My Awesome PhoneBook

El ejercicio consiste en diseñar un programa que se comporte como un software de agenda telefónica "crappy awesome".

Debes implementar dos clases:

1. PhoneBook
    - Tiene un arreglo de contactos.
    - Puede almacenar un máximo de 8 contactos. Si el usuario intenta agregar un noveno contacto, reemplaza el más antiguo por el nuevo.
    - Ten en cuenta que la asignación dinámica está prohibida.

2. Contact
    - Representa un contacto de la agenda telefónica.

En tu código, la agenda telefónica debe ser instanciada como una instancia de la clase PhoneBook. Lo mismo para los contactos: cada uno de ellos debe ser instanciado como una instancia de la clase Contact. Eres libre de diseñar las clases como quieras, pero ten en cuenta que todo lo que se usará siempre dentro de una clase es privado, y que todo lo que se pueda usar fuera de una clase es público.

Al iniciar el programa, la agenda telefónica está vacía y se le solicita al usuario que ingrese uno de tres comandos. El programa solo acepta ADD, SEARCH y EXIT.

- ADD: guardar un nuevo contacto
    - Si el usuario ingresa este comando, se le solicita que ingrese la información del nuevo contacto campo por campo. Una vez que se hayan completado todos los campos, agrega el contacto a la agenda telefónica.
    - Los campos del contacto son: nombre, apellido, apodo, número de teléfono y secreto más oscuro. Un contacto guardado no puede tener campos vacíos.

- SEARCH: mostrar un contacto específico
    - Muestra los contactos guardados como una lista de 4 columnas: índice, nombre, apellido y apodo.
    - Cada columna debe tener 10 caracteres de ancho. Un carácter de pipe (’|’) los separa. El texto debe estar alineado a la derecha. Si el texto es más largo que la columna, debe ser truncado y el último carácter visible debe ser reemplazado por un punto (’.’).
    - Luego, solicite al usuario nuevamente el índice de la entrada a mostrar. Si el índice está fuera de rango o es incorrecto, defina un comportamiento relevante. De lo contrario, muestre la información de contacto, un campo por línea.

- EXIT
    - ¡El programa se cierra y los contactos se pierden para siempre!

Cualquier otra entrada se descarta.

Una vez que se haya ejecutado correctamente un comando, el programa espera otro. Se detiene cuando el usuario ingresa EXIT.

Dale un nombre relevante a tu ejecutable.

★ Directorio: ex01
★ Ficheros que incluir en el directorio: Makefile, *.cpp, *.{h, hpp}
★ Funciones prohibidas: Ninguna

DEFINICIONES

¿Qué significa "instanciar"?

Imagina que tienes un molde para hacer galletas. Ese molde es como una clase en programación. Cada galleta que sacas del molde es una instancia de esa clase, es decir, un objeto con las mismas características definidas por el molde.

En programación, "instanciar" significa crear un objeto a partir de una clase. Es como tomar ese molde y crear una galleta concreta.

¿Qué es una clase?

Una clase es como un plano o una receta que define las características y comportamientos de un objeto. En el caso de nuestra agenda telefónica, la clase PhoneBook define qué es una agenda y qué puede hacer (por ejemplo, almacenar contactos, buscarlos, etc.).

¿Y por qué se dice que la agenda telefónica debe ser instanciada como una instancia de la clase PhoneBook?

Esto significa que cuando queramos usar nuestra agenda telefónica en nuestro programa, tendremos que crear un objeto específico de tipo PhoneBook. Es como decir: "Quiero una agenda telefónica, así que voy a crear una a partir del molde PhoneBook".

Ejemplo:

```
// Creando una instancia de la clase PhoneBook
PhoneBook miAgenda = new PhoneBook();
```

En esta línea de código, estamos:

- Declarando una variable: miAgenda será el nombre de nuestra agenda.
- Instanciando un objeto: new PhoneBook() crea un nuevo objeto a partir de la clase PhoneBook y lo asigna a la variable miAgenda.

¿Por qué es importante?

Al instanciar un objeto, estamos creando una copia concreta de esa clase. Esto nos permite tener múltiples agendas telefónicas, cada una con sus propios contactos y características. Si tuviéramos todo en una sola clase, sería como tener un solo molde para todas las galletas y no podríamos personalizarlas.

En resumen:

Cuando se dice que la agenda telefónica debe ser instanciada como una instancia de la clase PhoneBook, simplemente significa que, para usar nuestra agenda en el programa, debemos crear un objeto específico de ese tipo. Es como tomar un molde y crear un objeto real a partir de él.

Lo privado y lo público en una clase

Imagina una caja fuerte. Dentro de ella guardas tus objetos más valiosos, a los que solo tú puedes acceder. En programación, esta caja fuerte es como una clase, y los objetos que guarda son sus miembros.

En C++, podemos dividir los miembros de una clase en dos categorías principales:

- Miembros privados: Son como los objetos dentro de la caja fuerte. Solo pueden ser accedidos y modificados desde dentro de la misma clase. Esto significa que ningún código externo a la clase puede ver o cambiar estos miembros directamente.

- Miembros públicos: Son como la ranura de una caja fuerte. Puedes insertar o sacar objetos a través de esta ranura, pero no puedes ver directamente lo que hay dentro. En programación, los miembros públicos son los que otros objetos pueden usar o llamar.

¿Por qué usamos lo privado y lo público?

- Encapsulación: Al ocultar los detalles internos de una clase (miembros privados), estamos encapsulando la información. Esto hace que nuestro código sea más seguro y fácil de mantener. Si cambiamos algo dentro de la clase, no tenemos que preocuparnos de que afecte a otras partes del programa.

- Reutilización: Los miembros públicos son la interfaz de nuestra clase. Al definir una interfaz clara y concisa, podemos reutilizar esa clase en diferentes partes de nuestro programa sin tener que preocuparnos por los detalles internos.

  Ejemplo:

```
class Persona {
private:
  std::string nombre;
  int edad;

public:
  void setNombre(std::string nuevoNombre) {
    nombre = nuevoNombre;
  }

  std::string getNombre() {
    return nombre;
  }
};
```

En este ejemplo:

- nombre y edad son miembros privados. Solo la clase Persona puede acceder a ellos.
- setNombre y getNombre son miembros públicos. Otros objetos pueden llamar a estos métodos para establecer o obtener el nombre de una persona.

¿Cómo funciona en el contexto de nuestra agenda telefónica?

En nuestra clase PhoneBook, podríamos tener miembros privados como:

- Un arreglo para almacenar los contactos.
- Un contador para llevar la cuenta de cuántos contactos hay.

Y miembros públicos como:

- Un método para agregar un nuevo contacto.
- Un método para buscar un contacto.
- Un método para eliminar un contacto.

Al hacer esto, estamos encapsulando la lógica interna de cómo se almacena y gestiona la información de los contactos, mientras que exponemos una interfaz fácil de usar para otros objetos que quieran interactuar con la agenda.

En resumen:

- Miembros privados: Detalles internos de la clase, solo accesibles desde dentro de la clase.
- Miembros públicos: Interfaz de la clase, accesible desde fuera de la clase.
- Encapsulación: Ocultar los detalles internos para mejorar la seguridad y la mantenibilidad.
- Reutilización: Definir una interfaz clara para facilitar la reutilización de la clase.

EL CÓDIGO

phonebook.hpp

Este archivo de encabezado (phonebook.hpp) define dos clases: Contact y PhoneBook.

Veamos ahora cada parte del código:

```
#ifndef PHONEBOOK_HPP
# define PHONEBOOK_HPP

# include <iostream>
# include <iomanip>
# include <string>
```

1. Guardas de inclusión: Estas líneas aseguran que el archivo solo se incluya una vez para evitar problemas de redefinición.

   Verifica si PHONEBOOK_HPP no está definido. Si no lo está entonces se define.

   #endif al final del archivo cierra la condición.

2. Inclusión de bibliotecas: Las siguientes líneas incluyen las bibliotecas estándar necesarias para el código.

   - <iostream> para la entrada y salida estándar.
   - <iomanip> para la manipulación de la entrada y salida.
   - <string> para el uso de la clase std::string.

3. Clase Contact: Esta clase representa un contacto en la agenda telefónica. Consta de:

   - Atributos públicos: name (nombre del contacto), Last_name(apellido del contacto), Nick_name(apodo del contacto), secret (un secreto inconfesable del contacto), y Phone (Número de teléfono del contacto).

   - Método display: Este método imprime el nombre y el número de teléfono del contacto en la consola.

4. Clase PhoneBook

   Esta clase representa una agenda telefónica, que almacena los contactos. En ella tenemos los elementos con atributos públicos: Inputs (un arreglo de ocho objetos que almacena los contactos) y num (que se usa para llevar la cuenta de los contactos almacenados).

En resumen, este archivo define una estructura básica para una agenda telefónica con contactos, incluyendo métodos para mostrar la información de los contactos.

phonebook.cpp

1. Inclusión del archivo de cabecera:

   #include "phonebook.hpp": Esta línea incluye el archivo de cabecera "phonebook.hpp", que, como ya hemos visto, contiene la definición de la clase PhoneBook y la estructura de datos para almacenar los contactos.

2. Función putst():

   Esta función se encarga de imprimir una cadena de texto en una columna de 10 caracteres, alineada. Si la cadena es más larga que 10 caracteres, se trunca a 9 caracteres y se agrega un punto al final

   La clase **std::string** es parte de la biblioteca estándar de C++ y se utiliza para manejar cadenas de caracteres de manera más segura y conveniente que los arreglos de caracteres (char[]).

   La línea `s**td::cout** << "|";` imprime un carácter de barra vertical (`|`) en la consola. `std::cout` es el flujo de salida estándar en C++, y el operador `<<` se utiliza para insertar el carácter `|` en el flujo de salida, lo que resulta en que se muestre en la consola. Su equivalente en C seria printf.

   La expresión **tmp.size**() > 10 en C++ verifica si el tamaño de la cadena tmp es mayor que 10.

   La línea **tmp.resize**(9) en C++ ajusta el tamaño de la cadena tmp a 9 caracteres. Si la cadena original tiene más de 9 caracteres, se truncará a los primeros 9 caracteres. Si la cadena original tiene menos de 9 caracteres, se rellenará con caracteres nulos (\0) hasta alcanzar el tamaño de 9 caracteres. El equivalente en C para truncar una cadena a 9 caracteres se puede lograr utilizando la función strncpy.

   La función **std::setw**(10) en C++ se utiliza para establecer el ancho de campo de la siguiente salida en 10 caracteres. Esto significa que el siguiente valor que se imprima se alineará a la derecha y ocupará al menos 10 caracteres de ancho. Si el valor tiene menos de 10 caracteres, se rellenará con espacios en blanco a la izquierda.

3. Función main:

```
PhoneBook myPhoneBook;
std::string command;
int selector;
myPhoneBook.num = 0;
PhoneBook myPhoneBook;
```

   La primera línea declara una variable llamada myPhoneBook de tipo PhoneBook. PhoneBook es una clase definida en phonebook.hpp. En resumen: esta línea crea una instancia (una “copia”) de PhoneBook llamada myPhoneBook.

```
std::string command;
```

Esta línea declara una variable llamada command de tipo std::string. Esta variable se utilizará para almacenar comandos de entrada del usuario.

```
int selector;
```

Esta línea declara una variable llamada selector de tipo int (entero). Esta variable se utilizará para almacenar un índice del usuario.

```
myPhoneBook.num = 0;
```

Esta línea inicializa el miembro num de la instancia myPhoneBook a 0. num es un miembro de la clase o estructura PhoneBook , que se utiliza para llevar un conteo del número de contactos almacenados en el PhoneBook.

```
while (1)
```

A partir de aquí, se inicia un bucle while infinito. El 1 (o true) significa que la condición del bucle siempre es verdadera, por lo que el bucle continuará ejecutándose indefinidamente hasta que se encuentre una instrucción break o se termine el programa.

En la cuarta línea nos encontramos con un nuevo concepto:

std::getline(std::cin, command);

Esta línea de código realiza las siguientes acciones:

std::getline es una función de la biblioteca estándar de C++ que se utiliza para leer una línea completa de entrada desde un flujo de entrada. La función toma dos argumentos: el flujo de entrada desde el cual leer y una variable de tipo std::string donde almacenar la línea leída.

std::cin es el flujo de entrada estándar en C++, que generalmente se asocia con la entrada del teclado. En este contexto, std::cin se utiliza como el flujo de entrada desde el cual std::getline leerá la línea.

command es una variable de tipo std::string que ya habíamos declarado previamente en el código. En esta variable, se almacenará la línea completa de entrada que el usuario ingresa (std::getline) por teclado (std::cin).

En conjunto, std::getline(std::cin, command); hace lo siguiente:

- Espera a que el usuario ingrese una línea de texto y presione la tecla Enter.
- Lee toda la línea de texto ingresada por el usuario, incluyendo cualquier espacio en blanco.
- Almacena la línea leída en la variable command.

if (command == "ADD")

Este bloque de código maneja la adición de un nuevo contacto al PhoneBook. Si el PhoneBook está lleno, desplaza los contactos existentes para hacer espacio para el nuevo contacto. Luego, solicita al usuario que ingrese los detalles del nuevo contacto y los almacena en el primer lugar del array Inputs.

if (myPhoneBook.num == 8)

Se verifica si el número de contactos en myPhoneBook es 8, lo que indica que el PhoneBook está lleno. Si se da dicha condición:

for (int i = 7; i > 0; i--)

Este bucle for desplaza todos los contactos existentes una posición hacia abajo en el array Inputs, sobrescribiendo el contacto más antiguo.

myPhoneBook.Inputs[i] = myPhoneBook.Inputs[i - 1]; copia el contacto de la posición i-1 a la posición i.

Si el número de contactos es menor que 8, se ejecuta este bloque.

for (int i = myPhoneBook.num - 1; i >= 0; i--)

Este bucle for desplaza todos los contactos existentes una posición hacia abajo en el array Inputs, dejando la primera posición libre para el nuevo contacto.

myPhoneBook.Inputs[i + 1] = myPhoneBook.Inputs[i]; copia el contacto de la posición i a la posición i+1.

std::cin >> myPhoneBook.Inputs[0].name;

Lee la entrada del usuario y la almacena en el campo name del primer contacto en Inputs. Esto se repite con el resto de los campos de la agenda.

if (myPhoneBook.num < 8)

Verifica si el número de contactos es menor que 8.

myPhoneBook.num++; Incrementa el número de contactos en myPhoneBook.

continue; Vuelve al inicio del bucle while, esperando el siguiente comando del usuario.

else if (command == "SEARCH")

Este bloque de código maneja la búsqueda y visualización de contactos en el PhoneBook. Muestra una lista de contactos con sus índices, nombres, apellidos y apodos. Luego, solicita al usuario que ingrese el índice de un contacto para mostrar sus detalles completos. Si la entrada es inválida o el índice está fuera de rango, muestra un mensaje de error y vuelve al menú principal.

std::cout << std::setw(10) << i + 1;

Imprime el índice del contacto (empezando desde 1) en un campo de 10 caracteres de ancho, alineado a la derecha.

putstr(myPhoneBook.Inputs[i].name);

Llama a la función putstr para imprimir el nombre del contacto, truncado si es necesario. Esto mismo se realizará con el resto de los campos previstos.

std::cout << "Enter the index of the contact to display: ";

Imprime el mensaje "Enter the index of the contact to display: " en la consola, solicitando al usuario que ingrese el índice de un contacto para mostrar.

std::cin >> selector;

Lee la entrada del usuario y la almacena en la variable selector.

if (**std::cin.fail()**)

Verifica si la entrada del usuario no es válida (por ejemplo, si el usuario ingresó un carácter en lugar de un número).

std::cin.clear();

Limpia el estado de error del flujo de entrada.

std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

Explicación detallada:

std::cin.ignore es una función de la biblioteca estándar de C++ que se utiliza para ignorar (descartar) caracteres en el flujo de entrada std::cin. Esta función es útil para limpiar el búfer de entrada, especialmente después de leer datos que pueden dejar caracteres no deseados en el búfer, como un salto de línea (\n).

std::numeric_limits<std::streamsize>::max() devuelve el valor máximo que puede tener el tipo std::streamsize. **std::streamsize** es un tipo definido en la biblioteca estándar de C++ que representa el tamaño de un flujo de entrada/salida.

Al pasar este valor a std::cin.ignore, se indica que se deben ignorar hasta el máximo número posible de caracteres en el flujo de entrada.

'\n':

El segundo argumento de std::cin.ignore es el carácter delimitador hasta el cual se deben ignorar los caracteres. En este caso, '\n' (salto de línea) se utiliza como delimitador, lo que significa que std::cin.ignore ignorará todos los caracteres hasta encontrar un salto de línea o hasta alcanzar el número máximo de caracteres especificado por el primer argumento.

Propósito en el contexto del código:

En el contexto del código, esta línea se utiliza para limpiar el búfer de entrada después de leer un valor con std::cin. Esto es especialmente útil cuando se leen valores numéricos o cadenas de texto y se quiere asegurar que cualquier carácter residual (como un salto de línea) no interfiera con las siguientes operaciones de entrada.

En resumen, std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n'); se utiliza para limpiar el búfer de entrada, asegurando que cualquier carácter residual, como un salto de línea, no interfiera con las siguientes operaciones de entrada.

Despues de descartar la entrada no válida del búfer de entrada...

```
std::cout << "Invalid selection" << std::endl;
```

Imprime el mensaje "Invalid selection" en la consola.

```
continue;
```

Vuelve al inicio del bucle while, esperando el siguiente comando del usuario.

std::cin.ignore();

Ignora el carácter de nueva línea que queda en el búfer de entrada.

selector--;

Decrementa el valor de selector en 1 para convertirlo en un índice basado en cero.

if (selector >= 0 && selector < myPhoneBook.num): Esta línea verifica si el valor de selector está dentro del rango válido de índices de contactos en myPhoneBook. selector >= 0 asegura que el índice no sea negativo. selector < myPhoneBook.num asegura que el índice esté dentro del número actual de contactos almacenados en myPhoneBook.

std::cout << "First name: " << myPhoneBook.Inputs[selector].name << std::endl;:

Imprime el nombre del contacto seleccionado en la consola. Esto se repite con el resto de los campos de la agenda.

else: introduce un bloque de código alternativo que se ejecutará si la condición del if es falsa (es decir, si selector no está dentro del rango válido).

std::cout << "invalid selection" << std::endl;:
Imprime el mensaje "invalid selection" en la consola, indicando que el índice ingresado no es válido.

continue;:

La instrucción continue hace que el programa vuelva al inicio del bucle while, esperando el siguiente comando del usuario. Esto ignora cualquier código que venga después del continue dentro del bucle actual.

En resumen:

Este bloque de código maneja la selección de un contacto para mostrar sus detalles. Si el índice ingresado (selector) está dentro del rango válido de contactos, se imprimen los detalles del contacto seleccionado (nombre, apellido, apodo, número de teléfono y secreto más oscuro). Si el índice no es válido, se imprime un mensaje de "invalid selection" y el programa vuelve al inicio del bucle while para esperar el siguiente comando del usuario.

else if (command == "EXIT")

Este bloque de código maneja dos casos:

Comando "EXIT":

Si el usuario ingresa el comando "EXIT", el programa ejecuta break;, lo que termina el bucle while y hace que el programa salga del bucle infinito. Esto generalmente significa que el programa está a punto de finalizar.

Comando inválido:

Si el usuario ingresa cualquier otro comando que no sea "ADD", "SEARCH" o "EXIT", el programa imprime un mensaje de "Invalid command. Please enter ADD, SEARCH, or EXIT." en la consola. Esto informa al usuario que el comando ingresado no es válido y le recuerda las opciones válidas.

Después de imprimir el mensaje, el programa vuelve al inicio del bucle while y espera un nuevo comando del usuario.

En resumen, este fragmento de código asegura que el programa maneje adecuadamente el comando "EXIT" para salir del bucle y finalice el programa, y también maneja cualquier comando inválido proporcionando retroalimentación al usuario y esperando un nuevo comando.