人工智能学院

Data Structures and Algorithms

Stacks and Queues（栈和队列）1

Assessment

1. Definition
2. ADT
3. Implementation
4. Application

Definition

◆ 停车场管理

1. 停车场特点后进先出
2. 如何存储
3. 数据操作：进场、出场

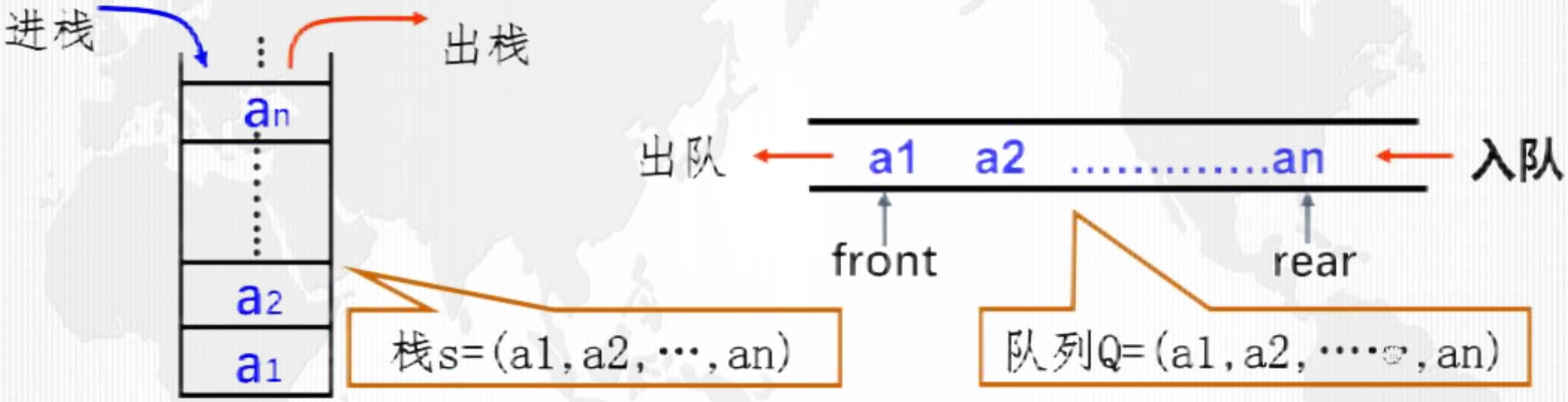
Definition

- 银行业务模拟

1. 停车场特点先进先出
2. 如何存储
3. 数据操作：到达、离开、求队伍长度

Definition

◆ 栈和队列逻辑结构



- 栈和队列是两种特殊的线性表
- 栈和队列是操作受限的线性表，称为限定数据结构（限定插入和删除只能在表的端点进行的线性表）

Definition

- 栈的定义及特点
  - 定义：栈是只允许在一端进行插入或者删除操作的线性表（仅在线性表的表尾进行插入或删除操作的线性表）
  - 特点：先进后出（FILO）或者后进先出（LIFO）
- 栈的操作：
  - 置空栈
  - 取栈顶元素
  - 判空栈
  - 入栈
  - 出栈

Python中的栈（list）相关C语言实现

https://github.com/python/cpython/blob/main/Objects/listobject.c

Abstract Data Type

ADT List {

Data object（数据对象）：$D = a_i | a_i \in ElemSet, i = 1, 2, \ldots, n, n \geq 0$

Data relation（数据关系）：$R_1 = \{< a_{i-1}, a_i > | a_{i-1}, a_i \in D, i = 2, 3 \ldots, n\}$

约定a_n为栈顶，a_1为栈地

Operations:

initStack(*s)：构造一个空的栈

destroyStack(*s):

pre-condition(初始条件): s is not NULL

Result: make s NULL

clearStack(*s):

pre-condition(初始条件): l is not NULL

Result: make l an empty list

stackEmpty(*s):

pre-condition(初始条件): l is not NULL

Result: if l is empty, returns TRUE, else returns FALSE

stackLength(*l):

pre-condition(初始条件): l is not NULL

Result: returns the number of elements in l

Abstract Data Type

```
        getTop(*s, *e):
            pre-condition(初始条件): s is not NULL
            Result: assign the value of the top element in s to e
        push(*s, e):
            pre-condition(初始条件): s is not NULL
            Result: push element e to the top of stack s
        pop(*s, *e):
            pre-condition(初始条件): s is not NULL
            Result: remove top of stack s and return its value in e
}
```

Implementation

- 栈的存储结构

  1. 顺序栈 2. 链栈

- 顺序栈的定义

```
#define STACK_INT_SIZE 100
#define STACK_INCREMENT 100

typedef int elemType;
typedef struct Stack {
    elemType *base;      // 栈底指针
    elemType *top;       // 栈顶指针
    int stackSize;       // 当前分配的存储容量
} SqStack;
```

Implementation

- 栈的存储结构

  1. 顺序栈 2. 链栈

- 顺序栈的定义

```
#define STACK_INT_SIZE 100
#define STACK_INCREMENT 100

typedef int elemType;
typedef struct Stack {
    elemType *base;      // 栈底指针
    elemType *top;       // 栈顶指针
    int stackSize;       // 当前分配的存储容量
} SqStack;
```

Implementation

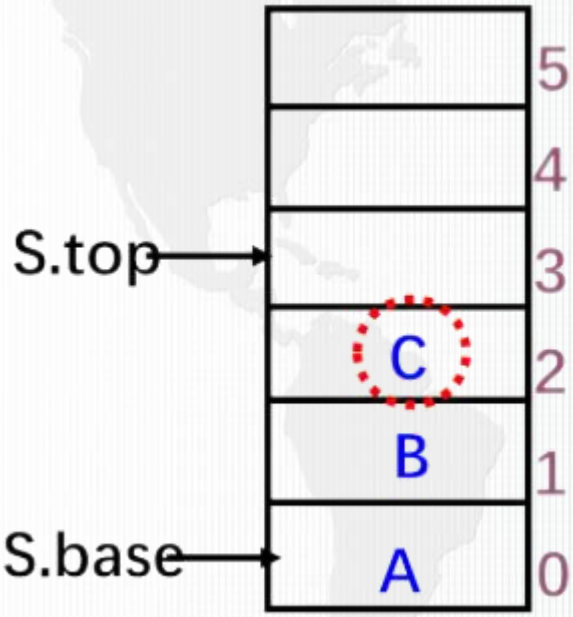
◆ 构造一个空顺序栈

```
Status init_stack(SqStack *s) {
    s->base = (elemType *) malloc( size: STACK_INT_SIZE *sizeof (elemType));
    if(!s->base) {
        return OVERFLOW;
    }
    s->top = s->base;
    s->stackSize = STACK_INT_SIZE;
    return OK;
}
```

Implementation

◆ 取栈顶元素

```
Status getTop(SqStack s, elemType *e) {
    if (s.top == s.base) {
        return ERROR;
    }
    *e = *(s.top - 1);
    return OK;
}
```



Implementation

◆ 构造一个空顺序栈

```
Status init_stack(SqStack *s) {
    s->base = (elemType *) malloc( size: STACK_INT_SIZE *sizeof (elemType));
    if(!s->base) {
        return OVERFLOW;
    }
    s->top = s->base;
    s->stackSize = STACK_INT_SIZE;
    return OK;
}
```

Implementation

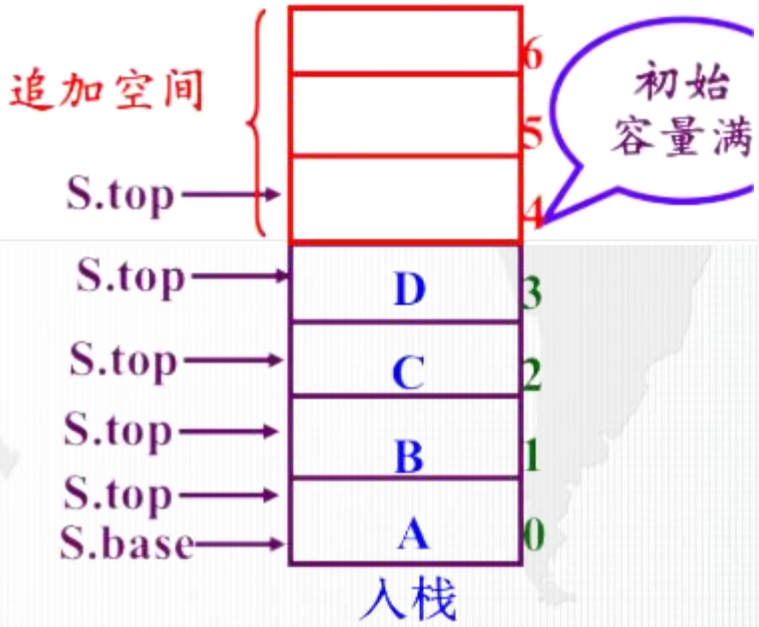
◆ 构造一个空顺序栈

```
Status init_stack(SqStack *s) {
    s->base = (elemType *) malloc( size: STACK_INT_SIZE *sizeof (elemType));
    if(!s->base) {
        return OVERFLOW;
    }
    s->top = s->base;
    s->stackSize = STACK_INT_SIZE;
    return OK;
}
```

Implementation

◆ 顺序栈入栈

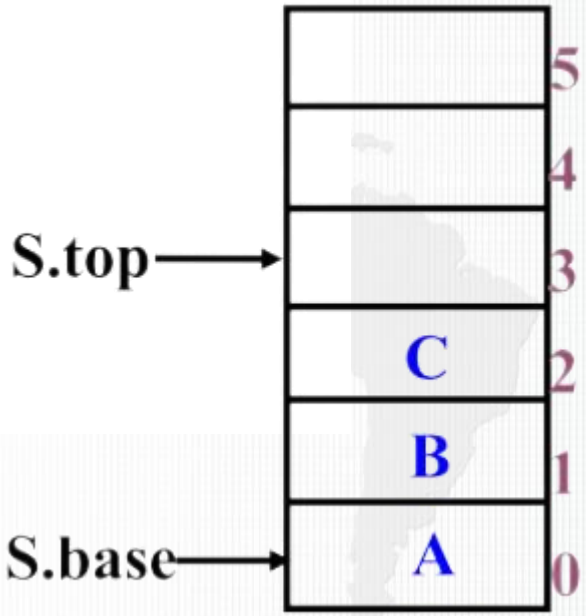
```
Status push(SqStack *s, elemType e) {
    if (s->top - s->base >= s->stackSize) {
        s->base = (elemType *) realloc( ptr: s->base, size: ((s->stackSize + STACK_INCREMENT) * sizeof (elemType)));
        if (!s->base) {
            return OVERFLOW;
        }
        s->top = s->base + s->stackSize;
        s->stackSize += STACK_INCREMENT;
    }
    *s->top++ = e;
    return OK;
}
```



Implementation

◆ 顺序栈出栈

```
51  }
52
53  Status pop(SqStack *s, elemType *e) {
54      if (s->top == s->base) {
55          return ERROR;
56      }
57      *e = *(--s->top);
58      return OK;
59  }
```



Practice

◆ 能否由入栈序列A, B, C, D, E得到出栈序列：C, B, D, A, E

出栈序列:

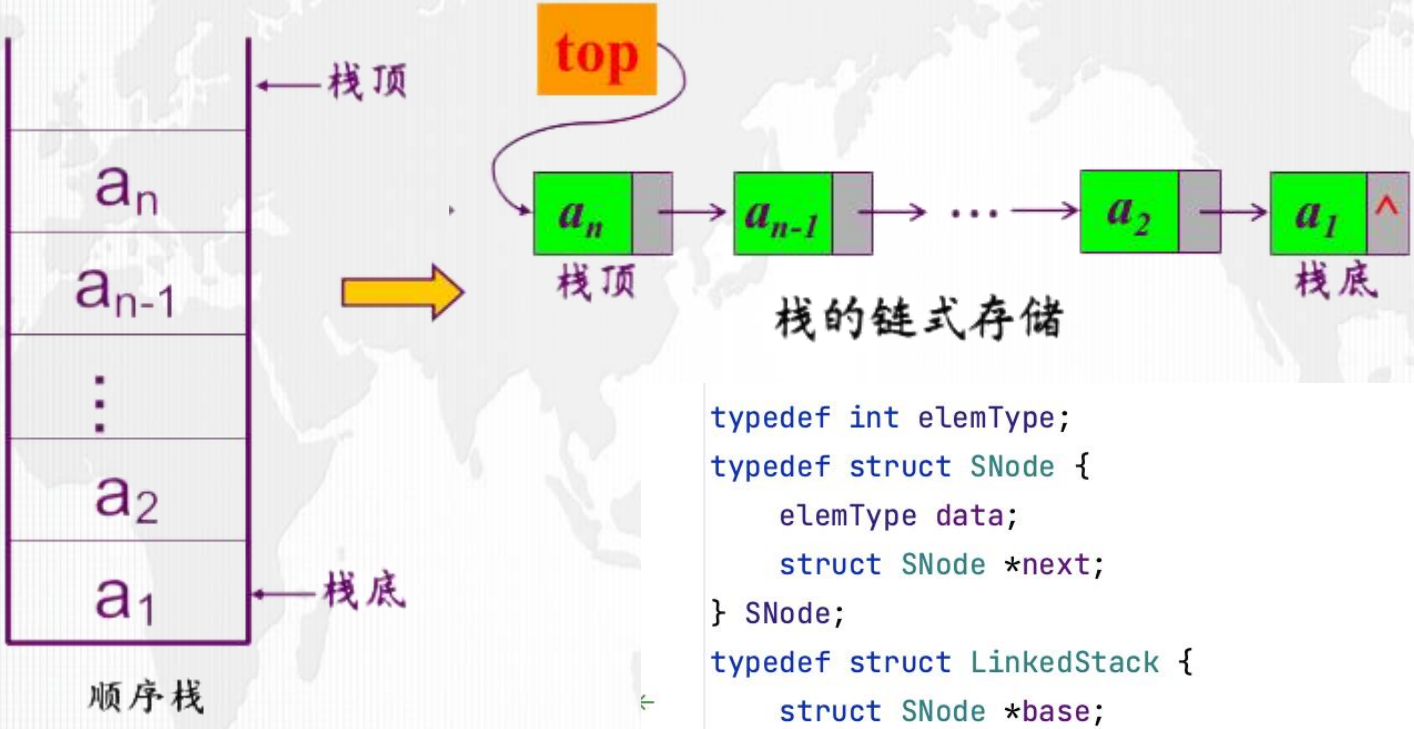
C B D A E

操作序列:

① 元素A入栈
② 元素B入栈
③ 元素C入栈
④ 元素C出栈
⑤ 元素B出栈
⑥ 元素D入栈
⑦ 元素D出栈
⑧ 元素A出栈
⑨ 元素E入栈
⑩ 元素E出栈

Implementation

◆ 链栈的定义




```
typedef int elemType;
typedef struct SNode {
    elemType data;
    struct SNode *next;
} SNode;
typedef struct LinkedStack {
    struct SNode *base;
    struct SNode *head;
} LinkedStack;
```

链栈通常不需要头节点

Implementation

- 构造一个空链栈
- 获取栈表第一个元素

```
Status init_stack(LinkedStack *s) {
    s->base = s->head = NULL;
    return OK;
}

Status get_top(LinkedStack *s, elemType *e) {
    if (s->head == NULL) {
        return ERROR;
    }
    *e = s->head->data;
    return OK;
}
```

Implementation

- 链栈的Push和Pop操作

```
Status push(LinkedStack *s, elemType e) {
    SNode *newNode = (SNode*) malloc( size: sizeof(SNode));
    if (newNode == NULL) {
        return OVERFLOW;
    }
    newNode->data = e;
    newNode->next = s->head;
    s->head = newNode;
    if (s->base == NULL) {
        s->base = newNode;
    }
    return OK;
}

Status pop(LinkedStack *s, elemType *e) {
    if (s->head == NULL) {
        return ERROR;
    }
    SNode *temp = s->head;
    *e = temp->data;
    s->head = s->head->next;
    free(temp);
    return OK;
}
```

Application

- 栈的应用
  - 函数的嵌套调用
  - 递归的实现
    - 递归函数
    - 汉诺塔问题
    - 迷宫问题
    - 回文游戏
    - 多进制转换
    - 表达式求值
    - 地图四染色

Application

◆ 栈的应用：函数的嵌套调用

```
main( )
{   int  m,n;
     . ..........
    first(m,n);
     ①:..............
}
void  first(int  s, int  t)
{   int   i;
    second(i);
     ②:.............
}
void  second(int  d)
{   int x,y;
     .............
}
```

★调用函数时，编译系统需要：
① 将返回地址及实参等信息，传递给被调用函数保存；
② 为被调用函数局部变量分配存储区；
③ 将控制权转移到被调用函数入口；

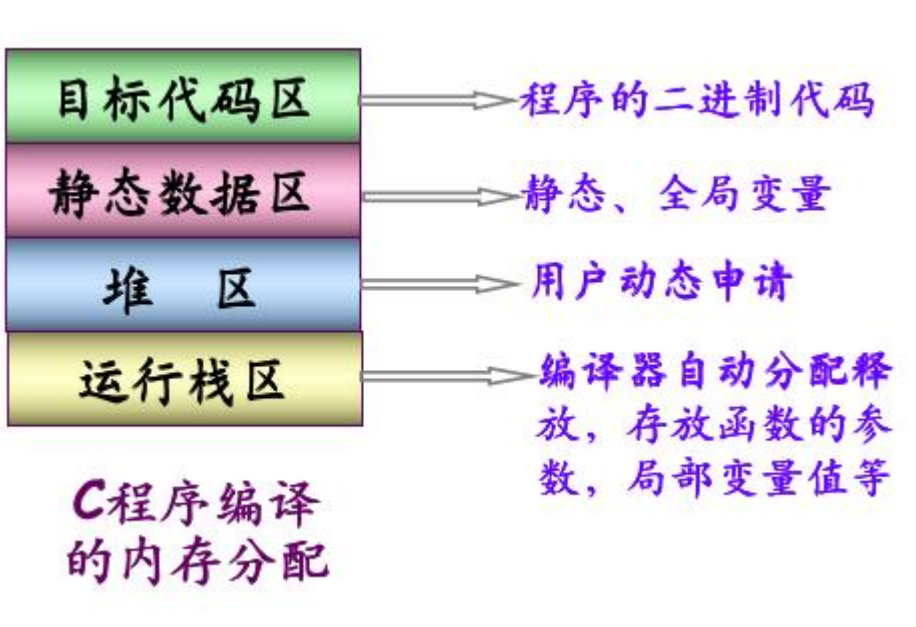
★被调用函数返回时，编译系统需要：
① 保存被调用函数计算结果；
② 释放被调用函数的数据区；
③ 依照保存的返回地址将控制权转移到调用函数。

★函数之间信息传递和控制转移必须通过“栈”来实现。

Application

◆ 栈的应用：函数的嵌套调用

```
main( )
{   int  m,n;
     ..........
    first(m,n);
     ①:..............
}
void  first(int  s, int  t)
{   int   i;
    second(i);
     ②:.............
}
void  second(int  d)
{   int x,y;
     .............
}
```



Application

◆ 栈的应用：递归过程及实现

递归：函数直接或间接调用自身
实现：建立递归工作栈

从前有座山，山上有座庙，庙里有个老和尚，老和尚给小和尚讲故事，讲的是：从前有座山，山上有座庙，庙里有个老和尚，老和尚给小和尚讲故事，讲的是。。。

Application

◆ 栈的应用：递归过程及实现

使用递归的三种情况：

1. 定义是递归的：Fibonacci数列、阶乘
2. 数据结构是递归的：链表、二叉树
3. 问题的解法是递归的
   - Hanoi塔问题
   - 八皇后问题
   - 迷宫问题

Homework

习题：

课本算法设计题（1）（3）

在main函数中提供对应的test case

PS：

在gitee上提交源代码（以xxx.c命名）

截止时间（9月21日24:00）

Thanks!

See you in the next session!