

```

const Express = require('express');
const ExpressSession = require('express-session');
const Cors = require('cors');
const Utils = require('./Utils');

```

Thu, 6 hours ago | 1 author (This)

```

class HttpService {
  /** @type {Express.Express} */
  app;

  /** @private */
  constructor() {
    this.app = Express();
    this.app.use(Express.json());
    this.app.use(ExpressSession({
      secret: Utils.randomUID(),
      resave: true,
      saveUninitialized: true
    }));
    this.app.use(Cors({
      origin: 'http://localhost:3000',
      credentials: true
    }));
    this.app.post('/user', HttpService.new_user);
    this.app.put('/user', HttpService.transfer_user);
    this.app.put('/session', HttpService.auth_session);
    this.app.get('/session', HttpService.check_session);
    this.app.get('/transfer_id', HttpService.generate_transfer_id);
    this.app.get('/click', HttpService.get_click);
    this.app.put('/click', HttpService.add_click);

    this.app.listen(8080);
    console.log('[HttpService] Listening on port 8080');
  }

  static async new() {
    return new HttpService();
  }

  /**
   * @param {Express.Response} res
   * @param {number} status
   * @param {Object} json
   */
  static endAndSend(res, status, json) {
    res.status(status);
    res.json(json);
    res.end();
  }

  /**
   * @private

```

資料結構作業

第一次作業

資 訊 二 乙
周冠謀D1210270

>> 目錄 Contents

一、 題目定義及作法說明 p3
二、 執行結果（測試資料及結果） p4
三、 心得與討論（遭遇困難及解決過程） p5

❖ 題目定義及作法說明

題目定義

撰寫一個程式，輸入一個數字 X ，輸出 X 個位元的所有 0 和 1 的排列變化，即列出所有可能的二進位組合。

作法及說明

為了產生所有 X 個位元的 0 和 1 的排列組合，我們採用了遞迴的方法。遞迴能夠便利地遍歷所有可能的組合，以下是詳細的作法說明：

遞迴函數設計：

- 建立一個遞迴函數 `generatePermutations(int n, vector<int>& permutation)`。
- n 表示還需要加入的位元數量。
- `permutation` 是目前已生成的位元陣列。

遞迴終止條件：

- 當 $n == 0$ 時，表示已經生成了一個完整的位元陣列，將其輸出。

遞迴過程：

- 加入 0：
 - 在 `permutation` 中加入一個 0。
 - 呼叫遞迴函數 `generatePermutations(n - 1, permutation)`。
 - 遞迴返回後，移除剛才加入的 0(回溯)。
- 加入 1：
 - 在回溯後的 `permutation` 中加入一個 1。
 - 呼叫遞迴函數 `generatePermutations(n - 1, permutation)`。
 - 遞迴返回後，移除剛才加入的 1(回溯)。

程式碼結構：

- 主函數中，讀取使用者輸入的位元數 X 。
- 初始化一個空的 `vector<int>`，用於存放位元陣列。
- 呼叫遞迴函數開始生成排列。

❖ 測試資料及結果

Enter the number of bits: 3	Enter the number of bits: 4
000	0000
001	0001
010	0010
011	0011
100	0100
101	0101
110	0110
111	0111
	1000
	1001
	1010
	1011
	1100
	1101
	1110
	1111

❖ 心得與討論(遭遇困難及解決過程)

遭遇的困難

1. 理解遞迴的實現：
 - 一開始對於如何使用遞迴產生所有組合感到困惑，特別是遞迴函數的參數設置和遞迴終止條件的確定。
2. 回溯機制的運用：
 - 在遞迴返回時，如何正確地移除之前加入的元素(即回溯)以避免影響後續的遞迴調用，是一個挑戰。

解決過程

1. 學習遞迴的原理：
 - 查閱相關資料，理解遞迴的基本概念，即一個函數調用自身，並且每次調用都需要向著終止條件邁進。
2. 手寫遞迴流程：
 - 在紙上模擬小規模輸入(如 $X = 2$)的遞迴過程，追蹤 **permutation** 的變化，加深對遞迴和回溯的理解。
3. 除錯程式：
 - 使用除錯工具或在程式中加入輸出語句，觀察遞迴函數的執行流程，特別是進入和退出遞迴時 **permutation** 的狀態。

心得體會

- 遞迴與回溯的重要性：
 - 這次作業讓我深刻體會到遞迴和回溯在解決組合問題中的強大之處。
- 學習方法的改進：
 - 未來在面對類似的問題時，會先嘗試手動模擬小規模的情況，以更好地理解算法執行機制。
- 程式能力的提升：
 - 通過這次實踐，不僅熟悉了 **C++** 中的遞迴函數設計，還提高了對程式除錯的能力。