Manual de la Biblioteca PTR
DRAFT

1. Instalación de la Biblioteca PTR

1.1. Linux y MacOS-X

1. Para saber la versión del compilador GNU G++ introduzca el siguiente comando desde un terminal:

   ```
   $ g++ --version
   ```

   También el siguiente comando mostrará las versiones instaladas:

   ```
   $ ls -l /usr/include/c++
   ```

2. Comprobar si la biblioteca ya está instalada. Introduzca el siguiente comando:

   ```
   $ ls -l /usr/include/c++/*/ext/ptr.hpp
   ```

   En caso de no estar previamente instalada, el comando anterior mostrará algo similar a:

   ```
   ls: no se puede acceder a /usr/include/c++/*/ext/ptr.hpp: No existe el fichero o el directorio
   ```

   Si la biblioteca ya se encuentra instalada, mostrará algo similar a: (donde <*version*> es la versión de su compilador GNU G++)

   ```
   -rw-r--r-- 1 root root 2143 dic 31  2008 /usr/include/c++/<version>/ext/ptr.hpp
   ```

3. Si la biblioteca no está previamente instalada, o desea actualizarla, continúe con los siguientes pasos.

4. Crear el directorio `ext` con el siguiente comando: (donde <*version*> es la versión de su compilador GNU G++)

   ```
   $ sudo mkdir -p /usr/include/c++/<version>/ext
   ```

5. Copiar el archivo `ptr.hpp` que se adjunta al directorio anterior:

   ```
   $ sudo cp ptr.hpp /usr/include/c++/<version>/ext
   ```

6. Comprobar que el archivo `ptr.hpp` se ha copiado al directorio adecuado

   ```
   $ ls -l /usr/include/c++/*/ext/ptr.hpp
   ```

1.2. Windows CodeBlocks C++

1. Comprobar si la carpeta `ext` se encuentra en la siguiente carpeta: (donde <*version*> es la versión de su compilador GNU G++)

   ```
   \Equipo\C:\Archivos de Programa\CodeBlocks\MinGW\lib\gcc\mingw32\<version>\include\c++\
   ```

2. Si la carpeta `ext` no existe, entonces debe crearla en la carpeta especificada anteriormente (paso 1).

3. Copiar el archivo `ptr.hpp` que se adjunta a la carpeta creada anteriormente (en caso de que ya exista, reemplazarlo):

   ```
   \Equipo\C:\Archivos de Programa\CodeBlocks\MinGW\lib\gcc\mingw32\<version>\include\c++\ext\
   ```

4. Comprobar que el archivo `ptr.hpp` se ha copiado a la carpeta especificada anteriormente (paso 3).

1.3. En Directorio o Carpeta de Trabajo

1. Copiar el archivo `ptr.hpp` que se adjunta al directorio o carpeta de trabajo.

2. Utilización de la Biblioteca PTR

La biblioteca *ptr* proporciona un mecanismo para realizar, en *tiempo de ejecución*, algunas comprobaciones de errores comunes en la manipulación de punteros, tanto en los sistemas operativos Linux, MacOS-X y Windows.

Una vez instalada la biblioteca, para poder utilizarla nuestro programa C++ debe incluir el siguiente archivo de cabecera:

```
#include <ext/ptr.hpp>
```

En caso de que la biblioteca se haya instalado en el directorio de trabajo, nuestro programa C++ deberá incluir la biblioteca de la siguiente manera:

```
#include "ptr.hpp"
```

Esta biblioteca se encuentra definida dentro del espacio de nombres `umalcc`, sin embargo, para facilitar su utilización dicho espacio de nombres se encuentra expandido explícitamente.

2.1. Definición de Tipos Punteros

Cuando se desea definir un tipo Puntero con control en tiempo de ejecución, se debe definir de la siguiente forma, y se utiliza como cualquier otro tipo puntero:

- Definición de tipo Puntero

```
#include <ptr.hpp>
struct Nodo;
// typedef Nodo* PNodo;  // definición usual bajo comentario
typedef Ptr<Nodo> PNodo; // nueva definición controlada
struct Nodo {
    PNodo sig;
    int dato;
};
```

- Ejemplo de utilización de variables de tipo Puntero:

```
int main()
{
    PNodo ptr = new Nodo;
    ptr->sig = NULL;
    ptr->dato = 1;
    PNodo aux = ptr;
    delete ptr;
}
```

- Ejemplo de compilación:

```
g++ -ansi -Wall -Werror -g -D_GLIBCXX_DEBUG -o prueba prueba.cpp
```

- Si se define la opción de compilación `-D_GLIBCXX_DEBUG`, entonces el tipo `Ptr<>` realiza chequeos en tiempo de ejecución. Sin embargo, si no se define dicho símbolo, entonces el tipo `Ptr<>` se comporta como un tipo puntero normal.

- Se debe compilar el programa con la opción de compilación `-g` para que se pueda mostrar el numero de línea del error.

- Si se desea que **SÍ** se compruebe la asignación de punteros no inicializados o ya liberados, entonces se debe compilar el programa con la opcion de compilación `-DCOPYCHECK`. Por ejemplo:

```
int main()
{
    PNodo ptr;
    PNodo aux = ptr;
}
```

- **NOTA:** La versión actual no es adecuada para trabajar con Herencia/Polimorfismo

2.2. Errores usuales en el manejo de punteros

Dada la siguiente definición de tipo Puntero:

```
#include <ptr.hpp>
struct Nodo;
// typedef Nodo* PNodo;  // definición usual bajo comentario
typedef Ptr<Nodo> PNodo; // nueva definición controlada
struct Nodo {
    PNodo sig;
    int dato;
};
```

A continuación se muestra un listado de los posibles errores que pueden ser detectados, en caso de que ocurran, durante la ejecución del programa, así como de los mensajes de error generados. Los ejemplos mostrados son situaciones simples que ilustran dichos errores, sin embargo, los mismos errores también suceden en otras ocasiones más complejas y difíciles de detectar, que con la ayuda de esta biblioteca pueden ser detectados fácilmente.

1. Liberación de un puntero NULO.

   Mensaje: `ERROR: [delete] Liberar Ptr<Nodo> NULO.`

   ```
   int main()
   {
       PNodo ptr = NULL;
       delete ptr;
   }
   ```

2. Liberación de un puntero no inicializado.

   Mensaje: `ERROR: [delete] Liberar Ptr<Nodo> no inicializado.`

   ```
   int main()
   {
       PNodo ptr;
       delete ptr;
   }
   ```

3. Liberación de un puntero ya liberado previamente.

   Mensaje: `ERROR: [delete] Liberar Ptr<Nodo> una zona de memoria ya liberada 0x257a030.`

   ```
   int main()
   {
       PNodo ptr = new Nodo;
       delete ptr;
       delete ptr;
   }
   ```

4. Utilización de un puntero no inicializado.

   Mensaje: `ERROR: [==] Utilizacion de Ptr<Nodo> no inicializado.`

   ```
   int main()
   {
       PNodo ptr;
       if (ptr == NULL) { /* ... */ }
   }
   ```

5. Utilización de un puntero ya liberado previamente.

   Mensaje: `ERROR: [==] Utilizacion de Ptr<Nodo> ya liberado 0x20d8030.`

   ```
   int main()
   {
       PNodo ptr = new Nodo;
       delete ptr;
       if (ptr == NULL) { /* ... */ }
   }
   ```

6. Desreferenciación de un puntero NULO.

   Mensaje: `ERROR: [->] Desreferenciacion de Ptr<Nodo> NULO.`

```
int main()
{
    PNodo ptr = NULL;
    if (ptr->dato == 0) { /* ... */ }
}
```

7. Desreferenciación de un puntero no inicializado.

   Mensaje: `ERROR: [->] Desreferenciacion de Ptr<Nodo> no incializado.`

```
int main()
{
    PNodo ptr;
    if (ptr->dato == 0) { /* ... */ }
}
```

8. Desreferenciación de un puntero ya liberado previamente.

   Mensaje: `ERROR: [->] Desreferenciacion de Ptr<Nodo> ya liberado 0x1713030.`

```
int main()
{
    PNodo ptr = new Nodo;
    delete ptr;
    if (ptr->dato == 0) { /* ... */ }
}
```

9. Pérdida de la memoria apuntada, debido a que termina el ámbito del único puntero que apunta a ella.

   Mensaje: `ERROR: [{}] Se pierde la referencia al Ptr<Nodo> 0x2570030.`

```
int main()
{
    PNodo ptr = new Nodo; // este nodo se pierde
}
```

10. Pérdida de la memoria apuntada, debido a que el único puntero que apunta a ella pasa a apuntar a a otro sitio (o `NULL`).

    Mensaje: `ERROR: [=] Se pierde la referencia al Ptr<Nodo> 0x2298030.`

```
int main()
{
    PNodo ptr = new Nodo; // este nodo se pierde
    ptr = NULL;
}
```

11. Pérdida de la memoria apuntada, debido a que termina la vida (por destrucción del nodo donde reside) del único puntero que apunta a ella.

    Mensaje: `ERROR: [delete] Se pierde la referencia al Ptr<Nodo> 0xf14070.`

```
int main()
{
    PNodo ptr = new Nodo; // este nodo se libera
    ptr->sig = new Nodo;  // este nodo se pierde
    delete ptr;
}
```

12. Copia o asignación de un puntero no inicializado. Este caso, aunque no es un error propiamente dicho, podría indicar situaciones de error. No obstante, puede haber situaciones correctas donde sí podría suceder, por ejemplo al copiar arrays o vectores con punteros. Por este motivo, la biblioteca, por defecto, no comprueba esta situación. Sin embargo, en caso de desear **activar** la comprobación de esta característica (Copia o asignación de un puntero no inicializado), entonces se puede activar su control utilizando explícitamente la siguiente opción de compilación: `-DCOPYCHECK`.

    Mensaje: `ERROR: [=] Copia de Ptr<Nodo> no inicializado.`

```
int main()
{
    PNodo ptr;
    PNodo aux = ptr;
}
```

13. Copia o asignación de un puntero ya liberado previamente. Este caso, aunque no es un error propiamente dicho, podría indicar situaciones de error. No obstante, puede haber situaciones correctas donde sí podría suceder, por ejemplo al copiar arrays o vectores con punteros. Por este motivo, la biblioteca, por defecto, no comprueba esta situación. Sin embargo, en caso de desear **activar** la comprobación de esta característica (Copia o asignación de un puntero ya liberado previamente), entonces se puede activar su control utilizando explícitamente la siguiente opción de compilación: `-DCOPYCHECK`.

    Mensaje: `ERROR: [=] Copia de Ptr<Nodo> ya liberado 0x1843030.`

```
int main()
{
    PNodo ptr = new Nodo;
    delete ptr;
    PNodo aux = ptr;
}
```