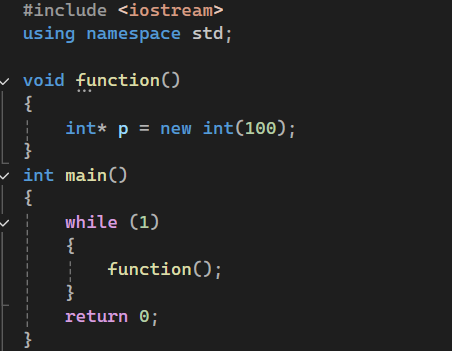
1. Phần dẫn dắt

Lý do ra đời :

Ta có ảnh chương trình sau :



Chương trình trên bị lỗi memory leak

Hàm function cấp phát vùng nhớ tại Heap, nhưng không thu hồi.

Khi chạy một khoảng thời gian, không gian Heap sẽ bị chiếm dụng hết.

Minh hoạ cụ thể :

A screenshot of a graph

Description automatically generated

Con trỏ p được lưu trong stack.

Trong khi đó, vùng nhớ int của nó trỏ tới một vùng nhớ được cấp phát trong Heap.

Khi ra khỏi scope ( phạm vi ) của hàm function thì p sẽ tự động giải phóng khỏi stack.

Nhưng vùng nhớ trong Heap đã được trỏ tới trước đó sẽ không được tự động thu hồi.

Vì vậy, người ta nói vùng nhớ này đã bị “leak”, nghĩa là chương trình không thể dùng nó được nữa.

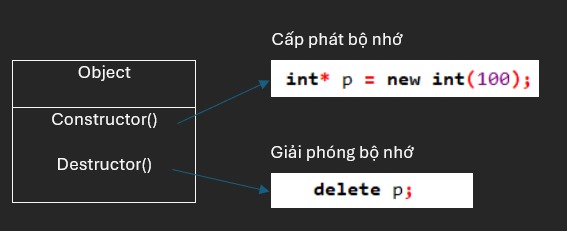
Để thu hồi vùng nhớ, ta phải sử dụng lệnh delete.

Trong các version cũ của C++, việc cấp phát và thu hồi bộ nhớ trong Heap được thực hiện thủ công.

Tuy nhiên, ta vẫn có thể tự động hoá việc cấp phát và thu hồi vùng nhớ bằng cách sử dụng object trong C++

Để tránh bị “leack”, cần thu hồi bộ nhớ sau khi không sử dụng.

Để tự động hoá việc này, ta có ý tưởng là sử dụng constructor của object để cấp phát, và destructor để thu hồi



Khi object rời khỏi scope ( phạm vi ), destructor sẽ được tự động gọi, khi đó lệnh delete cũng được gọi để giải phóng bộ nhớ.

Code thử smart pointer :

Tạo file SmartPtr.h và define class SmartPtr

Class SmartPtr này wrap một con trỏ dùng để truy cập đến bộ nhớ Heap sau này.

Tại constructor, nhận địa chỉ vùng nhớ được cấp phát thông qua 1 con trỏ và lưu vào con trỏ ptr của object.

A screen shot of a computer

Description automatically generated

Tại destructor, chúng ta giải phóng vùng nhớ nhờ con trỏ ptr của object

A computer screen with text and images

Description automatically generated

Để có thể sử dụng object smart pointer như một con trỏ thông thường, chúng ta cũng cần phải overloading toán tử dereference.

A screenshot of a computer

Description automatically generated

\*Chú ý : Khi một hàm thành viên được khai báo với const, nó cam kết với trình biên dịch rằng:

Hàm sẽ không thay đổi bất kỳ thành viên nào của đối tượng.

Điều này rất quan trọng khi bạn muốn đảm bảo tính an toàn và cho phép sử dụng đối tượng SmartPtr trong ngữ cảnh chỉ đọc.

Giờ chúng ta sẽ xem minh hoạ sử dụng smart pointer ở main.cpp :

A computer screen with text

Description automatically generated

Tại hầm f, sử dụng SmartPtr object để wrap lại con trỏ trả về từ lệnh new

Khi kết thúc hàm f, compiler sẽ tự động thu hồi biến ptr.

Vi ptr là một object của SmartPtr, destructor sẽ được gọi tự động, giúp giải phóng bộ nhớ.

1. Smart pointer là gì ?
2. Khái niệm

Smart Pointer (Con trỏ thông minh) trong C++ là một lớp (class) cung cấp các tính năng quản lý tài nguyên động như một con trỏ thông thường (raw pointer), nhưng tự động quản lý và giải phóng bộ nhớ khi tài nguyên không còn được sử dụng.

Điều này giúp tránh các vấn đề phổ biến như:

Memory leak (Rò rỉ bộ nhớ): Khi vùng nhớ trên heap được cấp phát nhưng không được giải phóng.

Dangling pointer (Con trỏ treo): Khi một con trỏ trỏ đến vùng nhớ không hợp lệ.

Quản lý tài nguyên khó khăn: Khi phải sử dụng new và delete thủ công.

Tại sao cần Smart Pointer?

Quản lý tự động: Bạn không cần gọi delete để giải phóng bộ nhớ.

An toàn: Tránh lỗi quản lý bộ nhớ thủ công như quên giải phóng hoặc giải phóng vùng nhớ nhiều lần.

Hiệu quả: Smart pointer tích hợp sẵn trong thư viện chuẩn C++ (<memory>), giúp bạn giảm công sức lập trình.

Kể từ C++ version 11, C++ cung cấp thư viện chuẩn chứa cài đặt của các smart pointer Để sử dụng, chúng ta chỉ cần include thư viện memory

1. Phân loại

Có 3 loại smart pointers được cung cấp bởi thư viện này là: unique pointer, shared pointer, và weak pointer :

1. Unique pointer :

Đặc điểm:

Chỉ có một đối tượng duy nhất sở hữu tài nguyên.

Không thể sao chép (copy) nhưng có thể chuyển quyền sở hữu bằng std::move.

Khi unique\_ptr bị hủy, tài nguyên được tự động giải phóng.

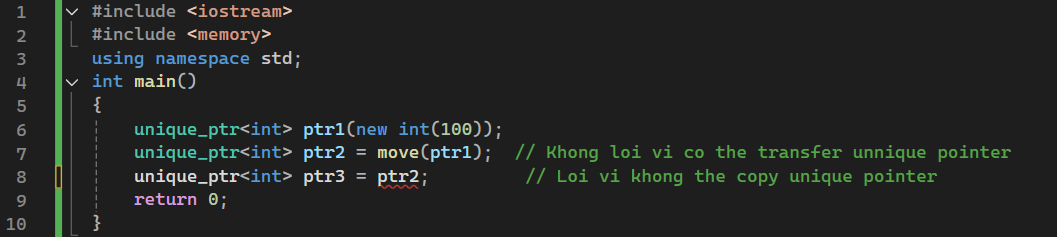
Điểm đặc biệt của Unique pointer là nó **sở hữu độc quyền** đối tượng được cấp phát động.

Điều này có nghĩa là Unique pointer ngăn chặn việc copy, nhưng vẫn có phép transfer, miễn sao chỉ có 1 unique pointer duy nhất quản lý đối tượng cấp phát động

A black screen with white text

Description automatically generated

Ở chương trình bên :



Ở dòng 5, chúng ta tạo một unique pointer ptr1 trỏ đến vùng nhớ được cấp phát kiểu int

Ở dòng 6, sử dụng hàm move để chuyển "ownership" từ ptr1 sang ptr2

Ở dòng 7, chương trình sẽ bắo lỗi, không thế copy unique pointer.

Unique pointer đảm bảo rằng chỉ có một owner (chủ sở hữu ) duy nhất tại một thời điểm.

Ứng dụng :

Dùng cho các đối tượng mà quyền sở hữu tài nguyên là duy nhất, như các đối tượng trong hàm hoặc lớp.

Để quản lý bộ nhớ động một cách hiệu quả.

1. Shared pointer :

Đặc điểm:

Cho phép nhiều con trỏ chia sẻ quyền sở hữu tài nguyên.

Sử dụng cơ chế reference counting (đếm tham chiếu):

Mỗi khi một shared\_ptr mới trỏ đến tài nguyên, bộ đếm tham chiếu tăng.

Khi một shared pointer bị hủy, bộ đếm giảm.

Khi bộ đếm về 0, tài nguyên được giải phóng.

Khác với Unique pointer, Shared Pointer cho phép nhiều pointer cùng trỏ đến đối tượng được cấp phát động (Cho phép nhiều con trỏ chia sẻ quyền sở hữu tài nguyên ).

A black background with white text and blue arrows

Description automatically generated

Điều này là nhờ shared pointer duy tri một "reference counter" để quản lý số lượng các pointer đang trỏ đến đối tượng

Khi có thêm một shared pointer trỏ đến, reference counter sẽ tăng thêm một

Khi một shared pointer bị huỷ, reference counter sẽ giảm đi một

Nếu không còn pointer nào trỏ đến, đối tượng sẽ được giải phóng.

A screen shot of a computer

Description automatically generated

Ở dòng 7, chúng ta tạo một shared pointer ptr1 trỏ đến vùng nhớ được cấp phát kiểu int

Ở dòng 9, chúng ta có thể copy ptr1 cho ptr2, lúc này ptr2 cũng trỏ đến cùng vùng nhớ đó

Ở dòng 11, hiển thị reference counter ( use\_count) là 2

Ở dòng 14, giải phóng ptr2

Ở dòng 15, hiển thị reference counter ( use\_count) lúc này đã giảm còn lại 1

Ứng dụng :

Khi bạn cần chia sẻ quyền sở hữu tài nguyên giữa nhiều đối tượng.

Dùng khi nhiều đối tượng cần dùng chung một tài nguyên, ví dụ trong các đồ thị hoặc cây.

1. Weak pointer

Đặc điểm:

Không sở hữu tài nguyên, chỉ "quan sát" tài nguyên được quản lý bởi một shared pointer.

Không làm tăng reference count.

Dùng để phá vỡ vòng tham chiếu (cyclic dependency) giữa các shared\_ptr.

Trước khi nói về weak pointer, hãy cùng xem qua vấn đề Cyclic Dependency của shared pointer

Điều này xảy ra khi chúng ta có 2 class chứa shared pointer trỏ vào nhau

A screenshot of a computer

Description automatically generated

Khi này, reference counter sẽ không bao giờ đếm về 0 được.

Điều này khiến cho đối tượng cấp phát động không được giải phóng

\*chú ý : Cyclic Dependency (vòng tham chiếu) xảy ra khi hai hoặc nhiều shared\_ptr trỏ lẫn nhau. Điều này làm cho bộ đếm tham chiếu (reference count) không bao giờ về 0, dẫn đến memory leak (rò rỉ bộ nhớ) : Giả sử bạn có hai lớp A và B, mỗi lớp đều chứa một shared\_ptr trỏ đến lớp kia

\*Nguyên nhân cụ thể : A và B sở hữu lẫn nhau:

A sở hữu ptrB, ptrB tăng reference count của B.

B sở hữu ptrA, ptrA tăng reference count của A.

Không có shared pointer nào bị reset hoàn toàn:

Khi a bị hủy, b->ptrA vẫn giữ một tham chiếu đến a.

Khi b bị hủy, a->ptrB vẫn giữ một tham chiếu đến b.

Kết quả: Vòng tham chiếu khiến cả hai đối tượng tồn tại mãi mãi, vì reference count không về 0.

Cách giải quyết :

Weak pointer được sử dụng để break cyclic dependency này.

Weak pointer, cüng tương tự như shared pointer ngoại trừ việc nó không duy trì reference counter.

Weak pointer là một loại con trỏ thông minh trong C++ không tham gia vào việc tăng reference count. Điều này giúp phá vỡ vòng tham chiếu.

Weak pointer chỉ quan sát tài nguyên được quản lý bởi shared pointer mà không sở hữu nó.

Khi shared pointer chính bị hủy, weak pointer trở thành nullptr, đảm bảo tài nguyên được giải phóng.

A black background with white text and blue arrows

Description automatically generated

Chúng ta hãy xem qua chương trình minh họa cách sử dụng weak pointer

A computer screen with text on it

Description automatically generated

Ở dòng 12, weak pointer ptr2 cùng trỏ đến vùng nhớ int như ptr1

Ở dòng 15, reference counter vẫn sẽ là 1 vì weak pointer không ảnh hưởng đến counter

Ứng dụng :

Phá vỡ vòng tham chiếu giữa các shared pointer

Giúp kiểm tra tài nguyên có còn tồn tại hay không trước khi sử dụng.

Khi bạn chỉ cần "quan sát" tài nguyên mà không sở hữu nó.

1. So sánh các pointer

|  |  |  |  |
| --- | --- | --- | --- |
| Thuộc tính | Unique pointer | Shared pointer | Weak pointer |
| Quyền sở hữu | Duy nhất | Chia sẻ | Không sở hữu |
| |  | | --- | | Reference Count |  |  | | --- | |  | | Không | Có | Không |
| Sao chép | Không  (chỉ có move ) | Có | Không |
| Phá vòng tham chiếu | Không | Không | Có |

\*Giải thích thêm về ứng dụng từng loại smart pointer :

Unique pointer :

Trong hệ thống game engine hoặc ứng dụng xử lý ảnh, một Texture (kết cấu) chỉ được sở hữu bởi một đối tượng duy nhất, ví dụ như một đối tượng Sprite. Khi Sprite bị hủy, Texture cũng cần được tự động giải phóng.

Shared pointer :

Cấu trúc cây (graph). Mỗi node có thể chứa các node con và đôi khi cũng cần liên kết đến node cha. Điều này dẫn đến vòng tham chiếu giữa node cha và các node con nếu sử dụng shared\_ptr.

Giải pháp: Sử dụng shared\_ptr để quản lý quyền sở hữu các node con, và weak\_ptr để node con giữ tham chiếu đến node cha mà không làm tăng reference count.

Weak pointer :

Phá vòng tham chiếu trong hệ thống Observer Pattern

Trong mô hình Observer Pattern, một Subject giữ danh sách các Observer, và mỗi Observer có thể giữ tham chiếu ngược lại Subject. Điều này dễ gây vòng tham chiếu nếu sử dụng shared\_ptr.