学习记录

JiangYiFu

2024 年 5 月 8 日

前言

JiangYiFu in USTC

2024 年 5 月 8 日

目录

第一章　函数极限与连续

1.1　如何求解函数方程

设函数 $f(x)$ 的定义域为 $(0,\infty)$，且满足 $2f(x)+x^2f(\frac{1}{x})=\frac{x^2+2x}{\sqrt{1+x^2}}$，则 $f(x)=$_____。

key: 轮换对称式，高中题目常见

Answer:

$$f(x)=\frac{x}{\sqrt{1+x^2}}$$

1.2　反函数

1. 严格单调函数必有反函数。

2. 有反函数的不一定是单调函数。

求函数 $y=f(x)=ln(x+\sqrt{x^2+1})$ 的反函数 $f^{-1}(x)$ 的表达式及其定义域.

key: 注意到代数变形

$$\frac{1}{x+\sqrt{x^2+1}}=\sqrt{x^2+1}-x$$

Answer:

$$f^{-1}(x)=\frac{e^x-e^{-x}}{2}$$

1.3　复合函数的奇偶性

$f[g(x)]$（内偶则偶，内奇同外）

设对任意 x，y，都有 $f(x+y)=f(x)+f(y)$，证明：$f(x)$ 是奇函数.

key: 对 x,y 赋值

$$f(x)=\frac{2^x-1}{2^x+1}, x\in\mathbb{R}$$

计算

$$\int_{-1}^{1}\frac{1}{2^x+1}dx$$

key: 注意到 $f(x)$ 是奇函数，且

$$\frac{1}{2^x+1}=\frac{1}{2}\frac{(2^x+1)-(2^x-1)}{2^x+1}$$

1.4　三角函数

$$sec(x)=\frac{1}{cos(x)}, csc(x)=\frac{1}{sin(x)}$$

$$1+tan(\alpha)^2=sec(\alpha)^2$$

1.5　函数极限定义

“$\varepsilon-\delta$”语言：$\lim\limits_{x\to x_0}f(x)=A \iff \forall\varepsilon>0,\exists\delta>0$, 当 $0<|x-x_0|<\delta$ 时，有 $|f(x)-A|<\varepsilon$.

函数极限存在的充要条件：$\lim\limits_{x\to x_0} f(x) = A \iff \lim\limits_{x\to x_0^-} f(x) = A$ 且 $\lim\limits_{x\to x_0^+} f(x) = A$

1.6　极限与有界性

极限的局部有界性（常用于证明题）：如果 $\lim\limits_{x\to x_0} f(x) = A$, 则存在正常数 M 和 δ，使得当 $0 < |x - x_0| < \delta$ 时，有 $|f(x)| \le M$.

若 $y = f(x)$ 在 $[a, b]$ 上为连续函数，则 $f(x)$ 在 $[a, b]$ 上必定有界.

若 $f(x)$ 在 (a, b) 内为连续函数，且 $\lim\limits_{x\to a^+} f(x)$ 与 $\lim\limits_{x\to b^-} f(x)$ 都存在，则 $f(x)$ 在 $[a, b]$ 内必定有界.

在下列区间内，函数

$$f(x) = \frac{xsin(x-3)}{(x-1)(x-3)^2}$$

有界的是（　）

A.$(-2, 1)$　B.$(-1, 0)$　C.$(1, 2)$　D.$(2, 3)$

key: 函数有界问题 $\iff$ 极限存在问题

Answer:B

局部保号性（*）

1.7　极限：“脱帽法”和“戴帽法”

$$\begin{cases} \lim f > 0 \Rightarrow f > 0 \\ \lim f < 0 \Rightarrow f < 0 \\ f \geq 0 \Rightarrow \lim f \geq 0 \\ f \leq 0 \Rightarrow \lim f \leq 0 \end{cases}$$

1.8　无穷小的比阶

并不是任意两个无穷小都可进行比阶.

1.9　极限计算

极限四则运算规则（*）

若 $\lim \frac{f(x)}{g(x)} = A$, 且 $\lim g(x) = 0$, 则 $\lim f(x) = 0$.
若 $\lim \frac{f(x)}{g(x)} = A \neq 0$, 且 $\lim f(x) = 0$, 则 $\lim g(x) = 0$.

设 $\lim\limits_{x\to 0} \frac{sinx}{e^x - a}(cosx - b) = 5$，那么 $b =$ _____.
Answer:-4

1.10　洛必达法则

法则一：设

1. 当 $x \to a$（或 $x \to \infty$）时，函数 $f(x)$ 及 $F(x)$ 都趋于零.

2.$f^{'}(x)$ 及 $F^{'}(x)$ 在点 a 的某去心邻域内（或当 $|x| > X$，此时 x 为充分大的正数）存在，且 $F^{'}(x) \neq 0$.

3.$\lim\limits_{x\to a}\frac{f^{'}(x)}{F^{'}(x)}$(或 $\lim\limits_{x\to\infty}\frac{f^{'}(x)}{F^{'}(x)}$) 存在或为无穷大，则

$$\lim_{x\to a}\frac{f(x)}{F(x)}=\lim_{x\to a}\frac{f^{'}(x)}{F^{'}(x)}$$

或

$$\lim_{x\to \infty}\frac{f(x)}{F(x)}=\lim_{x\to \infty}\frac{f^{'}(x)}{F^{'}(x)}$$

注意：这是一个后验逻辑命题

法则二：设

1. 当 $x\to a$（或 $x\to\infty$）时，函数 $f(x)$ 及 $F(x)$ 都趋于无穷大.

2.$f^{'}(x)$ 及 $F^{'}(x)$ 在点 a 的某去心邻域内（或当 $|x|>X$，此时 x 为充分大的正数）存在，且 $F^{'}(x)\neq 0$.

3.$\lim\limits_{x\to a}\frac{f^{'}(x)}{F^{'}(x)}$(或 $\lim\limits_{x\to\infty}\frac{f^{'}(x)}{F^{'}(x)}$) 存在或为无穷大，则

$$\lim_{x\to a}\frac{f(x)}{F(x)}=\lim_{x\to a}\frac{f^{'}(x)}{F^{'}(x)}$$

或

$$\lim_{x\to \infty}\frac{f(x)}{F(x)}=\lim_{x\to \infty}\frac{f^{'}(x)}{F^{'}(x)}$$

1.11　泰勒公式

设 $f(x)$ 在点 $x=0$ 处 n 阶可导，则存在 $x=0$ 的一个邻域，对于该邻域内的任一点 x，有

$$f(x)=f(0)+f^{'}(0)x+\frac{f^{''}(0)}{2!}x^2+...+\frac{f^{(n)}(0)}{n!}x^n+o(x^n).$$

常见结论：

$$sin(x) = x - \frac{x^3}{3!} + o(x^3)$$

$$cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} + o(x^4)$$

$$arcsin(x) = x + \frac{x^3}{3!} + o(x^3)$$

$$tan(x) = x + \frac{x^3}{3} + o(x^3)$$

$$arctan(x) = x - \frac{x^3}{3} + o(x^3)$$

$$ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} + o(x^3)$$

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + o(x^3)$$

$$(1+x)^\alpha = 1 + \alpha x + \frac{\alpha(\alpha-1)}{2!}x^2 + o(x^2)$$

1.12　两个重要极限

$$\lim_{x\to 0} \frac{sin(x)}{x} = 1.$$

$$\lim_{x\to \infty} (1 + \frac{1}{x})^x = e.$$

1.13　夹逼准则

如果函数 $f(x)$,$g(x)$ 及 $h(x)$ 满足下列条件：

（1）$h(x) \leq f(x) \leq g(x)$.

（2）$\lim g(x) = A, \lim h(x) = A$.

则 $\lim f(x)$ 存在，且 $\lim f(x) = A$.

1.14　极限计算例题

$$\lim_{x\to 0^+} \frac{(1+x)^{\frac{1}{x}} - e}{x} = \_\_\_\_$$

key:$u^v = e^{vln(u)}$，$e^x - 1 \sim x$

Answer:$-\frac{1}{2}e$

注意：记忆 $f(x) = (1+x)^{\frac{1}{x}}$ 和 $f(x) = (1+\frac{1}{x})^x$ 的函数图像

设函数

$$f(x) = \lim_{n\to\infty} \frac{x^2 + nx(1-x)sin^2\pi x}{1 + nsin^2\pi x}$$

则 $f(x) =$ ____.

key:$n \to \infty$ 专指 $n \to +\infty$，将 x 视作常数

Answer:

$$f(x) = \begin{cases} x^2, & x = k \\ x(1-x), & x \neq k \end{cases} (k \in Z)$$

求极限

$$\lim_{x\to 1^-} ln(x)ln(1-x).$$

key: 等价代换后换元，转换为

$$\lim_{x\to 0^+} x^{\alpha} ln(x) = \lim_{x\to 0^+} \frac{ln(x)}{x^{-\alpha}}$$

求极限

$$\lim_{x\to +\infty} [x^2(e^{\frac{1}{x}} - 1) - x].$$

key: 代数变形，减法变乘法，换元

Answer:$\frac{1}{2}$

求极限

$$\lim_{x\to +\infty} (x + \sqrt{1+x^2})^{\frac{1}{x}}$$

key: 幂指函数

Answer:1

1.15　函数的连续和间断

连续点的定义：

设函数 $f(x)$ 在点 x_0 的某一邻域内有定义，且有 $\lim\limits_{x\to x_0} f(x) = f(x_0)$, 则称函数 $f(x)$ 在点 x_0 处连续.

考察本质是极限的计算.

连续判定：

$$\lim_{x\to x_0^+} f(x) = \lim_{x\to x_0^-} f(x) = f(x_0) \iff f(x)\ is\ coiled\ in\ x_0$$

连续性四则运算法则（*）

间断点的定义和分类：

1. 可去间断点（*）

2. 跳跃间断点（*）

3. 无穷间断点（*）

4. 振荡间断点（*）

第二章　数列极限

求极限

$$\lim_{n\to\infty}(\frac{1}{1\cdot 2}+\frac{1}{2\cdot 3}+\cdots+\frac{1}{n\cdot(n+1)})^n$$

key:1^∞ 形式，考虑 $\lim u^v=\lim e^{vln(u-1)}$

Answer:e^{-1}

2.1　数列极限定义

设 $\{x_n\}$ 为一数列，若存在常数 a，对于任意的 $\epsilon>0$，总存在正整数 N，使得当 $n>N$ 时，$|x_n-a|<\epsilon$ 恒成立，则称常数 a 是数列 $\{x_n\}$ 的极限，或者称数列 $\{x_n\}$ 收敛于 a.

2.2　数据收敛与其子列收敛

若数列a_n 收敛，则其任何子列a_{n_k} 也收敛，且

$$\lim_{k\to\infty}a_{n_k}=\lim_{n\to\infty}a_n$$

.

通过该结论判断数列发散

1. 找到一个发散子列

2. 找到两个收敛子列，但它们收敛到不同极限。

2.3　收敛数列的性质

唯一性（*）

有界性（*）

保号性（*）

已知 $a_n = 1 - \frac{(-1)^n}{n}$，则$a_n$ 是否有最大值和最小值?

key: 保号性

Answer: 有，有

2.4　极限四则运算规则

四则运算规则可以推广至有限个数列情形.

2.5　海涅定理（归结原则）

$$\lim_{x\to a} f(x) = L \iff \forall\{a_n\}, if \lim_{n\to\infty} a_n = a, have \lim_{n\to\infty} f(a_n) = L$$

海涅定理是联系数列极限和函数极限的桥梁.

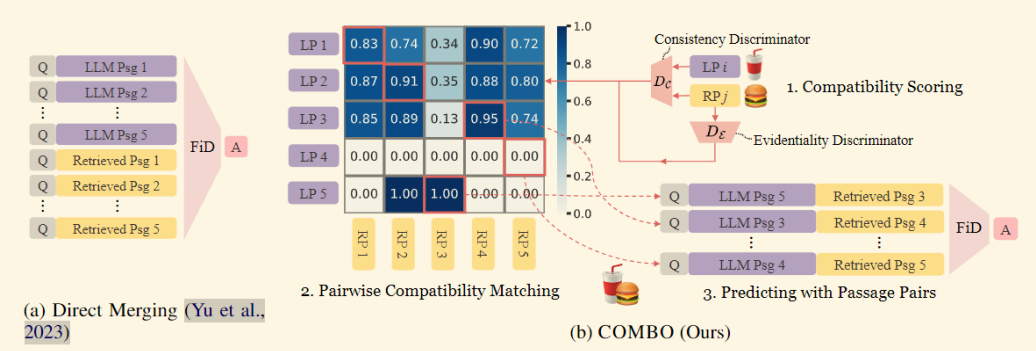
第三章　Article-Reading

3.1　COMBO 框架

Merging Generated and Retrieved Knowledge for Open-Domain QA

目标问题：COMBO 框架旨在 1. 利用 LLM 生成的相关知识：通过 LLM 的参数化知识来补充检索到的知识，以提高答案的相关性。2. 解决知识冲突：通过匹配兼容的段落对来减少 LLM 生成内容中的虚构信息对模型性能的负面影响。

设计原则：COMBO 框架基于一个关键直觉，即如果答案得到两种信息源（检索到的知识和 LLM 生成的知识）的支持，则更有可能是正确的。因此，框架旨在通过将检索到的和生成的段落配对成兼容对来利用这两种信息源。



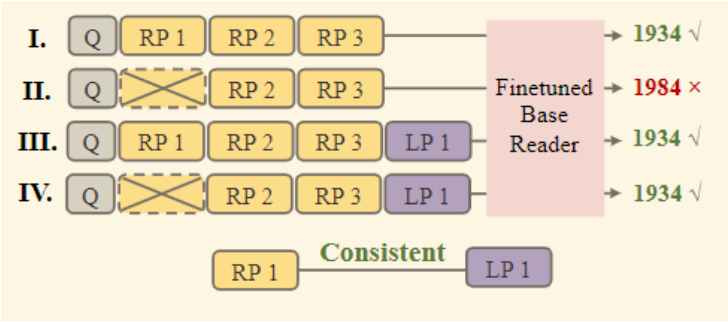
1.Compatibility Scoring：兼容性评分

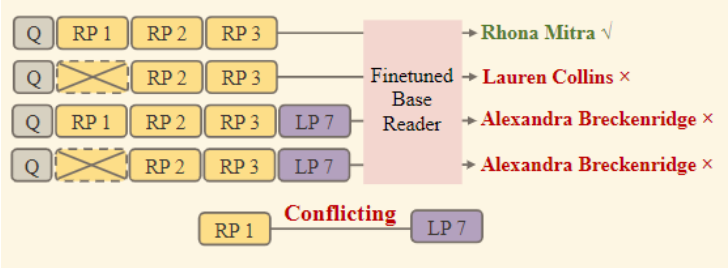
$$\overbrace{P(lp_i \vDash Q, rp_j \vDash Q)}^{\text{compatibility score}} = \underbrace{P(rp_j \vDash Q)}_{\text{evidentiality score}} \cdot \underbrace{P(lp_i \vDash Q \mid rp_j \vDash Q)}_{\text{consistency score}}. \quad (1)$$

这一步涉及计算 LLM 生成的段落（称为“LP”，即“LLM-generated Passage”）和检索到的段落（称为“RP”，即“Retrieved Passage”）之间的兼容性分数。

“Evidentiality Discriminator”评估检索到的段落是否包含与问题相关的正确证据，其实现原理为: 通过评估 QA 模型在包含或排除某个检索段落时的预测准确性变化，来自动识别该段落是否包含支持问题答案的关键证据，从而为该段落分配“事实性”标签

“Consistency Discriminator”评估 LLM 生成的段落与检索到的段落是否一致，即它们是否提供相互支持的证据，其实现原理为：





2.Pairwise Compatibility Matching：成对兼容性匹配

在计算了兼容性分数后，COMBO 框架将 LLM 生成的段落和检索到的段落配对，以最大化整体兼容性。配对的段落对用红框突出显示，表示它们是兼容的。

其原理为：兼容性分数用于构建一个二分图，其中每个 LP 和 RP 都是图中的节点。节点之间的边的权重是对应的 LP 和 RP 的兼容性分数。成对兼容性匹配的目标是在这个加权二分图中找到一个最大加权匹配，即选择一组段落对，使得它们的兼容性分数之和最大，考虑匈牙利算法实现。

3.Predicting with Passage Pairs：预测与段落对

配对的段落对根据它们的兼容性分数进行排序,然后输入到基于 Fusion-in-Decoder（FiD）的阅读器模型中，以产生最终答案。FiD 模型能够通过自注意力机制处理多个段落，并对每个段落赋予不同的注意力权重，从而整合证据并生成答案。

第四章　CS106L

4.1　Types and Structs

The STL

- Tons of general functionality
- Built in classes like maps, sets, vectors
- Accessed through the namespace std::
- Extremely powerful and well-maintained

STL = Standard Template Library

```
#include <string>
int val = 5; //32 bits (usually)
char ch = 'F'; //8 bits (usually)
float decimalVal1 = 5.0; //32 bits (usually)
double decimalVal2 = 5.0; //64 bits (usually)
bool bVal = true; //1 bit
std::string str = "Haven";
```

Fill in the blanks!

```
std::string a = "test";
_____ b = 3.2 * 5 - 1;
_____ c = 5 / 2;

_____ d(int foo) { return foo / 2; }
_____ e(double foo) { return foo / 2; }
_____ f(double foo) { return int(foo / 2); }

_____ g(double c) {
    std::cout << c << std::endl;
}
```

4.1.1 Function overloading

In C++, you cannot define multiple identical functions.But what if we want two versions of a function for two different types?—Function overloading.

Function Overloading = Define two functions with the same name but different types.

```
int half(int x) {
    std::cout << "1" << endl; // (1)
    return x / 2;
}

double half(double x) {
    cout << "2" << endl; // (2)
    return x / 2;
}

half(3) // uses version (1), returns 1
half(3.0) // uses version (2), returns 1.5
```

The principle for selecting the function to call in C++ function over-

loading is as follows:

1.The compiler first looks for an exact match between the arguments passed in the function call and the parameters of each overloaded function. If an exact match is found, that function is selected.

2.If an exact match is not found, the compiler tries to find a function where the arguments can be implicitly converted to match the parameter types. This includes promotions, such as converting an int to a double.

3.If no exact match or promotion is found, the compiler considers standard conversions, such as integral promotions and conversions between arithmetic types.

4.If the previous steps do not yield a match, the compiler checks for user-defined conversions, such as invoking constructors or conversion operators.

5.If none of the above rules match, the compiler considers functions with ellipsis (...) parameters, which accept a variable number of arguments.

```
int half(int x, int divisor = 2) { // (1)
    return x / divisor;
}

double half(double x) { // (2)
    return x / 2;
}

half(4)// uses version (1), returns 2
half(3, 3)// uses version (1), returns 1
half(3.0) // uses version (2), returns 1.5
```

4.1.2 struct

struct: a group of named variables each with their own type. A way to bundle different types together

```
struct Student {
    string name; // these are called fields
    string state; // separate these by semicolons
    int age;
};
Student s;
s.name = "Sarah";
s.state = "CA";
s.age = 21; // use . to access fields
```

Abbreviated Syntax to Initialize a struct.

```
Student s;
s.name = "Sarah";
s.state = "CA";
s.age = 21;
//is the same as ...
Student s = {"Sarah", "CA", 21};
```

4.1.3 std::pair

std::pair: An STL built-in struct with two fields of any type.

std::pair is a template: You specify the types of the fields inside <> for each pair object you make.

The fields in std::pairs are named first and second.

```
std::pair<int, string> numSuffix = {1,"st"};
cout << numSuffix.first << numSuffix.second;
//prints 1st

std::pair<bool, Student> lookupStudent(string name) {
    Student blank;
    if (notFound(name)) return std::make_pair(false, blank);
    Student result = getStudentWithName(name);
    return std::make_pair(true, result);
}
std::pair<bool, Student> output = lookupStudent("Julie");
```

4.1.4 auto

auto: Keyword used in lieu of type when declaring a variable, tells the compiler to deduce the type.

```
// What types are these?
auto a = 3; // int
auto b = 4.3; // double
auto c = 'X'; // char
auto d = "Hello"; // char* (a C string)
auto e = std::make_pair(3, "Hello");
// std::pair<int, char*>
```

Keypoint:

- Everything with a name in your program has a type
- Static type system prevent errors before your code runs!
- Structs are a way to bundle a bunch of variables of many types

- std::pair is a type of struct that had been defined for you and is in the STL

- So you access it through the std:: namespace (std::pair)

- auto is a keyword that tells the compiler to deduce the type of a variable, it should be used when the type is obvious or very cumbersome to write out

4.2 Initialization & References

Initialization: How we provide initial values to variables.

```
Student s; // initialization after we declare
s.name = "Sarah";
s.state = "CA";
s.age = 21;
//is the same as ...
Student s = {"Sarah", "CA", 21};
// initialization while we declare
```

Multiple ways to initialize a pair.

```
std::pair<int, string> numSuffix1 = {1,"st"};
std::pair<int, string> numSuffix2;
numSuffix2.first = 2;
numSuffix2.second = "nd";
std::pair<int, string> numSuffix2 = std::make_pair(3, "rd");
```

4.2.1 Uniform initialization

Uniform initialization: curly bracket initialization. Available for all types, immediate initialization on declaration!

```
std::vector<int> vec{1,3,5};
std::pair<int, string> numSuffix1{1,"st"};
Student s{"Sarah", "CA", 21};
// less common/nice for primitive types, but
possible!
int x{5};
string f{"Sarah"};
```

4.2.2 Structured Binding

Structured binding lets you initialize directly from the contents of a struct.

```
auto p =
std::make_pair("s", 5);
auto [a, b] = p;
// a is string, b is int
// auto [a, b] = std::make_pair(...);
```

Using Structured Binding.

```
int main() {
    int a, b, c;
    std::cin >> a >> b >> c;
    auto [found, solutions] = quadratic(a, b, c);
    if (found) {
```

```
        auto [x1, x2] = solutions;
        std::cout << x1 << " " << x2 << endl;
    } else {
        std::cout << "No solutions found!" << endl;
        }
}
```

4.2.3 Reference

Reference: An alias (another name) for a named variable.

```
void changeX(int& x){ // changes to x will persist
    x = 0;
}
void keepX(int x){
    x = 0;
}
int a = 100;
int b = 100;
changeX(a); // a becomes a reference to x
keepX(b); // b becomes a copy of x
cout << a << endl; //0
cout << b << endl; //100
```

4.2.4 std::vector

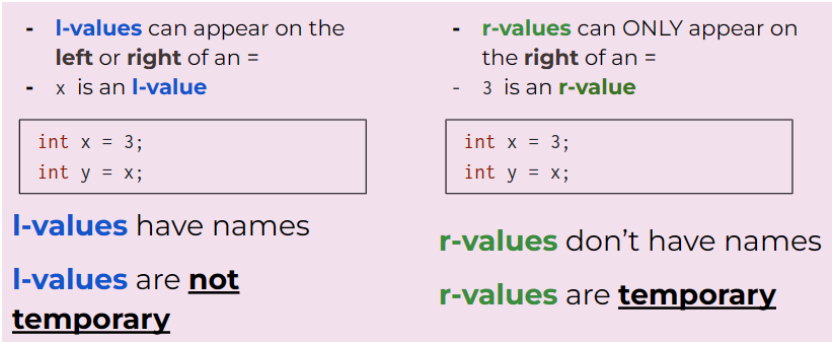
```
std::vector<int> v;
std::vector<int> v(n, k);
```

```
v.push_back(k);
v[i] = k;
int k = v[i];
v.empty();
v.size();
v.clear();
// stay tuned
// stay tuned
```

"=" automatically makes a copy! Must use & to avoid this.

```
void shift(vector<std::pair<int, int>>& nums) {
    for (size_t i = 0; i < nums.size(); ++i) {
        auto& [num1, num2] = nums[i];
        num1++;
        num2++;
    }
}
```

4.2.5 l-values vs r-values



The classic reference-rvalue error.

```
void shift(vector<std::pair<int, int>>& nums) {
    for (auto& [num1, num2]: nums) {
        num1++;
        num2++;
    }
}
shift({{1, 1}});
// {{1, 1}} is an rvalue, it can't be referenced
```

4.2.6 Const and Const References

const indicates a variable can't be modified!

```
std::vector<int> vec{1, 2, 3};
const std::vector<int> c_vec{7, 8}; // a const variable
std::vector<int>& ref = vec; // a regular reference
const std::vector<int>& c_ref = vec; // a const reference
vec.push_back(3); // OKAY
c_vec.push_back(3); // BAD - const
ref.push_back(3); // OKAY
c_ref.push_back(3); // BAD - const
```

Can't declare non-const reference to const variable!

```
const std::vector<int> c_vec{7, 8}; // a const variable
// BAD - can't declare non-const ref to const vector
std::vector<int>& bad_ref = c_vec;
```

const & subtleties

```
std::vector<int> vec{1, 2, 3};
```

```
const std::vector<int> c_vec{7, 8};
std::vector<int>& ref = vec;
const std::vector<int>& c_ref = vec;
auto copy = c_ref; // a non-const copy
const auto copy = c_ref; // a const copy
auto& a_ref = ref; // a non-const reference
const auto& c_aref = ref; // a const reference
```

When do we use references/const references?

- If we' re working with a variable that takes up little space in memory (e.g. int, double), we don' t need to use a reference and can just copy the variable

- If we need to alias the variable to modify it, we can use references

- If we don' t need to modify the variable, but it' s a big variable (e.g. std::vector), we can use const references

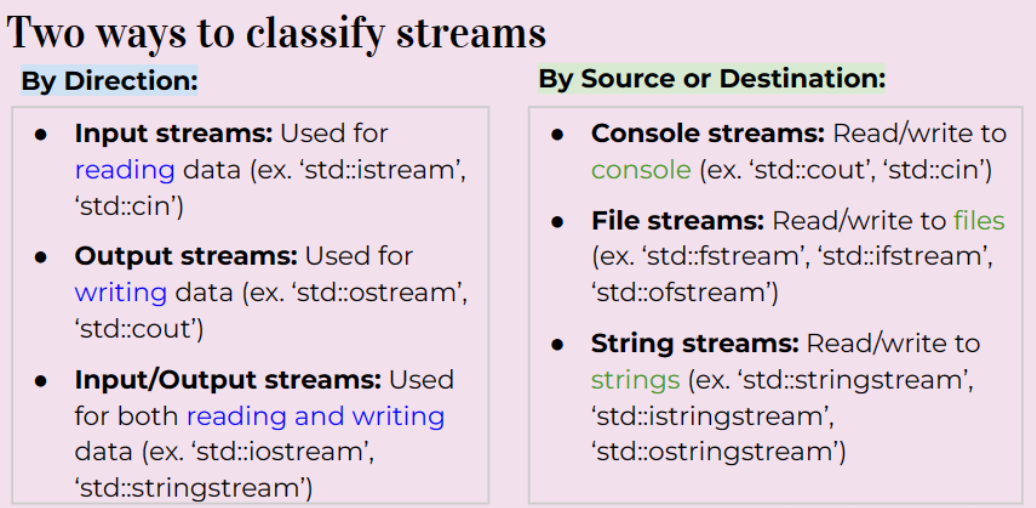
4.3 Streams

stream: an abstraction for input/output. Streams convert between data and the string representation of data.

```
// use a stream to print any primitive type!
std::cout << 5 << std::endl; // prints 5
// and most from the STL work!
std::cout << "Sarah" << std::endl;
// Mix types!
std::cout << "Sarah␣is␣" << 21 << std::endl;
// structs?
Student s = {"Sarah", "CA", 21};
```

```
std::cout << s << std::endl; //ERROR!
std::cout << s.name << s.age << std::endl;
```

std::cout is an output stream. It has type std::ostream.



4.3.1 Output Streams

- Have type std::ostream
- You can only send data to the stream
- Interact with the stream using the « operator
- Converts any type into string and sends it to the stream
- std::cout is the output stream that goes to the console

```
std::cout << 5 << std::endl;
// converts int value 5 to string "5"
// sends "5" to the console output stream
```

4.3.2 Output File Streams

- Have type std::ofstream

- You can only send data to file using the « operator

- Converts data of any type into a string and sends it to the file stream

- Must initialize your own ofstream object linked to your file

```
std::ofstream out("out.txt");
// out is now an ofstream that outputs to
out.txt
out << 5 << std::endl; // out.txt contains 5
```

4.3.3 Input Streams

- Have type std::istream

- You can only receive strings using the » operator

- Receives a string from the stream and converts it to data

- std::cin is the input stream that gets input from the console

```
int x;
string str;
std::cin >> x >> str;
//reads exactly one int then one string from
console
```

Nitty Gritty Details: std::cin

- First call to std::cin » creates a command line prompt that allows the user to type until they hit enter

- Each » ONLY reads until the next whitespace

- Whitespace = tab, space, newline

- Everything after the first whitespace gets saved and used the next time std::cin » is called

- The place its saved is called a buffer!

- If there is nothing waiting in the buffer, std::cin » creates a new command line prompt

- Whitespace is eaten; it won't show up in output

4.3.4 Stringstreams

```
std::string input = "123";
std::stringstream stream(input);
int number;
stream >> number;
std::cout << number << std::endl; // Outputs "123"
```

4.4 Containers

Container: An object that allows us to collect other objects together and interact with them in some way.

4.5 Ending

第五章　Leetcode

第六章　Computer Network

第七章　Git & Linux