W języku C można zdefiniować tablicę wielowymiarową w sposób podobny do ogólnej składni przedstawionej wcześniej. Poniżej znajduje się przykład tworzenia tablicy dwuwymiarowej o rozmiarze 3x3 w języku C:

```c

#include <stdio.h>

int main() {

int tablica[3][3] = {

{1, 2, 3},

{4, 5, 6},

{7, 8, 9}

};

// Dostęp do elementów tablicy

printf("%d\n", tablica[0][0]); // Wypisze 1

printf("%d\n", tablica[1][2]); // Wypisze 6

printf("%d\n", tablica[2][1]); // Wypisze 8

return 0;

}

```

W powyższym przykładzie zadeklarowano tablicę o nazwie "tablica" typu int, o wymiarach 3x3. Następnie zainicjalizowano jej elementy za pomocą listy inicjalizacyjnej. Przy użyciu indeksów można odczytywać lub modyfikować wartości poszczególnych elementów tablicy.

Warto zauważyć, że w języku C indeksowanie tablic zaczyna się od 0, więc pierwszy element tablicy ma indeks 0, a ostatni element tablicy o rozmiarze n ma indeks n-1.

W języku C struktury pozwalają definiować własne typy danych, które mogą zawierać różne pola różnych typów. Struktury są przydatne do grupowania powiązanych danych w jeden obiekt. Poniżej znajduje się przykład definicji i użycia struktury w języku C:

```c

#include <stdio.h>

// Definicja struktury

struct Osoba {

char imie[20];

char nazwisko[20];

int wiek;

};

int main() {

// Deklaracja i inicjalizacja struktury

struct Osoba osoba1 = {"Jan", "Kowalski", 30};

// Dostęp do pól struktury

printf("Imię: %s\n", osoba1.imie);

printf("Nazwisko: %s\n", osoba1.nazwisko);

printf("Wiek: %d\n", osoba1.wiek);

return 0;

}

```

W powyższym przykładzie zdefiniowano strukturę o nazwie "Osoba", która zawiera trzy pola: "imie" i "nazwisko" (tablice znaków) oraz "wiek" (liczba całkowita). Następnie zadeklarowano i zainicjalizowano zmienną "osoba1" typu "Osoba" i przypisano jej konkretne wartości dla pól. Można uzyskać dostęp do pól struktury, używając operatora kropki ".".

Struktury w języku C mogą być również wykorzystywane do tworzenia tablic struktur, tworzenia wskaźników na struktury, definiowania struktur zagnieżdżonych itp.

Oto kilka przykładowych zadań, wraz z rozwiązaniami opartymi na strukturach w języku C:

Zadanie 1: Stwórz strukturę "Punkt" zawierającą pola "x" i "y" typu int. Napisz funkcję, która przyjmuje dwa punkty jako argumenty i oblicza odległość między nimi.

```c

#include <stdio.h>

#include <math.h>

struct Punkt {

int x;

int y;

};

float obliczOdleglosc(struct Punkt p1, struct Punkt p2) {

int dx = p2.x - p1.x;

int dy = p2.y - p1.y;

return sqrt(dx\*dx + dy\*dy);

}

int main() {

struct Punkt punkt1 = {3, 4};

struct Punkt punkt2 = {6, 8};

float odleglosc = obliczOdleglosc(punkt1, punkt2);

printf("Odleglosc: %.2f\n", odleglosc);

return 0;

}

```

Zadanie 2: Stwórz strukturę "Kwadrat" zawierającą pole "bok" typu int. Napisz funkcję, która przyjmuje kwadrat jako argument i oblicza jego pole i obwód.

```c

#include <stdio.h>

struct Kwadrat {

int bok;

};

void obliczPoleIObwod(struct Kwadrat k) {

int pole = k.bok \* k.bok;

int obwod = 4 \* k.bok;

printf("Pole: %d\n", pole);

printf("Obwod: %d\n", obwod);

}

int main() {

struct Kwadrat kwadrat = {5};

obliczPoleIObwod(kwadrat);

return 0;

}

```

Zadanie 3: Stwórz strukturę "Osoba" zawierającą pola "imie" i "wiek" typu char[]. Napisz program, który pobiera od użytkownika informacje o trzech osobach, przechowuje je w tablicy struktur i następnie wyświetla te informacje.

```c

#include <stdio.h>

struct Osoba {

char imie[50];

int wiek;

};

int main() {

struct Osoba osoby[3];

for (int i = 0; i < 3; i++) {

printf("Osoba %d:\n", i + 1);

printf("Imie: ");

scanf("%s", osoby[i].imie);

printf("Wiek: ");

scanf("%d", &osoby[i].wiek);

}

printf("\n");

for (int i = 0; i < 3; i++) {

printf("Osoba %d:\n", i + 1);

printf("Imie: %s\n", osoby[i].imie);

printf("Wiek: %d\n", osoby[i].wiek);

printf("\n");

}

return 0;

}

```

Mam nadzieję, że powyższe przykłady pomogą Ci zrozumieć, jak można wykorzystywać struktury do rozwiązywania

Oto kilka bardziej skomplikowanych zadań, wraz z rozwiązaniami opartymi na strukturach w języku C:

Zadanie 1: Stwórz strukturę "Student" zawierającą pola "imie" i "oceny" (tablica liczb zmiennoprzecinkowych). Napisz funkcję, która przyjmuje tablicę studentów i oblicza średnią ocen dla każdego z nich.

```c

#include <stdio.h>

#define MAX\_OCENY 5

#define MAX\_STUDENCI 3

struct Student {

char imie[50];

float oceny[MAX\_OCENY];

};

void obliczSrednieOceny(struct Student studenci[], int liczbaStudentow) {

for (int i = 0; i < liczbaStudentow; i++) {

float sumaOcen = 0.0;

for (int j = 0; j < MAX\_OCENY; j++) {

sumaOcen += studenci[i].oceny[j];

}

float sredniaOcen = sumaOcen / MAX\_OCENY;

printf("Student %s: %.2f\n", studenci[i].imie, sredniaOcen);

}

}

int main() {

struct Student studenci[MAX\_STUDENCI] = {

{"Jan", {4.5, 3.5, 4.0, 4.5, 3.0}},

{"Anna", {3.0, 3.5, 4.0, 3.5, 4.0}},

{"Kamil", {4.0, 4.5, 3.5, 4.0, 4.5}}

};

obliczSrednieOceny(studenci, MAX\_STUDENCI);

return 0;

}

```

Zadanie 2: Stwórz strukturę "Książka" zawierającą pola "tytuł" i "autor" typu char[], oraz pole "rokWydania" typu int. Napisz program, który pobiera od użytkownika informacje o n książkach, przechowuje je w tablicy struktur i następnie wyświetla informacje o książkach wydanych po podanym roku.

```c

#include <stdio.h>

#define MAX\_KSIAZKI 5

struct Ksiazka {

char tytul[100];

char autor[100];

int rokWydania;

};

void wyswietlKsiazkiPoRoku(struct Ksiazka ksiazki[], int liczbaKsiazek, int rok) {

printf("Książki wydane po roku %d:\n", rok);

for (int i = 0; i < liczbaKsiazek; i++) {

if (ksiazki[i].rokWydania > rok) {

printf("Tytuł: %s\n", ksiazki[i].tytul);

printf("Autor: %s\n", ksiazki[i].autor);

printf("Rok wydania: %d\n", ksiazki[i].rokWydania);

printf("\n");

}

}

}

int main() {

int liczbaKsiazek;

printf("Podaj liczbę książek:

```c

scanf("%d", &liczbaKsiazek);

struct Ksiazka ksiazki[MAX\_KSIAZKI];

printf("Podaj informacje o książkach:\n");

for (int i = 0; i < liczbaKsiazek; i++) {

printf("Książka %d:\n", i + 1);

printf("Tytuł: ");

scanf("%s", ksiazki[i].tytul);

printf("Autor: ");

scanf("%s", ksiazki[i].autor);

printf("Rok wydania: ");

scanf("%d", &ksiazki[i].rokWydania);

printf("\n");

}

int rok;

printf("Podaj rok: ");

scanf("%d", &rok);

wyswietlKsiazkiPoRoku(ksiazki, liczbaKsiazek, rok);

return 0;

}

```

W powyższym programie użytkownik najpierw podaje liczbę książek, a następnie podaje informacje o każdej książce (tytuł, autor, rok wydania). Po wprowadzeniu wszystkich danych, użytkownik podaje rok, a program wyświetla informacje o książkach wydanych po tym roku.

Mam nadzieję, że powyższe przykłady pomogą Ci w lepszym zrozumieniu zastosowania struktur w bardziej skomplikowanych zadaniach w języku C.

Oto kilka przykładowych zadań, wraz z rozwiązaniami opartymi na tablicach wielowymiarowych w języku C:

Zadanie 1: Stwórz tablicę dwuwymiarową reprezentującą macierz 3x3 liczb całkowitych. Napisz funkcję, która oblicza sumę wszystkich elementów w macierzy.

```c

#include <stdio.h>

#define ROZMIAR 3

int obliczSumeMacierzy(int macierz[][ROZMIAR]) {

int suma = 0;

for (int i = 0; i < ROZMIAR; i++) {

for (int j = 0; j < ROZMIAR; j++) {

suma += macierz[i][j];

}

}

return suma;

}

int main() {

int macierz[ROZMIAR][ROZMIAR] = {

{1, 2, 3},

{4, 5, 6},

{7, 8, 9}

};

int suma = obliczSumeMacierzy(macierz);

printf("Suma elementów macierzy: %d\n", suma);

return 0;

}

```

Zadanie 2: Stwórz tablicę dwuwymiarową reprezentującą planszę do gry w życie (gra komórkowa). Napisz funkcję, która wyświetla planszę na ekranie, przyjmując odpowiednie wartości (np. 1 dla żywej komórki, 0 dla martwej komórki).

```c

#include <stdio.h>

#define ROZMIAR 5

void wyswietlPlansze(int plansza[][ROZMIAR]) {

for (int i = 0; i < ROZMIAR; i++) {

for (int j = 0; j < ROZMIAR; j++) {

printf("%d ", plansza[i][j]);

}

printf("\n");

}

}

int main() {

int plansza[ROZMIAR][ROZMIAR] = {

{1, 0, 0, 1, 0},

{0, 1, 1, 0, 1},

{1, 0, 1, 0, 1},

{0, 1, 0, 1, 0},

{1, 0, 0, 1, 1}

};

wyswietlPlansze(plansza);

return 0;

}

```

Zadanie 3: Stwórz tablicę dwuwymiarową reprezentującą szachownicę o wymiarach 8x8. Napisz program, który wypełnia tablicę wartościami odpowiadającymi szachownicy (np. 0 dla pustego pola, 1 dla białej bierki, 2 dla czarnej bierki) i wyświetla ją na ekranie.

```c

#include <stdio.h>

#define ROZMIAR 8

void wyswietlSzachownice(int szachownica[][ROZMIAR]) {

for (int i = 0; i < ROZMIAR; i++) {

for (int j

```c

for (int j = 0; j < ROZMIAR; j++) {

printf("%d ", szachownica[i][j]);

}

printf("\n");

}

}

int main() {

int szachownica[ROZMIAR][ROZMIAR] = {

{0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0}

};

// Wypełnianie szachownicy

for (int i = 0; i < ROZMIAR; i++) {

for (int j = 0; j < ROZMIAR; j++) {

if ((i + j) % 2 == 0) {

szachownica[i][j] = 1; // Białe pole

} else {

szachownica[i][j] = 2; // Czarne pole

}

}

}

wyswietlSzachownice(szachownica);

return 0;

}

```

W powyższym programie stworzona została tablica dwuwymiarowa `szachownica` o rozmiarze 8x8, reprezentująca szachownicę. Następnie, za pomocą pętli, tablica została wypełniona wartościami odpowiadającymi polom na szachownicy (0 dla pustego pola, 1 dla białej bierki, 2 dla czarnej bierki). Na końcu funkcja `wyswietlSzachownice()` wyświetla tablicę na ekranie.

Mam nadzieję, że powyższe przykłady pomogą Ci lepiej zrozumieć zastosowanie tablic wielowymiarowych w języku C.

Oto kilka bardziej zaawansowanych zadań z rozwiązaniami opartymi na tablicach wielowymiarowych w języku C:

Zadanie 1: Stwórz tablicę dwuwymiarową reprezentującą planszę do gry w statki (grę na polu bitwy). Napisz funkcję, która przyjmuje współrzędne x i y strzału oraz sprawdza, czy strzał trafił w statek (wartość 1 w tablicy). Funkcja powinna zwrócić true, jeśli strzał trafił, lub false w przeciwnym przypadku.

```c

#include <stdio.h>

#include <stdbool.h>

#define ROZMIAR 5

bool czyTrafiony(int plansza[][ROZMIAR], int x, int y) {

if (plansza[x][y] == 1) {

return true;

} else {

return false;

}

}

int main() {

int plansza[ROZMIAR][ROZMIAR] = {

{0, 0, 0, 0, 0},

{0, 1, 0, 0, 0},

{0, 0, 0, 1, 0},

{0, 0, 0, 0, 0},

{1, 0, 0, 0, 0}

};

int x, y;

printf("Podaj współrzędne strzału (x y): ");

scanf("%d %d", &x, &y);

bool trafiony = czyTrafiony(plansza, x, y);

if (trafiony) {

printf("Trafiłeś w statek!\n");

} else {

printf("Nie trafiłeś w statek.\n");

}

return 0;

}

```

Zadanie 2: Stwórz tablicę dwuwymiarową reprezentującą planszę do gry w kółko i krzyżyk. Napisz funkcję, która sprawdza, czy ktoś wygrał grę (czyli czy na planszy znajduje się linia 3 znaków "X" lub "O" w jednej linii, kolumnie lub przekątnej). Funkcja powinna zwracać znak gracza, który wygrał, lub znak pustej planszy w przypadku remisu.

```c

#include <stdio.h>

#include <stdbool.h>

#define ROZMIAR 3

char sprawdzWygrana(char plansza[][ROZMIAR]) {

// Sprawdzanie linii

for (int i = 0; i < ROZMIAR; i++) {

if (plansza[i][0] == plansza[i][1] && plansza[i][1] == plansza[i][2]) {

return plansza[i][0];

}

}

// Sprawdzanie kolumn

for (int i = 0; i < ROZMIAR; i++) {

if (plansza[0][i] == plansza[1][i] && plansza[1][i] == plansza[2][i]) {

return plansza[0][i];

}

}

// Sprawdzanie prz

```c

// Sprawdzanie przekątnych

if (plansza[0][0] == plansza[1][1] && plansza[1][1] == plansza[2][2]) {

return plansza[0][0];

}

if (plansza[0][2] == plansza[1][1] && plansza[1][1] == plansza[2][0]) {

return plansza[0][2];

}

// Jeżeli nie ma wygranej, zwracamy pusty znak

return ' ';

}

int main() {

char plansza[ROZMIAR][ROZMIAR] = {

{'X', 'O', 'X'},

{'O', 'X', 'O'},

{'X', 'O', ' '}

};

char wynik = sprawdzWygrana(plansza);

if (wynik != ' ') {

printf("Gracz %c wygrał!\n", wynik);

} else {

printf("Remis - plansza jest pełna.\n");

}

return 0;

}

```

W powyższym programie stworzona została tablica dwuwymiarowa `plansza` o rozmiarze 3x3, reprezentująca planszę do gry w kółko i krzyżyk. Następnie, za pomocą funkcji `sprawdzWygrana()`, sprawdzamy, czy na planszy znajduje się linia 3 znaków "X" lub "O" w jednej linii, kolumnie lub przekątnej. Funkcja zwraca znak gracza, który wygrał, lub pusty znak, jeśli nie ma wygranej. Na końcu w main() wypisywany jest odpowiedni komunikat na podstawie wyniku funkcji `sprawdzWygrana()`.

Mam nadzieję, że powyższe przykłady pomogą Ci w lepszym zrozumieniu zastosowania tablic wielowymiarowych w bardziej zaawansowanych zadaniach w języku C.

W języku C listy wskaźnikowe, zwane również listami jednokierunkowymi, są strukturami danych, które umożliwiają przechowywanie i manipulację danymi w sposób dynamiczny. Każdy element listy zawiera wartość oraz wskaźnik na następny element listy.

Oto przykład implementacji listy wskaźnikowej w języku C:

```c

#include <stdio.h>

#include <stdlib.h>

struct Element {

int value;

struct Element\* next;

};

typedef struct Element Element;

void dodajNaKoniec(Element\*\* head, int value) {

Element\* newElement = (Element\*)malloc(sizeof(Element));

newElement->value = value;

newElement->next = NULL;

if (\*head == NULL) {

\*head = newElement;

} else {

Element\* current = \*head;

while (current->next != NULL) {

current = current->next;

}

current->next = newElement;

}

}

void wyswietlListe(Element\* head) {

Element\* current = head;

while (current != NULL) {

printf("%d ", current->value);

current = current->next;

}

printf("\n");

}

void zwolnijPamiec(Element\* head) {

Element\* current = head;

while (current != NULL) {

Element\* temp = current;

current = current->next;

free(temp);

}

}

int main() {

Element\* lista = NULL;

dodajNaKoniec(&lista, 1);

dodajNaKoniec(&lista, 2);

dodajNaKoniec(&lista, 3);

wyswietlListe(lista);

zwolnijPamiec(lista);

return 0;

}

```

W powyższym przykładzie definiujemy strukturę `Element`, która przechowuje wartość oraz wskaźnik na następny element listy. Korzystając z tej struktury, implementujemy funkcje `dodajNaKoniec`, `wyswietlListe` i `zwolnijPamiec`.

Funkcja `dodajNaKoniec` tworzy nowy element, inicjalizuje go przekazaną wartością i dodaje go na koniec listy. Jeśli lista jest pusta, ustawia go jako głowę listy. W przeciwnym razie iteruje przez listę, aż znajdzie ostatni element, i przypisuje mu jako wskaźnik na następny nowy element.

Funkcja `wyswietlListe` iteruje przez listę, zaczynając od głowy, i wypisuje wartość każdego elementu.

Funkcja `zwolnijPamiec` iteruje przez listę, zwalniając pamięć dla każdego elementu.

W funkcji `main` tworzona jest lista, dodawane są do niej elementy, a następnie lista jest wyświetlana. Na końcu pamięć używana przez listę jest zwalniana.

To tylko podstawowy przykład implementacji listy wskaźnikowej w języku C. Istnieje wiele innych operacji, które można wykonać na listach, takich jak

Jeśli chcesz zaimplementować listę wskaźnikową bez elementu głowy, możemy zastosować strukturę rekurencyjną, w której każdy element listy przechowuje wartość oraz wskaźnik na następny element. W takim przypadku, dostęp do listy odbywa się poprzez wskaźnik na pierwszy element listy.

Oto przykład implementacji listy wskaźnikowej bez elementu głowy w języku C:

```c

#include <stdio.h>

#include <stdlib.h>

struct Element {

int value;

struct Element\* next;

};

typedef struct Element Element;

void dodajNaKoniec(Element\*\* head, int value) {

if (\*head == NULL) {

\*head = (Element\*)malloc(sizeof(Element));

(\*head)->value = value;

(\*head)->next = NULL;

} else {

dodajNaKoniec(&((\*head)->next), value);

}

}

void wyswietlListe(Element\* head) {

if (head == NULL) {

printf("Lista jest pusta.\n");

} else {

printf("Elementy listy: ");

while (head != NULL) {

printf("%d ", head->value);

head = head->next;

}

printf("\n");

}

}

void zwolnijPamiec(Element\* head) {

if (head == NULL) {

return;

}

zwolnijPamiec(head->next);

free(head);

}

int main() {

Element\* lista = NULL;

dodajNaKoniec(&lista, 1);

dodajNaKoniec(&lista, 2);

dodajNaKoniec(&lista, 3);

wyswietlListe(lista);

zwolnijPamiec(lista);

return 0;

}

```

W powyższym przykładzie funkcje `dodajNaKoniec`, `wyswietlListe` i `zwolnijPamiec` są implementowane rekurencyjnie.

W funkcji `dodajNaKoniec`, jeśli `head` (wskaźnik na pierwszy element listy) jest NULL, tworzony jest nowy element i ustawiany jako pierwszy element listy. W przeciwnym razie, rekurencyjnie wywołujemy `dodajNaKoniec` na wskaźniku `next` bieżącego elementu, aby dodać nowy element na końcu listy.

Funkcja `wyswietlListe` również jest implementowana rekurencyjnie. Jeśli `head` jest NULL, wypisywany jest komunikat o pustej liście. W przeciwnym razie, wypisujemy wartość bieżącego elementu, a następnie rekurencyjnie wywołujemy `wyswietlListe` na wskaźniku `next` bieżącego elementu.

Funkcja `zwolnijPamiec` również korzysta z rekurencji. Jeśli `head` jest NULL, zwracamy się z funkcji. W przeciwnym razie, rekurencyjnie wywołujemy `zwolnijPamiec` na wskaźniku `next` bieżącego elementu, a następnie zwalniam

Oto kilka zadań z rozwiązaniami wykorzystującymi listy wskaźnikowe bez elementu głowy w języku C:

Zadanie 1: Napisz funkcję, która dodaje nowy element na początek listy.

```c

#include <stdio.h>

#include <stdlib.h>

struct Element {

int value;

struct Element\* next;

};

typedef struct Element Element;

void dodajNaPoczatek(Element\*\* head, int value) {

Element\* newElement = (Element\*)malloc(sizeof(Element));

newElement->value = value;

newElement->next = \*head;

\*head = newElement;

}

void wyswietlListe(Element\* head) {

if (head == NULL) {

printf("Lista jest pusta.\n");

} else {

printf("Elementy listy: ");

while (head != NULL) {

printf("%d ", head->value);

head = head->next;

}

printf("\n");

}

}

void zwolnijPamiec(Element\* head) {

if (head == NULL) {

return;

}

zwolnijPamiec(head->next);

free(head);

}

int main() {

Element\* lista = NULL;

dodajNaPoczatek(&lista, 3);

dodajNaPoczatek(&lista, 2);

dodajNaPoczatek(&lista, 1);

wyswietlListe(lista);

zwolnijPamiec(lista);

return 0;

}

```

W powyższym przykładzie została dodana funkcja `dodajNaPoczatek`, która dodaje nowy element na początek listy. Nowy element jest tworzony, ustawiany jako pierwszy element listy, a wskaźnik na obecny pierwszy element listy jest przypisywany jako wskaźnik na następny element nowego elementu.

Zadanie 2: Napisz funkcję, która usuwa wszystkie elementy o podanej wartości z listy.

```c

#include <stdio.h>

#include <stdlib.h>

struct Element {

int value;

struct Element\* next;

};

typedef struct Element Element;

void usunElementy(Element\*\* head, int value) {

Element\* current = \*head;

Element\* previous = NULL;

while (current != NULL) {

if (current->value == value) {

if (previous == NULL) {

\*head = current->next;

} else {

previous->next = current->next;

}

Element\* temp = current;

current = current->next;

free(temp);

} else {

previous = current;

current = current->next;

}

}

}

void wyswietlListe(Element\* head) {

if (head == NULL) {

printf("Lista jest pusta.\n");

} else {

printf("Elementy listy: ");

while (head != NULL) {

printf("%d ", head->value);

head = head->next;

}

printf("\n");

}

}

void zwolnijPamiec(Element\* head) {

if (head == NULL) {

return;

}

zwolnijPamiec(head->next);

free(head);

}

int main() {

Element\* lista = NULL;

dodajNaPoczatek(&lista, 3);

dodajNa

```c

dodajNaPoczatek(&lista, 2);

dodajNaPoczatek(&lista, 1);

dodajNaPoczatek(&lista, 3);

dodajNaPoczatek(&lista, 2);

dodajNaPoczatek(&lista, 1);

dodajNaPoczatek(&lista, 3);

wyswietlListe(lista);

usunElementy(&lista, 2);

printf("Po usunieciu elementow o wartosci 2:\n");

wyswietlListe(lista);

zwolnijPamiec(lista);

return 0;

}

```

W tym przykładzie została dodana funkcja `usunElementy`, która usuwa wszystkie elementy o podanej wartości z listy. Funkcja przechodzi przez listę, sprawdzając wartość każdego elementu. Jeśli znaleziono element o podanej wartości, usuwa go z listy poprzez modyfikację wskaźników `next` w poprzednim elemencie i zwalnia pamięć dla usuniętego elementu.

W funkcji `main` zostają dodane elementy do listy, a następnie lista jest wyświetlana. Następnie wywołujemy funkcję `usunElementy` w celu usunięcia elementów o wartości 2. Po usunięciu elementów, lista jest ponownie wyświetlana.

Mam nadzieję, że te przykłady pomogą Ci zrozumieć implementację listy wskaźnikowej bez elementu głowy w języku C i zastosowanie jej w różnych zadaniach.

Oto kilka trudniejszych zadań z rozwiązaniami, wykorzystujących listy wskaźnikowe bez elementu głowy w języku C:

Zadanie 1: Napisz funkcję, która odwraca kolejność elementów w liście.

```c

#include <stdio.h>

#include <stdlib.h>

struct Element {

int value;

struct Element\* next;

};

typedef struct Element Element;

void odwrocListe(Element\*\* head) {

Element\* prev = NULL;

Element\* current = \*head;

Element\* next = NULL;

while (current != NULL) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

\*head = prev;

}

void wyswietlListe(Element\* head) {

if (head == NULL) {

printf("Lista jest pusta.\n");

} else {

printf("Elementy listy: ");

while (head != NULL) {

printf("%d ", head->value);

head = head->next;

}

printf("\n");

}

}

void zwolnijPamiec(Element\* head) {

if (head == NULL) {

return;

}

zwolnijPamiec(head->next);

free(head);

}

int main() {

Element\* lista = NULL;

dodajNaPoczatek(&lista, 3);

dodajNaPoczatek(&lista, 2);

dodajNaPoczatek(&lista, 1);

printf("Przed odwroceniem:\n");

wyswietlListe(lista);

odwrocListe(&lista);

printf("Po odwroceniu:\n");

wyswietlListe(lista);

zwolnijPamiec(lista);

return 0;

}

```

W powyższym przykładzie została dodana funkcja `odwrocListe`, która odwraca kolejność elementów w liście. Funkcja przechodzi przez listę, zmieniając wskaźniki `next` dla każdego elementu tak, aby wskazywały na poprzedni element. Na koniec, głowa listy zostaje ustawiona na ostatni element, aby odwrócić listę.

W funkcji `main` dodajemy kilka elementów do listy, a następnie wyświetlamy listę przed odwróceniem. Następnie wywołujemy funkcję `odwrocListe` i ponownie wyświetlamy listę po odwróceniu.

Zadanie 2: Napisz funkcję, która usuwa duplikaty z listy.

```c

#include <stdio.h>

#include <stdlib.h>

struct Element {

int value;

struct Element\* next;

};

typedef struct Element Element;

void usunDuplikaty(Element\*\* head) {

if (\*head == NULL) {

return;

}

Element\* current = \*head;

while (current != NULL) {

Element\* runner = current;

while (runner->next != NULL) {

if (runner->next->value == current->value) {

Element\* temp = runner->next;

runner->next = runner->next->next;

free(temp);

} else {

runner = runner->next;

}

}

current = current->next;

}

}

void wys

```c

void wyswietlListe(Element\* head) {

if (head == NULL) {

printf("Lista jest pusta.\n");

} else {

printf("Elementy listy: ");

while (head != NULL) {

printf("%d ", head->value);

head = head->next;

}

printf("\n");

}

}

void zwolnijPamiec(Element\* head) {

if (head == NULL) {

return;

}

zwolnijPamiec(head->next);

free(head);

}

int main() {

Element\* lista = NULL;

dodajNaPoczatek(&lista, 3);

dodajNaPoczatek(&lista, 2);

dodajNaPoczatek(&lista, 1);

dodajNaPoczatek(&lista, 3);

dodajNaPoczatek(&lista, 2);

dodajNaPoczatek(&lista, 1);

printf("Przed usunieciem duplikatow:\n");

wyswietlListe(lista);

usunDuplikaty(&lista);

printf("Po usunieciu duplikatow:\n");

wyswietlListe(lista);

zwolnijPamiec(lista);

return 0;

}

```

W tym przykładzie została dodana funkcja `usunDuplikaty`, która usuwa duplikaty z listy. Funkcja przechodzi przez listę, porównując wartość każdego elementu z wartościami kolejnych elementów. Jeśli zostanie znaleziony duplikat, usuwa się element poprzez modyfikację wskaźników `next` poprzedniego elementu.

W funkcji `main` dodajemy kilka elementów do listy, a następnie wyświetlamy listę przed usunięciem duplikatów. Następnie wywołujemy funkcję `usunDuplikaty` i ponownie wyświetlamy listę po usunięciu duplikatów.

Mam nadzieję, że te trudniejsze zadania z rozwiązaniami pomożą Ci lepiej zrozumieć i zaimplementować listy wskaźnikowe bez elementu głowy w języku C.

W języku C, napisy są reprezentowane jako tablice znaków zakończone znakiem null ('\0'). Operacje na napisach można wykonywać za pomocą różnych funkcji dostępnych w bibliotece standardowej języka C. Oto kilka podstawowych operacji na napisach:

1. Inicjalizacja napisu:

```c

char napis1[] = "Hello, world!"; // Inicjalizacja napisu za pomocą literałów znakowych

char napis2[10]; // Inicjalizacja pustego napisu o rozmiarze 10

```

2. Wyświetlanie napisu:

```c

#include <stdio.h>

char napis[] = "Hello, world!";

printf("%s\n", napis); // Wyświetlanie napisu na standardowym wyjściu

```

3. Kopiowanie napisu:

```c

#include <string.h>

char zrodlo[] = "Hello";

char cel[10];

strcpy(cel, zrodlo); // Kopiowanie zawartości napisu zrodlo do napisu cel

```

4. Łączenie napisów:

```c

#include <string.h>

char napis1[] = "Hello";

char napis2[] = "world!";

char wynik[20];

strcpy(wynik, napis1);

strcat(wynik, " ");

strcat(wynik, napis2); // Łączenie napisów napis1, spacji i napis2 w napis wynik

```

5. Porównywanie napisów:

```c

#include <string.h>

char napis1[] = "Hello";

char napis2[] = "Hello";

if (strcmp(napis1, napis2) == 0) {

printf("Napisy są identyczne.\n");

} else {

printf("Napisy są różne.\n");

}

```

6. Długość napisu:

```c

#include <string.h>

char napis[] = "Hello, world!";

int dlugosc = strlen(napis); // Zwraca długość napisu

```

7. Wyszukiwanie podnapisu w napisie:

```c

#include <string.h>

char napis[] = "Hello, world!";

char\* pozycja = strstr(napis, "world"); // Zwraca wskaźnik na początek podnapisu "world" w napisie

```

8. Zamiana liter napisu na małe lub duże:

```c

#include <ctype.h>

#include <stdio.h>

char napis[] = "Hello, world!";

for (int i = 0; napis[i] != '\0'; i++) {

napis[i] = tolower(napis[i]); // Zamienia literę na małą literę

// napis[i] = toupper(napis[i]); // Zamienia literę na dużą literę

}

printf("%s\n", napis);

```

To tylko kilka przykładów podstawowych operacji na napisach w języku C. Biblioteka standardowa języka C oferuje wiele innych funkcji, które umożliwiają bardziej zaawansowane operacje na napisach.

Oto kilka przykładowych zadań związanych z operacjami na napisach w języku C:

Zadanie 1: Napisać program, który sprawdza, czy dany napis jest palindromem.

```c

#include <stdio.h>

#include <string.h>

int czyPalindrom(const char\* napis) {

int dlugosc = strlen(napis);

int i, j;

for (i = 0, j = dlugosc - 1; i < j; i++, j--) {

if (napis[i] != napis[j]) {

return 0;

}

}

return 1;

}

int main() {

const char\* napis = "kajak";

if (czyPalindrom(napis)) {

printf("Napis '%s' jest palindromem.\n", napis);

} else {

printf("Napis '%s' nie jest palindromem.\n", napis);

}

return 0;

}

```

Zadanie 2: Napisać program, który zlicza ilość wystąpień danego podnapisu w napisie.

```c

#include <stdio.h>

#include <string.h>

int liczWystapienia(const char\* napis, const char\* podnapis) {

int dlugoscNapisu = strlen(napis);

int dlugoscPodnapisu = strlen(podnapis);

int licznik = 0;

if (dlugoscPodnapisu > dlugoscNapisu) {

return 0;

}

for (int i = 0; i <= dlugoscNapisu - dlugoscPodnapisu; i++) {

int znaleziono = 1;

for (int j = 0; j < dlugoscPodnapisu; j++) {

if (napis[i + j] != podnapis[j]) {

znaleziono = 0;

break;

}

}

if (znaleziono) {

licznik++;

}

}

return licznik;

}

int main() {

const char\* napis = "Ala ma kota, a kot ma Ale.";

const char\* podnapis = "ma";

int wystapienia = liczWystapienia(napis, podnapis);

printf("Podnapis '%s' występuje %d razy w napisie.\n", podnapis, wystapienia);

return 0;

}

```

Zadanie 3: Napisać program, który odwraca kolejność liter w danym napisie.

```c

#include <stdio.h>

#include <string.h>

void odwrocNapis(char\* napis) {

int dlugosc = strlen(napis);

int i, j;

for (i = 0, j = dlugosc - 1; i < j; i++, j--) {

char temp = napis[i];

napis[i] = napis[j];

napis[j] = temp;

}

}

int main() {

char napis[] = "Hello, world!";

printf("Przed odwroceniem: %s\n", napis);

odwrocNapis(napis);

printf("Po odwroceniu: %s\n", napis);

return 0;

}

```

To tylko kilka przykładów zadań

Oto kilka trudniejszych zadań związanych z operacjami na napisach w języku C:

Zadanie 1: Napisać program, który znajduje najdłuższy wspólny podciąg dwóch napisów.

```c

#include <stdio.h>

#include <string.h>

void najdluzszyWspolnyPodciag(const char\* napis1, const char\* napis2) {

int dlugosc1 = strlen(napis1);

int dlugosc2 = strlen(napis2);

int dlugosc[dlugosc1 + 1][dlugosc2 + 1];

int i, j, maxDlugosc = 0, koniec = 0;

for (i = 0; i <= dlugosc1; i++) {

for (j = 0; j <= dlugosc2; j++) {

if (i == 0 || j == 0) {

dlugosc[i][j] = 0;

} else if (napis1[i - 1] == napis2[j - 1]) {

dlugosc[i][j] = dlugosc[i - 1][j - 1] + 1;

if (dlugosc[i][j] > maxDlugosc) {

maxDlugosc = dlugosc[i][j];

koniec = i - 1;

}

} else {

dlugosc[i][j] = 0;

}

}

}

if (maxDlugosc == 0) {

printf("Nie ma wspolnego podciagu.\n");

return;

}

char wspolnyPodciag[maxDlugosc + 1];

for (i = 0; i < maxDlugosc; i++) {

wspolnyPodciag[i] = napis1[koniec - maxDlugosc + 1 + i];

}

wspolnyPodciag[maxDlugosc] = '\0';

printf("Najdluzszy wspolny podciag: %s\n", wspolnyPodciag);

}

int main() {

const char\* napis1 = "abcde";

const char\* napis2 = "acefg";

najdluzszyWspolnyPodciag(napis1, napis2);

return 0;

}

```

Zadanie 2: Napisać program, który sortuje tablicę napisów leksykograficznie.

```c

#include <stdio.h>

#include <string.h>

void sortujNapisy(char\*\* tablicaNapisow, int ilosc) {

int i, j;

char\* temp;

for (i = 0; i < ilosc - 1; i++) {

for (j = i + 1; j < ilosc; j++) {

if (strcmp(tablicaNapisow[i], tablicaNapisow[j]) > 0) {

temp = tablicaNapisow[i];

tablicaNapisow[i] = tablicaNapisow[j];

tablicaNapisow[j] = temp;

}

}

}

}

int main() {

char\* napisy[] = {"hello", "world", "open", "ai", "c"};

int ilosc = sizeof(napisy) / sizeof(napisy[0]);

printf("Przed sortow

Przed sortowaniem:

```c

for (int i = 0; i < ilosc; i++) {

printf("%s\n", napisy[i]);

}

```

Po sortowaniu:

```c

sortujNapisy(napisy, ilosc);

printf("Po sortowaniu:\n");

for (int i = 0; i < ilosc; i++) {

printf("%s\n", napisy[i]);

}

```

Zadanie 3: Napisać program, który zamienia liczby zapisane jako słowa na ich wartości liczbowe.

```c

#include <stdio.h>

#include <string.h>

int zamienNaLiczbe(const char\* liczba) {

int dlugosc = strlen(liczba);

int wynik = 0;

int jednostka = 0;

for (int i = 0; i < dlugosc; i++) {

switch (liczba[i]) {

case 'j':

jednostka = 1;

break;

case 'd':

jednostka = 2;

break;

case 't':

jednostka = 3;

break;

case 'c':

jednostka = 4;

break;

default:

wynik = wynik \* 10 + (liczba[i] - '0');

break;

}

}

return wynik \* jednostka;

}

int main() {

const char\* liczba = "stojdwa";

int wynik = zamienNaLiczbe(liczba);

printf("Wartość liczby '%s' to: %d\n", liczba, wynik);

return 0;

}

```

To są trudniejsze zadania związane z operacjami na napisach w języku C. Oczywiście istnieje wiele innych zadań i scenariuszy, w których można wykorzystać napisy w języku C.

Oto siedem trudnych zadań, które obejmują operacje na napisach, tablicach wielowymiarowych, strukturach i listach bez głowy w języku C:

Zadanie 1: Napisz program, który sprawdza, czy podany napis składa się tylko z unikalnych znaków.

Zadanie 2: Napisz funkcję, która zlicza liczbę wystąpień danego elementu w tablicy dwuwymiarowej.

Zadanie 3: Zaimplementuj strukturę danych reprezentującą graf skierowany. Napisz funkcje do dodawania wierzchołków i krawędzi, a także do przeszukiwania grafu (np. DFS, BFS).

Zadanie 4: Napisać program, który zamienia miejscami dwie kolumny w macierzy dwuwymiarowej.

Zadanie 5: Zaimplementuj stos przy użyciu listy bez głowy. Napisz funkcje do dodawania i usuwania elementów ze stosu.

Zadanie 6: Napisz program, który znajduje najdłuższy rosnący podciąg w tablicy liczb.

Zadanie 7: Zaimplementuj strukturę danych do przechowywania informacji o studentach, wraz z funkcjami dodawania nowego studenta, usuwania studenta i wyświetlania informacji o wszystkich studentach.

Te zadania wymagają zaawansowanej wiedzy i umiejętności związanych z napisami, tablicami wielowymiarowymi, strukturami i listami bez głowy w języku C. Przed rozwiązaniem tych zadań zaleca się gruntowną znajomość tych konceptów oraz umiejętność korzystania z odpowiednich funkcji i algorytmów.

Oto rozwiązania dla każdego z siedmiu zadań:

Zadanie 1: Sprawdzanie, czy napis składa się tylko z unikalnych znaków.

```c

#include <stdio.h>

#include <string.h>

int czyUnikalneZnaki(const char\* napis) {

int dlugosc = strlen(napis);

int wystapienia[256] = {0};

for (int i = 0; i < dlugosc; i++) {

int kodASCII = (int) napis[i];

if (wystapienia[kodASCII] > 0) {

return 0; // Znaleziono powtórzenie znaku

}

wystapienia[kodASCII]++;

}

return 1; // Wszystkie znaki są unikalne

}

int main() {

const char\* napis = "abcdefg";

if (czyUnikalneZnaki(napis)) {

printf("Napis '%s' sklada sie z unikalnych znakow.\n", napis);

} else {

printf("Napis '%s' zawiera powtorzone znaki.\n", napis);

}

return 0;

}

```

Zadanie 2: Zliczanie liczby wystąpień danego elementu w tablicy dwuwymiarowej.

```c

#include <stdio.h>

int liczWystapienia(int tablica[][3], int n, int m, int element) {

int licznik = 0;

for (int i = 0; i < n; i++) {

for (int j = 0; j < m; j++) {

if (tablica[i][j] == element) {

licznik++;

}

}

}

return licznik;

}

int main() {

int tablica[3][3] = {

{1, 2, 3},

{4, 5, 6},

{7, 8, 9}

};

int element = 5;

int wystapienia = liczWystapienia(tablica, 3, 3, element);

printf("Element %d wystepuje %d razy w tablicy.\n", element, wystapienia);

return 0;

}

```

Zadanie 3: Implementacja grafu skierowanego.

```c

#include <stdio.h>

#include <stdlib.h>

#define MAX\_WIERZCHOLKI 100

typedef struct Graf {

int liczbaWierzcholkow;

int macierzSasiedztwa[MAX\_WIERZCHOLKI][MAX\_WIERZCHOLKI];

} Graf;

void inicjalizujGraf(Graf\* graf, int liczbaWierzcholkow) {

graf->liczbaWierzcholkow = liczbaWierzcholkow;

for (int i = 0; i < liczbaWierzcholkow; i++) {

for (int j = 0; j < liczbaWierzcholkow; j++) {

graf->macierzSasiedztwa[i][j] = 0;

}

}

}

void dodajKrawedz(Graf\* graf, int wierzcholekPoczatkowy, int wierzcholekKoncowy) {

if (wierzcholekPoczatkowy >= 0 && wierzcholekPoczatkowy < graf

.liczbaWierzcholkow &&

wierzcholekKoncowy >= 0 && wierzcholekKoncowy < graf->liczbaWierzcholkow) {

graf->macierzSasiedztwa[wierzcholekPoczatkowy][wierzcholekKoncowy] = 1;

}

}

void dfs(Graf\* graf, int wierzcholek, int\* odwiedzone) {

odwiedzone[wierzcholek] = 1;

printf("%d ", wierzcholek);

for (int i = 0; i < graf->liczbaWierzcholkow; i++) {

if (graf->macierzSasiedztwa[wierzcholek][i] == 1 && !odwiedzone[i]) {

dfs(graf, i, odwiedzone);

}

}

}

void bfs(Graf\* graf, int wierzcholek, int\* odwiedzone) {

int kolejka[MAX\_WIERZCHOLKI];

int poczatek = 0;

int koniec = 0;

kolejka[koniec++] = wierzcholek;

odwiedzone[wierzcholek] = 1;

while (poczatek != koniec) {

int aktualnyWierzcholek = kolejka[poczatek++];

printf("%d ", aktualnyWierzcholek);

for (int i = 0; i < graf->liczbaWierzcholkow; i++) {

if (graf->macierzSasiedztwa[aktualnyWierzcholek][i] == 1 && !odwiedzone[i]) {

kolejka[koniec++] = i;

odwiedzone[i] = 1;

}

}

}

}

int main() {

Graf graf;

inicjalizujGraf(&graf, 6);

dodajKrawedz(&graf, 0, 1);

dodajKrawedz(&graf, 0, 2);

dodajKrawedz(&graf, 1, 3);

dodajKrawedz(&graf, 2, 3);

dodajKrawedz(&graf, 2, 4);

dodajKrawedz(&graf, 3, 4);

dodajKrawedz(&graf, 3, 5);

dodajKrawedz(&graf, 4, 5);

int odwiedzone[MAX\_WIERZCHOLKI] = {0};

printf("Przeszukiwanie DFS: ");

dfs(&graf, 0, odwiedzone);

printf("\n");

for (int i = 0; i < graf.liczbaWierzcholkow; i++) {

odwiedzone[i] = 0;

}

printf("Przeszukiwanie BFS: ");

bfs(&graf, 0, odwiedzone);

printf("\n");

return 0;

}

```

Zadanie 4: Zamiana miejscami dwóch kolumn w macierzy dwuwymiarowej.

```c

#include <stdio.h>

void zamienKolumny(int tablica[][3], int n, int kolumna1, int kolumna2) {

for (int i = 0; i < n; i++) {

int temp = tablica[i][kolumna1];

tablica

[i][kolumna1] = tablica[i][kolumna2];

tablica[i][kolumna2] = temp;

}

}

int main() {

int tablica[3][3] = {

{1, 2, 3},

{4, 5, 6},

{7, 8, 9}

};

int kolumna1 = 0;

int kolumna2 = 2;

printf("Przed zamiana:\n");

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

printf("%d ", tablica[i][j]);

}

printf("\n");

}

zamienKolumny(tablica, 3, kolumna1, kolumna2);

printf("\nPo zamianie:\n");

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

printf("%d ", tablica[i][j]);

}

printf("\n");

}

return 0;

}

```

Zadanie 5: Implementacja stosu przy użyciu listy bez głowy.

```c

#include <stdio.h>

#include <stdlib.h>

typedef struct Element {

int wartosc;

struct Element\* nastepny;

} Element;

void push(Element\*\* stos, int wartosc) {

Element\* nowy = (Element\*)malloc(sizeof(Element));

nowy->wartosc = wartosc;

nowy->nastepny = \*stos;

\*stos = nowy;

}

int pop(Element\*\* stos) {

if (\*stos == NULL) {

printf("Stos jest pusty.\n");

return -1;

}

int wartosc = (\*stos)->wartosc;

Element\* temp = \*stos;

\*stos = (\*stos)->nastepny;

free(temp);

return wartosc;

}

void wyswietlStos(Element\* stos) {

if (stos == NULL) {

printf("Stos jest pusty.\n");

return;

}

printf("Zawartosc stosu:\n");

Element\* aktualny = stos;

while (aktualny != NULL) {

printf("%d\n", aktualny->wartosc);

aktualny = aktualny->nastepny;

}

}

int main() {

Element\* stos = NULL;

push(&stos, 1);

push(&stos, 2);

push(&stos, 3);

wyswietlStos(stos);

int wartosc = pop(&stos);

printf("Zdjecie elementu ze stosu: %d\n", wartosc);

wyswietlStos(stos);

return 0;

}

```

Zadanie 6: Znalezienie najdłuższego rosnącego podciągu w tablicy liczb.

```c

#include <stdio.h>

void znajdzNajdluzszyPodciag(int tablica[], int n) {

int dlugoscAktualna = 1;

int dlugoscMax = 1;

int indeksStart = 0;

int indeksKoniec = 0;

for (int i = 1; i < n; i++) {

if (tablica[i] > tablica[i –

1]) {

dlugoscAktualna++;

if (dlugoscAktualna > dlugoscMax) {

dlugoscMax = dlugoscAktualna;

indeksStart = i - dlugoscMax + 1;

indeksKoniec = i;

}

} else {

dlugoscAktualna = 1;

}

}

printf("Najdluzszy rosnacy podciag: ");

for (int i = indeksStart; i <= indeksKoniec; i++) {

printf("%d ", tablica[i]);

}

printf("\n");

}

int main() {

int tablica[] = {1, 2, 3, 2, 5, 8, 7, 9, 10, 3, 6};

int n = sizeof(tablica) / sizeof(tablica[0]);

znajdzNajdluzszyPodciag(tablica, n);

return 0;

}

```

Zadanie 7: Odwrócenie napisu.

```c

#include <stdio.h>

#include <string.h>

void odwrocNapis(char\* napis) {

int dlugosc = strlen(napis);

int poczatek = 0;

int koniec = dlugosc - 1;

while (poczatek < koniec) {

char temp = napis[poczatek];

napis[poczatek] = napis[koniec];

napis[koniec] = temp;

poczatek++;

koniec--;

}

}

int main() {

char napis[] = "Hello, World!";

printf("Przed odwroceniem: %s\n", napis);

odwrocNapis(napis);

printf("Po odwroceniu: %s\n", napis);

return 0;

}

```

Mam nadzieję, że powyższe rozwiązania pomogą Ci w przygotowaniu się do trudniejszych zadań związanych z napisami, tablicami wielowymiarowymi, strukturami i listami bez głowy w języku C. Pamiętaj, że istnieje wiele różnych podejść do rozwiązywania problemów, więc te przykłady stanowią tylko jedną z możliwości.

Oto kolejne 7 zadań na kolokwium z napisów, tablic wielowymiarowych, struktur i list bez głowy w języku C:

Zadanie 1: Zamiana miejscami dwóch wierszy w macierzy dwuwymiarowej.

Zadanie 2: Obliczenie sumy dwóch macierzy dwuwymiarowych.

Zadanie 3: Zliczenie wystąpień danego znaku w napisie.

Zadanie 4: Usunięcie duplikatów z tablicy liczb.

Zadanie 5: Implementacja kolejki przy użyciu listy bez głowy.

Zadanie 6: Sortowanie napisów w alfabetycznej kolejności.

Zadanie 7: Przesunięcie elementów tablicy w lewo o określoną liczbę miejsc.

Jeżeli chcesz uzyskać pełne rozwiązania dla tych zadań, proszę oczekiwać, a ja postaram się dostarczyć odpowiedzi jak najszybciej.

Oto rozwiązania dla kolejnych 7 zadań na kolokwium:

Zadanie 1: Zamiana miejscami dwóch wierszy w macierzy dwuwymiarowej.

```c

void zamienWiersze(int tablica[][3], int wiersz1, int wiersz2) {

int temp[3];

for (int i = 0; i < 3; i++) {

temp[i] = tablica[wiersz1][i];

tablica[wiersz1][i] = tablica[wiersz2][i];

tablica[wiersz2][i] = temp[i];

}

}

```

Zadanie 2: Obliczenie sumy dwóch macierzy dwuwymiarowych.

```c

void dodajMacierze(int macierz1[][3], int macierz2[][3], int wynik[][3]) {

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

wynik[i][j] = macierz1[i][j] + macierz2[i][j];

}

}

}

```

Zadanie 3: Zliczenie wystąpień danego znaku w napisie.

```c

int zliczWystapienia(char\* napis, char znak) {

int wystapienia = 0;

int i = 0;

while (napis[i] != '\0') {

if (napis[i] == znak) {

wystapienia++;

}

i++;

}

return wystapienia;

}

```

Zadanie 4: Usunięcie duplikatów z tablicy liczb.

```c

void usunDuplikaty(int tablica[], int\* rozmiar) {

for (int i = 0; i < \*rozmiar; i++) {

for (int j = i + 1; j < \*rozmiar; j++) {

if (tablica[i] == tablica[j]) {

for (int k = j; k < \*rozmiar - 1; k++) {

tablica[k] = tablica[k + 1];

}

(\*rozmiar)--;

j--;

}

}

}

}

```

Zadanie 5: Implementacja kolejki przy użyciu listy bez głowy.

```c

#include <stdio.h>

#include <stdlib.h>

typedef struct Element {

int wartosc;

struct Element\* nastepny;

} Element;

void push(Element\*\* kolejka, int wartosc) {

Element\* nowy = (Element\*)malloc(sizeof(Element));

nowy->wartosc = wartosc;

nowy->nastepny = NULL;

if (\*kolejka == NULL) {

\*kolejka = nowy;

} else {

Element\* aktualny = \*kolejka;

while (aktualny->nastepny != NULL) {

aktualny = aktualny->nastepny;

}

aktualny->nastepny = nowy;

}

}

int pop(Element\*\* kolejka) {

if (\*kolejka == NULL) {

printf("Kolejka jest pusta.\n");

return -1;

}

int wartosc = (\*kolejka)->wartosc;

Element\* temp = \*kolejka;

\*k

olejka = (\*kolejka)->nastepny;

free(temp);

return wartosc;

}

void wyswietlKolejke(Element\* kolejka) {

if (kolejka == NULL) {

printf("Kolejka jest pusta.\n");

return;

}

printf("Zawartosc kolejki:\n");

Element\* aktualny = kolejka;

while (aktualny != NULL) {

printf("%d\n", aktualny->wartosc);

aktualny = aktualny->nastepny;

}

}

int main() {

Element\* kolejka = NULL;

push(&kolejka, 1);

push(&kolejka, 2);

push(&kolejka, 3);

wyswietlKolejke(kolejka);

int wartosc = pop(&kolejka);

printf("Zdjecie elementu z kolejki: %d\n", wartosc);

wyswietlKolejke(kolejka);

return 0;

}

```

Zadanie 6: Sortowanie napisów w alfabetycznej kolejności.

```c

#include <stdio.h>

#include <string.h>

void sortujNapisy(char\* napisy[], int n) {

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (strcmp(napisy[j], napisy[j + 1]) > 0) {

char\* temp = napisy[j];

napisy[j] = napisy[j + 1];

napisy[j + 1] = temp;

}

}

}

}

int main() {

char\* napisy[] = {"abc", "cba", "xyz", "def", "ghi"};

int n = sizeof(napisy) / sizeof(napisy[0]);

printf("Przed sortowaniem:\n");

for (int i = 0; i < n; i++) {

printf("%s ", napisy[i]);

}

printf("\n");

sortujNapisy(napisy, n);

printf("Po sortowaniu:\n");

for (int i = 0; i < n; i++) {

printf("%s ", napisy[i]);

}

printf("\n");

return 0;

}

```

Zadanie 7: Przesunięcie elementów tablicy w lewo o określoną liczbę miejsc.

```c

void przesunWLewo(int tablica[], int rozmiar, int k) {

k = k % rozmiar; // Jeśli k jest większe niż rozmiar, przesunięcie będzie cykliczne

for (int i = 0; i < k; i++) {

int temp = tablica[0];

for (int j = 0; j < rozmiar - 1; j++) {

tablica[j] = tablica[j + 1];

}

tablica[rozmiar - 1] = temp;

}

}

```

Mam nadzieję, że powyższe rozwiązania pomogą Ci w przygotowaniu się do trudniejszych zadań związanych z napisami, tablicami wielowymiarowymi, struktur

Oto kolejne 7 zadań na kolokwium z napisów, tablic wielowymiarowych, struktur i list bez głowy w języku C:

Zadanie 1: Sprawdzenie, czy dany napis jest palindromem (czy czyta się tak samo od przodu i od tyłu).

Zadanie 2: Obliczenie średniej arytmetycznej wartości znajdujących się na głównej przekątnej macierzy kwadratowej.

Zadanie 3: Zliczenie wystąpień danego słowa w napisie.

Zadanie 4: Usunięcie wszystkich elementów o określonej wartości z tablicy.

Zadanie 5: Implementacja stosu przy użyciu listy bez głowy.

Zadanie 6: Sortowanie dwuwymiarowej tablicy liczb w kolumnach w kolejności rosnącej.

Zadanie 7: Zamiana miejscami wartości dwóch struktur.

Jeżeli potrzebujesz pełnych rozwiązań dla tych zadań, proszę oczekiwać, a ja postaram się dostarczyć odpowiedzi jak najszybciej.

Oto rozwiązania dla kolejnych 7 zadań na kolokwium:

Zadanie 1: Sprawdzenie, czy dany napis jest palindromem.

```c

#include <stdio.h>

#include <string.h>

int czyPalindrom(char\* napis) {

int dlugosc = strlen(napis);

int poczatek = 0;

int koniec = dlugosc - 1;

while (poczatek < koniec) {

if (napis[poczatek] != napis[koniec]) {

return 0; // nie jest palindromem

}

poczatek++;

koniec--;

}

return 1; // jest palindromem

}

int main() {

char napis[] = "kajak";

if (czyPalindrom(napis)) {

printf("Napis jest palindromem.\n");

} else {

printf("Napis nie jest palindromem.\n");

}

return 0;

}

```

Zadanie 2: Obliczenie średniej arytmetycznej wartości znajdujących się na głównej przekątnej macierzy kwadratowej.

```c

#include <stdio.h>

float sredniaPrzekatnej(int macierz[][3], int rozmiar) {

float suma = 0.0;

int licznik = 0;

for (int i = 0; i < rozmiar; i++) {

suma += macierz[i][i];

licznik++;

}

return suma / licznik;

}

int main() {

int macierz[3][3] = {{1, 2, 3},

{4, 5, 6},

{7, 8, 9}};

float srednia = sredniaPrzekatnej(macierz, 3);

printf("Srednia arytmetyczna przekatnej macierzy: %.2f\n", srednia);

return 0;

}

```

Zadanie 3: Zliczenie wystąpień danego słowa w napisie.

```c

#include <stdio.h>

#include <string.h>

int zliczWystapieniaSlowa(char\* napis, char\* slowo) {

int wystapienia = 0;

char\* pozycja = strstr(napis, slowo);

while (pozycja != NULL) {

wystapienia++;

pozycja = strstr(pozycja + 1, slowo);

}

return wystapienia;

}

int main() {

char napis[] = "To jest przykladowy napis z przykladowym slowem";

char slowo[] = "przykladowym";

int wystapienia = zliczWystapieniaSlowa(napis, slowo);

printf("Liczba wystapien slowa '%s': %d\n", slowo, wystapienia);

return 0;

}

```

Zadanie 4: Usunięcie wszystkich elementów o określonej wartości z tablicy.

```c

#include <stdio.h>

void usunElementy(int tablica[], int\* rozmiar, int element) {

int indeks = 0;

for (int i = 0; i < \*rozmiar; i++) {

if (tablica[i

== element) {

continue;

}

tablica[indeks] = tablica[i];

indeks++;

}

\*rozmiar = indeks;

}

int main() {

int tablica[] = {1, 2, 3, 4, 3, 5, 6, 3};

int rozmiar = sizeof(tablica) / sizeof(tablica[0]);

int element = 3;

printf("Przed usunieciem elementow:\n");

for (int i = 0; i < rozmiar; i++) {

printf("%d ", tablica[i]);

}

printf("\n");

usunElementy(tablica, &rozmiar, element);

printf("Po usunieciu elementow:\n");

for (int i = 0; i < rozmiar; i++) {

printf("%d ", tablica[i]);

}

printf("\n");

return 0;

}

```

Zadanie 5: Implementacja stosu przy użyciu listy bez głowy.

```c

#include <stdio.h>

#include <stdlib.h>

typedef struct Element {

int wartosc;

struct Element\* nastepny;

} Element;

void push(Element\*\* stos, int wartosc) {

Element\* nowy = (Element\*)malloc(sizeof(Element));

nowy->wartosc = wartosc;

nowy->nastepny = \*stos;

\*stos = nowy;

}

int pop(Element\*\* stos) {

if (\*stos == NULL) {

printf("Stos jest pusty.\n");

return -1;

}

int wartosc = (\*stos)->wartosc;

Element\* temp = \*stos;

\*stos = (\*stos)->nastepny;

free(temp);

return wartosc;

}

void wyswietlStos(Element\* stos) {

if (stos == NULL) {

printf("Stos jest pusty.\n");

return;

}

printf("Zawartosc stosu:\n");

Element\* aktualny = stos;

while (aktualny != NULL) {

printf("%d\n", aktualny->wartosc);

aktualny = aktualny->nastepny;

}

}

int main() {

Element\* stos = NULL;

push(&stos, 1);

push(&stos, 2);

push(&stos, 3);

wyswietlStos(stos);

int wartosc = pop(&stos);

printf("Zdjecie elementu ze stosu: %d\n", wartosc);

wyswietlStos(stos);

return 0;

}

```

Zadanie 6: Sortowanie dwuwymiarowej tablicy liczb w kolumnach w kolejności rosnącej.

```c

#include <stdio.h>

void sortujKolumny(int tablica[][3], int wiersze, int kolumny) {

for (int j = 0; j < kolumny; j++) {

for (int i = 0; i < wiersze - 1; i++) {

for (int k = 0; k < wiersze - i - 1; k++) {

if (tablica[k][j] > tablica[k + 1][j]) {

int temp = tablica[k][j];

tablica[k][j] = tablica[k + 1][j];

tablica

[k + 1][j] = temp;

}

}

}

}

}

int main() {

int tablica[][3] = {{9, 3, 7},

{5, 1, 8},

{4, 6, 2}};

int wiersze = sizeof(tablica) / sizeof(tablica[0]);

int kolumny = sizeof(tablica[0]) / sizeof(tablica[0][0]);

printf("Przed sortowaniem:\n");

for (int i = 0; i < wiersze; i++) {

for (int j = 0; j < kolumny; j++) {

printf("%d ", tablica[i][j]);

}

printf("\n");

}

sortujKolumny(tablica, wiersze, kolumny);

printf("Po sortowaniu:\n");

for (int i = 0; i < wiersze; i++) {

for (int j = 0; j < kolumny; j++) {

printf("%d ", tablica[i][j]);

}

printf("\n");

}

return 0;

}

```

Zadanie 7: Zamiana miejscami wartości dwóch struktur.

```c

#include <stdio.h>

typedef struct Struktura {

int liczba;

char znak;

} Struktura;

void zamienStruktury(Struktura\* struktura1, Struktura\* struktura2) {

Struktura temp = \*struktura1;

\*struktura1 = \*struktura2;

\*struktura2 = temp;

}

int main() {

Struktura struktura1 = {1, 'a'};

Struktura struktura2 = {2, 'b'};

printf("Przed zamiana:\n");

printf("Struktura 1: liczba = %d, znak = %c\n", struktura1.liczba, struktura1.znak);

printf("Struktura 2: liczba = %d, znak = %c\n", struktura2.liczba, struktura2.znak);

zamienStruktury(&struktura1, &struktura2);

printf("Po zamianie:\n");

printf("Struktura 1: liczba = %d, znak = %c\n", struktura1.liczba, struktura1.znak);

printf("Struktura 2: liczba = %d, znak = %c\n", struktura2.liczba, struktura2.znak);

return 0;

}

```

Mam nadzieję, że powyższe rozwiązania pomogą Ci w przygotowaniu się do trudniejszych zadań związanych z napisami, tablicami wielowymiarowymi, strukturami i listami bez głowy. Pamiętaj, że w praktyce istnieje wiele różnych podejść i rozwiązań dla tych zadań, a powyższe przykłady stanowią tylko jedną z możliwości.

Oto kolejne 7 trudniejszych zadań na kolokwium z napisów, tablic wielowymiarowych, struktur i list bez głowy w języku C:

Zadanie 1: Znalezienie najdłuższego wspólnego podciągu dwóch napisów.

Zadanie 2: Zliczenie wystąpień wszystkich podciągów danej długości w napisie.

Zadanie 3: Mnożenie dwóch macierzy o różnych rozmiarach.

Zadanie 4: Implementacja kolejki przy użyciu listy bez głowy.

Zadanie 5: Sortowanie dwuwymiarowej tablicy liczb wierszami w kolejności malejącej.

Zadanie 6: Sprawdzenie, czy dwie struktury są identyczne.

Zadanie 7: Usunięcie duplikatów z tablicy.

Jeżeli potrzebujesz pełnych rozwiązań dla tych zadań, proszę oczekiwać, a ja postaram się dostarczyć odpowiedzi jak najszybciej.

Oto rozwiązania dla kolejnych 7 trudniejszych zadań na kolokwium:

Zadanie 1: Znalezienie najdłuższego wspólnego podciągu dwóch napisów.

```c

#include <stdio.h>

#include <string.h>

int max(int a, int b) {

return (a > b) ? a : b;

}

int najdluzszyWspolnyPodciag(char\* napis1, char\* napis2) {

int dlugosc1 = strlen(napis1);

int dlugosc2 = strlen(napis2);

int tablica[dlugosc1 + 1][dlugosc2 + 1];

for (int i = 0; i <= dlugosc1; i++) {

for (int j = 0; j <= dlugosc2; j++) {

if (i == 0 || j == 0)

tablica[i][j] = 0;

else if (napis1[i - 1] == napis2[j - 1])

tablica[i][j] = tablica[i - 1][j - 1] + 1;

else

tablica[i][j] = max(tablica[i - 1][j], tablica[i][j - 1]);

}

}

return tablica[dlugosc1][dlugosc2];

}

int main() {

char napis1[] = "abcdaf";

char napis2[] = "acbcf";

int wynik = najdluzszyWspolnyPodciag(napis1, napis2);

printf("Najdluzszy wspolny podciag: %d\n", wynik);

return 0;

}

```

Zadanie 2: Zliczenie wystąpień wszystkich podciągów danej długości w napisie.

```c

#include <stdio.h>

#include <string.h>

void zliczPodciagi(char\* napis, int dlugoscPodciagu) {

int dlugosc = strlen(napis);

if (dlugosc < dlugoscPodciagu) {

printf("Podany podciag jest dluzszy niz napis.\n");

return;

}

printf("Podciagi o dlugosci %d w napisie:\n", dlugoscPodciagu);

for (int i = 0; i <= dlugosc - dlugoscPodciagu; i++) {

for (int j = 0; j < dlugoscPodciagu; j++) {

printf("%c", napis[i + j]);

}

printf("\n");

}

}

int main() {

char napis[] = "abcdef";

int dlugoscPodciagu = 3;

zliczPodciagi(napis, dlugoscPodciagu);

return 0;

}

```

Zadanie 3: Mnożenie dwóch macierzy o różnych rozmiarach.

```c

#include <stdio.h>

void mnozenieMacierzy(int macierz1[][3], int wiersze1, int kolumny1, int macierz2[][2], int wiersze2, int kolumny2, int wynik[][2]) {

if (kolumny1 != wiersze2) {

printf("Nie mozna wykonac mnozenia mac

ierzyc o podanych rozmiarach.\n");

return;

}

for (int i = 0; i < wiersze1; i++) {

for (int j = 0; j < kolumny2; j++) {

wynik[i][j] = 0;

for (int k = 0; k < kolumny1; k++) {

wynik[i][j] += macierz1[i][k] \* macierz2[k][j];

}

}

}

}

void wyswietlMacierz(int macierz[][2], int wiersze, int kolumny) {

for (int i = 0; i < wiersze; i++) {

for (int j = 0; j < kolumny; j++) {

printf("%d ", macierz[i][j]);

}

printf("\n");

}

}

int main() {

int macierz1[][3] = {{1, 2, 3},

{4, 5, 6}};

int wiersze1 = sizeof(macierz1) / sizeof(macierz1[0]);

int kolumny1 = sizeof(macierz1[0]) / sizeof(macierz1[0][0]);

int macierz2[][2] = {{7, 8},

{9, 10},

{11, 12}};

int wiersze2 = sizeof(macierz2) / sizeof(macierz2[0]);

int kolumny2 = sizeof(macierz2[0]) / sizeof(macierz2[0][0]);

int wynik[wiersze1][kolumny2];

mnozenieMacierzy(macierz1, wiersze1, kolumny1, macierz2, wiersze2, kolumny2, wynik);

printf("Macierz wynikowa:\n");

wyswietlMacierz(wynik, wiersze1, kolumny2);

return 0;

}

```

Zadanie 4: Implementacja kolejki przy użyciu listy bez głowy.

```c

#include <stdio.h>

#include <stdlib.h>

typedef struct Element {

int wartosc;

struct Element\* nastepny;

} Element;

typedef struct Kolejka {

Element\* poczatek;

Element\* koniec;

} Kolejka;

void inicjalizujKolejke(Kolejka\* kolejka) {

kolejka->poczatek = NULL;

kolejka->koniec = NULL;

}

void dodajElement(Kolejka\* kolejka, int wartosc) {

Element\* nowy = (Element\*)malloc(sizeof(Element));

nowy->wartosc = wartosc;

nowy->nastepny = NULL;

if (kolejka->poczatek == NULL) {

kolejka->poczatek = nowy;

kolejka->koniec = nowy;

} else {

kolejka->koniec->nastepny = nowy;

kolejka->koniec = nowy;

}

}

int usunElement(Kolejka\* kolejka) {

if (kolejka->poczatek == NULL) {

printf("Kolejka jest pusta.\n");

return -1;

}

int wartosc = kolejka->poczatek->wartosc;

Element\* temp = kolejka->poczatek;

kolejka->poczatek = kolejka->poczatek->nastepny;

free(temp);

if (kolejka->poczatek == NULL) {

kolejka->koniec = NULL;

}

return wartosc;

}

void wyswietlKolejke(Kolejka\* kolejka) {

if (kolejka->poczatek == NULL) {

printf("Kolejka jest pusta.\n");

return;

}

printf("Elementy kolejki: ");

Element\* obecny = kolejka->poczatek;

while (obecny != NULL) {

printf("%d ", obecny->wartosc);

obecny = obecny->nastepny;

}

printf("\n");

}

int main() {

Kolejka kolejka;

inicjalizujKolejke(&kolejka);

dodajElement(&kolejka, 1);

dodajElement(&kolejka, 2);

dodajElement(&kolejka, 3);

wyswietlKolejke(&kolejka);

int usuniety = usunElement(&kolejka);

printf("Usuniety element: %d\n", usuniety);

wyswietlKolejke(&kolejka);

return 0;

}

```

Zadanie 5: Sortowanie dwuwymiarowej tablicy liczb wierszami w kolejności malejącej.

```c

#include <stdio.h>

void sortujTablice(int tablica[][3], int wiersze, int kolumny) {

for (int i = 0; i < wiersze; i++) {

for (int j = 0; j < kolumny - 1; j++) {

for (int k = 0; k < kolumny - j - 1; k++) {

if (tablica[i][k] < tablica[i][k + 1]) {

int temp = tablica[i][k];

tablica[i][k] = tablica[i][k + 1];

tablica[i][k + 1] = temp;

}

}

}

}

}

void wyswietlTablice(int tablica[][3], int wiersze, int kolumny) {

for (int i = 0; i < wiersze; i++) {

for (int j = 0; j < kolumny; j++) {

printf("%d ", tablica[i][j]);

}

printf("\n");

}

}

int main() {

int tablica[][3] = {{9, 3, 7},

{5, 1, 8},

{4, 6, 2}};

int wiersze = sizeof(tablica) / sizeof(tablica[0]);

int kolumny = sizeof(tablica[0]) / sizeof(tablica[0][0]);

printf("Przed sortowaniem:\n");

wyswietlTablice(tablica, wiersze, kolumny);

sortujTablice(tablica, wiersze, kolumny);

printf("Po sortowaniu:\n");

wyswietlTablice(tablica, wiersze, kolumny);

return 0;

}

```

Zadanie 6: Sprawdzenie, czy dwie struktury są identycz

y.

```c

#include <stdio.h>

#include <string.h>

typedef struct Osoba {

char imie[20];

int wiek;

} Osoba;

int porownajOsoby(Osoba osoba1, Osoba osoba2) {

if (strcmp(osoba1.imie, osoba2.imie) == 0 && osoba1.wiek == osoba2.wiek) {

return 1;

} else {

return 0;

}

}

int main() {

Osoba osoba1 = {"John", 30};

Osoba osoba2 = {"John", 30};

Osoba osoba3 = {"Jane", 25};

if (porownajOsoby(osoba1, osoba2)) {

printf("Osoba 1 i Osoba 2 sa identyczne.\n");

} else {

printf("Osoba 1 i Osoba 2 nie sa identyczne.\n");

}

if (porownajOsoby(osoba1, osoba3)) {

printf("Osoba 1 i Osoba 3 sa identyczne.\n");

} else {

printf("Osoba 1 i Osoba 3 nie sa identyczne.\n");

}

return 0;

}

```

Zadanie 7: Usuwanie duplikatów z listy bez głowy.

```c

#include <stdio.h>

#include <stdlib.h>

typedef struct Element {

int wartosc;

struct Element\* nastepny;

} Element;

void usunDuplikaty(Element\* lista) {

Element\* obecny = lista;

while (obecny != NULL) {

Element\* sprawdz = obecny;

while (sprawdz->nastepny != NULL) {

if (sprawdz->nastepny->wartosc == obecny->wartosc) {

Element\* temp = sprawdz->nastepny;

sprawdz->nastepny = sprawdz->nastepny->nastepny;

free(temp);

} else {

sprawdz = sprawdz->nastepny;

}

}

obecny = obecny->nastepny;

}

}

void wyswietlListe(Element\* lista) {

if (lista == NULL) {

printf("Lista jest pusta.\n");

return;

}

printf("Elementy listy: ");

Element\* obecny = lista;

while (obecny != NULL) {

printf("%d ", obecny->wartosc);

obecny = obecny->nastepny;

}

printf("\n");

}

int main() {

Element\* lista = (Element\*)malloc(sizeof(Element));

lista->wartosc = 1;

Element\* element2 = (Element\*)malloc(sizeof(Element));

element2->wartosc = 2;

lista->nastepny = element2;

Element\* element3 = (Element\*)malloc(sizeof(Element));

element3->wartosc = 3;

element2->nastepny = element3;

Element\* element4 = (Element\*)malloc(sizeof(Element));

element4->wartosc = 2;

element3->nastepny = element4;

Element\* element5 = (Element\*)malloc(sizeof(Element));

element5->wartosc = 1;

element4->nastepny = element5;

element5->nastepny = NULL;

printf("Przed us

uwaniem duplikatów:

```c

#include <stdio.h>

#include <stdlib.h>

typedef struct Element {

int wartosc;

struct Element\* nastepny;

} Element;

void usunDuplikaty(Element\* lista) {

Element\* obecny = lista;

while (obecny != NULL) {

Element\* sprawdz = obecny;

while (sprawdz->nastepny != NULL) {

if (sprawdz->nastepny->wartosc == obecny->wartosc) {

Element\* temp = sprawdz->nastepny;

sprawdz->nastepny = sprawdz->nastepny->nastepny;

free(temp);

} else {

sprawdz = sprawdz->nastepny;

}

}

obecny = obecny->nastepny;

}

}

void wyswietlListe(Element\* lista) {

if (lista == NULL) {

printf("Lista jest pusta.\n");

return;

}

printf("Elementy listy: ");

Element\* obecny = lista;

while (obecny != NULL) {

printf("%d ", obecny->wartosc);

obecny = obecny->nastepny;

}

printf("\n");

}

void usunListe(Element\* lista) {

Element\* obecny = lista;

while (obecny != NULL) {

Element\* temp = obecny;

obecny = obecny->nastepny;

free(temp);

}

}

int main() {

Element\* lista = (Element\*)malloc(sizeof(Element));

lista->wartosc = 1;

Element\* element2 = (Element\*)malloc(sizeof(Element));

element2->wartosc = 2;

lista->nastepny = element2;

Element\* element3 = (Element\*)malloc(sizeof(Element));

element3->wartosc = 3;

element2->nastepny = element3;

Element\* element4 = (Element\*)malloc(sizeof(Element));

element4->wartosc = 2;

element3->nastepny = element4;

Element\* element5 = (Element\*)malloc(sizeof(Element));

element5->wartosc = 1;

element4->nastepny = element5;

element5->nastepny = NULL;

printf("Przed usunięciem duplikatów:\n");

wyswietlListe(lista);

usunDuplikaty(lista);

printf("Po usunięciu duplikatów:\n");

wyswietlListe(lista);

usunListe(lista);

return 0;

}

```

To są rozwiązania dla kolejnych 7 trudniejszych zadań. Pamiętaj, że te rozwiązania mogą się różnić w zależności od używanego języka programowania. Upewnij się, że używasz odpowiednich składni i bibliotek dla danego języka.

W języku C tablica tablic i tablica wielowymiarowa to dwa różne sposoby reprezentowania struktur danych wielowymiarowych.

1. Tablica tablic (array of arrays):

Tablica tablic składa się z jednowymiarowych tablic, gdzie każda wewnętrzna tablica ma różne rozmiary. Możemy ją zdefiniować jako tablicę wskaźników na tablice jednowymiarowe. Przykładowa deklaracja tablicy tablic może wyglądać następująco:

```c

int\*\* tablica\_tablic;

```

Główną cechą tablicy tablic jest to, że możemy mieć tablice o różnych rozmiarach dla każdego wiersza. Nie jest to wymagane, ale możliwe.

2. Tablica wielowymiarowa (multidimensional array):

Tablica wielowymiarowa jest strukturą danych, w której dane są przechowywane w sposób jednoznaczny w wielu wymiarach. Może to być dwuwymiarowa, trójwymiarowa lub większa. Przykładowa deklaracja tablicy wielowymiarowej może wyglądać następująco:

```c

int tablica\_wielowymiarowa[3][3];

```

Tablica wielowymiarowa ma ustalony rozmiar w każdym z wymiarów i dane są przechowywane w sposób ciągły w pamięci.

Różnica między nimi polega na sposobie organizacji danych w pamięci. Tablica tablic pozwala na elastyczność w rozmiarach wewnętrznych tablic, ale może wymagać dynamicznego zarządzania pamięcią i wprowadzać większy narzut na dostęp do danych. Tablica wielowymiarowa jest bardziej zwięzła i efektywna pod względem dostępu do danych, ale wymaga określenia stałego rozmiaru w każdym z wymiarów.

Ostateczny wybór między tablicą tablic a tablicą wielowymiarową zależy od konkretnego problemu i wymagań dotyczących struktury danych.

W języku C tablica tablic (array of arrays) i tablica wielowymiarowa (multidimensional array) są dwiema różnymi koncepcjami, chociaż na pierwszy rzut oka mogą wydawać się podobne.

1. Tablica tablic: Tablica tablic jest złożona z pojedynczych tablic o różnych rozmiarach. Może to być zaimplementowane przy użyciu wskaźników na tablice. Każda tablica w tablicy tablic może mieć różną liczbę elementów. Przykładem może być deklaracja takiej tablicy:

```c

int tablicaTablic[3][2] = {

{1, 2},

{3, 4, 5},

{6, 7, 8, 9}

};

```

W tym przypadku mamy tablicę składającą się z trzech tablic, gdzie pierwsza tablica ma dwa elementy, druga tablica ma trzy elementy, a trzecia tablica ma cztery elementy.

2. Tablica wielowymiarowa: Tablica wielowymiarowa jest jedną tablicą o wielu wymiarach. Każdy wymiar ma stałą liczbę elementów. W języku C tablice wielowymiarowe są zazwyczaj implementowane jako bloki pamięci o stałej wielkości. Przykładem może być deklaracja takiej tablicy:

```c

int tablicaWielowymiarowa[3][2] = {

{1, 2},

{3, 4},

{5, 6}

};

```

W tym przypadku mamy tablicę dwuwymiarową o trzech wierszach i dwóch kolumnach.

Podsumowując, główna różnica między tablicą tablic a tablicą wielowymiarową polega na tym, że tablica tablic może mieć różne rozmiary dla każdej wewnętrznej tablicy, podczas gdy tablica wielowymiarowa ma stałe rozmiary we wszystkich wymiarach.