
UNSA

UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA

# Ciencia de la computación II

Docente: Alvaro Henry Mamani Aliaga

Actividad

Resumen

Escuela:

Ciencia de la computación (Segundo año)

Temas:

-POO, Punteros, Templates, Matrices,

Alumno:

Josue Gabriel Sumare Uscca

## 1.4. C++ Classes

En el texto se veran estructoras de datos como un objeto quemanipula datos a traves de la implementación y dicho objeto y metodos en una clase

### 1.4.1. Basic Class Syntax:

Explica la sintaxis de una clase haciendo especial énfasis en los miembros de una clase como lo son los datos miembros y metodos, subrayando que en la creación de un objeto no siempre se inicializa todos los datos miembros en los objetos ya que al declarar como "STATIC" un dato miembro quiere decir que el valor del dato miembro static se copiara en todos los objetos.

Los miembros private pueden ser accedidos mediante metodospublicos.

El hecho de poner un miembro dato como private nos permite cambiar la representación interna del objeto sin afectar aquellas partes en las que se usa dicho objeto.

## 1.4.2 Extra Constructor Syntax and Accessors

Ojo: En el codigo 1.6 se cambio o reduzco el crear dos constructores, a uno solo con valor predeterminado O sino se pasa nigún valor.

- **Default Parameters**

El constructor con un parametro predeterminado nos ayuda a reducir la elaboración de dos constructores, uno por defecto y otro por parametro a uno solo con un parametro predeterminado.

- **Initialization list**

Se utiliza para inicializar los miembros dato antes del cuerpo del constructor

- **Explicit Constructor**

No se puede realizar una operación de asignación entre un objeto de un tipo y otro de diferente tipo.

• Constant Member Function

Se le coloca un const a aquellas funciones miembro o metodos que no realizan cambios en los datos miembros como lo serian los metodos de escritura.

## 1.4.3. Separation of Interface and Implentation

Se recomienda seguir un orden preestablecido como lo seria la separación de las declaraciones de los métodos con respecto a sus implementaciones.

• Preprocessor Commands

El preprocesador es muy útil para cuando se desarrolle un proyecto que tenga diferentes interfaces esto con el fin de que este archivo ".h" no se lea dos veces definiendo tal archivo con el nombre de su clase

• Scope Resolution Operator

En el archivo de implementacion los metodos deben estar identificados dentro de la clase a la que pertenece esto atraves del operador "::".

- Signatures must match exactly

Las firmas deben coincidir con sus respectivas implementaciones, a excepción de los parametros predeterminados, que se dan en las firmas y se omiten en la implementación.

- Objects are declared like Primitive types

Los objetos son de tipo primitivo, no podemos igualar un objeto a un tipo entero, sin haber una sobrecarga de dicha operación, uno al crear un constructor parametrizado se usa los parentesis para pasar por valor la inicialización de dicho objeto, con excepción de ser un constructor sin parametros en los que no debe ir un parentesis.

## 1.4.4 Vector and string

El vector esta destinado a reemplazar a la matriz, cuyo problema era que no se comporta como un objeto de primera clase, una cadena es una matriz de caracteres.

Un vector es facil de usar y acceder a su tamaño mediante un método, ademas de poder cambiar su tamaño segun la necesidad

# 1.5 C++ Details

## 1.5.1 Pointers

Los punteros son variables que almacena la dirección de memoria de otros objetos, además de ser el mecanismo fundamental para la creación de estructura de datos.

Un ejemplo claro es la lista enlazada, que es una estructura lineal, en donde los objetos almacenados no son guardados en memoria continuamente sino que los objetos se conectan o enlazan mediante punteros entre los nodos.

- **Dynamic object creation**

A través del uso de un puntero se usara memoria dinámica para la creación de un objeto dinámico, reservando memoria con el operador "new", una vez se deje de usar se usara el operado "delete", para dejar de usar la memoria almacenada.

- **Garbage Collection and delete**

En algunos lenguajes diferentes a C++ si se libera la memoria almacenada dinámicamente estos se alma cenan y no se liberan hasta acabar el programa.

- Accesing members of an object trough a pointer

Para acceder a los miembros públicos si a través de la creación de un objeto dinámico mediante un puntero, se usa el operador: "→".

- Address of Operator ( & )

A través del operador de dirección: "&" podemos acceder a la ubicación en memoria del operando (objeto)

1.5.2 Lvalues, Rvalues and References

- Lvalue: Objeto no temporal (declaraciones)
- Rvalue: Objeto temporal o expresión.

- References Rvalue

```
string test = "a";
string &rtest = test. //
bool cond = (&rtest == &test)
```

## Referencia lvalue #1 aliasing complicated name

```
auto & whichList = theLists[myhash(x, theLists.size())];
```

Referencia a lvalue — Objeto

- La referencia a lvalue se puede usar como una variable que reemplace a una expresión complicada.

## Referencia lvalue #2 range for loops

```
for ( auto &x : arr )
    ++x;
```

- Al incluir una referencia al lvalue nos permitirá cambiar directamente los valores del arreglo, y no hacer una copia de los valores del arreglo.

## Referencia lvalue #3 : Avoiding a copy

```
auto & x = findMax( arr )
```

- Esta notación con una referencia a un lvalue nos ayuda a evitar la creación de copias.

## 1.5.3 Parameter Passing

Caso 1

Swap (double& a, double& b)

Se usa la referencia como parametro para hacer un intercambio el los valores originales y no en las copias creadas dentro del uso de la función.

Caso 2

string randomItem(const vector<string>& arr)

En este caso, debido a que lo quese quiere es traer un elemento del arreglo y no operar sobre el arreglo en si, usamos la referencia para evitar una copia innecesaria de un vector.

## 1.5.4 Return Passing

Const LargeType &randonItem2(const vector<LargeType>& arr)

{ return arr [randomInt(0, arr.size()-1)];}

Accediendo a esta función mediante una referencia constante evitamos una copia.

Ejm:

const LargeType &item3 = randomItem2(vec);

## 1.5.5 std::swap and std::move

figure 1.14

```
void swap(double& x, double& y)
{   double tmp = x;
    x = y;
    y = tmp;
}
```

figure 1.15

```
void swap (vector<string>& x, vector<string>& y)
{
vector<string> tmp = static_cast<vector<string>
    && >(x);

x = static_cast<vector<string>&&>(y);
y = static_cast<vector<string>&&>(tmp);

}

void swap( vector<string>& x, vector<string>& y)
{

vector<string> tmp = std::move(x);
x = std::move(y);
y = std::move(tmp);
}
```

La implementación de la figura 1.5, a través de la función std::move que convierte cualquier lvalue o rvalue a rvalue, además de que un vector es compatible con movimiento estos nos permite realizar movimientos en vez de copias, lo que beneficia al peso computacional.

1.5. 6. The Big-Five: Destructor, Copy Constructor, Copy Assignment operator =, Move Assignment operator =

- Destructor

Nos permite eliminar un objeto, liberando espacio en memoria, a través de la destrucción del objeto, todos sus metodos que componen a este.

- Copy Constructor and Move Constructor

Estos dos constructores tienen un gran parecido en su funcionamiento, sin embargo su diferencia radica en que:

- Constructor copia: si el objeto existente es un lvalue
- Constructor move: si el objeto existente es un rvalue.

2. Sintaxis de declaración:

1. Constructor Copia:

```
object(Object const & obj)
```

2. Constructor move

Object (object && obj)

La funcionalidad de estos constructores es la inicialización de un objeto a través de la copia de datos miembros de otro.

- Copy Assignment and Move Assignment (operator =)

Obj 1 = Obj 2

- Si obj2 es un lvalue se usa el constructor copia
- Si obj2 es un rvalue se usa el constructor move.

Dato: En el constructor copia, se realiza un constructor copia para cada dato miembro del objeto.

- Defaults

Los valores predeterminados de los datos miembros son usados, cuando estos no son inicializados con la clase, no sucedería ningun inconveniente en los datos miembro de tipo primitivo

Cuando un dato miembro es un puntero la copia sera superficial, es decir la copia toma el valor del puntero y no a los objetos que este apunta.

Para solucionar tal problema se debe implementar el destructor, constructor copia, asignación copia, constructor de movimiento y asignación de movimiento.

- When the Defaults do not work

Este problema y solución se expuso en el punto anterior, ya que se hablaba de un dato miembro de tipo puntero.

1.5.7. C-style Arrays and Strings

- int arr1 [10] : Matriz, arr1 es un puntero que apunta al primer elemento del arreglo, por lo tanto al pasarlo por parametro se pierde información como su tamaño

- int *arr2 = new int [n]; : Arreglo dinámico mediante el uso de punteros.

  delete []arr2;

  Se debe liberar la memoria reservada anteriormente

- En una cadena existe el mismo comportamiento de una matriz, con excepción de que en la cadena no es necesario pasar el tamaño de esta, sino se realiza o se usa el "\0" que se encuentra implícitamente en todas las cadenas, para usarlo como final lógico.

- Los vectores son una alternativa al uso de las matrices, ya que los vectores se comportan como objetos que contienen otros objetos, dándonos las facilidades al necesitar el tamaño de dicho arreglo, además de brindarnos métodos que contiene dicha clase vector.

## 1.6 Templates

Nos ayudara a realizar algoritmos o funciones que no dependan del tipo de dato; esto beneficiara al hecho de generalizar un código permitiendo realizar menos código.

### 1.6.1 Function templates

Una función template no es en si una función sino mas bien un patrón para la creación de una función según el tipo que se use.

16.2 Class Templates

Una clase template tiene la misma funcionalidad que la de una función template.

Una clara necesidad de la implementación de una clase genérica o template, se refleja en la clase vector, que es una estructura de datos que debe almacenar cualquier tipo de datos, sean estos o no primitivos.

16.5 Separate Compilation of Class Templates

Lo mas recomendable por los problemas que pueden traer consigo separar en declaración y desarrollo o implementación de una clase genérica, se recomienda ponerlo todo en la declaración.

Sin embargo de separlo en una implementación se necesitara colocar en cada implementación el template.

ejm:

.h

```
template <typename T>
    class Object {

        void method ( <T> obj );
```

.cpp

```
Template <Typename T>
Object <T> :: method ( T obj) { }
```

ç

1.7 Using Matrices

Una matriz bidimensional, se puede aplicar en vectores, a través del uso de vector de vectores.

El operado [ ] es el operador de indexación de la matriz.

1.7.1 The Data Members, Constructor, and Basic Accesors

El constructor nos construye una matriz con entradas de filas, estos con longitud cero.

Luego ingresamos al cuerpo del constructor y modificamos el tamaño de cada fila para tener columnas.

Esto nos arma la matriz bidimensional a traves del uso de vectores; ademas de incluir los accesores del número de filas y columnas de nuestra matriz.

1.7.2 Operator [ ]

Este operador nos tendra que devolver el objeto correspondiente a la ubicación del arr[i][j] i.

En el caso de la matriz bidimensional la posición i, nos tendra que devolver un objeto de tipo matriz.