RAII

RAII

RAII (del inglés resource acquisition is initialization, que podría traducirse como «adquirir recursos es inicializar») es un popular patrón de diseño en varios lenguajes de programación orientados a objetos como C++ y Ada. La técnica fue inventada por Bjarne Stroustrup para reservar y liberar recursos en C++. En este lenguaje, después de que una excepción es lanzada, el único código fuente que con seguridad es ejecutado es el de los destructores de objetos que residen en la pila. Por lo tanto los recursos necesitan ser gestionados con objetos adecuados. Los recursos son adquiridos durante la inicialización, cuando no hay posibilidad de que sean usados antes de estar disponibles, y liberados cuando se destruyen los mismos, algo que es garantizado que suceda incluso cuando se dan errores.

RAII

La técnica RAII es vital al escribir código C++ seguro frente a excepciones: para liberar recursos antes de permitir a las excepciones que se propaguen (para evitar fugas de memoria) el desarrollador puede escribir destructores apropiados una sola vez, ahorrándose escribir código de «limpieza» duplicado y disperso por el código fuente entre bloques de manejo de excepciones que pueden ser ejecutados o no.

RAII

La adquisición de recursos es inicialización o RAII, es una técnica de programación de C++ que vincula el ciclo de vida de un recurso que debe adquirirse antes de su uso (memoria de almacenamiento asignada, subproceso de ejecución, socket abierto, archivo abierto, exclusión mutua bloqueada, espacio en disco, conexión a base de datos, cualquier cosa que exista en cantidades limitadas) hasta la vida útil de un objeto.

RAII

RAII garantiza que el recurso está disponible para cualquier función que pueda acceder al objeto (la disponibilidad del recurso es una clase invariable , lo que elimina las pruebas de tiempo de ejecución redundantes). También garantiza que todos los recursos se liberan cuando finaliza la vida útil de su objeto de control, en orden inverso al de adquisición. Del mismo modo, si falla la adquisición de recursos (el constructor sale con una excepción), todos los recursos adquiridos por cada subobjeto base y miembro completamente construido se liberan en orden inverso a la inicialización. Esto aprovecha las características principales del lenguaje (vida útil del objeto , salida del alcance , orden de inicialización y desenredado de la pila).) para eliminar las fugas de recursos y garantizar la seguridad excepcional. Otro nombre para esta técnica es Administración de recursos con límite de alcance (SBRM), después del caso de uso básico en el que finaliza la vida útil de un objeto RAII debido a la salida del alcance.

RAII

RAII se puede resumir de la siguiente manera:

- encapsular cada recurso en una clase.
- el constructor adquiere el recurso y establece todas las clases invariantes o lanza una excepción si eso no se puede hacer.
- el destructor libera el recurso y nunca lanza excepciones;
- utilice siempre el recurso a través de una instancia de una clase RAII

RAII Microsoft

A diferencia de los lenguajes administrados, C++ no tiene una recolección automática de elementos no utilizados , un proceso interno que libera memoria en montículo y otros recursos a medida que se ejecuta un programa. Un programa C++ se encarga de devolver todos los recursos adquiridos al sistema operativo. La falta de liberación de un recurso no utilizado se denomina fuga . Los recursos filtrados no están disponibles para otros programas hasta que finaliza el proceso. Las fugas de memoria en particular son una causa común de errores en la programación de estilo C++.

RAII Microsoft

El C++ moderno evita usar la memoria del monticulo tanto como sea posible declarando objetos en la pila. Cuando un recurso es demasiado grande para la pila, debe ser propiedad de un objeto. A medida que el objeto se inicializa, adquiere el recurso que posee. El objeto es entonces responsable de liberar el recurso en su destructor. El propio objeto propietario se declara en la pila. El principio de que los objetos poseen recursos también se conoce como "la adquisición de recursos es inicialización" o RAII.

RAII Microsoft

Cuando un objeto de pila propietario de recursos queda fuera del alcance, su destructor se invoca automáticamente. De esta manera, la recolección de basura en C++ está estrechamente relacionada con la vida útil del objeto y es determinista. Un recurso siempre se libera en un punto conocido del programa, que puede controlar. Solo los destructores deterministas como los de C++ pueden manejar los recursos de memoria y los que no son de memoria por igual.

Ejemplo 1

```
class widget

private:    int* data;public:

    widget(const int size) { data = new int[size]; } // acquire

    ~widget() { delete[] data; } // release

    void do_something() {}};

void functionUsingWidget() {

    widget w(1000000);  // lifetime automatically tied to enclosing scop  constructs w, including the w.data member

    w.do_something();

} // automatic destruction and deallocation for w and w.data
```

_https://replit.com/join/wbyusxknja-maximo-arielari

Ejemplo 2 RAII no implementado

```
class CGameObj{
      private:
      string name;
      public:  CGameObj(string _name);  ~CGameObj();};
CGameObj::CGameObj(string _name){
  name=_name;  cout<<"\nObjeto :"<<name<<" Construido"; }
CGameObj::~CGameObj(){ cout<<"\nObjeto :"<<name<<" Destruido"; }
```

Ejemplo 2 RAII no implementado

```
void input(){cout<<"\nInput ";}
void render(){cout<<"\nRender ";}
void update(bool& p){
     cout<<"\nUpdate";
     //throw runtime_error("Excepcion en Ejecucion");
    p=false;
}
```

Ejemplo2 RAII no implementado

```
void rungame(){
     bool playing=true;
     CGameObj* g= new CGameObj("Ariel");
     while(playing){
       input();
       update(playing);
       render(); }
    delete g; }
```

Ejemplo2 RAII no implementado

```
int main() {
  try{
    rungame(); }
  catch(runtime_error e){
    cout<<e.what()<<" Cierre de proceso";  }
  return 0;}
```

Ejemplo

Ejemplo2 RAII implementado

En C++ los recursos generados de modo estático en el ámbito de Stack son destruidos automáticamente.

Se propone el mismo ejemplo se implementa la construcción del objeto GameObjets para ser soportado desde un puntero que vive en el ámbito de stack de una función. Este objeto GameObjets sigue naciendo en el Heap de memoria pero su ciclo de vida ahora queda supeditado al ciclo de vida del stack de una función.

Ejemplo

RAII

Empleando Archivo

https://replit.com/join/lvovqydxxb-maximopereira

Ejemplo Robot

https://replit.com/join/qvbqxinqfz-maximo-arielari //Sin RAII

https://replit.com/join/qldrmacrvi-maximo-arielari //Con RAII

Bibliografía

https://en.cppreference.com/w/cpp/language/raii

https://es.wikipedia.org/wiki/RAII

https://docs.microsoft.com/en-us/cpp/cpp/object-lifetime-and-resource-management-modern-cpp?view=msvc-170