Nombre del alumno: Said de Jesus Vidal Ramírez

Nombre del profesor: Leonardo Juárez Zucco

Nombre de la materia: Estructura de datos

Fecha: 14/03/2024

1.4 C++ Classes

1.4.1 Basic class Syntax

Este tema habla acerca del uso de las bases de c++ asi como también habla acerca las etiquetas como lo es la publica que se usa y sirve para poder cambiar datos de manera global, sin restricciones de ningún tipo, las privadas son diferentes a esta, ya que a estas se accede por medio de funciones que normalmente están en la misma etiqueta, tambien se aborda un poco acerca de dos tipos de constructores

1.4.2 Extra Constructor Syntax and Accessors

Primero nos profundiza mas acerca de los contructores dándonos a entender como intCell poner valores por preterminado a lo que son las variables para después mostrarnos un método de listas por el que es mas rápido poder tomar los valores asi como menos complicado ante datos mas complejos además de funcionar de la misma manera además enfatiza en que ahora se usan llaves en vez de paréntesis para nuevas versiones de c++ .

Asi como a su vez nos enseña la manera correcta de poder usar lo que son los intCell y como declarar y asignar de manera correcta datos de esta misma por medio de ejemplos además de darnos unos cuantos tips acerca del manejo de los datos

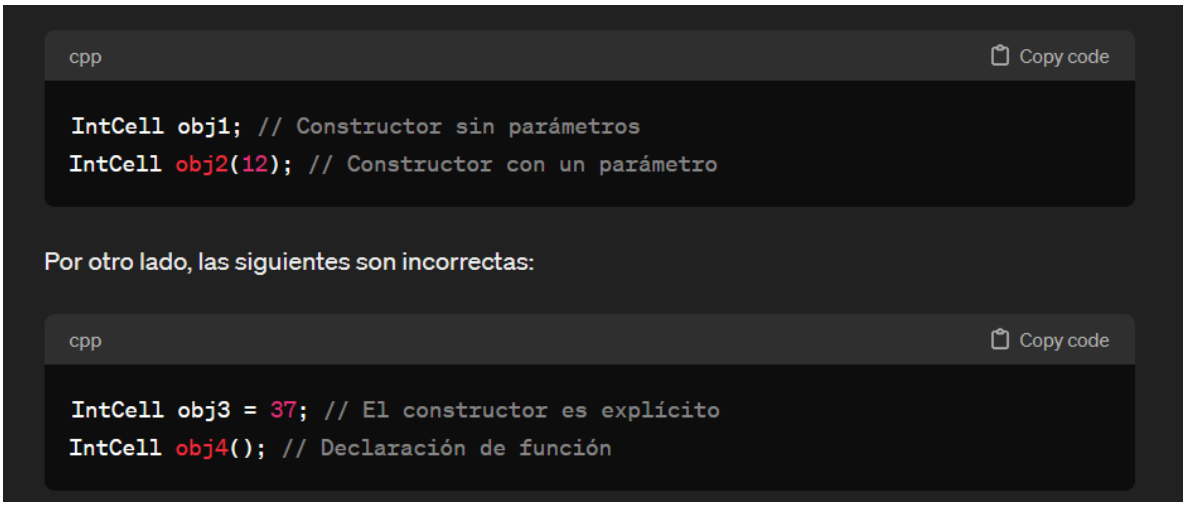
```
obj = 37;        // Should not compile: type mismatch
```

into

```
IntCell temporary = 37;
obj = temporary;
```

1.4.3 Separation of Interface and Implementation

Este tema nos habla acerca de como se separa un código por partes para lograr una mayor funcionalidad asi como comodidad a la hora de ejecutar un código nos habla de como mayormente la interfaz viene siendo un archivo .h en el que llama a todos las demas funciones para que este mismo funcione, vaya como lo haría una interfaz mientras como por otro lado nos habla acerca del correcto uso de lo legal e ilegal en c++

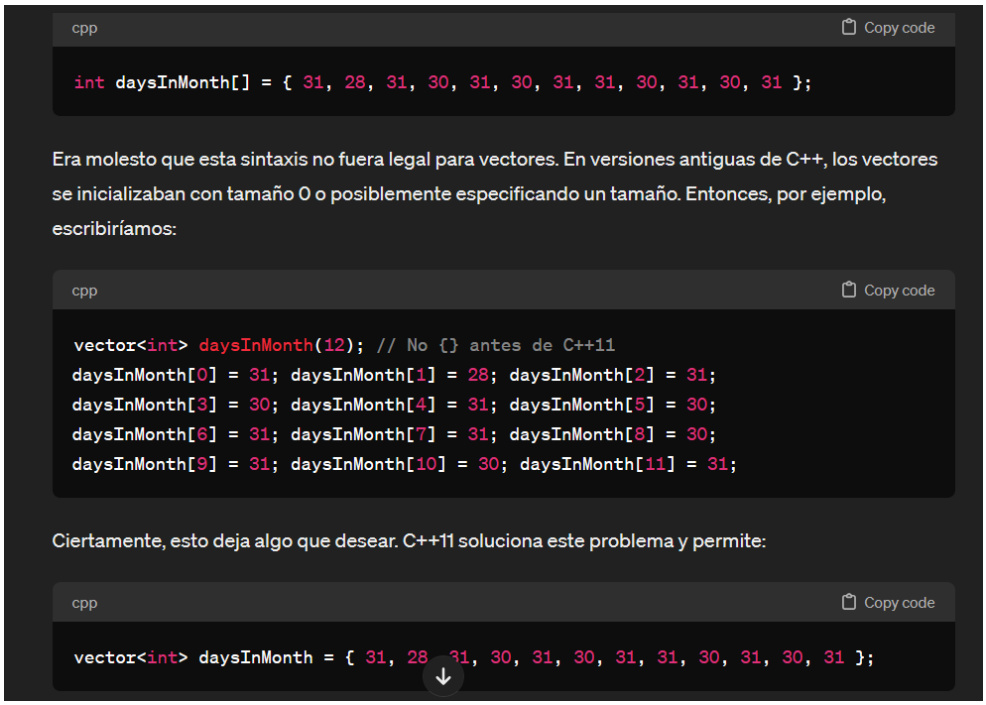


Además de enseñarnos como usarlos de manera correcta hoy en día

```cpp
IntCell obj1; // Constructor sin parámetros, igual que antes
IntCell obj2{12}; // Constructor con un parámetro, igual que antes
IntCell obj4{}; // Constructor sin parámetros
```

1.4.4 vector and string

En este tema nos habla acerca de lo que conozco como arrays o arreglos como dice en el libro, nos habla de las desventajas de los array integrados mismos que fueron reemplazados con vector y string, nos habla de los números problemas que tenían los array integrados cosas que los vectores arreglaron pero también nos habla de como los string son como los array integrados pero que cambiaron con unas cosas y trajeron nuevos problemas nos da un ejemplo sencillo de como usarlos de manera correcta en las versiones antiguas asi como las de las versiones mas nuevas



```cpp
int daysInMonth[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
```

Era molesto que esta sintaxis no fuera legal para vectores. En versiones antiguas de C++, los vectores se inicializaban con tamaño 0 o posiblemente especificando un tamaño. Entonces, por ejemplo, escribiríamos:

```cpp
vector<int> daysInMonth(12); // No {} antes de C++11
daysInMonth[0] = 31; daysInMonth[1] = 28; daysInMonth[2] = 31;
daysInMonth[3] = 30; daysInMonth[4] = 31; daysInMonth[5] = 30;
daysInMonth[6] = 31; daysInMonth[7] = 31; daysInMonth[8] = 30;
daysInMonth[9] = 31; daysInMonth[10] = 30; daysInMonth[11] = 31;
```

Ciertamente, esto deja algo que desear. C++11 soluciona este problema y permite:

```cpp
vector<int> daysInMonth = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
```

También nos pone un cuestionamiento de sintaxis en el que se pone a prueba a la nuevas versiones y se da un solución para el mismo ya que puede existir la ambigüedad con esto

vector<int> daysInMonth(12); // Se debe usar () para llamar al constructor que toma el tamaño

Aquí se muestra el ejemplo de como crear un vector con 12 espacios

Después nos enseña el correcto uso y la facilidad con la que se puede usar estos en operaciones matemáticas asi como el uso de la función auto que ya esta por preterminado en c++, esta nos sirve para que de manera automática el compilador le de un tipo de dato de manera automática, el que mas le convenga en todo caso

1.5 C++ Details

1.5.1 Pointers

Este tema nos explica sobre los punteros, como nos ayudan a guardar la ubicación de datos, en donde residen estos datos, estos nos facilita mucho las cosas ya que ayuda a manejar mejor la información asi como a optimizar el programa

Además también nos enseñan como crear objetos de manera dinámica mediante la función new

m = new IntCell{};

A su vez también nos advierten de que un objeto normal al ya no ser referenciado este de manera automática el compilador lo aparta y ya no lo toma en cuenta, caso contrario con el new ya que para estese tiene que aplicar la función delete.

Por ultimo para explicarnos algunos usos de los punteros para casos concretos.

1.5.2 Lvalues, Rvalues, and References

En este tema nos enseñan acerca de un nuevo tipo de referencia que se agrego con las ultimas versiones de c++, que son llamados “r” e “i” con estas declaraciones, arr, str, arr[x], &x, y, z, ptr, *ptr, (*ptr)[x] son todos valores de l. Además, x también es un valor de l, aunque no es un valor de l modificable. Como regla general, si tiene un nombre para una variable, es un valor de l, independientemente de si es modificable o no.

Para las declaraciones anteriores, 2, "foo", x+y, str.substr(0,1) son todos valores de r. 2 y "foo" son valores de r porque son literales. Intuitivamente, x+y es un valor de r porque su valor es temporal; ciertamente no es x ni y, pero se almacena en algún lugar antes de ser asignado a z. Una lógica similar se aplica para str.substr(0,1).

A su vez también nos muestran diferentes maneras de referenciar datos como por ejemplo:

lvalue references use #1: aliasing complicated names

Es usar una variable de referencia local únicamente con el propósito de renombrar un objeto conocido por una expresión complicada

Uso de las referencias de lvalor #2: bucles for de rango

Que ayuda a poder hacer una copia del valor dado de una forma mas eficiente

Uso de las referencias de lvalor #3: evitar una copia

Este método sirve para poder cambiar de nombre la variable sin necesidad de tener que hacer una copia

1.5.3 Parameter Passing

En este tema nos explican la correcta manera de usar varias funciones especializadas en la llamada de valores de valores como los avarege, que sirve para poder llamar datos de diferentes variables, swap que sirve para poder intercambiar el valor de dos variables entre si y randomItem, que permite agarrar un valor random de un array

Además de que el libro nos muestra el correcto uso de las mismas

1.5.4 Return Passing

Al igual que con el anterior tema, nos enseñan mas maneras de poder llamar datos de variables de manera correcta además de mucha información acerca de esta

```
double average( double a, double b );                    // returns average of a and b
LargeType randomItem( const vector<LargeType> & arr );   // potentially inefficient
vector<int> partialSum( const vector<int> & arr );       // efficient in C++11
```

1.5.6 The Big-Five: Destructor, Copy Constructor, Move

Constructor, Copy Assignment operator=, Move

Assignment operator=

En este tema una de las cosas mas importante que nos enseñan es acerca de los constructores

El destructor se llama cada vez que un objeto sale de ámbito o se somete a un delete. Típicamente, la única responsabilidad del destructor es liberar cualquier recurso que se haya adquirido durante el uso del objeto. Esto incluye llamar a delete para cualquier new correspondiente

Constructor de copia y constructor de movimiento

Hay dos constructores especiales que se requieren para construir un nuevo objeto, inicializado al mismo estado que otro objeto del mismo tipo. Estos son el constructor de copia si el objeto existente es un lvalue, y el constructor de movimiento si el objeto existente es un rvalue (es decir, un temporizador que está a punto de ser destruido de todos modos).

1.6 Templates

1.6.1 Function Templates

Los templates, son una clase de funciones que sirven como una plantilla para poder, luego convertirse en funciones, los templates de funciones son generalmente muy fáciles de escribir. Un template de función no es en realidad una función en sí, sino más bien un patrón para lo que podría convertirse en una función. La figura 1.19 ilustra un template de función llamado findMax. La línea que contiene la declaración del template indica que Comparable es el argumento del template: Puede ser reemplazado por cualquier tipo para generar una función.

1.6.2 Class Templates

Los templates de clase funcionan de manera similar a los templates de funciones. La Figura 1.21 muestra el template de clase MemoryCell. MemoryCell es similar a la clase IntCell, pero funciona para cualquier tipo de objeto.

```
/**
* Una clase para simular una celda de memoria.
*/
template <typename Object>
class MemoryCell
{
public:
explicit MemoryCell( const Object & initialValue = Object{ } )
: storedValue{ initialValue } { }
const Object & read( ) const
{ return storedValue; }
void write( const Object & x )
{ storedValue = x; }
private:
Object storedValue;
};
```

En resumen es una manera de hacer template, pero para las funciones para poder automatizar mas procesos, de manera eficiente

1.7.1 The Data Members, Constructor, and Basic Accessors

Este tema nos enseña acerca de la creación de matrices y como sirven

El atributo de datos matriz es representado por un vector de vector<Object>. El constructor primero construye la matriz como teniendo entradas de filas, cada una de tipo vector<Object> que es construido con el constructor de cero parámetros. Por lo tanto, tenemos filas de vectores de longitud cero de Object.

Luego, se ingresa al cuerpo del constructor y se redimensiona cada fila para tener cols columnas. Por lo tanto, el constructor termina con lo que parece ser una matriz bidimensional. Los accesores numrows y numcols son luego implementados fácilmente, como se muestra.

```
#ifndef MATRIX_H
#define MATRIX_H

#include <vector>
using namespace std;

template <typename Object>
class matrix
{
  public:
    matrix( int rows, int cols ) : array( rows )
    {
        for( auto & thisRow : array )
            thisRow.resize( cols );
    }

    matrix( vector<vector<Object>> v ) : array{ v }
      { }
    matrix( vector<vector<Object>> && v ) : array{ std::move( v ) }
      { }

    const vector<Object> & operator[]( int row ) const
      { return array[ row ]; }
    vector<Object> & operator[]( int row )
      { return array[ row ]; }

    int numrows( ) const
      { return array.size( ); }
    int numcols( ) const
      { return numrows( ) ? array[ 0 ].size( ) : 0; }
  private:
    vector<vector<Object>> array;
};
#endif
```

Conclusión:

En conclusión todas estas paginas me ayudaron a reforzar algunos temas, aunque algunos me quedaron duda o no lo entendí muy bien, me ayudo mucho a comprender lo que es el intCell, además me ayudo chatgpt para traducir el texto.

Bibliografía

[1] M. A. Weiss, Data Structures and Algorithm Analysis in C++, Florida : PEARSON, 2014.