1. 抽象数据类型——以栈为例
2. 栈的可执行操作
3. 创建空栈
4. 将数据压入栈底
5. 从栈顶弹出数据
6. 可查看栈是否已满
7. 可查看栈是否为空
8. 栈的数据成员

0、在类创建之前定义栈的大小const int size

1. 数组stack[size]（主要的被操作对象）（同时需要定义大小）
2. Top指针（指示栈顶位置） //将top指针初始化为0可实现初始化空栈

//初始化参数表实现示例-- stack():top(0){};

1. 栈的函数实现
2. 压入数据（push）

If(top<size) stack[top++]=ch;

1. 弹出数据（pop）

Return stack[--top];

1. 栈的链表实现
2. 栈节点的声明：class C-node( int data ,C-node \*pre )

Friend class C-stack

//data储存当前数据，pre为指向前一个入栈节点地址的指针

1. 初始化：第一个数据压入空栈时使第一个数据的pre为null

同时使当前的top为null

1. 入栈： void C-stack ::push( int data ){

C-node \*p=new C-node(data , top);

//创建新的目前节点，并将指针指向当前栈顶位置

top=p; //更新栈顶指针位置

}

1. 出栈： 需要出栈的数据是当前top所指示的数据，

出栈之后top需要更新为pre，故而需要tmp来暂存top的位置

1. node \*Tmp=top; (暂存栈顶位置)

If (top) top=top-->pre; (如果栈非空，则更新栈顶指针位置)

return tmp-->data() (返回需要出栈的数据，即原来的栈顶位置)

1. 稳定性增强：

可使用预编译操作来避免重复包含的编译错误

#ifndef stack.h

#endefine stack.h

1. 析构函数
2. 定义：与构造函数对应的，当一个对象消失或new创建的对象用delete删除时，程序会自动调用特殊的成员函数完成清理，称为析构函数。
3. 特点： 析构函数的名称是在类名前加~

析构函数没有参数

析构函数也可以被重载

1. 运行：如果用户没有定义或重载析构函数，系统将会自动进行隐式声明与调用
2. 以对象为参数
3. 存在的问题：实参传入时并不是将实参本体传入，而是传入实参的copy，从而导致开销的浪费。
4. 解决方法：形参参数表中以地址或指针为形参，则传入的便是实参的地址而非copy，从而减少了赋值的开销。
5. 示例：Print(circle &c){……}；

Main(){

Print(my\_circle); //此处不加&，否则会传入变量的地址为实参

}

4、对象数组（实质与struct相同）

Class circle;

Circle c[100];

C[i].get\_area();