

Теоретический материал

Рекурсия — это такой способ организации вспомогательного алгоритма (подпрограммы), при котором эта подпрограмма (процедура или функция) в ходе выполнения ее операторов обращается сама к себе. То есть в теле функции она вызывает саму себя.

Практически любую рекурсивную функцию можно переписать нерекурсивным образом с применением циклов и условных операторов.

Например, вычисление факториала числа N , т. е. вычисление произведения всех чисел от 1 до N (в математике обозначается $N!$) нерекурсивным способом можно на C++ записать в виде следующего цикла;

```
int N=10; //будем вычислять факториал числа 10
int fact = 1; //переменная для «накопления» значения факториала
for(int i=1; i<=N; i++)
{
    fact*=i;
}
//По окончании цикла в переменной fact будет факториал числа N
```

Рекурсивно посчитать факториал можно, написав функцию, которая при выполнении вызывает себя. Такое возможно, поскольку $N! = N \cdot (N-1)!$

```
int factorial(int i)
{
    if (i==0) return 1;
    else return i*factorial(i-1);
}
```

Пример 1 демонстрирует вычисление факториала рекурсивным и нерекурсивным образом.

Ассоциативный массив — абстрактный тип данных, в котором хранятся пары «ключ — значение» с уникальными ключами. **Пара «ключ — значение»** состоит

из двух фрагментов данных, отображаемых вместе: ключа и значения. **Ключ** — это фрагмент данных для извлечения значения. **Значение** — фрагмент данных, для извлечения которого используется ключ. В качестве ключа могут выступать как целые числа, так и строки или какие-либо другие данные. Элемент «**Значение**» — это данные любой природы и сложности (начиная от целых чисел и заканчивая объектами классов).

Существует много различных реализаций ассоциативного массива, но наиболее популярной реализацией являются хеш-таблицы. Хеш-таблица — линейная структура данных, в которой хранятся пары «ключ — значение» с уникальными ключами, а это значит, что вы не можете хранить дубликаты ключей в хеш-таблице.

Языки программирования, конечно, содержат в готовом виде хэш-таблицу. В Python это словари, в C++ это `std::unordered_set`.

Для «ручной» реализации хэш-таблицы используется обычный массив. При этом в паре «ключ-значение» ключ преобразуется в индекс элемента массива. Для преобразования используют хэш-функцию, которая преобразует ключ в число из диапазона от 0 до размера массива для хранения хэш-таблицы. Хэш-функции бывают разные (хотя бы потому, что в качестве ключа могут выступать как числовые значения, так и, например, строки).

Рассмотрим пример. Для простоты, в данном примере элемент «значение» пары будет равняться ключу. Пустая хэш-таблица (на основе массива) содержит 7 элементов и выглядит так:



Хеш-таблица

Будем заполнять хэш-таблицу элементами:

86, 90, 27, 29, 38, 39, 40.

Первое число, которое нужно сохранить, — 86. Чтобы сохранить 86 в хеш-таблице, необходима хеш-функция. Одна из простых хеш-функций состоит в том, чтобы взять каждое число и выполнить операцию остатка от деления на количество элементов массива. Например, чтобы получить хеш-значение для числа 86, вы вычисляете $86 \% 7$. Результат равен 2, а значит, помещаем 86 в ячейку с индексом 2 в массиве, который используете для сохранения данных хеш-таблицы.

0	1	2	3	4	5	6
		86				

Следующее число, которое нужно сохранить в хеш-таблице, — 90, поэтому вы вычисляете $90 \% 7$, что равно 6. Итак, помещаем 90 в индекс 6 в вашем массиве.

0	1	2	3	4	5	6
		86				90

И наконец, нужно добавить 21, 29, 38, 39 и 40 в хеш-таблицу. Вот что произойдет, когда найдем остаток от деления на 7 для этих чисел:

$$27 \% 7 = 0$$

$$29 \% 7 = 1$$

$$38 \% 7 = 3$$

$$39 \% 7 = 4$$

$$40 \% 7 = 5$$

0	1	2	3	4	5	6
21	29	86	38	39	40	90

До сих пор добавление данных в хеш-таблицу происходило по плану. Предположим, мы хотим еще добавить число 30. Поскольку $30 \% 7$ равно 2, мы должны добавить 30 в слот 2. Здесь, однако, возникает проблема, потому что в этом слоте уже есть число 86. Когда два числа попадают в один и тот же слот, возникает **коллизия**. Чтобы разрешить ее, мы помещаем 30 в следующий пустой слот (допустим, в нашем массиве еще есть пустые слоты). Такое решение работает, однако, если вам потребуется найти число 30, придется сделать следующее:

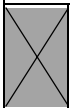
использовать хэш-функцию для поиска местоположения числа в массиве, проверить ячейку с индексом 2, понять, что в ней записано не число 30, а затем просматривать последующие ячейки до тех пор, пока вы не найдете искомое число. Все это добавляет вычислительной сложности.

Существуют и другие способы решения коллизий, такие как хранение в каждой ячейке массива указателя на «голову» связанного списка и помещение каждой конфликтующей пары в список, соответствующий исходной конфликтной ячейке. То есть в любом случае каждое «значение» помещается в связанный список, указатель на голову которого хранится в ячейке массива (хэш-таблицы). Если в связанном списке, предназначенном для добавляемого элемента, уже есть элемент, то новое значение добавляется в хвост хранимого связанного списка. Если элементов в списке нет, то указатель на голову является нулевым (`nullptr`).

Если есть необходимость сохранить сами ключи (отличающиеся от значений), то организуют еще один массив для хранения ключей. Индекс элемента для хранения ключа также определяется на основе хэш-функции.

Пример 1

Задача:



Написать на языке C++ программу, в которой факториал числа N считается с помощью цикла и с помощью рекурсивной процедуры.

Решение:

```

1  #include <iostream>
2
3  using namespace std;
4
5  //Рекурсивная функция для вычисления факториала
6  int factorial(int i)
7  {
8      if (i == 0) return 1;
9      else return i * factorial(i - 1);
10 }
11
12
13 int main()
14 {
15     setlocale(LC_ALL, "Russian");
16
17     int N = 7; //будем вычислять факториал числа 7
18
19     //Вычисление с помощью цикла
20     int fact = 1; //переменная для «накопления» значения факториала
21     for (int i = 1; i <= N; i++)
22     {
23         fact *= i;
24     } //По окончании цикла в переменной fact будет факториал числа N
25
26     std::cout << "Нерекурсивный расчет: " << N << "! = " << fact << endl;
27
28     //Вызов рекурсивной функции
29     std::cout << "Рекурсивный расчет: " << N << "! = " << factorial(N) << endl;
30
31 }

```

Ответ:

Консоль отладки Microsoft Visual Studio

```

Нерекурсивный расчет: 7! = 5040
Рекурсивный расчет: 7! = 5040

```

Задание 1

Задача:


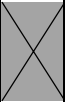


Написать программу с рекурсивной функцией нахождения НОД двух чисел по алгоритму Евклида (алгоритм объяснен в рабочей тетради 2, за основу можно взять нерекурсивный пример 1 из рабочей тетради 2). Рекурсивно НОД двух чисел определяется следующим образом:

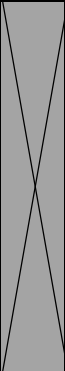


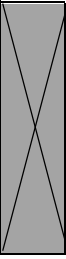
$$\text{НОД}(a, b) = \begin{cases} \text{НОД}(a - b, b), & \text{если } a > b \\ \text{НОД}(a, b - a), & \text{если } b > a \end{cases}$$

Выход из рекурсии происходит, когда $a=b$, это и есть НОД

Решение:

	Ответ:
Задание 2	
Задача:	
	Написать рекурсивную подпрограмму вычисления чисел Фибоначчи. $X_n = X_{n-1} + X_{n-2}$; $X_0 = 1$; $X_1 = 1$ (см. подсказки в слайдах с лекции)
Решение:	
Ответ:	
Задание 3	
Задача:	
	Переписать программу из рабочей тетради 1 для нахождения корня уравнения методом половинного деления, реализовав метод через рекурсивную процедуру (вариант тот же, что и в рабочей тетради 1, см. подсказки в слайдах с лекции)
Решение:	
Ответ:	
Задание 4*	
Задача:	
	Написать программу с рекурсивной функцией для нахождения суммы цифр числа.
Решение:	
Ответ:	
Задание 5*	
Задача:	
	Написать программу с рекурсивной функцией для нахождения значения функции Дейкстры: $\begin{cases} F(1) = 1 \\ F(2N) = F(N) \\ F(2N+1) = F(N) + F(N+1) \end{cases}$
Решение:	

Ответ:	
	
Задание 6**	
Задача:	
	Написать программу с рекурсивной функцией для нахождения решения головоломки «Ханойская башня» (без визуализации)
Решение:	
	
Ответ:	
	

Задание 7	
Задача:	
	Повторить пример, похожий на разобранный в «Теоретическом материале»: на вход программе подается цепочка целых чисел: 86, 90, 27, 29, 38, 30, 40. Сформировать из этого набора данных хэш-таблицу (ключ и значение совпадают). Размер массива равен 7. В качестве функции хэширования рассмотреть остаток от деления на 7. Выбрать любой способ избежания коллизии.
Решение:	
	
Ответ:	
	
Задание 8*	
Задача:	
	Создать класс «Хэш-таблица». Реализовать методы: 1) Добавление элемента в хэш-таблицу (при этом реализовать, как минимум, добавление пар «ключ-значение», соответствующих типам <int, string> и <string, string>. Избежание коллизий реализовать через связанный список.

- 2) Реализовать метод поиска элемента по ключу. В случае, если одному ключу соответствует несколько значений, вернуть значения в виде списка
- 3) Реализовать операцию удаления пары «ключ-значение». Если у ключа есть иные значения – удалять только значения
- 4) Продемонстрировать работу класса на примере телефонной книги, где ключи – фамилии, а значения – номера телефонов.

Ниже приведен пример хэш-функции для перевода строковых ключей (в виде массивов char) в индекс[^]

```
#define HASH_MUL 31
#define HASH_SIZE 128

unsigned int hash(char *s) {
    unsigned int h = 0;
    char *p;

    for (p = s; *p != '\0'; p++)
        h = h * HASH_MUL + (unsigned int)*p;
    return h % HASH_SIZE;
}
```

```
h = 0 * HASH_MUL + 105
h = 105 * HASH_MUL + 118
h = 3373 * HASH_MUL + 97
h = 104660 * HASH_MUL + 110
h = 3244570 * HASH_MUL + 111
h = 100581781 * HASH_MUL + 118
return 3118035329 % HASH_SIZE // hash("ivanov") = 1
```

i	v	a	n	o	v
105	118	97	110	111	118

HASH_SIZE – это размер хэш-таблицы.

Решение:

Ответ:

Задание 9

Задача:

Написать программу, на вход которой подается строка, а на выходе перечисляются содержащиеся в строке символы с указанием количества вхождений символа. Реализовать алгоритм с использованием хэш-таблицы. Можно использовать самостоятельно написанную хэш-таблицу, но можно использовать и готовые решения (например, словари в Python).

Рекомендации по созданию алгоритма:

- 1) Реализовать алгоритм в виде функции, на вход которой подается строка)
- 2) Создаем пустую хэш-таблицу
- 3) В цикле for перебираем символы строки

- 4) Если символа еще нет в хэш-таблице, добавляем в первый массив (в первую по счету пустую ячейку) новый ключ (символ), а во второй массив значение 1 (в ячейку с тем же индексом), поскольку символ впервые появляется в хэш-таблице.
- 5) Если символ есть в хэш-таблице, увеличиваем значение на 1
- 6) После прогона через цикл всей строки выводим на печать пары «ключ-значение»

Решение:

Ответ:

Задание 10*

Задача:

Задача «сумма двух». Вернуть индексы двух чисел в неотсортированном массиве, которые в сумме дают заданное значение. Решить задачу на основе хэш-таблицы. Например, для массива [-1, 2, 3, 4, 7] и заданного значения 5 выводятся индексы 1 и 2 (поскольку $2+3=5$).

Решение:

Ответ:

Задание 11**

Задача:

Преобразовать данные в JSON-формате в хэш-таблицу.

Решение:

Ответ: