

# Bài 50: CON TRỎ VOID (VOID POINTERS)

Xem bài học trên website để ủng hộ Kteam: [Con trỏ void \(Void pointers\)](#)

Mọi vấn đề về lỗi website làm ảnh hưởng đến bạn hoặc thắc mắc, mong muốn khóa học mới, nhằm hỗ trợ cải thiện Website. Các bạn vui lòng phản hồi đến Fanpage [How Kteam](#) nhé!

## Dẫn nhập

Ở bài học trước, bạn đã được giới thiệu về [BIẾN THAM CHIẾU](#) (Reference variables) trong C++.

Hôm nay, chúng ta sẽ tiếp tục tìm hiểu về những kiến thức liên quan đến con trỏ trong C++, cụ thể là **Con trỏ void (Void pointers)**.

---

## Nội dung

Để đọc hiểu bài này tốt nhất các bạn nên có kiến thức cơ bản về:

- [CON TRỎ TRONG C++](#) (Pointer)

Trong bài ta sẽ cùng tìm hiểu các vấn đề:

- Con trỏ void (void pointer)
- Ứng dụng con trỏ void
- Hạn chế sử dụng con trỏ void

---

## Con trỏ void (void pointer)

Bạn nên luôn luôn ghi nhớ rằng một con trỏ truyền tải **hai thông tin**:

- **Kiểu dữ liệu trỏ đến** (int, double, ...) quy định cách truy xuất dữ liệu của vùng nhớ.
- **Địa chỉ của vùng nhớ** mà nó trỏ tới quy định cụ thể nơi bạn có thể truy xuất dữ liệu.

Kiểu dữ liệu đang được trỏ đến là **kiểu của con trỏ** (int \*, double \*, ...), trong khi **địa chỉ của dữ liệu** là giá trị thực tế **chứa trong biến con trỏ**.

Vì vậy, một **con trỏ void (void \*)** là một con trỏ **không chỉ định thông tin kiểu dữ liệu**. Nó cho bạn biết dữ liệu được lưu ở đâu, nhưng nó không cho bạn biết cách truy xuất dữ liệu đó.

**Con trỏ void** còn được gọi là **con trỏ tổng quát**, là một kiểu con trỏ đặc biệt có thể trỏ đến các đối tượng của **bất kỳ kiểu dữ liệu nào!**

Con trỏ void được khai báo giống như một con trỏ bình thường, sử dụng từ khóa **void** làm kiểu con trỏ:

```
void *ptr; // ptr là con trỏ void
```

Con trỏ void có thể trỏ đến các đối tượng của bất kỳ kiểu dữ liệu nào:

```
int n;  
float f;  
double d;  
  
void *ptr;  
ptr = &n; // ok  
ptr = &f; // ok  
ptr = &d; // ok
```

Tuy nhiên, con trỏ void **không xác định được kiểu dữ liệu của vùng nhớ mà nó trỏ tới**, chúng ta **không thể truy xuất trực tiếp** nội dung thông qua toán tử **dereference (\*)** được. Vì vậy, con trỏ kiểu void cần phải được ép kiểu một cách rõ ràng sang con trỏ có kiểu dữ liệu khác trước khi sử dụng toán tử **dereference (\*)**.

```
int value = 10;
void *voidPtr = &value;

// cout << *voidPtr << endl; // lỗi, không thể dereference con trỏ void

int *intPtr = static_cast<int*>(voidPtr); // ép kiểu thành con trỏ int
cout << *intPtr << endl; // ok
```

Trong đoạn code trên, **voidPtr** và **intPtr** đều trỏ vào địa chỉ của biến **value**, nhưng chúng ta chỉ có thể sử dụng toán tử **dereference (\*)** lên con trỏ **intPtr** chứ không thể sử dụng cho con trỏ **voidPtr**, vì trình biên dịch không thể biết con trỏ **voidPtr** trỏ đến kiểu dữ liệu gì.

## Ứng dụng con trỏ void

Nếu một con trỏ void **không xác định được kiểu dữ liệu của vùng nhớ mà nó trỏ tới**, làm thế nào để chúng ta biết kiểu dữ liệu mà nó trỏ tới là gì (int, double, ...).

Thông thường, để sử dụng được con trỏ void, bạn phải lấy thông tin về kiểu con trỏ theo cách khác (sử dụng thêm **tham số** hoặc **biến** cho biết kiểu con trỏ), sau đó ép kiểu con trỏ đó đến một kiểu cụ thể (int, double, ...) và sau đó sử dụng nó như bình thường.

```
#include <iostream>
using namespace std;

enum Type
{
    INT,
    DOUBLE
};

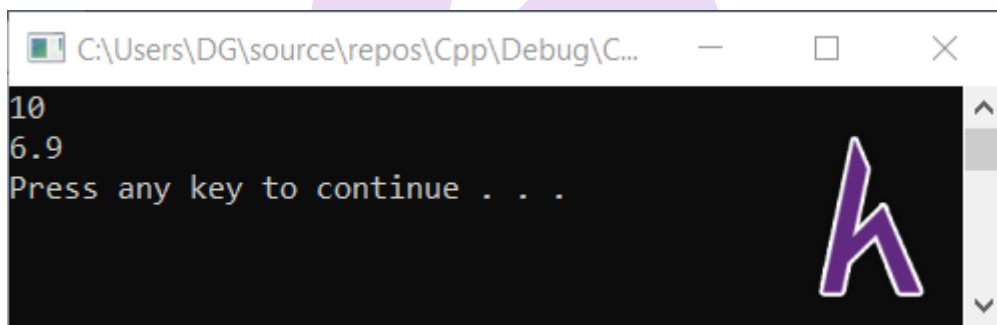
void printValueOfPointer(void *ptr, Type type)
{
    switch (type)
    {
        case INT:
```

```
        cout << *(static_cast<int*>(ptr)) << '\n'; // ép con trỏ void thành
con trỏ int
        break;
    case DOUBLE:
        cout << *(static_cast<double*>(ptr)) << '\n'; // ép con trỏ void
thành con trỏ double
        break;
    }
}

int main()
{
    int nValue = 10;
    double dValue = 6.9;

    printValueOfPointer(&nValue, INT);
    printValueOfPointer(&dValue, DOUBLE);

    system("pause");
    return 0;
}
```



Đoạn code trên hiển thị thông tin của con trỏ **nValue** và **dValue**. Thông thường, chúng ta sẽ phải viết **2 hàm** hiển thị cho 2 loại con trỏ này. Tuy nhiên, chúng ta có thể sử dụng tham số kiểu **void\***.

Khi vào hàm **printValueOfPointer()**, dựa vào biến **type** để ép từ kiểu **void\*** về kiểu dữ liệu tương ứng. Điều này làm **giảm thiểu việc viết nhiều function cùng chức năng** nhưng khác kiểu dữ liệu đầu vào.

# Hạn chế sử dụng con trỏ void

Thông thường, bạn nên **tránh sử dụng con trỏ void** trừ khi thật cần thiết. Vì chúng cho phép bạn tránh kiểm tra kiểu, điều này cho phép bạn vô tình làm những việc không có ý nghĩa, và trình biên dịch sẽ không thông báo lỗi.

Mặc dù hàm **printValueOfPointer()** là một cách ngắn gọn để xây dựng một hàm duy nhất xử lý nhiều kiểu dữ liệu. Tuy nhiên, C++ đã cung cấp một cách tốt hơn để làm điều tương tự thông qua  **nạp chồng hàm (function overloading)**, hoặc **khung mẫu hàm (function template)**. Những khái niệm này sẽ được giới thiệu trong những bài học sau.

---

## Kết luận

Qua bài học này, bạn đã nắm được Con trỏ void (Void pointers) trong C++. Lưu ý rằng bạn nên hạn chế lạm dụng nó, trừ khi không còn cách giải quyết nào khác.

Trong bài tiếp theo, mình sẽ giới thiệu cho các bạn khái niệm CON TRỎ TRỞ ĐẾN CON TRỎ (Pointers to pointers) trong C++.

Cảm ơn các bạn đã theo dõi bài viết. Hãy để lại bình luận hoặc góp ý của mình để phát triển bài viết tốt hơn. Đừng quên **"Luyện tập – Thử thách – Không ngại khó"**.