**Declaration :** a function or variable exist somewhere and program do not allocated memory for them.

Definition : program allocate memory for that function or variable

**Extern Keyword:** used for variable, function includes this keyword by default

This comes in useful when you have global variables and want to share it between some source file. You declare the existence of global variables in a header and **you only need to** **“define” it once in one of your source** **files**. so that each source file that includes the header knows about it

**Virtual key word: use to overriding**

* Define virtual function in base class.
* Re-define it in derived class

 Runtime polymorphism is achieved only through a pointer (or reference) of base class type. A base class pointer can point to the objects of base class as well as to the objects of derived class. If the content of base class pointer is address of derived class object, they will call overrieded function

**Static keyword:**

* **Static variable in a function :** When a variable is declared as static, space for **it gets allocated for the lifetime of the program**. Even if the function is called multiple times, space for the static variable is allocated only once and the value of variable in the previous call gets carried through the next function call
* **Static variables in a class**: As the variables declared as static are initialized only once as they are allocated space in separate static storage so, the static variables **in a class are shared by the objects.** There can not be multiple copies of same static variables for different objects. Also because of this reason static variables can not be initialized using constructors.

 So, a static variable inside a class should be initialized explicitly by the user using the class name and scope resolution operator outside the class as shown below int GfG::i = 1

**Static member functions** are used to maintain a single copy of a class member function across various objects of the class. Static member functions can be called either by itself, independent of any object, by using class name and :: (scope resolution operator) or in connection with an object.  
- Restrictions on static member functions are:  
  
1. They can directly refer to other static members of the class.  
2. Static member functions do not have **this** pointer.  
3. Static member function can not be **virtual**.

**Function Pointer :store the addess of function or point to the start of execution code**

int (\*foo)(int);

using when want to pass a function to another function or callbacks function for example (pthread\_create(p\_thread, functionA))

**Disadvantage and advantage of array**

* **Advantage :** easy to get an element.
* **Disadvantage:** fixed size, consecutive memory. To delete and add one element to array takes a lot of time because you must move the other element after the element added to next index.

**Malloc vs calloc**

* Malloc is memory allocation while calloc is contiguous allocation

**Stack vs heap**

stack is used to store local variable and allocated in compiling time while heap is located during runtime

stack has fixed size while heap size depends on OS

when a function return, stack delete memory storing local variable of the function while memory in heap is released by programmer.

**Static\_cast<type> to cast a C++ type**

# List >< Vector

Constructor : same constructor for vector and list

1 vector<int> V: empty

2. vector<int> V(2, 100) : 2 elements =100;

3. vector <int> V(V1.begine, V1.end): initialize a range V

4. vector <int> V(V1): copy constructor

Access elements of Vector and List:

* Random access for Vector (at(i), [i] to access content of element i)
* Iterator to access List (list<int>::interator it=list.begin(), it!=list.end(), it++ and then deference \*it to take value of each element)

List stores its elements in non continuos memory while vector stores in continuous memeory. And each node contains a pointer to next node. So, we must iterate from first node to the end.

Insert() and erase to add and delete an element.

It is more efficient to add and remove a element in list than do in vector. Because in list, just swap 2 pointer while in vector must move all behind element backward by 1.

**These are the simple conclusions on usage of each data structure:**

* For**linear search**: use std::vector
* For **Random Insert/Remove**: use std::list (if data size very small then use std::vector)
* For **big data size**: use std::list (not if intended for searching)

Vector nD vector<vector<vector<vector<int> > > >

**Con tro ham : int f (int (\*function)(int, int), int): pass address of function to f**

**Pass array to function is pass a pointer to function while pass a vector to function is copy the vector to function**

1. **Diference between reference and pointer**

There are three critical attributes of pointers that differentiate them  
from references.  
  
1. You use pointer syntax to access the values "pointed to" by the pointer.  
2. You can redirect the pointer to point it to a different "target" variable.  
3. You can make a pointer point to nothing (ie, NULL pointer).

References have a couple of key characteristics  
that differentiate them from pointers:  
1. References must be initialized at the point of instantiation.  
2. References must ALWAYS "refer to" a named or unnamed variable  
(that is, you cannot have a reference variable that refers to  
nothing, the equivalent of a NULL pointer).  
3. Once a reference is set to refer to a particular variable, you  
cannot, during its lifetime, "rebind" it to refer to a different  
variable.  
4. You use normal "value" syntax to access the value being referred to

**When to pass parameters by value, by reference, and by pointer**

1. Pass by value when the function does not want to modify the parameter and the value is easy to copy (ints, doubles, char, bool, etc... simple types.std::string,std::vector, and all other STL containers are NOT simple types.)  
     
   2. Pass by const pointer when the value is expensive to copy AND the function does not want to modify the value pointed to AND NULL is a valid, expected value that the function handles.  
     
   3. Pass by non-const pointer when the value is expensive to copy AND the function wants to modify the value pointed to AND NULL is a valid,expected value that thefunction handles.  
     
   4. Pass by const reference when the value is expensive to copy AND the function does not want to modify the value referred to AND NULL would not be a valid value if a pointer was used instead.  
     
   5. Pass by non-cont reference when the value is expensive to copy AND the function wants to modify the value referred to AND NULL would not be a valid value if a pointer was used instead.
2. Difference between struct and class?

A Structure is not secure and cannot hide its implementation details from the end user while a class is secure and can hide its programming and designing details

1. Members of a class are private by default and members of a struct are public by default.
2. When deriving a struct from a class/struct, default access-specifier for a base class/struct is public. And when deriving a class, default access specifier is private.

**Public**: All the class members declared under public will be available to everyone. The public members of a class can be accessed from anywhere in the program using the direct member access operator (.) with the object of that class.

**Private**: the class members declared as *private* can be accessed only by the functions inside the class. They are not allowed to be accessed directly by any object or function outside the class. **Only the member functions or the [friend functions](https://www.geeksforgeeks.org/friend-class-function-cpp/)** are allowed to access the private data members of a class.

**Protected** access modifier is similar to that of private access modifiers, the difference is that the class member declared as Protected are inaccessible outside the class but they can be accessed by any subclass(derived class) of that class

1. Purpose of inheritance

To reuse existing code

**Modes of Inheritance**

1. **Public mode**: If we derive a sub class from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in derived class.
2. **Protected mode**: If we derive a sub class from a Protected base class. Then both public member and protected members of the base class will become protected in derived class.
3. **Private mode**: If we derive a sub class from a Private base class. Then both public member and protected members of the base class will become Private in derived class.

**Note :**The private members in the base class cannot be directly accessed in the derived class even when in public mode, while protected members can be directly accessed.

1. **What is virtual keyword?**

Virtual Function in C++

A virtual function a member function which is declared within a base class and is re-defined(Overriden) by a derived class. **When you refer to a derived class object using a pointer or a reference of the base class, you can call a virtual function for that object and execute the derived class’s version of the function**.

A class may have [virtual destructor](https://www.geeksforgeeks.org/virtual-destructor/) but it cannot have a virtual constructor.

They are always defined in base class and overridden in derived class.4

Pure virtual function:   virtual void show() = 0; you don’t know how to implement the function in base class. Must be define in derived clasee

Overloading vs overriding

1. **Inheritance:** Overriding of functions occurs when one class is inherited from another class. Overloading can occur without inheritance.
2. **Function Signature:** Overloaded functions must differ in function signature ie either number of parameters or type of parameters should differ. In overriding, function signatures must be same.
3. **Scope of functions:** Overridden functions are in different scopes; whereas overloaded functions are in same scope.
4. **Behavior of functions:** Overriding is needed when derived class function has to do some added or different job than the base class function. Overloading is used to have same name functions which behave differently depending upon parameters passed to them.
5. Copy constructor, assignment constructor, move constructor?

Constructor is automatically called right after memory is allocated for an object. The object is initialized.

C library provide a build in copy constructor. However, when there is a pointer in member data of class, both copied object and copying object point to the same block memory. Possibly both objects destroy the same block of memory when destructor is called. That cause crash**.**

**The copy constructor** is used to initilize the new object with the previously created object of the same class. Point(const Point &p2)

Therefore to remove this problem we wrote deep copy so both objects have their own copy of attributes in a memory. The copying object copy content of pointer.

**Assignment operator** is called when an already initialized object is assigned a new value from another existing object.

DeepCopy& operator=(DeepCopy&A)

{

delete ptr;

ptr=A.ptr;

return \*this;

}

**Move constructor** Move mean that b move to a, so b disappear. Move constructor is used to initialize a by move b to a while b is unnamed object that returned by function and constructor or type cast

DeepCopy (DeepCopy &&A): ptr(A.ptr){A.ptr=nullptr;};

# inheritance vs composition

Inheritance is is- a- relationship. an object acquires the characteristics of one or more other objects. A child class inherits both behaviors (member functions) and properties (member variables) from the parent

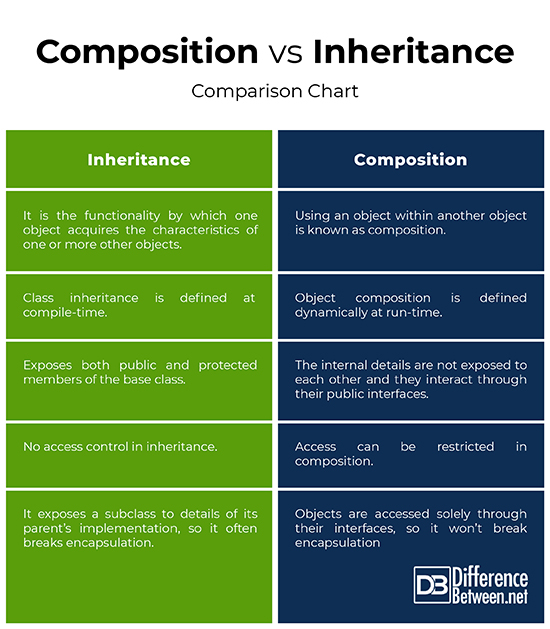
Composition is has-a relationship. Using an object within another object is known as composition.

* The part (member) is part of the object (class)
* The part (member) can only belong to one object (class) at a time
* The part (member) has its existence managed by the object (class)
* The part (member) does not know about the existence of the object (class)

**Aggregation is also has a relationship**, a whole object and its parts must have the following relationship:

* The part (member) is part of the object (class)
* The part (member) can belong to more than one object (class) at a time
* The part (member) does *not* have its existence managed by the object (class)
* The part (member) does not know about the existence of the object (class)

For example: person and address, address is not destroyed after person has gone and address also belong to some others.



In inheritance, we can adjust some virtual function in order to fit subclass

1. How template works?

Function templates are special functions that can operate with *generic types*. this allows us to create a function template whose functionality can be adapted to more than one type or class without repeating the entire code for each type.

Template works like function overloading.

There are 2 type of template

**Function template**

template<typename T0, typename T1, typename R = std::common\_type\_t<T0, T1>>

R max (T1 x, T0 y)

{

Return ® x>y? x:y;  
}

Or

#include <stdio.h>

#include<iostream>

template <typename T1, typename T2>

void print\_max(T1 x, T2 y)

{

std::cout<<(x>y? x:y);

}

int main()

{

print\_max(5,7.4);

return 0;

}

**Template class**

Put all cpp file in header file and add template <class T> before each function you write below declaration of the class. Like below

Template <class T>

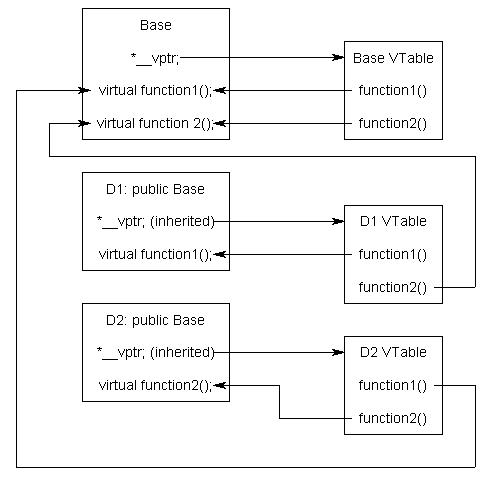
Void Class<T> :: print()

{  
 }

Template class >< template function: function can infer the type of argument based on actual data while class can not

1. How to avoid virtual table???

Each class use virtual function, there is a virtual table for the class. Each entry of the table is a function pointer to the most derived virtual function. It costs much memory to save function pointers



The way to avoid Vtable:

Derived class keeps being inheritance from Base class and do not use virtual function. In derived class, rewrite a function that is member function of base class. And then we create a pointer of Derived class. Convert this pointer to Base type by casting

derived \*d = new derived;

base \*b = (base\*)d;

d->greet();

b->greet();

delete d;

1. What is dependency injection

When you have a class (A) which has a object of another class B (composition) as its member. When you want to create an instance of B, you must create an instance of A.

Dependency injection means **transferring the task of creating the object B to someone else and directly using the dependency.**

**There are three type of DI**

1. **constructor injection:** the dependencies are provided through a class constructor.
2. **setter injection:** the client exposes a setter method that the injector uses to inject the dependency.
3. **interface injection:** the dependency provides an injector method that will inject the dependency into any client passed to it. Clients must implement an interface that exposes a [setter method](https://en.wikipedia.org/wiki/Setter_method) that accepts the dependency.

Used to do unit test for any class you choose

1. **Devoloper feeling that singleton is not good, do you agree or disagree?**

No, everything has pros and cons

Pros: class can control instantiation itself. Only one object of this class is created and easy to access globally via static method getInstance()

Cons: the pattern hide constructor, and tight coupling. So difficult to do unit test. Thread safety must do because we share resource

1. What is ABI (application binary interface) is an [interface](https://en.wikipedia.org/wiki/Interface_(computing)" \o "Interface (computing)) between two binary program modules; often, one of these modules is a [library](https://en.wikipedia.org/wiki/Library_(computing)" \o "Library (computing)) or [operating system](https://en.wikipedia.org/wiki/Operating_system) facility, and the other is a program that is being run by a user. It provide details on how binary (object file )must be complied and interface with platform

It covers:

Calling convenstion ( how stack control, routine, data pass and return, state save

Programing code and programing data(the sizes, layouts, and [alignments](https://en.wikipedia.org/wiki/Data_structure_alignment" \o "Data structure alignment) of basic [data types](https://en.wikipedia.org/wiki/Data_type" \o "Data type) that the processor can directly access)

Build process :

1. preprocessor replace # by content of macro
2. compiler translate source code to assembly code
3. assembler translate assembly code to object code
4. linker links all object code to a single executable file and use linker file to map code and data to fit memory sections of microcontroller.

Object code is binary code – human unreadable.

# 12 Which design patterns do you know

Factory pattern:

**define an interface or abstract class for creating an object but let the subclasses decide which class to instantiate. Virtual constructor.**

**You can change type of object at runtime and do not need to compile your code.**

**When your library change(add more classes), you just change interface and recompile it.**

1. Proxy injection

That is a middle layer which control to access service.

# Friendship is considered as inheritance

Friends keyword

Friend function: declare the prototype of this function inside a class with friend keyword in front. That means the function can access to private data of the class.

Friend class: declare the friend keyword in the front of class prototype (A) inside another class (B). That means A is friend of B, B is not fiend of A and member function of class A can access to private member of B.

*Note:*

*you must pass an object of B class to a member function of A. because there is not \*this pointer inside fiend class and function.*

*Should not use fiend keyword because it break encapsulation of data.*

Friendship is not inheritance:

1. If class A is supper class of class B, B own all properties of A and B is A
2. If class A is friend of class B, class A member can access private property of B
3. Class A do not have this pointer of B. I must pass a B object to A
4. What is translation unit?

A translation unit is the basic unit of compilation in C++. It consists of the contents of a single source file, plus the contents of any header files directly or indirectly included by it, minus those lines that were ignored using conditional preprocessing statements.

A C program consists of *units* called [*source files*](https://en.wikipedia.org/wiki/Source_code) (or *preprocessing files*), which, in addition to source code, includes directives for the [C preprocessor](https://en.wikipedia.org/wiki/C_preprocessor" \o "C preprocessor). A translation unit is the output of the C preprocessor – a source file after it has been [preprocessed](https://en.wikipedia.org/wiki/Preprocessor" \o "Preprocessor).

1. How to implement signal&slot mechanism?

Use thread, mutex and conditional variable.

The thread that intends to modify the variable has to

1. acquire a std::mutex (typically via [std::lock\_guard](https://en.cppreference.com/w/cpp/thread/lock_guard))
2. perform the modification while the lock is held
3. execute [notify\_one](https://en.cppreference.com/w/cpp/thread/condition_variable/notify_one) or [notify\_all](https://en.cppreference.com/w/cpp/thread/condition_variable/notify_all) on the std::condition\_variable (the lock does not need to be held for notification

Any thread that intends to wait on std::condition\_variable has to

1. acquire a [std::unique\_lock](http://en.cppreference.com/w/cpp/thread/unique_lock)<[std::mutex](http://en.cppreference.com/w/cpp/thread/mutex)>, on the same mutex as used to protect the shared variable
2. execute [wait](https://en.cppreference.com/w/cpp/thread/condition_variable/wait), [wait\_for](https://en.cppreference.com/w/cpp/thread/condition_variable/wait_for), or [wait\_until](https://en.cppreference.com/w/cpp/thread/condition_variable/wait_until). The wait operations atomically release the mutex and suspend the execution of the thread.
3. When the condition variable is notified, a timeout expires, or a [spurious wakeup](https://en.wikipedia.org/wiki/Spurious_wakeup) occurs, the thread is awakened, and the mutex is atomically reacquired. The thread should then check the condition and resume waiting if the wake up was spurious.
4. After finishing its task, it unlock and notify one or all

Note thread::join() means: The main thread calls t1.join() to wait for the t1 thread to finish and return to main thread

nếu vậy cậu ở nhà chăm con vài tháng, trong khi đó cậu tìm hiểu những kiến thức này : + Multithread + IPC (Message queue, socket...) + Hướng đối tượng + Design pattern (chủ yếu là singleton, factory, observer, adapter ) + Cấu trúc dữ liệu và thuật toán ( cái này ôn Pro ở SS chắc học nhiều rồi) + Tiếng anh toeic trên 600 còn hạn (để nếu cậu được lên title chief thì cũng đủ điều kiện)