

```
1 0 (Automated Data Generation with C){
2 //Written from 14:50 20/03/2025
3 //cLLM.c : C Large Language model
4
5 /*
6
7 setup(){
8 Here's a step-by-step guide to install MSYS2 on your system:
9
10 1. Download MSYS2: Go to the [official MSYS2 website](https://www.msys2.org/) and download the installer.
11
12 2. Run the Installer: Execute the downloaded installer and choose a suitable installation directory (e.g., `C:\msys64`). Avoid paths with spaces.
13
14 3. Update MSYS2: Open the MSYS2 MinGW 64-bit shell and run the following commands to update the package database and core system:
15 ```bash
16 pacman -Syu
17 pacman -Su
18 ```
19
20 4. Install the GCC Toolchain: Run the following command to install the necessary development tools:
21 ```bash
22 pacman -S --needed base-devel mingw-w64-x86_64-toolchain
23 ```
24
25 5. Add to PATH: Add the `C:\msys64\mingw64\bin` directory to your system's PATH environment variable. This allows you to use GCC from PowerShell. You can do this by running the
following command in PowerShell:
26 ```powershell
27 $env:Path += ";C:\msys64\mingw64\bin"
28 ```
29
30 6. Verify Installation: Open PowerShell and run `gcc --version`. You should see the GCC version information.
31
32 For more detailed instructions, you can refer to the [MSYS2 installation guide](https://www.msys2.org/wiki/MSYS2-installation/).
33
34 Let me know if you need any further assistance!
35 }
36
37 -:: Prompt Engineered by Dominic Alexander Cooper at 19:35 09/03/2025
38 -:: cd C:/Users/dacoo/Documents/C
39 -:: gcc -o 1 1.c
40 -:: .\1.exe
41 */
42
43 /*
44
45 setup(){
46 Here's a step-by-step guide to install MSYS2 on your system:
47
48 1. Download MSYS2: Go to the [official MSYS2 website](https://www.msys2.org/) and download the installer.
49
50 2. Run the Installer: Execute the downloaded installer and choose a suitable installation directory (e.g., `C:\msys64`). Avoid paths with spaces.
51
52 3. Update MSYS2: Open the MSYS2 MinGW 64-bit shell and run the following commands to update the package database and core system:
53 ```bash
54 pacman -Syu
55 pacman -Su
56 ```
57
58 4. Install the GCC Toolchain: Run the following command to install the necessary development tools:
59 ```bash
60 pacman -S --needed base-devel mingw-w64-x86_64-toolchain
61 ```
62
63 5. Add to PATH: Add the `C:\msys64\mingw64\bin` directory to your system's PATH environment variable. This allows you to use GCC from PowerShell. You can do this by running the
following command in PowerShell:
64 ```powershell
65 $env:Path += ";C:\msys64\mingw64\bin"
66 ```
67
68 6. Verify Installation: Open PowerShell and run `gcc --version`. You should see the GCC version information.
69
70 For more detailed instructions, you can refer to the [MSYS2 installation guide](https://www.msys2.org/wiki/MSYS2-installation/).
71
```

```

72 Let me know if you need any further assistance!
73 }
74
75 -:: Prompt Engineered by Dominic Alexander Cooper at 22:23 09/03/2025
76 -:: cd C:/Users/dacoo/Documents/C
77 -:: gcc -o CLLM cLLM.c
78 -:: .\CLLM.exe
79 */
80
81 #include <stdio.h>
82 #include <stdlib.h>
83 #include <string.h>
84 #include <math.h>
85
86 int main(){
87
88     FILE *p; p = fopen("fs.txt", "w");
89     char alphabet[] =
90     {'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z','A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','
91     'T','U','V','W','X','Y','Z','0','1','2','3','4','5','6','7','8','9','\\','|',' ','<','.', '>','/','?',';',':','\','@','#','~','[','{','}','}','\`','!','"','$','%','^','&','*','(',')'
92     ,'-','_','=','+'};
93     int k = strlen(alphabet) - 1;
94     int cardinality = k + 1;
95     printf("alphabet cardinality is : %d\n", (k + 1));
96     int noc;
97     scanf("%d", &noc);
98     int n = noc;
99     printf("Per file character cardinality is : %d\n", n);
100     int row, cell, col, rdiv, id;
101     id = 0;
102     int nbr_comb = pow(cardinality, n);
103
104     for(row = 0; row < nbr_comb; row++){
105
106         id++; fprintf(p, "%d\t() {\n\t", id);
107
108         for(col = n - 1; col >= 0; col--){
109
110             rdiv = pow(cardinality, col);
111             cell = (row/rdiv) % cardinality;
112             fprintf(p, "%c", alphabet[cell]);
113
114         }
115
116         fprintf(p, "\n} []\n\n");
117     }
118
119     fclose(p);
120     return 0;
121 }[
122
123 AI Prompts - RELATIONAL OBJECTS, DEFINITIONS, IMPLEMENTATIONS
124
125 1 Create a TAB indented, and integer numbered list of mathematical/ computer instruction set actions that the string '<lowercase string>' could denote.
126
127 [2000] points (Claude-3.7-Sonnet)
128
129 2 Formally define each of the following mathematical/ computer instruction set actions:
130
131 <list of mathematical/ computer instruction set actions>
132
133 Refrain from using abbreviations, placeholder symbols, and symbols. Use technical terms and technical language to construct each definition.
134
135 [40000] points (Claude-3.7-Sonnet)
136
137 3 Create a TAB indented, and integer numbered list of mathematical/ computer instruction set actions that the string '<capitalized string>' could denote (Where the denotation cannot
138 be made using the lowercase version of the letter).
139
140 [2000] points (Claude-3.7-Sonnet)

```

```
141 4   Create a 10 to 4 C Programming Language implementations (With expository comments), where all 20 of the following mathematical/ computer instruction set actions are used at least
142 once in the context of the provided definitions:
143
144 <list of mathematical/ computer instruction set actions>
145
146 <DEFINITIONS of the list of mathematical/ computer instruction set actions>
147
148 5   writing sequences as 6 XOR 7
149
150 6   1 2 4
151
152 7   3 2 4
153
154 ]
155
156 1   (1 2){
157     a
158   }[
159
160 1   Area (in geometric calculations)
161 2   Accumulator (in CPU registers)
162 3   Addition operation (in arithmetic)
163 4   Address (in memory references)
164 5   Acceleration (in physics calculations)
165 6   Array (in data structures)
166 7   ASCII value (in character encoding)
167 8   Argument (in function calls)
168 9   Assignment operator (in programming)
169 10  Alpha channel (in graphics processing)
170 11  Attribute (in markup languages or databases)
171 12  Atomic operation (in concurrent programming)
172 13  Ampere (unit of electrical current in calculations)
173 14  Assembly instruction (in low-level programming)
174 15  AND operator (in Boolean logic)
175 16  Access modifier (in object-oriented programming)
176 17  Angle (in trigonometric functions)
177 18  Append operation (for collections)
178 19  Average function (in statistical operations)
179 20  Algorithm complexity constant (in Big O notation)
180
181 DEFINITIONS
182
183 1. Area (in geometric calculations): A quantitative measure of the two-dimensional extent of a closed region in a plane, typically expressed in square units. It represents the amount
184 of space enclosed within the boundaries of a geometric figure.
185
186 2. Accumulator (in CPU registers): A dedicated register in a central processing unit that temporarily stores the results of arithmetic and logical operations. It serves as both a
187 source operand and destination for computational results in the instruction execution cycle.
188
189 3. Addition operation (in arithmetic): A binary mathematical operation that combines two numerical values to produce their sum. It is a fundamental arithmetic operation characterized
190 by commutativity and associativity properties.
191
192 4. Address (in memory references): A numerical identifier that specifies a unique location in computer memory where data or instructions are stored. It enables direct access to
193 specific memory cells within the memory address space.
194
195 5. Acceleration (in physics calculations): The rate of change of velocity with respect to time. It is a vector quantity that measures how quickly an object's velocity changes,
196 expressed in units of distance per time squared.
197
198 6. Array (in data structures): A contiguous collection of elements of the same data type, stored in sequential memory locations and accessed via numerical indices. It provides
199 constant-time access to individual elements based on their position.
200
201 7. ASCII value (in character encoding): A numerical representation of a character according to the American Standard Code for Information Interchange encoding scheme. Each character is
202 assigned a unique integer value between 0 and 127.
```

```
203 11. Attribute (in markup languages or databases): A named property or characteristic associated with an element or entity that provides additional information about it. It consists of
204 a name-value pair that qualifies or modifies the element or entity.
205 12. Atomic operation (in concurrent programming): An indivisible and uninterruptible operation that appears to occur instantaneously from the perspective of concurrent processes. It
206 completes entirely or not at all, with no observable intermediate states.
207 13. Ampere (unit of electrical current in calculations): The International System of Units base unit for electric current, defined as the constant current which, if maintained in two
208 straight parallel conductors of infinite length and negligible cross-section, would produce a force of  $2 \times 10^{-7}$  newtons per meter between the conductors.
209 14. Assembly instruction (in low-level programming): A human-readable mnemonic representation of a machine code instruction that corresponds directly to a specific operation the
210 processor can execute. It represents the fundamental unit of work in assembly language programming.
211 15. AND operator (in Boolean logic): A binary logical operation that produces a true result only when both of its operands are true. It implements logical conjunction, returning true
212 if and only if all inputs are true.
213 16. Access modifier (in object-oriented programming): A keyword that specifies the visibility and accessibility scope of a class, method, or field within a program. It controls which
214 other classes or components can interact with the designated element.
215 17. Angle (in trigonometric functions): A measure of rotation or the amount of turn between two rays or lines with a common endpoint. It serves as the input to trigonometric functions
216 and is typically measured in radians or degrees.
217 18. Append operation (for collections): A procedure that adds an element to the end of a sequence or collection data structure, increasing its length by one. It maintains the existing
218 order of elements while extending the collection.
219 19. Average function (in statistical operations): A calculation that determines the central tendency of a set of numerical values by summing all values and dividing by the count of
220 values. Also known as the arithmetic mean, it represents a typical value from the dataset.
221 20. Algorithm complexity constant (in Big O notation): A multiplicative factor in the time or space complexity expression that remains fixed regardless of input size but affects the
222 overall performance. While typically omitted in asymptotic analysis, it becomes significant when comparing algorithms with identical growth rates.
223 IMPLEMENTATIONS
224
225 /*
226  * File: comprehensive_demo.c
227  * Description: A comprehensive demonstration of 20 mathematical/computer instruction concepts
228  * This program demonstrates various computing concepts through practical implementations
229  */
230
231 #include <stdio.h>
232 #include <stdlib.h>
233 #include <math.h>
234 #include <string.h>
235 #include <stdint.h>
236 #include <stdatomic.h>
237 #include <pthread.h>
238
239 /* Access modifier simulation through struct encapsulation */
240 typedef struct {
241     /* Private attributes (simulating private access modifier) */
242     double _voltage;
243     double _resistance;
244
245     /* Public attributes (conceptually accessible to all) */
246     double current; /* Measured in Amperes */
247 } Circuit;
248
249 /* Function to calculate rectangle area - demonstrates Area in geometric calculations */
250 double calculateRectangleArea(double length, double width) {
251     /* Area calculation as length multiplied by width */
252     return length * width;
253 }
254
255 /* Function demonstrating acceleration calculation in physics */
256 double calculateAcceleration(double initialVelocity, double finalVelocity, double time) {
257     /* Acceleration is the rate of change of velocity with respect to time */
258     return (finalVelocity - initialVelocity) / time;
259 }
260
261 /* Function implementing average calculation - demonstrates statistical operations */
262 double calculateAverage(int values[], int count) {
263     double sum = 0.0;
264     /* Addition operation used in accumulating values */
265     for (int i = 0; i < count; i++) {
```

```

266         sum += values[i];
267     }
268     /* Returning arithmetic mean by dividing sum by count */
269     return sum / count;
270 }
271
272 /* Function to append a value to an array - demonstrates append operation for collections */
273 int* appendToArray(int array[], int* size, int value) {
274     /* Allocate new memory with increased size */
275     int* newArray = (int*)malloc((*size + 1) * sizeof(int));
276
277     /* Copy existing elements */
278     for (int i = 0; i < *size; i++) {
279         newArray[i] = array[i];
280     }
281
282     /* Append the new value to the end */
283     newArray[*size] = value;
284
285     /* Update size and return new array */
286     (*size)++;
287     return newArray;
288 }
289
290 /* Function that uses angle in trigonometric operations */
291 double calculateSineWave(double amplitude, double frequency, double angle) {
292     /* Using angle as input to sine function */
293     return amplitude * sin(angle * frequency);
294 }
295
296 /* Atomic counter for thread-safe operations */
297 atomic_int sharedCounter = 0;
298
299 /* Thread function demonstrating atomic operations in concurrent programming */
300 void* incrementCounter(void* arg) {
301     for (int i = 0; i < 1000; i++) {
302         /* Atomic increment operation - indivisible and uninterruptible */
303         atomic_fetch_add(&sharedCounter, 1);
304     }
305     return NULL;
306 }
307
308 /* Calculates current in a circuit using Ohm's Law - demonstrates Ampere unit */
309 double calculateCurrentInAmperes(double voltage, double resistance) {
310     /* Current (Amperes) = Voltage / Resistance */
311     return voltage / resistance;
312 }
313
314 /* Getter function for voltage - demonstrates simulated access modifier pattern */
315 double getVoltage(Circuit* circuit) {
316     return circuit->_voltage;
317 }
318
319 /* Setter function for voltage - demonstrates simulated access modifier pattern */
320 void setVoltage(Circuit* circuit, double voltage) {
321     circuit->_voltage = voltage;
322     /* Update current using Ohm's Law when voltage changes */
323     circuit->current = calculateCurrentInAmperes(voltage, circuit->_resistance);
324 }
325
326 /* Function demonstrating memory address usage and pointer arithmetic */
327 void demonstrateMemoryAddressing(int array[], int size) {
328     printf("Memory addressing demonstration:\n");
329     /* Accessing and displaying memory addresses */
330     for (int i = 0; i < size; i++) {
331         printf("Element %d value: %d, address: %p\n",
332             i, array[i], (void*)&array[i]);
333     }
334 }
335
336 /* Function to find algorithm complexity constant in linear search */
337 double measureAlgorithmConstant(int array[], int size, int searches) {
338     clock_t start, end;

```

```

339     int target, found;
340     double totalTime = 0.0;
341
342     /* Run multiple searches to get a stable measurement */
343     for (int s = 0; s < searches; s++) {
344         target = rand() % 1000;
345         start = clock();
346
347         found = 0;
348         for (int i = 0; i < size; i++) {
349             if (array[i] == target) {
350                 found = 1;
351                 break;
352             }
353         }
354
355         end = clock();
356         totalTime += (double)(end - start) / CLOCKS_PER_SEC;
357     }
358
359     /* Time per element gives us the constant factor in O(n) */
360     return (totalTime / searches) / size;
361 }
362
363 /* Function to create an RGBA color value with alpha channel */
364 uint32_t createRGBAColor(uint8_t red, uint8_t green, uint8_t blue, uint8_t alpha) {
365     /* Combine components with alpha channel for transparency */
366     return (red << 24) | (green << 16) | (blue << 8) | alpha;
367 }
368
369 /* Demonstration of AND operator in boolean logic */
370 int checkAccessPermission(int userPermission, int requiredPermission) {
371     /* Using AND to verify that user has the required permission bits */
372     return (userPermission & requiredPermission) == requiredPermission;
373 }
374
375 /* Function simulating assembly instruction by using inline assembly */
376 int asmAddition(int a, int b) {
377     int result;
378
379     /* Using inline assembly for addition - demonstrates assembly instruction concept */
380     #ifdef __GNUC__
381     asm ("addl %1, %0" : "=r" (result) : "r" (b), "0" (a));
382     #else
383     /* Fallback for non-GCC compilers */
384     result = a + b;
385     #endif
386
387     return result;
388 }
389
390 /* Structure representing a database record with attributes */
391 typedef struct {
392     int id;          /* Primary key attribute */
393     char name[50];   /* Name attribute */
394     double value;    /* Value attribute */
395     char type[20];   /* Type attribute - demonstrates attributes in databases */
396 } Record;
397
398 /* Parse CSV data demonstrating ASCII values in character encoding */
399 void parseCSVLine(char* line, Record* record) {
400     int field = 0;
401     char* token = strtok(line, ",");
402
403     while (token != NULL) {
404         switch (field) {
405             case 0:
406                 record->id = atoi(token);
407                 break;
408             case 1:
409                 strncpy(record->name, token, 49);
410                 record->name[49] = '\0';
411                 break;

```

```

412         case 2:
413             record->value = atof(token);
414             break;
415         case 3:
416             strncpy(record->type, token, 19);
417             record->type[19] = '\0';
418             break;
419     }
420
421     /* Find ASCII values of first character in each field */
422     if (token[0] != '\0') {
423         printf("ASCII value of first character in field %d: %d\n",
424             field, (int)token[0]);
425     }
426
427     field++;
428     token = strtok(NULL, ",");
429 }
430 }
431
432 int main(int argc, char* argv[]) {
433     /* Using arguments passed to the program - demonstrates Arguments in function calls */
434     printf("Program name: %s\n", argv[0]);
435     printf("Number of arguments: %d\n\n", argc);
436
437     /* Area calculation demonstration */
438     double length = 5.0;
439     double width = 3.0;
440     double area = calculateRectangleArea(length, width);
441     printf("Rectangle area (%.1f x %.1f): %.2f square units\n\n", length, width, area);
442
443     /* Array demonstration - creating and accessing an array */
444     int dataArray[5] = {10, 20, 30, 40, 50};
445     int arraySize = 5;
446
447     printf("Array contents:\n");
448     for (int i = 0; i < arraySize; i++) {
449         printf("dataArray[%d] = %d\n", i, dataArray[i]);
450     }
451     printf("\n");
452
453     /* Assignment operator demonstration */
454     int accumulator = 0; /* Initializing an accumulator variable */
455     printf("Assignment and accumulation demonstration:\n");
456     printf("Initial accumulator value: %d\n", accumulator);
457
458     /* Using assignment with addition operation */
459     accumulator = accumulator + 5; /* Explicit addition */
460     printf("After adding 5: %d\n", accumulator);
461
462     accumulator += 10; /* Compound assignment */
463     printf("After adding 10 more: %d\n\n", accumulator);
464
465     /* Demonstrate angle in trigonometric functions */
466     printf("Sine wave values at different angles:\n");
467     for (double angle = 0.0; angle <= M_PI; angle += M_PI/4) {
468         printf("sin(%.2f radians) = %.4f\n", angle, sin(angle));
469     }
470     printf("\n");
471
472     /* Acceleration calculation */
473     double initialVelocity = 0.0; /* meters per second */
474     double finalVelocity = 20.0; /* meters per second */
475     double time = 5.0; /* seconds */
476     double acceleration = calculateAcceleration(initialVelocity, finalVelocity, time);
477     printf("Acceleration calculation: %.2f m/s²\n\n", acceleration);
478
479     /* Average calculation demonstration */
480     int values[] = {78, 92, 86, 65, 88, 95};
481     int count = sizeof(values) / sizeof(values[0]);
482     double average = calculateAverage(values, count);
483     printf("Average of values: %.2f\n\n", average);
484

```

```

485     /* Memory addressing demonstration */
486     demonstrateMemoryAddressing(dataArray, arraySize);
487     printf("\n");
488
489     /* Append operation demonstration */
490     printf("Array before append: ");
491     for (int i = 0; i < arraySize; i++) {
492         printf("%d ", dataArray[i]);
493     }
494     printf("\n");
495
496     int newValue = 60;
497     int* newArray = appendToArray(dataArray, &arraySize, newValue);
498
499     printf("Array after append: ");
500     for (int i = 0; i < arraySize; i++) {
501         printf("%d ", newArray[i]);
502     }
503     printf("\n\n");
504
505     /* Alpha channel demonstration in RGBA color */
506     uint32_t redColor = createRGBAColor(255, 0, 0, 255);    /* Opaque red */
507     uint32_t transBlue = createRGBAColor(0, 0, 255, 128);    /* Semi-transparent blue */
508     printf("RGBA Colors with Alpha channel:\n");
509     printf("Opaque red: 0x%08X\n", redColor);
510     printf("Semi-transparent blue: 0x%08X\n\n", transBlue);
511
512     /* Atomic operations demonstration with threads */
513     pthread_t thread1, thread2;
514     printf("Demonstrating atomic operations with threads...\n");
515     pthread_create(&thread1, NULL, incrementCounter, NULL);
516     pthread_create(&thread2, NULL, incrementCounter, NULL);
517
518     pthread_join(thread1, NULL);
519     pthread_join(thread2, NULL);
520
521     printf("Final counter value after atomic increments: %d\n\n",
522         atomic_load(&sharedCounter));
523
524     /* Circuit calculation demonstrating Amperes and access modifiers */
525     Circuit myCircuit;
526     myCircuit._resistance = 100.0;    /* ohms */
527     setVoltage(&myCircuit, 12.0);    /* volts */
528
529     printf("Circuit demonstration (Ohm's Law):\n");
530     printf("Voltage: %.2f V\n", getVoltage(&myCircuit));
531     printf("Resistance: %.2f Ω\n", myCircuit._resistance);
532     printf("Current: %.2f A\n\n", myCircuit.current);
533
534     /* AND operator demonstration for permission checking */
535     int userPermission = 0b1101;    /* Binary representation of permissions */
536     int readPermission = 0b0001;
537     int writePermission = 0b0010;
538     int executePermission = 0b0100;
539
540     printf("Permission checking with AND operator:\n");
541     printf("User has read permission: %s\n",
542         checkAccessPermission(userPermission, readPermission) ? "Yes" : "No");
543     printf("User has write permission: %s\n",
544         checkAccessPermission(userPermission, writePermission) ? "Yes" : "No");
545     printf("User has execute permission: %s\n",
546         checkAccessPermission(userPermission, executePermission) ? "Yes" : "No");
547     printf("\n");
548
549     /* Assembly instruction demonstration */
550     int num1 = 25, num2 = 17;
551     int asmResult = asmAddition(num1, num2);
552     printf("Assembly addition result: %d + %d = %d\n\n", num1, num2, asmResult);
553
554     /* CSV parsing demonstration with ASCII values */
555     char csvLine[] = "101,Database Record,42.5,Primary";
556     Record record;
557     printf("Parsing CSV with ASCII values:\n");

```



```
558     parseCSVLine(csvLine, &record);
559     printf("Parsed record - ID: %d, Name: %s, Value: %.1f, Type: %s\n\n",
560           record.id, record.name, record.value, record.type);
561
562     /* Algorithm complexity constant measurement */
563     int testArray[1000];
564     for (int i = 0; i < 1000; i++) {
565         testArray[i] = rand() % 1000;
566     }
567
568     double complexityConstant = measureAlgorithmConstant(testArray, 1000, 100);
569     printf("Algorithm complexity constant for linear search: %.9f seconds per element\n",
570           complexityConstant);
571     printf("This constant factor affects actual performance even though O(n) notation\n");
572     printf("omits it in asymptotic analysis.\n");
573
574     /* Clean up dynamically allocated memory */
575     free(newArray);
576
577     return 0;
578 }
579
580 ]
581
582 2   (1 3){
583     b
584 }[
585
586 1   Base (in logarithmic functions or number systems)
587 2   Bit  (in binary operations)
588 3   Byte (in memory allocation)
589 4   Boolean value (in logic operations)
590 5   Buffer (in I/O operations)
591 6   Branch instruction (in assembly language)
592 7   Break statement (in loop control)
593 8   Block size (in storage allocation)
594 9   Bandwidth (in network calculations)
595 10  B-register (in CPU architecture)
596 11  Binary operator (in mathematical expressions)
597 12  Backup operation (in data management)
598 13  Bias value (in neural networks)
599 14  Boundary condition (in algorithms)
600 15  Breadth (in geometric calculations)
601 16  Backtracking step (in search algorithms)
602 17  Bucket (in hash tables)
603 18  Baud rate (in communication protocols)
604 19  Batch size (in processing operations)
605 20  Billion bytes (alternative notation for gigabytes)
606
607 DEFINITIONS
608
609 1. Base (in logarithmic functions or number systems): The reference value in a positional number system that determines the value of each digit according to its position. In
   logarithmic functions, it represents the fixed positive number used as the implicit exponent to which another number is raised to yield the original number.
610
611 2. Bit (in binary operations): The fundamental and indivisible unit of digital information capable of existing in one of two states, conventionally represented as 0 or 1. It
   constitutes the smallest addressable element in digital computing and serves as the foundation for all binary operations.
612
613 3. Byte (in memory allocation): A contiguous sequence of eight bits that operates as a fundamental unit of digital storage and memory addressing. It represents the minimum addressable
   unit of memory in most computer architectures and serves as the standard unit for representing a single character.
614
615 4. Boolean value (in logic operations): A data type with exactly two possible values representing truth values in propositional logic, typically denoted as "true" and "false." It
   serves as the foundational element for logical decision-making in programming and computational processes.
616
617 5. Buffer (in I/O operations): A temporary data storage region that holds information while it is being transferred between two devices or processes that may operate at different
   speeds or with different priorities. It facilitates asynchronous operations and manages timing discrepancies between data producer and consumer.
618
619 6. Branch instruction (in assembly language): A machine-level directive that alters the control flow of program execution by transferring execution to a different instruction address
   based on specified conditions. It enables conditional execution paths and implements decision structures within assembly programs.
620
621 7. Break statement (in loop control): A control flow construct that terminates the enclosing iterative structure when encountered, transferring execution to the first statement
   following the loop. It provides a mechanism for exiting loops prematurely when certain conditions are met.
622
623 8. Block size (in storage allocation): The fixed quantum of contiguous memory or storage space allocated as a single unit during memory management operations. It defines the
```

```
granularity of resource allocation and often represents the minimum unit of data transfer between hierarchical storage levels.
624
625 9. Bandwidth (in network calculations): The maximum rate of data transfer across a communication channel within a given time period, typically measured in bits per second. It
quantifies the data-carrying capacity of a network connection or interface.
626
627 10. B-register (in CPU architecture): A general-purpose processor register designated for temporary data storage and manipulation during execution of instructions. It often serves
specialized functions in certain instruction sequences and addressing modes within the central processing unit.
628
629 11. Binary operator (in mathematical expressions): A mathematical or logical operation that requires exactly two operands to produce a result. It forms expressions by combining two
input values according to specific rules defined by the operation semantics.
630
631 12. Backup operation (in data management): A procedural function that creates and stores duplicate copies of data to enable recovery in case of data loss, corruption, or system
failure. It preserves organizational information assets by maintaining point-in-time copies separate from primary storage.
632
633 13. Bias value (in neural networks): A trainable parameter added to the weighted sum of inputs before activation in an artificial neuron, allowing the activation function to be shifted
along its input axis. It enables the neural network to learn patterns that do not pass through the origin.
634
635 14. Boundary condition (in algorithms): A constraint or criterion that defines the valid limits or edge cases for algorithm operation, often specifying behavior at the extremes of
input domains. It establishes how algorithms handle special cases occurring at the periphery of their operational scope.
636
637 15. Breadth (in geometric calculations): A measurement of the shorter dimension of a rectangular or elongated two-dimensional object, perpendicular to its length. It quantifies the
extent of an object in one of its principal directions.
638
639 16. Backtracking step (in search algorithms): A recursive algorithmic technique that incrementally builds candidate solutions and abandons partial solutions when they are determined to
be invalid, reverting to previous states to explore alternative paths. It systematically eliminates non-viable solution branches to reduce search space.
640
641 17. Bucket (in hash tables): A storage unit that contains all elements mapping to the same hash value in a hash table implementation. It provides a containment mechanism for resolving
hash collisions by grouping elements with identical hash codes.
642
643 18. Baud rate (in communication protocols): The number of signal state changes or symbols transmitted per second over a communication channel, regardless of the information content of
those symbols. It defines the signaling rate for data transmission in serial communication systems.
644
645 19. Batch size (in processing operations): The quantity of items or data records processed as a single operational unit before results are returned or committed. It optimizes
processing efficiency by amortizing overhead costs across multiple items and controlling resource utilization.
646
647 20. Billion bytes (alternative notation for gigabytes): A unit of digital information storage capacity equivalent to 10^9 bytes (1,000,000,000 bytes) in the decimal-based International
System of Units. It provides a standardized measurement for expressing large data volumes in computing and storage contexts.
648
649 IMPLEMENTATIONS
650
651 /*
652  * File: b_concepts_demo.c
653  * Description: Comprehensive demonstration of 20 "B" computing concepts
654  *
655  * This program demonstrates various computing concepts starting with 'B'
656  * through practical implementations in C
657  */
658
659 #include <stdio.h>
660 #include <stdlib.h>
661 #include <math.h>
662 #include <string.h>
663 #include <stdbool.h>
664 #include <time.h>
665 #include <stdint.h>
666
667 /* Define constants for system parameters */
668 #define BUFFER_SIZE 1024
669 #define BLOCK_SIZE 4096
670 #define BAUD_RATE 9600
671 #define BATCH_SIZE 64
672 #define ONE_BILLION_BYTES 1000000000 /* Alternative notation for gigabytes */
673 #define BANDWIDTH_MBPS 100 /* Network bandwidth in Mbps */
674
675 /* Structure to simulate a basic neural network neuron */
676 typedef struct {
677     double* weights;
678     double bias; /* Bias value in neural networks */
679     int num_inputs;
680 } Neuron;
681
682 /* Structure to represent a hash table bucket */
683 typedef struct Node {
```

```

684     int key;
685     int value;
686     struct Node* next;
687 } Node;
688
689 typedef struct {
690     Node** buckets; /* Array of bucket pointers */
691     int bucket_count;
692 } HashTable;
693
694 /* Structure to emulate CPU registers */
695 typedef struct {
696     uint32_t a_register;
697     uint32_t b_register; /* B-register in CPU architecture */
698     uint32_t c_register;
699     uint32_t instruction_pointer;
700 } CPURegisters;
701
702 /* Function to calculate logarithm with custom base */
703 double log_base(double value, double base) {
704     /* Demonstrates the concept of base in logarithmic functions */
705     /* Using the change of base formula: log_b(x) = log_c(x) / log_c(b) */
706     return log(value) / log(base);
707 }
708
709 /* Function to convert decimal to binary representation */
710 void decimal_to_binary(int decimal, char* binary, int num_bits) {
711     /* Demonstrates bit manipulation in binary operations */
712     for (int i = num_bits - 1; i >= 0; i--) {
713         /* Extract each bit using bitwise AND operator */
714         binary[num_bits - 1 - i] = ((decimal >> i) & 1) ? '1' : '0';
715     }
716     binary[num_bits] = '\0';
717 }
718
719 /* Function to allocate memory in specified block sizes */
720 void* block_allocate(size_t num_bytes) {
721     /* Calculates number of blocks needed to store the requested bytes */
722     int num_blocks = (num_bytes + BLOCK_SIZE - 1) / BLOCK_SIZE;
723     size_t total_size = num_blocks * BLOCK_SIZE;
724
725     printf("Allocating %zu bytes in %d blocks of %d bytes each\n",
726           num_bytes, num_blocks, BLOCK_SIZE);
727
728     /* Allocate memory in multiples of BLOCK_SIZE */
729     return malloc(total_size);
730 }
731
732 /* Function to calculate rectangle area with length and breadth */
733 double rectangle_area(double length, double breadth) {
734     /* Demonstrates breadth in geometric calculations */
735     return length * breadth;
736 }
737
738 /* Function to simulate data transfer with bandwidth calculation */
739 double calculate_transfer_time(double file_size_bytes, double bandwidth_mbps) {
740     /* Convert bandwidth from Mbps to bytes per second (B/s) */
741     double bandwidth_bytes_per_sec = (bandwidth_mbps * 1000000) / 8;
742
743     /* Calculate transfer time in seconds */
744     return file_size_bytes / bandwidth_bytes_per_sec;
745 }
746
747 /* Function that performs a binary operation */
748 double binary_operation(double a, double b, char operator) {
749     /* Demonstrates binary operator in mathematical expressions */
750     switch (operator) {
751         case '+': return a + b;
752         case '-': return a - b;
753         case '*': return a * b;
754         case '/': return a / b;
755         case '^': return pow(a, b);
756         default: return 0;

```

```

757     }
758 }
759
760 /* Function that creates a neural network neuron with bias */
761 Neuron* create_neuron(int num_inputs, double bias) {
762     Neuron* neuron = (Neuron*)malloc(sizeof(Neuron));
763
764     neuron->num_inputs = num_inputs;
765     neuron->bias = bias; /* Setting the bias value for the neuron */
766
767     /* Allocate memory for weights */
768     neuron->weights = (double*)malloc(num_inputs * sizeof(double));
769
770     /* Initialize weights with random values */
771     for (int i = 0; i < num_inputs; i++) {
772         neuron->weights[i] = ((double)rand() / RAND_MAX) * 2 - 1; /* Range: -1 to 1 */
773     }
774
775     printf("Created neuron with %d inputs and bias %.4f\n", num_inputs, bias);
776     return neuron;
777 }
778
779 /* Function that performs neuron activation with bias */
780 double activate_neuron(Neuron* neuron, double* inputs) {
781     double sum = neuron->bias; /* Start with the bias value */
782
783     /* Calculate weighted sum of inputs */
784     for (int i = 0; i < neuron->num_inputs; i++) {
785         sum += neuron->weights[i] * inputs[i];
786     }
787
788     /* Apply activation function (sigmoid) */
789     return 1.0 / (1.0 + exp(-sum));
790 }
791
792 /* Hash function for the hash table */
793 int hash_function(int key, int bucket_count) {
794     return key % bucket_count; /* Simple modulo hash function */
795 }
796
797 /* Create a new hash table */
798 HashTable* create_hash_table(int bucket_count) {
799     HashTable* table = (HashTable*)malloc(sizeof(HashTable));
800     table->bucket_count = bucket_count;
801
802     /* Allocate memory for buckets array */
803     table->buckets = (Node**)malloc(bucket_count * sizeof(Node*));
804
805     /* Initialize all buckets to NULL */
806     for (int i = 0; i < bucket_count; i++) {
807         table->buckets[i] = NULL;
808     }
809
810     printf("Created hash table with %d buckets\n", bucket_count);
811     return table;
812 }
813
814 /* Insert a key-value pair into the hash table */
815 void hash_table_insert(HashTable* table, int key, int value) {
816     /* Compute bucket index for this key */
817     int bucket_idx = hash_function(key, table->bucket_count);
818
819     /* Create a new node */
820     Node* new_node = (Node*)malloc(sizeof(Node));
821     new_node->key = key;
822     new_node->value = value;
823
824     /* Insert at the beginning of the bucket's linked list */
825     new_node->next = table->buckets[bucket_idx];
826     table->buckets[bucket_idx] = new_node;
827
828     printf("Inserted key %d at bucket %d\n", key, bucket_idx);
829 }

```

```

830
831 /* Function to backup a file (demonstrate backup operation) */
832 bool backup_file(const char* source_path, const char* backup_path) {
833     /* Open source file for reading in binary mode */
834     FILE* source = fopen(source_path, "rb");
835     if (!source) {
836         printf("Error: Cannot open source file %s\n", source_path);
837         return false;
838     }
839
840     /* Open backup file for writing in binary mode */
841     FILE* backup = fopen(backup_path, "wb");
842     if (!backup) {
843         printf("Error: Cannot create backup file %s\n", backup_path);
844         fclose(source);
845         return false;
846     }
847
848     /* Create a buffer for file I/O operations */
849     char buffer[BUFFER_SIZE];
850     size_t bytes_read;
851
852     /* Read from source and write to backup in chunks of BUFFER_SIZE */
853     while ((bytes_read = fread(buffer, 1, BUFFER_SIZE, source)) > 0) {
854         fwrite(buffer, 1, bytes_read, backup);
855     }
856
857     /* Close both files */
858     fclose(source);
859     fclose(backup);
860
861     printf("Successfully backed up %s to %s\n", source_path, backup_path);
862     return true;
863 }
864
865 /* Function to demonstrate batch processing */
866 void process_in_batches(int* data, int total_items, int batch_size) {
867     int batch_count = (total_items + batch_size - 1) / batch_size;
868
869     printf("Processing %d items in batches of %d (%d batches total)\n",
870         total_items, batch_size, batch_count);
871
872     for (int batch = 0; batch < batch_count; batch++) {
873         int start_idx = batch * batch_size;
874         int end_idx = (batch + 1) * batch_size;
875
876         /* Apply boundary condition for the last batch */
877         if (end_idx > total_items) {
878             end_idx = total_items;
879         }
880
881         int current_batch_size = end_idx - start_idx;
882         printf("Processing batch %d (%d items): ", batch + 1, current_batch_size);
883
884         /* Process each item in the batch */
885         for (int i = start_idx; i < end_idx; i++) {
886             /* For demonstration, we just double each value */
887             data[i] *= 2;
888             printf("%d ", data[i]);
889         }
890         printf("\n");
891     }
892 }
893
894 /* Function to solve N-Queens problem using backtracking */
895 bool is_safe(int* board, int row, int col, int n) {
896     /* Check if a queen can be placed at board[row][col] */
897
898     /* Check this row on left side */
899     for (int i = 0; i < col; i++) {
900         if (board[i] == row) {
901             return false;
902         }

```

```

903     }
904
905     /* Check upper diagonal on left side */
906     for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {
907         if (board[j] == i) {
908             return false;
909         }
910     }
911
912     /* Check lower diagonal on left side */
913     for (int i = row, j = col; i < n && j >= 0; i++, j--) {
914         if (board[j] == i) {
915             return false;
916         }
917     }
918
919     return true;
920 }
921
922 bool solve_n_queens(int* board, int col, int n) {
923     /* Base case: If all queens are placed, return true */
924     if (col >= n) {
925         return true;
926     }
927
928     /* Try placing queen in all rows of this column */
929     for (int row = 0; row < n; row++) {
930         /* Check if queen can be placed here */
931         if (is_safe(board, row, col, n)) {
932             /* Place the queen */
933             board[col] = row;
934
935             /* Recursively place rest of the queens */
936             if (solve_n_queens(board, col + 1, n)) {
937                 return true;
938             }
939
940             /* If placing queen in board[row][col] doesn't lead to a solution,
941              then BACKTRACK by removing queen from board[row][col] */
942             board[col] = -1; /* Demonstrates backtracking step */
943         }
944     }
945
946     /* If queen cannot be placed in any row in this column */
947     return false;
948 }
949
950 /* Function to print board configuration for N-Queens */
951 void print_n_queens_solution(int* board, int n) {
952     printf("N-Queens solution:\n");
953     for (int i = 0; i < n; i++) {
954         for (int j = 0; j < n; j++) {
955             if (board[j] == i) {
956                 printf("Q ");
957             } else {
958                 printf(". ");
959             }
960         }
961         printf("\n");
962     }
963 }
964
965 /* Function to simulate assembly branch instruction */
966 void simulate_branch_instruction(CPURegisters* cpu, bool condition, uint32_t target_address) {
967     printf("Current instruction pointer: 0x%08X\n", cpu->instruction_pointer);
968
969     if (condition) {
970         /* Branch taken - simulate changing the instruction pointer */
971         printf("Branch condition TRUE - jumping to target address\n");
972         cpu->instruction_pointer = target_address;
973     } else {
974         /* Branch not taken - increment instruction pointer */
975         printf("Branch condition FALSE - continuing sequential execution\n");

```

```

976         cpu->instruction_pointer += 4; /* Assuming 4-byte instructions */
977     }
978
979     printf("New instruction pointer: 0x%08X\n", cpu->instruction_pointer);
980 }
981
982 /* Function to calculate data transfer with baud rate */
983 double calculate_serial_transfer_time(int data_bytes, int baud_rate) {
984     /* Convert bytes to bits (8 bits per byte + 2 bits for start/stop) */
985     int total_bits = data_bytes * 10;
986
987     /* Calculate time in seconds */
988     return (double)total_bits / baud_rate;
989 }
990
991 /* Main function demonstrating all 20 concepts */
992 int main() {
993     srand(time(NULL));
994
995     printf("==== B Concepts Demonstration Program ==== \n\n");
996
997     /* 1. Base in logarithmic functions */
998     double number = 1024.0;
999     double base2_log = log_base(number, 2.0);
1000    double base10_log = log_base(number, 10.0);
1001
1002    printf("1. BASE in logarithmic functions:\n");
1003    printf("    log_2(%.1f) = %.2f\n", number, base2_log);
1004    printf("    log_10(%.1f) = %.2f\n\n", number, base10_log);
1005
1006    /* 2. Bit in binary operations */
1007    int decimal_value = 171; /* 10101011 in binary */
1008    char binary_str[33];
1009    decimal_to_binary(decimal_value, binary_str, 8);
1010
1011    printf("2. BIT in binary operations:\n");
1012    printf("    Decimal %d in 8-bit binary: %s\n", decimal_value, binary_str);
1013
1014    /* Demonstrate bit manipulation */
1015    int set_bit_pos = 3;
1016    int bit_value = (decimal_value >> set_bit_pos) & 1;
1017    printf("    Bit at position %d is: %d\n\n", set_bit_pos, bit_value);
1018
1019    /* 3. Byte in memory allocation */
1020    char* byte_array = (char*)malloc(10 * sizeof(char));
1021    printf("3. BYTE in memory allocation:\n");
1022    printf("    Allocated 10 bytes of memory at address %p\n", (void*)byte_array);
1023    printf("    Size of each element: %zu bytes\n\n", sizeof(char));
1024
1025    /* 4. Boolean value in logic operations */
1026    bool condition1 = true;
1027    bool condition2 = false;
1028
1029    printf("4. BOOLEAN VALUE in logic operations:\n");
1030    printf("    condition1 = %s\n", condition1 ? "true" : "false");
1031    printf("    condition2 = %s\n", condition2 ? "true" : "false");
1032    printf("    condition1 AND condition2 = %s\n", (condition1 && condition2) ? "true" : "false");
1033    printf("    condition1 OR condition2 = %s\n\n", (condition1 || condition2) ? "true" : "false");
1034
1035    /* 5. Buffer in I/O operations */
1036    printf("5. BUFFER in I/O operations:\n");
1037    printf("    Using a buffer of size %d bytes for file operations\n", BUFFER_SIZE);
1038    printf("    This improves efficiency by reducing system calls\n\n");
1039
1040    /* 6. Branch instruction in assembly language */
1041    CPURegisters cpu = {0};
1042    cpu.instruction_pointer = 0x1000;
1043    cpu.b_register = 42; /* Set B-register value */
1044
1045    printf("6. BRANCH INSTRUCTION in assembly language:\n");
1046    printf("    B-register value: %u\n", cpu.b_register);
1047    /* Simulate a branch if b_register > 30 */
1048    simulate_branch_instruction(&cpu, cpu.b_register > 30, 0x2000);

```

```

1049     printf("\n");
1050
1051     /* 7. Break statement in loop control */
1052     printf("7. BREAK statement in loop control:\n");
1053     printf("    Looking for the first multiple of 7 greater than 50:\n");
1054
1055     for (int i = 1; i <= 100; i++) {
1056         if (i * 7 > 50) {
1057             printf("    Found: %d (7 × %d)\n", i * 7, i);
1058             break; /* Terminate loop when condition is met */
1059         }
1060     }
1061     printf("\n");
1062
1063     /* 8. Block size in storage allocation */
1064     printf("8. BLOCK SIZE in storage allocation:\n");
1065     void* block_memory = block_allocate(10000);
1066     free(block_memory);
1067     printf("\n");
1068
1069     /* 9. Bandwidth in network calculations */
1070     double file_size_mb = 50.0;
1071     double file_size_bytes = file_size_mb * 1000000;
1072
1073     printf("9. BANDWIDTH in network calculations:\n");
1074     printf("    File size: %.1f MB (%.0f bytes)\n", file_size_mb, file_size_bytes);
1075     printf("    Network bandwidth: %d Mbps\n", BANDWIDTH_MBPS);
1076
1077     double transfer_seconds = calculate_transfer_time(file_size_bytes, BANDWIDTH_MBPS);
1078     printf("    Estimated transfer time: %.2f seconds\n\n", transfer_seconds);
1079
1080     /* 10. B-register in CPU architecture - already used in branch instruction demo */
1081     printf("10. B-REGISTER in CPU architecture:\n");
1082     printf("    Used in branch instruction demonstration (value: %u)\n\n", cpu.b_register);
1083
1084     /* 11. Binary operator in mathematical expressions */
1085     double operand1 = 15.0, operand2 = 3.0;
1086
1087     printf("11. BINARY OPERATOR in mathematical expressions:\n");
1088     printf("    %g + %g = %g\n", operand1, operand2, binary_operation(operand1, operand2, '+'));
1089     printf("    %g - %g = %g\n", operand1, operand2, binary_operation(operand1, operand2, '-'));
1090     printf("    %g * %g = %g\n", operand1, operand2, binary_operation(operand1, operand2, '*'));
1091     printf("    %g / %g = %g\n", operand1, operand2, binary_operation(operand1, operand2, '/'));
1092     printf("    %g ^ %g = %g\n\n", operand1, operand2, binary_operation(operand1, operand2, '^'));
1093
1094     /* 12. Backup operation in data management */
1095     printf("12. BACKUP OPERATION in data management:\n");
1096     /* For demonstration purposes, create a test file */
1097     const char* test_file = "test_data.txt";
1098     const char* backup_file = "test_data.bak";
1099
1100     FILE* test = fopen(test_file, "w");
1101     if (test) {
1102         fprintf(test, "This is test data that needs to be backed up.\n");
1103         fprintf(test, "It demonstrates the backup operation in data management.\n");
1104         fclose(test);
1105
1106         /* Perform backup */
1107         backup_file(test_file, backup_file);
1108     } else {
1109         printf("    Error creating test file\n");
1110     }
1111     printf("\n");
1112
1113     /* 13. Bias value in neural networks */
1114     printf("13. BIAS VALUE in neural networks:\n");
1115     Neuron* neuron = create_neuron(3, 0.5); /* Create neuron with bias 0.5 */
1116
1117     /* Test the neuron */
1118     double test_inputs[3] = {0.2, 0.7, 0.9};
1119     double activation = activate_neuron(neuron, test_inputs);
1120
1121     printf("    Neuron activation result: %.4f\n\n", activation);

```



```

1122
1123 /* 14. Boundary condition in algorithms */
1124 printf("14. BOUNDARY CONDITION in algorithms:\n");
1125 /* Create an array to process */
1126 int data[25];
1127 for (int i = 0; i < 25; i++) {
1128     data[i] = i + 1;
1129 }
1130
1131 /* Process the data in batches, handling boundary conditions */
1132 process_in_batches(data, 25, BATCH_SIZE);
1133 printf("\n");
1134
1135 /* 15. Breadth in geometric calculations */
1136 double length = 8.5;
1137 double breadth = 5.25;
1138
1139 printf("15. BREADTH in geometric calculations:\n");
1140 printf("    Rectangle with length %.2f and breadth %.2f\n", length, breadth);
1141 printf("    Area: %.2f square units\n\n", rectangle_area(length, breadth));
1142
1143 /* 16. Backtracking step in search algorithms */
1144 printf("16. BACKTRACKING STEP in search algorithms:\n");
1145 printf("    Solving 4-Queens problem using backtracking:\n");
1146
1147 int board_size = 4;
1148 int* queens_board = (int*)malloc(board_size * sizeof(int));
1149
1150 /* Initialize board with -1 in all positions */
1151 for (int i =
1152     0; i < board_size; i++) {
1153     queens_board[i] = -1;
1154 }
1155
1156 if (solve_n_queens(queens_board, 0, board_size)) {
1157     print_n_queens_solution(queens_board, board_size);
1158 } else {
1159     printf("    No solution exists\n");
1160 }
1161 printf("\n");
1162
1163 /* 17. Bucket in hash tables */
1164 printf("17. BUCKET in hash tables:\n");
1165 HashTable* hash_table = create_hash_table(5); /* Create hash table with 5 buckets */
1166
1167 /* Insert some key-value pairs */
1168 hash_table_insert(hash_table, 5, 100);
1169 hash_table_insert(hash_table, 10, 200);
1170 hash_table_insert(hash_table, 15, 300);
1171 hash_table_insert(hash_table, 20, 400);
1172 hash_table_insert(hash_table, 25, 500);
1173
1174 /* Demonstrate hash collision (5 and 10 will go to the same bucket) */
1175 hash_table_insert(hash_table, 30, 600); /* 30 % 5 = 0, same as 5 */
1176 printf("\n");
1177
1178 /* 18. Baud rate in communication protocols */
1179 int message_size = 1024; /* bytes */
1180
1181 printf("18. BAUD RATE in communication protocols:\n");
1182 printf("    Message size: %d bytes\n", message_size);
1183 printf("    Baud rate: %d symbols per second\n", BAUD_RATE);
1184
1185 double serial_transfer_time = calculate_serial_transfer_time(message_size, BAUD_RATE);
1186 printf("    Serial transmission time: %.2f seconds\n\n", serial_transfer_time);
1187
1188 /* 19. Batch size in processing operations - already used in boundary conditions */
1189 printf("19. BATCH SIZE in processing operations:\n");
1190 printf("    Used batch size of %d in boundary conditions demonstration\n", BATCH_SIZE);
1191 printf("    Proper batch sizing optimizes processing efficiency\n\n");
1192
1193 /* 20. Billion bytes (alternative notation for gigabytes) */
1194 printf("20. BILLION BYTES (alternative notation for gigabytes):\n");

```

```
1195     double storage_in_gb = (double)ONE_BILLION_BYTES / ONE_BILLION_BYTES;
1196     double storage_in_gib = (double)ONE_BILLION_BYTES / (1024 * 1024 * 1024);
1197
1198     printf("    1 billion bytes = %.1f GB (decimal)\n", storage_in_gb);
1199     printf("    1 billion bytes = %.2f GiB (binary)\n", storage_in_gib);
1200     printf("    The difference illustrates the distinction between\n");
1201     printf("    decimal (10^9) and binary (2^30) notations for storage.\n");
1202
1203     /* Clean up allocated resources */
1204     free(byte_array);
1205     free(queens_board);
1206     free(neuron->weights);
1207     free(neuron);
1208
1209     /* Clean up hash table */
1210     for (int i = 0; i < hash_table->bucket_count; i++) {
1211         Node* current = hash_table->buckets[i];
1212         while (current != NULL) {
1213             Node* temp = current;
1214             current = current->next;
1215             free(temp);
1216         }
1217     }
1218     free(hash_table->buckets);
1219     free(hash_table);
1220
1221     /* Remove test files */
1222     remove(test_file);
1223     remove(backup_file);
1224
1225     return 0;
1226 }
1227
1228 ]
1229
1230 3   (1 4){
1231     c
1232 }[
1233
1234 1   Count (in iterations or loops)
1235 2   Constant (in mathematical equations)
1236 3   Complement (in set theory)
1237 4   Carry bit (in binary addition)
1238 5   Character (in string operations)
1239 6   Cache (in memory hierarchy)
1240 7   Comparison operator (in conditional statements)
1241 8   Coordinate (in geometric positioning)
1242 9   Clear operation (for registers or memory)
1243 10  Clock cycle (in CPU timing)
1244 11  Coefficient (in polynomial expressions)
1245 12  Capacity (in resource allocation)
1246 13  Concatenation (in string operations)
1247 14  Checksum (in data integrity)
1248 15  Counter register (in processor architecture)
1249 16  Compression ratio (in data compression)
1250 17  Control flow instruction (in programming)
1251 18  Current (in electrical circuit calculations)
1252 19  Copy operation (in memory management)
1253 20  Color value (in graphics programming)
1254
1255 DEFINITIONS
1256
1257 1. Count (in iterations or loops): A cumulative integer value that tracks the number of completed repetitions in an iterative process. It serves as both a record of traversed elements
and a control mechanism to determine loop termination when a predetermined threshold is reached.
1258
1259 2. Constant (in mathematical equations): A fixed numerical value that does not change throughout a computational process or mathematical operation. It represents an invariant quantity
whose magnitude remains stable regardless of changes in other variables within the equation.
1260
1261 3. Complement (in set theory): The collection of all elements in the universal set that are not contained in a specified subset. It represents the logical negation of set membership
and is fundamental to operations involving set difference and mutual exclusivity.
1262
1263 4. Carry bit (in binary addition): A binary digit generated when the sum of two bits plus any previous carry exceeds the value representable in a single bit position. It propagates
excess value to the next higher bit position during arithmetic operations.
```

```
1264
1265 5. Character (in string operations): A discrete textual or symbolic unit that serves as the atomic component of string data. It represents a single letter, digit, punctuation mark, or
control code according to a specific character encoding standard.

1266
1267 6. Cache (in memory hierarchy): A high-speed temporary storage component that retains frequently accessed data to reduce average memory access latency. It exploits locality principles
to maintain copies of data from slower memory tiers for accelerated subsequent access.

1268
1269 7. Comparison operator (in conditional statements): A relational function that evaluates the relationship between two values and produces a Boolean result indicating whether the
specified condition holds true. It enables decision-making constructs by testing equality, inequality, or relative ordering.

1270
1271 8. Coordinate (in geometric positioning): A numerical value that specifies the position of a point along a dimensional axis within a reference frame. It provides a precise location
identifier within a coordinate system for spatial representation and manipulation.

1272
1273 9. Clear operation (for registers or memory): An instruction that resets the contents of a storage location to a predetermined initial state, typically zero. It initializes memory
regions or processor registers by eliminating previous values to establish a known baseline state.

1274
1275 10. Clock cycle (in CPU timing): The fundamental timing interval in a synchronous digital system, determined by the period of the processor's oscillating timing signal. It establishes
the basic unit of time for instruction execution and sequential circuit operation.

1276
1277 11. Coefficient (in polynomial expressions): A numerical multiplier associated with a variable term in a polynomial or algebraic expression. It quantifies the contribution of the term
to the overall expression and determines its magnitude within the computational result.

1278
1279 12. Capacity (in resource allocation): The maximum quantity of data units or elements that a container, storage medium, or communication channel can accommodate simultaneously. It
defines the upper bound on resource utilization and constrains system scalability.

1280
1281 13. Concatenation (in string operations): A binary operation that sequentially combines two strings by appending the second string to the end of the first, preserving the original
character sequence of both operands. It produces a new string containing all characters from both source strings.

1282
1283 14. Checksum (in data integrity): A derived value computed from a data sequence using a deterministic algorithm to detect errors in transmission or storage. It enables validation of
data integrity by comparing checksums calculated before and after data transfer operations.

1284
1285 15. Counter register (in processor architecture): A specialized processor register designed to maintain a sequential count that can be automatically incremented or decremented by
hardware without explicit arithmetic instructions. It facilitates iteration control and event counting operations.

1286
1287 16. Compression ratio (in data compression): A quantitative measure expressing the relative reduction in data volume achieved by compression algorithms, calculated as the ratio between
the uncompressed and compressed data sizes. It quantifies compression efficiency and storage economy.

1288
1289 17. Control flow instruction (in programming): A directive that alters the sequential execution order of program instructions by transferring control to a different location in the
program. It enables conditional execution, iteration, and subroutine invocation through non-linear execution paths.

1290
1291 18. Current (in electrical circuit calculations): The rate of flow of electric charge through a conductive medium, typically measured in amperes. It represents the movement of charged
particles and serves as a fundamental parameter in electrical circuit analysis and design.

1292
1293 19. Copy operation (in memory management): A data transfer procedure that duplicates information from a source location to a destination location while preserving the original content.
It creates independent replicas of data structures to enable operations on separate instances.

1294
1295 20. Color value (in graphics programming): A numerical representation of a specific color within a defined color space, typically encoding intensity levels for primary color
components. It provides a standardized method for specifying visual appearance in digital imaging and rendering systems.

1296
1297 IMPLEMENTATIONS
1298
1299 /*
1300  * File: c_concepts_demo.c
1301  * Description: Comprehensive demonstration of 20 C-related computing concepts
1302  *
1303  * This program demonstrates various computing concepts through practical
1304  * implementations in C programming language
1305  */
1306
1307 #include <stdio.h>
1308 #include <stdlib.h>
1309 #include <string.h>
1310 #include <stdbool.h>
1311 #include <math.h>
1312 #include <time.h>
1313 #include <stdint.h>
1314
1315 /* Constants for system parameters */
1316 #define MAX_CAPACITY 1024 /* Maximum data capacity */
1317 #define UNIVERSAL_SET_SIZE 100 /* Size of universal set for set operations */
1318 #define PI 3.14159265358979323846 /* Constant in mathematical equations */
1319 #define CLOCK_SPEED_MHZ 3200 /* CPU clock speed in MHz */
1320 #define CACHE_SIZE 256 /* Size of cache in bytes */
```

```

1321 #define CHECKSUM_INIT 0xFFFF /* Initial value for checksum calculations */
1322
1323 /* Structure to represent a 2D coordinate */
1324 typedef struct {
1325     double x; /* x-coordinate */
1326     double y; /* y-coordinate */
1327 } Coordinate;
1328
1329 /* Structure to simulate a processor register set */
1330 typedef struct {
1331     uint32_t general_purpose[4]; /* General purpose registers */
1332     uint32_t counter_register; /* Counter register for iteration tracking */
1333     uint32_t status_register; /* Status register for flags */
1334 } ProcessorRegisters;
1335
1336 /* Structure to represent an electrical circuit */
1337 typedef struct {
1338     double voltage; /* Voltage in volts */
1339     double resistance; /* Resistance in ohms */
1340     double current; /* Current in amperes */
1341 } Circuit;
1342
1343 /* Structure for RGBA color representation */
1344 typedef struct {
1345     uint8_t red; /* Red component (0-255) */
1346     uint8_t green; /* Green component (0-255) */
1347     uint8_t blue; /* Blue component (0-255) */
1348     uint8_t alpha; /* Alpha component (0-255) for transparency */
1349 } ColorRGBA;
1350
1351 /* Structure to represent a polynomial expression */
1352 typedef struct {
1353     double* coefficients; /* Array of coefficients for each term */
1354     int degree; /* Degree of the polynomial */
1355 } Polynomial;
1356
1357 /*
1358  * Function to calculate carry in binary addition
1359  * Demonstrates carry bit in binary addition
1360  */
1361 uint8_t add_with_carry(uint8_t a, uint8_t b, uint8_t* carry) {
1362     uint16_t sum = (uint16_t)a + (uint16_t)b + (uint16_t)*carry;
1363     *carry = (sum > 255) ? 1 : 0; /* Set carry bit if sum exceeds byte capacity */
1364     return (uint8_t)(sum & 0xFF); /* Return lower 8 bits */
1365 }
1366
1367 /*
1368  * Function to perform binary addition with carry propagation
1369  * Demonstrates carry bit and binary arithmetic
1370  */
1371 void binary_add_bytes(uint8_t* a, uint8_t* b, uint8_t* result, int byte_count) {
1372     uint8_t carry = 0;
1373
1374     printf("Binary addition with carry propagation:\n");
1375
1376     for (int i = 0; i < byte_count; i++) {
1377         /* Add current bytes with carry from previous addition */
1378         result[i] = add_with_carry(a[i], b[i], &carry);
1379
1380         printf(" Byte %d: %u + %u = %u (carry: %u)\n",
1381             i, a[i], b[i], result[i], carry);
1382     }
1383
1384     /* Handle final carry if present */
1385     if (carry) {
1386         printf(" Final carry bit: %u (overflow occurred)\n", carry);
1387     }
1388 }
1389
1390 /*
1391  * Function to calculate set complement
1392  * Demonstrates complement in set theory
1393  */

```

```

1394 void calculate_set_complement(bool* set, bool* universal, bool* result, int size) {
1395     printf("Set complement operation:\n");
1396     printf("  Original set: { ");
1397
1398     int count = 0; /* Using count to track elements */
1399     for (int i = 0; i < size; i++) {
1400         if (set[i]) {
1401             printf("%d ", i);
1402             count++; /* Count elements in the set */
1403         }
1404     }
1405     printf("} (count: %d)\n", count);
1406
1407     printf("  Complement: { ");
1408     count = 0; /* Reset count for complement set */
1409
1410     /* Calculate set complement (elements in universal set but not in given set) */
1411     for (int i = 0; i < size; i++) {
1412         /* Comparison operator used to check set membership */
1413         if (universal[i] && !set[i]) {
1414             result[i] = true;
1415             printf("%d ", i);
1416             count++; /* Count elements in the complement */
1417         } else {
1418             result[i] = false;
1419         }
1420     }
1421     printf("} (count: %d)\n", count);
1422 }
1423
1424 /*
1425  * Function to evaluate a polynomial expression
1426  * Demonstrates coefficients in polynomial expressions
1427  */
1428 double evaluate_polynomial(Polynomial* poly, double x) {
1429     double result = 0.0;
1430
1431     printf("Evaluating polynomial with coefficients: ");
1432     for (int i = 0; i <= poly->degree; i++) {
1433         printf("%.2f", poly->coefficients[i]);
1434         if (i > 0) {
1435             printf("x^%d", i);
1436         }
1437         if (i < poly->degree) {
1438             printf(" + ");
1439         }
1440     }
1441     printf("\n");
1442
1443     /* Calculate polynomial value using Horner's method */
1444     for (int i = poly->degree; i >= 0; i--) {
1445         result = result * x + poly->coefficients[i];
1446     }
1447
1448     return result;
1449 }
1450
1451 /*
1452  * Function to create a polynomial with specified coefficients
1453  */
1454 Polynomial* create_polynomial(double* coeffs, int degree) {
1455     Polynomial* poly = (Polynomial*)malloc(sizeof(Polynomial));
1456
1457     poly->degree = degree;
1458     poly->coefficients = (double*)malloc((degree + 1) * sizeof(double));
1459
1460     /* Copy coefficients */
1461     for (int i = 0; i <= degree; i++) {
1462         poly->coefficients[i] = coeffs[i];
1463     }
1464
1465     return poly;
1466 }

```

```

1467
1468 /*
1469  * Function to concatenate two strings
1470  * Demonstrates concatenation in string operations
1471  */
1472 char* concatenate_strings(const char* str1, const char* str2) {
1473     /* Calculate the length of the concatenated string */
1474     size_t len1 = strlen(str1);
1475     size_t len2 = strlen(str2);
1476
1477     /* Allocate memory for the new string (plus space for null terminator) */
1478     char* result = (char*)malloc(len1 + len2 + 1);
1479
1480     /* Copy the first string using character-wise copying */
1481     for (size_t i = 0; i < len1; i++) {
1482         result[i] = str1[i]; /* Character-by-character copy */
1483     }
1484
1485     /* Append the second string */
1486     for (size_t i = 0; i < len2; i++) {
1487         result[len1 + i] = str2[i];
1488     }
1489
1490     /* Add null terminator */
1491     result[len1 + len2] = '\0';
1492
1493     return result;
1494 }
1495
1496 /*
1497  * Function to calculate distance between two coordinates
1498  * Demonstrates coordinates in geometric positioning
1499  */
1500 double calculate_distance(Coordinate point1, Coordinate point2) {
1501     /* Calculate differences in x and y coordinates */
1502     double dx = point2.x - point1.x;
1503     double dy = point2.y - point1.y;
1504
1505     /* Calculate Euclidean distance */
1506     return sqrt(dx*dx + dy*dy);
1507 }
1508
1509 /*
1510  * Function to calculate the midpoint between two coordinates
1511  */
1512 Coordinate calculate_midpoint(Coordinate point1, Coordinate point2) {
1513     Coordinate midpoint;
1514     midpoint.x = (point1.x + point2.x) / 2.0;
1515     midpoint.y = (point1.y + point2.y) / 2.0;
1516     return midpoint;
1517 }
1518
1519 /*
1520  * Function to simulate CPU cycles for a task
1521  * Demonstrates clock cycle in CPU timing
1522  */
1523 double simulate_cpu_execution(int instruction_count, double clock_speed_mhz) {
1524     /* Calculate cycles per instruction (CPI) - assume average CPI of 2.5 */
1525     double cpi = 2.5;
1526
1527     /* Calculate total cycle count */
1528     double total_cycles = instruction_count * cpi;
1529
1530     /* Calculate execution time in nanoseconds */
1531     double cycle_time_ns = 1000.0 / clock_speed_mhz; /* Time per cycle in ns */
1532     double execution_time_ns = total_cycles * cycle_time_ns;
1533
1534     printf("CPU execution timing:\n");
1535     printf("  Instructions: %d\n", instruction_count);
1536     printf("  Clock speed: %.1f MHz\n", clock_speed_mhz);
1537     printf("  Cycles per instruction: %.1f\n", cpi);
1538     printf("  Total cycles: %.1f\n", total_cycles);
1539     printf("  Cycle time: %.3f ns\n", cycle_time_ns);

```

```

1540     printf("  Execution time: %.3f ns (%.6f ms)\n",
1541           execution_time_ns, execution_time_ns / 1000000.0);
1542
1543     return execution_time_ns;
1544 }
1545
1546 /*
1547  * Function to implement a simple cache simulator
1548  * Demonstrates cache in memory hierarchy
1549  */
1550 void simulate_cache(int* memory, int memory_size, int cache_size) {
1551     /* Create a simple direct-mapped cache */
1552     int* cache = (int*)malloc(cache_size * sizeof(int));
1553     int* cache_tags = (int*)malloc(cache_size * sizeof(int));
1554     bool* cache_valid = (bool*)malloc(cache_size * sizeof(bool));
1555
1556     /* Clear the cache by setting valid bits to false */
1557     for (int i = 0; i < cache_size; i++) {
1558         cache_valid[i] = false; /* Clear operation for cache entries */
1559     }
1560
1561     /* Statistics */
1562     int access_count = 0;
1563     int hit_count = 0;
1564
1565     printf("Cache simulation starting (size: %d entries)\n", cache_size);
1566
1567     /* Simulate memory accesses with a simple pattern */
1568     for (int i = 0; i < 100; i++) {
1569         /* Calculate memory address to access (simulate some locality) */
1570         int addr = rand() % memory_size;
1571         if (rand() % 10 < 8) { /* 80% chance to access recent location */
1572             addr = (addr + 1) % memory_size;
1573         }
1574
1575         int cache_index = addr % cache_size; /* Simple direct mapping */
1576         int tag = addr / cache_size;
1577
1578         access_count++;
1579
1580         if (cache_valid[cache_index] && cache_tags[cache_index] == tag) {
1581             /* Cache hit */
1582             hit_count++;
1583             printf("  Access %3d: Address %3d - Cache HIT (index: %d)\n",
1584                   access_count, addr, cache_index);
1585         } else {
1586             /* Cache miss - load from memory */
1587             cache[cache_index] = memory[addr];
1588             cache_tags[cache_index] = tag;
1589             cache_valid[cache_index] = true;
1590             printf("  Access %3d: Address %3d - Cache MISS (loaded to index: %d)\n",
1591                   access_count, addr, cache_index);
1592         }
1593
1594         /* Only show first 10 accesses in detail */
1595         if (i == 9) {
1596             printf("    ... remaining accesses omitted for brevity ...\n");
1597         }
1598     }
1599
1600     double hit_rate = (double)hit_count / access_count * 100.0;
1601     printf("  Final cache hit rate: %d/%d (%.1f%%)\n",
1602           hit_count, access_count, hit_rate);
1603
1604     /* Clean up */
1605     free(cache);
1606     free(cache_tags);
1607     free(cache_valid);
1608 }
1609
1610 /*
1611  * Function to calculate a simple 16-bit checksum
1612  * Demonstrates checksum in data integrity

```

```

1613 */
1614 uint16_t calculate_checksum(uint8_t* data, size_t length, uint16_t init) {
1615     uint16_t checksum = init;
1616
1617     /* Process data in 8-bit chunks */
1618     for (size_t i = 0; i < length; i++) {
1619         /* Add each byte to the checksum */
1620         checksum += data[i];
1621
1622         /* Handle overflow with wrap-around */
1623         if (checksum < data[i]) {
1624             checksum++; /* Add carry to the result */
1625         }
1626     }
1627
1628     return ~checksum; /* Return one's complement */
1629 }
1630
1631 /*
1632  * Function to verify data integrity using checksum
1633  */
1634 bool verify_checksum(uint8_t* data, size_t length, uint16_t checksum, uint16_t init) {
1635     /* Calculate checksum of received data */
1636     uint16_t calculated = calculate_checksum(data, length, init);
1637
1638     /* Compare with expected checksum */
1639     return calculated == checksum;
1640 }
1641
1642 /*
1643  * Function to simulate a processor with a counter register
1644  * Demonstrates counter register in processor architecture
1645  */
1646 void simulate_counter_register(ProcessorRegisters* proc, int iterations) {
1647     printf("Counter register simulation:\n");
1648
1649     /* Clear the counter register initially */
1650     proc->counter_register = 0; /* Clear operation for register */
1651
1652     printf("  Initial counter value: %u\n", proc->counter_register);
1653
1654     /* Simulate instruction execution with counter increments */
1655     for (int i = 0; i < iterations; i++) {
1656         /* Perform some operation (simulated) */
1657         proc->general_purpose[0] += 1;
1658
1659         /* Increment the counter register */
1660         proc->counter_register++;
1661
1662         if (i < 5 || i > iterations - 3) {
1663             printf("  Iteration %d: Counter value = %u\n",
1664                 i, proc->counter_register);
1665         } else if (i == 5) {
1666             printf("  ... (intermediate iterations) ...\n");
1667         }
1668     }
1669
1670     printf("  Final counter value: %u\n", proc->counter_register);
1671 }
1672
1673 /*
1674  * Function to calculate compression ratio for run-length encoding
1675  * Demonstrates compression ratio in data compression
1676  */
1677 double calculate_rle_compression(const char* data) {
1678     size_t original_size = strlen(data);
1679     if (original_size == 0) return 0.0;
1680
1681     /* Estimate compressed size using run-length encoding */
1682     size_t compressed_size = 0;
1683     char current = data[0];
1684     int run_length = 1;
1685

```



```

1686     for (size_t i = 1; i <= original_size; i++) {
1687         if (i < original_size && data[i] == current) {
1688             run_length++;
1689         } else {
1690             /* End of run */
1691             if (run_length > 3) {
1692                 /* Format: count + character (2 bytes) */
1693                 compressed_size += 2;
1694             } else {
1695                 /* Literal characters */
1696                 compressed_size += run_length;
1697             }
1698
1699             if (i < original_size) {
1700                 current = data[i];
1701                 run_length = 1;
1702             }
1703         }
1704     }
1705
1706     /* Calculate compression ratio */
1707     double ratio = (double)original_size / compressed_size;
1708
1709     printf("Run-length encoding compression:\n");
1710     printf("  Original data: \"%s\"\n", data);
1711     printf("  Original size: %zu bytes\n", original_size);
1712     printf("  Estimated compressed size: %zu bytes\n", compressed_size);
1713     printf("  Compression ratio: %.2f:1\n", ratio);
1714
1715     return ratio;
1716 }
1717
1718 /*
1719  * Function to demonstrate control flow instructions
1720  * Shows control flow instructions in programming
1721  */
1722 int factorial_with_control_flow(int n) {
1723     printf("Factorial calculation with control flow:\n");
1724
1725     int result = 1;
1726     int i = 1;
1727
1728     while (true) { /* Infinite loop with conditional break */
1729         printf("  Iteration %d: result = %d * %d = ", i, result, i);
1730
1731         /* Multiply by current number */
1732         result *= i;
1733
1734         printf("%d\n", result);
1735
1736         i++;
1737
1738         /* Break statement - control flow instruction */
1739         if (i > n) {
1740             printf("  Break condition met (i > n), exiting loop\n");
1741             break;
1742         }
1743
1744         /* Continue statement - control flow instruction */
1745         if (result > 1000) {
1746             printf("  Result exceeds 1000, returning early\n");
1747             return result; /* Early return - control flow instruction */
1748         }
1749     }
1750
1751     return result;
1752 }
1753
1754 /*
1755  * Function to calculate current in a circuit using Ohm's Law
1756  * Demonstrates current in electrical circuit calculations
1757  */
1758 void calculate_circuit_properties(Circuit* circuit) {

```

```

1759     /* Apply Ohm's Law: I = V/R */
1760     circuit->current = circuit->voltage / circuit->resistance;
1761
1762     printf("Circuit calculation (Ohm's Law):\n");
1763     printf("  Voltage: %.2f V\n", circuit->voltage);
1764     printf("  Resistance: %.2f Ω\n", circuit->resistance);
1765     printf("  Current: %.2f A\n", circuit->current);
1766
1767     /* Calculate power: P = I²R or P = VI */
1768     double power = circuit->voltage * circuit->current;
1769     printf("  Power: %.2f W\n", power);
1770 }
1771
1772 /*
1773  * Function to perform deep copy of memory
1774  * Demonstrates copy operation in memory management
1775  */
1776 void* deep_copy_memory(void* source, size_t size) {
1777     /* Allocate new memory of the specified size */
1778     void* destination = malloc(size);
1779
1780     if (destination != NULL) {
1781         /* Copy memory content from source to destination */
1782         memcpy(destination, source, size);
1783     }
1784
1785     return destination;
1786 }
1787
1788 /*
1789  * Function to blend two colors with alpha
1790  * Demonstrates color value in graphics programming
1791  */
1792 ColorRGBA blend_colors(ColorRGBA color1, ColorRGBA color2, float blend_factor) {
1793     ColorRGBA result;
1794
1795     /* Ensure blend factor is between 0 and 1 */
1796     if (blend_factor < 0.0f) blend_factor = 0.0f;
1797     if (blend_factor > 1.0f) blend_factor = 1.0f;
1798
1799     /* Linear interpolation between color components */
1800     result.red   = (uint8_t)(color1.red   * (1 - blend_factor) + color2.red   * blend_factor);
1801     result.green = (uint8_t)(color1.green * (1 - blend_factor) + color2.green * blend_factor);
1802     result.blue  = (uint8_t)(color1.blue  * (1 - blend_factor) + color2.blue  * blend_factor);
1803     result.alpha = (uint8_t)(color1.alpha * (1 - blend_factor) + color2.alpha * blend_factor);
1804
1805     return result;
1806 }
1807
1808 /*
1809  * Function to print color as hexadecimal representation
1810  */
1811 void print_color(ColorRGBA color) {
1812     printf("#%02X%02X%02X%02X", color.red, color.green, color.blue, color.alpha);
1813 }
1814
1815 /* Main function demonstrating all concepts */
1816 int main() {
1817     srand(time(NULL));
1818
1819     printf("==== C Concepts Demonstration Program =====\n\n");
1820
1821     /* 1. Count in iterations or loops */
1822     printf("1. COUNT in iterations or loops:\n");
1823     int sum = 0;
1824     int count = 0; /* Initialize count variable */
1825
1826     for (int i = 1; i <= 10; i++) {
1827         sum += i;
1828         count++; /* Increment count for each iteration */
1829     }
1830
1831     printf("  Sum of numbers 1 to 10 = %d (calculated in %d iterations)\n\n", sum, count);

```

```

1832
1833 /* 2. Constant in mathematical equations */
1834 printf("2. CONSTANT in mathematical equations:\n");
1835 double radius = 5.0;
1836 double area = PI * radius * radius; /* PI is a constant */
1837
1838 printf("    Area of circle with radius %.1f = %.2f (using pi = %.5f)\n\n",
1839        radius, area, PI);
1840
1841 /* 3 & 7. Complement in set theory & Comparison operator */
1842 printf("3. COMPLEMENT in set theory with COMPARISON operators:\n");
1843
1844 /* Create universal set and a subset */
1845 bool universal_set[UNIVERSAL_SET_SIZE];
1846 bool set_a[UNIVERSAL_SET_SIZE];
1847 bool complement_a[UNIVERSAL_SET_SIZE];
1848
1849 /* Initialize universal set to all true */
1850 for (int i = 0; i < UNIVERSAL_SET_SIZE; i++) {
1851     universal_set[i] = true;
1852 }
1853
1854 /* Create set A with even numbers from 0 to 99 */
1855 for (int i = 0; i < UNIVERSAL_SET_SIZE; i++) {
1856     set_a[i] = (i % 2 == 0); /* Comparison operator to check even numbers */
1857 }
1858
1859 /* Calculate complement of set A */
1860 calculate_set_complement(set_a, universal_set, complement_a, UNIVERSAL_SET_SIZE);
1861 printf("\n");
1862
1863 /* 4. Carry bit in binary addition */
1864 printf("4. CARRY BIT in binary addition:\n");
1865 uint8_t num1[4] = {255, 128, 0, 50};
1866 uint8_t num2[4] = {1, 128, 200, 75};
1867 uint8_t result[4] = {0};
1868
1869 binary_add_bytes(num1, num2, result, 4);
1870 printf("\n");
1871
1872 /* 5. Character in string operations */
1873 printf("5. CHARACTER in string operations:\n");
1874 const char* text = "Hello, World!";
1875
1876 printf("    String: \"%s\"\n", text);
1877 printf("    Character by character: ");
1878
1879 for (int i = 0; text[i] != '\0'; i++) {
1880     printf('%c ', text[i]); /* Access individual characters */
1881 }
1882 printf("\n\n");
1883
1884 /* 6. Cache in memory hierarchy */
1885 printf("6. CACHE in memory hierarchy:\n");
1886
1887 /* Create a simulated memory area */
1888 int memory_size = 1000;
1889 int* memory = (int*)malloc(memory_size * sizeof(int));
1890
1891 /* Initialize memory with some values */
1892 for (int i = 0; i < memory_size; i++) {
1893     memory[i] = i * 10;
1894 }
1895
1896 /* Simulate cache operations */
1897 simulate_cache(memory, memory_size, CACHE_SIZE);
1898 printf("\n");
1899
1900 /* 8. Coordinate in geometric positioning */
1901 printf("8. COORDINATE in geometric positioning:\n");
1902 Coordinate point1 = {1.0, 2.0};
1903 Coordinate point2 = {4.0, 6.0};
1904

```

```

1905     printf("    Point 1: (%.1f, %.1f)\n", point1.x, point1.y);
1906     printf("    Point 2: (%.1f, %.1f)\n", point2.x, point2.y);
1907
1908     double distance = calculate_distance(point1, point2);
1909     printf("    Distance between points: %.2f\n", distance);
1910
1911     Coordinate midpoint = calculate_midpoint(point1, point2);
1912     printf("    Midpoint: (%.1f, %.1f)\n\n", midpoint.x, midpoint.y);
1913
1914     /* 9. Clear operation for registers or memory */
1915     printf("9. CLEAR operation for registers or memory:\n");
1916
1917     /* Create a memory block to demonstrate clearing */
1918     int* memory_block = (int*)malloc(10 * sizeof(int));
1919
1920     /* Initialize with non-zero values */
1921     for (int i = 0; i < 10; i++) {
1922         memory_block[i] = 100 + i;
1923     }
1924
1925     printf("    Memory before clearing: ");
1926     for (int i = 0; i < 10; i++) {
1927         printf("%d ", memory_block[i]);
1928     }
1929     printf("\n");
1930
1931     /* Clear the memory block by setting to zero */
1932     for (int i = 0; i < 10; i++) {
1933         memory_block[i] = 0;    /* Clear operation */
1934     }
1935
1936     printf("    Memory after clearing: ");
1937     for (int i = 0; i < 10; i++) {
1938         printf("%d ", memory_block[i]);
1939     }
1940     printf("\n\n");
1941
1942     /* 10. Clock cycle in CPU timing */
1943     printf("10. CLOCK CYCLE in CPU timing:\n");
1944     simulate_cpu_execution(1000, CLOCK_SPEED_MHZ);
1945     printf("\n");
1946
1947     /* 11. Coefficient in polynomial expressions */
1948     printf("11. COEFFICIENT in polynomial expressions:\n");
1949     double coeffs[] = {2.0, -3.0, 1.0};    /* 2 - 3x + x² */
1950     Polynomial* polynomial = create_polynomial(coeffs, 2);
1951
1952     double x_value = 2.0;
1953     double result = evaluate_polynomial(polynomial, x_value);
1954
1955     printf("    p(%.1f) = %.2f\n\n", x_value, result);
1956
1957     /* 12. Capacity in resource allocation */
1958     printf("12. CAPACITY in resource allocation:\n");
1959     printf("    System has maximum capacity of %d elements\n", MAX_CAPACITY);
1960
1961     /* Demonstrate allocation within capacity */
1962     int requested_size = 800;
1963
1964     printf("    Requested allocation: %d elements\n", requested_size);
1965
1966     if (requested_size <= MAX_CAPACITY) {
1967         printf("    Allocation successful (within capacity)\n");
1968     } else {
1969         printf("    Allocation failed (exceeds capacity)\n");
1970     }
1971
1972     /* Demonstrate allocation exceeding capacity */
1973     requested_size = 1200;
1974
1975     printf("    Requested allocation: %d elements\n", requested_size);
1976
1977     if (requested_size <= MAX_CAPACITY) {

```

```

1978     printf("    Allocation successful (within capacity)\n");
1979 } else {
1980     printf("    Allocation failed (exceeds capacity)\n");
1981 }
1982 printf("\n");
1983
1984 /* 13. Concatenation in string operations */
1985 printf("13. CONCATENATION in string operations:\n");
1986 const char* first_part = "Hello, ";
1987 const char* second_part = "world!";
1988
1989 printf("    First string: \"%s\"\n", first_part);
1990 printf("    Second string: \"%s\"\n", second_part);
1991
1992 char* combined = concatenate_strings(first_part, second_part);
1993
1994 printf("    Concatenated result: \"%s\"\n\n", combined);
1995
1996 /* 14. Checksum in data integrity */
1997 printf("14. CHECKSUM in data integrity:\n");
1998 uint8_t data[] = {'H', 'e', 'l', 'l', 'o', ' ', 'D', 'a', 't', 'a'};
1999 size_t data_length = sizeof(data);
2000
2001 printf("    Original data: \"\");
2002 for (size_t i = 0; i < data_length; i++) {
2003     printf("%c", data[i]);
2004 }
2005 printf("\n\n");
2006
2007 uint16_t data_checksum = calculate_checksum(data, data_length, CHECKSUM_INIT);
2008 printf("    Calculated checksum: 0x%04X\n", data_checksum);
2009
2010 /* Verify unchanged data */
2011 bool integrity_ok = verify_checksum(data, data_length, data_checksum, CHECKSUM_INIT);
2012 printf("    Data integrity check: %s\n", integrity_ok ? "PASSED" : "FAILED");
2013
2014 /* Modify data and check again */
2015 data[3] = 'x'; /* Change 'l' to 'x' */
2016 printf("    Modified data: \"\");
2017 for (size_t i = 0; i < data_length; i++) {
2018     printf("%c", data[i]);
2019 }
2020 printf("\n\n");
2021
2022 integrity_ok = verify_checksum(data, data_length, data_checksum, CHECKSUM_INIT);
2023 printf("    Data integrity check after modification: %s\n\n",
2024     integrity_ok ? "PASSED" : "FAILED");
2025
2026 /* 15. Counter register in processor architecture */
2027 printf("15. COUNTER REGISTER in processor architecture:\n");
2028 ProcessorRegisters processor = {0}; /* Initialize all registers to 0 */
2029 simulate_counter_register(&processor, 20);
2030 printf("\n");
2031
2032 /* 16. Compression ratio in data compression */
2033 printf("16. COMPRESSION RATIO in data compression:\n");
2034 const char* compress_data1 = "AAAAABBBBBCCCCDDDDDD";
2035 const char* compress_data2 = "ABCDEFGHIIJKLMNOPQRST";
2036
2037 printf("    Test case 1 (repeating characters):\n");
2038 calculate_rle_compression(compress_data1);
2039
2040 printf("    Test case 2 (unique characters):\n");
2041 calculate_rle_compression(compress_data2);
2042 printf("\n");
2043
2044 /* 17. Control flow instruction in programming */
2045 printf("17. CONTROL FLOW instruction in programming:\n");
2046 int n = 5;
2047 int fact = factorial_with_control_flow(n);
2048
2049 printf("    Final result: %d! = %d\n\n", n, fact);
2050

```

```

2051 /* 18. Current in electrical circuit calculations */
2052 printf("18. CURRENT in electrical circuit calculations:\n");
2053 Circuit circuit = {12.0, 4.0, 0.0}; /* Voltage = 12V, Resistance = 4Ω */
2054 calculate_circuit_properties(&circuit);
2055 printf("\n");
2056
2057 /* 19. Copy operation in memory management */
2058 printf("19. COPY operation in memory management:\n");
2059 int source_array[5] = {10, 20, 30, 40, 50};
2060
2061 printf("    Source array: ");
2062 for (int i = 0; i < 5; i++) {
2063     printf("%d ", source_array[i]);
2064 }
2065 printf("\n");
2066
2067 /* Perform deep copy */
2068 int* copy_array = (int*)deep_copy_memory(source_array, 5 * sizeof(int));
2069
2070 printf("    Copied array: ");
2071 for (int i = 0; i < 5; i++) {
2072     printf("%d ", copy_array[i]);
2073 }
2074 printf("\n");
2075
2076 /* Modify source to demonstrate independence */
2077 source_array[2] = 99;
2078
2079 printf("    Source after modification: ");
2080 for (int i = 0; i < 5; i++) {
2081     printf("%d ", source_array[i]);
2082 }
2083 printf("\n");
2084
2085 printf("    Copy after source modification: ");
2086 for (int i = 0; i < 5; i++) {
2087     printf("%d ", copy_array[i]);
2088 }
2089 printf("\n\n");
2090
2091 /* 20. Color value in graphics programming */
2092 printf("20. COLOR VALUE in graphics programming:\n");
2093
2094 ColorRGBA red = {255, 0, 0, 255}; /* Opaque red */
2095 ColorRGBA blue = {0, 0, 255, 255}; /* Opaque blue */
2096 ColorRGBA semi_transparent = {128, 128, 128, 128}; /* Semi-transparent gray */
2097
2098 printf("    Red color: ");
2099 print_color(red);
2100 printf("\n");
2101
2102 printf("    Blue color: ");
2103 print_color(blue);
2104 printf("\n");
2105
2106 /* Blend red and blue with different factors */
2107 printf("    Color blending:\n");
2108 for (int i = 0; i <= 10; i++) {
2109     float blend_factor = i / 10.0f;
2110     ColorRGBA blended = blend_colors(red, blue, blend_factor);
2111
2112     printf("    %.1f Red + %.1f Blue = ", 1.0f - blend_factor, blend_factor);
2113     print_color(blended);
2114     printf("\n");
2115 }
2116
2117 /* Clean up allocated memory */
2118 free(memory);
2119 free(memory_block);
2120 free(polynomial->coefficients);
2121 free(polynomial);
2122 free(combined);
2123 free(copy_array);

```

```
2124
2125     return 0;
2126 }
2127
2128 ]
2129
2130 4   (1 5){
2131     d
2132 }[
2133
2134 1   Delta (change in value)
2135 2   Denominator (in fractions)
2136 3   Decrement operation (decrease by one)
2137 4   Data register (in CPU architecture)
2138 5   Distance (in geometric calculations)
2139 6   Dimension (in array declarations)
2140 7   Division operation (in arithmetic)
2141 8   Derivative (in calculus)
2142 9   Depth (in tree structures)
2143 10  Destination (in memory transfers)
2144 11  Debug flag (in debugging tools)
2145 12  Default value (in parameter settings)
2146 13  Deletion operation (in data structures)
2147 14  Degree (in polynomial equations or angles)
2148 15  Density (in physical calculations)
2149 16  Double precision (in floating-point format)
2150 17  Duration (in time measurements)
2151 18  Directory (in file system operations)
2152 19  Displacement (in physics calculations)
2153 20  Digit (in numerical representation)
2154
2155 DEFINITIONS
2156
2157 1. Delta (change in value): The finite difference between two states of a variable, representing the magnitude and direction of quantitative change over a specified domain interval. It
measures the absolute variation between successive values in sequential processes or comparative analyses.
2158
2159 2. Denominator (in fractions): The divisor component positioned below the fraction bar in a rational number expression that indicates the number of equal parts into which the unit is
divided. It establishes the granularity of the fractional division and determines the magnitude of each fractional part.
2160
2161 3. Decrement operation (decrease by one): An arithmetic procedure that reduces a numerical value by exactly one unit, commonly implemented as a unary operation in programming
languages. It modifies the operand through subtraction of unity while preserving the variable's data type.
2162
2163 4. Data register (in CPU architecture): A processor-internal storage element of fixed bit width designated for holding operands, intermediate results, and computational products during
instruction execution. It provides high-speed access to data values within the central processing unit execution pipeline.
2164
2165 5. Distance (in geometric calculations): A non-negative scalar measure quantifying the spatial separation between two points in a metric space according to a defined distance function.
It represents the minimum path length connecting two positions in accordance with the applicable geometric principles.
2166
2167 6. Dimension (in array declarations): The number of indices required to specify a unique element within a multidimensional array structure, representing the distinct ordinal
hierarchies of the array's organization. It determines both the addressing complexity and the organizational topology of stored elements.
2168
2169 7. Division operation (in arithmetic): A binary mathematical procedure that computes the quotient of two values, determining how many times the divisor is contained within the
dividend. It represents the inverse of multiplication and distributes the dividend into equal portions specified by the divisor.
2170
2171 8. Derivative (in calculus): The instantaneous rate of change of a function with respect to one of its variables, representing the slope of the tangent line at a specific point on the
function's graph. It quantifies the sensitivity of the dependent variable to infinitesimal changes in the independent variable.
2172
2173 9. Depth (in tree structures): The length of the path from the root node to a specified node within a hierarchical tree data structure, measured by counting the number of edges
traversed. It quantifies the vertical position of a node within the tree's layered organization.
2174
2175 10. Destination (in memory transfers): The target location specified as the recipient of data during a movement operation between storage locations. It denotes the memory address,
register, or device where transferred information will reside following the completion of the data transfer instruction.
2176
2177 11. Debug flag (in debugging tools): A conditional indicator that can be programmatically set or cleared to control the execution of diagnostic code sections or the generation of
intermediate state information. It enables selective activation of debugging functionality without modifying primary program logic.
2178
2179 12. Default value (in parameter settings): A predefined constant assigned to a variable, parameter, or field when no explicit value is provided by the user or calling process. It
establishes initialization behavior and maintains operational consistency in the absence of specific configuration.
2180
2181 13. Deletion operation (in data structures): A structural modification procedure that removes a specified element from a collection while maintaining the integrity and organizational
properties of the data structure. It adjusts internal references and reestablishes connectivity between remaining elements.
2182
2183 14. Degree (in polynomial equations or angles): In polynomial contexts, the highest exponent applied to the variable in the expression; in angular measurement, the unit equal to 1/360
```

```

of a complete rotation around a circle. It quantifies computational complexity or rotational displacement respectively.
2184
2185 15. Density (in physical calculations): The ratio of an object's mass to its volume, expressing the compactness of matter within spatial boundaries. It characterizes material
properties by quantifying the concentration of mass per unit volume within a substance or object.
2186
2187 16. Double precision (in floating-point format): A numerical representation format that allocates approximately twice the number of bits for storing floating-point values compared to
single precision, typically conforming to IEEE 754 binary64 standard. It provides extended range and precision for representing real numbers in computational systems.
2188
2189 17. Duration (in time measurements): The continuous temporal interval between two defined instants, representing the persistence of an event, process, or state. It quantifies elapsed
time as a scalar quantity measurable in standardized chronological units.
2190
2191 18. Directory (in file system operations): A specialized file containing metadata entries that associate filenames with their corresponding file system locations and attributes. It
implements hierarchical organization of data storage by providing a container mechanism for related files and subdirectories.
2192
2193 19. Displacement (in physics calculations): A vector quantity representing both the straight-line distance and direction between an object's initial position and its final position,
regardless of the actual path traversed. It captures net positional change in spatial coordinates over a specified time interval.
2194
2195 20. Digit (in numerical representation): An atomic symbolic element used within a positional numbering system to represent quantities according to place value rules. It constitutes the
fundamental character set for expressing numbers within a given numerical base or radial system.
2196
2197 ]
2198
2199 5 (1 6){
2200 e
2201 }[
2202
2203 1 Exponential constant (271828)
2204 2 Element (in set theory or arrays)
2205 3 Expression (in programming languages)
2206 4 Edge (in graph theory)
2207 5 Error value (in error handling)
2208 6 Exponent (in floating-point representation)
2209 7 Equality comparison (in boolean operations)
2210 8 Entry point (in program execution)
2211 9 Extension register (in CPU architecture)
2212 10 Encryption key (in cryptography)
2213 11 Event handler (in event-driven programming)
2214 12 Escape sequence (in string formatting)
2215 13 Enumeration type (in type systems)
2216 14 Euler's method parameter (in numerical analysis)
2217 15 Energy (in physics calculations)
2218 16 Expansion factor (in data structures)
2219 17 Evaluation metric (in machine learning)
2220 18 Epsilon value (small constant in numerical methods)
2221 19 Exit code (in process termination)
2222 20 Endpoint (in networking or ranges)
2223
2224 DEFINITIONS
2225
2226 1. Exponential constant (2.71828): The irrational mathematical constant representing the base of the natural logarithm, defined as the limit of (1 + 1/n)^n as n approaches infinity. It
serves as the foundation for exponential growth models and appears as the unique number whose natural logarithm equals one.
2227
2228 2. Element (in set theory or arrays): A discrete object or value that belongs to a collection, where membership is defined by inclusion within the specified set or by occupation of an
indexed position within an array. It constitutes an individual component subject to the operations applicable to the containing structure.
2229
2230 3. Expression (in programming languages): A combination of values, variables, operators, and function invocations that follows syntactic rules to specify a computation yielding a
single result value. It encodes a sequence of operations that produces a deterministic output when evaluated in a given context.
2231
2232 4. Edge (in graph theory): A connection between two vertices in a graph structure that establishes a relationship or pathway between the connected nodes. It represents a binary
association that may possess directionality and weight attributes depending on the graph type.
2233
2234 5. Error value (in error handling): A specialized return value or object that signifies the occurrence of an exceptional condition or operational failure during program execution. It
communicates fault information to enable appropriate remediation or graceful degradation in response to anomalous states.
2235
2236 6. Exponent (in floating-point representation): The component in a floating-point number that specifies the power to which the implicit base is raised, determining the scale factor
applied to the significand. It controls the numerical range by indicating the position of the decimal or binary point.
2237
2238 7. Equality comparison (in boolean operations): A relational operation that determines whether two values possess identical content according to type-specific equivalence rules,
producing a truth value indicating complete correspondence. It implements the mathematical concept of equivalence relation in computational logic.
2239
2240 8. Entry point (in program execution): The instruction address where execution of a program or subroutine begins, marking the initial control transfer location when a module is
invoked. It serves as the designated commencement position for procedural flow in executable code sections.
2241
```


2242 9. Extension register (in CPU architecture): A supplementary processor register that expands the standard register set to provide additional storage capacity or specialized functionality. It augments the computational capabilities of the central processing unit through extended operand accessibility.

2243

2244 10. Encryption key (in cryptography): A parameter that controls the transformation of plaintext into ciphertext through a cryptographic algorithm, determining the specific permutation or substitution pattern applied. It establishes the security foundation by enabling only authorized parties to perform decryption.

2245

2246 11. Event handler (in event-driven programming): A function or method designated to respond when a specific event occurs within a software system, encapsulating the processing logic for the associated event type. It implements the observer pattern by associating computational responses with detected events.

2247

2248 12. Escape sequence (in string formatting): A combination of characters beginning with a designated escape character that specifies a non-literal interpretation of subsequent characters in text processing. It enables representation of control characters, special symbols, or formatting directives within string literals.

2249

2250 13. Enumeration type (in type systems): A data type consisting of a set of named constant values, providing a mechanism for defining categorical variables with a restricted range of possible states. It establishes type safety for values representing distinct classifications or modes.

2251

2252 14. Euler's method parameter (in numerical analysis): A step size coefficient that controls the granularity of approximation in the numerical integration of ordinary differential equations using Euler's method. It determines the trade-off between computational efficiency and approximation accuracy.

2253

2254 15. Energy (in physics calculations): A scalar quantity representing the capacity of a physical system to perform work, measured in joules within the International System of Units. It manifests in various forms including kinetic, potential, thermal, electrical, and chemical energy, subject to conservation principles.

2255

2256 16. Expansion factor (in data structures): A multiplicative coefficient that determines the increase in capacity when a dynamic data structure requires resizing to accommodate additional elements. It controls the trade-off between memory efficiency and reallocation frequency during growth operations.

2257

2258 17. Evaluation metric (in machine learning): A quantitative measure that assesses the performance or quality of a predictive model according to a specific aspect of its behavior on input data. It provides an objective function for comparing model effectiveness and guiding optimization processes.

2259

2260 18. Epsilon value (small constant in numerical methods): An arbitrarily small positive quantity used to establish convergence thresholds, prevent division by zero, or define proximity in floating-point comparisons. It accommodates computational limitations by formalizing acceptable approximation boundaries.

2261

2262 19. Exit code (in process termination): An integer value returned by a program to its parent process or operating system upon completion, conveying information about the execution outcome. It communicates success, failure, or specific termination conditions through standardized or application-defined status codes.

2263

2264 20. Endpoint (in networking or ranges): In networking contexts, a communication termination point identified by an address and port; in ranges, the boundary value that defines the extreme limit of an interval. It establishes the terminus of a communication channel or the inclusive/exclusive boundary of a continuous set.

2265

2266]

2267

2268 6 (1 7){

2269 f

2270 }[

2271

2272 1 Function (in mathematical operations)

2273 2 Flag register (in CPU states)

2274 3 Frequency (in signal processing)

2275 4 Float value (in numeric data types)

2276 5 File descriptor (in I/O operations)

2277 6 Frame pointer (in stack frames)

2278 7 Feedback parameter (in control systems)

2279 8 Format specifier (in output formatting)

2280 9 Filter operation (in data processing)

2281 10 Force (in physics calculations)

2282 11 Field (in data structures or records)

2283 12 Factor (in factorization)

2284 13 Fetch operation (in CPU instruction cycle)

2285 14 FIFO queue (first-in-first-out data structure)

2286 15 Flip operation (bit inversion)

2287 16 Front index (in queue implementations)

2288 17 Fractional part (in decimal numbers)

2289 18 Feature vector (in machine learning)

2290 19 Fork process (in parallel computing)

2291 20 Fibonacci sequence term (in recursive algorithms)

2292

2293 DEFINITIONS

2294

2295 1. Function (in mathematical operations): A relation between sets that associates each element of the domain with exactly one element in the codomain, specifying a computational procedure that transforms input values into deterministic output values. It establishes a systematic mapping that preserves the uniqueness of outputs for given inputs.

2296

2297 2. Flag register (in CPU states): A specialized processor register containing individual boolean indicators that reflect the current operational state and the results of recent computations. It maintains status bits that signal conditions such as zero result, carry generation, overflow detection, and negative values for conditional branch decision-making.

2298

2299 3. Frequency (in signal processing): The rate at which a periodic signal completes a full oscillation cycle per unit time, typically measured in hertz. It quantifies the temporal density of repetitive patterns in waveforms and determines fundamental properties of signals in both time and frequency domains.

2300

2301

2302

2303

2304

2305

2306

2307

2308

2309

2310

2311

2312

2313

2314

2315

2316

2317

2318

2319

2320

2321

2322

2323

2324

2325

2326

2327

2328

2329

2330

2331

2332

2333

2334

2335

2336

2337

2338

2339

2340

2341

2342

2343

2344

2345

2346

2347

2348

2349

2350

2351

2352

2353

2354

2355

4. Float value (in numeric data types): A machine representation of a real number using a finite binary encoding that separates the significant digits from the decimal scaling factor through a format containing sign, exponent, and mantissa components. It enables approximate representation of continuous numerical values within digital systems.

5. File descriptor (in I/O operations): An abstract numeric handle generated by the operating system that uniquely identifies an open file or input/output resource within a process. It serves as an index into the process file table for directing subsequent read, write, and control operations to the correct system resource.

6. Frame pointer (in stack frames): A dedicated register that maintains a reference to the base address of the current procedure's activation record in the call stack. It provides stable addressing for local variables and parameters regardless of dynamic stack modifications during function execution.

7. Feedback parameter (in control systems): A coefficient that determines how strongly a measured output deviation affects subsequent control inputs in a closed-loop system. It quantifies the reactive adjustment strength and influences system stability, response time, and error correction behavior.

8. Format specifier (in output formatting): A syntactical construct within a format string that defines how a corresponding argument should be converted, formatted, and presented in the output text. It prescribes type interpretation, alignment, precision, and presentation style for data values during textual rendering.

9. Filter operation (in data processing): A transformation that selectively modifies or excludes elements from a data stream based on predefined criteria. It implements selective information transmission by attenuating unwanted components while preserving or enhancing desired characteristics of the input.

10. Force (in physics calculations): A vector quantity that causes an object with mass to accelerate, measured in newtons in the International System of Units. It represents the rate of change of momentum and manifests as a push, pull, or interaction that alters the motion state of an object.

11. Field (in data structures or records): A designated storage location within a composite data structure that contains a specific attribute or property of the entity represented by the record. It establishes typed data compartmentalization within structured information aggregates.

12. Factor (in factorization): A divisor that produces an integer quotient when dividing another number, or a component expression that, when multiplied with other factors, generates the original expression. It represents a fundamental constituent in decomposition of numbers or algebraic expressions.

13. Fetch operation (in CPU instruction cycle): The initial phase of instruction execution wherein the processor retrieves the next instruction from memory at the address specified by the program counter. It transfers the instruction encoding from memory to the instruction register for subsequent decoding and execution.

14. FIFO queue (first-in-first-out data structure): A sequential collection that constrains element access such that the earliest added element must be processed before elements added subsequently. It implements temporal ordering through strict adherence to chronological insertion sequence during removal operations.

15. Flip operation (bit inversion): A unary bitwise transformation that reverses the state of each binary digit, changing ones to zeros and zeros to ones. It implements logical negation at the bit level by complementing individual bit values throughout a binary word.

16. Front index (in queue implementations): A positional indicator that references the location of the oldest element in a queue data structure, identifying the next item to be removed. It maintains a reference to the head position for dequeue operations in sequential access patterns.

17. Fractional part (in decimal numbers): The portion of a real number that appears after the decimal point, representing values less than one in the number's composition. It quantifies the non-integer component as decimal fractions of unity within the positional notation system.

18. Feature vector (in machine learning): An ordered collection of numerical attributes that characterizes an observation instance by quantifying its relevant properties. It transforms raw data into a standardized representation suitable for algorithmic processing within statistical learning frameworks.

19. Fork process (in parallel computing): A system call that creates a new process by duplicating the calling process, with execution continuing in both the parent and child processes from the point of invocation. It enables concurrent execution paths by establishing process-level parallelism through replication.

20. Fibonacci sequence term (in recursive algorithms): An element within the integer sequence where each number equals the sum of the two preceding numbers, beginning with zero and one. It exemplifies recursive definition through its self-referential construction rule applicable to computing arbitrary sequence positions.

]

7 (1 8){

g

}[

1 Gravitational constant (in physics calculations)

2 Gradient (in vector calculus)

3 General-purpose register (in CPU architecture)

4 Graph (in data structures)

5 Growth rate (in algorithm analysis)

6 Global variable (in programming scopes)

7 Gain factor (in signal processing)

8 Greater than comparison (in relational operations)

9 Grid coordinate (in spatial indexing)

10 Generator function (in iterative processing)

11 Glyph index (in typography)

12 Goto instruction (in control flow)

13 Guard condition (in state machines)

14 Group operation (in algebraic structures)

15 Geometric mean (in statistics)

235616 Gate signal (in digital logic)

235717 Ground reference (in electrical circuits)

235818 Gamma function parameter (in calculus)

235919 Granularity level (in parallel processing)

236020 Greatest common divisor (in number theory)

2361

2362DEFINITIONS

2363

23641. Gravitational constant (in physics calculations): The fundamental physical constant characterizing the strength of gravitational attraction between bodies, with an approximate value of 6.67430×10^-11 cubic meters per kilogram per second squared in the International System of Units. It defines the proportionality between gravitational force and the product of masses divided by the square of the distance in the universal law of gravitation.

2365

23662. Gradient (in vector calculus): A vector differential operator that maps a scalar field to its corresponding vector field, composed of partial derivatives with respect to each coordinate direction. It represents the direction and magnitude of the maximum rate of change of a multivariable function at each point in space.

2367

23683. General-purpose register (in CPU architecture): A high-speed storage location within the central processing unit designed to hold data, addresses, or intermediate results accessible to the arithmetic logic unit for computational operations. It provides versatile temporary storage for operands and results during instruction execution.

2369

23704. Graph (in data structures): A non-linear data structure comprising a finite set of vertices connected by edges, representing binary relationships between discrete entities. It models complex networks through node connectivity patterns that can incorporate directionality, weights, and cyclical relationships.

2371

23725. Growth rate (in algorithm analysis): The asymptotic behavior of resource consumption as input size approaches infinity, typically expressed using big O notation. It characterizes algorithm efficiency by quantifying how execution time or space requirements scale with increasing problem complexity.

2373

23746. Global variable (in programming scopes): A data storage entity declared outside any procedural scope, accessible throughout the entire program without explicit parameter passing. It maintains its value and accessibility across function boundaries and throughout program execution lifetime.

2375

23767. Gain factor (in signal processing): A multiplicative coefficient applied to a signal that amplifies or attenuates its amplitude without altering other characteristics. It controls signal strength by scaling input-to-output magnitude ratios in linear systems.

2377

23788. Greater than comparison (in relational operations): A binary operation that evaluates whether the first operand exceeds the second operand according to a defined ordering relation, producing a Boolean result. It implements strict ordering tests based on numerical value, lexicographical sequence, or other comparable attributes.

2379

23809. Grid coordinate (in spatial indexing): A tuple of discrete values that specifies the position of a point relative to a regular subdivision of space. It enables efficient spatial data organization through cellular decomposition of coordinate systems into addressable units.

2381

238210. Generator function (in iterative processing): A specialized function that yields a sequence of values incrementally while preserving execution state between invocations. It implements lazy evaluation by suspending execution after each value production until the next value is requested.

2383

238411. Glyph index (in typography): A numerical identifier that references a specific character representation within a font or character set. It provides direct access to the visual representation of a character through an indexing scheme independent of character encoding.

2385

238612. Goto instruction (in control flow): An unconditional branch operation that transfers program execution to a specified labeled location in the code. It implements non-structured control flow by directly modifying the program counter to point to the target instruction address.

2387

238813. Guard condition (in state machines): A Boolean expression evaluated during a state transition that must evaluate to true for the transition to occur. It constrains state changes by enforcing conditional requirements beyond simple event triggering.

2389

239014. Group operation (in algebraic structures): A binary function that combines two elements of a set to produce a third element satisfying closure, associativity, identity, and invertibility properties. It defines the fundamental combination method that characterizes the algebraic structure of a group.

2391

239215. Geometric mean (in statistics): The nth root of the product of n non-negative real numbers, representing the central tendency of values that are naturally multiplicative rather than additive. It measures typical values in data sets where proportional changes are more significant than absolute differences.

2393

239416. Gate signal (in digital logic): A control pulse that enables or disables the passage of another signal through a circuit element during a specified time interval. It implements temporal selection by conditionally allowing information transfer based on the gate signal state.

2395

239617. Ground reference (in electrical circuits): A common reference point in an electrical system designated as zero potential against which all other voltages are measured. It establishes a baseline potential for circuit analysis and provides a return path for electrical current flow.

2397

239818. Gamma function parameter (in calculus): A complex number input to the gamma function, which extends the factorial operation to non-integer values through an improper integral definition. It serves as the independent variable in this special function central to mathematical analysis.

2399

240019. Granularity level (in parallel processing): The size of computational units into which a problem is decomposed for concurrent execution across multiple processing elements. It determines the ratio between computation and communication overhead in parallel algorithms.

2401

240220. Greatest common divisor (in number theory): The largest positive integer that divides each of the given integers without remainder. It quantifies the shared factors between numbers and forms the foundation for fraction simplification and modular arithmetic operations.

2403

2404]

2405

24068 (9) {

2407h

```
2408 }[
2409
2410 1   Height (in geometric calculations)
2411 2   Hash value (in hash functions)
2412 3   Horizontal coordinate (in coordinate systems)
2413 4   Hexadecimal digit (in number representation)
2414 5   Header pointer (in linked data structures)
2415 6   Halt instruction (in program execution)
2416 7   Hold register (in CPU operations)
2417 8   High bit (in binary representation)
2418 9   Hypothesis (in statistical testing)
2419 10  Hamming distance (in information theory)
2420 11  Handle (to system resources)
2421 12  Heap allocation (in memory management)
2422 13  Hour (in time calculations)
2423 14  Hyperparameter (in machine learning)
2424 15  Heuristic value (in search algorithms)
2425 16  Host address (in networking)
2426 17  Harmonic mean (in statistics)
2427 18  Heat transfer coefficient (in thermodynamics)
2428 19  Hidden layer node (in neural networks)
2429 20  Homogeneous coordinate (in computer graphics)
2430
2431 DEFINITIONS
2432
2433 1. Height (in geometric calculations): A perpendicular measurement from the base to the most distant point of a geometric object, representing the maximum vertical extent when oriented
    in standard position. It quantifies the orthogonal dimension that contributes to area and volume calculations in various geometric forms.
2434
2435 2. Hash value (in hash functions): A fixed-length numeric or alphanumeric string generated from input data of arbitrary size through a deterministic mathematical transformation. It
    creates a compressed digest that serves as a content identifier for efficient data retrieval, integrity verification, and cryptographic applications.
2436
2437 3. Horizontal coordinate (in coordinate systems): The positional value along the primary axis that runs left to right in a two-dimensional reference frame. It specifies the lateral
    displacement of a point from the vertical reference axis within the coordinate plane.
2438
2439 4. Hexadecimal digit (in number representation): A single character from the set of sixteen symbols {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F} used in base-16 numerical notation. It represents
    four binary digits as a single character, facilitating compact representation of binary data in human-readable form.
2440
2441 5. Header pointer (in linked data structures): A reference variable that maintains the memory address of the first node in a linked structure, providing the initial access point for
    traversal operations. It serves as the primary entry point that enables navigation through the entire linked sequence.
2442
2443 6. Halt instruction (in program execution): A machine language operation that terminates program execution by placing the processor in an idle state, preventing further instruction
    processing. It signals intentional execution completion rather than error-based termination.
2444
2445 7. Hold register (in CPU operations): A specialized storage location that temporarily preserves an operand value during multi-stage instruction execution. It maintains intermediate
    data between processing steps when the source register may be modified before the operation completes.
2446
2447 8. High bit (in binary representation): The most significant bit in a binary sequence that contributes the largest value to the numeric interpretation and indicates the sign in signed
    number representations. It occupies the leftmost position when written in conventional notation.
2448
2449 9. Hypothesis (in statistical testing): A formal assertion about a population parameter subject to validation through empirical evidence and probability-based evaluation. It
    establishes a proposed explanation or expected relationship to be confirmed or refuted through statistical analysis of sample data.
2450
2451 10. Hamming distance (in information theory): The number of positions at which corresponding symbols differ between two strings of equal length. It quantifies the minimum number of
    substitutions required to transform one string into another, serving as a metric for error detection and correction.
2452
2453 11. Handle (to system resources): An abstract reference identifier provided by an operating system or runtime environment that encapsulates access permissions to a protected resource.
    It mediates interactions with system objects while concealing implementation details and maintaining access control.
2454
2455 12. Heap allocation (in memory management): The dynamic reservation of memory blocks from the unstructured memory pool during program execution, performed explicitly through
    programmatic requests rather than automatic stack allocation. It enables flexible memory utilization for variable-sized data with lifetimes not bound to lexical scope.
2456
2457 13. Hour (in time calculations): A fundamental chronological unit equal to 3600 seconds or 1/24 of a solar day in standard timekeeping systems. It serves as an intermediate temporal
    measure between minutes and days for expressing event duration and scheduling.
2458
2459 14. Hyperparameter (in machine learning): A configuration variable external to the model that cannot be learned from training data but must be specified before the learning process
    begins. It controls aspects of model architecture, optimization behavior, and regularization strength that influence the learning trajectory.
2460
2461 15. Heuristic value (in search algorithms): A problem-specific estimation function that approximates the distance or cost from the current state to the goal state in optimization
    problems. It guides search strategies by providing informed prioritization of exploration paths without guaranteeing optimality.
2462
2463 16. Host address (in networking): A numerical identifier assigned to a network interface that uniquely specifies a device endpoint within a defined network topology. It enables precise
    message routing and delivery to specific machines connected to the communication infrastructure.
2464
```

```
2465 17. Harmonic mean (in statistics): The reciprocal of the arithmetic mean of the reciprocals of a data set, calculated as the number of observations divided by the sum of reciprocals.
2466 It appropriately represents central tendency when dealing with rates, ratios, or quantities where the relationship is inversely proportional.
2467
2468 18. Heat transfer coefficient (in thermodynamics): A proportionality constant that relates the heat flux through a boundary to the temperature difference across that boundary in
2469 thermal systems. It quantifies the thermal conductance at an interface between different materials or phases.
2470
2471 19. Hidden layer node (in neural networks): A computational unit situated between input and output layers that performs weighted summation of inputs followed by non-linear
2472 transformation through an activation function. It enables intermediate feature extraction and representation learning in deep network architectures.
2473
2474 20. Homogeneous coordinate (in computer graphics): An extended representation of a point in projective geometry using one additional coordinate component, allowing affine
2475 transformations including translation to be expressed as matrix multiplications. It unifies geometric transformation operations by representing points, vectors, and transformations in
2476 a consistent mathematical framework.
2477
2478 ]
2479
2480 9 (1 10){
2481 i
2482 }[
2483
2484 1 Index (in arrays and loops)
2485 2 Increment operation (increase by one)
2486 3 Integer value (in data types)
2487 4 Imaginary unit (in complex numbers)
2488 5 Input parameter (in functions)
2489 6 Instruction pointer (in CPU architecture)
2490 7 Iteration counter (in loops)
2491 8 Inverse function (in mathematics)
2492 9 Insertion operation (in data structures)
2493 10 Interrupt flag (in system control)
2494 11 Identity element (in algebraic structures)
2495 12 Initial value (in iterative processes)
2496 13 Integral (in calculus)
2497 14 Information bit (in information theory)
2498 15 Instance variable (in object-oriented programming)
2499 16 Indirection level (in pointer operations)
2500 17 Inequality comparison (in relational operations)
2501 18 Immediate value (in assembly instructions)
2502 19 Intersection operation (in set theory)
2503 20 Irrational number (in number theory)
2504
2505 DEFINITIONS
2506
2507 1. Index (in arrays and loops): A non-negative integer value that denotes the position of an element within an ordered collection, providing direct access to specific components
2508 through positional referencing. It enables element selection by numerical offset from the initial element and facilitates sequential traversal in iterative control structures.
2509
2510 2. Increment operation (increase by one): A unary arithmetic transformation that augments a numeric value by exactly one unit, preserving the original data type while producing the
2511 successor value. It implements ordinal progression for counter variables and sequential address generation in computational processes.
2512
2513 3. Integer value (in data types): A numerical datum representing a whole number without fractional components, capable of expressing positive quantities, negative quantities, and zero
2514 depending on signedness constraints. It implements exact arithmetic on discrete values within a bounded range determined by its binary representation width.
2515
2516 4. Imaginary unit (in complex numbers): The fundamental mathematical constant that satisfies the equation of squaring to negative one, serving as the basis for the complex number
2517 system. It enables representation of quantities that exist perpendicular to the real number line in the complex plane.
2518
2519 5. Input parameter (in functions): A named variable declaration in a function definition that receives a value when the function is invoked, establishing a formal binding for
2520 externally provided data. It creates a communication channel for passing information into the function's local execution context.
2521
2522 6. Instruction pointer (in CPU architecture): A specialized processor register that contains the memory address of the next instruction to be executed in the program sequence. It
2523 controls execution flow by incremental progression through the instruction stream, subject to modification by branch operations.
2524
2525 7. Iteration counter (in loops): A numeric variable that tracks the current repetition count in an iterative process, typically incremented after each cycle completion. It enables
2526 termination determination, element indexing, and progress monitoring during repeated execution of code blocks.
2527
2528 8. Inverse function (in mathematics): A function that reverses the effect of another function such that their composition yields the identity function, mapping each output of the
2529 original function back to its corresponding input. It performs the reverse transformation, undoing the operation of its counterpart function.
2530
2531 9. Insertion operation (in data structures): A structural modification procedure that incorporates a new element into an existing collection at a specified position while preserving
2532 the integrity and organizational properties of the data structure. It expands the collection size and establishes appropriate references to the newly added element.
2533
2534 10. Interrupt flag (in system control): A processor status bit that indicates whether external hardware interruptions of the current program flow are permitted. It provides a mechanism
2535 for temporarily disabling interrupt handling during critical operations that must execute atomically.
2536
2537 11. Identity element (in algebraic structures): A specialized value within a set equipped with a binary operation, which, when combined with any element of the set, leaves that element
```

unchanged. It establishes a neutral element that preserves operand values under the defined operation.

12. Initial value (in iterative processes): The starting assignment given to a variable before commencing a sequence of computational steps that will subsequently modify its value. It establishes the first state in a progression of values that evolve according to defined transformation rules.

13. Integral (in calculus): The mathematical operation that produces the accumulated effect of a function over a specified interval, representing the signed area between the function curve and the horizontal axis. It performs summation of infinitesimal contributions across a continuous domain.

14. Information bit (in information theory): The fundamental unit of information that resolves uncertainty between two equally probable alternatives, quantifying the minimum data required to distinguish between binary states. It serves as the atomic measurement unit for information content and entropy.

15. Instance variable (in object-oriented programming): A data member associated with each instantiated object of a class rather than with the class itself, maintaining distinct state information for individual instances. It implements object-specific properties that persist throughout the object's lifetime.

16. Indirection level (in pointer operations): The number of dereference operations required to access the target data value from a reference chain. It quantifies the depth of reference traversal needed to resolve the ultimate value in multi-level pointer relationships.

17. Inequality comparison (in relational operations): A binary operation that evaluates whether two values differ according to a defined ordering relation, producing a Boolean result indicating non-equivalence. It implements non-equality tests based on numerical magnitude, lexicographical sequence, or other comparable attributes.

18. Immediate value (in assembly instructions): A constant operand embedded directly within the instruction encoding rather than referenced from a register or memory location. It provides literal data values accessible without additional memory access operations during instruction execution.

19. Intersection operation (in set theory): A binary operation that produces a new set containing only elements present in all constituent sets, implementing the logical conjunction of set memberships. It identifies common elements shared among multiple collections according to membership criteria.

20. Irrational number (in number theory): A real number that cannot be expressed as a ratio of two integers, possessing a non-repeating, non-terminating decimal expansion. It represents quantities that cannot be precisely captured through finite fractional representation, such as transcendental and certain algebraic numbers.

]

10 (1 11){
j
}[

1 Jump instruction (in assembly language)
2 Join operation (in relational databases)
3 Jacobian matrix (in vector calculus)
4 Job identifier (in batch processing)
5 Jerk (third derivative of position in physics)
6 Joule (energy unit in calculations)
7 Junction point (in network analysis)
8 JSON index (in data serialization)
9 Jacobi method iteration (in numerical methods)
10 Java reference (in programming)
11 Joint probability (in statistics)
12 Journal entry (in transaction logs)
13 J-register (in some CPU architectures)
14 Justification factor (in text formatting)
15 Jitter value (in signal processing)
16 JWT token identifier (in authentication)
17 Job scheduling priority (in operating systems)
18 Julian date (in date calculations)
19 Juxtaposition operation (in matrix operations)
20 Jaro distance (in string similarity metrics)

DEFINITIONS

1. Jump instruction (in assembly language): A machine-level control transfer directive that unconditionally modifies the program counter to reference a non-sequential instruction address. It implements direct control flow redirection by explicitly setting the next execution location without conditional evaluation.

2. Join operation (in relational databases): A combinatorial procedure that creates a new relation by merging rows from two or more tables based on related column values according to a specified condition. It establishes associations between distinct data entities through matched attribute values to facilitate integrated data retrieval.

3. Jacobian matrix (in vector calculus): A rectangular array containing the first-order partial derivatives of a vector-valued function with respect to each of its input variables. It represents the best linear approximation to a differentiable function near a given point and enables transformation of differential elements between coordinate systems.

4. Job identifier (in batch processing): A unique alphanumeric designation assigned to a computational task within a multi-job processing environment. It provides an unambiguous reference for tracking, prioritization, and resource allocation throughout the job lifecycle.

5. Jerk (third derivative of position in physics): The rate of change of acceleration with respect to time, representing the time derivative of acceleration or the third time derivative of position. It quantifies the rapidity of force application in mechanical systems and contributes to analysis of motion smoothness.

2581

6. Joule (energy unit in calculations): The derived unit of energy in the International System of Units, defined as the work done when a force of one newton displaces an object one meter in the direction of the force. It quantifies energy transfer or transformation across mechanical, electrical, and thermal domains.

2582

2583

7. Junction point (in network analysis): A node in a network topology where three or more pathways or edges converge, forming a nexus of connectivity. It represents a critical interconnection location where traffic from multiple sources can redistribute along divergent paths.

2584

2585

8. JSON index (in data serialization): A numeric or string-based key that identifies a specific element within a JavaScript Object Notation structure, enabling direct access to nested values. It provides a navigational reference for retrieving or modifying particular components within hierarchical data representations.

2586

2587

9. Jacobi method iteration (in numerical methods): A recursive computational procedure for solving diagonally dominant systems of linear equations through successive approximation. It isolates individual variables and computes updated values based on the previous iteration's results until convergence criteria are satisfied.

2588

2589

10. Java reference (in programming): A typed pointer-like construct that stores the memory location of an object instance rather than containing the object's data directly. It implements indirect access to heap-allocated objects while abstracting memory management details from the programmer.

2590

2591

11. Joint probability (in statistics): The likelihood assigned to the simultaneous occurrence of two or more events within a probability space. It quantifies the combined chance of multiple outcomes happening together and forms the basis for analyzing event dependencies and correlations.

2592

2593

12. Journal entry (in transaction logs): A sequential, timestamped record documenting a state change operation in a persistent transaction logging system. It preserves the chronological order and complete details of modifications to enable system recovery and action reconstruction.

2594

2595

13. J-register (in some CPU architectures): A specialized processor register designated for specific computational roles such as index offsetting, temporary value storage, or jump target addressing. It augments the general register set with dedicated functionality in certain instruction sequences.

2596

2597

14. Justification factor (in text formatting): A numerical parameter that controls the distribution of whitespace when aligning text to both left and right margins. It determines the spacing adjustments between words and characters to achieve uniform line lengths while maintaining readability.

2598

2599

15. Jitter value (in signal processing): The quantitative measure of timing variability in a periodic signal, representing deviation from perfect periodicity due to noise or system instability. It characterizes temporal uncertainty in signal transitions that can degrade communication reliability.

2600

2601

16. JWT token identifier (in authentication): A unique reference value embedded within a JSON Web Token that distinguishes it from other security credentials in authentication systems. It enables token revocation, tracking, and validation against issuer records during authorization processes.

2602

2603

17. Job scheduling priority (in operating systems): A numeric value assigned to a process or task that determines its relative importance for processor time allocation in a multitasking environment. It influences execution sequencing decisions made by the scheduler to optimize system resource utilization.

2604

2605

18. Julian date (in date calculations): A continuous count of days elapsed since a defined epoch, typically noon on January 1, 4713 BCE in the proleptic Julian calendar. It facilitates chronological computations by representing calendar dates as a single numerical value for interval determination.

2606

2607

19. Juxtaposition operation (in matrix operations): The side-by-side arrangement of matrices that creates a composite matrix by horizontally concatenating their structures. It combines matrices by merging their columns while preserving row alignment to form an expanded representation.

2608

2609

20. Jaro distance (in string similarity metrics): A probabilistic measure that quantifies the character-level similarity between two text strings based on matching characters and transposition patterns. It produces a normalized score between zero and one that reflects string resemblance while accommodating character misplacements.

2610

2611

2612

2613

11

()

{

2614

k

2615

}{[]

2616

2617

12

()

{

2618

l

2619

}{[]

2620

2621

13

()

{

2622

m

2623

}{[]

2624

2625

14

()

{

2626

n

2627

}{[]

2628

2629

15

()

{

2630

o

2631

}{[]

2632

2633

16

()

{

2634

p

2635

}{[]

2636

2637

17

()

{

2638

q

```
2639     } []
2640
2641     18     () {
2642         r
2643     } []
2644
2645     19     () {
2646         s
2647     } []
2648
2649     20     () {
2650         t
2651     } []
2652
2653     21     () {
2654         u
2655     } []
2656
2657     22     () {
2658         v
2659     } []
2660
2661     23     () {
2662         w
2663     } []
2664
2665     24     () {
2666         x
2667     } []
2668
2669     25     () {
2670         y
2671     } []
2672
2673     26     () {
2674         z
2675     } []
2676
2677     27     () {
2678         A
2679     } []
2680
2681     28     () {
2682         B
2683     } []
2684
2685     29     () {
2686         C
2687     } []
2688
2689     30     () {
2690         D
2691     } []
2692
2693     31     () {
2694         E
2695     } []
2696
2697     32     () {
2698         F
2699     } []
2700
2701     33     () {
2702         G
2703     } []
2704
2705     34     () {
2706         H
2707     } []
2708
2709     35     () {
2710         I
2711     } []
```


2712
2713 36 () {
2714 J
2715 } []
2716
2717 37 () {
2718 K
2719 } []
2720
2721 38 () {
2722 L
2723 } []
2724
2725 39 () {
2726 M
2727 } []
2728
2729 40 () {
2730 N
2731 } []
2732
2733 41 () {
2734 O
2735 } []
2736
2737 42 () {
2738 P
2739 } []
2740
2741 43 () {
2742 Q
2743 } []
2744
2745 44 () {
2746 R
2747 } []
2748
2749 45 () {
2750 S
2751 } []
2752
2753 46 () {
2754 T
2755 } []
2756
2757 47 () {
2758 U
2759 } []
2760
2761 48 () {
2762 V
2763 } []
2764
2765 49 () {
2766 W
2767 } []
2768
2769 50 () {
2770 X
2771 } []
2772
2773 51 () {
2774 Y
2775 } []
2776
2777 52 () {
2778 Z
2779 } []
2780
2781 53 () {
2782 O
2783 } []
2784

2785

2786

2787

2788

2789

2790

2791

2792

2793

2794

2795

2796

2797

2798

2799

2800

2801

2802

2803

2804

2805

2806

2807

2808

2809

2810

2811

2812

2813

2814

2815

2816

2817

2818

2819

2820

2821

2822

2823

2824

2825

2826

2827

2828

2829

2830

2831

2832

2833

2834

2835

2836

2837

2838

2839

2840

2841

2842

2843

2844

2845

2846

2847

2848

2849

2850

2851

2852

2853

2854

2855

2856

2857

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

() {

1

}

[]

() {

2

}

[]

() {

3

}

[]

() {

4

}

[]

() {

5

}

[]

() {

6

}

[]

() {

7

}

[]

() {

8

}

[]

() {

9

}

[]

() {

\

}

[]

() {

|

}

[]

() {

,

}

[]

() {

<

}

[]

() {

.

}

[]

() {

>

}

[]

() {

/

}

[]

() {

?

}

[]

() {

;

}

[]

() {

2858 :
2859 } []
2860
2861 73 () {
2862 ,
2863 } []
2864
2865 74 () {
2866 @
2867 } []
2868
2869 75 () {
2870 #
2871 } []
2872
2873 76 () {
2874 ~
2875 } []
2876
2877 77 () {
2878 [
2879 } []
2880
2881 78 () {
2882 {
2883 } []
2884
2885 79 () {
2886]
2887 } []
2888
2889 80 () {
2890 }
2891 } []
2892
2893 81 () {
2894 ,
2895 } []
2896
2897 82 () {
2898 !
2899 } []
2900
2901 83 () {
2902 "
2903 } []
2904
2905 84 () {
2906 \$
2907 } []
2908
2909 85 () {
2910 %
2911 } []
2912
2913 86 () {
2914 ^
2915 } []
2916
2917 87 () {
2918 &
2919 } []
2920
2921 88 () {
2922 *
2923 } []
2924
2925 89 () {
2926 (
2927 } []
2928
2929 90 () {
2930)

```
2931 }[]
2932
2933 91  (){
2934     -
2935 }[]
2936
2937 92  (){
2938     -
2939 }[]
2940
2941 93  (){
2942     =
2943 }[]
2944
2945 94  (){
2946     +
2947 }[]
2948
2949 95  (Construction Framework){
2950
2951     //alphabet.js
2952     // Global Variables
2953     let grids = []; // Array to store color sequences of each grid
2954     let currentGridIndex = 0; // Index of the currently displayed grid
2955
2956     // Function to convert hex color code to a unique numeric ID (BigInt)
2957     function hexToID(hex) {
2958         return BigInt('0x' + hex.replace('#', '')) + BigInt(1);
2959     }
2960
2961     // Function to convert numeric ID (BigInt) back to hex color code
2962     function idToHex(id) {
2963         id = BigInt(id);
2964         if (id <= 0n) {
2965             return '#000000'; // Default to black or handle as needed
2966         }
2967         let hex = (id - BigInt(1)).toString(16).padStart(6, '0');
2968         return `#${hex}`;
2969     }
2970
2971     // Function to calculate the compounded grid ID from a color sequence
2972     function calculateCompoundedGridID(colorSequence) {
2973         return colorSequence.map(color => hexToID(color).toString()).join('');
2974     }
2975
2976     // Function to calculate grid dimensions based on the number of tiles
2977     function calculateGridDimensions(numTiles) {
2978         const sideLength = Math.ceil(Math.sqrt(numTiles));
2979         return { rows: sideLength, cols: sideLength };
2980     }
2981
2982     // Function to validate hex color code
2983     function isValidHexColor(hex) {
2984         return /^#([0-9A-F]{6})$/i.test(hex);
2985     }
2986
2987     // Function to update the grid based on user inputs
2988     function updateGrid() {
2989         const tileColor = document.getElementById('tile-color').value;
2990         const tileSize = parseInt(document.getElementById('tile-size').value);
2991
2992         if (isNaN(tileSize) || tileSize <= 0) {
2993             alert("Please enter a valid number for tile size.");
2994             return;
2995         }
2996
2997         const numTiles = 1; // Start with a single tile
2998         const colorSequence = Array(numTiles).fill(tileColor);
2999         grids = [colorSequence]; // Reset grids array with the new grid
3000         currentGridIndex = 0; // Reset to the first grid
3001
3002         displayCurrentGrid();
3003     }
```

```

3004
3005 // Function to display the current grid based on currentGridIndex
3006 function displayCurrentGrid() {
3007     const canvas = document.getElementById('grid-canvas');
3008     const context = canvas.getContext('2d');
3009
3010     if (grids.length === 0) {
3011         context.clearRect(0, 0, canvas.width, canvas.height);
3012         return;
3013     }
3014
3015     const tileSize = parseInt(document.getElementById('tile-size').value) || 100;
3016     const colorSequence = grids[currentGridIndex];
3017     const numTiles = colorSequence.length;
3018     const { rows, cols } = calculateGridDimensions(numTiles);
3019
3020     // Set canvas dimensions
3021     canvas.width = cols * tileSize;
3022     canvas.height = rows * tileSize;
3023
3024     // Clear the canvas
3025     context.clearRect(0, 0, canvas.width, canvas.height);
3026
3027     // Draw the grid
3028     colorSequence.forEach((color, i) => {
3029         const col = i % cols;
3030         const row = Math.floor(i / cols);
3031         const x = col * tileSize;
3032         const y = row * tileSize;
3033
3034         context.fillStyle = color;
3035         context.fillRect(x, y, tileSize, tileSize);
3036
3037         // Optional: Add a border to each tile
3038         context.strokeStyle = '#ccc';
3039         context.strokeRect(x, y, tileSize, tileSize);
3040     });
3041
3042     // Remove existing event listeners to prevent stacking
3043     canvas.onclick = null;
3044
3045     // Add a single event listener to the canvas
3046     canvas.addEventListener('click', function(event) {
3047         const rect = canvas.getBoundingClientRect();
3048         const clickX = event.clientX - rect.left;
3049         const clickY = event.clientY - rect.top;
3050
3051         const clickedCol = Math.floor(clickX / tileSize);
3052         const clickedRow = Math.floor(clickY / tileSize);
3053         const clickedIndex = clickedRow * cols + clickedCol;
3054
3055         if (clickedIndex >= 0 && clickedIndex < colorSequence.length) {
3056             const newColor = prompt("Enter new hex color (e.g., #00ff00):", colorSequence[clickedIndex]);
3057             if (newColor && isValidHexColor(newColor)) {
3058                 colorSequence[clickedIndex] = newColor;
3059                 displayCurrentGrid(); // Redraw the grid
3060             } else if (newColor) {
3061                 alert('Please enter a valid hex color code.');

```

```

3077         const listItem = document.createElement('li');
3078         listItem.textContent = `ID ${hexToID(color)}: ${color}`;
3079         colorList.appendChild(listItem);
3080     });
3081
3082     const compoundedGridID = calculateCompoundedGridID(colorSequence);
3083     const gridIdElement = document.getElementById('current-grid-id');
3084     gridIdElement.textContent = `Compounded Grid ID: ${compoundedGridID}`;
3085
3086     const gridPositionElement = document.getElementById('current-grid-position');
3087     gridPositionElement.textContent = `Grid ${currentGridIndex + 1} of ${grids.length}`;
3088 }
3089
3090 // Function to update the navigation buttons' disabled state
3091 function updateNavigationButtons() {
3092     const prevButton = document.getElementById('prev-grid');
3093     const nextButton = document.getElementById('next-grid');
3094
3095     prevButton.disabled = currentGridIndex === 0;
3096     nextButton.disabled = currentGridIndex === grids.length - 1;
3097 }
3098
3099 // Function to split a compounded ID into tile IDs
3100 function splitCompoundedID(compoundedID, tileIDLength) {
3101     const tileIDs = [];
3102     let index = 0;
3103     while (index < compoundedID.length) {
3104         const tileID = compoundedID.substr(index, tileIDLength);
3105         tileIDs.push(tileID);
3106         index += tileIDLength;
3107     }
3108     return tileIDs;
3109 }
3110
3111 // Function to regenerate grids from an array of compounded grid IDs
3112 function regenerateGrids(compoundedIDs) {
3113     grids = []; // Reset the grids array
3114
3115     compoundedIDs.forEach(idString => {
3116         const tileIDLength = 7; // Adjust this based on your tile ID length
3117         let tileIDs = [];
3118
3119         idString = idString.trim();
3120
3121         if (/^\d+$/.test(idString)) {
3122             // It's a single large number; split it into tile IDs
3123             tileIDs = splitCompoundedID(idString, tileIDLength);
3124         } else {
3125             alert('Invalid Compounded Grid ID format. Please enter a single large integer.');
```

```

3150     if (!file) {
3151         alert("Please select a sequences.txt file before executing.");
3152         return;
3153     }
3154
3155     readSequenceFile(file); // Parse the file and generate grids
3156 }
3157
3158 // Function to read the sequences.txt file and parse the compound IDs
3159 function readSequenceFile(file) {
3160     const reader = new FileReader();
3161
3162     reader.onload = function (event) {
3163         const content = event.target.result;
3164         const lines = content.split('\n').map(line => line.trim());
3165
3166         const validIDs = lines.filter(line => /^\d+$/.test(line)); // Only numeric lines
3167         if (validIDs.length > 0) {
3168             regenerateGrids(validIDs); // Process the valid compound IDs
3169         } else {
3170             alert("No valid compound IDs found in the file.");
3171         }
3172     };
3173
3174     reader.onerror = function () {
3175         alert("Failed to read the file.");
3176     };
3177
3178     reader.readAsText(file);
3179 }
3180
3181
3182 // Function to create a grid from the parsed colors
3183 function createGridFromColors(colors) {
3184     grids = [colors]; // Reset the grids array with the new grid
3185     currentIndex = 0; // Reset to the first grid
3186     displayCurrentGrid();
3187 }
3188
3189 // Function to generate a canvas for each grid and add it to a zip
3190 function downloadGridsAsZip(colors) {
3191     const zip = new JSZip();
3192
3193     colors.forEach((color, index) => {
3194         const canvas = document.createElement('canvas');
3195         const context = canvas.getContext('2d');
3196
3197         // Set canvas dimensions
3198         const tileSize = 100; // Adjust tile size as needed
3199         canvas.width = tileSize;
3200         canvas.height = tileSize;
3201
3202         // Draw the tile
3203         context.fillStyle = color;
3204         context.fillRect(0, 0, tileSize, tileSize);
3205
3206         // Convert canvas to data URL and add to zip
3207         const imageData = canvas.toDataURL('image/png').split(',')[1];
3208         zip.file(`grid_${index + 1}.png`, imageData, { base64: true });
3209     });
3210 }
3211
3212 // Generate and download the zip file
3213 zip.generateAsync({ type: "blob" })
3214     .then(function (content) {
3215         const link = document.createElement('a');
3216         link.href = URL.createObjectURL(content);
3217         link.download = "grids.zip";
3218         link.click();
3219     })
3220     .catch(function (error) {
3221         console.error("Error creating ZIP file:", error);
3222     });

```

```

3223 }
3224
3225
3226 // Event listener for the "Update Grid" button
3227 document.addEventListener('DOMContentLoaded', function() {
3228
3229     // Event listener for the Execute button
3230     document.getElementById('execute-button').addEventListener('click', executeGridProcessing);
3231
3232     document.getElementById('update-grid').addEventListener('click', updateGrid);
3233
3234     // Event listener for the "Download Tile ID" button
3235     document.getElementById('download-colors').addEventListener('click', function() {
3236         const colorSequence = grids[currentGridIndex];
3237         const compoundedGridID = calculateCompoundedGridID(colorSequence);
3238         const fileContent = colorSequence.map((color, index) => `ID ${hexToID(color)}: ${color}`).join('\n');
3239         const fullContent = `Compounded Grid ID: ${compoundedGridID}\n\n${fileContent}`;
3240         const blob = new Blob([fullContent], { type: 'text/plain' });
3241         const url = URL.createObjectURL(blob);
3242
3243         const a = document.createElement('a');
3244         a.href = url;
3245         a.download = 'color_sequence.txt';
3246         a.click();
3247
3248         URL.revokeObjectURL(url);
3249     });
3250
3251     // Event listener for the "Generate Grids" button
3252     document.getElementById('generate-grid-by-id').addEventListener('click', function() {
3253         const inputFields = document.querySelectorAll('.compounded-id-input');
3254         const compoundedIDStrings = [];
3255
3256         inputFields.forEach(input => {
3257             const idString = input.value.trim();
3258             if (idString !== '') {
3259                 compoundedIDStrings.push(idString);
3260             }
3261         });
3262
3263         if (compoundedIDStrings.length === 0) {
3264             alert('Please enter at least one Compounded Grid ID.');
```



```
3296     });
3297
3298     // Event listener for the "Next Grid" button
3299     document.getElementById('next-grid').addEventListener('click', function() {
3300         if (currentGridIndex < grids.length - 1) {
3301             currentGridIndex++;
3302             displayCurrentGrid();
3303         }
3304     });
3305
3306     // Event listener for the "Download Grid Image" button
3307     document.getElementById('download-grid-image').addEventListener('click', function() {
3308         const canvas = document.getElementById('grid-canvas');
3309         const image = canvas.toDataURL('image/png');
3310
3311         // Create a link element
3312         const link = document.createElement('a');
3313         link.download = `grid_${currentGridIndex + 1}.png`;
3314         link.href = image;
3315         link.click();
3316     });
3317
3318     // Initialize the grid on page load
3319     const defaultCompoundedID = '1443664'; // Adjust this as needed
3320     document.querySelector('.compounded-id-input').value = defaultCompoundedID;
3321
3322     updateGrid();
3323 });
3324
3325 //build.html
3326 <!DOCTYPE html>
3327 <html lang="en">
3328 <head>
3329     <meta charset="UTF-8">
3330     <meta name="viewport" content="width=device-width, initial-scale=1.0">
3331     <title>Memory Slot Manager & Logic Framework</title>
3332     <style>
3333         :root {
3334             --primary-color: #0078d7;
3335             --secondary-color: #4a4a4a;
3336             --background-color: #f9f9f9;
3337             --terminal-bg: #1e1e1e;
3338             --terminal-text: #f0f0f0;
3339             --border-color: #dbdbdb;
3340             --highlight-color: #e6f2ff;
3341         }
3342
3343         * {
3344             box-sizing: border-box;
3345             margin: 0;
3346             padding: 0;
3347             font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
3348         }
3349
3350         body {
3351             background-color: var(--background-color);
3352             line-height: 1.6;
3353             color: var(--secondary-color);
3354             padding: 20px;
3355             max-width: 1200px;
3356             margin: 0 auto;
3357         }
3358
3359         header {
3360             text-align: center;
3361             margin-bottom: 30px;
3362             padding: 20px;
3363             background-color: white;
3364             border-radius: 5px;
3365             box-shadow: 0 2px 5px rgba(0,0,0,0.1);
3366         }
3367
3368         h1 {
```

```
3369         color: var(--primary-color);
3370         margin-bottom: 10px;
3371     }
3372
3373     .system-info {
3374         font-size: 0.9em;
3375         color: var(--secondary-color);
3376     }
3377
3378     .main-container {
3379         display: flex;
3380         flex-direction: column;
3381         gap: 20px;
3382     }
3383
3384     @media (min-width: 992px) {
3385         .main-container {
3386             flex-direction: row;
3387         }
3388
3389         .terminal-section {
3390             flex: 1;
3391         }
3392
3393         .results-section {
3394             flex: 1;
3395         }
3396     }
3397
3398     .section {
3399         background-color: white;
3400         border-radius: 5px;
3401         padding: 20px;
3402         box-shadow: 0 2px 5px rgba(0,0,0,0.1);
3403     }
3404
3405     .section-title {
3406         font-size: 1.2em;
3407         margin-bottom: 15px;
3408         padding-bottom: 10px;
3409         border-bottom: 1px solid var(--border-color);
3410         color: var(--primary-color);
3411         display: flex;
3412         justify-content: space-between;
3413         align-items: center;
3414     }
3415
3416     .section-title-actions {
3417         display: flex;
3418         gap: 10px;
3419     }
3420
3421     .action-button {
3422         padding: 3px 8px;
3423         font-size: 0.8em;
3424         background-color: var(--primary-color);
3425         color: white;
3426         border: none;
3427         border-radius: 3px;
3428         cursor: pointer;
3429     }
3430
3431     .action-button:hover {
3432         background-color: #005a9e;
3433     }
3434
3435     .terminal {
3436         background-color: var(--terminal-bg);
3437         color: var(--terminal-text);
3438         padding: 15px;
3439         border-radius: 5px;
3440         font-family: 'Consolas', 'Courier New', monospace;
3441         height: 350px;
```

```
3442         overflow-y: auto;
3443         margin-bottom: 15px;
3444     }
3445
3446     .terminal-input {
3447         display: flex;
3448         margin-bottom: 10px;
3449     }
3450
3451     .terminal-input span {
3452         margin-right: 10px;
3453         color: #4caf50;
3454     }
3455
3456     #commandInput {
3457         width: 100%;
3458         padding: 10px;
3459         font-family: 'Consolas', 'Courier New', monospace;
3460         margin-bottom: 10px;
3461         border: 1px solid var(--border-color);
3462         border-radius: 5px;
3463     }
3464
3465     .button {
3466         background-color: var(--primary-color);
3467         color: white;
3468         border: none;
3469         padding: 10px 15px;
3470         border-radius: 5px;
3471         cursor: pointer;
3472         font-size: 0.9em;
3473         transition: background-color 0.3s;
3474     }
3475
3476     .button:hover {
3477         background-color: #005a9e;
3478     }
3479
3480     .button-secondary {
3481         background-color: #6c757d;
3482     }
3483
3484     .button-secondary:hover {
3485         background-color: #5a6268;
3486     }
3487
3488     #multilineInput {
3489         width: 100%;
3490         min-height: 100px;
3491         padding: 10px;
3492         font-family: 'Consolas', 'Courier New', monospace;
3493         margin-bottom: 10px;
3494         border: 1px solid var(--border-color);
3495         border-radius: 5px;
3496         display: none;
3497         white-space: pre;
3498         overflow-wrap: normal;
3499         overflow-x: auto;
3500     }
3501
3502     .memory-display {
3503         background-color: white;
3504         border: 1px solid var(--border-color);
3505         border-radius: 5px;
3506         padding: 15px;
3507         height: 500px;
3508         overflow-y: auto;
3509     }
3510
3511     .memory-file {
3512         margin-bottom: 20px;
3513     }
3514
```

```
3515 .memory-file-title {
3516     font-weight: bold;
3517     margin-bottom: 5px;
3518     padding-bottom: 5px;
3519     border-bottom: 1px solid var(--border-color);
3520     display: flex;
3521     justify-content: space-between;
3522     align-items: center;
3523 }
3524
3525 .memory-file-actions {
3526     display: flex;
3527     gap: 5px;
3528 }
3529
3530 .file-action {
3531     font-size: 0.7em;
3532     padding: 2px 5px;
3533     background-color: var(--primary-color);
3534     color: white;
3535     border: none;
3536     border-radius: 3px;
3537     cursor: pointer;
3538 }
3539
3540 .file-action:hover {
3541     background-color: #005a9e;
3542 }
3543
3544 .memory-slot {
3545     padding: 8px;
3546     border-bottom: 1px solid #f0f0f0;
3547 }
3548
3549 .memory-slot:hover {
3550     background-color: var(--highlight-color);
3551 }
3552
3553 .memory-slot-title {
3554     font-weight: bold;
3555     margin-bottom: 5px;
3556 }
3557
3558 .memory-slot-value {
3559     white-space: pre-wrap;
3560     font-family: 'Consolas', 'Courier New', monospace;
3561     padding: 5px;
3562     background-color: #f8f8f8;
3563     border-radius: 3px;
3564     font-size: 0.9em;
3565 }
3566
3567 .quick-commands {
3568     display: flex;
3569     flex-wrap: wrap;
3570     gap: 10px;
3571     margin-top: 15px;
3572 }
3573
3574 .quick-command {
3575     background-color: #f0f0f0;
3576     padding: 5px 10px;
3577     border-radius: 3px;
3578     font-size: 0.8em;
3579     cursor: pointer;
3580     transition: background-color 0.3s;
3581 }
3582
3583 .quick-command:hover {
3584     background-color: #e0e0e0;
3585 }
3586
3587 .modal {
```

```
3588         display: none;
3589         position: fixed;
3590         z-index: 1;
3591         left: 0;
3592         top: 0;
3593         width: 100%;
3594         height: 100%;
3595         overflow: auto;
3596         background-color: rgba(0,0,0,0.5);
3597     }
3598
3599     .modal-content {
3600         background-color: #fefefe;
3601         margin: 15% auto;
3602         padding: 20px;
3603         border: 1px solid #888;
3604         width: 80%;
3605         max-width: 600px;
3606         border-radius: 5px;
3607     }
3608
3609     .close-modal {
3610         color: #aaa;
3611         float: right;
3612         font-size: 28px;
3613         font-weight: bold;
3614         cursor: pointer;
3615     }
3616
3617     .close-modal:hover {
3618         color: black;
3619     }
3620
3621     .progress-bar-container {
3622         width: 100%;
3623         background-color: #e0e0e0;
3624         border-radius: 3px;
3625         margin: 10px 0;
3626     }
3627
3628     .progress-bar {
3629         height: 20px;
3630         background-color: var(--primary-color);
3631         border-radius: 3px;
3632         width: 0%;
3633         transition: width 0.3s;
3634     }
3635
3636     #generationStatus {
3637         margin-top: 10px;
3638         font-size: 0.9em;
3639     }
3640
3641     /* File system styles */
3642     .files-section {
3643         margin-top: 20px;
3644     }
3645
3646     .file-system-actions {
3647         display: flex;
3648         gap: 10px;
3649         margin-bottom: 10px;
3650     }
3651
3652     /* Radio button styling */
3653     .radio-group {
3654         margin: 15px 0;
3655     }
3656
3657     .radio-option {
3658         margin-bottom: 10px;
3659         display: flex;
3660         align-items: flex-start;
```

```
3661     }
3662
3663     .radio-option input[type="radio"] {
3664         margin-top: 3px;
3665         margin-right: 10px;
3666     }
3667
3668     .radio-option .option-label {
3669         font-weight: bold;
3670     }
3671
3672     .radio-option .option-description {
3673         font-size: 0.85em;
3674         color: #666;
3675         margin-top: 2px;
3676     }
3677
3678     /* File input styling */
3679     .file-input-container {
3680         margin: 15px 0;
3681     }
3682
3683     .file-input-label {
3684         font-weight: bold;
3685         margin-bottom: 5px;
3686         display: block;
3687     }
3688
3689     .file-input {
3690         width: 100%;
3691         padding: 8px;
3692         border: 1px solid var(--border-color);
3693         border-radius: 4px;
3694     }
3695
3696     /* Logic engine styles */
3697     .nav-tabs {
3698         display: flex;
3699         border-bottom: 1px solid var(--border-color);
3700         margin-bottom: 15px;
3701     }
3702
3703     .nav-tab {
3704         padding: 8px 15px;
3705         cursor: pointer;
3706         border: 1px solid transparent;
3707         border-bottom: none;
3708         border-radius: 5px 5px 0 0;
3709         margin-right: 5px;
3710         background-color: #f8f8f8;
3711     }
3712
3713     .nav-tab.active {
3714         background-color: white;
3715         border-color: var(--border-color);
3716         border-bottom: 1px solid white;
3717         margin-bottom: -1px;
3718         font-weight: bold;
3719     }
3720
3721     .tab-content {
3722         display: none;
3723     }
3724
3725     .tab-content.active {
3726         display: block;
3727     }
3728
3729     #logicTerminal {
3730         background-color: var(--terminal-bg);
3731         color: var(--terminal-text);
3732         padding: 15px;
3733         border-radius: 5px;
```

```
3734         font-family: 'Consolas', 'Courier New', monospace;
3735         height: 300px;
3736         overflow-y: auto;
3737         margin-bottom: 15px;
3738     }
3739
3740     @media (max-width: 768px) {
3741         body {
3742             padding: 10px;
3743         }
3744
3745         .terminal {
3746             height: 250px;
3747         }
3748
3749         .memory-display {
3750             height: 300px;
3751         }
3752
3753         .modal-content {
3754             width: 95%;
3755             margin: 10% auto;
3756         }
3757
3758         .file-system-actions {
3759             flex-direction: column;
3760         }
3761     }
3762 }
3763 </style>
3764 </head>
3765 <body>
3766     <header>
3767         <h1>INTEGRATED MEMORY & LOGIC FRAMEWORK</h1>
3768         <div class="system-info">
3769             <div id="systemTime"></div>
3770             <div>MEMORY MANAGER & LOGIC ENGINE LOADED</div>
3771             <div>READY FOR INPUT</div>
3772         </div>
3773     </header>
3774
3775     <div class="main-container">
3776         <div class="terminal-section">
3777             <div class="section">
3778                 <div class="nav-tabs">
3779                     <div class="nav-tab active" data-tab="memory">Memory Manager</div>
3780                     <div class="nav-tab" data-tab="logic">Logic Engine</div>
3781                 </div>
3782
3783                 <div id="memoryTab" class="tab-content active">
3784                     <div class="section-title">
3785                         Command Terminal
3786                         <div class="section-title-actions">
3787                             <button class="action-button" onclick="clearTerminal()">Clear</button>
3788                         </div>
3789                     </div>
3790                     <div id="terminal" class="terminal"></div>
3791                     <input type="text" id="commandInput" placeholder="Enter command (type 'help' for list of commands)">
3792                     <textarea id="multilineInput" placeholder="Enter multi-line content (e.g. for assign or search)" wrap="off"></textarea>
3793                     <div class="quick-commands">
3794                         <span class="quick-command" onclick="insertCommand('help')">help</span>
3795                         <span class="quick-command" onclick="insertCommand('display')">display</span>
3796                         <span class="quick-command" onclick="insertCommand('assign')">assign</span>
3797                         <span class="quick-command" onclick="insertCommand('read')">read</span>
3798                         <span class="quick-command" onclick="insertCommand('search')">search</span>
3799                         <span class="quick-command" onclick="insertCommand('generate')">generate</span>
3800                         <span class="quick-command" onclick="insertCommand('save_all')">save_all</span>
3801                         <span class="quick-command" onclick="insertCommand('load_all')">load_all</span>
3802                     </div>
3803                     <div style="display: flex; gap: 10px; margin-top: 10px;">
3804                         <button id="submitCommand" class="button">Execute Command</button>
3805                         <button id="submitMultiline" class="button" style="display: none;">Submit</button>
3806                         <button id="cancelMultiline" class="button button-secondary" style="display: none;">Cancel</button>
3807                     </div>
3808                 </div>
3809             </div>
3810         </div>
3811     </div>
3812 </body>
3813 </html>
```

```
3807         </div>
3808
3809         <div id="logicTab" class="tab-content">
3810             <div class="section-title">
3811                 Proof by Contradiction Engine
3812                 <div class="section-title-actions">
3813                     <button class="action-button" onclick="clearLogicTerminal()">Clear</button>
3814                 </div>
3815             </div>
3816             <div id="logicTerminal" class="terminal"></div>
3817             <input type="text" id="logicCommandInput" placeholder="Enter instruction (type 'help' for available instructions)">
3818             <div class="quick-commands">
3819                 <span class="quick-command" onclick="insertLogicCommand('help')">help</span>
3820                 <span class="quick-command" onclick="insertLogicCommand('ASSERT')">ASSERT</span>
3821                 <span class="quick-command" onclick="insertLogicCommand('NOT')">NOT</span>
3822                 <span class="quick-command" onclick="insertLogicCommand('AND')">AND</span>
3823                 <span class="quick-command" onclick="insertLogicCommand('OR')">OR</span>
3824                 <span class="quick-command" onclick="insertLogicCommand('IMPLIES')">IMPLIES</span>
3825                 <span class="quick-command" onclick="insertLogicCommand('CONTRADICTION')">CONTRADICTION</span>
3826             </div>
3827             <div style="display: flex; gap: 10px; margin-top: 10px;">
3828                 <button id="submitLogicCommand" class="button">Execute Instruction</button>
3829                 <button id="checkContradictions" class="button">Check Contradictions</button>
3830                 <button id="clearStatements" class="button button-secondary">Clear Statements</button>
3831                 <button id="generateStatements" class="button">Generate Statements</button>
3832             </div>
3833         </div>
3834     </div>
3835
3836     <div class="section files-section">
3837         <div class="section-title">File System</div>
3838         <div class="file-system-actions">
3839             <button class="button" onclick="saveAllMemorySlots()">Save All Files</button>
3840             <button class="button" onclick="loadMemorySlotsFromFile()">Load Files</button>
3841             <button class="button button-secondary" onclick="clearAllMemorySlots()">Clear All Files</button>
3842         </div>
3843     </div>
3844 </div>
3845
3846 <div class="results-section">
3847     <div class="section">
3848         <div class="section-title">
3849             Memory Slots
3850             <div class="section-title-actions">
3851                 <button class="action-button" onclick="updateMemoryDisplay()">Refresh</button>
3852             </div>
3853         </div>
3854         <div id="memoryDisplay" class="memory-display">
3855             <div style="text-align: center; color: #888; margin-top: 20px;">
3856                 No memory slots available.
3857                 <br><br>
3858                 Use the terminal to add memory slots.
3859             </div>
3860         </div>
3861     </div>
3862 </div>
3863 </div>
3864
3865 <div id="generationModal" class="modal">
3866     <div class="modal-content">
3867         <span class="close-modal" onclick="document.getElementById('generationModal').style.display='none'">&times;</span>
3868         <h2>Generating Combinations</h2>
3869
3870         <div class="radio-group">
3871             <div class="radio-option">
3872                 <input type="radio" id="generateSingleFile" name="generateType" value="singleFile" checked>
3873                 <div>
3874                     <div class="option-label">Single File with Multiple Slots</div>
3875                     <div class="option-description">Store all combinations in one file with each combination as a separate slot</div>
3876                 </div>
3877             </div>
3878             <div class="radio-option">
3879                 <input type="radio" id="generateMultiFile" name="generateType" value="multiFile">
```



```
3880         <div>
3881             <div class="option-label">One File Per Combination</div>
3882             <div class="option-description">Create a separate file for each generated combination</div>
3883         </div>
3884     </div>
3885 </div>
3886
3887 <div id="fileInputContainer" class="file-input-container">
3888     <div class="file-input-label">Enter Filename for Single File Mode:</div>
3889     <input type="text" id="fileNameInput" class="file-input" value="combinations" placeholder="Enter filename (e.g. 3, combinations, etc.)">
3890 </div>
3891
3892 <div class="progress-bar-container">
3893     <div id="generationProgress" class="progress-bar"></div>
3894 </div>
3895 <div id="generationStatus">Ready to generate</div>
3896 <div style="margin-top: 20px; display: flex; gap: 10px; flex-wrap: wrap;">
3897     <button id="startGenerationBtn" class="button">Start Generation</button>
3898     <button id="downloadGeneratedBtn" class="button" disabled>Download Results</button>
3899     <button id="importGeneratedBtn" class="button" disabled>Import to Memory Slots</button>
3900     <button class="button button-secondary" onclick="document.getElementById('generationModal').style.display='none'">Close</button>
3901 </div>
3902 </div>
3903 </div>
3904
3905 <div id="statementsGenerationModal" class="modal">
3906     <div class="modal-content">
3907         <span class="close-modal" onclick="document.getElementById('statementsGenerationModal').style.display='none'">&times;</span>
3908         <h2>Generate Statements</h2>
3909
3910         <div class="file-input-container">
3911             <div class="file-input-label">Number of statements to generate:</div>
3912             <input type="number" id="numStatementsInput" class="file-input" value="5" min="1" max="100">
3913         </div>
3914
3915         <div style="margin-top: 20px; display: flex; gap: 10px; flex-wrap: wrap;">
3916             <button id="startStatementsGenBtn" class="button">Generate Statements</button>
3917             <button class="button button-secondary" onclick="document.getElementById('statementsGenerationModal').style.display='none'">Cancel</button>
3918         </div>
3919     </div>
3920 </div>
3921
3922 <div id="helpModal" class="modal">
3923     <div class="modal-content">
3924         <span class="close-modal" onclick="document.getElementById('helpModal').style.display='none'">&times;</span>
3925         <h2>Available Commands</h2>
3926         <pre style="white-space: pre-wrap; max-height: 400px; overflow-y: auto; margin-top: 10px;">
3927 AVAILABLE COMMANDS:
3928 -----
3929 assign <fileNumber> <slotNumber>;
3930     <value>;           : Assign a multi-line value to a slot
3931                        (Enter your multi-line value in the text area)
3932
3933 read <fileNumber> <slotNumber>;
3934     : Read the value from a slot
3935
3936 last_slot <fileNumber>;
3937     : Get the last slot number in a file
3938
3939 search
3940     <value>;           : Search for a value across all slots
3941
3942 call <fileNumber> <startSlot> <endSlot>;
3943     : Call values from a range of slots
3944
3945 export <fileNumber>;   : Export a file's memory slots to a text file
3946
3947 generate <n>;          : Generate all combinations with n cells
3948                        using the standard character set
3949                        Results are ready to use with this application
3950
3951                        Generation Options:
3952                        - Single File: All combinations in one file with multiple slots
```

```
3953         - One File Per Combination: Each combination in its own file
3954
3955 custom_generate <fileNumber> : Generate combinations with custom character set
3956
3957 display      : Show all memory slots
3958 clear        : Clear the terminal
3959 help         : Display this help message
3960
3961 save_json <fileNumber> : Save a specific file to JSON
3962 save_all      : Save all memory slots to a ZIP archive
3963 load_json     : Load memory slots from a JSON file
3964 load_all      : Load memory slots from a ZIP archive
3965
3966 File System Operations:
3967 -----
3968 The file system supports downloading all your data as a ZIP archive
3969 and uploading it again for continuous use. This provides an automatic
3970 way to save your work and continue where you left off.
3971
3972 Generated files are automatically structured to be used with the
3973 memory slot manager system without any additional processing.
3974 </pre>
3975 </div>
3976 </div>
3977
3978 <div id="logicHelpModal" class="modal">
3979   <div class="modal-content">
3980     <span class="close-modal" onclick="document.getElementById('logicHelpModal').style.display='none'">&times;</span>
3981     <h2>Proof by Contradiction Engine Help</h2>
3982     <pre style="white-space: pre-wrap; max-height: 400px; overflow-y: auto; margin-top: 10px;">
3983 Proof by Contradiction Engine Help
3984 -----
3985 1. Add Instruction:
3986   - ASSERT <statement>;
3987   - NOT <statement>;
3988   - AND <statement1>; <statement2>;
3989   - OR <statement1>; <statement2>;
3990   - IMPLIES <statement1>; <statement2>;
3991   - CONTRADICTION <statement1>; <statement2>;
3992 2. Clear Statements: Clears all statements from the current framework.
3993 3. Check Contradictions: Checks for contradictions within the current framework using defined rules.
3994 4. Generate Statements: Automatically generates a framework of statements.
3995 5. Help: Displays this help message.
3996
3997 Example:
3998   ASSERT Statement1
3999   NOT Statement1
4000   CONTRADICTION Statement1 NOT Statement1
4001   (This will detect a contradiction based on the statement_not_statement rule)
4002   </pre>
4003   </div>
4004 </div>
4005
4006 <script>
4007   class MemorySlotManager {
4008     constructor() {
4009       this.memorySlots = {};
4010     }
4011
4012     assignSlot(fileNumber, slotNumber, value) {
4013       if (!fileNumber || !slotNumber || !value) {
4014         return "Error: Invalid input parameters.";
4015       }
4016
4017       if (this.valueExists(value)) {
4018         return "Error: The value already exists in another memory slot.";
4019       }
4020
4021       if (!this.memorySlots[fileNumber]) {
4022         this.memorySlots[fileNumber] = {};
4023       }
4024
4025       this.memorySlots[fileNumber][slotNumber] = value;
```

```

4026         return `Assigned value to slot ${slotNumber} in file ${fileNumber}.`;
4027     }
4028
4029     readSlot(fileNumber, slotNumber) {
4030         if (this.memorySlots[fileNumber] && this.memorySlots[fileNumber][slotNumber] !== undefined) {
4031             return this.memorySlots[fileNumber][slotNumber];
4032         }
4033         return `Slot ${slotNumber} not found in file ${fileNumber}.`;
4034     }
4035
4036     getLastSlotNumber(fileNumber) {
4037         if (this.memorySlots[fileNumber] && Object.keys(this.memorySlots[fileNumber]).length > 0) {
4038             try {
4039                 const slots = [];
4040                 for (const slot in this.memorySlots[fileNumber]) {
4041                     if (/^\d+$/.test(slot)) {
4042                         slots.push(parseInt(slot));
4043                     }
4044                 }
4045
4046                 if (slots.length > 0) {
4047                     return Math.max(...slots);
4048                 }
4049             } catch (e) {
4050                 // Ignore exceptions
4051             }
4052         }
4053         return null;
4054     }
4055
4056     searchValue(value) {
4057         const results = [];
4058         const searchValueLower = value.toLowerCase();
4059
4060         for (const file in this.memorySlots) {
4061             for (const slot in this.memorySlots[file]) {
4062                 const val = this.memorySlots[file][slot];
4063                 if (val.toLowerCase().includes(searchValueLower)) {
4064                     results.push({
4065                         file,
4066                         slot,
4067                         value: val
4068                     });
4069                 }
4070             }
4071         }
4072
4073         return results;
4074     }
4075
4076     valueExists(value) {
4077         const valueLower = value.toLowerCase();
4078
4079         for (const file in this.memorySlots) {
4080             for (const slot in this.memorySlots[file]) {
4081                 if (this.memorySlots[file][slot].toLowerCase() === valueLower) {
4082                     return true;
4083                 }
4084             }
4085         }
4086
4087         return false;
4088     }
4089
4090     callSlotRange(fileNumber, startSlot, endSlot) {
4091         if (!this.memorySlots[fileNumber]) {
4092             return `File ${fileNumber} not found.`;
4093         }
4094
4095         const results = [];
4096         for (let slot = startSlot; slot <= endSlot; slot++) {
4097             const slotStr = slot.toString();
4098             if (this.memorySlots[fileNumber][slotStr] !== undefined) {

```

```

4099         results.push(`Slot ${slotStr}:\n${this.memorySlots[fileNumber][slotStr]}`);
4100     } else {
4101         results.push(`Slot ${slotStr} not found.`);
4102     }
4103 }
4104
4105 return results.join('\n\n');
4106 }
4107
4108 exportToTextFile(fileNumber) {
4109     if (!this.memorySlots[fileNumber]) {
4110         return `Error: File ${fileNumber} not found.`;
4111     }
4112
4113     try {
4114         let content = '';
4115         const slots = this.memorySlots[fileNumber];
4116         let slotKeys = [];
4117
4118         try {
4119             const numericKeys = [];
4120             for (const key in slots) {
4121                 if (/^\d+$/.test(key)) {
4122                     numericKeys.push(parseInt(key));
4123                 }
4124             }
4125             numericKeys.sort((a, b) => a - b);
4126             slotKeys = numericKeys.map(key => key.toString());
4127         } catch (e) {
4128             // If conversion fails, sort alphabetically
4129             slotKeys = Object.keys(slots).sort();
4130         }
4131
4132         for (const slot of slotKeys) {
4133             const value = slots[slot];
4134             // Format: SLOT <number>
4135             content += `SLOT ${slot}\n${value}\n-----\n`;
4136         }
4137
4138         // Create a download link
4139         const blob = new Blob([content], { type: 'text/plain' });
4140         const url = URL.createObjectURL(blob);
4141         const a = document.createElement('a');
4142         a.href = url;
4143         a.download = `file${fileNumber}.txt`;
4144         a.click();
4145         URL.revokeObjectURL(url);
4146
4147         return `Exported file ${fileNumber} to file${fileNumber}.txt`;
4148     } catch (e) {
4149         return `Error exporting to text file: ${e.message}`;
4150     }
4151 }
4152
4153 saveToJSON(fileNumber) {
4154     try {
4155         if (fileNumber && !this.memorySlots[fileNumber]) {
4156             return `Error: File ${fileNumber} not found.`;
4157         }
4158
4159         let jsonData;
4160         let filename;
4161
4162         if (fileNumber) {
4163             jsonData = this.memorySlots[fileNumber];
4164             filename = `file${fileNumber}.json`;
4165         } else {
4166             jsonData = this.memorySlots;
4167             filename = 'memory_slots.json';
4168         }
4169
4170         const jsonString = JSON.stringify(jsonData, null, 2);
4171         const blob = new Blob([jsonString], { type: 'application/json' });

```

```

4172         const url = URL.createObjectURL(blob);
4173         const a = document.createElement('a');
4174         a.href = url;
4175         a.download = filename;
4176         a.click();
4177         URL.revokeObjectURL(url);
4178
4179         return `Memory slots saved to ${filename}`;
4180     } catch (e) {
4181         return `Error saving to JSON: ${e.message}`;
4182     }
4183 }
4184
4185 loadJSON(jsonData) {
4186     try {
4187         const content = JSON.parse(jsonData);
4188
4189         if (typeof content === 'object' && content !== null) {
4190             // Check if this is our expected format
4191             let isFullStructure = false;
4192             for (const file in content) {
4193                 if (typeof content[file] === 'object' && content[file] !== null) {
4194                     isFullStructure = true;
4195                     break;
4196                 }
4197             }
4198
4199             if (isFullStructure) {
4200                 // Merge with existing memory slots
4201                 for (const file in content) {
4202                     if (!this.memorySlots[file]) {
4203                         this.memorySlots[file] = {};
4204                     }
4205
4206                     for (const slot in content[file]) {
4207                         this.memorySlots[file][slot] = content[file][slot];
4208                     }
4209                 }
4210             } else {
4211                 // Assume it's a single file's slots
4212                 const fileNumber = '1'; // Default file number
4213                 if (!this.memorySlots[fileNumber]) {
4214                     this.memorySlots[fileNumber] = {};
4215                 }
4216
4217                 for (const slot in content) {
4218                     this.memorySlots[fileNumber][slot] = content[slot];
4219                 }
4220             }
4221
4222             return 'Successfully loaded JSON data.';
4223         }
4224
4225         return 'Error: Invalid JSON structure.';
4226     } catch (e) {
4227         return `Error loading JSON: ${e.message}`;
4228     }
4229 }
4230
4231 // Save all memory slots as a ZIP file
4232 saveAllAsZip() {
4233     try {
4234         // We'll use JSZip to create a ZIP file
4235         const zip = new JSZip();
4236
4237         // Add JSON files
4238         for (const fileNumber in this.memorySlots) {
4239             const slots = this.memorySlots[fileNumber];
4240
4241             // Create JSON file
4242             const jsonContent = JSON.stringify(slots, null, 2);
4243             zip.file(`json/file_${fileNumber}.json`, jsonContent);
4244

```

```

4245         // Also create text version
4246         let txtContent = '';
4247         const slotNumbers = Object.keys(slots);
4248         for (const slotNumber of slotNumbers) {
4249             txtContent += `SLOT ${slotNumber}\n${slots[slotNumber]}\n-----\n`;
4250         }
4251         zip.file(`txt/file_${fileNumber}.txt`, txtContent);
4252     }
4253
4254     // Add complete structure as one JSON file
4255     const allJson = JSON.stringify(this.memorySlots, null, 2);
4256     zip.file('all_memory_slots.json', allJson);
4257
4258     // Generate the ZIP file
4259     return zip.generateAsync({ type: 'blob' })
4260         .then(blob => {
4261             // Create download link
4262             const url = URL.createObjectURL(blob);
4263             const a = document.createElement('a');
4264             a.href = url;
4265             a.download = 'memory_slots.zip';
4266             a.click();
4267             URL.revokeObjectURL(url);
4268
4269             return 'Memory slots saved to memory_slots.zip';
4270         });
4271     } catch (e) {
4272         return Promise.reject(`Error creating ZIP archive: ${e.message}`);
4273     }
4274 }
4275
4276 // Load memory slots from a ZIP file
4277 loadFromZip(zipData) {
4278     return JSZip.loadAsync(zipData)
4279         .then(zip => {
4280             // Try to load the complete structure first
4281             if (zip.file('all_memory_slots.json')) {
4282                 return zip.file('all_memory_slots.json').async('string')
4283                     .then(content => {
4284                         const data = JSON.parse(content);
4285                         this.memorySlots = data;
4286                         return 'Loaded all memory slots from ZIP archive';
4287                     });
4288             } else {
4289                 // Load individual JSON files
4290                 const jsonFiles = [];
4291
4292                 if (zip.folder('json')) {
4293                     zip.folder('json').forEach((relativePath, file) => {
4294                         if (!file.dir) {
4295                             jsonFiles.push(
4296                                 file.async('string').then(content => {
4297                                     const match = relativePath.match(/file_(\d+)\.json/);
4298                                     if (match) {
4299                                         const fileNumber = match[1];
4300                                         this.memorySlots[fileNumber] = JSON.parse(content);
4301                                     }
4302                                 })
4303                             );
4304                         }
4305                     });
4306                 }
4307
4308                 if (jsonFiles.length > 0) {
4309                     return Promise.all(jsonFiles)
4310                         .then(() => `Loaded ${jsonFiles.length} files from ZIP archive`);
4311                 } else {
4312                     return Promise.reject('No valid files found in ZIP archive');
4313                 }
4314             }
4315         });
4316     }
4317 }

```

```

4318 generateCharSet() {
4319     let chars =
4320         "abcdefghijklmnopqrstuvwxyz" +
4321         "ABCDEFGHIJKLMNOPQRSTUVWXYZ" +
4322         "0123456789" +
4323         "!@#$%^&*()-_=[{}|;:,<>/?" +
4324         "\t\n\r" +
4325         "~`'\\"";
4326
4327     // Ensure exactly 100 characters
4328     if (chars.length > 100) {
4329         chars = chars.substring(0, 100);
4330     } else if (chars.length < 100) {
4331         // Pad with additional characters if needed
4332         while (chars.length < 100) {
4333             chars += '?';
4334         }
4335     }
4336
4337     return chars;
4338 }
4339
4340 async generateCombinations(chars, n, generateType, fileName, progressCallback) {
4341     // The number of possible combinations is (k+1)^n
4342     const k = chars.length - 1;
4343     const nbrComb = Math.pow(k + 1, n);
4344
4345     let fileContent = '';
4346
4347     if (nbrComb > 1000000) {
4348         progressCallback(0, "Warning: Generating a large number of combinations. This may take a while.");
4349     }
4350
4351     // Clear any existing memory slots that would be affected
4352     if (generateType === 'singleFile') {
4353         // Clear just the target file
4354         this.memorySlots[fileName] = {};
4355     } else if (generateType === 'multiFile') {
4356         // We'll create new files, no need to clear
4357     }
4358
4359     // To avoid blocking the UI for too long, break up the work
4360     const batchSize = 10000; // Process in batches to keep UI responsive
4361     const totalBatches = Math.ceil(nbrComb / batchSize);
4362
4363     for (let batch = 0; batch < totalBatches; batch++) {
4364         await new Promise(resolve => setTimeout(resolve, 0)); // Allow UI to update
4365
4366         const startRow = batch * batchSize;
4367         const endRow = Math.min((batch + 1) * batchSize, nbrComb);
4368
4369         for (let row = startRow; row < endRow; row++) {
4370             const id = row + 1;
4371             let combination = '';
4372
4373             for (let col = n - 1; col >= 0; col--) {
4374                 const rdiv = Math.pow(k + 1, col);
4375                 const cell = Math.floor(row / rdiv) % (k + 1);
4376                 combination += chars[cell];
4377             }
4378
4379             // Handle different generation types
4380             if (generateType === 'singleFile') {
4381                 // Store all combinations in a single file with the given name
4382                 const slotNumber = id.toString();
4383                 this.memorySlots[fileName][slotNumber] = combination;
4384             }
4385             else if (generateType === 'multiFile') {
4386                 // Each combination gets its own file
4387                 const fileNumber = id.toString();
4388                 if (!this.memorySlots[fileNumber]) {
4389                     this.memorySlots[fileNumber] = {};
4390                 }

```

```

4391         // Always use slot 1 for each file
4392         this.memorySlots[fileNumber]['1'] = combination;
4393     }
4394
4395     // Add to file content for download
4396     if (generateType === 'singleFile') {
4397         fileContent += `SLOT ${id}\n${combination}\n-----\n`;
4398     } else {
4399         fileContent += `FILE ${id}\nSLOT 1\n${combination}\n-----\n`;
4400     }
4401 }
4402
4403 // Update progress approximately every 5%
4404 if (batch % Math.max(1, Math.floor(totalBatches / 20)) === 0 || batch === totalBatches - 1) {
4405     const progress = ((batch + 1) / totalBatches) * 100;
4406     progressCallback(progress, `Generated ${Math.min((batch + 1) * batchSize, nbrComb)} of ${nbrComb} combinations (${progress.toFixed(1)}%)`);
4407 }
4408 }
4409
4410 // Format the complete output in a way that's ready for the application
4411 let finalContent = `GENERATED COMBINATIONS\n`;
4412 finalContent += `Generation Mode: ${generateType === 'singleFile' ? 'Single File' : 'One File Per Combination'}\n`;
4413 finalContent += `(k+1)^n = (${k} + 1)^${n} = ${nbrComb}\n`;
4414 finalContent += `=====\n`;
4415 finalContent += fileContent;
4416 finalContent += `\n\nEnd. Total combinations: ${nbrComb}`;
4417
4418 return {
4419     content: finalContent,
4420     count: nbrComb,
4421     type: generateType,
4422     fileName: fileName
4423 };
4424 }
4425
4426 // Import generated combinations from text based on generation type
4427 importCombinationsFromText(text, type, fileName) {
4428     try {
4429         if (type === 'singleFile') {
4430             // Parse single file format
4431             const pattern = /SLOT (\d+)\n([\s\S]*?)(?=-{20}|\Z)/g;
4432             let match;
4433             let count = 0;
4434
4435             // Create or clear the target file
4436             if (!this.memorySlots[fileName]) {
4437                 this.memorySlots[fileName] = {};
4438             } else {
4439                 // Clear existing slots
4440                 this.memorySlots[fileName] = {};
4441             }
4442
4443             while ((match = pattern.exec(text)) !== null) {
4444                 const slotNumber = match[1];
4445                 let value = match[2].trim();
4446
4447                 // Store in memory slots
4448                 this.memorySlots[fileName][slotNumber] = value;
4449                 count++;
4450             }
4451
4452             return `Imported ${count} combinations into file "${fileName}`;
4453         }
4454         else if (type === 'multiFile') {
4455             // Parse multi-file format
4456             const filePattern = /FILE (\d+)\nSLOT (\d+)\n([\s\S]*?)(?=-{20}|\Z)/g;
4457             let fileMatch;
4458             let count = 0;
4459
4460             while ((fileMatch = filePattern.exec(text)) !== null) {
4461                 const fileNumber = fileMatch[1];
4462                 const slotNumber = fileMatch[2];
4463                 let value = fileMatch[3].trim();

```



```

4464
4465         // Create file if it doesn't exist
4466         if (!this.memorySlots[fileNumber]) {
4467             this.memorySlots[fileNumber] = {};
4468         }
4469
4470         // Store in memory slots
4471         this.memorySlots[fileNumber][slotNumber] = value;
4472         count++;
4473     }
4474
4475     // Fallback for standard format without FILE headers
4476     if (count === 0) {
4477         const pattern = /SLOT (\d+)\n([\s\S]*?) (?:\-{20}|\Z)/g;
4478         let match;
4479         let fileNumber = 1;
4480
4481         while ((match = pattern.exec(text)) !== null) {
4482             const slotNumber = match[1];
4483             let value = match[2].trim();
4484
4485             // Create file if it doesn't exist
4486             if (!this.memorySlots[fileNumber.toString()]) {
4487                 this.memorySlots[fileNumber.toString()] = {};
4488             }
4489
4490             // Store in memory slots
4491             this.memorySlots[fileNumber.toString()][slotNumber] = value;
4492             count++;
4493             fileNumber++;
4494         }
4495     }
4496
4497     return `Imported ${count} combinations into separate files`;
4498 }
4499
4500     return "Unknown generation type for import";
4501 } catch (e) {
4502     return `Error importing combinations: ${e.message}`;
4503 }
4504 }
4505
4506 // Clear all memory slots
4507 clearAll() {
4508     this.memorySlots = {};
4509     return 'All memory slots cleared';
4510 }
4511 }
4512
4513 // Proof by Contradiction Engine integration
4514 class ProofByContradictionEngine {
4515     constructor() {
4516         this.statementsFileId = "statements"; // File ID for storing statements in memory manager
4517         this.char_set = "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ+-*/=<>!@#$$%^&()[]{};,:.~`|";
4518         this.MAX_STATEMENT_LENGTH = 256;
4519         this.NUM_CHARACTERS = this.char_set.length;
4520
4521         // Initialize the statements file
4522         if (!memoryManager.memorySlots[this.statementsFileId]) {
4523             memoryManager.memorySlots[this.statementsFileId] = {};
4524         }
4525     }
4526
4527     clearStatements() {
4528         memoryManager.memorySlots[this.statementsFileId] = {};
4529         return "All statements cleared.";
4530     }
4531
4532     addStatement(statement) {
4533         // Get the next slot number
4534         let slotNumber = 1;
4535         const existingSlots = Object.keys(memoryManager.memorySlots[this.statementsFileId])
4536             .map(key => parseInt(key))

```

```

4537         .filter(num => !isNaN(num));
4538
4539         if (existingSlots.length > 0) {
4540             slotNumber = Math.max(...existingSlots) + 1;
4541         }
4542
4543         // Store the statement
4544         memoryManager.memorySlots[this.statementsFileId][slotNumber] = statement;
4545         return slotNumber;
4546     }
4547
4548     getStatements() {
4549         const statements = [];
4550         const slots = memoryManager.memorySlots[this.statementsFileId];
4551
4552         const sortedKeys = Object.keys(slots)
4553             .map(key => parseInt(key))
4554             .filter(num => !isNaN(num))
4555             .sort((a, b) => a - b);
4556
4557         for (const key of sortedKeys) {
4558             statements.push(slots[key]);
4559         }
4560
4561         return statements;
4562     }
4563
4564     generateStatement() {
4565         const length = Math.floor(Math.random() * this.MAX_STATEMENT_LENGTH) + 1;
4566         let statement = '';
4567
4568         for (let i = 0; i < length; i++) {
4569             statement += this.char_set[Math.floor(Math.random() * this.NUM_CHARACTERS)];
4570         }
4571
4572         return statement;
4573     }
4574
4575     generateStatements(numStatements) {
4576         this.clearStatements();
4577
4578         for (let i = 0; i < numStatements; i++) {
4579             const statement = this.generateStatement();
4580             this.addStatement(statement);
4581         }
4582
4583         return `${numStatements} statements generated.`;
4584     }
4585
4586     checkContradiction() {
4587         const statements = this.getStatements();
4588
4589         // Implementation of rule_statement_not_statement
4590         for (let i = 0; i < statements.length; i++) {
4591             const statement = statements[i];
4592             const notStatement = `NOT ${statement}`;
4593
4594             for (let j = 0; j < statements.length; j++) {
4595                 if (i !== j && statements[j] === notStatement) {
4596                     return {
4597                         found: true,
4598                         indices: [i, j],
4599                         statements: [statement, notStatement]
4600                     };
4601                 }
4602             }
4603         }
4604
4605         // Implementation of rule_identical_statements
4606         for (let i = 0; i < statements.length; i++) {
4607             for (let j = i + 1; j < statements.length; j++) {
4608                 if (statements[i] === statements[j]) {
4609                     return {

```

```

4610         found: true,
4611         indices: [i, j],
4612         statements: [statements[i], statements[j]],
4613         rule: "identical_statements"
4614     };
4615 }
4616 }
4617 }
4618
4619     return { found: false };
4620 }
4621
4622 processInstruction(instruction) {
4623     const parts = instruction.split(' ');
4624     const cmd = parts[0];
4625
4626     switch (cmd) {
4627         case 'ASSERT':
4628             if (parts.length < 2) {
4629                 return "Error: ASSERT requires a statement.";
4630             }
4631             const statement = parts.slice(1).join(' ');
4632             const slotNumber = this.addStatement(statement);
4633             return `Added statement "${statement}" at position ${slotNumber}.`;
4634
4635         case 'NOT':
4636             if (parts.length < 2) {
4637                 return "Error: NOT requires a statement.";
4638             }
4639             const notStatement = parts.slice(1).join(' ');
4640             const notSlotNumber = this.addStatement(`NOT ${notStatement}`);
4641             return `Added statement "NOT ${notStatement}" at position ${notSlotNumber}.`;
4642
4643         case 'AND':
4644             if (parts.length < 3) {
4645                 return "Error: AND requires two statements.";
4646             }
4647
4648             // More sophisticated parsing for statement boundaries
4649             let andStatement1 = '';
4650             let andStatement2 = '';
4651             let inQuotes = false;
4652             let bracketCount = 0;
4653             let splitIndex = -1;
4654
4655             // First, check if statements are in quotes
4656             const fullText = parts.slice(1).join(' ');
4657             const quotedRegex = /^"(["]*)"(?:\s+|\s*,\s*)"(["]*)"$/;
4658             const quotedMatch = fullText.match(quotedRegex);
4659
4660             if (quotedMatch) {
4661                 // If statements are wrapped in quotes, use those boundaries
4662                 andStatement1 = quotedMatch[1];
4663                 andStatement2 = quotedMatch[2];
4664             } else {
4665                 // Otherwise, try to identify statement boundaries by parsing brackets and logical operators
4666                 for (let i = 1; i < parts.length; i++) {
4667                     const word = parts[i];
4668
4669                     // Track quotes
4670                     for (let j = 0; j < word.length; j++) {
4671                         if (word[j] === '"' && (j === 0 || word[j-1] !== '\\')) {
4672                             inQuotes = !inQuotes;
4673                         }
4674                     }
4675
4676                     // Track brackets (only when not in quotes)
4677                     if (!inQuotes) {
4678                         for (let j = 0; j < word.length; j++) {
4679                             if (word[j] === '(' || word[j] === '{' || word[j] === '[') {
4680                                 bracketCount++;
4681                             } else if (word[j] === ')' || word[j] === '}' || word[j] === ']') {
4682                                 bracketCount--;

```

```

4683         }
4684     }
4685 }
4686
4687 // Check for logical keyword boundaries when not in quotes and brackets are balanced
4688 if (!inQuotes && bracketCount === 0 &&
4689     (word === 'AND' || word === 'OR' || word === 'NOT' || word === 'IMPLIES')) {
4690     splitIndex = i;
4691     break;
4692 }
4693 }
4694
4695 // If a logical keyword was found as a boundary
4696 if (splitIndex > 1) {
4697     andStatement1 = parts.slice(1, splitIndex).join(' ');
4698     andStatement2 = parts.slice(splitIndex + 1).join(' ');
4699 } else {
4700     // Default to splitting halfway if no clear boundary found
4701     const midpoint = Math.ceil(parts.length / 2);
4702     andStatement1 = parts.slice(1, midpoint).join(' ');
4703     andStatement2 = parts.slice(midpoint).join(' ');
4704 }
4705 }
4706
4707 // Validate both parts are present
4708 if (!andStatement1 || !andStatement2) {
4709     return "Error: AND requires two valid statements. Use quotes for complex statements: AND \"statement1\" \"statement2\"";
4710 }
4711
4712 // Check the statements repository to see if these statements exist
4713 const statements = this.getStatements();
4714 let statement1Exists = false;
4715 let statement2Exists = false;
4716
4717 // Try to find exact matches first
4718 for (const stmt of statements) {
4719     if (stmt === andStatement1) statement1Exists = true;
4720     if (stmt === andStatement2) statement2Exists = true;
4721 }
4722
4723 // If statements don't exist, check if they're statement IDs
4724 if (!statement1Exists && /^\\d+$/ .test(andStatement1)) {
4725     const stmtIndex = parseInt(andStatement1) - 1;
4726     if (stmtIndex >= 0 && stmtIndex < statements.length) {
4727         andStatement1 = statements[stmtIndex];
4728         statement1Exists = true;
4729     }
4730 }
4731
4732 if (!statement2Exists && /^\\d+$/ .test(andStatement2)) {
4733     const stmtIndex = parseInt(andStatement2) - 1;
4734     if (stmtIndex >= 0 && stmtIndex < statements.length) {
4735         andStatement2 = statements[stmtIndex];
4736         statement2Exists = true;
4737     }
4738 }
4739
4740 // Warn if statements don't exist in the repository
4741 let warningMessage = "";
4742 if (!statement1Exists) {
4743     warningMessage += `Warning: Statement "${andStatement1}" is not in the repository.\\n`;
4744 }
4745 if (!statement2Exists) {
4746     warningMessage += `Warning: Statement "${andStatement2}" is not in the repository.\\n`;
4747 }
4748
4749 // Create and store the AND statement
4750 const andStatement = `(${andStatement1}) AND (${andStatement2})`;
4751 const andSlotNumber = this.addStatement(andStatement);
4752
4753 // Return result with optional warnings
4754 return `${warningMessage}Added statement "${andStatement}" at position ${andSlotNumber}.`;
4755

```

```

4756     case 'OR':
4757         if (parts.length < 3) {
4758             return "Error: OR requires two statements.";
4759         }
4760         const orStatement1 = parts[1];
4761         const orStatement2 = parts.slice(2).join(' ');
4762         const orSlotNumber = this.addStatement(`${orStatement1} OR ${orStatement2}`);
4763         return `Added statement "${orStatement1} OR ${orStatement2}" at position ${orSlotNumber}.`;
4764
4765     case 'IMPLIES':
4766         if (parts.length < 3) {
4767             return "Error: IMPLIES requires two statements.";
4768         }
4769         const impliesStatement1 = parts[1];
4770         const impliesStatement2 = parts.slice(2).join(' ');
4771         const impliesSlotNumber = this.addStatement(`${impliesStatement1} IMPLIES ${impliesStatement2}`);
4772         return `Added statement "${impliesStatement1} IMPLIES ${impliesStatement2}" at position ${impliesSlotNumber}.`;
4773
4774     case 'CONTRADICTION':
4775         if (parts.length < 3) {
4776             return "Error: CONTRADICTION requires two statements.";
4777         }
4778
4779         const contradictionStatement1 = parts[1];
4780         const contradictionStatement2 = parts.slice(2).join(' ');
4781
4782         // Check if the statements exist in our framework
4783         statements = this.getStatements();
4784         let found1 = false, found2 = false;
4785         let index1 = -1, index2 = -1;
4786
4787         for (let i = 0; i < statements.length; i++) {
4788             if (statements[i] === contradictionStatement1) {
4789                 found1 = true;
4790                 index1 = i;
4791             }
4792             if (statements[i] === contradictionStatement2) {
4793                 found2 = true;
4794                 index2 = i;
4795             }
4796         }
4797
4798         if (found1 && found2) {
4799             return `Contradiction found between statements ${index1 + 1} and ${index2 + 1}: \n  ${contradictionStatement1} \n  ${contradictionStatement2}`;
4800         } else {
4801             return "No contradiction found between the specified statements.";
4802         }
4803
4804     default:
4805         return `Unknown instruction: ${cmd}`;
4806 }
4807 }
4808 }
4809
4810 // Load JSZip library dynamically
4811 const loadJSZip = () => {
4812     return new Promise((resolve, reject) => {
4813         if (window.JSZip) {
4814             resolve();
4815             return;
4816         }
4817
4818         const script = document.createElement('script');
4819         script.src = 'https://cdnjs.cloudflare.com/ajax/libs/jszip/3.10.1/jszip.min.js';
4820         script.onload = () => resolve();
4821         script.onerror = () => reject(new Error('Failed to load JSZip library'));
4822         document.head.appendChild(script);
4823     });
4824 };
4825
4826 // Initialize the applications
4827 const memoryManager = new MemorySlotManager();
4828 const logicEngine = new ProofByContradictionEngine();

```

```

4829
4830 // DOM Elements
4831 const terminal = document.getElementById('terminal');
4832 const commandInput = document.getElementById('commandInput');
4833 const multilineInput = document.getElementById('multilineInput');
4834 const submitCommandBtn = document.getElementById('submitCommand');
4835 const submitMultilineBtn = document.getElementById('submitMultiline');
4836 const cancelMultilineBtn = document.getElementById('cancelMultiline');
4837 const memoryDisplay = document.getElementById('memoryDisplay');
4838 const systemTimeEl = document.getElementById('systemTime');
4839
4840 const logicTerminal = document.getElementById('logicTerminal');
4841 const logicCommandInput = document.getElementById('logicCommandInput');
4842 const submitLogicCommandBtn = document.getElementById('submitLogicCommand');
4843 const checkContradictionsBtn = document.getElementById('checkContradictions');
4844 const clearStatementsBtn = document.getElementById('clearStatements');
4845 const generateStatementsBtn = document.getElementById('generateStatements');
4846
4847 const generationModal = document.getElementById('generationModal');
4848 const generationProgress = document.getElementById('generationProgress');
4849 const generationStatus = document.getElementById('generationStatus');
4850 const downloadGeneratedBtn = document.getElementById('downloadGeneratedBtn');
4851 const importGeneratedBtn = document.getElementById('importGeneratedBtn');
4852 const startGenerationBtn = document.getElementById('startGenerationBtn');
4853 const fileNameInput = document.getElementById('fileNameInput');
4854 let generatedResults = null;
4855
4856 // Set current time
4857 const now = new Date();
4858 systemTimeEl.textContent = `SYSTEM INITIALIZED: ${now.toISOString().replace('T', ' ').substr(0, 19)}`;
4859
4860 // Add initial welcome message
4861 appendToTerminal('INTEGRATED MEMORY & LOGIC FRAMEWORK', 'system');
4862 appendToTerminal('Type "help" for a list of commands.', 'system');
4863
4864 // Add initial welcome message to logic terminal
4865 appendToLogicTerminal('PROOF BY CONTRADICTION ENGINE INITIALIZED', 'system');
4866 appendToLogicTerminal('Type "help" for a list of instructions.', 'system');
4867
4868 // Set up tab navigation
4869 document.querySelectorAll('.nav-tab').forEach(tab => {
4870   tab.addEventListener('click', function() {
4871     // Remove active class from all tabs
4872     document.querySelectorAll('.nav-tab').forEach(t => {
4873       t.classList.remove('active');
4874     });
4875
4876     // Add active class to clicked tab
4877     this.classList.add('active');
4878
4879     // Hide all tab content
4880     document.querySelectorAll('.tab-content').forEach(content => {
4881       content.classList.remove('active');
4882     });
4883
4884     // Show the selected tab content
4885     const tabId = this.getAttribute('data-tab');
4886     document.getElementById(`${tabId}Tab`).classList.add('active');
4887   });
4888 });
4889
4890 // Event handler for generate type change
4891 document.querySelectorAll('input[name="generateType"]').forEach(radio => {
4892   radio.addEventListener('change', function() {
4893     const fileInputContainer = document.getElementById('fileInputContainer');
4894     if (this.value === 'singleFile') {
4895       fileInputContainer.style.display = 'block';
4896     } else {
4897       fileInputContainer.style.display = 'none';
4898     }
4899   });
4900 });
4901

```

```

4902 // Start generation button handler
4903 startGenerationBtn.addEventListener('click', async () => {
4904     const generateType = document.querySelector('input[name="generateType"]:checked').value;
4905     let fileName = fileNameInput.value.trim();
4906
4907     // Default to "combinations" if empty
4908     if (generateType === 'singleFile' && !fileName) {
4909         fileName = 'combinations';
4910     }
4911
4912     try {
4913         startGenerationBtn.disabled = true;
4914         downloadGeneratedBtn.disabled = true;
4915         importGeneratedBtn.disabled = true;
4916         generationProgress.style.width = '0%';
4917         generationStatus.textContent = 'Starting generation...';
4918
4919         // Get n value from stored data
4920         const n = parseInt(generationModal.dataset.n);
4921         const chars = generationModal.dataset.customChars || memoryManager.generateCharSet();
4922
4923         generatedResults = await memoryManager.generateCombinations(
4924             chars,
4925             n,
4926             generateType,
4927             fileName,
4928             (progress, status) => {
4929                 generationProgress.style.width = `${progress}%`;
4930                 generationStatus.textContent = status;
4931             }
4932         );
4933
4934         downloadGeneratedBtn.disabled = false;
4935         importGeneratedBtn.disabled = false;
4936
4937         let modeDesc = generateType === 'singleFile' ?
4938             `single file "${fileName}"` :
4939             'separate files (one per combination)';
4940
4941         generationStatus.textContent = `Generation complete! ${generatedResults.count} combinations generated in ${modeDesc}.`;
4942
4943         // If the count is manageable, automatically import
4944         if (generatedResults.count <= 1000) {
4945             const result = memoryManager.importCombinationsFromText(
4946                 generatedResults.content,
4947                 generatedResults.type,
4948                 generatedResults.fileName
4949             );
4950             appendToTerminal(result, 'system');
4951             updateMemoryDisplay();
4952         }
4953     } catch (e) {
4954         generationStatus.textContent = `Error during generation: ${e.message}`;
4955         appendToTerminal(`Error during generation: ${e.message}`, 'system');
4956     } finally {
4957         startGenerationBtn.disabled = false;
4958     }
4959 });
4960
4961 // Generate statements button handler
4962 generateStatementsBtn.addEventListener('click', () => {
4963     document.getElementById('statementsGenerationModal').style.display = 'block';
4964 });
4965
4966 // Start statements generation button handler
4967 document.getElementById('startStatementsGenBtn').addEventListener('click', () => {
4968     const numStatements = parseInt(document.getElementById('numStatementsInput').value);
4969     if (isNaN(numStatements) || numStatements <= 0) {
4970         appendToLogicTerminal('Error: Please enter a valid positive number.', 'system');
4971         return;
4972     }
4973
4974     const result = logicEngine.generateStatements(numStatements);

```

```

4975     appendToLogicTerminal(result, 'system');
4976     document.getElementById('statementsGenerationModal').style.display = 'none';
4977
4978     // Display the current statements
4979     const statements = logicEngine.getStatements();
4980     let output = 'Current statements in framework:';
4981     for (let i = 0; i < statements.length; i++) {
4982         output += `\n${i + 1}. ${statements[i]}`;
4983     }
4984     appendToLogicTerminal(output, 'system');
4985 });
4986
4987 // Check contradictions button handler
4988 checkContradictionsBtn.addEventListener('click', () => {
4989     const statements = logicEngine.getStatements();
4990
4991     if (statements.length === 0) {
4992         appendToLogicTerminal('No statements in framework.', 'system');
4993         return;
4994     }
4995
4996     let output = 'Current statements in framework:';
4997     for (let i = 0; i < statements.length; i++) {
4998         output += `\n${i + 1}. ${statements[i]}`;
4999     }
5000     appendToLogicTerminal(output, 'system');
5001
5002     const result = logicEngine.checkContradiction();
5003     if (result.found) {
5004         let contradictionMsg = `Contradiction found between statements ${result.indices[0] + 1} and ${result.indices[1] + 1}: \n`;
5005         contradictionMsg += ` ${result.statements[0]} \n  ${result.statements[1]} \n`;
5006
5007         if (result.rule) {
5008             contradictionMsg += ` (Detected by ${result.rule} rule) `;
5009         }
5010
5011         appendToLogicTerminal(contradictionMsg, 'system');
5012     } else {
5013         appendToLogicTerminal('No contradictions found in the current framework.', 'system');
5014     }
5015 });
5016
5017 // Clear statements button handler
5018 clearStatementsBtn.addEventListener('click', () => {
5019     const result = logicEngine.clearStatements();
5020     appendToLogicTerminal(result, 'system');
5021 });
5022
5023 // Enable tab key in the multiline input
5024 multilineInput.addEventListener('keydown', function(e) {
5025     if (e.key === 'Tab') {
5026         e.preventDefault();
5027
5028         // Insert a tab at the current cursor position
5029         const start = this.selectionStart;
5030         const end = this.selectionEnd;
5031         const value = this.value;
5032
5033         // Insert the tab character
5034         this.value = value.substring(0, start) + '\t' + value.substring(end);
5035
5036         // Move the cursor after the tab
5037         this.selectionStart = this.selectionEnd = start + 1;
5038     }
5039 });
5040
5041 // Event listeners for memory manager
5042 commandInput.addEventListener('keydown', (e) => {
5043     if (e.key === 'Enter') {
5044         const command = commandInput.value.trim();
5045         submitCommand(command);
5046     }
5047 });

```



```
5048
5049 submitCommandBtn.addEventListener('click', () => {
5050     const command = commandInput.value.trim();
5051     submitCommand(command);
5052 });
5053
5054 submitMultilineBtn.addEventListener('click', () => {
5055     const multilineValue = multilineInput.value;
5056     finishMultilineInput(multilineValue);
5057 });
5058
5059 cancelMultilineBtn.addEventListener('click', () => {
5060     // Reset the UI
5061     resetToCommandMode();
5062     appendToTerminal('Multi-line input canceled.', 'system');
5063 });
5064
5065 // Event listeners for logic engine
5066 logicCommandInput.addEventListener('keydown', (e) => {
5067     if (e.key === 'Enter') {
5068         const command = logicCommandInput.value.trim();
5069         submitLogicCommand(command);
5070     }
5071 });
5072
5073 submitLogicCommandBtn.addEventListener('click', () => {
5074     const command = logicCommandInput.value.trim();
5075     submitLogicCommand(command);
5076 });
5077
5078 downloadGeneratedBtn.addEventListener('click', () => {
5079     if (generatedResults) {
5080         const blob = new Blob([generatedResults.content], { type: 'text/plain' });
5081         const url = URL.createObjectURL(blob);
5082         const a = document.createElement('a');
5083         a.href = url;
5084         a.download = 'GENERATED_COMBINATIONS.txt';
5085         a.click();
5086         URL.revokeObjectURL(url);
5087     }
5088 });
5089
5090 importGeneratedBtn.addEventListener('click', () => {
5091     if (generatedResults) {
5092         // Import the generated combinations into memory slots
5093         const result = memoryManager.importCombinationsFromText(
5094             generatedResults.content,
5095             generatedResults.type,
5096             generatedResults.fileName
5097         );
5098         appendToTerminal(result, 'system');
5099         updateMemoryDisplay();
5100         document.getElementById('generationModal').style.display = 'none';
5101     }
5102 });
5103
5104 // Function to append text to the terminal
5105 function appendToTerminal(text, type = 'command') {
5106     const entryDiv = document.createElement('div');
5107
5108     if (type === 'input') {
5109         entryDiv.className = 'terminal-input';
5110         entryDiv.innerHTML = `<span>></span> ${escapeHtml(text)}`;
5111     } else if (type === 'system') {
5112         entryDiv.className = 'terminal-system';
5113         entryDiv.textContent = text;
5114     } else {
5115         entryDiv.className = 'terminal-output';
5116         entryDiv.innerHTML = text.replace(/\n/g, '<br>');
5117     }
5118
5119     terminal.appendChild(entryDiv);
5120     terminal.scrollTop = terminal.scrollHeight;
```

```

5121     }
5122
5123     // Function to append text to the logic terminal
5124     function appendToLogicTerminal(text, type = 'command') {
5125         const entryDiv = document.createElement('div');
5126
5127         if (type === 'input') {
5128             entryDiv.className = 'terminal-input';
5129             entryDiv.innerHTML = `<span>></span> ${escapeHtml(text)}`;
5130         } else if (type === 'system') {
5131             entryDiv.className = 'terminal-system';
5132             entryDiv.textContent = text;
5133         } else {
5134             entryDiv.className = 'terminal-output';
5135             entryDiv.innerHTML = text.replace(/\n/g, '<br>');
5136         }
5137
5138         logicTerminal.appendChild(entryDiv);
5139         logicTerminal.scrollTop = logicTerminal.scrollHeight;
5140     }
5141
5142     function escapeHtml(text) {
5143         const div = document.createElement('div');
5144         div.textContent = text;
5145         return div.innerHTML;
5146     }
5147
5148     // Clear the terminal
5149     function clearTerminal() {
5150         terminal.innerHTML = '';
5151         appendToTerminal('Terminal cleared.', 'system');
5152     }
5153
5154     // Clear the logic terminal
5155     function clearLogicTerminal() {
5156         logicTerminal.innerHTML = '';
5157         appendToLogicTerminal('Terminal cleared.', 'system');
5158     }
5159
5160     // Process a command for memory manager
5161     async function submitCommand(command) {
5162         if (!command) return;
5163
5164         appendToTerminal(command, 'input');
5165         commandInput.value = '';
5166
5167         const commandParts = command.split(' ');
5168         const cmd = commandParts[0].toLowerCase();
5169
5170         try {
5171             switch (cmd) {
5172                 case 'help':
5173                 case '?':
5174                     document.getElementById('helpModal').style.display = 'block';
5175                     break;
5176
5177                 case 'assign':
5178                     if (commandParts.length < 3) {
5179                         appendToTerminal('Error: Missing arguments for assign command.', 'system');
5180                         break;
5181                     }
5182
5183                     const fileNumber = commandParts[1];
5184                     const slotNumber = commandParts[2];
5185
5186                     // Switch to multi-line input mode
5187                     switchToMultilineMode('assign', fileNumber, slotNumber);
5188                     break;
5189
5190                 case 'read':
5191                     if (commandParts.length < 3) {
5192                         appendToTerminal('Error: Missing arguments for read command.', 'system');
5193                         break;

```

```

5194     }
5195
5196     const readResult = memoryManager.readSlot(commandParts[1], commandParts[2]);
5197     appendToTerminal(readResult, 'system');
5198     break;
5199
5200 case 'last_slot':
5201     if (commandParts.length < 2) {
5202         appendToTerminal('Error: Missing file number for last_slot command.', 'system');
5203         break;
5204     }
5205
5206     const lastSlot = memoryManager.getLastSlotNumber(commandParts[1]);
5207     if (lastSlot !== null) {
5208         appendToTerminal(`Last slot number in file ${commandParts[1]} is ${lastSlot}.`, 'system');
5209     } else {
5210         appendToTerminal(`No slots found in file ${commandParts[1]}.`, 'system');
5211     }
5212     break;
5213
5214 case 'search':
5215     // Switch to multi-line input mode
5216     switchToMultilineMode('search');
5217     break;
5218
5219 case 'call':
5220     if (commandParts.length < 4) {
5221         appendToTerminal('Error: Missing arguments for call command.', 'system');
5222         break;
5223     }
5224
5225     try {
5226         const callFileNumber = commandParts[1];
5227         const startSlot = parseInt(commandParts[2]);
5228         const endSlot = parseInt(commandParts[3]);
5229         const callResult = memoryManager.callSlotRange(callFileNumber, startSlot, endSlot);
5230         appendToTerminal(callResult, 'system');
5231     } catch (e) {
5232         appendToTerminal('Error: Slot numbers must be integers.', 'system');
5233     }
5234     break;
5235
5236 case 'export':
5237     if (commandParts.length < 2) {
5238         appendToTerminal('Error: Missing file number for export command.', 'system');
5239         break;
5240     }
5241
5242     const exportResult = memoryManager.exportToTextFile(commandParts[1]);
5243     appendToTerminal(exportResult, 'system');
5244     break;
5245
5246 case 'generate':
5247     if (commandParts.length < 2) {
5248         appendToTerminal('Error: Please specify the number of cells (n).', 'system');
5249         break;
5250     }
5251
5252     const n = parseInt(commandParts[1]);
5253     if (n <= 0) {
5254         appendToTerminal('Error: n must be a positive integer.', 'system');
5255         break;
5256     }
5257
5258     if (n > 4) {
5259         appendToTerminal('Warning: Large values of n will generate very large outputs.', 'system');
5260         if (!confirm('Continue with generating a large number of combinations? n=' + n)) {
5261             appendToTerminal('Generation canceled.', 'system');
5262             break;
5263         }
5264     }
5265
5266     const charSet = memoryManager.generateCharSet();

```

```

5267     appendToTerminal(`Using standard character set (${charSet.length} chars)\`, 'system');
5268
5269     // Show the generation modal with options
5270     generationModal.style.display = 'block';
5271     generationProgress.style.width = '0%';
5272     generationStatus.textContent = 'Select options and click "Start Generation"';
5273
5274     // Set up initial state for the modal
5275     document.querySelector('#generateSingleFile').checked = true;
5276     document.getElementById('fileInputContainer').style.display = 'block';
5277     fileNameInput.value = 'combinations';
5278     downloadGeneratedBtn.disabled = true;
5279     importGeneratedBtn.disabled = true;
5280     startGenerationBtn.disabled = false;
5281
5282     // Store n value for later use
5283     generationModal.dataset.n = n;
5284     generationModal.dataset.customChars = '';
5285
5286     break;
5287
5288     case 'custom_generate':
5289         if (commandParts.length < 2) {
5290             appendToTerminal('Error: Please specify the number of cells (n).', 'system');
5291             break;
5292         }
5293
5294         const customN = parseInt(commandParts[1]);
5295         if (customN <= 0) {
5296             appendToTerminal('Error: n must be a positive integer.', 'system');
5297             break;
5298         }
5299
5300         // Switch to multi-line input mode for custom character set
5301         switchToMultilineMode('custom_generate', customN);
5302         break;
5303
5304     case 'display':
5305     case 'show':
5306         updateMemoryDisplay();
5307         appendToTerminal('Memory display updated.', 'system');
5308         break;
5309
5310     case 'clear':
5311         clearTerminal();
5312         break;
5313
5314     case 'save_json':
5315         const saveFileNumber = commandParts.length > 1 ? commandParts[1] : null;
5316         const saveResult = memoryManager.saveToJSON(saveFileNumber);
5317         appendToTerminal(saveResult, 'system');
5318         break;
5319
5320     case 'save_all':
5321         await loadJSZip();
5322         try {
5323             const result = await memoryManager.saveAllAsZip();
5324             appendToTerminal(result, 'system');
5325         } catch (error) {
5326             appendToTerminal(error, 'system');
5327         }
5328         break;
5329
5330     case 'load_json':
5331         // Create a file input element
5332         const fileInput = document.createElement('input');
5333         fileInput.type = 'file';
5334         fileInput.accept = '.json';
5335         fileInput.style.display = 'none';
5336         document.body.appendChild(fileInput);
5337
5338         fileInput.onchange = function(e) {
5339             const file = e.target.files[0];

```

```

5340         if (!file) {
5341             appendToTerminal('No file selected.', 'system');
5342             return;
5343         }
5344
5345         const reader = new FileReader();
5346         reader.onload = function(e) {
5347             const result = memoryManager.loadJSON(e.target.result);
5348             appendToTerminal(result, 'system');
5349             updateMemoryDisplay();
5350         };
5351         reader.onerror = function() {
5352             appendToTerminal('Error reading file.', 'system');
5353         };
5354         reader.readAsText(file);
5355     };
5356
5357     fileInput.click();
5358     document.body.removeChild(fileInput);
5359     break;
5360
5361     case 'load_all':
5362         await loadJSZip();
5363
5364         // Create a file input element
5365         const zipInput = document.createElement('input');
5366         zipInput.type = 'file';
5367         zipInput.accept = '.zip';
5368         zipInput.style.display = 'none';
5369         document.body.appendChild(zipInput);
5370
5371         zipInput.onchange = async function(e) {
5372             const file = e.target.files[0];
5373             if (!file) {
5374                 appendToTerminal('No file selected.', 'system');
5375                 return;
5376             }
5377
5378             try {
5379                 const result = await memoryManager.loadFromZip(file);
5380                 appendToTerminal(result, 'system');
5381                 updateMemoryDisplay();
5382             } catch (error) {
5383                 appendToTerminal(`Error: ${error}`, 'system');
5384             }
5385         };
5386
5387         zipInput.click();
5388         document.body.removeChild(zipInput);
5389         break;
5390
5391     case 'exit':
5392     case 'quit':
5393         appendToTerminal('This is a web application. To exit, close the browser tab.', 'system');
5394         break;
5395
5396     default:
5397         appendToTerminal(`Unknown command: ${cmd}`, 'system');
5398         appendToTerminal("Type 'help' for available commands.", 'system');
5399     }
5400 } catch (error) {
5401     appendToTerminal(`Error executing command: ${error.message}`, 'system');
5402 }
5403 }
5404
5405 // Process a command for logic engine
5406 function submitLogicCommand(command) {
5407     if (!command) return;
5408
5409     appendToLogicTerminal(command, 'input');
5410     logicCommandInput.value = '';
5411
5412     if (command.toLowerCase() === 'help') {

```

```

5413         document.getElementById('logicHelpModal').style.display = 'block';
5414         return;
5415     }
5416
5417     try {
5418         const result = logicEngine.processInstruction(command);
5419         appendToLogicTerminal(result, 'system');
5420     } catch (error) {
5421         appendToLogicTerminal(`Error: ${error.message}`, 'system');
5422     }
5423 }
5424
5425 // Switch to multi-line input mode
5426 function switchToMultilineMode(mode, ...args) {
5427     // Hide command input and button
5428     commandInput.style.display = 'none';
5429     submitCommandBtn.style.display = 'none';
5430
5431     // Show multi-line input, submit and cancel buttons
5432     multilineInput.style.display = 'block';
5433     submitMultilineBtn.style.display = 'inline-block';
5434     cancelMultilineBtn.style.display = 'inline-block';
5435
5436     // Clear the multi-line input
5437     multilineInput.value = '';
5438
5439     // Focus the multi-line input
5440     multilineInput.focus();
5441
5442     // Store the mode and arguments
5443     multilineInput.dataset.mode = mode;
5444     multilineInput.dataset.args = JSON.stringify(args);
5445
5446     // Set appropriate placeholder text
5447     if (mode === 'assign') {
5448         multilineInput.placeholder = `Enter value for file ${args[0]}, slot ${args[1]} (TAB key supported)`;
5449     } else if (mode === 'search') {
5450         multilineInput.placeholder = 'Enter search value (TAB key supported)';
5451     } else if (mode === 'custom_generate') {
5452         multilineInput.placeholder = 'Enter custom character set for combination generation';
5453     }
5454
5455     appendToTerminal(`Enter multi-line value for "${mode}" (Submit when finished):`, 'system');
5456 }
5457
5458 // Finish multi-line input
5459 async function finishMultilineInput(value) {
5460     const mode = multilineInput.dataset.mode;
5461     const args = JSON.parse(multilineInput.dataset.args || '[]');
5462
5463     resetToCommandMode();
5464
5465     if (mode === 'assign') {
5466         const [fileNumber, slotNumber] = args;
5467         const result = memoryManager.assignSlot(fileNumber, slotNumber, value);
5468         appendToTerminal(result, 'system');
5469         updateMemoryDisplay();
5470     } else if (mode === 'search') {
5471         if (!value.trim()) {
5472             appendToTerminal('Error: No search value provided.', 'system');
5473             return;
5474         }
5475
5476         const searchResults = memoryManager.searchValue(value);
5477         if (searchResults.length > 0) {
5478             let resultOutput = 'Search Results: ';
5479             for (const result of searchResults) {
5480                 resultOutput += `\nFile: ${result.file}, Slot: ${result.slot}, Value: \n${result.value}\n`;
5481             }
5482             appendToTerminal(resultOutput, 'system');
5483         } else {
5484             appendToTerminal('No results found.', 'system');
5485         }
5486     }
5487 }

```

```

5486     } else if (mode === 'custom_generate') {
5487         const customN = args[0];
5488         const customCharSet = value.trim() || memoryManager.generateCharSet();
5489
5490         appendToTerminal(`Using ${value.trim() ? 'custom' : 'default'} character set (${customCharSet.length} chars)\`, 'system');
5491
5492         // Show the generation modal with options
5493         generationModal.style.display = 'block';
5494         generationProgress.style.width = '0%';
5495         generationStatus.textContent = 'Select options and click "Start Generation"';
5496
5497         // Set up initial state for the modal
5498         document.querySelector('#generateSingleFile').checked = true;
5499         document.getElementById('fileInputContainer').style.display = 'block';
5500         fileNameInput.value = 'combinations';
5501         downloadGeneratedBtn.disabled = true;
5502         importGeneratedBtn.disabled = true;
5503         startGenerationBtn.disabled = false;
5504
5505         // Store data for later use
5506         generationModal.dataset.n = customN;
5507         generationModal.dataset.customChars = customCharSet;
5508     }
5509 }
5510
5511 // Reset to command mode
5512 function resetToCommandMode() {
5513     multilineInput.style.display = 'none';
5514     submitMultilineBtn.style.display = 'none';
5515     cancelMultilineBtn.style.display = 'none';
5516
5517     commandInput.style.display = 'block';
5518     submitCommandBtn.style.display = 'inline-block';
5519
5520     commandInput.focus();
5521 }
5522
5523 // Update the memory display
5524 function updateMemoryDisplay() {
5525     const memorySlots = memoryManager.memorySlots;
5526
5527     if (Object.keys(memorySlots).length === 0) {
5528         memoryDisplay.innerHTML = `
5529             <div style="text-align: center; color: #888; margin-top: 20px;">
5530                 No memory slots available.
5531                 <br><br>
5532                 Use the terminal to add memory slots.
5533             </div>
5534         `;
5535         return;
5536     }
5537
5538     let html = '';
5539
5540     // Sort file numbers numerically if possible
5541     const sortedFiles = Object.keys(memorySlots).sort((a, b) => {
5542         if (/^\d+$/.test(a) && /^\d+$/.test(b)) {
5543             return parseInt(a) - parseInt(b);
5544         }
5545         return a.localeCompare(b);
5546     });
5547
5548     for (const fileNumber of sortedFiles) {
5549         const slots = memorySlots[fileNumber];
5550         const slotCount = Object.keys(slots).length;
5551
5552         // Skip displaying the statements file in memory display - it's shown in the logic terminal
5553         if (fileNumber === logicEngine.statementsFileId) {
5554             continue;
5555         }
5556
5557         html += `<div class="memory-file">`;
5558         html += `

```

```

5559         <div class="memory-file-title">
5560             FILE ${fileNumber} (${slotCount} slots)
5561             <div class="memory-file-actions">
5562                 <button class="file-action" onclick="exportFile('${fileNumber}')">Export</button>
5563                 <button class="file-action" onclick="saveFileAsJson('${fileNumber}')">Save JSON</button>
5564             </div>
5565         </div>
5566     `;
5567
5568     // Sort slot numbers numerically if possible
5569     const sortedSlots = Object.keys(slots).sort((a, b) => {
5570         if (/^\d+$/.test(a) && /^\d+$/.test(b)) {
5571             return parseInt(a) - parseInt(b);
5572         }
5573         return a.localeCompare(b);
5574     });
5575
5576     // Limit display to max 10 slots per file for readability
5577     const displaySlots = sortedSlots.slice(0, 10);
5578     const remainingSlots = sortedSlots.length - 10;
5579
5580     for (const slotNumber of displaySlots) {
5581         const value = slots[slotNumber];
5582
5583         // Truncate long values for display
5584         let displayValue = value;
5585         if (displayValue.length > 100) {
5586             displayValue = displayValue.substring(0, 97) + '...';
5587         }
5588
5589         html += `<div class="memory-slot">`;
5590         html += `<div class="memory-slot-title">Slot ${slotNumber}</div>`;
5591         html += `<div class="memory-slot-value">${escapeHtml(displayValue)}</div>`;
5592         html += `</div>`;
5593     }
5594
5595     if (remainingSlots > 0) {
5596         html += `<div style="font-style: italic; margin-top: 10px; text-align: center;">
5597             ... and ${remainingSlots} more slots (not displayed)
5598         </div>`;
5599     }
5600
5601     html += `</div>`;
5602 }
5603
5604 if (html === '') {
5605     memoryDisplay.innerHTML = `
5606         <div style="text-align: center; color: #888; margin-top: 20px;">
5607             No memory slots available.
5608             <br><br>
5609             Use the terminal to add memory slots.
5610         </div>
5611     `;
5612 } else {
5613     memoryDisplay.innerHTML = html;
5614 }
5615 }
5616
5617 // Export a file
5618 function exportFile(fileNumber) {
5619     const result = memoryManager.exportToTextFile(fileNumber);
5620     appendToTerminal(result, 'system');
5621 }
5622
5623 // Save a file as JSON
5624 function saveFileAsJson(fileNumber) {
5625     const result = memoryManager.saveToJSON(fileNumber);
5626     appendToTerminal(result, 'system');
5627 }
5628
5629 // Save all memory slots to a ZIP file
5630 async function saveAllMemorySlots() {
5631     await loadJSZip();

```



```

5632     try {
5633         const result = await memoryManager.saveAllAsZip();
5634         appendToTerminal(result, 'system');
5635     } catch (error) {
5636         appendToTerminal(error, 'system');
5637     }
5638 }
5639
5640 // Load memory slots from a file
5641 async function loadMemorySlotsFromFile() {
5642     await loadJSZip();
5643
5644     // Create a file input element
5645     const fileInput = document.createElement('input');
5646     fileInput.type = 'file';
5647     fileInput.accept = '.zip,.json';
5648     fileInput.style.display = 'none';
5649     document.body.appendChild(fileInput);
5650
5651     fileInput.onchange = async function(e) {
5652         const file = e.target.files[0];
5653         if (!file) {
5654             appendToTerminal('No file selected.', 'system');
5655             return;
5656         }
5657
5658         try {
5659             let result;
5660
5661             if (file.name.endsWith('.zip')) {
5662                 result = await memoryManager.loadFromZip(file);
5663             } else if (file.name.endsWith('.json')) {
5664                 const reader = new FileReader();
5665                 const content = await new Promise((resolve, reject) => {
5666                     reader.onload = e => resolve(e.target.result);
5667                     reader.onerror = () => reject('Error reading file');
5668                     reader.readAsText(file);
5669                 });
5670
5671                 result = memoryManager.loadJSON(content);
5672             } else {
5673                 throw new Error('Unsupported file format');
5674             }
5675
5676             appendToTerminal(result, 'system');
5677             updateMemoryDisplay();
5678         } catch (error) {
5679             appendToTerminal(`Error: ${error}`, 'system');
5680         }
5681     };
5682
5683     fileInput.click();
5684     document.body.removeChild(fileInput);
5685 }
5686
5687 // Clear all memory slots
5688 function clearAllMemorySlots() {
5689     if (confirm('Are you sure you want to clear all memory slots? This cannot be undone.')) {
5690         const result = memoryManager.clearAll();
5691         appendToTerminal(result, 'system');
5692         updateMemoryDisplay();
5693     }
5694 }
5695
5696 // Insert command to input field
5697 function insertCommand(command) {
5698     commandInput.value = command;
5699     commandInput.focus();
5700 }
5701
5702 // Insert logic command to logic input field
5703 function insertLogicCommand(command) {
5704     logicCommandInput.value = command;

```

```

5705         logicCommandInput.focus();
5706     }
5707
5708     // Make utility functions global for access from HTML
5709     window.exportFile = exportFile;
5710     window.saveFileAsJson = saveFileAsJson;
5711     window.saveAllMemorySlots = saveAllMemorySlots;
5712     window.loadMemorySlotsFromFile = loadMemorySlotsFromFile;
5713     window.clearAllMemorySlots = clearAllMemorySlots;
5714     window.clearTerminal = clearTerminal;
5715     window.clearLogicTerminal = clearLogicTerminal;
5716     window.updateMemoryDisplay = updateMemoryDisplay;
5717     window.insertCommand = insertCommand;
5718     window.insertLogicCommand = insertLogicCommand;
5719 </script>
5720 </body>
5721 </html>
5722
5723 //map.js
5724 document.addEventListener('DOMContentLoaded', function() {
5725     const charsetConfig = [
5726         { name: 'Basic Latin', range: [0x0020, 0x007F] },
5727         { name: 'Latin-1 Supplement', range: [0x0080, 0x00FF] },
5728         { name: 'Latin Extended-A', range: [0x0100, 0x017F] },
5729         { name: 'Latin Extended-B', range: [0x0180, 0x024F] },
5730         { name: 'Greek and Coptic', range: [0x0370, 0x03FF] },
5731         { name: 'Cyrillic', range: [0x0400, 0x04FF] },
5732         { name: 'Arabic', range: [0x0600, 0x06FF] },
5733         { name: 'Hebrew', range: [0x0590, 0x05FF] },
5734         { name: 'Devanagari', range: [0x0900, 0x097F] },
5735         { name: 'Mathematical Operators', range: [0x2200, 0x22FF] },
5736         { name: 'Supplemental Mathematical Operators', range: [0x2A00, 0x2AFF] },
5737         { name: 'Miscellaneous Technical', range: [0x2300, 0x23FF] },
5738         { name: 'Miscellaneous Symbols and Arrows', range: [0x2190, 0x21FF] },
5739         { name: 'CJK Unified Ideographs', range: [0x4E00, 0x9FFF] },
5740         { name: 'Hangul Syllables', range: [0xAC00, 0xD7AF] },
5741         { name: 'Hiragana', range: [0x3040, 0x309F] },
5742         { name: 'Katakana', range: [0x30A0, 0x30FF] },
5743         { name: 'Bopomofo', range: [0x3100, 0x312F] },
5744         { name: 'Currency Symbols', range: [0x20A0, 0x20CF] }, // Currency symbols like €, £, ¥, etc.
5745         { name: 'Additional Punctuation', range: [0x2000, 0x206F] },
5746         // Additional symbols and characters can be added similarly.
5747     ];
5748
5749
5750     function isPerfectSquare(n) {
5751         if (n <= 0) return false; // Grids must have a positive number of tiles
5752         const sqrt = Math.sqrt(n);
5753         return sqrt === Math.floor(sqrt);
5754     }
5755
5756
5757     function areValidColorIndexes(indexes) {
5758         const maxColorIndex = Math.pow(16, 6); // Maximum valid color index is 16777216
5759         return indexes.every(index => index >= 1 && index <= maxColorIndex);
5760     }
5761
5762
5763
5764     function updateColorBar(indexes) {
5765         const colorBar = document.getElementById('color-bar');
5766         if (isPerfectSquare(indexes.length) && areValidColorIndexes(indexes)) {
5767             colorBar.style.backgroundColor = 'green'; // Valid grid
5768         } else {
5769             colorBar.style.backgroundColor = 'red'; // Invalid grid
5770         }
5771     }
5772
5773
5774
5775
5776     function decodeIDtoColorIndexes(idString) {
5777         const indexes = [];

```

```

5778     const segmentLength = 7; // Each color index is represented by a 7-digit segment
5779
5780     // Split the integer string into 7-digit segments
5781     for (let i = 0; i < idString.length; i += segmentLength) {
5782         const segment = idString.slice(i, i + segmentLength);
5783         const index = parseInt(segment, 10);
5784
5785         if (!isNaN(index)) {
5786             indexes.push(index);
5787         }
5788     }
5789     return indexes;
5790 }
5791
5792
5793 function generateCharsetFromConfig(config) {
5794     const charset = new Set();
5795     config.forEach(block => {
5796         for (let i = block.range[0]; i <= block.range[1]; i++) {
5797             try {
5798                 charset.add(String.fromCharCode(i));
5799             } catch (e) {
5800                 console.error(`Failed to add character code ${i}: ${e.message}`);
5801             }
5802         }
5803     });
5804     // Explicitly add newline character
5805     charset.add('\n');
5806     charset.add('\t');
5807     return Array.from(charset);
5808 }
5809
5810 const uniqueCharset = generateCharsetFromConfig(charsetConfig);
5811 console.log(uniqueCharset);
5812
5813 document.getElementById('main-menu').style.display = 'block';
5814
5815 function showOption(optionId) {
5816     const options = document.querySelectorAll('#option-container > div');
5817     options.forEach(opt => opt.style.display = 'none');
5818     document.getElementById(optionId).style.display = 'block';
5819     document.getElementById('option-container').style.display = 'block';
5820 }
5821
5822 document.getElementById('show-generate-combinations').addEventListener('click', () => showOption('generate-combinations'));
5823 document.getElementById('show-calculate-string-id').addEventListener('click', () => showOption('calculate-string-id'));
5824 document.getElementById('show-decode-id').addEventListener('click', () => showOption('decode-id'));
5825 document.getElementById('show-find-optimal-variable').addEventListener('click', () => showOption('find-optimal-variable'));
5826
5827 document.getElementById('generate-combinations-btn').addEventListener('click', generateCombinations);
5828 document.getElementById('calculate-string-id-btn').addEventListener('click', calculateStringID);
5829 document.getElementById('decode-id-btn').addEventListener('click', decodeID);
5830 document.getElementById('find-optimal-variable-btn').addEventListener('click', findOptimalVariable);
5831
5832 function generateCombinations() {
5833     try {
5834         const n = BigInt(document.getElementById('combination-size').value);
5835         const outputFile = document.getElementById('output-file-generate').value;
5836         if (isNaN(Number(n)) || n <= 0n) {
5837             throw new Error("Invalid combination size. Please enter a positive integer.");
5838         }
5839
5840         let combinations = '';
5841         const k = BigInt(uniqueCharset.length);
5842         const nbrComb = k ** n;
5843         for (let i = 0n; i < nbrComb; i++) {
5844             let id = i;
5845             let combination = '';
5846             for (let j = 0n; j < n; j++) {
5847                 combination = uniqueCharset[Number(id % k)] + combination;
5848                 id = id / k;
5849             }
5850             combinations += combination + '\n';

```

```

5851     }
5852     alert("Combinations generated and saved to " + outputFile);
5853     console.log(combinations);
5854     downloadFile(outputFile, combinations);
5855 } catch (error) {
5856     alert(`Error: ${error.message}`);
5857 }
5858 }
5859
5860 function displayCharacterAndWordCount(inputText) {
5861     const characterCount = inputText.length;
5862     const wordCount = inputText.trim() === '' ? 0 : inputText.trim().split(/\s+/).length;
5863
5864     document.getElementById('character-word-count').innerText =
5865         `Characters: ${characterCount}, Words: ${wordCount}`;
5866 }
5867
5868
5869 function calculateStringID() {
5870     try {
5871         const inputString = document.getElementById('custom-string').value;
5872         if (inputString.trim() === '') {
5873             throw new Error("Input string cannot be empty.");
5874         }
5875
5876         let id = 0n;
5877         for (const char of inputString) {
5878             const charIndex = uniqueCharset.indexOf(char);
5879             if (charIndex === -1) {
5880                 throw new Error(`Character "${char}" is not in the charset.`);
5881             }
5882             id = id * BigInt(uniqueCharset.length) + BigInt(charIndex);
5883         }
5884
5885         const result = id.toString() + "\t\n\n" + inputString;
5886         document.getElementById('string-id-result').innerText = "The ID for the string is: " + id.toString();
5887
5888         // Call the function to display character and word count
5889         displayCharacterAndWordCount(inputString);
5890
5891         // Decode ID to color indexes using the new method
5892         const colorIndexes = decodeIDtoColorIndexes(id.toString());
5893
5894         // Validate and update the color bar for any grid size
5895         updateColorBar(colorIndexes);
5896
5897         const autoDownload = confirm("Do you want to automatically download the ID?");
5898         if (autoDownload) {
5899             const outputFile = `${colorIndexes.length}.txt`;
5900             downloadFile(outputFile, result);
5901         }
5902     } catch (error) {
5903         alert(`Error: ${error.message}`);
5904     }
5905 }
5906
5907
5908
5909
5910
5911 function decodeID() {
5912     try {
5913         let id = BigInt(document.getElementById('string-id').value);
5914         if (id < 0n) {
5915             throw new Error("ID cannot be negative.");
5916         }
5917
5918         const decodedString = [];
5919         while (id > 0n) {
5920             decodedString.push(uniqueCharset[Number(id % BigInt(uniqueCharset.length))]);
5921             id = id / BigInt(uniqueCharset.length);
5922         }
5923

```

```

5924         if (decodedString.length === 0) {
5925             throw new Error("Decoded string is empty.");
5926         }
5927
5928         document.getElementById('decoded-string-result').innerText = "The decoded string is: " + decodedString.reverse().join('');
5929     } catch (error) {
5930         alert(`Error: ${error.message}`);
5931     }
5932 }
5933
5934 function optimalVariableGenerator(userNumber, charsetLength, extendedCharsetLength) {
5935     let k = 1n;
5936     return {
5937         next: function() {
5938             while (true) {
5939                 const x = k * userNumber;
5940                 if (x % extendedCharsetLength < charsetLength) {
5941                     k += 1n;
5942                     return { value: x, done: false };
5943                 }
5944                 k += 1n;
5945             }
5946         }
5947     };
5948 }
5949
5950 function findOptimalVariable() {
5951     try {
5952         const userNumber = BigInt(document.getElementById('user-number').value);
5953         const outputFile = document.getElementById('output-file-optimal').value;
5954         const charsetLength = BigInt(uniqueCharset.length);
5955         const extendedCharsetLength = BigInt(uniqueCharset.length);
5956         const generator = optimalVariableGenerator(userNumber, charsetLength, extendedCharsetLength);
5957         const autoDownload = confirm("Do you want to automatically download the generated file?");
5958
5959         const results = [];
5960         for (let i = 0; i < 100; i++) {
5961             const optVar = generator.next().value;
5962             const decodedString = decodeIDFromNumber(optVar, uniqueCharset);
5963             results.push(optVar + '\t' + decodedString);
5964         }
5965
5966         const resultString = results.join('\n');
5967
5968         if (autoDownload) {
5969             downloadFile(outputFile, resultString);
5970         } else {
5971             alert("Optimal variables and their corresponding strings saved to " + outputFile);
5972             console.log(resultString);
5973         }
5974     } catch (error) {
5975         alert(`Error: ${error.message}`);
5976     }
5977 }
5978
5979 function decodeIDFromNumber(id, charset) {
5980     const decodedString = [];
5981     while (id > 0n) {
5982         decodedString.push(charset[Number(id % BigInt(charset.length))]);
5983         id = id / BigInt(charset.length);
5984     }
5985     return decodedString.reverse().join('');
5986 }
5987
5988 // Helper Function: Display Character and Word Count
5989 function displayCharacterAndWordCount(inputText) {
5990     const characterCount = inputText.length;
5991     const wordCount = inputText.trim() === '' ? 0 : inputText.trim().split(/\s+/).length;
5992
5993     const charWordCountElement = document.getElementById('character-word-count');
5994     if (charWordCountElement) {
5995         charWordCountElement.innerText = `Characters: ${characterCount}, Words: ${wordCount}`;
5996     } else {

```

```

5997         console.error("Element with ID 'character-word-count' not found.");
5998     }
5999 }
6000
6001 // Event Listener for Dynamic Character and Word Count
6002 const customStringTextarea = document.getElementById('custom-string');
6003 if (customStringTextarea) {
6004     customStringTextarea.addEventListener('input', function(event) {
6005         const inputString = event.target.value;
6006         displayCharacterAndWordCount(inputString);
6007     });
6008 } else {
6009     console.error("Textarea with ID 'custom-string' not found.");
6010 }
6011
6012 function downloadFile(filename, content) {
6013     const blob = new Blob([content], { type: 'text/plain' });
6014     const url = URL.createObjectURL(blob);
6015     const a = document.createElement('a');
6016     a.href = url;
6017     a.download = filename;
6018     document.body.appendChild(a);
6019     a.click();
6020     document.body.removeChild(a);
6021     URL.revokeObjectURL(url);
6022 }
6023 });
6024
6025 //storm-new.js
6026 // Global variables
6027 let quadtreeRoot = null;
6028 let currentDepth = 0;
6029 let maxDepth = 4; // reintroduce as editable dimension
6030 let actionHistory = [];
6031 let redoStack = [];
6032 let currentCell = null; // For pasting images
6033
6034 // Default size settings for the root quadtree cell
6035 // In the old system, we had rows/cols and cell sizes. Here, let's define a single large square:
6036 let rootCellSize = 600; // can be adjusted by user
6037
6038 const quadtreeContainer = document.getElementById('quadtreeContainer');
6039
6040 // QuadtreeNode class with expression support
6041 class QuadtreeNode {
6042     constructor(x, y, size, depth) {
6043         this.x = x;
6044         this.y = y;
6045         this.size = size;
6046         this.depth = depth;
6047         this.children = [];
6048         this.data = {};
6049         this.element = this.createElement();
6050         this.updateVisualState();
6051     }
6052
6053     createElement() {
6054         const cell = document.createElement('div');
6055         cell.className = 'quadtree-cell';
6056
6057         cell.style.left = `${this.x}px`;
6058         cell.style.top = `${this.y}px`;
6059         cell.style.width = `${this.size}px`;
6060         cell.style.height = `${this.size}px`;
6061
6062         const content = document.createElement('div');
6063         content.className = 'cell-content';
6064         this.updateContent(content);
6065         cell.appendChild(content);
6066
6067         cell.quadtreeNode = this;
6068
6069         cell.addEventListener('click', (e) => {

```

```

6070         e.stopPropagation();
6071         onCellClick(e, cell);
6072     });
6073
6074     cell.addEventListener('contextmenu', (e) => {
6075         e.preventDefault();
6076         onCellRightClick(e, cell);
6077     });
6078
6079     quadtreeContainer.appendChild(cell);
6080     return cell;
6081 }
6082
6083 updateVisualState() {
6084     if (this.depth < maxDepth) {
6085         this.element.classList.add('can-subdivide');
6086         this.element.title = 'Click to subdivide or right-click for options';
6087     } else {
6088         this.element.classList.remove('can-subdivide');
6089         this.element.title = 'Max depth reached. Right-click for properties.';
6090     }
6091 }
6092
6093 // Evaluate expression if any and update displayed content
6094 updateContent(content) {
6095     const cellData = this.data;
6096     let displayText = cellData.text || `Depth: ${this.depth}`;
6097     if (cellData.expression) {
6098         try {
6099             const result = math.evaluate(cellData.expression);
6100             displayText = result.toString();
6101         } catch (e) {
6102             displayText = 'Error';
6103         }
6104     }
6105     content.textContent = displayText;
6106
6107     if (cellData.bgColor) this.element.style.backgroundColor = cellData.bgColor;
6108     if (cellData.fontSize) content.style.fontSize = `${cellData.fontSize}px`;
6109     if (cellData.fontColor) content.style.color = cellData.fontColor;
6110
6111     // If image is present
6112     if (cellData.imageSrc) {
6113         content.innerHTML = '';
6114         const img = document.createElement('img');
6115         img.src = cellData.imageSrc;
6116         img.style.width = '100%';
6117         img.style.height = '100%';
6118         content.appendChild(img);
6119     }
6120 }
6121
6122 redraw() {
6123     const content = this.element.querySelector('.cell-content');
6124     this.updateContent(content);
6125 }
6126
6127 subdivide() {
6128     if (this.children.length > 0 || this.depth >= maxDepth) return false;
6129
6130     const childSize = this.size / 2;
6131
6132     this.children.push(
6133         new QuadtreeNode(this.x, this.y, childSize, this.depth + 1),
6134         new QuadtreeNode(this.x + childSize, this.y, childSize, this.depth + 1),
6135         new QuadtreeNode(this.x, this.y + childSize, childSize, this.depth + 1),
6136         new QuadtreeNode(this.x + childSize, this.y + childSize, childSize, this.depth + 1)
6137     );
6138
6139     this.element.style.display = 'none';
6140     return true;
6141 }
6142

```

```

6143     merge() {
6144         // Remove children if present and restore this cell
6145         if (this.children.length > 0) {
6146             this.children.forEach(child => {
6147                 if (child.element.parentNode) {
6148                     quadtreeContainer.removeChild(child.element);
6149                 }
6150             });
6151             this.children = [];
6152             this.element.style.display = 'block';
6153         }
6154     }
6155 }
6156
6157 function initQuadtree() {
6158     quadtreeContainer.innerHTML = '';
6159     currentDepth = 0;
6160     actionHistory = [];
6161     redoStack = [];
6162
6163     quadtreeRoot = new QuadtreeNode(0, 0, rootCellSize, 0);
6164 }
6165
6166 // Handle window resizing by resetting the quadtree
6167 window.addEventListener('resize', () => {
6168     initQuadtree();
6169 });
6170
6171 function zoomIn() {
6172     if (currentDepth < maxDepth) {
6173         currentDepth++;
6174         const leafNodes = [];
6175         traverseQuadtree(quadtreeRoot, (node) => {
6176             if (node.depth === currentDepth - 1 && node.children.length === 0) {
6177                 leafNodes.push(node);
6178             }
6179         });
6180
6181         leafNodes.forEach(node => {
6182             node.subdivide();
6183             node.children.forEach(child => {
6184                 child.element.style.opacity = '0';
6185                 child.element.style.transform = 'scale(0.9)';
6186                 requestAnimationFrame(() => {
6187                     child.element.style.transition = 'opacity 0.3s, transform 0.3s';
6188                     child.element.style.opacity = '1';
6189                     child.element.style.transform = 'scale(1)';
6190                 });
6191             });
6192         });
6193     }
6194 }
6195
6196 function zoomOut() {
6197     if (currentDepth > 0) {
6198         const parentNodes = [];
6199         traverseQuadtree(quadtreeRoot, (node) => {
6200             if (node.depth === currentDepth - 1 && node.children.length > 0) {
6201                 parentNodes.push(node);
6202             }
6203         });
6204
6205         parentNodes.forEach(node => {
6206             node.children.forEach(child => {
6207                 child.element.style.transition = 'opacity 0.3s, transform 0.3s';
6208                 child.element.style.opacity = '0';
6209                 child.element.style.transform = 'scale(0.9)';
6210             });
6211
6212             setTimeout(() => {
6213                 node.merge();
6214                 node.element.style.opacity = '0';
6215                 node.element.style.transform = 'scale(0.9)';

```



```

6216         requestAnimationFrame(() => {
6217             node.element.style.transition = 'opacity 0.3s, transform 0.3s';
6218             node.element.style.opacity = '1';
6219             node.element.style.transform = 'scale(1)';
6220         });
6221     }, 300);
6222 });
6223
6224     currentDepth--;
6225 }
6226 }
6227
6228 function downloadQuadtreeAsPNG() {
6229     const canvas = document.createElement('canvas');
6230     const context = canvas.getContext('2d');
6231
6232     const quadtreeRect = quadtreeContainer.getBoundingClientRect();
6233     canvas.width = quadtreeRect.width;
6234     canvas.height = quadtreeRect.height;
6235
6236     context.fillStyle = 'white';
6237     context.fillRect(0, 0, canvas.width, canvas.height);
6238
6239     async function drawNode(node) {
6240         if (!node || !node.element) return;
6241
6242         const styles = window.getComputedStyle(node.element);
6243         context.fillStyle = styles.backgroundColor;
6244         context.fillRect(node.x, node.y, node.size, node.size);
6245
6246         context.strokeStyle = styles.borderColor || '#000';
6247         context.lineWidth = parseInt(styles.borderWidth) || 1;
6248         context.strokeRect(node.x, node.y, node.size, node.size);
6249
6250         const content = node.element.querySelector('.cell-content');
6251         if (content) {
6252             const contentStyles = window.getComputedStyle(content);
6253             const img = content.querySelector('img');
6254             if (img) {
6255                 await new Promise((resolve, reject) => {
6256                     const image = new Image();
6257                     image.crossOrigin = 'Anonymous';
6258                     image.onload = () => {
6259                         context.drawImage(image, node.x, node.y, node.size, node.size);
6260                         resolve();
6261                     };
6262                     image.onerror = reject;
6263                     image.src = img.src;
6264                 });
6265             } else {
6266                 const text = content.textContent;
6267                 if (text) {
6268                     context.fillStyle = contentStyles.color;
6269                     context.font = `${contentStyles.fontSize} ${contentStyles.fontFamily}`;
6270                     context.textAlign = 'center';
6271                     context.textBaseline = 'middle';
6272                     context.fillText(
6273                         text,
6274                         node.x + (node.size / 2),
6275                         node.y + (node.size / 2)
6276                     );
6277                 }
6278             }
6279         }
6280     }
6281
6282     async function drawAllNodes(node) {
6283         if (!node) return;
6284         if (node.element.style.display !== 'none') {
6285             await drawNode(node);
6286         }
6287         for (const child of node.children) {
6288             await drawAllNodes(child);

```

```

6289     }
6290 }
6291
6292 drawAllNodes(quadtreeRoot).then(() => {
6293     const link = document.createElement('a');
6294     link.download = 'quadtree.png';
6295     link.href = canvas.toDataURL('image/png');
6296     link.click();
6297 });
6298 }
6299
6300 function resetQuadtree() {
6301     initQuadtree();
6302 }
6303
6304 function traverseQuadtree(node, callback) {
6305     if (!node) return;
6306     callback(node);
6307     if (node.children && node.children.length > 0) {
6308         node.children.forEach(child => traverseQuadtree(child, callback));
6309     }
6310 }
6311
6312 // Cell click: subdivide if possible, else edit properties
6313 function onCellClick(event, cell) {
6314     const node = cell.quadtreeNode;
6315     if (node.depth < maxDepth && node.children.length === 0) {
6316         saveActionState(cell);
6317         if (node.subdivide()) {
6318             node.children.forEach(child => {
6319                 child.element.style.opacity = '0';
6320                 child.element.style.transform = 'scale(0.9)';
6321                 requestAnimationFrame(() => {
6322                     child.element.style.transition = 'opacity 0.3s, transform 0.3s';
6323                     child.element.style.opacity = '1';
6324                     child.element.style.transform = 'scale(1)';
6325                 });
6326             });
6327         }
6328     } else {
6329         openPropertiesModal(cell);
6330     }
6331 }
6332
6333 function onCellRightClick(event, cell) {
6334     openCellContextMenu(event, cell);
6335 }
6336
6337 // Properties modal for editing cell data (reintroducing expression, colors)
6338 function openPropertiesModal(cell) {
6339     const node = cell.quadtreeNode;
6340     const cellData = node.data;
6341
6342     const modal = createModal();
6343     const content = modal.querySelector('.modal-content');
6344
6345     content.innerHTML = `
6346         <h2>Edit Cell Properties</h2>
6347         <label>Text:</label>
6348         <input type="text" id="cellText" value="${cellData.text || ''}">
6349         <label>Expression:</label>
6350         <input type="text" id="cellExpression" value="${cellData.expression || ''}">
6351         <label>Background Color:</label>
6352         <input type="color" id="cellBgColor" value="${cellData.bgColor || '#F5F5DC'}">
6353         <label>Font Size:</label>
6354         <input type="number" id="cellFontSize" value="${cellData.fontSize || 14}">
6355         <label>Font Color:</label>
6356         <input type="color" id="cellFontColor" value="${cellData.fontColor || '#333333'}">
6357         <button id="applyCellProperties">Apply</button>
6358     `;
6359
6360     document.getElementById('applyCellProperties').addEventListener('click', () => {
6361         applyCellProperties(cell, modal);

```

```

6362     });
6363 }
6364
6365 function applyCellProperties(cell, modal) {
6366     saveActionState(cell);
6367
6368     const node = cell.quadtreeNode;
6369     const cellData = node.data;
6370
6371     cellData.text = document.getElementById('cellText').value;
6372     cellData.expression = document.getElementById('cellExpression').value;
6373     cellData.bgColor = document.getElementById('cellBgColor').value;
6374     cellData.fontSize = parseInt(document.getElementById('cellFontSize').value);
6375     cellData.fontColor = document.getElementById('cellFontColor').value;
6376
6377     node.redraw();
6378     document.body.removeChild(modal);
6379 }
6380
6381 // Create modal
6382 function createModal() {
6383     const modal = document.createElement('div');
6384     modal.className = 'modal';
6385     const modalContent = document.createElement('div');
6386     modalContent.className = 'modal-content';
6387     modal.appendChild(modalContent);
6388     document.body.appendChild(modal);
6389     return modal;
6390 }
6391
6392 function saveActionState(cell) {
6393     const node = cell.quadtreeNode;
6394     const cellDataCopy = JSON.parse(JSON.stringify(node.data));
6395     actionHistory.push({ cell, cellData: cellDataCopy, childrenData: node.children.map(c => ({...c.data})) });
6396     redoStack = [];
6397 }
6398
6399 // Undo action
6400 function undoAction() {
6401     if (actionHistory.length > 0) {
6402         const { cell, cellData, childrenData } = actionHistory.pop();
6403         const node = cell.quadtreeNode;
6404         redoStack.push({ cell, cellData: JSON.parse(JSON.stringify(node.data)), childrenData: node.children.map(c => ({...c.data})) });
6405
6406         // Restore
6407         node.data = JSON.parse(JSON.stringify(cellData));
6408         // If children existed before, we'd need a more complex restore, but for simplicity we restore only data.
6409         node.redraw();
6410     }
6411 }
6412
6413 // Redo action
6414 function redoAction() {
6415     if (redoStack.length > 0) {
6416         const { cell, cellData, childrenData } = redoStack.pop();
6417         const node = cell.quadtreeNode;
6418         actionHistory.push({ cell, cellData: JSON.parse(JSON.stringify(node.data)), childrenData: node.children.map(c => ({...c.data})) });
6419
6420         node.data = JSON.parse(JSON.stringify(cellData));
6421         node.redraw();
6422     }
6423 }
6424
6425 function openCellContextMenu(event, cell) {
6426     const node = cell.quadtreeNode;
6427     const menu = document.createElement('div');
6428     menu.className = 'context-menu';
6429     menu.style.top = `${event.pageY}px`;
6430     menu.style.left = `${event.pageX}px`;
6431
6432     const uploadImageOption = document.createElement('div');
6433     uploadImageOption.textContent = 'Upload Image';
6434     uploadImageOption.addEventListener('click', () => {

```

```

6435         openImageUploadDialog(cell);
6436         document.body.removeChild(menu);
6437     });
6438
6439     const pasteImageOption = document.createElement('div');
6440     pasteImageOption.textContent = 'Paste Image';
6441     pasteImageOption.addEventListener('click', async () => {
6442         currentCell = cell;
6443         document.body.removeChild(menu);
6444         try {
6445             const clipboardItems = await navigator.clipboard.read();
6446             for (const clipboardItem of clipboardItems) {
6447                 for (const type of clipboardItem.types) {
6448                     if (type.startsWith('image/')) {
6449                         const blob = await clipboardItem.getType(type);
6450                         const reader = new FileReader();
6451                         reader.onload = function(e) {
6452                             pasteImageIntoCell(currentCell, e.target.result);
6453                             currentCell = null;
6454                         };
6455                         reader.readAsDataURL(blob);
6456                         return;
6457                     }
6458                 }
6459             }
6460             alert('No image found on the clipboard!');
6461         } catch (err) {
6462             console.error('Failed to read clipboard contents: ', err);
6463             alert('Could not access clipboard contents. Please try again.');
```

```

6508         };
6509         reader.readAsDataURL(blob);
6510         break;
6511     }
6512 }
6513 });
6514
6515 function pasteImageIntoCell(cell, imageData) {
6516     saveActionState(cell);
6517     const node = cell.quadtreeNode;
6518     node.data.imageSrc = imageData;
6519     node.redraw();
6520 }
6521
6522 function openImageUploadDialog(cell) {
6523     const input = document.createElement('input');
6524     input.type = 'file';
6525     input.accept = 'image/*';
6526     input.onChange = event => {
6527         const file = event.target.files[0];
6528         if (file) {
6529             saveActionState(cell);
6530             const reader = new FileReader();
6531             reader.onload = e => {
6532                 const node = cell.quadtreeNode;
6533                 node.data.imageSrc = e.target.result;
6534                 node.redraw();
6535             };
6536             reader.readAsDataURL(file);
6537         }
6538     };
6539     input.click();
6540 }
6541
6542 // Edit quadtree dimensions modal: allows editing rootCellSize and maxDepth
6543 function openQuadtreeDimensionModal() {
6544     const modal = createModal();
6545     const content = modal.querySelector('.modal-content');
6546
6547     content.innerHTML = `
6548 <h2>Edit Quadtree Dimensions</h2>
6549 <label>Root Cell Size (px):</label>
6550 <input type="number" id="rootCellSizeInput" value="${rootCellSize}" min="100">
6551 <label>Max Depth:</label>
6552 <input type="number" id="maxDepthInput" value="${maxDepth}" min="1">
6553 <button id="applyDimensions">Apply</button>
6554 `;
6555
6556     document.getElementById('applyDimensions').addEventListener('click', () => {
6557         const newSize = parseInt(document.getElementById('rootCellSizeInput').value);
6558         const newMaxDepth = parseInt(document.getElementById('maxDepthInput').value);
6559         applyQuadtreeDimensions(newSize, newMaxDepth);
6560         document.body.removeChild(modal);
6561     });
6562 }
6563
6564 function applyQuadtreeDimensions(size, depth) {
6565     rootCellSize = size;
6566     maxDepth = depth;
6567     initQuadtree();
6568 }
6569
6570 // Toolbar actions
6571 document.getElementById('quadtreeActions').addEventListener('change', (e) => {
6572     const action = e.target.value;
6573     if (action === 'newQuadtree') initQuadtree();
6574     else if (action === 'editDimensions') openQuadtreeDimensionModal();
6575     else if (action === 'zoomIn') zoomIn();
6576     else if (action === 'zoomOut') zoomOut();
6577     else if (action === 'resetQuadtree') resetQuadtree();
6578     else if (action === 'undo') undoAction();
6579     else if (action === 'redo') redoAction();
6580     else if (action === 'downloadPNG') downloadQuadtreeAsPNG();

```

```
6581     e.target.value = '';
6582 });
6583
6584 // Initialize on load
6585 window.onload = initQuadtree;
6586
6587 //alphabet.html
6588 <!DOCTYPE html>
6589 <html lang="en">
6590 <head>
6591     <!-- Meta Tags -->
6592     <meta charset="UTF-8">
6593     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6594     <title>Tile Color Viewer</title>
6595     <!-- CSS Stylesheet -->
6596     <link rel="stylesheet" href="alphabet.css">
6597     <script src="https://cdnjs.cloudflare.com/ajax/libs/jsczip/3.10.1/jsczip.min.js"></script>
6598 </head>
6599 <body>
6600     <!-- Hero Section -->
6601     <div class="hero">
6602         <h1>Tile Color Viewer</h1>
6603         <p>Create and customize colorful grids with ease.</p>
6604     </div>
6605
6606     <!-- Controls Section -->
6607     <div id="controls">
6608         <label for="tile-color">Tile Color: </label>
6609         <input type="color" id="tile-color" value="#ff0000">
6610
6611         <label for="tile-size">Tile Size (px): </label>
6612         <input type="number" id="tile-size" value="100" placeholder="Tile size in pixels">
6613
6614         <button id="update-grid">Update Grid</button>
6615     </div>
6616
6617     <!-- Grid Display Section -->
6618     <canvas id="grid-canvas" style="border: 1px solid #ccc;"></canvas>
6619
6620     <!-- Navigation Controls -->
6621     <div id="navigation-controls">
6622         <button id="prev-grid" aria-label="Go to previous grid">Previous Grid</button>
6623         <button id="next-grid" aria-label="Go to next grid">Next Grid</button>
6624         <button id="download-grid-image" aria-label="Download current grid as image">Download Grid Image</button>
6625     </div>
6626
6627     <!-- Current Grid Information -->
6628     <div id="current-grid-id"></div>
6629     <div id="current-grid-position"></div>
6630
6631     <!-- Color List Container -->
6632     <div id="color-list-container">
6633         <h2>Sequence List</h2>
6634         <ul id="color-list"></ul>
6635         <button id="download-colors">Download Tile ID</button>
6636     </div>
6637
6638     <!-- Compounded ID Input Container -->
6639     <div id="compounded-id-input-container">
6640         <h2>Generate Grids by Compounded IDs</h2>
6641         <div id="compounded-id-inputs">
6642             <div class="compounded-id-input-row">
6643                 <input type="text" class="compounded-id-input" placeholder="Enter Compounded Grid ID">
6644             </div>
6645         </div>
6646         <button id="add-compounded-id">+</button>
6647         <button id="generate-grid-by-id">Generate Grids</button>
6648         <br>
6649         <!-- File Upload Input -->
6650         <p>Upload a file containing compounded grid IDs (e.g., sequences.txt).</p>
6651         <input type="file" id="file-input" accept=".txt">
6652
6653         <!-- Execute Button -->
```

```
6654         <button id="execute-button" aria-label="Process uploaded file">Execute</button>
6655     </div>
6656
6657     <!-- Footer Section -->
6658     <footer>
6659         <p>&copy; 2024 Tile Color Viewer. All rights reserved.</p>
6660     </footer>
6661
6662     <!-- JavaScript File -->
6663     <script src="alphabet.js"></script>
6664 </body>
6665 </html>
6666
6667 //storm-new.css
6668 /* Styles for Quadtree System */
6669 #quadtreeContainer {
6670     position: relative;
6671     width: 100%;
6672     height: 100vh;
6673     background-color: #FFFFFF; /* White */
6674     overflow: hidden;
6675     border: 2px solid #BDB76B; /* Dark Khaki */
6676 }
6677
6678 .quadtree-cell {
6679     position: absolute;
6680     border: 1px solid #BDB76B; /* Dark Khaki */
6681     background-color: #F5F5DC; /* Beige */
6682     display: flex;
6683     justify-content: center;
6684     align-items: center;
6685     font-family: "Fira Code", "Consolas", monospace;
6686     color: #333333; /* Dark gray */
6687     text-align: center;
6688     cursor: pointer;
6689     transition: transform 0.2s ease;
6690 }
6691
6692 .quadtree-cell:hover {
6693     transform: scale(1.05);
6694     box-shadow: 0px 4px 8px rgba(0, 0, 0, 0.2);
6695 }
6696
6697 /* Cell Content */
6698 .cell-content {
6699     width: 100%;
6700     height: 100%;
6701     display: flex;
6702     justify-content: center;
6703     align-items: center;
6704     overflow: hidden;
6705 }
6706
6707 /* Dropdown for Quadtree Actions */
6708 #toolbar {
6709     margin: 10px;
6710 }
6711
6712 #toolbar select {
6713     background-color: #4B5320; /* Army Green */
6714     color: #FFFFFF; /* White */
6715     border: 2px solid #BDB76B; /* Dark Khaki */
6716     padding: 10px;
6717     font-size: 14px;
6718     font-family: "Fira Code", "Consolas", monospace;
6719     cursor: pointer;
6720     transition: background-color 0.3s, color 0.3s;
6721 }
6722
6723 #toolbar select:hover {
6724     background-color: #BDB76B; /* Dark Khaki */
6725     color: #333333; /* Dark gray */
6726 }
```

```
6727
6728 /* Modal Styles */
6729 .modal {
6730     position: fixed;
6731     top: 0;
6732     left: 0;
6733     width: 100%;
6734     height: 100%;
6735     background-color: rgba(0,0,0,0.5);
6736     display: flex;
6737     justify-content: center;
6738     align-items: center;
6739 }
6740
6741 .modal-content {
6742     position: relative;
6743     background-color: #fff;
6744     padding: 20px;
6745     width: 320px;
6746     border: 1px solid #888;
6747     font-family: "Fira Code", "Consolas", monospace;
6748 }
6749
6750 .modal-content h2 {
6751     margin-top: 0;
6752 }
6753
6754 .modal-content label {
6755     display: block;
6756     margin-top: 10px;
6757 }
6758
6759 .modal-content input[type="text"],
6760 .modal-content input[type="number"],
6761 .modal-content input[type="color"] {
6762     width: 100%;
6763     padding: 5px;
6764     margin-top: 5px;
6765     box-sizing: border-box;
6766 }
6767
6768 #quadtreeContainer {
6769     position: relative;
6770     width: min(100vh, 100vw); /* Make it square */
6771     height: min(100vh, 100vw); /* Make it square */
6772     margin: 0 auto;
6773     background-color: #FFFFFF;
6774     overflow: hidden;
6775     border: 2px solid #BDB76B;
6776 }
6777
6778 .modal-content button {
6779     margin-top: 15px;
6780     padding: 10px 20px;
6781     background-color: #4B5320; /* Army Green */
6782     color: #FFFFFF; /* White */
6783     border: none;
6784     cursor: pointer;
6785 }
6786
6787 .modal-content button:hover {
6788     background-color: #BDB76B; /* Dark Khaki */
6789     color: #333333; /* Dark gray */
6790 }
6791
6792 /* Context Menu Styles */
6793 .context-menu {
6794     position: absolute;
6795     background-color: #fff;
6796     border: 1px solid #888;
6797     z-index: 1000;
6798     font-family: "Fira Code", "Consolas", monospace;
6799     min-width: 120px;
```



```
6800 }
6801
6802 .context-menu div {
6803     padding: 8px 12px;
6804     cursor: pointer;
6805 }
6806
6807 .context-menu div:hover {
6808     background-color: #f1f1f1;
6809 }
6810
6811 .quadtree-cell {
6812     position: absolute;
6813     border: 1px solid #BDB76B;
6814     background-color: #F5F5DC;
6815     display: flex;
6816     justify-content: center;
6817     align-items: center;
6818     font-family: "Fira Code", "Consolas", monospace;
6819     color: #333333;
6820     text-align: center;
6821     cursor: pointer;
6822     transition: transform 0.2s ease, opacity 0.3s ease, background-color 0.3s;
6823 }
6824
6825 .quadtree-cell.can-subdivide::before {
6826     content: '+';
6827     position: absolute;
6828     top: 5px;
6829     right: 5px;
6830     font-size: 12px;
6831     opacity: 0;
6832     transition: opacity 0.2s;
6833 }
6834
6835 .quadtree-cell.can-subdivide:hover::before {
6836     opacity: 0.7;
6837 }
6838
6839 .quadtree-cell.can-subdivide:hover {
6840     background-color: #f0f0e0;
6841     transform: scale(1.02);
6842 }
6843
6844 .cell-content {
6845     font-size: 12px;
6846     pointer-events: none;
6847 }
6848
6849 //storm-new.html
6850 <!DOCTYPE html>
6851 <html lang="en">
6852 <head>
6853     <meta charset="UTF-8">
6854     <title>Mathematical Quadtree Sandbox</title>
6855     <link rel="stylesheet" href="storm-new.css">
6856     <link rel="stylesheet" href="storm-new.css">
6857     <!-- math.js for expressions -->
6858     <script src="https://cdnjs.cloudflare.com/ajax/libs/mathjs/11.5.0/math.min.js"></script>
6859 </head>
6860 <body>
6861
6862 <div id="toolbar">
6863     <select id="quadtreeActions">
6864         <option value="">Choose Action</option>
6865         <option value="newQuadtree">New Quadtree</option>
6866         <option value="editDimensions">Edit Quadtree Dimensions</option>
6867         <option value="zoomIn">Zoom In</option>
6868         <option value="zoomOut">Zoom Out</option>
6869         <option value="resetQuadtree">Reset Quadtree</option>
6870         <option value="undo">Undo</option>
6871         <option value="redo">Redo</option>
6872         <option value="downloadPNG">Download as PNG</option>
```

```
6873     </select>
6874 </div>
6875
6876 <div id="quadtreeContainer"></div>
6877
6878 <!-- Modal placeholder -->
6879 <div id="modalContainer"></div>
6880
6881 <!-- Script -->
6882 <script src="storm-new.js"></script>
6883
6884 </body>
6885 </html>
6886
6887 //map.css
6888
6889 /* Basic Reset */
6890 * {
6891     margin: 0;
6892     padding: 0;
6893     box-sizing: border-box;
6894 }
6895
6896 /* Body Styling */
6897 body {
6898     font-family: "Fira Code", "Consolas", monospace;
6899     background: linear-gradient(135deg, #4B5320, #BDB76B); /* Army Green to Dark Khaki */
6900     color: #333333; /* Dark Gray */
6901     display: flex;
6902     flex-direction: column;
6903     align-items: center;
6904     justify-content: center;
6905     min-height: 100vh;
6906     padding: 20px;
6907 }
6908
6909 /* Heading Styles */
6910 h1 {
6911     color: #FFFFFF; /* White */
6912     font-size: 2.5em;
6913     text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.4);
6914     margin-bottom: 20px;
6915     text-transform: uppercase;
6916     letter-spacing: 1px;
6917 }
6918
6919 h2, h3 {
6920     color: #FFFFFF;
6921     margin-bottom: 15px;
6922     text-shadow: 1px 1px 3px rgba(0, 0, 0, 0.3);
6923     font-weight: 600;
6924 }
6925
6926 /* Button Styles */
6927 button {
6928     background-color: #4B5320; /* Army Green */
6929     color: #FFFFFF;
6930     padding: 10px 20px;
6931     font-size: 16px;
6932     border: 2px solid #BDB76B; /* Dark Khaki */
6933     cursor: pointer;
6934     transition: background-color 0.3s ease, color 0.3s ease;
6935     font-family: "Fira Code", "Consolas", monospace;
6936     margin-top: 10px;
6937     /* Removed border-radius for square corners */
6938 }
6939
6940 button:hover {
6941     background-color: #BDB76B; /* Dark Khaki */
6942     color: #333333; /* Dark Gray */
6943 }
6944
6945 /* Input and Textarea Styles */
```

```
6946 input[type="text"], input[type="number"], textarea, select {
6947     width: calc(100% - 20px);
6948     padding: 10px;
6949     font-size: 16px;
6950     background-color: #F5F5DC; /* Beige */
6951     color: #333333; /* Dark Gray */
6952     border: 2px solid #BDB76B; /* Dark Khaki */
6953     font-family: "Fira Code", "Consolas", monospace;
6954     margin-bottom: 10px;
6955     /* Removed border-radius for square corners */
6956 }
6957
6958 textarea {
6959     resize: none; /* Disables resizing of the textarea */
6960 }
6961
6962 /* Display Containers */
6963 #option-container > div {
6964     background: rgba(245, 245, 220, 0.9); /* Beige with transparency */
6965     padding: 20px;
6966     border: 2px solid #BDB76B; /* Dark Khaki */
6967     box-shadow: 0 2px 4px rgba(0, 0, 0, 0.4);
6968     width: 100%;
6969     max-width: 600px;
6970     margin-bottom: 20px;
6971 }
6972
6973 /* Results Styling */
6974 #string-id-result, #decoded-string-result {
6975     background-color: #FFFFFF; /* White */
6976     padding: 10px;
6977     border: 2px solid #BDB76B; /* Dark Khaki */
6978     font-family: "Fira Code", "Consolas", monospace;
6979     box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
6980     color: #333333; /* Dark Gray */
6981     word-wrap: break-word;
6982     white-space: pre-wrap; /* Preserves spaces and line breaks */
6983 }
6984
6985 /* Styling for the color bar */
6986 #color-bar {
6987     width: 100%; /* Full width */
6988     height: 20px; /* Fixed height */
6989     background-color: #BDB76B; /* Dark Khaki */
6990     margin-top: 10px; /* Space above the bar */
6991     border: 2px solid #4B5320; /* Army Green border */
6992     box-shadow: 0 2px 5px rgba(0, 0, 0, 0.3); /* Subtle shadow */
6993     transition: background-color 0.3s ease; /* Smooth transition for color change */
6994 }
6995
6996 //map.html
6997 <!DOCTYPE html>
6998 <html lang="en">
6999 <head>
7000     <meta charset="UTF-8">
7001     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7002     <title>Charset App</title>
7003     <link rel="stylesheet" href="map.css">
7004 </head>
7005 <body>
7006     <h1>Welcome to the Charset App (Purposed for: Spatial Systems Engineering)</h1>
7007     <div id="main-menu">
7008         <h2>Choose an option:</h2>
7009         <button id="show-generate-combinations">Generate Combinations</button>
7010         <button id="show-calculate-string-id">Calculate String ID</button>
7011         <button id="show-decode-id">Decode ID to String</button>
7012         <button id="show-find-optimal-variable">Find Optimal Variable and Save to File</button>
7013     </div>
7014     <!-- Add this inside the <body> tag, below the main-menu div -->
7015     <div id="color-bar" style="width: 100%; height: 20px; background-color: gold; margin-top: 10px;"></div>
7016
7017     <br><br>
7018     <div id="option-container" style="display: none;">
```

```
7019 <!-- Mode 1: Generate Combinations -->
7020 <div id="generate-combinations" style="display: none;">
7021   <h3>Generate Combinations</h3>
7022   <label for="combination-size">Enter the size of combinations (n):</label>
7023   <input type="number" id="combination-size">
7024   <br>
7025   <label for="output-file-generate">Enter the name of the output file:</label>
7026   <input type="text" id="output-file-generate">
7027   <br>
7028   <label for="use-multithreading">Use Multithreading:</label>
7029   <select id="use-multithreading">
7030     <option value="yes">Yes</option>
7031     <option value="no">No</option>
7032   </select>
7033   <br>
7034   <button id="generate-combinations-btn">Generate</button>
7035 </div>
7036 <!-- Mode 2: Calculate String ID -->
7037
7038 <div id="calculate-string-id" style="display: none;">
7039   <h3>Calculate String ID</h3>
7040   <label for="custom-string">Enter your custom string:</label>
7041   <textarea id="custom-string" rows="10" cols="50" wrap="off"></textarea>
7042   <br>
7043   <button id="calculate-string-id-btn">Calculate</button>
7044   <p id="string-id-result"></p>
7045   <div id="character-word-count">Characters: 0, Words: 0</div>
7046 </div>
7047
7048 <!-- Mode 3: Decode ID to String -->
7049 <div id="decode-id" style="display: none;">
7050   <h3>Decode ID to String</h3>
7051   <label for="string-id">Enter the ID to decode:</label>
7052   <input type="number" id="string-id">
7053   <br>
7054   <button id="decode-id-btn">Decode</button>
7055   <p id="decoded-string-result"></p>
7056 </div>
7057 <!-- Mode 4: Find Optimal Variable -->
7058 <div id="find-optimal-variable" style="display: none;">
7059   <h3>Find Optimal Variable and Save to File</h3>
7060   <label for="user-number">Enter the user number:</label>
7061   <input type="number" id="user-number">
7062   <br>
7063   <label for="output-file-optimal">Enter the name of the output file:</label>
7064   <input type="text" id="output-file-optimal">
7065   <br>
7066   <button id="find-optimal-variable-btn">Find and Save</button>
7067 </div>
7068 </div>
7069 <script src="map.js"></script>
7070 </body>
7071 </html>
7072
7073 //alphabet.css
7074 /* HDMCSS - Applied to Your CSS Code */
7075
7076 /* CSS Variables for Theming */
7077 :root {
7078   --primary-color: #4B5320; /* Army Green */
7079   --secondary-color: #FFFFFF; /* White */
7080   --accent-color: #BDB76B; /* Dark Khaki */
7081   --accent-hover-color: #D2B48C; /* Tan */
7082   --background-color: #F5F5DC; /* Beige */
7083   --text-color: #333333; /* Dark Gray */
7084 }
7085
7086 /* General Reset */
7087 * {
7088   margin: 0;
7089   padding: 0;
7090   box-sizing: border-box;
7091 }
```

```
7092
7093 /* Body Styling */
7094 body {
7095     font-family: "Fira Code", "Consolas", monospace;
7096     background-color: var(--background-color);
7097     color: var(--text-color);
7098     display: flex;
7099     flex-direction: column;
7100     align-items: center;
7101     min-height: 100vh;
7102     padding: 20px;
7103     line-height: 1.6;
7104 }
7105
7106 /* Hero Section */
7107 .hero {
7108     width: 100%;
7109     max-width: 1200px;
7110     text-align: center;
7111     padding: 50px 20px;
7112     background-color: var(--primary-color);
7113     color: var(--secondary-color);
7114     margin-bottom: 40px;
7115 }
7116
7117 .hero h1 {
7118     font-size: 3rem;
7119     margin-bottom: 20px;
7120     font-weight: 600;
7121     text-transform: uppercase;
7122     letter-spacing: 1px;
7123 }
7124
7125 .hero p {
7126     font-size: 1.2rem;
7127 }
7128
7129 /* Controls Container */
7130 #controls,
7131 #compounded-id-input-container,
7132 #color-list-container {
7133     background: var(--secondary-color);
7134     padding: 20px;
7135     border: 2px solid var(--accent-color);
7136     /* Removed border-radius to square off corners */
7137     box-shadow: 0 2px 4px rgba(0, 0, 0, 0.4);
7138     max-width: 800px;
7139     width: 100%;
7140     margin-bottom: 30px;
7141 }
7142
7143 #controls {
7144     display: flex;
7145     flex-wrap: wrap;
7146     justify-content: space-between;
7147     align-items: center;
7148     gap: 10px;
7149 }
7150
7151 #controls label {
7152     flex: 1 1 200px;
7153     font-size: 1rem;
7154     color: var(--primary-color);
7155     font-weight: 500;
7156 }
7157
7158 #controls input {
7159     flex: 1 1 200px;
7160     padding: 8px 12px;
7161     border: 2px solid var(--accent-color);
7162     /* Removed border-radius to square off corners */
7163     font-size: 1rem;
7164     color: var(--text-color);
```

```
7165     background-color: #F5F5DC; /* Beige */
7166     font-family: "Fira Code", "Consolas", monospace;
7167 }
7168
7169 #controls button {
7170     flex: 1 1 100%;
7171     padding: 12px 25px;
7172     font-size: 1rem;
7173     color: var(--secondary-color);
7174     background-color: var(--primary-color);
7175     border: 2px solid var(--accent-color);
7176     /* Removed border-radius to square off corners */
7177     cursor: pointer;
7178     transition: background-color 0.3s ease, transform 0.2s ease, color 0.3s ease;
7179     margin-top: 10px;
7180     font-family: "Fira Code", "Consolas", monospace;
7181 }
7182
7183 #controls button:hover {
7184     background-color: var(--accent-color);
7185     color: var(--text-color);
7186     transform: translateY(-2px);
7187 }
7188
7189 /* Grid Canvas */
7190 #grid-canvas {
7191     border: 2px solid var(--accent-color);
7192     margin-bottom: 30px;
7193 }
7194
7195 /* Navigation Controls */
7196 #navigation-controls {
7197     display: flex;
7198     justify-content: center;
7199     align-items: center;
7200     gap: 20px;
7201     max-width: 800px;
7202     width: 100%;
7203     margin-bottom: 30px;
7204 }
7205
7206 #navigation-controls button {
7207     padding: 12px 25px;
7208     font-size: 1rem;
7209     color: var(--secondary-color);
7210     background-color: var(--primary-color);
7211     border: 2px solid var(--accent-color);
7212     /* Removed border-radius to square off corners */
7213     cursor: pointer;
7214     font-weight: bold;
7215     transition: background-color 0.3s ease, transform 0.2s ease, color 0.3s ease;
7216     font-family: "Fira Code", "Consolas", monospace;
7217 }
7218
7219 #navigation-controls button:hover {
7220     background-color: var(--accent-color);
7221     color: var(--text-color);
7222     transform: translateY(-2px);
7223 }
7224
7225 #navigation-controls button:disabled {
7226     background-color: #ccc;
7227     border: 2px solid #999;
7228     color: #666;
7229     cursor: not-allowed;
7230     transform: none;
7231 }
7232
7233 /* Current Grid Information */
7234 #current-grid-id,
7235 #current-grid-position {
7236     font-size: 1rem;
7237     color: var(--primary-color);
```

```
7238     margin-bottom: 10px;
7239     text-align: center;
7240     font-weight: 500;
7241     font-family: "Fira Code", "Consolas", monospace;
7242 }
7243
7244 /* Color List Container */
7245 #color-list-container h2 {
7246     font-size: 1.5rem;
7247     margin-bottom: 10px;
7248     color: var(--primary-color);
7249     font-weight: 600;
7250     text-transform: uppercase;
7251     letter-spacing: 1px;
7252 }
7253
7254 #color-list {
7255     list-style-type: none;
7256     padding: 0;
7257     margin: 0;
7258 }
7259
7260 #color-list li {
7261     background: #F5F5DC; /* Beige */
7262     padding: 10px;
7263     margin-bottom: 5px;
7264     border: 2px solid var(--accent-color);
7265     /* Removed border-radius to square off corners */
7266     font-size: 1rem;
7267     color: var(--text-color);
7268     font-family: "Fira Code", "Consolas", monospace;
7269 }
7270
7271 /* Download Colors Button */
7272 #download-colors {
7273     margin-top: 10px;
7274     padding: 12px 25px;
7275     font-size: 1rem;
7276     color: var(--secondary-color);
7277     background-color: var(--primary-color);
7278     border: 2px solid var(--accent-color);
7279     /* Removed border-radius to square off corners */
7280     cursor: pointer;
7281     transition: background-color 0.3s ease, transform 0.2s ease, color 0.3s ease;
7282     font-family: "Fira Code", "Consolas", monospace;
7283 }
7284
7285 #download-colors:hover {
7286     background-color: var(--accent-color);
7287     color: var(--text-color);
7288     transform: translateY(-2px);
7289 }
7290
7291 /* Compounded ID Input Container */
7292 #compounded-id-input-container {
7293     margin-bottom: 30px;
7294 }
7295
7296 #compounded-id-inputs {
7297     display: flex;
7298     flex-direction: column;
7299     gap: 10px;
7300     margin-bottom: 10px;
7301 }
7302
7303 .compounded-id-input-row {
7304     display: flex;
7305     align-items: center;
7306 }
7307
7308 .compounded-id-input {
7309     flex: 1;
7310     padding: 8px 12px;
```

```
7311     border: 2px solid var(--accent-color);
7312     /* Removed border-radius to square off corners */
7313     font-size: 1rem;
7314     color: var(--text-color);
7315     background-color: #F5F5DC; /* Beige */
7316     font-family: "Fira Code", "Consolas", monospace;
7317 }
7318
7319 /* Add Compounded ID Button */
7320 #add-compounded-id {
7321     padding: 10px;
7322     font-size: 1.5rem;
7323     color: var(--secondary-color);
7324     background-color: var(--primary-color);
7325     border: 2px solid var(--accent-color);
7326     /* Removed border-radius to square off corners */
7327     cursor: pointer;
7328     transition: background-color 0.3s ease, transform 0.2s ease, color 0.3s ease;
7329     margin-bottom: 10px;
7330     width: 40px;
7331     height: 40px;
7332     line-height: 20px;
7333     font-family: "Fira Code", "Consolas", monospace;
7334 }
7335
7336 #add-compounded-id:hover {
7337     background-color: var(--accent-color);
7338     color: var(--text-color);
7339     transform: translateY(-2px);
7340 }
7341
7342 /* Generate Grids Button */
7343 #generate-grids-by-ids {
7344     padding: 12px 25px;
7345     font-size: 1rem;
7346     color: var(--secondary-color);
7347     background-color: var(--primary-color);
7348     border: 2px solid var(--accent-color);
7349     /* Removed border-radius to square off corners */
7350     cursor: pointer;
7351     transition: background-color 0.3s ease, transform 0.2s ease, color 0.3s ease;
7352     font-family: "Fira Code", "Consolas", monospace;
7353 }
7354
7355 #generate-grids-by-ids:hover {
7356     background-color: var(--accent-color);
7357     color: var(--text-color);
7358     transform: translateY(-2px);
7359 }
7360
7361 /* Footer Styling */
7362 footer {
7363     width: 100%;
7364     text-align: center;
7365     padding: 20px;
7366     background-color: var(--primary-color);
7367     color: var(--secondary-color);
7368     margin-top: auto;
7369     font-family: "Fira Code", "Consolas", monospace;
7370     font-size: 1em;
7371 }
7372
7373 /* Responsive Styling */
7374 @media (max-width: 600px) {
7375     #controls,
7376     #compounded-id-input-container,
7377     #navigation-controls {
7378         flex-direction: column;
7379         align-items: stretch;
7380     }
7381
7382     #controls label,
7383     #controls input,
```



```
7384     #controls button {
7385         flex: 1 1 100%;
7386         margin-bottom: 10px;
7387     }
7388
7389     #controls label,
7390     #controls input {
7391         margin-bottom: 5px;
7392     }
7393
7394     #navigation-controls button {
7395         width: 100%;
7396     }
7397
7398     #grid-canvas {
7399         width: 100%;
7400         height: auto;
7401     }
7402 }
7403
7404 }[]
7405
7406
```