堆栈模拟

栈是一种数据结构，满足先进后出的规则. 入栈的元素放在栈顶，出栈时将栈顶元素出栈. 这里我们可以利用链表来实现栈. 记录链表尾，用来存栈顶元素. 当新元素入栈时，把该数据添加到链表末尾，更新链表尾. 出栈时，将链表尾前移一格，把最后的节点删掉即可.

因为链表尾需要前移，所以这里用双向链表实现. 链表节点除了记录前驱后继之外，还需要记录存储的数据.

```
//链表节点结构体
struct StackNode
{
    int val; //数据
    StackNode* pre, * nxt; //前驱 后继
};
```

对于用双向链表实现的栈来说，需要记录的有链表头和链表尾. 在新建一个栈时，需要创建链表头节点并初始化. 这里链表头节点是不记录信息的.

```
StackNode* head; //链表头节点
StackNode* top; //栈顶节点
int size; //栈大小
Stack() //构造函数
{
    head = (StackNode*)malloc(sizeof(StackNode));
    head->nxt = head->pre = nullptr;
    top = head;
    size = 0;
}
```

获取栈顶元素时，只需要把链表尾元素返回即可. 如果链表尾与链表头指向同一个元素，说明栈为空.

```
int gettop() const
{
    if (top == head) //栈为空
    {
        return 0;
    }
    return top->val;
}
```

添加元素时，只需要在链表尾后加入新节点，然后更新链表尾和栈大小.

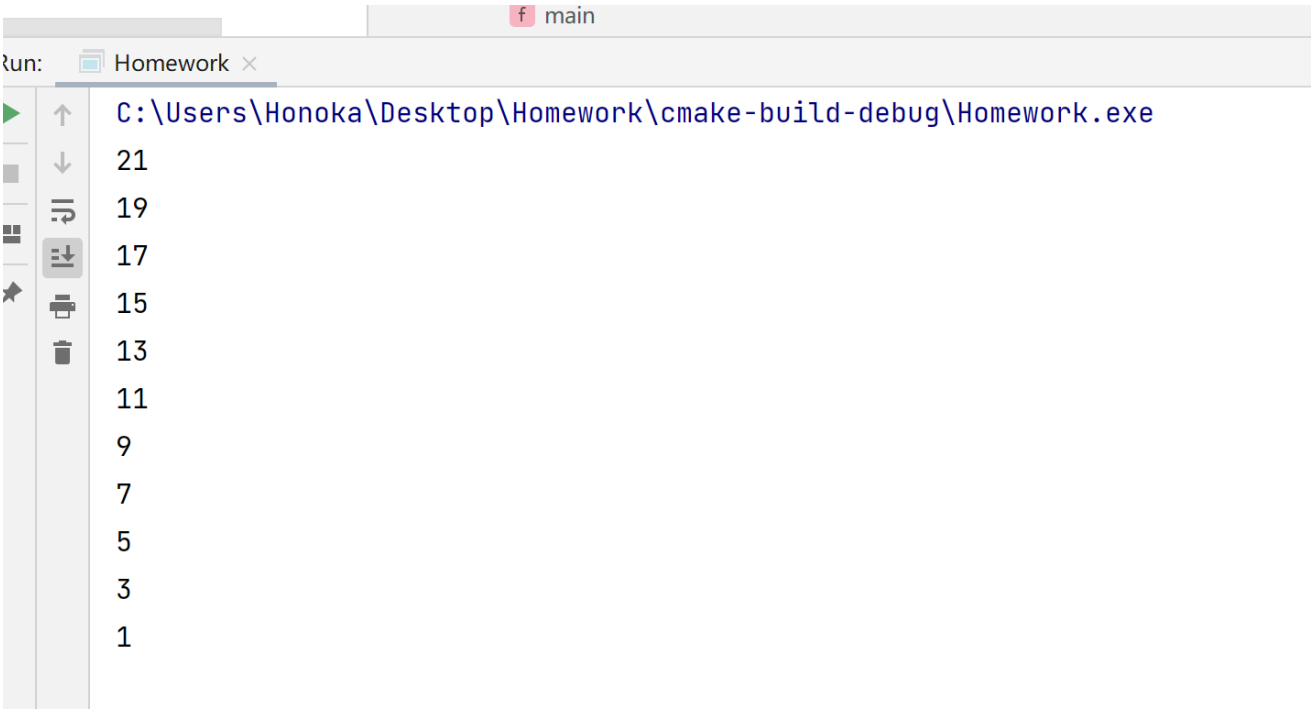
```
void push(int x)
{
    auto* t = (StackNode*)malloc(sizeof(StackNode));
    //链表尾添加节点
    top->nxt = t;
    t->pre = top;
    t->nxt = nullptr;
    t->val = x;
    top = t;
    size++;
    return;
}
```

出栈时，前移链表尾，并删除链表最后的元素.

```
void pop()
{
    if (head == top) return; //栈为空
    auto t = top;
    top = top->pre;
    top->nxt = nullptr;
    size--;
    free(t); //释放内存
    return;
}
```

用如下代码进行测试:

```
int main()
{
    Stack s;
    for (int i = 0; i <= 10; i++)
    {
        s.push(2 * i + 1);
    }
    while (s.size != 0)
    {
        cout << s.gettop() << endl;
        s.pop();
    }
    return 0;
}
```



输出结果正确.

小结

链表作为一种基本的数据结构,适合动态分配内存,缺点是不能随机访问.对于栈这种长度不固定,但是访问仅出现在栈顶的数据结构来说,用链表实现是最适合不过的了.