C++ Questionnaire

1.Write a function template named add that allows the users to add 2 values of the same type. The following program should run:

#include <iostream>

// write your add function template here

int main()

{

std::cout << add(2, 3) << '\n';

std::cout << add(1.2, 3.4) << '\n';

return 0;

}

Answer:

template <typename T>

T add(T x, T y)

{

return x + y;

}

2.What’s wrong with each of these snippets, and how would you fix it?

int main()

{

int array[]{ 0, 1, 2, 3 };

for (std::size\_t count{ 0 }; count <= std::size(array); ++count)

{

std::cout << array[count] << ' ';

}

std::cout << '\n';

return 0;

}

Answer:The loop has an off-by-one error, and tries to access the array element with index 4, which does not exist. The conditional in the for loop should use < instead of <=.

3.What’s wrong with each of these snippets, and how would you fix it?

int main()

{

int x{ 5 };

int y{ 7 };

const int\* ptr{ &x };

std::cout << \*ptr << '\n';

\*ptr = 6;

std::cout << \*ptr << '\n';

ptr = &y;

std::cout << \*ptr << '\n';

return 0;

}

Answer:ptr is a pointer to a const int. You can’t assign the value 6 to it. You can fix this by making ptr non-const.

4.Write function prototypes for the following cases. Use const if/when necessary.

a) A function named max() that takes two doubles and returns the larger of the two.

Answer:double max(double x, double y) const;

b)A function named swap() that swaps two integers.

Answer: void swap(int& x, int& y);

c) A function named getLargestElement() that takes a dynamically allocated array of integers , length parameter and returns the largest number in such a way that the caller can change the value of the element returned

Answer:

int& getLargestElement(int\* array, int length);

7)What’s wrong with these programs?

int& doSomething()

{

int array[]{ 1, 2, 3, 4, 5 };

return array[3];

}

Answer:doSomething() returns a reference to a local variable that will be destroyed when doSomething terminates.

8)What’s wrong with these programs?

#include <iostream>

int main()

{

int array[100000000]{};

for (auto x: array)

std::cout << x << ' ';

std::cout << '\n';

return 0;

}

Answer:The array is too large to be allocated on the stack. It should be dynamically allocated.

9)Write a destructor for this class:

#include <iostream>

class HelloWorld

{

private:

char\* m\_data{};

public:

HelloWorld()

{

m\_data = new char[14];

const char\* init{ "Hello, World!" };

}

~HelloWorld()

{

// replace this comment with your destructor implementation

}

void print() const

{

std::cout << m\_data << '\n';

}

};

int main()

{

HelloWorld hello{};

hello.print();

return 0;

}

Answer:

~HelloWorld()

{

delete[] m\_data;

}

10)From below table of all of the access specifiers, Please specify the inheritance types for below combinations

|  |  |  |  |
| --- | --- | --- | --- |
| Access specifier in base class | Access specifier when inherited publicly  Class Derived: public Base | Access specifier when inherited privately  Class Derived: private Base | Access specifier when inherited protectedly  Class Derived: protected Base |
| Public | Public/Private/Protected? |  |  |
| Protected |  |  |  |
| Private |  |  |  |

Answer:

|  |  |  |  |
| --- | --- | --- | --- |
| Access specifier in base class | Access specifier when inherited publicly | Access specifier when inherited privately | Access specifier when inherited protectedly |
| Public | Public | Private | Protected |
| Protected | Protected | Private | Protected |
| Private | Inaccessible | Inaccessible | Inaccessible |

11)Point out the error

#include <iostream>

class Base

{

public:

int m\_value{};

};

class Derived : public Base

{

private:

using Base::m\_value;

public:

Derived(int value) : Base { value }

{

}

};

int main()

{

Derived derived{ 7 };

std::cout << derived.m\_value;

Base& base{ static\_cast<Base&>(derived) };

std::cout << base.m\_value;

return 0;

}

Answer:

std::cout << derived.m\_value; // error: m\_value is private in Derived

std::cout << base.m\_value; // okay: m\_value is public in Base

11)Determine the output

#include <string\_view>

class Base

{

public:

std::string\_view getName() const { return "Base"; }

};

class Derived: public Base

{

public:

std::string\_view getName() const { return "Derived"; }

};

int main()

{

Derived derived{ };

Base& rBase{ derived };

Base\* pBase{ &derived };

Base base {derived};

std::cout << "derived is a " << derived.getName() << '\n';

std::cout << "rBase is a " << rBase.getName() << '\n';

std::cout << "pBase is a " << pBase->getName() << '\n';

std::cout << "Base is a " << base.getName() << '\n';

return 0;

}

Answer:

derived is a Derived

rBase is a Base

pBase is a Base

Base is a Base

Explanation:

Because rBase and pBase are a Base reference and pointer, they can only see members of Base (or any classes that Base inherited). So even though Derived::getName() shadows (hides) Base::getName() for Derived objects, the Base pointer/reference can not see Derived::getName(). Consequently, they call Base::getName(),

Base base {derived};

When we assign a Derived object to a Base object, only the Base portion of the Derived object is copied. The Derived portion is not. In the example above, base receives a copy of the Base portion of derived, but not the Derived portion. That Derived portion has effectively been “sliced off”. Consequently, the assigning of a Derived class object to a Base class object is called object slicing (or slicing for short).

12)Determine the output

#include <iostream>

#include <string\_view>

class A

{

public:

virtual std::string\_view getName() const { return "A"; }

};

class B: public A

{

public:

// note: no virtual keyword in B, C, and D

std::string\_view getName() const { return "B"; }

};

class C: public B

{

public:

std::string\_view getName() const { return "C"; }

};

class D: public C

{

public:

std::string\_view getName() const { return "D"; }

};

int main()

{

C c {};

B& rBase{ c }; // note: rBase is a B this time

std::cout << rBase.getName() << '\n';

return 0;

}

Answer:

C

Explanation:. Even though B and C aren’t marked as virtual functions, A::getName() is virtual and B::getName() and C::getName() are overrides. Therefore, B::getName() and C::getName() are considered implicitly virtual, and thus the call to rBase.getName() resolves to C::getName(), not B::getName().

13)Determine the output

#include <iostream>

#include <string\_view>

class A

{

public:

virtual std::string\_view getName() const { return "A"; }

};

class B: public A

{

public:

// Note: Functions in B, C, and D are non-const.

virtual std::string\_view getName() { return "B"; }

};

class C: public B

{

public:

virtual std::string\_view getName() { return "C"; }

};

class D: public C

{

public:

virtual std::string\_view getName() { return "D"; }

};

int main()

{

C c {};

A& rBase{ c };

std::cout << rBase.getName() << '\n';

return 0;

}

Answer: A

Explanation: This one is a little trickier. rBase is an A reference to a C object, so rBase.getName() would normally call A::getName(). But A::getName() is virtual, so it calls the most derived version of the function between A and C. And that is A::getName(). Because B::getName() and C::getName() are not const, they are not considered overrides! Consequently, this program prints A.

14) Determine the output

#include <iostream>

class Base

{

public:

~Base() // note: not virtual

{

std::cout << "Calling ~Base()\n";

}

};

class Derived: public Base

{

private:

int\* m\_array {};

public:

Derived(int length)

: m\_array{ new int[length] }

{

}

~Derived() // note: not virtual (your compiler may warn you about this)

{

std::cout << "Calling ~Derived()\n";

delete[] m\_array;

}

};

int main()

{

Derived\* derived { new Derived(5) };

Base\* base { derived };

delete base;

return 0;

}

Answer:

Calling ~Base()

Explanation:Because base is a Base pointer, when base is deleted, the program looks to see if the Base destructor is virtual. It’s not, so it assumes it only needs to call the Base destructor.

15)What is the problem in above program and How will you fix the above program

Answer: Memory leak is present

virtual ~Base() // note: virtual

Whenever you are dealing with inheritance, you should make destructors virtual. if the base class destructor isn’t marked as virtual, then the program is at risk for leaking memory

16) Can we change the hierarchy of operators by overloading them?

Answer: No

17) Give practical uses of “this” pointer

Answer:Use of “this” pointer in returning values from member functions and overloaded operators

We cant return object by value, since that creates extra copy of object, which means wastage of memory.

Also cant use reference returns on variables that are local to function since local variables gets destroyed when the function returns, This problems can be overcome using a “this” pointer

18)If there are 4 objects used in program how many this pointers would exist for these objects and why?

Answer: 4 this pointers, one this pointer for each object.

19)State True or false

When we traverse a BST in inorder we get the list in ascending order

Answer: True

20)Identify the problem and Fix them in below code

#include <iostream>

#include <memory> // for std::shared\_ptr

class Resource

{

public:

std::shared\_ptr<Resource> m\_ptr {}; // initially created empty

Resource() { std::cout << "Resource acquired\n"; }

~Resource() { std::cout << "Resource destroyed\n"; }

};

int main()

{

auto ptr1 { std::make\_shared<Resource>() };

ptr1->m\_ptr = ptr1; // m\_ptr is now sharing the Resource that contains it

return 0;

}

Answer: Cyclic references, destructor is never called and hence memory leak

Solution: Line 7: std::weak\_ptr<Resource> m\_ptr {}; // use std::weak\_ptr so m\_ptr doesn't keep the Resource alive

21) Write a program to check if number is palindrome

Palindrome: 1234321

bool isPalindrome(int number) noexcept

{

std::string strNum{std::to\_string(number)};

std::string revNum{strNum};

std::reverse(revNum.begin(), revNum.end());

return (strNum == revNum);

}

22)write a program to find factorial

long int factorial(int n) noexcept

{

long int factorial{1};

for(int i{1}; i<=n; ++i)

factorial \*= i;

return factorial;

}

23) Give the Output

void givetheOutput()

{

int a{5}, b{6};

int c {a>b ? a:b};

std::cout<<"c: "<<c<<std::endl;

return;

}

1.5

2.4

3. 7

4.6

Answer: 6

24) Give the Output

void givetheOutput()

{

int i;

int j{10};

i= (j++, j+100, 999+j);

std::cout<<i<<std::endl;

return;

}

1. 1001
2. 11
3. 1000
4. 1010

Answer: 1010, the value of j is incremented to 11, then j value is added to 100 but not assigned, and at last the value of j i.e 11 is added to 999 which gives us 1010.

### 25) What should be the correct statement about string objects in C++?

1. String objects should necessarily be terminated by a null character
2. String objects have a static size
3. String objects have a dynamic size
4. String objects use extra memory than required

Answer 3 - String objects have a dynamic size.

26) Why do we need pure virtual functions?

1. To make the base class abstract, so that it can't be instantiated, but a child class must override the pure virtual methods to form a concrete class. This is a good way to define an interface in C++. This forces a derived class to define the function.

2. Since the base class is abstract and cant be instantiated, it prevents accidental assignment of base class object to derived class object, and hence preventing the object slicing.

3.The virtual keyword gives C++ its' ability to support polymorphism. When you have a pointer to an object of some class such as:

class Animal

{

public:

virtual int GetNumberOfLegs() = 0;

};

class Duck : public Animal

{

public:

int GetNumberOfLegs() { return 2; }

};

class Horse : public Animal

{

public:

int GetNumberOfLegs() { return 4; }

};

void SomeFunction(Animal \* pAnimal)

{

cout << pAnimal->GetNumberOfLegs();

}

27)What is the difference between reference and pointer?

|  |  |
| --- | --- |
| Reference | Pointers |
| Reference is used to refer to an existing variable in another name | Pointers are used to store the address of a variable |
| References cannot have a null value assigned | The pointer can have a null value assigned |
| A reference variable can be referenced bypassing by the value | The pointer can be referenced but passed by reference |
| A reference must be initialized on the declaration | Pointers no need to be initialized on the declaration |
| A reference shares the same memory address with the original variable and takes up some space on the stack | Pointer has its own memory address and size on the stack |
| Cant be performed | Pointer arithmetic can be performed, like ptr++, ++ptr |

28)How to reverse a string in C++?

std::string str{“reverse”};

std::reverse( str.begin(), str.end());

29) What do you mean by a Move Constructor?

The Move Constructor and Move Assignment shifts the ownership of the resources from one object to other. This is less expensive than copy. A Move Constructor and Move Assignment operator is not provided by the compiler by default and one needs to implement the same.

30). What are class and object in C++?

A class is a user-defined data type that has data members and member functions. Data members are the data variables and member functions are the functions that are used to perform operations on these variables.

An object is an instance of a class. Since a class is a user-defined data type so an object can also be called a variable of that data type.

### 31. What are the static members and static member functions?

When a variable in a class is declared static, space for it is allocated for the lifetime of the program. No matter how many objects of that class have been created, there is only one copy of the static member. So same static member can be accessed by all the objects of that class.

A static member function can be called even if no objects of the class exist and the static function are accessed using only the class name and the scope resolution operator ::

32) Consider the following C++ program

#include<iostream>

class A

{

public:

virtual void a()=0;

A(){ std::cout<<"A "; }

};

class B: public A

{

public:

B(){ std::cout<<"B "; }

};

int main()

{

A \*a=new B();

}

What will be output?

A B

B A

Compile-time error

None of the above

Answer: Compile-time error

33) How will you correct above program

Add implementation for virtual method in class B.

void a() { std::cout<<"a() B "<<std::endl;; }

Fix resource leak

delete a;

34)Memory used by an array is

Contiguous

Non-contiguous

Not determined

None of the above

Answer)Contiguous

35)Which of the following statement is correct?

An object is an instance of the class

A friend function can access private members of a class

Members of the class are private by default

All of the above

35. Discuss the difference between new and malloc

new malloc

new is an operator malloc() is a function

It calls the constructor The malloc function doesn’t call the constructor

There is no need to specify memory You have to size while using new() specify the memory size

new operator can be overloaded malloc() can never be overloaded

### 36. Which of the following is not a member of a class?

1. Static function
2. Virtual function
3. Const function
4. Friend function

4 - Among the following, friend function is not a member of the class

37) Which among the following statements is correct about the program given below?

int main()

{

int x{7};

int \*ptr = &x;

std::cout<<\*ptr;

return 0;

}

1. The output will be 14
2. The output will be 0
3. The output will be 1
4. The output will be 7

4 - Output will be 7.  Pointer p has the memory address of x, and you display the pointer with a dereference operator that will display the value 7.

38)What is the correct syntax for lambda expression in C++11?

(A) [capture](parameters)->return-type {body}  
(B) [parameters](capture)->return-type {body}  
(C) [capture][parameters]->return-type {body}  
(D) (capture)(parameters)->return-type {body}

Ans:A

39)Which keyword in C++ 11 is used to prevent a virtual function from being overridden?

(A) final  
(B) static  
(C) stop  
(D) friend

Ans: A

40)Which one is correct syntax to use C++11 features while compilation?

(A) g++ -std=c++-11 -o output.out source.cpp  
(B) g++ -std=c++11 -o output.out source.cpp  
(C) g++ -std=c++ 11 -o output.out source.cpp  
(D) g++ -std=cpp11 -o output.out source.cpp

Ans : B

41)

Is below program of C++ 11 valid & runs without compilation error?

#include <vector>

#include <iostream>

#include <algorithm>

using namespace std;

void print\_square(int i)

{

cout << i\*i << endl;

}

int main()

{

vector<int> v;

// vector gets filled

for\_each(v.begin(), v.end(), print\_square);

return 0;

}

(A) Yes  
(B) No  
(C) Depends on Compiler

Ans:A

42)Correct syntax for creating a thread using Function Pointer?

thread\_function is the name of function which need to be executed

in separate thread.

(A) std::thread threadObj(thread\_function);  
(B) std::thread threadObj(&thread\_function);  
(C) std::thread threadObj((void \*)thread\_function);  
(D) std::thread threadObj(thread\_function());

Ans:A

43)What is output of below program.

#include<thread>

int main()

{

std::thread thd([](){std::cout<<"Heloo thread\n"; });

thd.join();

thd.join();

std::cout<<”Done\n”;

return 0;

}

1. Heloo thread

2. Compile error

3. Crash

4. Heloo thread

Done

Answer: Crash

44)How to correct above program.

Answer:

if (thd.joinable())

thd.join()

if (thd.joinable())

thd.join()

45) Determine the output of the program

#include<thread>

#include<unistd.h>

int main()

{

std::thread thd([](){std::cout<<"Heloo thread\n"; });

sleep(1);

std::cout<<”Done\n”;

return 0;

}

1. Heloo thread

2. Compile error

3. Crash

4. Heloo thread

Done

46) What is output of the program,

#include<iostream>

#include<thread>

void createThread()

{

auto func = [](int& x)

{

std::cout<<"Heloo thread "<<x<<std::endl;

x=20;

std::cout<<"after modifying x "<<x<<std::endl;

};

int x{10};

std::thread thd(func, std::ref(x));

std::chrono::milliseconds duration{10};

std::this\_thread::sleep\_for(duration);

std::cout<<"Main thread "<<x<<std::endl;

if (thd.joinable())

thd.join();

}

int main()

{

createThread();

return 0;

}

Answer:

Heloo thread 10

after modifying x 20

Main thread 20

47)What are the 6 synchronizations primitive available in Multithreading?

Answer:  
They are as follows –

* Mutex
* Join
* Condition Variable
* Barriers
* Spin Lock
* Semaphore

48)What are the ways to create a thread in C++?

Answer:  
There are 4 ways of doing this which are as follows –

* Thread creation [using the function pointer](https://www.educba.com/function-pointer-in-c/)
* Thread creating using the function object
* Thread creating using lambda
* Thread creation using the member function

49)What do you understand by priority inversion terminology?

Answer:  
A higher priority thread must wait behind a lower priority thread in a case where the lower priority threads hold a lock-in which a higher priority thread is waiting. This is a case of priority inversion.

50) Small Program. Sail Ship: Consider 3 actors for this implementation. Captain, engine crew, and cleaners. Captain will be represented by main thread in your program. Captain can issue 3 commands.

Each role should be performed by different function to perform a given order.

Captain can order cleaning crew to clean,And Captain does not have to wait on this command until it is done.

Next two commands are full speed ahead and stop the engine commands. Captain has to wait until engine crew finish these commands to the next command.

Write small program which takes input from console and execute these commands .Input it integers1:cleaning, 2: Full speed, 3: for stop and 100 exit program. Other number: Invalid order

Answer:

#include<iostream>

#include<thread>

#include<memory>

class thread\_guard{

std::thread &t;

bool joinability;

public:

explicit thread\_guard(bool \_joinability, std::thread &\_t)

: t(\_t),

joinability(\_joinability)

{}

~thread\_guard()

{

if (t.joinable())

{

if (joinability)

t.join();

else

t.detach();

}

}

thread\_guard (const thread\_guard &t) = delete;

thread\_guard& operator= (const thread\_guard &t) = delete;

};

enum class Command: std::uint16\_t

{

INVALID\_COMMAND,

CLEAN=1,

FULL\_SPEED=2,

STOP\_ENGINE=3,

PROGRAM\_EXIT=100

};

Command InttoCommand(std::uint16\_t input)

{

Command retVal = Command::INVALID\_COMMAND;

switch(static\_cast<Command>(input))

{

case Command::CLEAN:

retVal = Command::CLEAN;

break;

case Command::FULL\_SPEED:

retVal = Command::FULL\_SPEED;

break;

case Command::STOP\_ENGINE:

retVal = Command::STOP\_ENGINE;

break;

case Command::PROGRAM\_EXIT:

retVal = Command::PROGRAM\_EXIT;

break;

default: retVal = Command::INVALID\_COMMAND;

break;

}

return retVal;

}

void cleanShip()

{

std::cout<<"Cleaning the ship id"<<std::this\_thread::get\_id()<<"\n";

std::this\_thread::sleep\_for(std::chrono::milliseconds(5000));

std::cout<<"Cleaning the ship done "<<"\n";

}

void fullSpeed()

{

std::cout<<"Ship on Full speed - id "<<std::this\_thread::get\_id()<<"\n";

std::this\_thread::sleep\_for(std::chrono::milliseconds(1000));

std::cout<<" Ship changed the speed done "<<"\n";

}

void stopEngine()

{

std::cout<<"Stop the Engine - id "<<std::this\_thread::get\_id()<<"\n";

std::this\_thread::sleep\_for(std::chrono::milliseconds(1000));

std::cout<<" Stop the Engine done "<<"\n";

}

class Captain

{

public:

Captain()

{}

bool issueCommand(const Command& command)

{

switch(command)

{

case Command::CLEAN:

{

std::thread crew\_thread(cleanShip);

bool joinability = false;

thread\_guard crew\_tg(joinability, crew\_thread);

}

return true;

case Command::FULL\_SPEED:

{

std::thread full\_speed\_thread(fullSpeed);

bool joinability = true;

thread\_guard full\_speed\_tg(joinability, full\_speed\_thread);

}

return true;

case Command::STOP\_ENGINE:

{

std::thread stop\_engine\_thread(stopEngine);

bool joinability = true;

thread\_guard stop\_engine\_tg(joinability, stop\_engine\_thread);

}

return true;

case Command::PROGRAM\_EXIT:

std::cout<<"Main: Docking the ship\n";

return false;

case Command::INVALID\_COMMAND:

default: std::cout<<"Invalid order\n";

return true;

}

}

};

void provideTheCommandCaptain()

{

std::cout<<"Captain please issue the command\n"

"1: Clean the ship\n"

"2: Full speed sailing\n"

"3: Stop the ship\n"

"100: End the program ship\n";

}

int main()

{

auto captain = std::make\_unique<Captain>();

provideTheCommandCaptain();

bool isSailing = true;

while(isSailing)

{

std::cout<<"Captian : Please give the command\n";

std::uint16\_t input;

std::cin>>input;

const auto& command = InttoCommand(input);

isSailing = captain->issueCommand(command);

}

return 0;

}

51. What is the purpose of Placement new operator

To create an object at a specific memory location and call its constructor.

There is no danger of memory allocation failure, as memory is already allocated, so it is useful in env with limited resource.

We need to call delete/dtor explicitly if needed

52.What is purpose of std::shared\_mutex

To allow multiple threads to read a shared resource simultaneously

mutable std::shared\_mutex mutex\_;

unsigned int read() const

{

[std::shared\_lock](http://en.cppreference.com/w/cpp/thread/shared_lock) lock(mutex\_);

return value\_;

}

53. C++17 feature provides mechanism for concurrent execution of tasks and is designed to manage thread pool

std::async

54. C++17 feature provides a way to perform llel execution of range based for loop

std::for\_each

wrong: std::mutex, std::shared\_mutex, std::condition\_variable

55.atomic operation mean in the context of multi threading

An operation that cannot be executed by multiple threads simultaneously.

Wrong: an operation that involves complex arithmetic calculations, that cannot be interrupted by other threads, that requires external synchronisation

56.What is functor (STL)

function that overloads the unary operator()

wrong: An adapter class used to customise container behaviour, member function of a class defined inside container,global function that takes 2 arguments.

57.what happens if a derived class does not define ctor and dtor.

Base class ctor and dtor are automatically used.

Wrong: an error occurs at runtime, program will not compile, the derived class inherits ctors and dtors from base class.

58. When is default ctor automatically provided in c++

when there are no ctors defined in class

wrong: When the class has atleast 1 parameterised ctor, when class contains a dtor, c++ does not provide a default ctor automatically.

59.

class Shape

{

public:

virtual void draw()

{

std::cout<<”Drawing a shape”<<std::endl;

}

virtual void draw() const

{

std::cout<<”Drawing a shape”<<std::endl;

}

};

class Circle: public Shape

{

public:

virtual void draw()

{

std::cout<<”Drawing a Circle”<<std::endl;

}

void draw() const override

{

std::cout<<”Drawing a Circle”<<std::endl;

}

};

int main()

{

Shape \*shapeptr = new Circle();

shapeptr→draw();

std::Vector<Shape\*> shapes{new Shape(), new Circle()};

for(const auto &n:shapes)

{

std::cout<<n->draw()<<” “;

}

return 0;

}

Drawing a Circle

60.

int main()

{

std::vector<int> val{1,3,5,7,9};

int tgt{4};

auto lower{std::lower\_bound(val.begin(), val.end(), tgt)};

auto upper{std::upper\_bound(val.begin(), val.end(), tgt)};

std::cout<<(lower-val.begin())<<” “ <<(upper-val.begin())<<std::endl;

return 0;

}

61.

std::atomic<int> counter{0};

void increment()

{

for(int i{0}; i< 10000; ++i)

++counter;

}

int main()

{

std::thread thd1(increment);

std::thread thd2(increment);

thd1.join();

thd1.join();

std::cout<<”counter”<<counter.load()<<std::endl;

std::set<int> nums{5,2,8,3,1};

auto it{nums.find(8)};

if ( it != nums.end())

{

nums.erase(it);

}

for(const auto &n:nums)

{

std::cout<<num<<” “;

}

std::vector<int> nums{3,7,1,9,2};

std::sort(val.begin(), val.end(), [](int a, int b){ return a> b;});

for(const auto &n:nums)

{

std::cout<<num<<” “;

}

}

class A

{

public:

virtual void foo()

{

std::cout<<”A foo”<<std::endl;

}

};

class B : public A

{

public:

void foo() override

{

std::cout<<”B foo”<<std::endl;

}

}

int main()

{

A \*a = new B;

B \*b = dynamic\_cast<B\*>(a);

if (b)

b→foo();

else

std::cout<<”invalid”;

delete a;

}

class vehicle

{

public:

virtual void draw()

{

std::cout<<”vehicle ”<<std::endl;

}

};

class car: public vehicle

{

public:

void draw()

{

std::cout<<”car”<<std::endl;

}

};

int main()

{

vehicle v;

car c;

vehicle \*ptr = &c;

ptr->draw();

}

* 1. Diagnol traversal of binary tree
  2. Pair, Number of repeated strings in string
  3. Solving 24-game
  4. Special prime numbers
  5. Credit card validation using luhn algorithm
  6. Finishing all the apples
  7. Museum visit
  8. Number of lights
  9. email and password validator
  10. Weird sum