以下是有關在樹莓派（Raspberry Pi）上使用 C++ 進行程式設計的詳細教學，涵蓋基礎到進階的內容。本教學將從環境設定開始，逐步介紹 C++ 基礎語法、物件導向程式設計、標準模板庫（STL）、樹莓派 GPIO 控制，以及進階應用如多執行緒、感測器整合與網路程式設計。範例程式碼均針對樹莓派環境設計，並附有詳細解釋與硬體連線指引。

目錄

1. **環境設定**
2. **C++ 基礎教學**
    - 變數與資料型別
    - 控制結構
    - 函數
3. **物件導向程式設計**
    - 類別與物件
    - 繼承與多型
4. **標準模板庫（STL）**
    - 容器（vector、map）
    - 演算法
5. **樹莓派 GPIO 控制**
    - 使用 WiringPi 控制 LED
    - 按鈕輸入
6. **進階應用**
    - 多執行緒程式設計

- 感測器數據讀取（DHT11）
- 網路程式設計（TCP 客戶端/伺服器）

7. **除錯與最佳化**
8. **學習資源與建議**
9. **結論**

---

1. 環境設定

在樹莓派上使用 C++ 需要安裝編譯器、函式庫與開發工具。以下是詳細步驟：

硬體與作業系統

- **硬體**：樹莓派（推薦 Pi 4 或 Pi 5，Pi 3 亦可）、microSD 卡（16GB 或以上）、電源、鍵盤、滑鼠、顯示器（或透過 SSH/VNC 遠端連線）。
- **作業系統**：Raspberry Pi OS（64 位元 Desktop 版推薦）。使用 Raspberry Pi Imager 燒錄至 microSD 卡。
- **更新系統**：

  ```bash
  sudo apt update
  sudo apt upgrade
  ```

安裝 C++ 編譯器

Raspberry Pi OS 預裝 GCC，支援 C++ 編譯。

- 安裝 GCC 與相關工具：

  ```bash
  sudo apt install g++ make build-essential
  ```

- 檢查版本：

  ```bash
  g++ --version
  ```

  應顯示 GCC 版本（如 10.x 或更高）。

安裝開發工具

- **文字編輯器**：
  - **Nano**（簡單）：
    ```bash
    sudo apt install nano
    ```
  - **Vim**（進階）：
    ```bash
    sudo apt install vim
    ```
- **整合開發環境（IDE）**：
  - **Geany**（輕量）：
    ```bash
    sudo apt install geany
    ```

```

- **Visual Studio Code**（功能豐富，Pi 4 以上推薦）：

  ```bash
  sudo apt install code
  ```

- **GPIO 控制函式庫**：

  - 安裝 WiringPi（用於 GPIO 控制）：

    ```bash
    sudo apt install wiringpi
    ```

  - 檢查 WiringPi：

    ```bash
    gpio -v
    ```

測試環境

創建並運行第一個 C++ 程式：

1. 建立檔案 `test.cpp`：

   ```bash
   nano test.cpp
   ```

2. 輸入程式碼：

   ```cpp
   #include <iostream>
```
```

```
int main() {
    std::cout << "Welcome to Raspberry Pi C++ Programming!" << std::endl;
    return 0;
}
```

3. 編譯與執行：

```bash
g++ test.cpp -o test
./test
```

應輸出：`Welcome to Raspberry Pi C++ Programming!`

---

2. C++ 基礎教學

以下介紹 C++ 的基礎語法，適合初學者，並以樹莓派為背景設計範例。

2.1 變數與資料型別

C++ 支援多種資料型別，用於儲存不同類型的數據。

- 範例：宣告與使用變數

```cpp
#include <iostream>
#include <string>
int main() {
```

```
    int age = 25;                    // 整數
    double temperature = 23.5; // 浮點數
    char grade = 'A';                // 字元
    std::string name = "PiUser"; // 字串
    bool isActive = true;            // 布林值

    std::cout << "Name: " << name << std::endl;
    std::cout << "Age: " << age << ", Temp: " << temperature << "°C" << std::endl;
    std::cout << "Grade: " << grade << ", Active: " << isActive << std::endl;
    return 0;
}
```

- **編譯與執行**：

  ```bash
  g++ variables.cpp -o variables
  ./variables
  ```

- **說明**：`std::cout` 用於輸出，`<<` 是串流運算子，`std::endl` 換行。

2.2 控制結構

控制結構用於條件判斷與迴圈。

- 範例：if 與 for 迴圈

  ```cpp
  ```

```
#include <iostream>

int main() {
    int sensorValue = 42;
    // 條件判斷
    if (sensorValue > 50) {
        std::cout << "Sensor value too high!" << std::endl;
    } else if (sensorValue > 30) {
        std::cout << "Sensor value normal." << std::endl;
    } else {
        std::cout << "Sensor value low." << std::endl;
    }
    // 迴圈
    for (int i = 1; i <= 5; i++) {
        std::cout << "Loop " << i << ": Sensor check" << std::endl;
    }
    return 0;
}
```

- **輸出**：

```
Sensor value normal.
Loop 1: Sensor check
Loop 2: Sensor check
Loop 3: Sensor check
```

Loop 4: Sensor check

Loop 5: Sensor check

```

### **2.3 函數**

函數用於模組化程式碼，提高可讀性與重用性。

- 範例：計算溫度的平均值

```cpp
#include <iostream>
double calculateAverage(double a, double b, double c) {
    return (a + b + c) / 3.0;
}
int main() {
    double temp1 = 22.5, temp2 = 23.0, temp3 = 24.5;
    double avg = calculateAverage(temp1, temp2, temp3);
    std::cout << "Average temperature: " << avg << "°C" << std::endl;
    return 0;
}
```

- **說明**：函數 `calculateAverage` 接受三個參數並返回平均值。

---

3. 物件導向程式設計
```

C++ 的物件導向特性適合樹莓派上的模組化開發，如硬體控制系統。

3.1 類別與物件

類別是物件導向的核心，用於封裝數據與行為。

- 範例：模擬 LED 控制

```cpp
#include <iostream>
class LED {
private:
    int pinNumber;
    bool isOn;
public:
    LED(int pin) : pinNumber(pin), isOn(false) {
        std::cout << "LED initialized on pin " << pin << std::endl;
    }
    void turnOn() {
        isOn = true;
        std::cout << "LED on pin " << pinNumber << " is ON" << std::endl;
    }
    void turnOff() {
        isOn = false;
        std::cout << "LED on pin " << pinNumber << " is OFF" << std::endl;
    }
    bool getState() {
```

```
        return isOn;
    }
};
int main() {
    LED led1(18); // GPIO 18
    led1.turnOn();
    led1.turnOff();
    std::cout << "LED state: " << (led1.getState() ? "ON" : "OFF") << std::endl;
    return 0;
}
```

- **說明**：`LED` 類別封裝了引腳編號與狀態，模擬硬體控制邏輯。

3.2 繼承與多型

繼承允許類別重用， 多型實現靈活的行為。

- 範例：感測器基類與衍生類

```cpp
#include <iostream>
#include <string>
class Sensor {
protected:
    std::string name;
public:
    Sensor(const std::string& n) : name(n) {}
```

```
    virtual void readData() = 0; //  純虛函數
    virtual ~Sensor() {}
};
class TemperatureSensor : public Sensor {
private:
    double temperature;
public:
    TemperatureSensor(const std::string& n) : Sensor(n), temperature(0.0) {}
    void readData() override {
        temperature = 23.5; //  模擬讀取
        std::cout << name << " reads: " << temperature << "°C" << std::endl;
    }
};
class HumiditySensor : public Sensor {
private:
    double humidity;
public:
    HumiditySensor(const std::string& n) : Sensor(n), humidity(0.0) {}
    void readData() override {
        humidity = 65.0; //  模擬讀取
        std::cout << name << " reads: " << humidity << "%" << std::endl;
    }
};
int main() {
```

```
        Sensor* sensors[] = {
            new TemperatureSensor("Temp1"),
            new HumiditySensor("Hum1")
        };
        for (Sensor* s : sensors) {
            s->readData();
            delete s;
        }
        return 0;
    }
    ```
- **輸出**：
    ```
    Temp1 reads: 23.5°C
    Hum1 reads: 65%
    ```
- **說明**：`Sensor` 是抽象基類，`TemperatureSensor` 與 `HumiditySensor` 實現多型。

---

4. 標準模板庫（STL）

STL 提供容器與演算法，簡化數據處理。

4.1 容器

- **vector**：動態陣列

  ```cpp
  #include <iostream>
  #include <vector>
  int main() {
      std::vector<double> temperatures = {22.5, 23.0, 24.5};
      temperatures.push_back(25.0);
      std::cout << "Temperatures: ";
      for (double temp : temperatures) {
          std::cout << temp << " ";
      }
      std::cout << std::endl;
      return 0;
  }
  ```

- **map**：鍵值對

  ```cpp
  #include <iostream>
  #include <map>
  #include <string>
  int main() {
      std::map<std::string, double> sensorData;
      sensorData["Temp1"] = 23.5;
```

```
    sensorData["Hum1"] = 65.0;
    for (const auto& pair : sensorData) {
        std::cout << pair.first << ": " << pair.second << std::endl;
    }
    return 0;
}
```

4.2 演算法

- 範例：排序與查找

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
int main() {
    std::vector<int> values = {5, 2, 9, 1, 5, 6};
    std::sort(values.begin(), values.end());
    std::cout << "Sorted: ";
    for (int v : values) {
        std::cout << v << " ";
    }
    std::cout << std::endl;
    auto it = std::find(values.begin(), values.end(), 5);
    if (it != values.end()) {
```

```
            std::cout << "Found 5 at position: " << (it - values.begin()) <<
std::endl;
        }
        return 0;
    }
    ```

---

5. 樹莓派 GPIO 控制

GPIO 是樹莓派的核心功能，C++ 搭配 WiringPi 可輕鬆控制硬體。

5.1 控制 LED

- **硬體連線**：
    - LED 正極（長腳）連至 GPIO 18（引腳 12）。
    - 負極（短腳）透過 220Ω 電阻連至 GND（引腳 6）。
- **程式碼**：
    ```cpp
    #include <wiringPi.h>
    #include <iostream>
    int main() {
        if (wiringPiSetupGpio() == -1) {
            std::cerr << "WiringPi setup failed" << std::endl;
            return 1;
```

```
        }
        int ledPin = 18;
        pinMode(ledPin, OUTPUT);
        for (int i = 0; i < 5; i++) {
            digitalWrite(ledPin, HIGH);
            std::cout << "LED ON" << std::endl;
            delay(1000);
            digitalWrite(ledPin, LOW);
            std::cout << "LED OFF" << std::endl;
            delay(1000);
        }
        return 0;
    }
```

- **編譯與執行**：

```bash
g++ -o led led.cpp -lwiringPi
sudo ./led
```

- **說明**：LED 每秒閃爍 5 次，`sudo` 因 GPIO 需要 root 權限。

5.2 按鈕輸入

- **硬體連線**：
  - 按鈕一端連至 GPIO 17（引腳 11），另一端連至 3.3V（引腳 1）。

- 使用 10kΩ 電阻從 GPIO 17 連至 GND（下拉電阻）。

- **程式碼**：

```cpp
#include <wiringPi.h>
#include <iostream>
int main() {
    if (wiringPiSetupGpio() == -1) {
        std::cerr << "WiringPi setup failed" << std::endl;
        return 1;
    }
    int buttonPin = 17;
    int ledPin = 18;
    pinMode(buttonPin, INPUT);
    pinMode(ledPin, OUTPUT);
    pullUpDnControl(buttonPin, PUD_DOWN);
    while (true) {
        if (digitalRead(buttonPin) == HIGH) {
            digitalWrite(ledPin, HIGH);
            std::cout << "Button pressed, LED ON" << std::endl;
        } else {
            digitalWrite(ledPin, LOW);
            std::cout << "Button released, LED OFF" << std::endl;
        }
        delay(100);
```

```
    }
    return 0;
}
```

- **說明**：按下按鈕時 LED 點亮，鬆開則熄滅。

---

6. 進階應用

以下介紹三個進階應用，展示 C++ 在樹莓派上的強大功能。

6.1 多執行緒程式設計

C++11 引入 `<thread>`，適合並行處理。

- 範例：模擬兩個感測器同時讀取

```cpp
#include <iostream>
#include <thread>
#include <chrono>
void readTemperature(int id) {
    for (int i = 0; i < 3; i++) {
        std::cout << "Sensor " << id << ": Temp = " << (22.0 + i * 0.5) << "°C" << std::endl;
        std::this_thread::sleep_for(std::chrono::milliseconds(1000));
    }
```

```
}
void readHumidity(int id) {
    for (int i = 0; i < 3; i++) {
        std::cout << "Sensor " << id << ": Humidity = " << (60.0 + i * 2.0) << "%"
<< std::endl;
        std::this_thread::sleep_for(std::chrono::milliseconds(1500));
    }
}
int main() {
    std::thread t1(readTemperature, 1);
    std::thread t2(readHumidity, 2);
    t1.join();
    t2.join();
    std::cout << "All sensors done." << std::endl;
    return 0;
}
```

- **編譯**：

```bash
g++ -o threads threads.cpp -pthread
```

- **說明**：兩個執行緒同時模擬感測器讀取，展示並行處理。

6.2 感測器數據讀取（DHT11）

DHT11 是常用的溫溼度感測器。

- **硬體連線**：
  - DHT11 VCC 連至 3.3V（引腳 1）。
  - GND 連至 GND（引腳 6）。
  - DATA 連至 GPIO 4（引腳 7），並用 10kΩ 電阻上拉至 3.3V。
- **程式碼**（需第三方 DHT11 函式庫）：

```cpp
#include <wiringPi.h>
#include <dht11.h>
#include <iostream>
int main() {
    if (wiringPiSetupGpio() == -1) {
        std::cerr << "WiringPi setup failed" << std::endl;
        return 1;
    }
    DHT11 dht;
    dht.pin = 4; // GPIO 4
    while (true) {
        int result = dht.read();
        if (result == DHT11::OK) {
            std::cout << "Temperature: " << dht.temperature << "°C, Humidity: " << dht.humidity << "%" << std::endl;
        } else {
            std::cout << "Read error: " << result << std::endl;
```

```
            }
            delay(2000);
        }
        return 0;
    }
```

- **安裝 DHT11 函式庫**：從 GitHub 下載（如 `wiringPiDHT11`），並鏈結編譯。
- **說明**：每 2 秒讀取溫溼度數據。

6.3 網路程式設計（TCP 客戶端/伺服器）

樹莓派可用於簡單的網路應用，如遠端控制。

- **伺服器程式碼**（運行於樹莓派）：

```cpp
#include <iostream>
#include <string>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
int main() {
    int server_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (server_fd == -1) {
        std::cerr << "Socket creation failed" << std::endl;
        return 1;
```

```
}
sockaddr_in server_addr;
server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = INADDR_ANY;
server_addr.sin_port = htons(8080);
if (bind(server_fd, (sockaddr*)&server_addr, sizeof(server_addr)) < 0) {
    std::cerr << "Bind failed" << std::endl;
    return 1;
}
if (listen(server_fd, 3) < 0) {
    std::cerr << "Listen failed" << std::endl;
    return 1;
}
std::cout << "Server listening on port 8080..." << std::endl;
sockaddr_in client_addr;
socklen_t client_len = sizeof(client_addr);
int client_fd = accept(server_fd, (sockaddr*)&client_addr, &client_len);
if (client_fd < 0) {
    std::cerr << "Accept failed" << std::endl;
    return 1;
}
char buffer[1024] = {0};
read(client_fd, buffer, 1024);
std::cout << "Received: " << buffer << std::endl;
```

```
        std::string response = "Hello from Raspberry Pi!";
        write(client_fd, response.c_str(), response.length());
        close(client_fd);
        close(server_fd);
        return 0;
    }
    ```
- **客戶端程式碼**（可運行於另一台電腦）：
    ```cpp
    #include <iostream>
    #include <string>
    #include <sys/socket.h>
    #include <arpa/inet.h>
    #include <unistd.h>
    int main() {
        int sock = socket(AF_INET, SOCK_STREAM, 0);
        if (sock == -1) {
            std::cerr << "Socket creation failed" << std::endl;
            return 1;
        }
        sockaddr_in server_addr;
        server_addr.sin_family = AF_INET;
        server_addr.sin_port = htons(8080);
        if (inet_pton(AF_INET, "樹莓派 IP 地址", &server_addr.sin_addr) <= 0) {
```

```
        std::cerr << "Invalid address" << std::endl;
        return 1;
    }
    if (connect(sock, (sockaddr*)&server_addr, sizeof(server_addr)) < 0) {
        std::cerr << "Connection failed" << std::endl;
        return 1;
    }
    std::string message = "Hello, Pi Server!";
    send(sock, message.c_str(), message.length(), 0);
    char buffer[1024] = {0};
    read(sock, buffer, 1024);
    std::cout << "Server response: " << buffer << std::endl;
    close(sock);
    return 0;
}
```

- **編譯與執行**：

  ```bash
  g++ -o server server.cpp
  g++ -o client client.cpp
  sudo ./server
  ```

  客戶端在另一台電腦上執行，需替換「樹莓派 IP 地址」為實際 IP（如 `192.168.1.100`）。

- **說明**：伺服器監聽 8080 端口，接收客戶端消息並回應。

---

7. 除錯與最佳化

- **除錯工具**：
  - 使用 GDB：

    ```bash
    g++ -g program.cpp -o program
    gdb ./program
    ```

    在 GDB 中使用 `break`、`run`、`next` 等命令。
  - Valgrind（檢查記憶體洩漏）：

    ```bash
    sudo apt install valgrind
    valgrind ./program
    ```

- **最佳化技巧**：
  - 使用 `-O2` 或 `-O3` 編譯選項提高性能：

    ```bash
    g++ -O2 program.cpp -o program
    ```

  - 避免頻繁的 GPIO 操作，減少延遲。
  - 使用 `const` 與引用（`&`）減少數據複製。

---

8. 學習資源與建議

- **書籍**：
  - 《C++ Primer 5th Edition》：全面的 C++ 入門。
  - 《Effective C++》：進階技巧與最佳實踐。
- **線上資源**：
  - 樹莓派官方文件（raspberrypi.org）：GPIO 與硬體控制。
  - C++ Reference（cppreference.com）：語法與 STL 查詢。
  - 艾鍗學院、Coursera：C++ 與嵌入式課程。
- **專案建議**：
  - 智慧家居系統：整合感測器與網路控制。
  - 機器人控制：使用 GPIO 控制馬達與攝影機。
  - 資料記錄器：記錄感測器數據至檔案或雲端。

---

9. 結論

本教學從基礎到進階，涵蓋了在樹莓派上使用 C++ 的關鍵內容。從環境設定到 GPIO 控制，再到多執行緒與網路程式設計，您可以根據需求選擇學習路徑。C++ 的高效能與靈活性使其成為樹莓派開發的理想選擇，特別適合嵌入式系統與物聯網應用。

如果您有特定專案（如影像處理、資料庫整合）或需要更深入的範例，請提供

詳細需求，我將進一步客製化內容！