```

/home/igor > Sent to mshell (2793 bytes)

Received from GUI editor:

---

mshell C Language Edition

Pattern 23 CONFIG + WHILE + Multi-Model: Adaptive C Code Pipeline

```config
problem=implement a stack using a fixed-size array in C
target_level=junior C programmer
quality_threshold=7
```

```c >problem
#include <stdio.h>
int main(void) { printf("implement a stack using a fixed-size array in C\n"); return 0; }
```

```c >target_level
#include <stdio.h>
int main(void) { printf("junior C programmer\n"); return 0; }
```

```c >iteration
#include <stdio.h>
```

```
int main(void) { printf("0\n"); return 0; }
```

```c >quality
#include <stdio.h>
int main(void) { printf("0\n"); return 0; }
```

```c >code
#include <stdio.h>
int main(void) { printf("\n"); return 0; }
```

```c >status
#include <stdio.h>
int main(void) { printf("running\n"); return 0; }
```

<!--@while status:running-->

```c <iteration >iteration
#include <stdio.h>
#include <stdlib.h>
int main(void) {
  FILE *f = fopen(getenv("MSH_VAR_iteration"), "r");
  int v; fscanf(f, "%d", &v); fclose(f); v++;
  f = fopen(getenv("MSH_VAR_iteration"), "w");
  fprintf(f, "%d\n", v); fclose(f);
  printf("%d\n", v);
  return 0;
}
```

<!--@1 <problem <target_level >code

Write clean C code for the problem, appropriate for the target level.

Return only C source code. No fences, no explanation.

-->

<!--@2 <code <target_level >quality

Rate the C code 1-10 for correctness and readability for the given level.

Reply with a SINGLE INTEGER ONLY. No words, no punctuation. Example: 8

-->

```c <iteration <quality >status
#include <stdio.h>
#include <stdlib.h>
int main(void) {
   FILE *f; int it, sc;
   f = fopen(getenv("MSH_VAR_iteration"), "r"); fscanf(f, "%d", &it); fclose(f);
   f = fopen(getenv("MSH_VAR_quality"),   "r"); fscanf(f, "%d", &sc); fclose(f);
   printf("[Iter %d] Quality: %d\n", it, sc);
   /* Write status to file */
   f = fopen(getenv("MSH_VAR_status"), "w");
   fprintf(f, "%s\n", sc >= 7 ? "done" : "running"); fclose(f);
   /* Last stdout line = status for WHILE condition check */
   printf("%s\n", sc >= 7 ? "done" : "running");
   return 0;
}
```

<!--@end_while-->

<!--@3 <code >documented_code

Add detailed inline comments explaining each significant line for a junior C programmer.

Return only the commented C source code, no fences.

-->

```c <documented_code <quality <iteration
#include <stdio.h>
#include <stdlib.h>
int main(void) {
   FILE *f; int q, it;
   f = fopen(getenv("MSH_VAR_quality"),   "r"); fscanf(f, "%d", &q);  fclose(f);
   f = fopen(getenv("MSH_VAR_iteration"), "r"); fscanf(f, "%d", &it); fclose(f);
   printf("=== Final Code (score=%d, iters=%d) ===\n", q, it);
   f = fopen(getenv("MSH_VAR_documented_code"), "r"); int c;
   while ((c = fgetc(f)) != EOF) putchar(c); fclose(f);
   return 0;
}
```

[config] problem=implement a stack using a fixed-size array in C

[config] target_level=junior C programmer

[config] quality_threshold=7

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

implement a stack using a fixed-size array in C

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

junior C programmer

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

0

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

0

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

running

[while] iteration 1 \u2014 condition met, executing body

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

1

```
#include <stdio.h>

#include <stdbool.h>

#define STACK_SIZE 100

typedef struct {

   int data[STACK_SIZE];

   int top;

} Stack;

void stack_init(Stack *s) {

   s->top = -1;

}
```

```
bool stack_is_empty(Stack *s) {
   return s->top == -1;
}
bool stack_is_full(Stack *s) {
   return s->top == STACK_SIZE - 1;
}
bool stack_push(Stack *s, int value) {
   if (stack_is_full(s)) {
      return false;
   }
   s->data[++s->top] = value;
   return true;
}
bool stack_pop(Stack *s, int *value) {
   if (stack_is_empty(s)) {
      return false;
   }
   *value = s->data[s->top--];
   return true;
}
bool stack_peek(Stack *s, int *value) {
   if (stack_is_empty(s)) {
      return false;
   }
   *value = s->data[s->top];
   return true;
}
```

```
int main() {
   Stack stack;
   stack_init(&stack);
   stack_push(&stack, 10);
   stack_push(&stack, 20);
   stack_push(&stack, 30);
   int value;
   while (stack_pop(&stack, &value)) {
      printf("%d\n", value);
   }
   return 0;
}
10
[DEBUG compile_c] libs: [-lm]
[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out
/tmp/mshell_validate_301141.c -lm 2>&1]
[Iter 1] Quality: 10
done
[while] condition 'status:running' not met, exiting after 1 iter
#include <stdio.h>  // Standard input/output functions (printf)
#include <stdbool.h>  // Provides bool, true, false for boolean types
// Define maximum size for the stack (100 elements)
#define STACK_SIZE 100
// Stack structure: holds data array and top pointer
typedef struct {
   int data[STACK_SIZE];  // Array to store stack elements (fixed size)
   int top;              // Tracks index of top element (-1 = empty)
```

```
} Stack;
// Initialize stack: sets top to -1 (empty state)
void stack_init(Stack *s) {
   s->top = -1;  // -1 indicates no elements are present
}
// Check if stack is empty: returns true when top is -1
bool stack_is_empty(Stack *s) {
   return s->top == -1;  // Empty condition
}
// Check if stack is full: returns true when top reaches last index
bool stack_is_full(Stack *s) {
   return s->top == STACK_SIZE - 1;  // Full condition
}
// Push value onto stack: returns false if full, true on success
bool stack_push(Stack *s, int value) {
   if (stack_is_full(s)) {  // Check space before adding
      return false;       // Fail if stack is full
   }
   s->data[++s->top] = value;  // Pre-increment top, then store value
   return true;  // Success
}
// Pop value from stack: stores top value in *value, returns false if empty
bool stack_pop(Stack *s, int *value) {
   if (stack_is_empty(s)) {  // Can't pop from empty stack
      return false;
   }
   *value = s->data[s->top--];  // Copy top value, then decrement top
```

```
    return true;  // Success
}
// Peek at top value without removing: stores top in *value, returns false if empty
bool stack_peek(Stack *s, int *value) {
    if (stack_is_empty(s)) {  // Empty stack has no top element
        return false;
    }
    *value = s->data[s->top];  // Copy current top element
    return true;  // Success
}
int main() {
    Stack stack;         // Declare stack variable (on main's stack frame)
    stack_init(&stack);   // Initialize the stack to empty state
    // Push three values (10,20,30) - order: 10 (bottom), 20, 30 (top)
    stack_push(&stack, 10);
    stack_push(&stack, 20);
    stack_push(&stack, 30);
    int value;  // Variable to store popped values
    // Pop and print all elements until stack is empty
    while (stack_pop(&stack, &value)) {  // Returns false when empty
        printf("%d\n", value);  // Prints 30, then 20, then 10
    }
    return 0;  // Exit program successfully
}
[DEBUG compile_c] libs: [-lm]
[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out
/tmp/mshell_validate_301141.c -lm 2>&1]
```

```
/home/igor > === Final Code (score=10, iters=1) ===
#include <stdio.h>  // Standard input/output functions (printf)
#include <stdbool.h>  // Provides bool, true, false for boolean types
// Define maximum size for the stack (100 elements)
#define STACK_SIZE 100
// Stack structure: holds data array and top pointer
typedef struct {
   int data[STACK_SIZE];  // Array to store stack elements (fixed size)
   int top;              // Tracks index of top element (-1 = empty)
} Stack;
// Initialize stack: sets top to -1 (empty state)
void stack_init(Stack *s) {
   s->top = -1;  // -1 indicates no elements are present
}
// Check if stack is empty: returns true when top is -1
bool stack_is_empty(Stack *s) {
   return s->top == -1;  // Empty condition
}
// Check if stack is full: returns true when top reaches last index
bool stack_is_full(Stack *s) {
   return s->top == STACK_SIZE - 1;  // Full condition
}
// Push value onto stack: returns false if full, true on success
bool stack_push(Stack *s, int value) {
   if (stack_is_full(s)) {  // Check space before adding
      return false;       // Fail if stack is full
   }
```

```
    s->data[++s->top] = value;  // Pre-increment top, then store value
    return true;  // Success
}
// Pop value from stack: stores top value in *value, returns false if empty
bool stack_pop(Stack *s, int *value) {
    if (stack_is_empty(s)) {  // Can't pop from empty stack
        return false;
    }
    *value = s->data[s->top--];  // Copy top value, then decrement top
    return true;  // Success
}
// Peek at top value without removing: stores top in *value, returns false if empty
bool stack_peek(Stack *s, int *value) {
    if (stack_is_empty(s)) {  // Empty stack has no top element
        return false;
    }
    *value = s->data[s->top];  // Copy current top element
    return true;  // Success
}
int main() {
    Stack stack;         // Declare stack variable (on main's stack frame)
    stack_init(&stack);   // Initialize the stack to empty state
    // Push three values (10,20,30) - order: 10 (bottom), 20, 30 (top)
    stack_push(&stack, 10);
    stack_push(&stack, 20);
    stack_push(&stack, 30);
    int value;  // Variable to store popped values
```

```
    // Pop and print all elements until stack is empty
    while (stack_pop(&stack, &value)) {  // Returns false when empty
        printf("%d\n", value);  // Prints 30, then 20, then 10
    }
    return 0;  // Exit program successfully
}
```

/home/igor > Sent to mshell (2110 bytes)

Received from GUI editor:

Pattern 24 FOREACH + TRY/CATCH: Fault-Tolerant C Batch Compilation

```c >sources
#include <stdio.h>
int main(void) {
    printf("/tmp/msh_src1.c\n/tmp/msh_broken.c\n/tmp/msh_src2.c\n");
    return 0;
}
```

<!--@foreach src in sources-->

<!--@try-->

```c <src >compiled
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(void) {
    char src_path[256] = {0};
    FILE *f = fopen(getenv("MSH_VAR_src"), "r");
```

```
    fgets(src_path, sizeof(src_path), f); fclose(f);
    src_path[strcspn(src_path, "\n")] = '\0';
    FILE *out = fopen(src_path, "w");
    if (strstr(src_path, "broken"))
        fprintf(out, "#include <stdio.h>\nint main(void) { printf(\"hello\");\n");
    else if (strstr(src_path, "src1"))
        fprintf(out, "#include <stdio.h>\nint main(void){int i;for(i=1;i<=5;i++)printf(\"%d
\",i*i);printf(\"\\n\");return 0;}\n");
    else
        fprintf(out, "#include <stdio.h>\n#include <math.h>\nint
main(void){printf(\"pi=%.6f\\n\",4.0*atan(1.0));return 0;}\n");
    fclose(out);
    char bin[256], cmd[512];
    strcpy(bin, src_path);
    char *dot = strrchr(bin, '.'); if (dot) *dot = '\0';
    snprintf(cmd, sizeof(cmd), "gcc -Wall -std=c11 -o %s %s -lm 2>&1", bin, src_path);
    if (system(cmd) != 0) return 1;
    system(bin);
    return 0;
}
```

<!--@1 <compiled >insight

The input contains output of a compiled C program.

In one sentence, describe what this C program computes.

-->

```c <insight
#include <stdio.h>
#include <stdlib.h>
```

```
int main(void) {
   printf("[OK] ");
   FILE *f = fopen(getenv("MSH_VAR_insight"), "r"); int c;
   while ((c = fgetc(f)) != EOF) putchar(c); fclose(f); printf("\n");
   return 0;
}
```

<!--@catch >compile_error-->

```c
#include <stdio.h>
int main(void) { printf("[ERR] Compilation failed: try_block_failed\n"); return 0; }
```

<!--@end_try-->

<!--@end_foreach-->

```c
#include <stdio.h>
int main(void) {
   printf("=== Batch complete. Errors isolated, pipeline never stopped. ===\n");
   return 0;
}
```

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

/tmp/msh_src1.c

/tmp/msh_broken.c

/tmp/msh_src2.c

[foreach] iter 1: src=/tmp/msh_src1.c

[try] executing try block

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

/tmp/mshell_code_301141.c: In function \u2018main\u2019:

/tmp/mshell_code_301141.c:15:91: warning: format \u2018%d\u2019 expects a matching \u2018int\u2019 argument [-Wformat=]

```
   15 |       fprintf(out, "#include <stdio.h>\nint main(void){int i;for(i=1;i<=5;i++)printf(\"%d \",i*i);printf(\"\\n\");return 0;}\n");
      |                                                                                          ~^
      |                                                                                           |
      |                                                                                           int
```

/tmp/mshell_code_301141.c:17:92: warning: format \u2018%f\u2019 expects a matching \u2018double\u2019 argument [-Wformat=]

```
   17 |       fprintf(out, "#include <stdio.h>\n#include <math.h>\nint main(void){printf(\"pi=%.6f\\n\",4.0*atan(1.0));return 0;}\n");
      |                                                                                         ~~~^
      |                                                                                            |
      |                                                                                            double
```

/tmp/mshell_code_301141.c:23:58: warning: \u2018%s\u2019 directive output may be truncated writing up to 255 bytes into a region of size between 234 and 489 [-Wformat-truncation=]

```
   23 |     snprintf(cmd, sizeof(cmd), "gcc -Wall -std=c11 -o %s %s -lm 2>&1", bin, src_path);
      |                                          ^~             ~~~~~~~~
```

/tmp/mshell_code_301141.c:23:5: note: \u2018snprintf\u2019 output between 33 and 543 bytes into a destination of size 512

```
   23 |     snprintf(cmd, sizeof(cmd), "gcc -Wall -std=c11 -o %s %s -lm 2>&1", bin, src_path);
      |     ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

/tmp/msh_src1.c: In function \u2018main\u2019:

/tmp/msh_src1.c:2:46: warning: too many arguments for format [-Wformat-extra-args]

```
   2 | int main(void){int i;for(i=1;i<=5;i++)printf("391745640 ",i*i);printf("\n");return 0;}
     |                                              ^~~~~~~~~~~~
```

391745640 391745640 391745640 391745640 391745640

This C program was intended to compute and print the squares of integers from 1 to 5, but due to a printf format error, it prints the literal string "391745640" five times instead of the calculated values.

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

[OK] This C program was intended to compute and print the squares of integers from 1 to 5, but due to a printf format error, it prints the literal string "391745640" five times instead of the calculated values.

[try] try block succeeded

[foreach] iter 2: src=/tmp/msh_broken.c

[try] executing try block

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

/tmp/mshell_code_301141.c: In function \u2018main\u2019:

/tmp/mshell_code_301141.c:15:91: warning: format \u2018%d\u2019 expects a matching \u2018int\u2019 argument [-Wformat=]

```
  15 |       fprintf(out, "#include <stdio.h>\nint main(void){int i;for(i=1;i<=5;i++)printf(\"%d \",i*i);printf(\"\\n\");return 0;}\n");
     |                                                                                          ~^
     |                                                                                           |
     |                                                                                           int
```

/tmp/mshell_code_301141.c:17:92: warning: format \u2018%f\u2019 expects a matching \u2018double\u2019 argument [-Wformat=]

```
  17 |       fprintf(out, "#include <stdio.h>\n#include <math.h>\nint main(void){printf(\"pi=%.6f\\n\",4.0*atan(1.0));return 0;}\n");
```

```
    |                                                         ~~~^
    |                                                           |
    |                                                           double
```

/tmp/mshell_code_301141.c:23:58: warning: \u2018%s\u2019 directive output may be truncated writing up to 255 bytes into a region of size between 234 and 489 [-Wformat-truncation=]

```
   23 |     snprintf(cmd, sizeof(cmd), "gcc -Wall -std=c11 -o %s %s -lm 2>&1", bin, src_path);
      |                                          ^~             ~~~~~~~~
```

/tmp/mshell_code_301141.c:23:5: note: \u2018snprintf\u2019 output between 33 and 543 bytes into a destination of size 512

```
   23 |     snprintf(cmd, sizeof(cmd), "gcc -Wall -std=c11 -o %s %s -lm 2>&1", bin, src_path);
      |     ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

/tmp/msh_broken.c: In function \u2018main\u2019:

/tmp/msh_broken.c:2:1: error: expected declaration or statement at end of input

```
    2 | int main(void) { printf("hello");
      | ^~~
```

This C program fails to compile due to a missing closing brace, but if fixed, it would simply print the string "hello" to standard output.

[try] try block failed, executing catch block

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

[ERR] Compilation failed: try_block_failed

[foreach] iter 3: src=/tmp/msh_src2.c

[try] executing try block

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

/tmp/mshell_code_301141.c: In function \u2018main\u2019:

/tmp/mshell_code_301141.c:15:91: warning: format \u2018%d\u2019 expects a matching \u2018int\u2019 argument [-Wformat=]

```
  15 |       fprintf(out, "#include <stdio.h>\nint main(void){int i;for(i=1;i<=5;i++)printf(\"%d \",i*i);printf(\"\\n\");return 0;}\n");
     |                                                                                          ~^
     |                                                                                           |
     |                                                                                           int
```

/tmp/mshell_code_301141.c:17:92: warning: format \u2018%f\u2019 expects a matching \u2018double\u2019 argument [-Wformat=]

```
  17 |       fprintf(out, "#include <stdio.h>\n#include <math.h>\nint main(void){printf(\"pi=%.6f\\n\",4.0*atan(1.0));return 0;}\n");
     |                                                                                         ~~~^
     |                                                                                            |
     |                                                                                            double
```

/tmp/mshell_code_301141.c:23:58: warning: \u2018%s\u2019 directive output may be truncated writing up to 255 bytes into a region of size between 234 and 489 [-Wformat-truncation=]

```
  23 |     snprintf(cmd, sizeof(cmd), "gcc -Wall -std=c11 -o %s %s -lm 2>&1", bin, src_path);
     |                                          ^~            ~~~~~~~~
```

/tmp/mshell_code_301141.c:23:5: note: \u2018snprintf\u2019 output between 33 and 543 bytes into a destination of size 512

```
  23 |     snprintf(cmd, sizeof(cmd), "gcc -Wall -std=c11 -o %s %s -lm 2>&1", bin, src_path);
     |     ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

/tmp/msh_src2.c: In function \u2018main\u2019:

/tmp/msh_src2.c:3:23: warning: too many arguments for format [-Wformat-extra-args]

```
   3 | int main(void){printf("pi=0.000000\n",4.0*atan(1.0));return 0;}
     |                       ^~~~~~~~~~~~~~~~
```

pi=0.000000

This C program computes the value of pi using the mathematical identity 4*atan(1.0), but due to a printf format error (missing %f specifier), it prints "pi=0.000000" instead of displaying the calculated pi value.

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

[OK] This C program computes the value of pi using the mathematical identity 4*atan(1.0), but due to a printf format error (missing %f specifier), it prints "pi=0.000000" instead of displaying the calculated pi value.

[try] try block succeeded

[DEBUG compile_c] libs: [-lm]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_301141.out /tmp/mshell_validate_301141.c -lm 2>&1]

/home/igor > === Batch complete. Errors isolated, pipeline never stopped. ===

/home/igor >

References:

mshell Workflow Patterns — C Language Edition (Patterns 1–24) — Complete Reference Guide (P1–P24). Created by Igor Lukyanov, Art2Dec SoftLab Based on the original mshell Workflow Patterns Reference Guides Part I & Part II

Resources: - Common examples Part I (P1–P12): https://www.appservgrid.com/paw92/index.php/2026/02/26/mshell-workflow-patterns-reference-guide-part-i-p1-p13/

Resources: - Common examples Part II (P13–P24): https://www.appservgrid.com/paw92/index.php/2026/03/11/mshell-workflow-patterns-reference-guide-part-ii-p13-p24/

- mshell v1.4.1 cheatsheet: https://www.appservgrid.com/paw92/index.php/2026/02/04/mshell-v-1-4-1-cheatsheet-january-26th-2026/