# 8. Practical Applications

## 8.1 Polynomial Addition

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

// Node for polynomial term
struct PolyNode {
    int coeff;  // Coefficient
    int exp;    // Exponent
    struct PolyNode *next;
};

// Insert term in descending order of exponent
void insertTerm(struct PolyNode **poly, int coeff, int exp) {
    struct PolyNode *newNode = (struct PolyNode*)malloc(sizeof(struct PolyNode));
    newNode->coeff = coeff;
    newNode->exp = exp;
    newNode->next = NULL;

    if (*poly == NULL || (*poly)->exp < exp) {
        newNode->next = *poly;
        *poly = newNode;
    } else {
        struct PolyNode *temp = *poly;
        while (temp->next != NULL && temp->next->exp > exp) {
            temp = temp->next;
        }
        newNode->next = temp->next;
        temp->next = newNode;
    }
}

// Add two polynomials
struct PolyNode* addPolynomials(struct PolyNode *poly1, struct PolyNode *poly2) {
```

```
    struct PolyNode *result = NULL;

    while (poly1 != NULL && poly2 != NULL) {
        if (poly1->exp > poly2->exp) {
            insertTerm(&result, poly1->coeff, poly1->exp);
            poly1 = poly1->next;
        } else if (poly1->exp < poly2->exp) {
            insertTerm(&result, poly2->coeff, poly2->exp);
            poly2 = poly2->next;
        } else {
            // Same exponent - add coefficients
            int sumCoeff = poly1->coeff + poly2->coeff;
            if (sumCoeff != 0) {
                insertTerm(&result, sumCoeff, poly1->exp);
            }
            poly1 = poly1->next;
            poly2 = poly2->next;
        }
    }

    // Add remaining terms
    while (poly1 != NULL) {
        insertTerm(&result, poly1->coeff, poly1->exp);
        poly1 = poly1->next;
    }

    while (poly2 != NULL) {
        insertTerm(&result, poly2->coeff, poly2->exp);
        poly2 = poly2->next;
    }

    return result;
}

// Display polynomial
void displayPoly(struct PolyNode *poly) {
    if (poly == NULL) {
        printf("0\n");
        return;
    }
```

```
    while (poly != NULL) {
        printf("%dx^%d", poly->coeff, poly->exp);
        poly = poly->next;
        if (poly != NULL) {
            printf(" + ");
        }
    }
    printf("\n");
}

int main() {
    struct PolyNode *poly1 = NULL;
    struct PolyNode *poly2 = NULL;

    // Create first polynomial: 5x^3 + 4x^2 + 2
    insertTerm(&poly1, 5, 3);
    insertTerm(&poly1, 4, 2);
    insertTerm(&poly1, 2, 0);

    // Create second polynomial: 3x^3 + 2x + 1
    insertTerm(&poly2, 3, 3);
    insertTerm(&poly2, 2, 1);
    insertTerm(&poly2, 1, 0);

    printf("Polynomial 1: ");
    displayPoly(poly1);

    printf("Polynomial 2: ");
    displayPoly(poly2);

    struct PolyNode *result = addPolynomials(poly1, poly2);

    printf("Sum: ");
    displayPoly(result);

    return 0;
}

/* Output:
```

```
Polynomial 1: 5x^3 + 4x^2 + 2x^0
Polynomial 2: 3x^3 + 2x^1 + 1x^0
Sum: 8x^3 + 4x^2 + 2x^1 + 3x^0
*/
```

# 8.2 Music Playlist Implementation

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

struct Song {
    char title[100];
    char artist[100];
    int duration;  // in seconds
    struct Song *next;
};

// Add song to playlist
void addSong(struct Song **playlist, const char *title, const char *artist, int duration) {
    struct Song *newSong = (struct Song*)malloc(sizeof(struct Song));
    strcpy(newSong->title, title);
    strcpy(newSong->artist, artist);
    newSong->duration = duration;
    newSong->next = NULL;

    if (*playlist == NULL) {
        *playlist = newSong;
    } else {
        struct Song *temp = *playlist;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newSong;
    }
}

// Display playlist
void displayPlaylist(struct Song *playlist) {
```

```
    if (playlist == NULL) {
        printf("Playlist is empty!\n");
        return;
    }

    int count = 1;
    printf("\n=== Playlist ===\n");
    while (playlist != NULL) {
        printf("%d. %s - %s (%d:%02d)\n",
               count++,
               playlist->title,
               playlist->artist,
               playlist->duration / 60,
               playlist->duration % 60);
        playlist = playlist->next;
    }
}

// Remove song by title
void removeSong(struct Song **playlist, const char *title) {
    if (*playlist == NULL) {
        printf("Playlist is empty!\n");
        return;
    }

    // If first song matches
    if (strcmp((*playlist)->title, title) == 0) {
        struct Song *temp = *playlist;
        *playlist = (*playlist)->next;
        free(temp);
        printf("Song removed: %s\n", title);
        return;
    }

    // Search for song
    struct Song *temp = *playlist;
    while (temp->next != NULL && strcmp(temp->next->title, title) != 0) {
        temp = temp->next;
    }
```

```
    if (temp->next == NULL) {
        printf("Song not found: %s\n", title);
        return;
    }

    struct Song *toRemove = temp->next;
    temp->next = toRemove->next;
    free(toRemove);
    printf("Song removed: %s\n", title);
}

// Calculate total duration
int totalDuration(struct Song *playlist) {
    int total = 0;
    while (playlist != NULL) {
        total += playlist->duration;
        playlist = playlist->next;
    }
    return total;
}

int main() {
    struct Song *myPlaylist = NULL;

    // Add songs
    addSong(&myPlaylist, "Bohemian Rhapsody", "Queen", 354);
    addSong(&myPlaylist, "Stairway to Heaven", "Led Zeppelin", 482);
    addSong(&myPlaylist, "Hotel California", "Eagles", 391);
    addSong(&myPlaylist, "Imagine", "John Lennon", 183);

    displayPlaylist(myPlaylist);

    int total = totalDuration(myPlaylist);
    printf("\nTotal duration: %d:%02d\n", total / 60, total % 60);

    // Remove a song
    removeSong(&myPlaylist, "Hotel California");

    displayPlaylist(myPlaylist);
```

```
    return 0;
}
```

# 8.3 Browser History (Back/Forward Navigation)

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

struct Page {
    char url[200];
    struct Page *prev;
    struct Page *next;
};

struct Browser {
    struct Page *current;
};

// Visit new page
void visitPage(struct Browser *browser, const char *url) {
    struct Page *newPage = (struct Page*)malloc(sizeof(struct Page));
    strcpy(newPage->url, url);
    newPage->next = NULL;

    if (browser->current != NULL) {
        // Clear forward history
        struct Page *temp = browser->current->next;
        while (temp != NULL) {
            struct Page *toDelete = temp;
            temp = temp->next;
            free(toDelete);
        }

        browser->current->next = newPage;
        newPage->prev = browser->current;
    } else {
        newPage->prev = NULL;
    }
```

```
    browser->current = newPage;
    printf("Visited: %s\n", url);
}

// Go back
void goBack(struct Browser *browser) {
    if (browser->current == NULL || browser->current->prev == NULL) {
        printf("Cannot go back!\n");
        return;
    }

    browser->current = browser->current->prev;
    printf("Back to: %s\n", browser->current->url);
}

// Go forward
void goForward(struct Browser *browser) {
    if (browser->current == NULL || browser->current->next == NULL) {
        printf("Cannot go forward!\n");
        return;
    }

    browser->current = browser->current->next;
    printf("Forward to: %s\n", browser->current->url);
}

// Display current page
void displayCurrent(struct Browser *browser) {
    if (browser->current == NULL) {
        printf("No page loaded!\n");
    } else {
        printf("Current page: %s\n", browser->current->url);
    }
}

int main() {
    struct Browser browser = {NULL};

    visitPage(&browser, "https://google.com");
    visitPage(&browser, "https://github.com");
    visitPage(&browser, "https://stackoverflow.com");
```

```
    printf("\n");
    goBack(&browser);
    goBack(&browser);

    printf("\n");
    goForward(&browser);

    printf("\n");
    visitPage(&browser, "https://reddit.com");

    printf("\n");
    displayCurrent(&browser);

    return 0;
}
```

---

Revision #1
Created 2025-10-27 05:06:41 UTC by DS
Updated 2025-10-27 05:07:15 UTC by DS