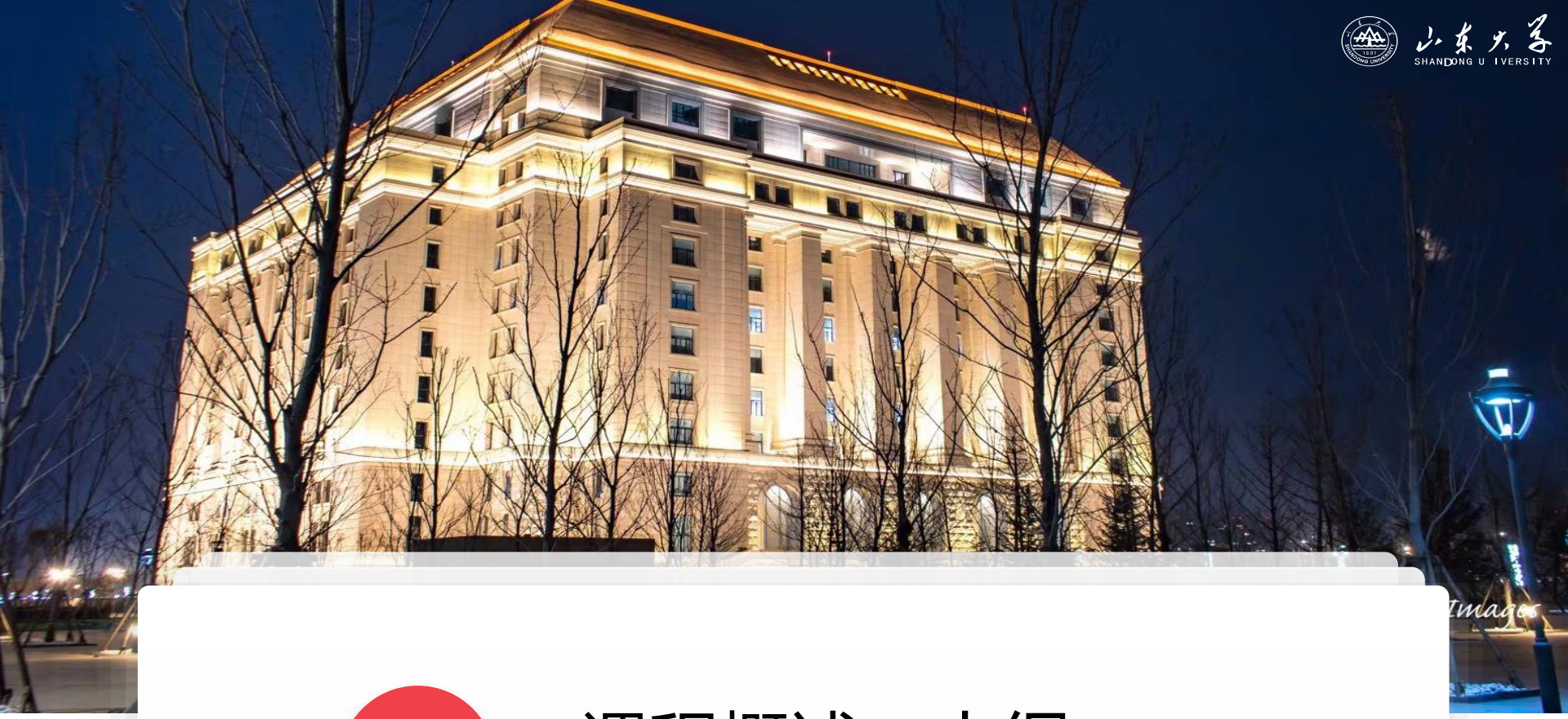
程序设计思维与实践

Thinking and Practice in Programming

算法复杂度与程序的调试 | 内容负责：郑得贤



1 课程概述、大纲

The outline of course

课程概述

- 课程概述
  - 课程名：程序设计思维与实践 (Thinking and Practice in Programming)
  - 前置课程：计算导论与程序设计、数据结构与算法
  - 后续课程：算法设计与分析、创新创业教育实践（CCF CSP 认证）
  - 意义：
    - 是前置课程应用于解决实际问题的最佳实践，真正落实到代码实现层**针对具体问题**去思考**数据结构的应用**和**算法的设计**；
    - 是后续课程学习强有力的助攻，比如图论算法的实践与后续算法设计课中涉及的原理证明，课程的实验课与 CCF CSP 认证课 (1 学分)；
    - 培养计算机思维；
    - …

课程概述

- 课程团队及上课分配
  - 四位讲师郑得贤、师浩晏、叶苏伟、杨柳混合讲课
  - 周二第 5 大节：1-16 周 实验
  - 周三第 5 大节：1-16 周 讲课
- 课程实时反馈：对于知识点、作业等难度的反馈，根据反馈考虑调整授课大纲。

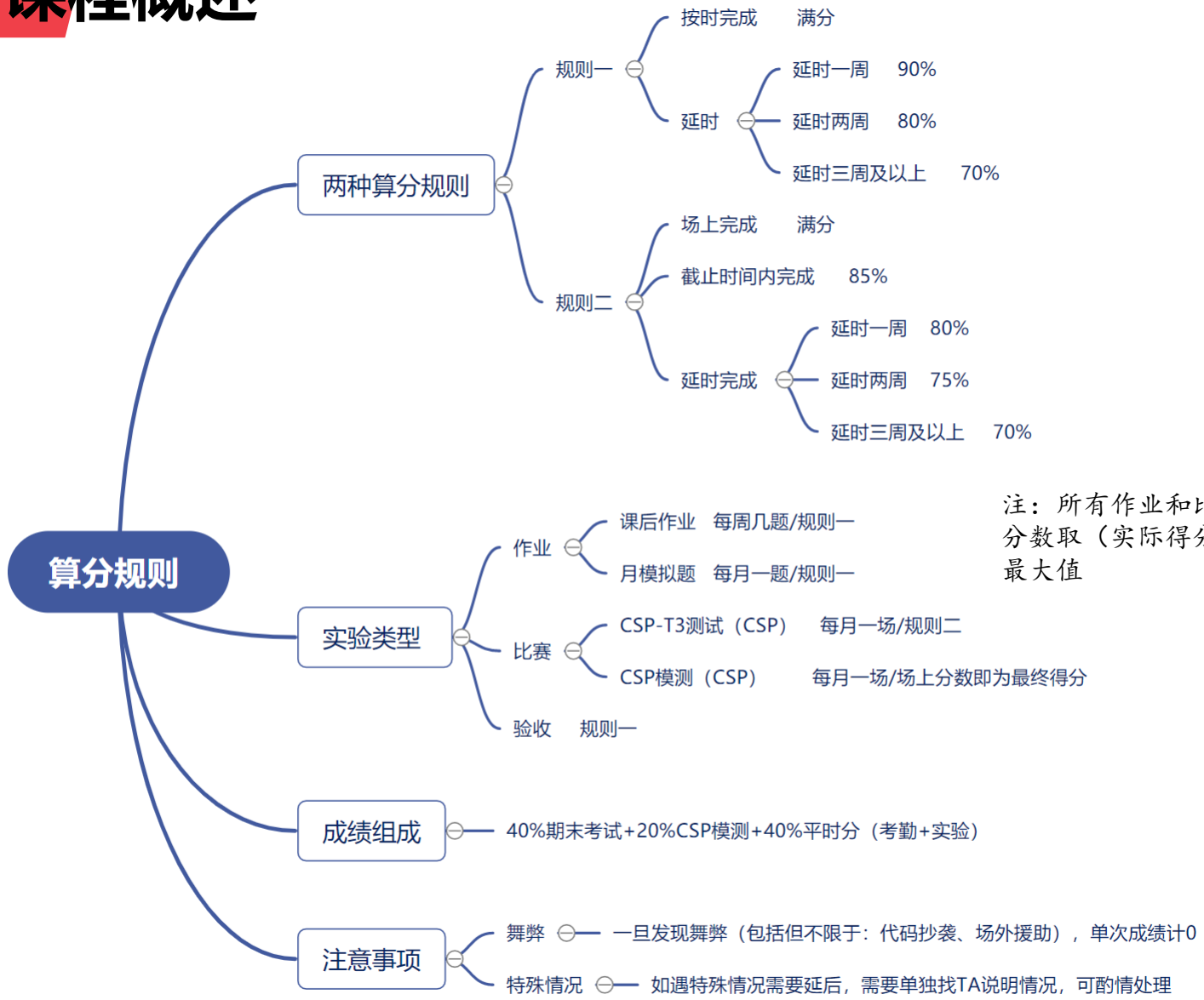
课程概述

- 上课方式
  - 每周 2 小时理论课，2 小时上机实验
  - 每周理论课后布置课堂内容相关的作业题
  - **实验课**以 4 周为一轮，每 4 周一道 CSP-T3 练习，一次限时 CSP-T3 题型比赛，一次 CSP 赛制模拟考试。具体形式如下：
    - 第 i 周：布置这一轮的 CSP-T3 练习（4周内完成即可），CSP 赛制的CSP-T3 题型比赛。
    - 第 i+1 周：完成作业与CSP-T3 练习
    - 第 i+2 周：一次 CSP 赛制模拟考试。（面向 CSP-T1、T2、T4）
    - 每 i+3 周：完成作业与CSP-T3 练习
- 编程语言：C / C++ / Java / Python，即 CCF CSP 认证要求语言，课程讲述应用C++
- 对应 IDE：VS-Code/ Eclipse / IDLE
- PS：实验课所有代码查重，一旦发现雷同，则取消该次实验分数。

课程概述

- 作业/实验 验收形式：
  - 每周会提出与作业或实验相关的问题，在实验报告中完成问题，问题考察包括但不限于算法复杂度、程序实现框架的细节等。实验报告需在下次讲课之前提交

课程概述



注：所有作业和比赛均采用IOI赛制，分数取（实际得分*得分时折扣）的最大值

课程大纲

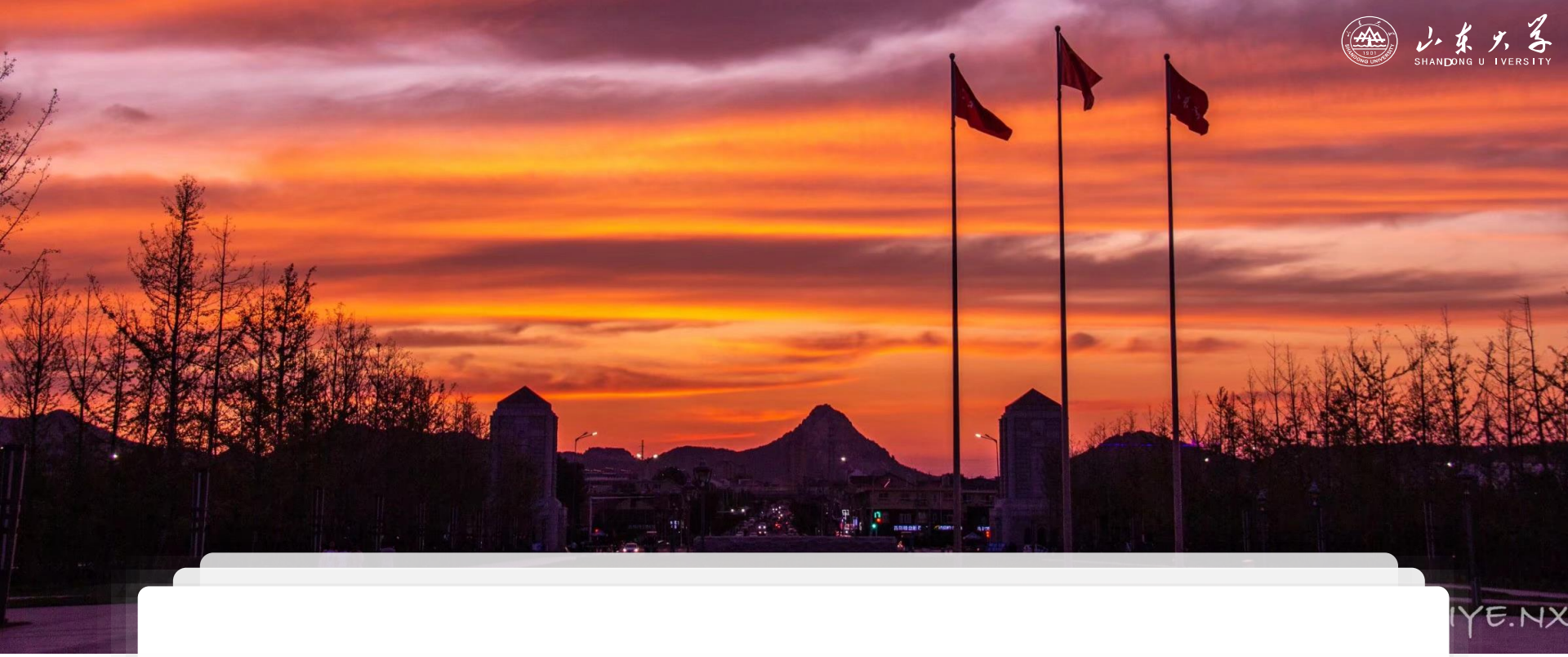
周次	主题	课程内容
1	概述、 算法复杂度、 程序的调试	1、 课程概述/大纲（10min） 2、 程序设计竞赛相关名词，如 VJ、OJ、AC、RE 等，以及评测系统的使用（10min） 3、 算法复杂度分析（时空）（30min） 4、 数据范围（值域）、时限以及复杂度三者关系（10min） 5、 Input/Ouput scanf(cin)/printf(cout)(高级用法)(10min) 6、 使用 IDE 进行调试（30min） 作业：CSP T1 * 2，输入输出
2	C++与STL	c++语法 浮点数精度问题 字符串操作（字符串与数字） 1、vector/list，迭代器（10min） 2、sort，结构体（重载），多关键字排序（15min） 3、unique（5min） 4、stack（5min） 5、queue、priority_queue（10min） 6、map/unordered_map、set/unordered_set，（10min） 资料：附加文档（常用技巧、IDE、白盒测试总结）， 作业：CSP T1 * 2，STL 练习题
3	搜索	BFS（10min） BFS 例题 洪水填充（40min） DFS 及其剪枝（20min） DFS例题：八皇后 CSP-再卖菜（80分做法） （30min） 作业：BFS，DFS

课程大纲

4	贪心 二分	1、 贪心算法概念、贪心指标（15min） 2、 端点排序贪心（20min） 3、 区间选点问题证明 +典型例题 4、 二分算法框架（比较不同写法产生的边界问题）（20min）浮点数二分 5、 离散化 6、 二分答案（25min） 作业：贪心 + 二分
5	数学基础与方法应用	1、 *单调栈，及其例题 （25min） 3、 *单调队列，及其例题 （25min） 1、数论基础（基本性质，快速幂，二进制） 2、计算几何基础 3、 尺取 例题×2（30min） 4、 一维前缀和、二维前缀和，及其例题（20min） 5、 差分（一维，二维） 作业：尺取 前缀和 差分
6	图和树的性质与应用（上）	1、 树的存储（邻接表、前向星）（15min） 2、 树的遍历（DFS、BFS）（35min） 3、 并查集与 Kruskal（prim）（50min） 作业：树的直径题、最小生成树题
7	图和树的性质与应用（中）	1、 多元最短路floyd（20min） 2、 最短路 Dijkstra（20min） 3、 Bellman-ford及队列优化（40min） 作业：最短路练习2题
8	图和树的性质与应用（下）	1、 差分约束系统（30min） 2、 图的拓扑排序（20min） 3、 图的强连通分量分解（30min） 作业：差分约束一道，拓扑排序一道

课程大纲

10	树形数据结构	1、树状数组 2、线段树（单点，*区间） 作业：树状数组一道、线段树一道
11	动态规划（一）	1、 递推（20min） 2、 数字三角形（10min） 3、 记忆化（30min） 4、 LIS（25min） 5、 LCS（15min） 作业：递推 LIS
12	动态规划（二）	1. 背包 DP 0-1型、滚动数组 （40min） 2. 多重背包及二进制拆分（30min） 3. 完全背包（20min） 4. 输出方案 作业：0-1背包 多重背包
13	动态规划（三）	1、 区间dp（40min） 2、 状压dp（50min） 作业：区间dp例题、状压dp例题
14	动态规划（四）	1、 树形DP（60min） 2、 DP优化（前缀和，数据结构）（45min） 作业：树形dp例题、dp优化
15	矩阵的应用与*字符串	1、 矩阵结构体（15min） 2、 矩阵快速幂（40min） 3、 矩阵快速幂优化DP（30min） 1、 *KMP（50min）+ 例题 2、 *Trie（50min）+例题 作业：矩阵快速幂优化DP
16	复习课	总复习串讲+答疑
17	答疑+验收	答疑+验收
18	期末考试	



2 程序设计竞赛的相关名词

The related terms of programming competition

相关名词

- OJ，Online Judge，在线评测系统。如数据结构实验课使用的评测系统。
  它可以将用户的程序代码编译运行，将输出结果与正确答案进行比对，返回比对结论。
- VJ，Virtual Judge，虚拟评测系统，并非真实具有评测能力的系统，只是利用爬虫技术将用户的程序代码提交到各大 OJ 上，并爬取评测结果展现。
- 评测结果
  - AC，Accepted，该程序产生了与答案输出相同或检查程序（通常称为 Special Judge）认为正确的输出。
  - WA，Wrong Answer，该程序产生的输出错误
  - PE，Presentation Error，该程序产生的结果是正确的但是输出格式有误
  - TLE，Time Limit Exceeded，该程序超过限定的运行时间，程序超时
  - MLE，Memory Limit Exceeded，该程序使用的内存超限
  - RE，Runtime Error，该程序在运行期间产生了无法处理的异常
  - OLE，Output Limit Exceeded，该程序产生了过多的输出
  - CE，Compile Error，该程序代码未能成功通过编译

(OJ  : ”本地编译通过了再交，行吗？”)

相关名词

- OI赛制、IOI赛制，CSP赛制
  - 相关人员的口头术语
  - OI赛制（原CSP赛制）意指：不实时评测，无排行榜的，选手程序将在赛后进行统一评测，只以最终提交的程序的评测结果作为最终的成绩
  - IOI赛制意指：可以实时评测、看排行榜（其他人的提交与得分情况），可以进行多次提交，能够看到自己每次的评测结果，取最优的成绩作为最终成绩
  - 现CSP赛制：与IOI赛制相似，但是在比赛过程中，无法看到排行榜，只能看到自己的成绩。

3 复 杂 度 分 析

The analysis of program complexity

算法评估

- 通常设计一个“好”的算法应考虑达到以下目标：

（1）正确性

（2）可读性

（3）健壮性（鲁棒性）

（4）高效率与低存储量需求

时间限制：	1.0s
内存限制：	256.0MB

- 算法效率的度量是通过**时间复杂度**和**空间复杂度**来描述的。

在绝大多数竞赛中都会给出时间限制和空间限制，这也是我们关注算法复杂度的重要原因之一。

时间复杂度

在进行算法分析，语句总得执行次数 T(n) 是关于问题规模 n 的函数，进而分析 T(n) 随 n 的变化情况并确定 T(n) 数量级。算法的时间复杂度，也就是算法的时间量度，记作：**T(n) = O(f(n))**，它表示随问题规模 n 的增大算法执行时间的增长率和 f(n) 的增长率相同，称作算法的渐近时间复杂度，简称为时间复杂度，其中**f(n) 是问题规模 n 的某个函数。**

时间复杂度求解的具体步骤

（1）找出算法中的基本语句

算法中执行次数最多的那条语句就是基本语句，通常是最内层循环的循环体。

（2）计算基本语句的执行次数的数量级

只需计算基本语句执行次数的数量级，这就意味着只要保证基本语句执行次数的函数中的最高次幂正确即可。**可以忽略所有低次幂和最高次幂的系数。**这样能够简化算法分析，并且使注意力集中在最重要的一点上：增长率。

（3）用大 O 记号表示算法的时间性能

将基本语句执行次数的数量级放入大 O 记号中。

大 O 中的 O 的意思就是"order of"（大约是），它是种概念，就比如 大型车、小型车和中型车，忽略具体大小尺寸，来描述汽车。

时间复杂度计算方法

（1）用常数 1 取代运行时间中的所有加法常数。

（2）在修改后的运行次数函数中，**只保留最高阶项**。

（3）如果最高阶存在且不是 1，则去除与这个项相乘的常数。最后，得到的最后结果就是时间复杂度。

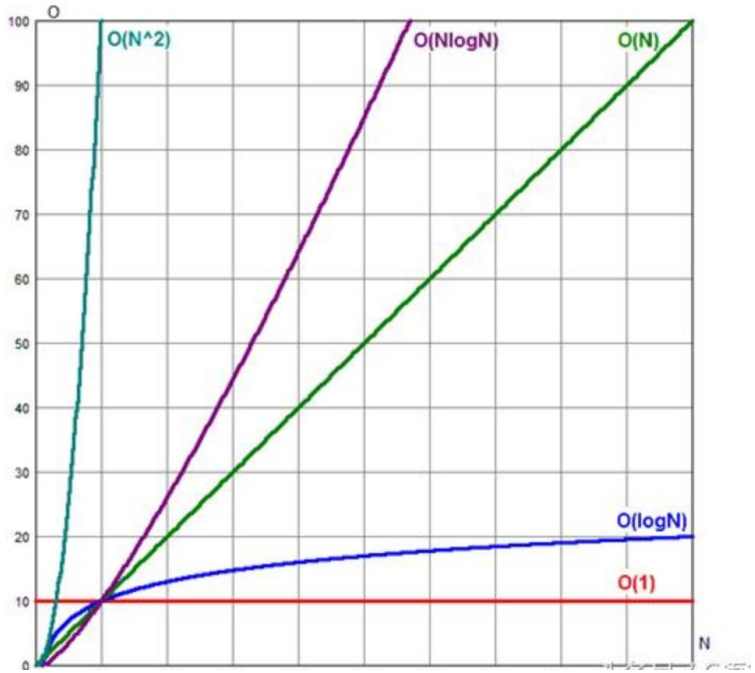
简单来说，就是保留求出次数的最高次幂，并且把系数去掉。

T(n) = 2n^2+n+1 = O(n^2)

T(n)=n*(n+1)/2=O(n^2)

常见时间复杂度

（1）常数级复杂度：O(1)
（2）对数级复杂度：O(logN)
（3）线性级复杂度：O(N)
（4）线性对数级复杂度：O(NlogN)
（5）平方级复杂度：O(N^2)



时间复杂度分析举例

```
for (int i=1;i<=n;i++)
    for (int j=1;j<=m;j++) {
        /*...*/
        for (int k=1;k<=t;k++) {
            /*...*/
        }
    }
```

O(n*m*t)

循环结构按照乘法进行计算

O(logn)

```
int i=1;
while(i<n) {
    i=i*2;
}
```

```
void solve(int l,int r)
{
    if (l==r) return;
    int mid=(l+r)/2;
    solve(l,mid); solve(mid+1,r);
    int i=l; int j=mid+1; int k=l-1;
    while(i<=mid&&j<=r) {
        if (sum[i]<sum[j]) c[++k]=sum[i++];
        else c[++k]=sum[j++];
    }
    while(i<=mid) c[++k]=sum[i++];
    while(j<=r) c[++k]=sum[j++];
    for (int i=l;i<=r;i++) sum[i]=c[i];
}
```

O(nlogn)

左侧代码为归并排序

时限与时间复杂度

- 计算机每秒的计算量有限，只有效率更优的算法，才能解决更大的问题规模。
- 不同的复杂度对应的计算量不同
- 假设机器速度是每秒 10^8 次基本运算，在计算机 1s 之内能解决的最大问题规模 n 如表所示：

运算量	$n!$	2^n	n^3	n^2	$n\log_2 n$	n
最大规模	11	26	464	10000	$4.5*10^6$	100000000
速度扩大两倍后	11	27	584	14142	$8.6*10^6$	200000000

- 如果我们知道一个题目的时限是 1s，数据范围是 1e6
  - 显然地，我们无法接受 $O(n^2)$ 及以上的算法
  - O(n)、O(nlogn) 是可以接受的

空间复杂度

- 一个程序的空间复杂度是指运行完一个程序所需内存的大小。利用程序的空间复杂度，可以对程序的运行所需要的内存多少有个预先估计。一个程序执行时除了需要存储空间和存储本身所使用的指令、常数、变量和输入数据外，还需要一些对数据进行操作的工作单元和存储一些为现实计算所需信息的辅助空间。

- 如何计算?

例如：int a[10000000] => 4*10000000/1024/1024约等于39MB

注意：
1. 不要把 **超过1M** 的数组开成临时变量。
2. 使用数组尽量略微开大一些，增加容错率。（例如：int a[10010]）

部分分的获取

- 回到以分段形式给出的测试点规模当我们设计出较优解，无法一次性解决全部测试点时，思考：

如何“骗分”？

- 答案是：解决一部分测试点
- 框架如下：

```
int main() {
    scanf("%d%d", &n, &m);
    if ( n<=10 && m<=10 )         test1();
    else if ( n<=50 && m<=50 )    test2();
    else if ( n<=300 && m<=300 ) test3();
    // ... 以此类推
}
void test1() {
    // 解决小规模问题的暴力算法
}
void test2() {
    // 尝试解决较小规模问题的暴力算法
}
void test3() {
    // ... 瞎写
}
```

输入格式

输入只有一行，包括恰好一个正整数n（4 ≤ n ≤ 1000）。

输出格式

输出只有一行，包括恰好n 位的整数中有趣的数的个数除以1000000007的余数。

子任务

共有 10 个测试点，各测试点特点如下：

测试点1：n=10，m=10。

测试点2：n=50，m=50。

测试点3：n=300，m=300。

测试点4：n=2000，m=2000。

测试点5：n=2000，m=100000。

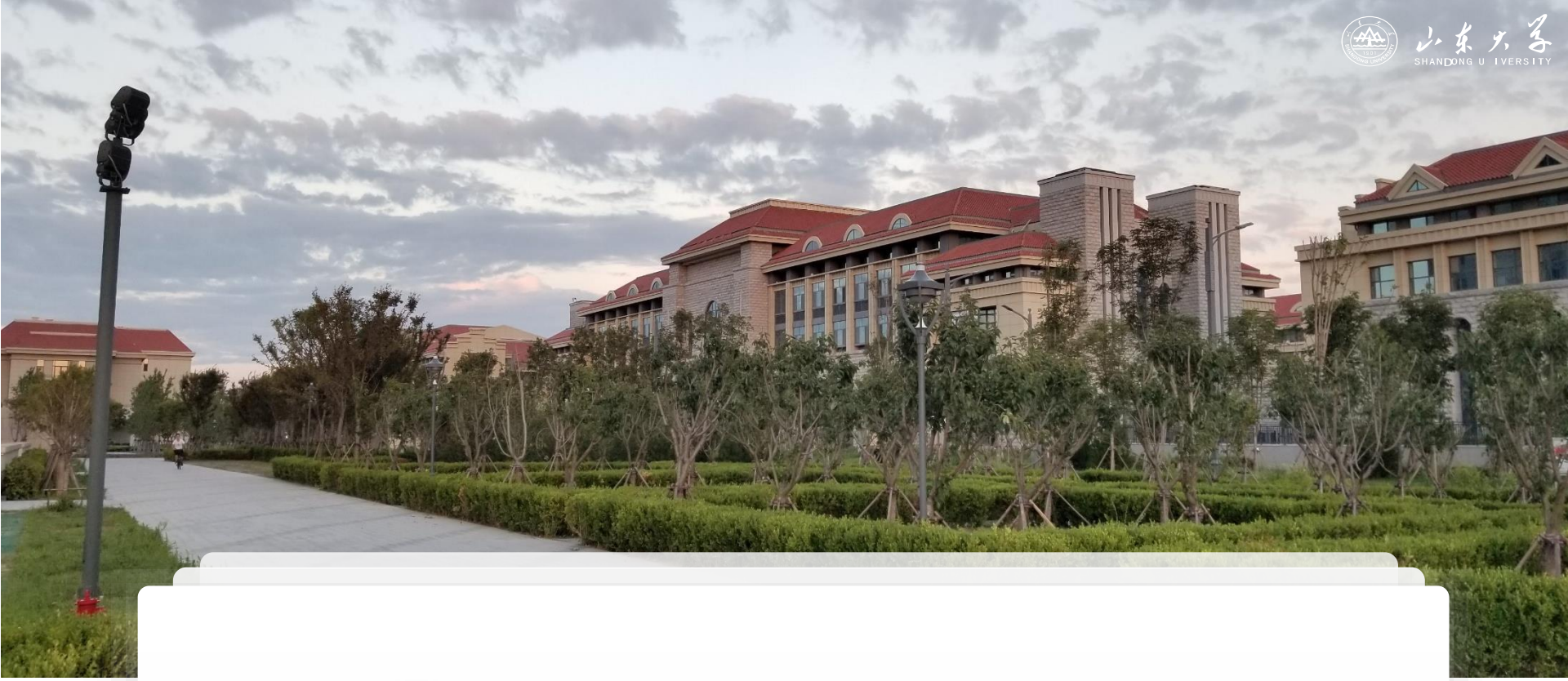
测试点6：n=100000，m=100000。保证 $f_i=i$。

测试点7：n=100000，m=100000。保证 $f_i=\lfloor(i+1)/2\rfloor$，

测试点8：n=100000，m=100000。保证对于同一组数据

测试点9：n=100000，m=100000。

测试点10：n=100000，m=100000。



4 程 序 的 调 试

Program debugging

简单的文件操作

- 在调试时加入文件，可以极大的提高调试的效率
- 在评测平台提交代码时**一定要去掉文件操作**

```cpp
#include<cstdio>
using namespace std;
int main()
{
    freopen("a.in","r",stdin);
    freopen("a.out","w",stdout);

    /*...*/

    return 0;
}
```

样例输入

```
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 1 0 0 0
1 1 1 0 0 0 1 1 1 1
0 0 0 0 1 0 0 0 0 0
0 0 0 0
0 1 1 1
0 0 0 1
0 0 0 0
3
```

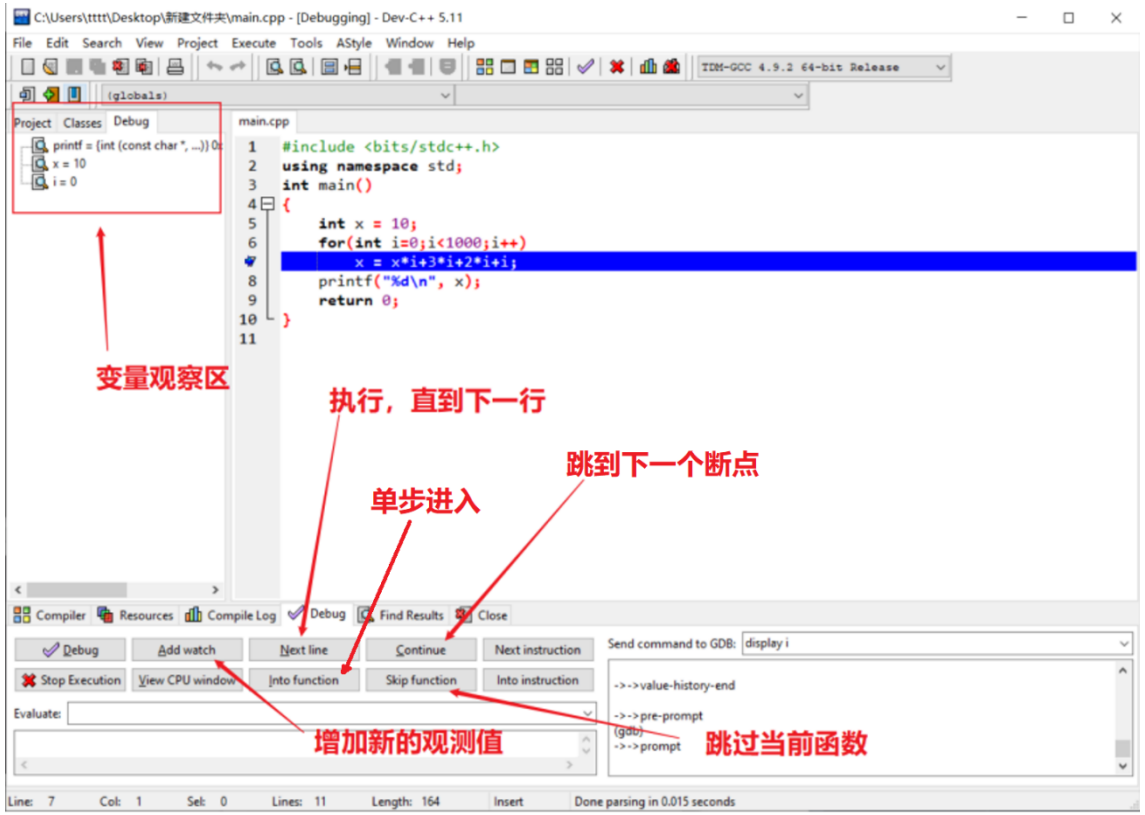
Windows下——Dev C++为例

（1）选择debug模式（配置：工具-编译选项-代码生成/优化-连接器-产生调试信息-Yes）

（2）在合适的位置打上断点

（3）按F5启动调试
（或点击紫色的对号）

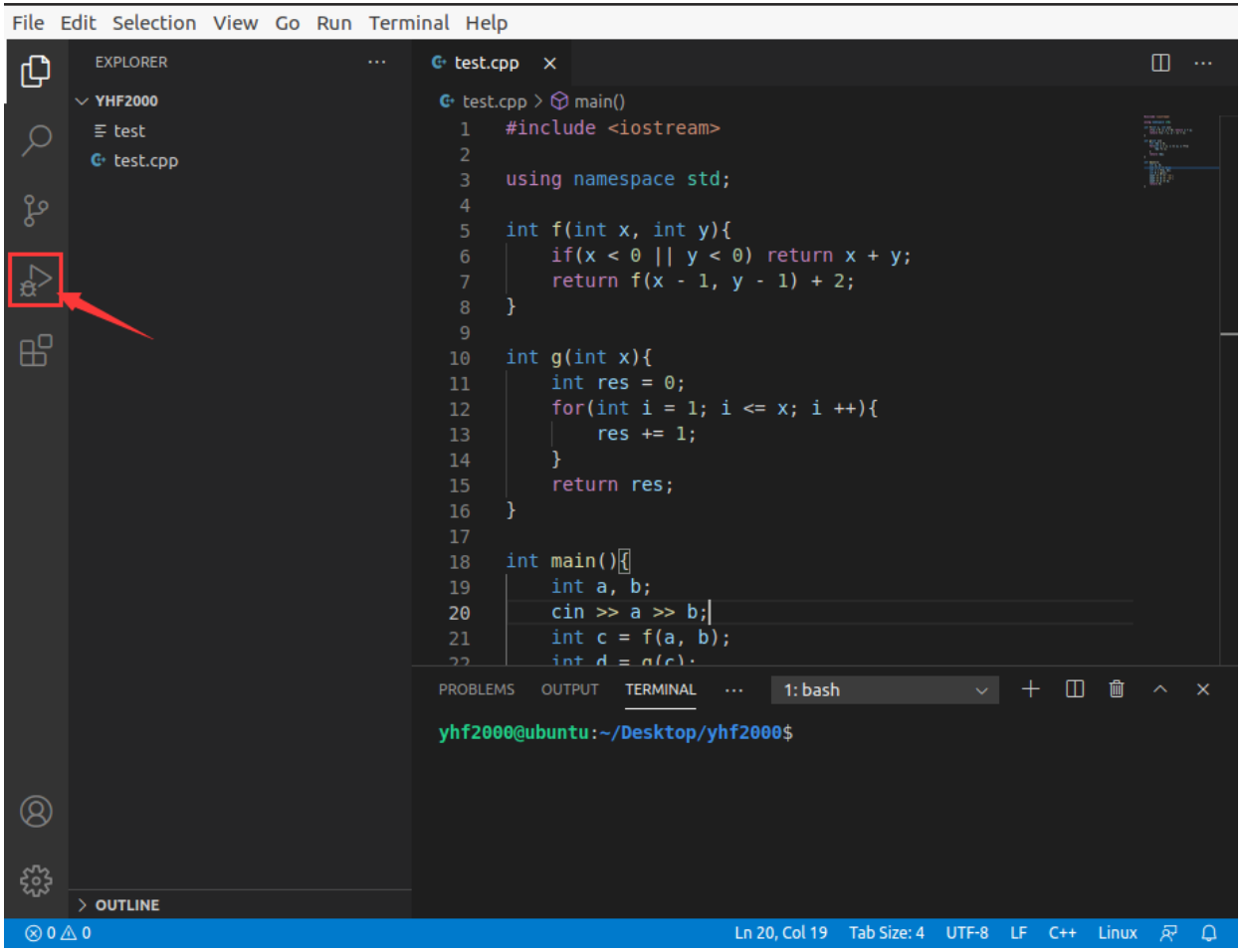
常用快捷键：
F7 执行下一条语句但不进入
F8 执行并进入下一条语句



配置与具体方法请参考附件资料。

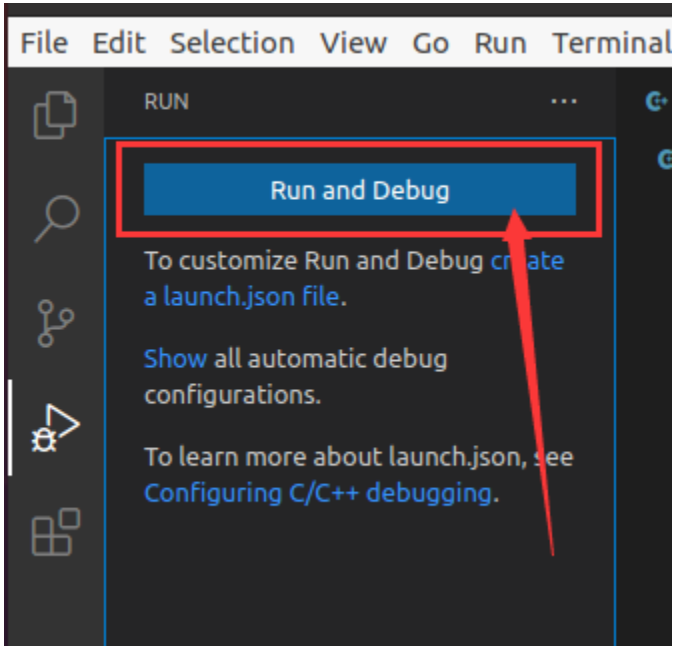
Linux下——VS Code 为例

- csp测试是在Linux（Ubuntu）下进行的，提供 Vscode
- 第一步：点击左侧调试按钮

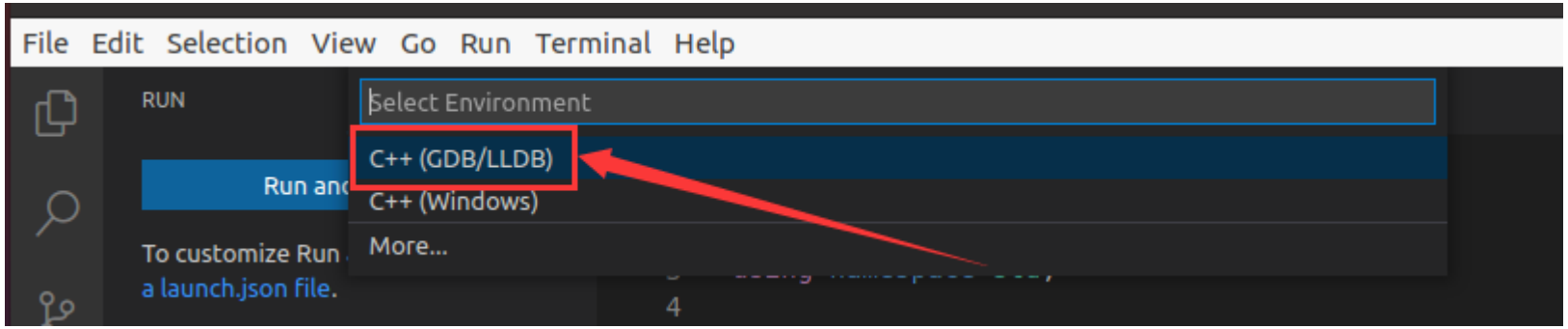


Linux下——VS Code 为例

- 第二步：点击 Run and Debug

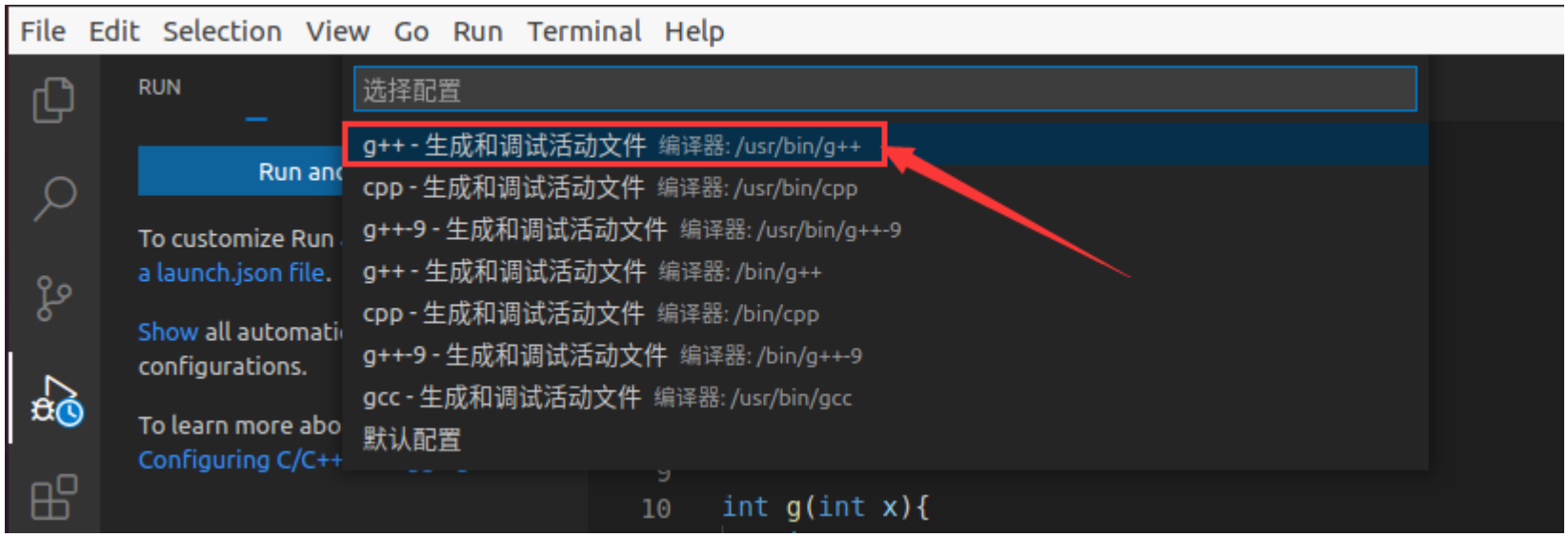


- 第三步：点击 C++ (GDB/LLDB)



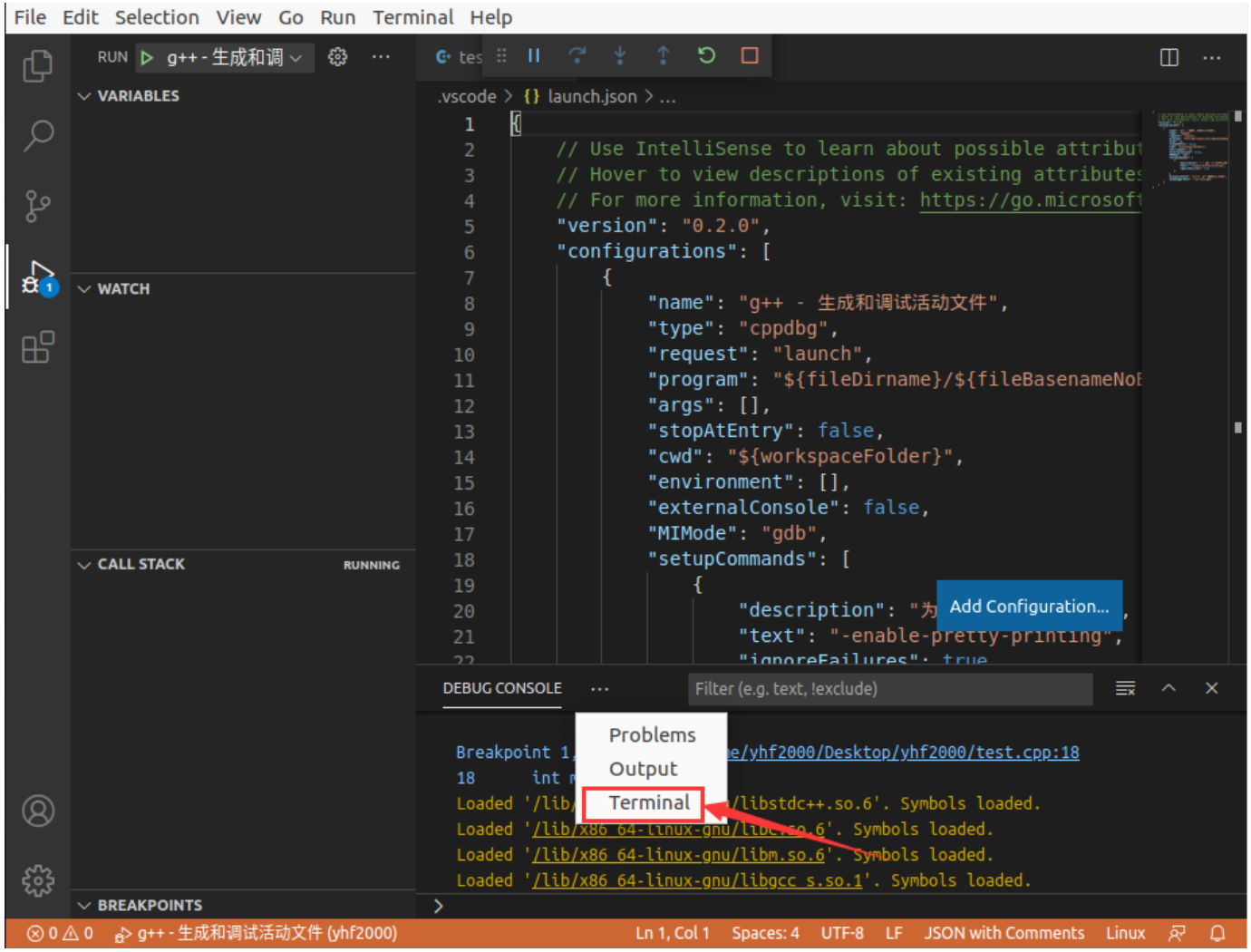
Linux下——VS Code 为例

- 第四步：选择一个g++环境



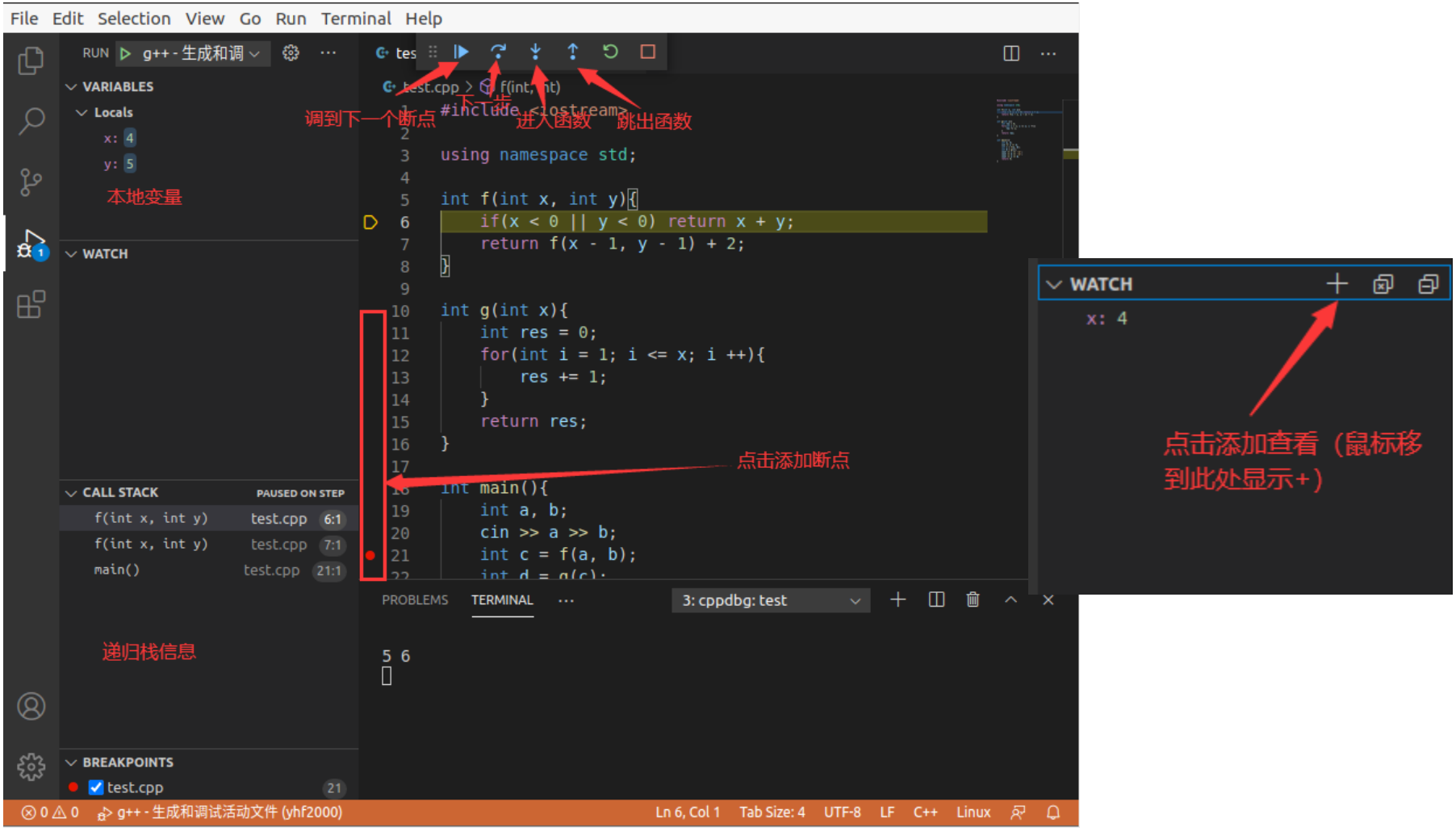
Linux下——VS Code 为例

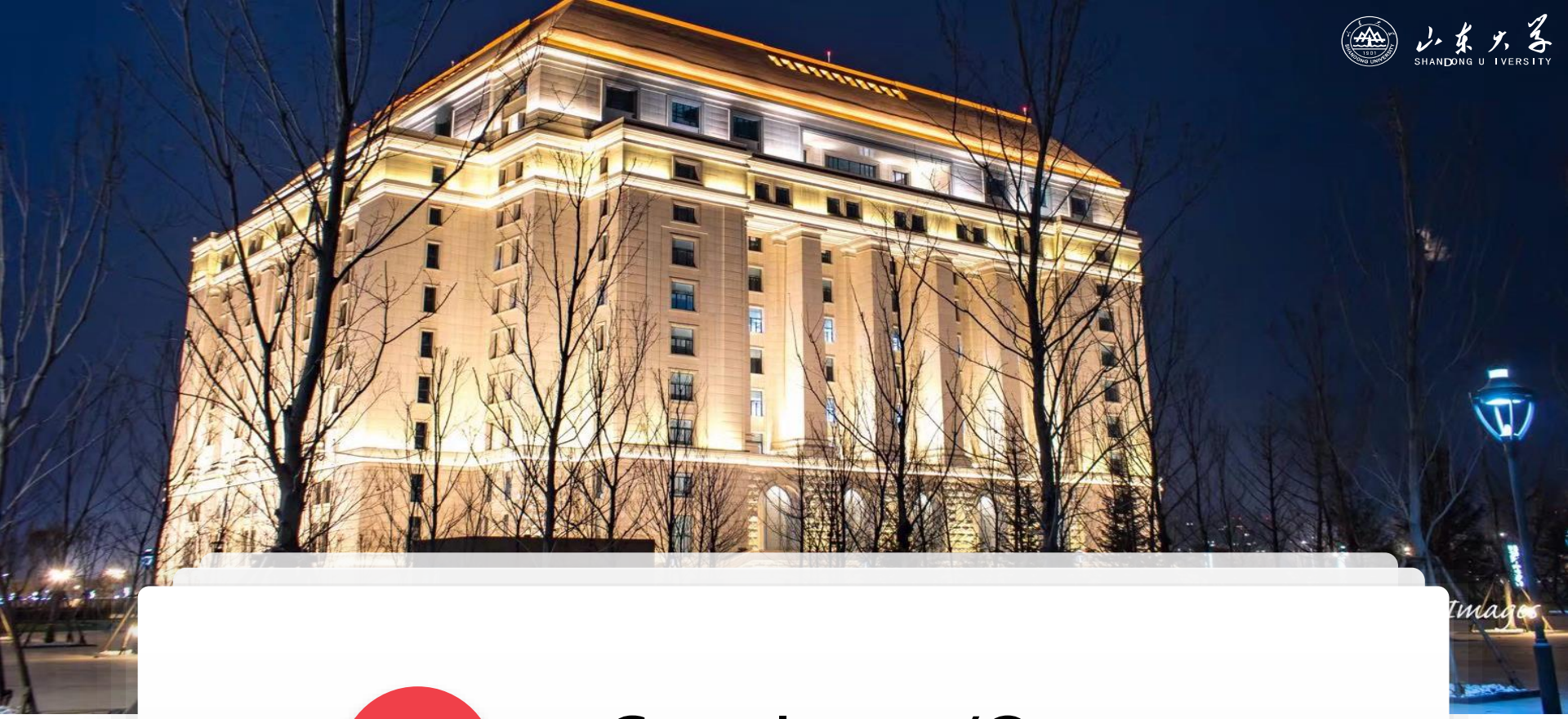
- 第五步：返回终端（输入输出都在终端内）



Linux下——VS Code 为例

- 第六步：开始调试（演示）





5

C++ Input/Output

C++ Input&Output

Input/Output

- 输入与输出
  - 在 ACM 比赛中，一般使用标准输入输出，读入题目输入，将程序输出与答案进行对比，来判定程序的正确性。
  - cstdio (C Standard Input and Output Library) 是 C 语言原生库，比起 C++ 新定义的流对象操作库 iostream 在效率方面自然更优，故建议是**只用 cstdio，不用 iostream**，因为前者熟练使用起来几乎可以替代后者，且在时间效率上无懈可击。
  - cstdio 这个库较为重要的知识在于五个方面：
    - 格式化输入输出到标准 IO （掌握）
    - 格式化输入输出到字符串（了解）
    - 单字符输入输出（了解）
    - 快速读入、缓冲区冲刷 （了解）

Input/Output

- 格式化输入输出到标准 IO
  - 主要关注两个函数
    int scanf (const char * format, ...);
    int printf (const char * format, ...);

```
/* 变量定义 */
int i;  double d;  char c;  int is[15];  unsigned int ui;  long long ll;

/* 读入 */
scanf("%d %lf %c %d %u %lld", &i, &d, &c, &is[1], &ui ,&ll); // 其中&aa[1]可换为`aa + 1`
scanf("%1d", &i); // 只读1位到a

/* 输出 */
printf("%d %.2f %c %s %d %u %lld", i, d, c, s+1, is[1], ui, ll);  // 注意duoble输出
是%f,%.2f表示四舍五入到百分位

/* 有关进制的输入输出 */
int a, b; scanf("%o %x", &a, &b); // 表示将输入看作8进制、16进制读入，比如输入"0777
0x3FFFFFF"(可以没有前导0、0x)。输出同理。

/* 变量地址的输出 */
int a; printf("%p %#p", a); // 输出类似00000053 0x53
```

Input/Output

- 格式化输入输出到标准 IO
  - 主要关注两个函数
    int scanf (const char * format, ...);
    int printf (const char * format, ...);

```
/* 关于char[]的输入输出 */
char s[105];
scanf("%s", s); // 读入一个字符串，且首指针放到s+0
scanf("%s%s%s", s+1, s+21, s+41); // 读入三个字符串，且首指针分贝放到s+1、s+21、s+41，注意本样例若
前两个字符串长度超过19，就会在内存有重叠而造成使用出错
```

  - 其他例如“关于 string 的输入输出”等知识点可自行拓展，见二维码

Input/Output

- 格式化输入输出到字符串
  - 主要关注 sscanf、sprintf。和 scanf、printf 的使用一致，只是输入、输出的对象从标准输入输出变成了 char[]。

```
char s[] = "123 52.2 adas";
int a; double b; char ss[105];
sscanf(s, "%d%lf%s", &a, &b, ss); // a=123, b=52.2, ss="adas"
a++; ss[0]='b';
sprintf(s, "%d%lf%s", a, b, ss); // s="12452.2bdas"  后面还追有 '\0' 's' '\0'
```

- 单字符输入输出
  - 大多数情况下 scanf、printf 配合 %c 可以适应单字符输入输出的要求。但是对于优化效率，getchar()、putchar()十分有用。

```
char c = getchar();  // 从标准输入读入一个字符
putchar(c);          // 输出一个字符到标准输出
```

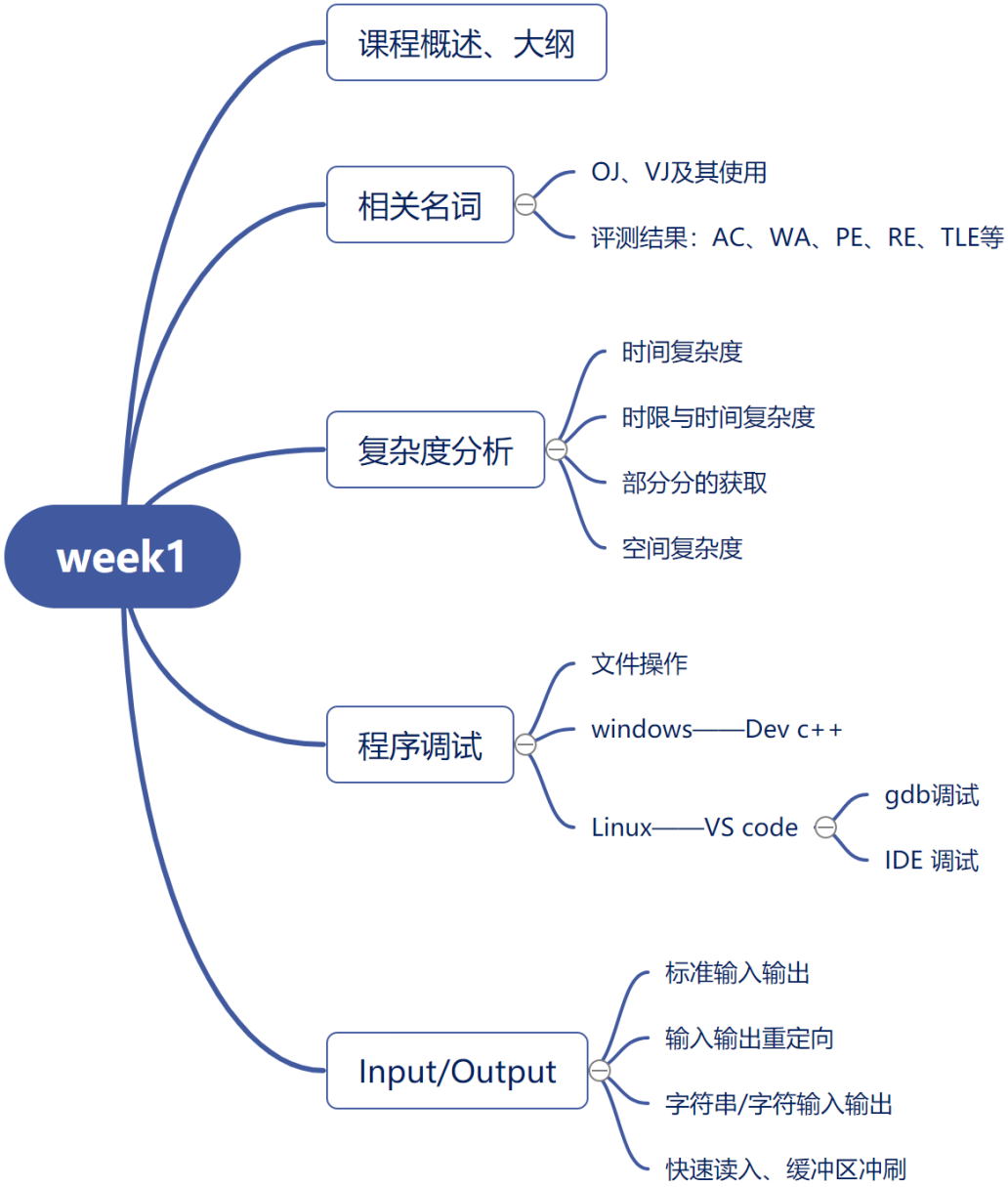
Input/Output

- 快速读入、缓冲区冲刷 (了解)
  - 读入和输出时耗同样计入程序运行时间对于一些大量数据读入的题目，格式化的输入，在底层对于识别格式串时已浪费不少时间。
  - 快速读入是一种声明了众多针对不同数据类型的读入函数，且预先将所有 bytes 当作 char 类型读到数组作为缓冲区，随后再从缓冲区中模拟读入的一种优化读入效率操作。
  - 粗略的原理代码如下，如有兴趣，可自行阅读约 50 行代码的快读模板

```
char s[]="-123456"; // 模拟的缓冲区
void read(int &x)
{
    x=0;
    bool sign=0;  int i=0;
    if(s[i] == '-') sign=1, i++;
    for(; s[i]!='\0'; i++) x = x*10 + (s[i]-'0');
    if(sign) x=-x;
}
```

- 缓冲区冲刷 (了解计算机底层并非 printf 后就立即输出，而是存在一个缓冲区)
  - fflush(stdout);
    可以将输出缓冲区中的内容当即输出，在一些交互题上是必须要使用的。

总结


week1
课程概述、大纲
相关名词
OJ、VJ及其使用
评测结果：AC、WA、PE、RE、TLE等
复杂度分析
时间复杂度
时限与时间复杂度
部分分的获取
空间复杂度
程序调试
文件操作
windows——Dev c++
Linux——VS code
gdb调试
IDE 调试
Input/Output
标准输入输出
输入输出重定向
字符串/字符输入输出
快速读入、缓冲区冲刷

感谢收听

Thank You For Your Listening