**C++ Notes**

Contents

[1.0 Vectors 2](#_Toc63238675)

[1.1 What are vectors 2](#_Toc63238676)

[1.2 How to create vectors 2](#_Toc63238677)

[1.3 Iterators 2](#_Toc63238678)

[2.0 POINTERS 4](#_Toc63238679)

[3. Classes, functions and variables 5](#_Toc63238680)

[3.1 Header files 5](#_Toc63238681)

[3.2 Code files 6](#_Toc63238682)

[3.3 Main method 6](#_Toc63238683)

[3.4 Friend function 7](#_Toc63238684)

[3.5 This pointer 7](#_Toc63238685)

[3.6 Static class member 8](#_Toc63238686)

[3.7 Virtual function specifier 8](#_Toc63238687)

[3.8 Casting 8](#_Toc63238688)

[4. Inheritance and Polymorphism 9](#_Toc63238689)

[4.1 Inheritance 9](#_Toc63238690)

[4.2 Private members 9](#_Toc63238691)

[4.3 Polymorphism 10](#_Toc63238692)

[4.4 Abstract 10](#_Toc63238693)

s

# 1.0 Vectors

## 1.1 What are vectors

Vectors are a sequence container that store elements. Specifically used in order to work with dynamic data, c++ vectors can expand and scale down as needed. Efficient when you don’t know how big an array will be, however, more costly than a fixed size array in terms of memory and performance.

## 1.2 How to create vectors

Common syntax for vectors is as follows;

vector <type> variable (elements)

For example:

vector <int> rooms (9);

The type and variable name are mandatory for initializing a vector, however, the number of elements is optional.

## 1.3 Iterators

An interator allows you to access the data elements in a vector. There are 5 types of iterators

When iterating through a vector, use size\_t instead of int, as size\_t is the type returned by the .size function.

For(size\_t i = 0; i<vector.size(); i++){

However, the better syntax for iterating over a vector, especially one of generic objects, is to use the following

for (std::vector<**type**>::iterator it = myvector.begin() ; it != myvector.end(); ++it)

To use the ‘it” iterator, use ‘it\*’ which should return the number of the iterator (0, 1, 2 etc.)

Std::cout << it\* << std::endl;

Will print out the iterator with a new line at the end.

<https://www.bitdegree.org/learn/c-plus-plus-vector#:~:text=C%2B%2B%20vectors%20are%20sequence%20containers,vectors%20can%20automatically%20manage%20storage>.

<https://stackoverflow.com/questions/15802006/how-can-i-create-objects-while-adding-them-into-a-vector>

<http://www.cplusplus.com/forum/beginner/223033/>

Read more into vectors

Graphical user interface, text, application, email

Description automatically generated

# 2.0 POINTERS

Table

Description automatically generated

# 3. Classes, functions and variables

## 3.1 Header files

Text

Description automatically generatedHeader files are designed to provide the information that s needed in code files. They define things such as class declarations, their functions prototypes and variables. This is a general format for how the header class is created, with the different things you can do with it.

A smart way to construct a class is by giving the parameters a default value for any missing parameters when the constructor is initiated.

Text

Description automatically generatedThe ‘const’ keyword at the ends of these lines show that the methods can not change in any way during their runtime.

## 3.2 Code files

Text

Description automatically generatedCode files (.cpp) are, as expected, where the code is. They use the structure from the header file and give the methods and variables meaning. This simple screenshot shows the general gist of how code files are written.

**A picture containing diagram

Description automatically generatedConstant** variables and methods still need to be catered for when writing their code version.

Some classes have variables using other classes data types (Such as a date for example). This is known as **composition**.

## 3.3 Main method

Main methods are where code is tested and executed. Below is how an object is initiated, one being a constant object and the other a normal object.

Text

Description automatically generated

## 3.4 Friend function

A friend function is a function or method which is assigned outside of the class, but has access to the private and protected data members of the class. A function is a friend function when it precedes the keyword “friend”.

[Useful information](https://www.softwaretestinghelp.com/friend-functions-in-cpp/#:~:text=A%20friend%20function%20in%20C%2B%2B,data%20members%20of%20the%20class.)

## 3.5 This pointer

The “this” pointer is an important pointer which every C++ object has. It is an implicit parameter to all member functions. It can be used to refer to the invoking object. Friend object do not have “this” pointers as they are not members of the class, and need pointers to objects to be used.

The “this” pointer can also be returned to invoke multiple functions within the same method, like so:

Text, letter

Description automatically generatedThis is done by doing the following in the header, and the code in the .cpp file in the subsequent screenshot.

## 3.6 Static class member

A static data member is a copy of a variable shared between all objects of the same class. An example would be a count. A normal variable of count will keep track of separate integers within the separate objects. However, if the count variable is static, all objects of this class share the same count.

## 3.7 Virtual function specifier

Research it

## 3.8 Casting

To cast a variable, use the variable data type in brackets before the data, like so:

float totalSum = (float)balance;

Where balance is an integer and the total sum needs to use decimals.

## 3.9 Constants

A constant is a variable stored in a memory location. The difference between variables and constants is that a constant’s value cannot be changed. In c++ a constant is denoted by the const keyword.

Timeline

Description automatically generatedNon-constant objects can call any type of method, whereas constant objects can only call constant methods, and cannot call non-constant methods. wakeup is a non-const and noon is a const. The highlighted lines are errors and cannot be called.

# 4. Inheritance and Polymorphism

## 4.1 Inheritance

Inheritance absorbs existing class’s data and behaviour and enhances it with new capabilities. The main relationship is between the base class and derived class, also known as parent and child, or super and sub classes in java.

Generally the derived class represents a more specialised group of objects, by implementing additional methods on another class, such as in a game an enemy class is a parent or base class, and a type of enemy like a zombie is a child or derived class.

C++ supports multiple inheritance.

A has-a relationship represents composition, and an is-a relationship represents inheritance. A Person **has-a** bank account, and a Student **is-a** Person.

Diagram

Description automatically generatedExamples

## 4.2 Private members

A derived-class function may need access to base-class data.

Base-class members that should not be accessible to derived classes should be declared as private.

Derived classes can only change access base-class private members through non-private base-class functions.

A base class’s public members are accessible within its body and anywhere in the program that has a handle or one of its derived classes. Private members are only accessible within its body and to the friends of the base class, and using **protected** access offers an intermediate level.

## 4.3 Polymorphism

Polymorphism refers to a programming language’s ability to process objets differently depending on their data type or class. Using the same method but with different uses for each child class, such as area() for different shapes.

## 4.4 Abstract

Abstract classes are declared as abstract, they may or may not include abstract methods. They cannot be instantiated, but they can be subclassed. Abstract classes are declared without any implementation,

# Design patterns

## Creational patterns

Creational patterns are concerned with the construction of object instances. Generally, they separate application operation from how objects are created. This decoupling gives the designer flexibility in configuring all aspects of object creation.

**Singleton**

Singleton pattern is used to ensure there is only ever one instance of a class is created, and no more. An example could be when a management system holds information regarding a company, such as a name, head office, employee details. It only wants one instance of this information; this is where the singleton pattern would come in to use.

This information should only be held in one place but will be used by many objects.

Graphical user interface, text, application

Description automatically generatedOne solution is to use global data that can be accessed by all objects. However, this violates encapsulation. The creation of a Company class would overcome this problem by encapsulating attributes.

Now comes the problem that when an object wants to access this Company object, they must know its identifier, which can’t be globally available.

The solution is to make the class static. This way it makes the class accessible to anywhere in the software.

A static getter can be used to return the instance, and the only reason it would not return is if it did not exist, so creates a new instance.

If(companyInstance == null)

{

companyInstance = new Company();

}

## Structural patterns

Concerned with how classes and objects are organised. An effective way of using inheritance, aggregation and composition to satisfy requirements.

For example, it might be a requirement that the application should easily be extensible. S/w should be designed with constructs that minimise the side-effects of future change.

**Composite**

Diagram

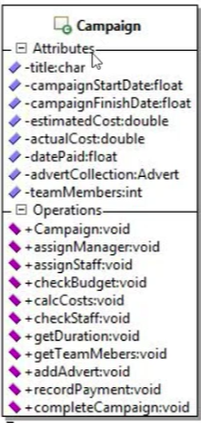
Description automatically generatedA multimedia application can store and play components of an advert. Advert consists of sound clips and video clips. Each can be played individually or as part of an advert.

## Behavioural patterns

Address problems that arise when responsibilities are assigned to classes and in designing algorithms. Describe the relationships and communication between objects. They may use inheritance to spread behaviour across subclasses or they may use aggregation and composition to build complex behaviour.

**State pattern**

Can be used when objects have significant state-dependent behaviour. An example is of a Campaign object that can be in one of four main states; Commissioned, Active, Completed, and Paid.

The following diagram would be a possible scenario for constructing a working version of the Campaign.

Diagram

Description automatically generatedUnfortunately, this would be a complex class that is further complicated by state-dependent operations. Lots of if-else statements or switch cases. A better solution for this would be the following:

Using classes to represent each state makes the class cleaner and much more maintainable. This is the state pattern.