9. Practical Examples with Dynamic Memory

9.1 Dynamic Array with User Input

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

int main() {
    int n, *arr;

    printf("How many numbers do you want to enter? ");
    scanf("%d", &n);

    // Allocate memory
    arr = (int*)malloc(n * sizeof(int));
    if (arr == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }

    // Input
    printf("Enter %d numbers:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    // Find min and max
    int min = arr[0], max = arr[0];
    for (int i = 1; i < n; i++) {
        if (arr[i] < min) min = arr[i];
        if (arr[i] > max) max = arr[i];
    }
```

```
    printf("Minimum: %d\n", min);
    printf("Maximum: %d\n", max);

    free(arr);
    return 0;
}
```

9.2 Growing Array (Dynamic Resize)

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

int main() {
    int capacity = 2;
    int size = 0;
    int *arr;
    int input;

    // Initial allocation
    arr = (int*)malloc(capacity * sizeof(int));
    if (arr == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }

    printf("Enter integers (enter -1 to stop):\n");

    while (1) {
        scanf("%d", &input);
        if (input == -1) break;

        // Check if we need more space
        if (size >= capacity) {
            capacity *= 2;  // Double the capacity
            arr = (int*)realloc(arr, capacity * sizeof(int));
            if (arr == NULL) {
                printf("Memory reallocation failed!\n");
                return 1;
            }
```

```
            printf("Array resized to capacity: %d\n", capacity);
        }

        arr[size++] = input;
    }

    // Print results
    printf("\nYou entered %d numbers:\n", size);
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    free(arr);
    return 0;
}
```

9.3 Dynamic String Operations

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

char* concatenateStrings(const char *s1, const char *s2) {
    int len1 = strlen(s1);
    int len2 = strlen(s2);

    // Allocate memory for result
    char *result = (char*)malloc((len1 + len2 + 1) * sizeof(char));

    if (result == NULL) {
        return NULL;
    }

    // Copy first string
    strcpy(result, s1);
    // Concatenate second string
    strcat(result, s2);
```

```
    return result;
}

int main() {
    char str1[] = "Hello, ";
    char str2[] = "World!";

    char *combined = concatenateStrings(str1, str2);

    if (combined != NULL) {
        printf("Result: %s\n", combined);
        free(combined);
    }

    return 0;
}
```

9.4 Remove Duplicates from Dynamic Array

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

int* removeDuplicates(int *arr, int size, int *newSize) {
    // Allocate worst-case size (all unique)
    int *result = (int*)malloc(size * sizeof(int));
    if (result == NULL) {
        return NULL;
    }

    int count = 0;

    for (int i = 0; i < size; i++) {
        int isDuplicate = 0;

        // Check if current element already exists in result
        for (int j = 0; j < count; j++) {
            if (arr[i] == result[j]) {
                isDuplicate = 1;
                break;
```

```
            }
        }

        if (!isDuplicate) {
            result[count++] = arr[i];
        }
    }

    // Resize to actual size needed
    result = (int*)realloc(result, count * sizeof(int));
    *newSize = count;

    return result;
}

int main() {
    int arr[] = {1, 2, 3, 2, 4, 1, 5, 3, 6};
    int size = sizeof(arr) / sizeof(arr[0]);
    int newSize;

    printf("Original array: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    int *unique = removeDuplicates(arr, size, &newSize);

    if (unique != NULL) {
        printf("Array without duplicates: ");
        for (int i = 0; i < newSize; i++) {
            printf("%d ", unique[i]);
        }
        printf("\n");

        free(unique);
    }

    return 0;
}
```

9.5 Dynamic Matrix Operations

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

// Allocate matrix
int** createMatrix(int rows, int cols) {
    int **matrix = (int**)malloc(rows * sizeof(int*));
    if (matrix == NULL) return NULL;

    for (int i = 0; i < rows; i++) {
        matrix[i] = (int*)malloc(cols * sizeof(int));
        if (matrix[i] == NULL) {
            // Free already allocated rows
            for (int j = 0; j < i; j++) {
                free(matrix[j]);
            }
            free(matrix);
            return NULL;
        }
    }

    return matrix;
}

// Free matrix
void freeMatrix(int **matrix, int rows) {
    for (int i = 0; i < rows; i++) {
        free(matrix[i]);
    }
    free(matrix);
}

// Multiply two matrices
int** multiplyMatrices(int **A, int **B, int r1, int c1, int c2) {
    int **result = createMatrix(r1, c2);
    if (result == NULL) return NULL;

    for (int i = 0; i < r1; i++) {
        for (int j = 0; j < c2; j++) {
```

```
            result[i][j] = 0;
            for (int k = 0; k < c1; k++) {
                result[i][j] += A[i][k] * B[k][j];
            }
        }
    }

    return result;
}

int main() {
    int r1 = 2, c1 = 3, c2 = 2;

    // Create matrices
    int **A = createMatrix(r1, c1);
    int **B = createMatrix(c1, c2);

    // Fill matrix A
    printf("Matrix A:\n");
    int val = 1;
    for (int i = 0; i < r1; i++) {
        for (int j = 0; j < c1; j++) {
            A[i][j] = val++;
            printf("%d ", A[i][j]);
        }
        printf("\n");
    }

    // Fill matrix B
    printf("\nMatrix B:\n");
    val = 1;
    for (int i = 0; i < c1; i++) {
        for (int j = 0; j < c2; j++) {
            B[i][j] = val++;
            printf("%d ", B[i][j]);
        }
        printf("\n");
    }

    // Multiply
    int **C = multiplyMatrices(A, B, r1, c1, c2);
```

```
    printf("\nMatrix C (A × B):\n");
    for (int i = 0; i < r1; i++) {
        for (int j = 0; j < c2; j++) {
            printf("%d ", C[i][j]);
        }
        printf("\n");
    }

    // Free all matrices
    freeMatrix(A, r1);
    freeMatrix(B, c1);
    freeMatrix(C, r1);

    return 0;
}
```

Revision #1
Created 2025-10-01 03:30:34 UTC by DS
Updated 2025-10-01 03:40:06 UTC by DS