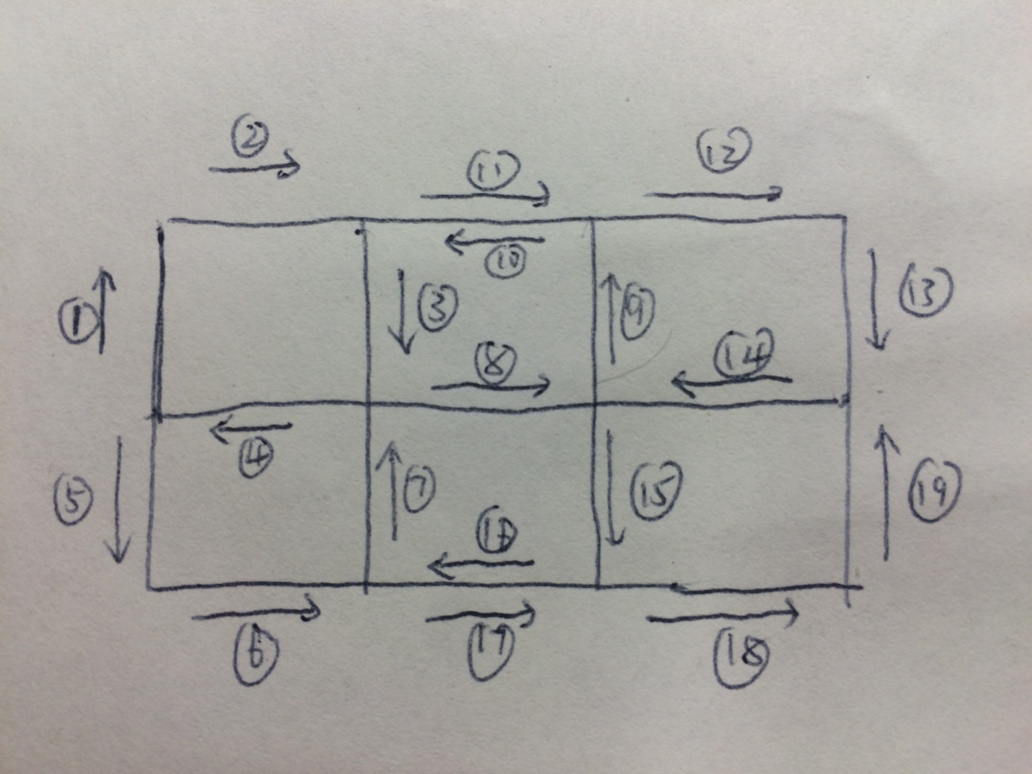
1 Math/Stat.

Q1.

Strictly speaking, test is any [statistical](https://en.wikipedia.org/wiki/Statistical) [hypothesis test](https://en.wikipedia.org/wiki/Hypothesis_test) in which the [sampling distribution](https://en.wikipedia.org/wiki/Sampling_distribution) of the test statistic is a [chi-square distribution](https://en.wikipedia.org/wiki/Chi-square_distribution) when the [null hypothesis](https://en.wikipedia.org/wiki/Null_hypothesis) is true. In most cases, test statistic arises from an assumption of independent normally distributed data, and a chi-squared test can then be used to reject the hypothesis that the data are independent.

Q2.

The answer is 19. We’ll first show that the shortest path cannot be either 17 or 18. If it is 17 then every edge is covered exactly once. In this situation, every vertex must have even degree except for the starting and end point. However, there are 6 vertices in the graph with odd degree – a conflict. If the shortest path is 18, then there is exactly one edge that is repeated and only repeated once. By replicating one edge in the graph, there will still be at least 4 vertices in the graph with odd degree. And by replicating, the 18 edges are covered exactly once. But this is impossible based on the same reasoning. Now we list one path with shortest path 19.

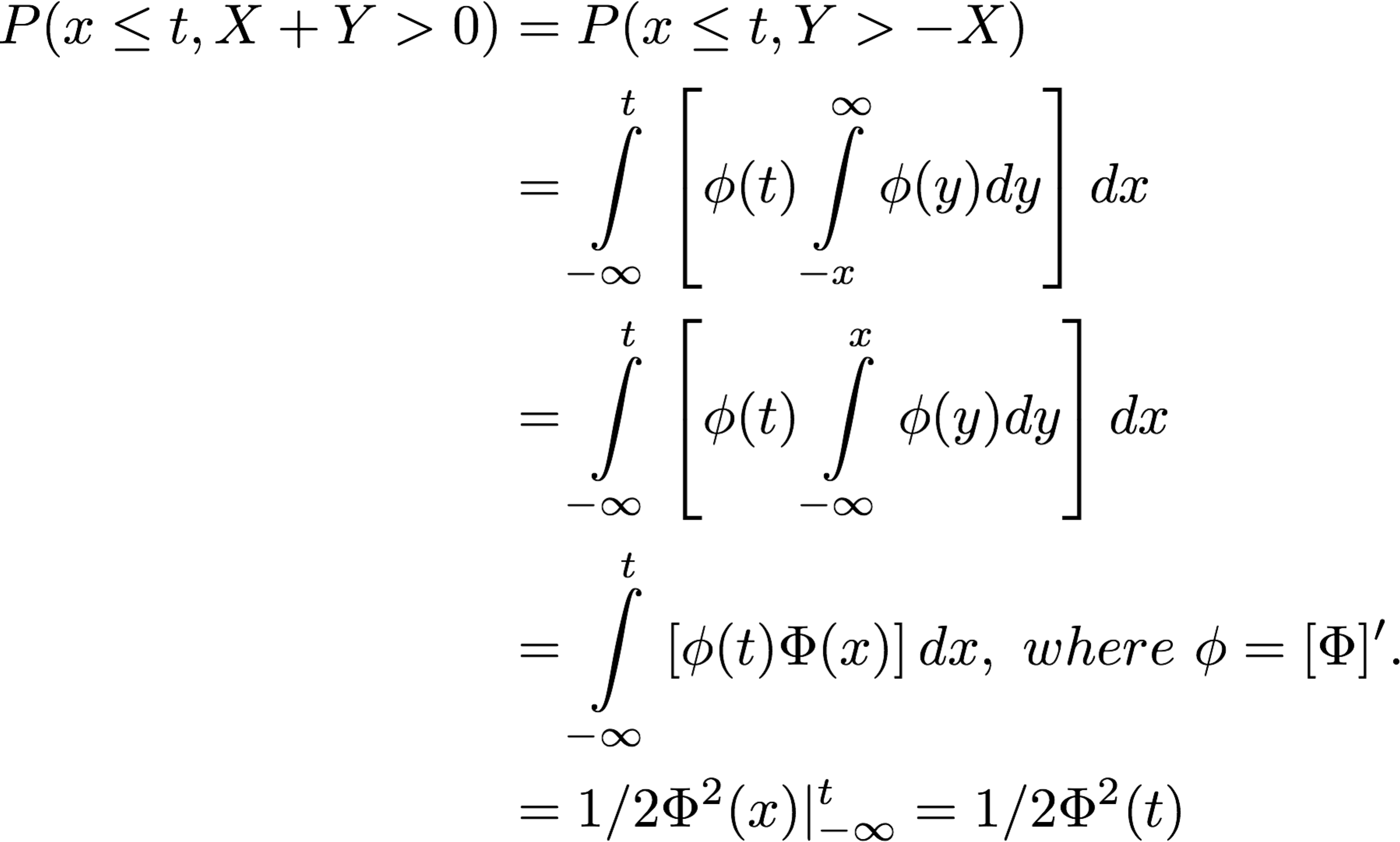


Q3.

X, Y are iid N(0,1), then for any t in R,

P(X<=t|X+Y>0)= P(X<=t, X+Y>0) /P(X+Y>0)

For the numerator,



For the denominator, since X+Y are N(0,2), P(X+Y>0)=1/2

Then where is the CDF of standard normal distribution

Q4.

(1) Whether to keep rolling or stop depends on whether the expected gain of keep rolling is larger than zero. If next is 1, the gain is -35, otherwise we could safely reach 43 or above, corresponding to a gain of 8 or larger. Hence the expected gain of keep rolling is larger than -35/6 + 8 \* 5/ 6 >0. Therefore, it’s better to keep rolling.

(2) The 1st rolling could be 2, 3, 4, 5, 6. Then we consider the most likely value when we stop after the first rolling.

If the 1st rolling is 2, we continue with at least another two rollings, and the most probable outcome of two rollings’ sum is 7 ( = 1+6 = 2+5 = 3+4 = 4+3 = 5+2 = 6+1, 6 cases reach 7), giving a total stop value at 35+2+7 = 44.

if the 1st rolling is 3, we would stop if 2nd rolling is 6, with total value reaching 44. Otherwise, we continue with at least another two rolling, with the most probable outcome at 7 or 6. Hence, the most likely ending value is still 44.

if the 1st rolling is 4, we would stop if 2nd rolling is 5 or 6, with total value reaching 44 or 45. Otherwise, we continue with another two rollings, which gives the most probable value at 7 or 6 or 5. Hence the most likely ending value is 44 or 45. By the same token, any other case all have the most probable ending value at 44. Therefore the 44 is the most probable outcome.

Q5. The solution is the same as it is in problem 43 The Broken Bar in the book Fifty Challenging Problems in Probability and Solutions.

Q6.

There are N! possible permutation of the N persons 1,2,…,N with equal probability, hence each permutation has probability 1/N!. Denote Xi as the hat that the ith person gets.

PN (X1=1, X2=2, …, XN-2=N-2, XN-1=N-1, XN=N) = 1/N!

PN (X1=1, X2=2, …, XN-2=N-2 ,XN-1=N-1) =1/N!

PN (X1=1, X2=2, …, XN-2=N-2) = 2!/N!

……

PN (X1=1) = (N-1)!/N!

1. E(Y) = E (sum(i=1 to N) 1{Xi=i} ) = N\*P(Xi=i) =N\* (N-1)!/N!=1
2. E(Y^2)= E(sum(i=1 to N) 1{Xi=i} )^2

= sum(i=1 to N) 1{Xi=i} + 2\*sum(1<=i<j<=N) 1{Xi=i, Xj=j}

= E(Y) +2 \* C(N,2) \* P(Xi=i, Xj=j) =1+N(N-1) \* (N-2)!/N!=1+1=2

Then Var(Y) =E(Y^2) –[E(Y)]^2 =1.

Since RN = Y(N)+RN-Y(N)

E(RN) = E(Y(N)) +E(RN-Y(N)).

E(RN) = 1+ P(Y(N)=0) \* E(RN) + P(Y(N)=1) \* E(RN-1) +…… +P(Y(N)=N-1) \* E(R1)

We want to get P(Y(N)=k) ==denoted as== PN(Y=k).

PN (Y=0) = 1 - PN (Y>=1).

Since PN (Y>=1) = PN ( {X1=1} U {X2=2} U {X3=3} U…U {XN-1=N-1}… U {XN=N} )

= Sum(i=1 to N) PN ({Xi=si}) –Sum(1<=i<j<=n) PN ({Xi=i}\*{Xj=j}) +Sum(1<=i<j<k<=n)PN ({Xi=i}\*{Xj=j}\*{Xk=k})-… +(-1)^N Sum(1<=i1<i2<…<iN-1<=N) PN ({Xi1=i1}\*{Xi2=i2}\*….\*{XiN-1=iN-1})

+(-1)^(N+1) Sum(1<=i1<i2<…<iN<=N) PN ({Xi1=i1}\*{Xi2=i2}\*….\*{XiN-1=iN-1}\*{XiN=iN})

=C(N,1)\*PN (X1=1)-C(N,2)\*PN (X1=1,X2=2)+C(N,3)\*PN (X1=1,X2=2,X3=3)-… +(-1)^N\*C(N,N-1)\*PN (X1=1, X2=2, …, XN-2=N-2 ,XN-1=N-1) +(-1)^(N+1)\* C(N,N)\*PN (X1=1, X2=2, …, XN-2=N-2, XN-1=N-1, XN=N)

=C(N,1)\* (N-1)!/N! –C(N,2)\* (N-2)!/N!+C(N,3)\* (N-3)!/N! -…+(-1)^N\*C(N,N-1)\*1!/N! +(-1)^(N+1)\* C(N,N)\*0!/N!

=1/1!-1/2!+1/3!-…. +(-1)^N\*1/(N-1)! +(-1)^(N+1)\*1/N!

We have PN (Y=0) = 1 - PN (Y>=1) = 1/2!-1/3!+…. +(-1)^(N-1)\*1/(N-1)! +(-1)^N\*1/N!

Consider Y=k (1<=k<=N-2), which could be thought of as the first k persons out of N can take their own hats, while for the other N-k, none can get his own hat.

PN (Y=k) = 1/k! \* PN-k (Y=0) =1/k! \*[ 1/2!-1/3!+…. +(-1)^(N-k)\*1/(N-k)! ]

For k>=N-1, we can easily get PN (Y=N-1)=0, P(Y=N)=1/N!

So E(RN) = 1+ (1/2!-1/3!+…. +(-1)^(N-1)\*1/(N-1)! +(-1)^N\*1/N!) \* E(RN)

+1/1!\*(1/2!-1/3!+…. +(-1)^(N-1)\*1/(N-1)!) \* E(RN-1)

+1/2!\*(1/2!-1/3!+…. +(-1)^(N-2)\*1/(N-2)!) \* E(RN-2)

+……

+1/k!\*(1/2!-1/3!+…. +(-1)^(N-k)\*1/(N-k)!) \* E(RN-k)

+……

+1/(N-3)!\*(1/2!-1/3!) \* E(R3)

+1/(N-2)!\*(1/2!) \* E(R2)

Since E(R1)=1,

E(R2)=1+1/2!\*E(R2) => E(R2)=2.

E(R3)=1+(1/2!-1/3!)\*E(R3)+1/2!\*E(R2) => E(R3)=3.

We try mathematical induction method. Assume E(Rk)=k for all k<=m, then

E(Rm+1)= 1+(1/2!-1/3!+…. +(-1)^m\*1/m! +(-1)^(m+1)\*1/(m+1)!) \* E(Rm+1)

+1/1!\*(1/2!-1/3!+…. +(-1)^m\*1/m!) \* m

+1/2!\*(1/2!-1/3!+…. +(-1)^(m-1)\*1/(m-1)!) \* (m-1)

+……

+1/k!\*(1/2!-1/3!+…. +(-1)^(m+1-k)\*1/(m+1-k)!) \* (m+1-k)

+……

+1/(m+1-3)!\*(1/2!-1/3!) \* 3

+1/(m+1-2)!\*(1/2!) \* 2

Reorganize the equation, and by calculation, we get E(Rm+1) =m+1.

So E(Rk)=k for all k<=m+1.

Then by mathematical induction method, we get E(RN)=N for all N>=1.

1. Since SN = N+SN-Y(N)

E(SN) = N+E(SN-Y(N)).

E(SN) = N+ P(Y(N)=0) \* E(SN) + P(Y(N)=1) \* E(SN-1) +…… +P(Y(N)=N-1) \* E(S1)

Note that E[ SN-N(N-1)/2 ] – E[ SN-Y(N)- (N-Y(N)) (N-Y(N)-1)/2 ]

=E[ SN- SN-Y(N) ] –N(N-1)/2 + N(N-1)/2 + ½ \* E[Y^2(N) -2N\*Y(N) +Y(N) ]

=N+1/2\* [1-2N+1]=1

So E[ SN-N(N-1)/2 ] = 1+ E[ SN-Y(N)- (N-Y(N)) (N-Y(N)-1)/2 ].

So E[ SN-N(N-1)/2 ] has the same property as that of E[RN].

Then E[ SN-N(N-1)/2 ] = E[RN]=N.

So E[ SN ]= N(N-1)/2+N = N(N+1)/2.

1. The expected number of false selections for each of the N persons

1/N\* E(SN)-1 = 1/N\* N(N+1)/2 – 1 = (N-1)/2.

Thanks to example 1.3(a) and 1.5(e) in the book “Stochastic Processes” by S.M. Ross.

Macintosh HD:Users:XIAOYI:Google Drive:琪石第五期 Quant 推进班学员文档:C组（组长：谢晓宜）:Qishi Quiz:Quiz2:Quiz2_7.pdf

Macintosh HD:Users:XIAOYI:Desktop:Quiz2_7_2.pdf

Q8.

We only have to generate numbers 0-(n-1) with equal probability. Let 2^k <= n-1 < 2^{k+1}, we toss the coin k times and use the result to make a k bit binary number (head as 1, tail as 0). If the number is >= n, discard this number and generate again until we have some number <n. This random number generator can produce numbers from 0 to n-1 with equal probability 1/n.

Q9.

Denote as the expected number of acrossed bridges starting from island .

Easy to obtain from island 1, we have:

On the other hand, we have:

By solving above two equations, we have

2 Programming.

Q10.const function

Input: reference to ( a const pointer to a const integer) it is a reference to pointer

Output: const pointer to a const integer

Q11.





Q12.













Q13.

The answers are found in the following two FAQs at https://isocpp.org/wiki/faq/virtual-functions

### What is a “virtual constructor”?

An idiom that allows you to do something that C++ doesn’t directly support. You can get the effect of a virtual constructor by a virtual clone() member function (for copy constructing), or a virtual create() member function (for the [default constructor](https://isocpp.org/wiki/faq/ctors#default-ctor)).

1. class Shape {
2. public:
3. virtual ~Shape() { } *// A virtual destructor*
4. virtual void draw() = 0; *// A pure virtual function*
5. virtual void move() = 0;
6. *// ...*
7. virtual Shape\* clone() const = 0; *// Uses the copy constructor*
8. virtual Shape\* create() const = 0; *// Uses the default constructor*
9. };
10. class Circle : public Shape {
11. public:
12. Circle\* clone() const; *// Covariant Return Types; see below*
13. Circle\* create() const; *// Covariant Return Types; see below*
14. *// ...*
15. };
16. Circle\* Circle::clone() const { return new Circle(\*this); }
17. Circle\* Circle::create() const { return new Circle(); }

In the clone() member function, the new Circle(\*this) code calls Circle’s copy constructor to copy the state of this into the newly created Circle object. (Note: unless Circle is known to be [final (AKA a leaf)](https://isocpp.org/wiki/faq/strange-inheritance#final-classes), you can reduce the chance of [slicing](https://isocpp.org/wiki/faq/value-vs-ref-semantics#pass-by-value) by making its copy constructor protected.) In the create() member function, the new Circle() code calls Circle’s [default constructor](https://isocpp.org/wiki/faq/ctors#default-ctor).

Users use these as if they were “virtual constructors”:

1. void userCode(Shape& s)
2. {
3. Shape\* s2 = s.clone();
4. Shape\* s3 = s.create();
5. *// ...*
6. delete s2; *// You need a virtual destructor here*
7. delete s3;
8. }

This function will work correctly regardless of whether the Shape is a Circle, Square, or some other kind-of Shape that doesn’t even exist yet.

Note: The return type of Circle’s clone() member function is intentionally different from the return type of Shape’s clone() member function. This is called Covariant Return Types, a feature that was not originally part of the language. If your compiler complains at the declaration of Circle\* clone() const within class Circle (e.g., saying “The return type is different” or “The member function’s type differs from the base class virtual function by return type alone”), you have an old compiler and you’ll have to change the return type to Shape\*.

**Why don’t we have virtual constructors?**

A virtual call is a mechanism to get work done given partial information. In particular, virtual allows us to call a function knowing only an interfaces and not the exact type of the object. To create an object you need complete information. In particular, you need to know the exact type of what you want to create. Consequently, a “call to a constructor” cannot be virtual.

Techniques for using an indirection when you ask to create an object are often referred to as “Virtual constructors”. For example, see [TC++PL3](http://stroustrup.com/3rd.html) 15.6.2.

For example, here is a technique for generating an object of an appropriate type using an abstract class:

1. struct F { *// interface to object creation functions*
2. virtual A\* make\_an\_A() const = 0;
3. virtual B\* make\_a\_B() const = 0;
4. };
5. void user(const F& fac)
6. {
7. A\* p = fac.make\_an\_A(); *// make an A of the appropriate type*
8. B\* q = fac.make\_a\_B(); *// make a B of the appropriate type*
9. *// ...*
10. }
11. struct FX : F {
12. A\* make\_an\_A() const { return new AX(); } *// AX is derived from A*
13. B\* make\_a\_B() const { return new BX(); } *// BX is derived from B*
14. };
15. struct FY : F {
16. A\* make\_an\_A() const { return new AY(); } *// AY is derived from A*
17. B\* make\_a\_B() const { return new BY(); } *// BY is derived from B*
18. };
19. int main()
20. {
21. FX x;
22. FY y;
23. user(x); *// this user makes AXs and BXs*
24. user(y); *// this user makes AYs and BYs*
25. user(FX()); *// this user makes AXs and BXs*
26. user(FY()); *// this user makes AYs and BYs*
27. *// ...*
28. }

This is a variant of what is often called “the factory pattern”. The point is that user() is completely isolated from knowledge of classes such as AX and AY.

Q14.

The answer is found as a FAQ here : https://isocpp.org/wiki/faq/strange-inheritance#calling-virtuals-from-base

Q16 How to inverse a string of sentence (without reverse the word) ?



Q17



Q18.

Analysis and code for this problem can be found at <http://blog.csdn.net/skyworth0103/article/details/38472639>. The problem is about how to construct Hamiltonian cycle given ORE condition (which is a sufficient condition on the existence of Hamiltonian cycle).