Bài 7: Struct - Union

# Struct

Trong ngôn ngữ lập trình C, struct là một cấu trúc dữ liệu cho phép lập trình viên tự định nghĩa một kiểu dữ liệu mới bằng cách nhóm các biến có các kiểu dữ liệu khác nhau lại với nhau. struct cho phép tạo ra một thực thể dữ liệu lớn hơn và có tổ chức hơn từ các thành viên (members) của nó.

Cú pháp:

| **struct** **TenStruct** {  kieuDuLieu1 thanhVien1;  kieuDuLieu2 thanhVien2;  // ... }; |
| --- |

Ví dụ:

| **struct** **Point** {  **int** x;  **int** y; }; |
| --- |

1. Thành Viên (Member): Mỗi biến trong struct được gọi là một thành viên hoặc trường. Trong ví dụ struct Point, x và y là các thành viên của Point.
2. Khởi Tạo và Truy Cập Thành Viên:

Để khởi tạo một biến thuộc kiểu struct, bạn sử dụng cú pháp như sau:

| **struct** **Point** **p1** = {10, 20}; |
| --- |

Để truy cập các thành viên, bạn sử dụng toán tử ( . ):

| printf("X coordinate: %d\n", p1.x); |
| --- |

1. Sử Dụng Tham Số Trong Hàm:

Bạn có thể truyền một biến thuộc kiểu struct như một tham số cho một hàm.

| **void** **printPoint**(struct Point p) {  printf("(%d, %d)\n", p.x, p.y); } |
| --- |

1. Truyền Con Trỏ đến Struct:

Bạn có thể truyền con trỏ đến struct như một tham số cho một hàm, cho phép thay đổi giá trị của struct bên trong hàm.

| **void** **updatePoint**(struct Point\* p, **int** newX, **int** newY) {  p->x = newX;  p->y = newY; } |
| --- |

1. Sử Dụng typedef để Tạo Bí Danh:

Bạn có thể sử dụng typedef để tạo bí danh cho struct, giúp rút ngắn cú pháp khi khai báo biến.

| **typedef** **struct** **Point** {  **int** x;  **int** y; } Point; |
| --- |

Sau đó, bạn có thể khai báo biến như sau:

| Point p1 = {10, 20}; |
| --- |

struct giúp tổ chức dữ liệu một cách hiệu quả và làm cho mã nguồn trở nên dễ đọc và bảo trì hơn.

# Union

Trong ngôn ngữ lập trình C, union là một cấu trúc dữ liệu giúp lập trình viên kết hợp nhiều kiểu dữ liệu khác nhau vào một cùng một vùng nhớ. Mục đích chính của union là tiết kiệm bộ nhớ bằng cách chia sẻ cùng một vùng nhớ cho các thành viên của nó. Điều này có nghĩa là, trong một thời điểm, chỉ một thành viên của union có thể được sử dụng.

Cú pháp định nghĩa union trong C như sau:

| **union** TenUnion {  kieuDuLieu1 thanhVien1;  kieuDuLieu2 thanhVien2;  // ... }; |
| --- |

Ví dụ:

| **union** Data {  **int** i;  **float** f;  **char** str[20]; }; |
| --- |

Trong đoạn mã trên, union Data định nghĩa một kiểu dữ liệu mới có ba thành viên là i (kiểu int), f (kiểu float), và str (mảng kiểu char).

Các khái niệm chính của union trong C bao gồm:

1. Tiết kiệm bộ nhớ:

union tiết kiệm bộ nhớ bằng cách chia sẻ cùng một vùng nhớ cho các thành viên. Kích thước của union là kích thước của thành viên lớn nhất.

1. Truy cập thành viên:

Bạn truy cập các thành viên của union bằng cách sử dụng toán tử . như struct.

| **union** Data data; data.i = 10; printf("Value of i: %d\n", data.i); |
| --- |

1. Sử dụng trong các tình huống đặc biệt

union thường được sử dụng khi bạn có một biến có thể chứa một trong các kiểu dữ liệu khác nhau và bạn chỉ cần sử dụng một kiểu dữ liệu tại một thời điểm.

| **union** Value {  **int** intValue;  **float** floatValue;  **char** stringValue[20]; }; |
| --- |

1. Biểu diễn dữ liệu tương đồng

Khi bạn có các dữ liệu có ý nghĩa tương đồng nhưng có kiểu dữ liệu khác nhau, union giúp bạn biểu diễn chúng một cách hiệu quả.

| **union** Measurement {  **int** height;  **float** depth;  **double** distance; }; |
| --- |

1. Lưu trữ dữ liệu tạm thời

union có thể được sử dụng để lưu trữ dữ liệu tạm thời khi bạn cần một biến có thể chứa một trong các kiểu dữ liệu khác nhau tùy thuộc vào tình huống.

| **union** TemporaryData {  **int** intValue;  **float** floatValue;  **char** stringValue[20]; }; |
| --- |

Lưu ý rằng khi sử dụng union, cần phải kiểm soát rất cẩn thận để tránh những lỗi gây ra bởi việc truy cập vào thành viên không hợp lệ, vì bạn chỉ nên đọc giá trị của thành viên bạn đã ghi vào gần đây nhất.