链表**：**

**（1）优点**

1.任意位置插入删除O（1）

2.按需申请访问

缺点：

1.不支持下标随机访问

顺序表：

（1）缺点

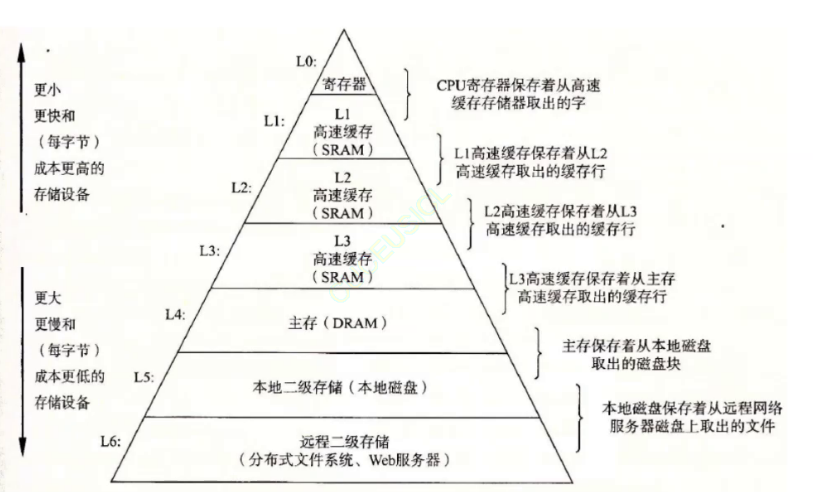
1、前面部分插入数据，效率是O（N），需要挪动数据

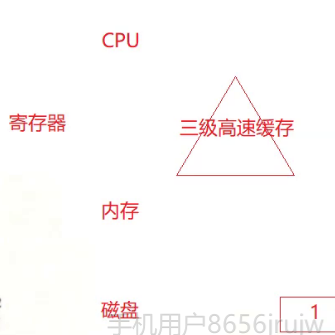
2.空间不够，需要扩容，扩容是需要付出代价的，一般会伴随空间浪费

（2）优点

1.尾差尾删效率不错

2.下标的随机访问





1.先去看看数据是否在缓存，在就交缓存命中，则直接访问

2.不在就不命中，先加载数据到缓存，再访问

**一、栈**

（1）栈的概念：线性表，规则为固定的一端插入和删除元素操作。操作的一端称为栈顶，另一端为栈底初入数据都在栈顶

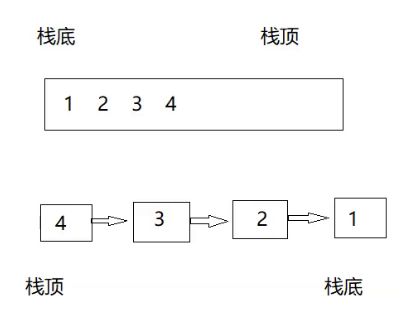
例题：

入栈顺序 1 2 3 4

出栈顺序（不唯一）4 3 2 1

1 2 3 4

1 3 2 4



数组栈

typedef int STDataType;

struct Stack

{

STDataType \* a;

int top;

//类似于size，可以指向最后一个进栈的元素，

//也可以指向最后一个元素的后一位

//对应的是空栈TOP指向 0或者-1

int capaicty;

}ST;

void STInit(ST \* pst)

{

assert(pst);

pst->a = NULL;

pst->top = 0

//pst->top = -1;

pst->capacity = 0;

}

void STDestroy(ST \* pst)

{

assert(pst);

free(pst);

pst->a = NULL;

pst->top = pst->capacity = 0;

}

void STPush(ST \* pst, STDataType x)

{

if(pst->top == pst->capacity)

{

int newCapacity = pst->capacity == 0 ? 4 : pst->capacity \* 2;

STDataType \* tmp = (STDataType \*)realloc(pst->a,newCapacity \* sizeof(STDataType));

if(tem == NULL)

{

perror("realloc fail");

return;

}

pst->a = tmp;

pst->capacity = newCapacity;

}

pst->a[pst->top] = x;

top++;

}

void STPop(ST \* pst)

{

assert(pst);

assert(!STEmpty(pst));

pst->top--;//--就行，不用抹除数据

}

bool STEmpty(ST \* pst)

{

assert(pst);

return (pst->top == 0);

}

STDataType STTop(ST \* pst)

{

assert(pst);

assert(!STEmpty(pst));

}

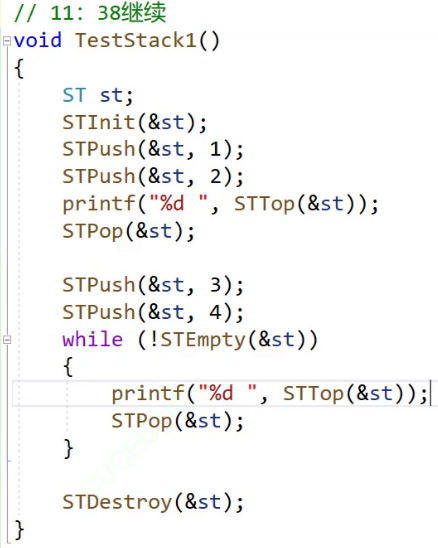
int STSize(ST \* pst)

{

assert(pst);

return pst->top;

}



（链式栈）

**二、队列**

**2.1队列的概念以及结构**

**队列：**只允许在一端进行数据插入操作，在另一端删除数据操作的特殊线性表，

先进先出的特性，

入队列：进行插入操作的一端称为队尾

出队列：进行删除操作的一端称为队头

其他用途：

DFS--深度优先遍历--递归/栈实现非递归

BFS--广度优先遍历--队列

