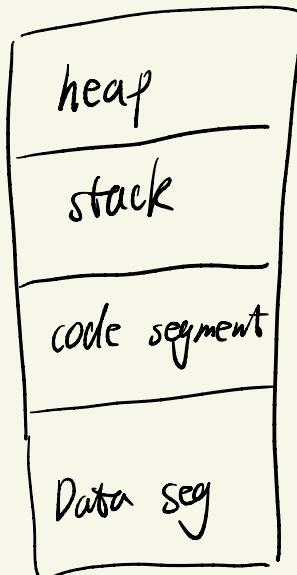


# C++ 内存

C++ 内存：



Data Segment:

1. 存储内容： global . static | data  
global static | local static  
member static

2. 生命周期： main 函数运行 为 分界，在 main 函数运行前 存储完 并 对 Data  
初始化。 持续到 main 函数运行结束

Code Segment:

1. 存储内容： 所有函数

2. 生命周期： main 函数运行前， 到 main 函数运行结束。

### 3. Code Segment 中函数如何运行：

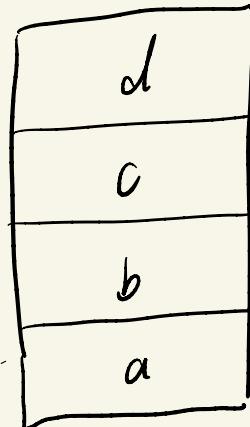
在汇编语言中通过 pointer, assignment, move 实现

stack :

1. 运行方式：执行栈的 “first in last out”

e.g. in:  $a \rightarrow b \rightarrow c \rightarrow d$

out:  $d \rightarrow c \rightarrow b \rightarrow a$



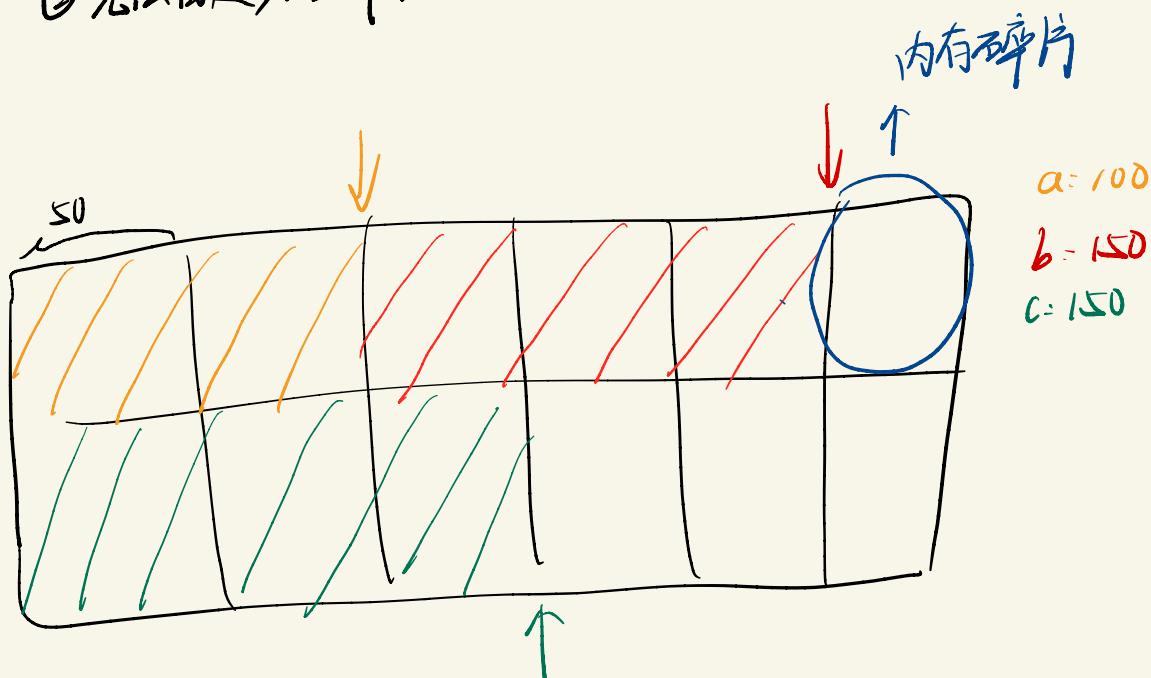
2. 优点：① 快：不需要寻址

② 没有内存碎片：按顺序存储和释放

缺点：① 小

② 无法自定义生命周期

heap:



1. 运行方式：实际内存 → 虚拟内存  
系统处理

ptr 根据编译器规定的规则（“快速寻址”、“最佳寻址”、“折衷寻址”）  
去寻找指定规格的内存

2. 维护方式：地址 + 内存块长度

2. 优点：① 内存大

② 可以自定义生命周期

e.g. C++ :

new, ① 分配内存

② 初始化

C:

malloc : 分配内存

free : 释放内存

delete : ① destruct : 删除内存中的数据

② free

C++ 独有

缺点：① 慢

② 有内存碎片

# heap 与 stack 区别

1. heap：对虚拟内存操作，无法把变量名与内存绑定。

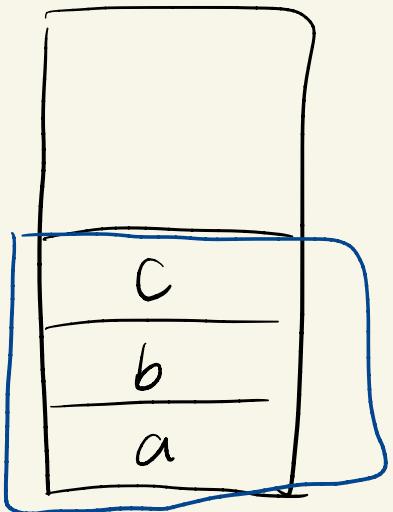
只能通过指针操作

e.g. 可以通过 ptr 对 obj a, b, c 操作

ptr →

也可以将变量名 w 与内存绑定，对 w.a, w.b, w.c

操作



2. stack：对实际内存操作，可以把变量名与内存绑定，可以通过变量

名操作

# 如何读取数据

C++: ① 地址:

① 直接寻址: 变量名  
② 间接寻址: 指针

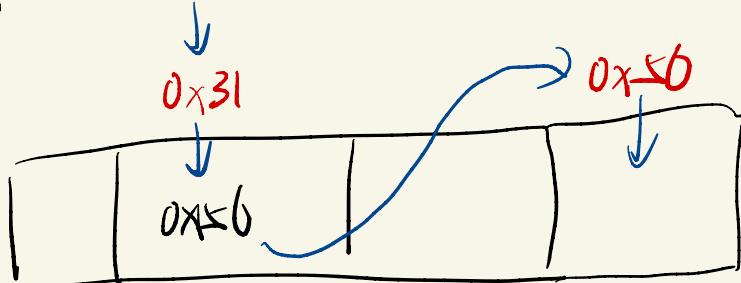
② 类型

① 数据解释方式  
② 对象的大小

一块内存

正确的数据

指针 间接寻址过程     $\text{ptr} = 0x56$



1. 找到  $0x31$  ( $\text{ptr}$ ) 中的地址  $0x56$

2. 找到  $0x56$  中的数据，用正确的解释方式解释