BraiiiiiiinnnzzzZ

Este ejercicio consiste en crear una clase Zombie en C++, que simula el comportamiento de los zombis con ciertas características y funcionalidades. A continuación, se explica paso a paso lo qué debes hacer:

1. Crear la clase Zombie:

   - Debes crear una clase llamada Zombie.
   - La clase debe tener un atributo privado, que será una cadena de texto (string) llamada name. Este atributo almacenará el nombre del zombi.

2. Añadir una función miembro announce():

   - Annoounce: anunciar, notificar, avisar, informar...
   - La clase debe tener una función miembro llamada announce(), que será responsable de hacer que el zombi se presente.
   - El zombi debe anunciarse diciendo su nombre seguido del sonido característico de los zombis, que es: "BraiiiiiiinnnzzzZ...".
   - El formato del mensaje es: <name>: BraiiiiiiinnnzzzZ.... No se deben imprimir los signos de menor y mayor (< y >).

   Ejemplo: En el caso de que el zombi en cuestión se llama "Foo", el mensaje que se debe mostrar será: Foo: BraiiiiiiinnnzzzZ....

3. Implementar la función newZombie():

   - Debes crear una función llamada newZombie() que reciba un parámetro name de tipo std::string.
   - Esta función debe crear un nuevo objeto de tipo Zombie, asignarle el nombre proporcionado y devolver un puntero a este objeto.
   - El propósito de esta función es crear un zombi, darle un nombre, y permitir que el puntero al zombi sea utilizado fuera del ámbito de la función.

4. Implementar la función randomChump():

   - Debes crear una función llamada randomChump() que también reciba un parámetro name de tipo std::string.
   - Esta función debe crear un zombi, asignarle el nombre y hacer que el zombi se anuncie a sí mismo usando la función announce().
   - Sin embargo, el zombi creado dentro de randomChump() debe ser destruido de manera automática cuando termine la función (es decir, no es necesario devolver un puntero, ya que el zombi se crea y destruye en el mismo ámbito).

5. Manejo de memoria: Stack vs Heap:

- El objetivo de este proyecto es que determines cuándo es mejor usar la pila (stack) y cuándo es mejor usar el montón (heap) para asignar memoria para los zombis.
- Pila (stack): Es más rápida y la memoria se libera automáticamente cuando la función termina, pero no puedes usar objetos de larga duración que necesiten ser utilizados fuera de la función que los creó.
- Montón (heap): Permite crear objetos cuyo ciclo de vida puede ser controlado manualmente. Estos objetos deben ser liberados (destruidos) cuando ya no los necesites.
- Tendrás que decidir si es mejor crear los zombis en el stack (para objetos temporales) o en el heap (cuando necesites que los objetos existan fuera del alcance de la función).

6. Destructor de la clase Zombie:

- La clase Zombie debe tener un destructor. El propósito del destructor es destruir el objeto cuando ya no se necesita.
- Además, el destructor debe imprimir un mensaje con el nombre del zombi para fines de depuración. Esto es útil para verificar que los zombis se están eliminando correctamente.

★ Resumen de lo que debes hacer:

- ✓ Crear una clase Zombie con un atributo privado name.
- ✓ Añadir una función announce() para que el zombi se anuncie con el formato <name>: BraiiiiiiinnnzzzZ....
- ✓ Implementar una función newZombie() que cree un zombi y devuelva su puntero.
- ✓ Implementar una función randomChump() que cree un zombi, lo nombre y lo haga anunciarse.
- ✓ Decidir cuándo asignar zombis en la pila o en el montón.
- ✓ Añadir un destructor que imprima un mensaje con el nombre del zombi al ser destruido.

Este proyecto tiene como objetivo familiarizarte con el manejo de punteros, la gestión de memoria dinámica (heap) y la gestión de memoria automática (stack) en C++, así como entender cómo funciona la creación y destrucción de objetos.

Zombie.hpp

1. Directivas de Preprocesador

   ```
   #ifndef ZOMBIE_HPP
   # define ZOMBIE_HPP
   ```

   Estas líneas son parte de una directiva de preprocesador que se utiliza para evitar que el archivo de cabecera (header file) sea incluido más de una vez en el proyecto. Si el archivo ya ha sido incluido, la directiva #define ZOMBIE_HPP evita que se vuelva a incluir el archivo en la compilación, lo que previene errores de redefinición.

2. Inclusión de Archivos de Cabecera

   - <cstdlib>: Proporciona funciones de utilidad como la gestión de memoria dinámica y conversiones de tipo.
   - <iostream>: Proporciona funcionalidades para la entrada/salida estándar, como std::cout para imprimir en la consola.
   - <stdio.h>: Es una biblioteca estándar de C para la entrada/salida.
   - <string>: Proporciona la clase std::string que se usa para manejar cadenas de texto en C++.

3. Declaración de la Clase Zombie

   class Zombie{

   Aquí comienza la definición de la clase Zombie. Esta clase tiene tanto atributos como métodos, y su propósito es representar un zombi con nombre.

4. Métodos Públicos

   Dentro de la clase Zombie, hay varios métodos públicos:

   - void announce(void) const; Este método se usa para hacer que el zombi se anuncie con el formato "BraiiiiiiinnnzzzZ...". Se declara como const porque no modifica el estado del objeto.
   - const std::string& getName(void) const; Este método devuelve el nombre del zombi. El uso de const std::string& indica que la función devuelve una referencia constante al nombre del zombi para evitar la copia del valor y asegurar que no se modifique fuera de la clase.
   - void setName(std::string name); Este método establece el nombre del zombi. Recibe un std::string y lo asigna al atributo privado name.
   - ~Zombie(void); Es el destructor de la clase Zombie. Este método se ejecutará de forma automática cuando un objeto de tipo Zombie sea destruido. El propósito del destructor es liberar cualquier recurso que el objeto haya podido usar.

5. Atributos Privados

El atributo privado name es de tipo std::string y almacena el nombre del zombi. Como está en la sección privada, solo puede ser accedido y modificado a través de los métodos públicos.

6. Declaración de Funciones Fuera de la Clase

   Estas son las declaraciones de dos funciones globales (fuera de la clase Zombie):

   - Zombie* newZombie(std::string name); Esta función crea un nuevo objeto Zombie, le asigna el nombre proporcionado y devuelve un puntero al objeto Zombie creado.
   - void randomChump(std::string name); Esta función crea un zombi, le asigna el nombre y hace que el zombi se anuncie a sí mismo (llama al método announce()). En este caso, el zombi se destruye automáticamente al final de la función.

7. Fin del Archivo de Cabecera

#endif

Esta línea termina el bloque de preprocesador iniciado con #ifndef. Garantiza que todo el contenido del archivo de cabecera solo se defina una vez durante la compilación.

Resumen de lo que hace este código:

1. Clase Zombie: Define la clase con un atributo privado name y tres métodos públicos: announce() (para que el zombi se anuncie), getName() (para obtener el nombre del zombi) y setName() (para establecer el nombre del zombi). También tiene un destructor para manejar su destrucción.

2. Funciones Externas:

   - newZombie(): Crea un zombi, le asigna un nombre y devuelve un puntero a ese zombi.

   - randomChump(): Crea un zombi, lo nombra y lo hace anunciarse.

   En todo caso, este archivo es solo la declaración de la clase y las funciones. En el archivo de implementación (Zombie.cpp), se definirá el comportamiento de los métodos, como la lógica de la función announce(), el constructor y el destructor.

main.cpp

Este código en C++ se enfoca en trabajar con objetos de tipo Zombie, y realiza varias operaciones interactivas con el usuario. A continuación, te explico qué hace el código de manera clara y detallada:

1. Inclusión del archivo de encabezado "Zombie.hpp"

Aquí se incluye un archivo de encabezado llamado Zombie.hpp. Este archivo contiene la declaración de una clase Zombie que será utilizada en el código y define la estructura y las funciones asociadas a los objetos de tipo Zombie.

2. Entrada de datos desde el usuario

```
std::string data;
std::cout << " 🧟 Enter the name of the stacked zombie: ";
std::getline(std::cin, data);
```

Se declara una variable data de tipo std::string, que almacenará el nombre que el usuario ingrese. A continuación, se le pide al usuario que ingrese el nombre de un "zombie apilado" (probablemente un zombie que se maneja en la pila, es decir, con un objeto local). El programa espera a que el usuario ingrese una cadena de texto (nombre del zombie) y la almacena en data.

3. Detección de fin de archivo (EOF)

```
if (std::cin.eof() == true){
   std::cin.clear();
   clearerr(stdin);
   std::cout << "EOF detected, exiting program." << std::endl;
}
```

Esta parte verifica si se ha alcanzado el fin de archivo (EOF, por sus siglas en inglés). Esto puede ocurrir si el usuario cierra la entrada estándar (por ejemplo, presionando Ctrl+D en una terminal). Si el fin de archivo es detectado, el flujo de entrada se limpia con std::cin.clear() y clearerr(stdin), y el programa imprime un mensaje indicando que el fin de archivo fue detectado y que el programa terminará.

4. Llamada a la función randomChump()

```
randomChump(data);
```

Aquí se llama a la función randomChump, pasándole como argumento el nombre del zombie que el usuario ingresó previamente.

5. Ingreso del nombre de un zombie para el "heap"

```
std::cout << " 🧟 Enter the name of the heap zombie: ";
std::getline(std::cin, data);
```

De nuevo, se pide al usuario que ingrese el nombre de un "zombie del heap" (un zombie que se maneja dinámicamente, es decir, utilizando memoria dinámica). El nombre se almacena en la variable data.

6. Detección de fin de archivo (EOF)

Al igual que antes, el programa verifica si se ha detectado un EOF y, si es así, limpia la entrada y muestra el mensaje correspondiente.

7. Creación dinámica de un zombie

```
Zombie* other_zombie = newZombie(data);
```

Aquí se llama a la función newZombie, que crea un nuevo objeto Zombie en el montón (heap) de memoria, utilizando el nombre que el usuario ha proporcionado como parámetro. Esta función devuelve un puntero a un objeto Zombie recién creado.

8. Liberación de memoria

```
delete other_zombie;
```

Una vez que el objeto Zombie ha sido creado y utilizado, el puntero other_zombie se elimina con delete. Esto libera la memoria dinámica que fue asignada al zombie en el heap.

9. Finalización del programa

Finalmente, el programa termina y devuelve 0, lo que indica que se ejecutó correctamente.

★ En resumen:

- El programa pide al usuario ingresar dos nombres de zombies.
- El primer zombie se maneja en la pila (stack), mediante la función randomChump.
- El segundo zombie se maneja dinámicamente en el heap mediante la función newZombie.
- En ambas interacciones, el programa verifica si se ha alcanzado el fin de archivo (EOF).
- La memoria dinámica utilizada para el zombie del heap es liberada con delete.

newZombie.cpp