

Задачи на автомат

Задача 1 - Генерация треугольника Паскаля

Постановка задачи:

Напишите программу, которая выводит первые N строк треугольника Паскаля.

- Вход. Целое число N – количество требуемых строк треугольника Паскаля.
- Выход. N строк, каждая из которых содержит соответствующие коэффициенты треугольника Паскаля, разделённые пробелами.

- Требования. Для хранения каждой строки треугольника динамически выделять массив соответствующего

размера. Не использовать фиксированные размерности массивов, расчёт должен работать для любого N

разумного размера. После генерации всех строк освободить всю выделенную память (не допускать утечки памяти).

Математическая модель:

Треугольник Паскаля - это таблица чисел, где каждое число является суммой двух чисел, расположенных непосредственно выше. Крайние числа каждой строки равны 1. Строки начинаются с 1.

Список идентификаторов:

Имя переменной	Тип данных	Описание
n	size_t	Количество строк треугольника
i	size_t	Индекс строки
j	size_t	Индекс элемента в строке
triangle	int**	Массив указателей на строки

Код программы:

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

int main() {
    size_t n, i, j; // Переменные для количества строк и итераций

    // Запрос количества строк треугольника у пользователя
    printf("Enter number of lines: ");
    if (scanf("%zu", &n) != 1 || n <= 0) { // Проверка корректности ввода
        printf("Incorrect enter!");
        return 1; // Выход с кодом ошибки
    }

    // Выделение памяти для массива указателей на строки треугольника
```

```

int **triangle = (int **)malloc(n * sizeof(int *));
if (triangle == NULL) { // Проверка успешности выделения памяти
    printf("Error!");
    return 1;
}

// Выделение памяти для каждой строки треугольника
for (i = 0; i < n; i++) {
    // Каждая i-тая строка имеет i+1 элементов
    triangle[i] = (int *)malloc((i + 1) * sizeof(int));
    if (triangle[i] == NULL) { // Проверка выделения памяти
        printf("Error!");
        // Освобождение ранее выделенной памяти в случае ошибки
        for (size_t k = 0; k < i; k++) {
            free(triangle[k]);
        }
        free(triangle);
        return 1;
    }
}

// Заполнение треугольника Паскаля
for (i = 0; i < n; i++) {
    for (j = 0; j <= i; j++) {
        if (j == 0 || j == i) {
            // Крайние элементы строки всегда равны 1
            triangle[i][j] = 1;
        } else {
            // Остальные элементы - сумма двух вышестоящих
            triangle[i][j] = triangle[i-1][j-1] + triangle[i-1][j];
        }
    }
}

// Вывод треугольника Паскаля
printf("Pascal's triangle for %zu lines:", n);
for (i = 0; i < n; i++) {
    for (j = 0; j <= i; j++) {
        printf("%d ", triangle[i][j]); // Вывод элемента
    }
    printf("\n"); // Переход на новую строку
}

// Освобождение выделенной памяти
for (i = 0; i < n; i++) {
    free(triangle[i]); // Освобождение памяти для каждой строки
}
free(triangle); // Освобождение памяти для массива указателей

return 0; // Успешное завершение программы
}

```

Результаты работы программы:

```
C:\Users\ioan1\Desktop\C_Pro  X  +  v

Enter number of lines: 15
Pascal's triangle for 15 lines:
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1
1 11 55 165 330 462 462 330 165 55 11 1
1 12 66 220 495 792 924 792 495 220 66 12 1
1 13 78 286 715 1287 1716 1716 1287 715 286 78 13 1
1 14 91 364 1001 2002 3003 3432 3003 2002 1001 364 91 14 1

Process returned 0 (0x0)  execution time : 1.941 s
Press any key to continue.
```

```
C:\Users\ioan1\Desktop\C_Pro  X  +  v

Enter number of lines: number
Incorrect enter!

Process returned 1 (0x1)  execution time : 6.820 s
Press any key to continue.
```

Задача 2 - Проверка сбалансированности скобок в выражении

Постановка задачи:

Напишите программу для проверки корректности расстановки круглых, фигурных и квадратных скобок в заданной строке.

- Вход. Строка с выражением, содержащим скобки (может включать и другие символы).
- Выход. Вывести YES, если все типы скобок в строке корректно сбалансированы, или NO – если допущена ошибка в порядке скобок.
- Требования. Для проверки использовать стек. Алгоритм должен учитывать соответствие типов скобок (например, [соответствует]) и их порядок вложенности. Программа игнорирует несвязанные символы и анализирует только скобки. При обнаружении несбалансированной скобочной структуры обработать ситуацию и вывести NO. Использование стека должно быть реализовано вручную (например, через массив или связный список), без использования сторонних коллекций.

Математическая модель:

Программа использует структуру данных "стек" для хранения открывающих скобок. Когда встречается закрывающая скобка, программа извлекает из стека последнюю добавленную открытую скобку и проверяет, совпадают ли они. Если в процессе проверки стек остается пустым, значит, скобки сбалансированы.

Список идентификаторов:

Имя переменной	Тип данных	Описание
stack	struct	Стек для хранения открывающих скобок
input	char[]	Входная строка для проверки
current	char	Текущий символ в строке
last	char	Последняя открывающая скобка в стеке

Код программы:

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

#define MAX_SIZE 1000 // Максимальный размер стека

// Структура стека
typedef struct {
    char data[MAX_SIZE];
    int top;
} Stack;

// Инициализация стека
void initStack(Stack *s) {
```

```

    s->top = -1; // Индекс -1 означает, что стек пуст
}

// Проверка, пуст ли стек
bool isEmpty(Stack *s) {
    return s->top == -1;
}

// Добавление элемента в стек
bool push(Stack *s, char c) {
    if (s->top >= MAX_SIZE - 1) {
        return false; // Стек переполнен
    }
    s->data[++(s->top)] = c;
    return true;
}

// Извлечение элемента из стека
char pop(Stack *s) {
    if (isEmpty(s)) {
        return ' '; // Стек пуст
    }
    return s->data[(s->top)--];
}

// Проверка соответствия скобок
bool isMatchingPair(char opening, char closing) {
    return (opening == '(' && closing == ')') ||
           (opening == '{' && closing == '}') ||
           (opening == '[' && closing == ']');
}

// Основная функция проверки сбалансированности скобок
bool isBalanced(const char *str) {
    Stack stack;
    initStack(&stack);

    for (int i = 0; str[i] != ' '; i++) {
        char current = str[i];

        // Если текущий символ - открывающая скобка, помещаем в стек
        if (current == '(' || current == '{' || current == '[') {
            if (!push(&stack, current)) {
                return false; // Переполнение стека
            }
        }
        // Если текущий символ - закрывающая скобка
        else if (current == ')' || current == '}' || current == ']') {
            // Если стек пуст, значит нет соответствующей открывающей скобки
            if (isEmpty(&stack)) {
                return false;
            }

            // Извлекаем последнюю открывающую скобку из стека
            char last = pop(&stack);

```

```

        // Проверяем соответствие скобок
        if (!isMatchingPair(last, current)) {
            return false;
        }
    }
    // Игнорируем все остальные символы
}

// Если стек пуст, значит все скобки сбалансированы
return isEmpty(&stack);
}

int main() {
    char input[MAX_SIZE];

    printf("Enter line for checking: ");
    fgets(input, sizeof(input), stdin); // Чтение строки с консоли

    if (isBalanced(input)) {
        printf("YES");
    } else {
        printf("NO");
    }

    return 0;
}

```

Результаты работы программы:

```

C:\Users\ioan1\Desktop\C_Pro x + v
Enter line for checking: ([[{}]])
YES

Process returned 0 (0x0) execution time : 22.619 s
Press any key to continue.

```

```
C:\Users\ioan1\Desktop\C_Pro  X  +  v
Enter line for checking: ({[(())])}
NO

Process returned 0 (0x0)   execution time : 20.775 s
Press any key to continue.
|
```

Задача 3 - Поиск и замена подстроки в строке

Постановка задачи:

Реализуйте программу, которая во входной строке заменяет все вхождения заданной подстроки на другую подстроку.

- Вход. Три строки: первая – исходный текст, вторая – подстрока, которую нужно найти, третья – подстрока, на которую нужно заменить все вхождения.
- Выход. Одна строка – результат преобразования исходного текста, в котором все вхождения искомой подстроки заменены на новую подстроку.
- Требования. Реализовать поиск всех непересекающихся вхождений заданной подстроки в исходной строке и выполнение замены, формируя новую строку (в исходной строке замены не производятся, результат помещается в новый буфер). Программа должна корректно работать в случаях, когда новая подстрока короче или длиннее исходной (общая длина результата может изменяться). Не использовать готовые функции поиска подстрок из стандартных библиотек или регулярные выражения – логика поиска и замены должна быть реализована вручную (например, с помощью цикла и операций с указателями/индексами). При реализации следует уделить внимание управлению памятью: заранее оценить максимальный возможный размер результирующей строки и выделить соответствующий буфер (или перераспределять память по мере необходимости). По завершении работы освободить выделенную память.

Математическая модель:

Программа ищет все вхождения старой подстроки в исходной строке и заменяет их на новую подстроку. При этом вычисляется количество вхождений и создается новая строка, в которую будут записаны все изменения.

Список идентификаторов:

Имя переменной	Тип данных	Описание
source	char[]	Исходная строка
old_sub	char[]	Подстрока, которую нужно найти
new_sub	char[]	Подстрока, на которую нужно заменить
result	char*	Результирующая строка с заменами
current	const char*	Текущий указатель на символ в исходной строке
match	const char*	Указатель на начало найденного вхождения

Код программы:

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

// Функция для поиска подстроки в строке
// Возвращает указатель на первое вхождение или NULL
const char* find_substring(const char* str, const char* substr) {
    if (*substr == '\0') return str; // Пустая подстрока

    const char* p_str;
    const char* p_substr;
    const char* p_start = str;

    while (*p_start != '\0') {
        p_str = p_start;
        p_substr = substr;

        // Поиск совпадения
        while (*p_str != '\0' && *p_substr != '\0' && *p_str == *p_substr) {
            p_str++;
            p_substr++;
        }

        // Если дошли до конца подстроки - найдено вхождение
        if (*p_substr == '\0') {
            return p_start;
        }

        p_start++;
    }

    return NULL;
}

// Функция замены всех вхождений подстроки
char* replace_all(const char* source, const char* old_sub, const char* new_sub) {
    // Проверка на NULL
    if (source == NULL || old_sub == NULL || new_sub == NULL) {
```



```

    return NULL;
}

size_t old_len = strlen(old_sub);
size_t new_len = strlen(new_sub);
size_t source_len = strlen(source);

// Если заменяемая подстрока пустая, вернуть копию исходной строки
if (old_len == 0) {
    char* result = malloc(source_len + 1);
    if (result == NULL) return NULL;
    strcpy(result, source);
    return result;
}

// Подсчет количества вхождений
size_t count = 0;
const char* tmp = source;
while ((tmp = find_substring(tmp, old_sub)) != NULL) {
    count++;
    tmp += old_len; // Пропускаем найденное вхождение
}

// Вычисление размера новой строки
size_t result_len = source_len + count * (new_len - old_len);
char* result = malloc(result_len + 1); // +1 для нуль-терминатора
if (result == NULL) return NULL;

const char* current = source;
char* dest = result;

while (*current != '\0') {
    const char* match = find_substring(current, old_sub);

    // Если больше нет вхождений - копируем остаток строки
    if (match == NULL) {
        strcpy(dest, current);
        break;
    }

    // Копируем часть до вхождения
    size_t bytes_to_copy = match - current;
    strncpy(dest, current, bytes_to_copy);
    dest += bytes_to_copy;

    // Копируем новую подстроку
    strncpy(dest, new_sub, new_len);
    dest += new_len;

    // Пропускаем старую подстроку в исходной строке
    current = match + old_len;
}

// Завершаем строку нуль-терминатором
result[result_len] = '\0';

```

```

    return result;
}

int main() {
    char source[1024];
    char old_sub[256];
    char new_sub[256];

    printf("Enter original string: ");
    fgets(source, sizeof(source), stdin);
    source[strcspn(source, "\n")] = '\0'; // Удаляем символ новой строки

    printf("Enter substring for searching: ");
    fgets(old_sub, sizeof(old_sub), stdin);
    old_sub[strcspn(old_sub, "\n")] = '\0';

    printf("Enter substring for replacing: ");
    fgets(new_sub, sizeof(new_sub), stdin);
    new_sub[strcspn(new_sub, "\n")] = '\0';

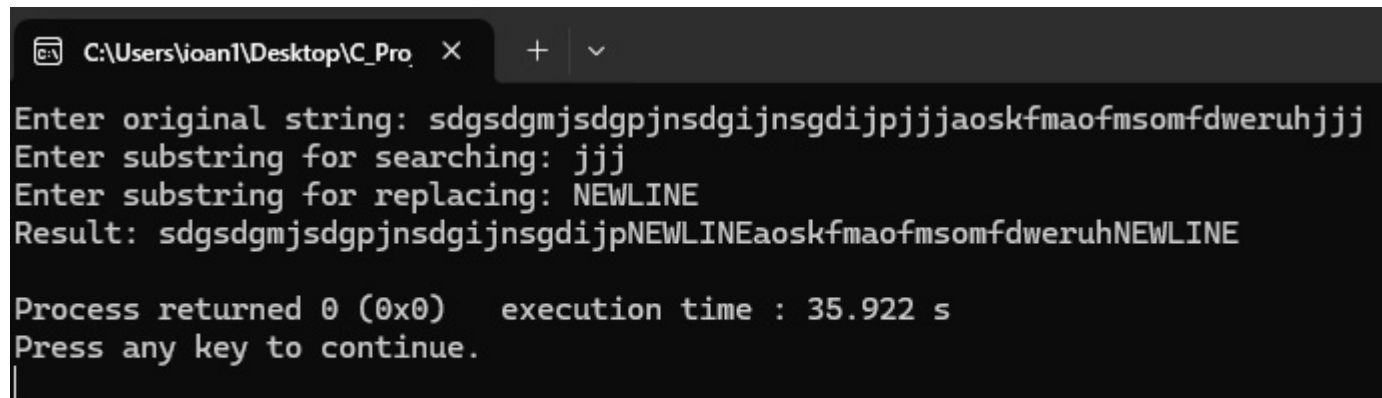
    char* result = replace_all(source, old_sub, new_sub);
    if (result == NULL) {
        printf("Error!\n");
        return 1;
    }

    printf("Result: %s\n", result);

    free(result); // Освобождаем память
    return 0;
}

```

Результаты работы программы:



```

C:\Users\ioan1\Desktop\C_Pro x + v
Enter original string: sdgsdgmjsdgpjnsdgijsdgiipjjjaoskfmaofmsomfdweruhjjj
Enter substring for searching: jjj
Enter substring for replacing: NEWLINE
Result: sdgsdgmjsdgpjnsdgijsdgiipNEWLINEaoskfmaofmsomfdweruhNEWLINE

Process returned 0 (0x0)   execution time : 35.922 s
Press any key to continue.
|

```

```
C:\Users\ioan1\Desktop\C_Pro  X + v
Enter original string: sdgsgdszgdpasfpoamsFok-mS-okfmsokmf-oaimfdnsjg0hsdng0anso
Enter substring for searching: 10
Enter substring for replacing: LINE
Result: sdgsgdszgdpasfpoamsFok-mS-okfmsokmf-oaimfdnsjg0hsdng0ansdg0

Process returned 0 (0x0)   execution time : 19.704 s
Press any key to continue.
|
```

Задача 4 - Арифметические операции с дробями

Постановка задачи:

. Разработайте программу для выполнения основных операций с обыкновенными дробями.

- Вход. Одна строка в формате A/B or C/D, где A/B и C/D – две дроби (числитель и знаменатель – целые числа, знаменатель не равен 0), а op – один из операторов +, -, * или /.
 - Выход. Результат операции в виде несократимой дроби P/Q (числитель и знаменатель — целые числа без общих делителей, знаменатель положительный). Если в результате получается целое число, вывести его как дробь с знаменателем 1 (например, 3/1).
 - Требования. Представить дроби с помощью структуры (с полями для числителя и знаменателя). Реализовать функции для вычисления суммы, разности, произведения и частного дробей, а также функцию для сокращения дроби (например, с использованием алгоритма Евклида для нахождения НОД).
- Программа должна корректно обрабатывать отрицательные дроби и случаи, когда в ходе вычислений происходит сокращение. В случае попытки деления на ноль (например, во входных данных знаменатель одной из дробей равен 0 или второй операнд равен 0/что-то при делении) вывести сообщение об ошибке.

Математическая модель:

Для выполнения операций над дробями используется стандартная арифметика с учетом числителей и знаменателей. Все дроби приводятся к общему знаменателю при выполнении операций сложения и вычитания. В случае деления дроби на другую дробь, числитель первой дроби умножается на знаменатель второй, а знаменатель первой дроби умножается на числитель второй.

После выполнения каждой операции дробь упрощается с использованием наибольшего общего делителя (НОД).

Список идентификаторов:

Имя переменной	Тип данных	Описание
a	int	Первая дробь
b	int	Вторая дробь
result	int	Результат выполнения операции
op	char	Оператор арифметической операции
input	char[]	Строка для ввода выражения
token	char*	Токен для разбора строки

Код программы:

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

// структура для представления дроби
typedef struct {
    int numerator; // Числитель
    int denominator; // Знаменатель
} Fraction;

// вычисление наибольшего общего делителя (НОД)
int gcd(int a, int b) {
    a = abs(a);
    b = abs(b);
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

// сокращение дроби
void simplify(Fraction *f) {
    int common_divisor = gcd(f->numerator, f->denominator);
    f->numerator /= common_divisor;
    f->denominator /= common_divisor;

    // проверка на положительный знаменатель
    if (f->denominator < 0) {
        f->numerator *= -1;
        f->denominator *= -1;
    }
}

// сложение двух дробей
Fraction add(Fraction a, Fraction b) {
    Fraction result;
```

```

    result.numerator = a.numerator * b.denominator + b.numerator * a.denominator;
    result.denominator = a.denominator * b.denominator;
    simplify(&result);
    return result;
}

// вычитание двух дробей
Fraction subtract(Fraction a, Fraction b) {
    Fraction result;
    result.numerator = a.numerator * b.denominator - b.numerator * a.denominator;
    result.denominator = a.denominator * b.denominator;
    simplify(&result);
    return result;
}

// умножение двух дробей
Fraction multiply(Fraction a, Fraction b) {
    Fraction result;
    result.numerator = a.numerator * b.numerator;
    result.denominator = a.denominator * b.denominator;
    simplify(&result);
    return result;
}

// деление двух дробей
Fraction divide(Fraction a, Fraction b) {
    if (b.numerator == 0) {
        fprintf(stderr, "Error! Division by zero");
        exit(EXIT_FAILURE);
    }
    Fraction result;
    result.numerator = a.numerator * b.denominator;
    result.denominator = a.denominator * b.numerator;
    simplify(&result);
    return result;
}

// парсинг дроби из строки
Fraction parseFraction(const char *str) {
    Fraction f;
    if (sscanf(str, "%d/%d", &f.numerator, &f.denominator) != 2) {
        fprintf(stderr, "Error! Incorrect format");
        exit(EXIT_FAILURE);
    }
    if (f.denominator == 0) {
        fprintf(stderr, "Error! Denominator can't be zero");
        exit(EXIT_FAILURE);
    }
    return f;
}

// печать дроби
void printFraction(Fraction f) {
    if (f.numerator == 0) {
        printf("0"); // Выводим просто 0, если числитель равен 0
    }
}

```

```

    } else if (f.denominator == 1) {
        printf("%d", f.numerator); // Выводим целое число, если знаменатель 1
    } else {
        printf("%d/%d", f.numerator, f.denominator); // Выводим обычную дробь
    }
}

int main() {
    char input[100];
    Fraction a, b, result;
    char op;

    printf("Enter the expression in the A/B op C/D format (for example, 1/2 + 3/4):");
    fgets(input, sizeof(input), stdin);

    // Парсинг входных данных
    char *token = strtok(input, " ");
    if (token == NULL) {
        fprintf(stderr, "Error! Empty");
        return EXIT_FAILURE;
    }
    a = parseFraction(token);

    token = strtok(NULL, " ");
    if (token == NULL || strlen(token) != 1 || strchr("+-*/", token[0]) == NULL) {
        fprintf(stderr, "Error! Incorrect operator");
        return EXIT_FAILURE;
    }
    op = token[0];

    token = strtok(NULL, " ");
    if (token == NULL) {
        fprintf(stderr, "Error! The second fraction is missing");
        return EXIT_FAILURE;
    }
    b = parseFraction(token);

    // Выполнение операции
    switch (op) {
        case '+': result = add(a, b); break;
        case '-': result = subtract(a, b); break;
        case '*': result = multiply(a, b); break;
        case '/': result = divide(a, b); break;
        default:
            fprintf(stderr, "Error! Unknown operator");
            return EXIT_FAILURE;
    }

    // Вывод результата
    printf("Result: ");
    printFraction(result);

    return 0;
}

```

Результаты работы программы:

```
"C:\Users\ioan1\Desktop\C_Pri X + v
Enter the expression in the A/B op C/D format (for example, 1/2 + 3/4): 2/3 + 1/2
Result: 5/6

Process returned 0 (0x0)    execution time : 13.738 s
Press any key to continue.
|
```

```
"C:\Users\ioan1\Desktop\C_Pri X + v
Enter the expression in the A/B op C/D format (for example, 1/2 + 3/4): 1/6 - 2/3
Result: -1/2

Process returned 0 (0x0)    execution time : 11.509 s
Press any key to continue.
|
```

```
"C:\Users\ioan1\Desktop\C_Pri X + v
Enter the expression in the A/B op C/D format (for example, 1/2 + 3/4): 3/4 * 2/3
Result: 1/2

Process returned 0 (0x0)    execution time : 19.583 s
Press any key to continue.
|
```

"C:\Users\ioan1\Desktop\C_Pri X

+

▼

Enter the expression in the A/B op C/D format (for example, 1/2 + 3/4): 3/5 / 5
Result: 9/25

Process returned 0 (0x0) execution time : 17.663 s
Press any key to continue.

|

"C:\Users\ioan1\Desktop\C_Pri X

+

▼

Enter the expression in the A/B op C/D format (for example, 1/2 + 3/4): 3/5 / 0/
Error! Division by zero

Process returned 1 (0x1) execution time : 116.047 s
Press any key to continue.

"C:\Users\ioan1\Desktop\C_Pri X

+

▼

Enter the expression in the A/B op C/D format (for example, 1/2 + 3/4): 4/2 / 2/
Error! Denominator can't be zero

Process returned 1 (0x1) execution time : 17.850 s
Press any key to continue.


```
"C:\Users\ioan1\Desktop\C_Pr... X + v
Enter the expression in the A/B op C/D format (for example, 1/2 + 3/4): 2/5 / 3
Error! Incorrect format

Process returned 1 (0x1)   execution time : 6.679 s
Press any key to continue.
|
```

Информация о студенте:

Топюк И.А., 1 курс, ИВТ 1-1