ساختمان داده‌ها و الگوریتم‌ها
Data Structures and Algorithms

دانشگاه دامغان

مدرس: علی متقی

ساختمان داده‌ها- مقدمه

صف و پشته : نوعي ليست
ليست

- لیست: مجموعه ای از عناصر که رابطه آنها رابطه ترتیب است، مانند لیست روزهای هفته، اسامی دانشجویان کلاس،...
- صف (Queue): یک لیست دوطرفه، FIFO: First In First Out
- پشته(Stack): یک طرفه، LIFO: Last In First Out

صف

- FIFO
- پیاده سازی صف: آرایه، لیست پیوندی
- آرایه: ظرفیت: size
- دسترسی به عناصر: f,r
- منابع رایانه ای: حافظه، پردازشگر،...
- برنامه های کاربر
- سیستم عامل،

Stack ADT

- Objects:….?
- Methods:
  - constructor
  - isEmpty(): boolean
  - isFull(): Boolean
  - Push(object)
  - pop(): Object
  - Top(): Object

```
Class  Stack{
    int  size= ٢٥ ;
    int  top;
    Object elements[];
    stack(){
        elements=new …
        top=..
    }
    stack(int  len){
      size=len;
    elements=new …
    top=..
    }
                        void
push(object  data){
    }
Object pop(){
    return
    }
Object getTop(){  return
}
bool   isfull(){
    …
    }
bool   isEmpty(){
    }
} // end of class

S1= new Stack(20)
```

Stack

- ترتیب بیان مطالب ساختمان داده ها (یک بحث جانبی):

شیوه ۱:

1. تعریف
2. ADT
3. پیاده سازی
4. کاربردها

شیوه ۲:

1. تعریف
2. ADT
3. کاربردها
4. پیاده سازی

کاربردهاي پشته

- استفاده از stack به عنوان یک نوع داده موجود(پیش ساخته):

Stack s1=new Stack(12); S1: Empty

Stack s2=new Stack(25);

s1.push(51) S1: 51

s1.push(11) S1:51 11

X=s1.pop() S1: 51 , X←11

If (s1.isfull()) ….if(False)

X= s1.top() s1:51, X←51

Stack s1 = new Stack(6)
stack s2 =new Stack(4)

- S1.push(10), S1.push(15), S1.push(20), s2.push(14),
- S2.push(s1.top()), s2.push(s1.pop()): S1:10 15, s2:14 20 20
- s1.push(25), s1.push(s2.pop()),
- Print(S1.pop()), s2.pop()
- Stack underflow,
- Stack overflow

مسأله: جایگشهای قابل تولید؟

کل جایگشتهای ممکن با سه علامت ۱ ۲ و ۳

1 2 3:

1 2 3

1 3 2

2 1 3

2 3 1

3 1 2

3 2 1

3*2*1 =6 = 3!

Push, pop

آزمونک: با در نظر گرفتن داده های ورودی ، عملیات زیر را پشت سر هم انجام دهید و در هر مرحله وضعیت پشته را نشان دهید. A b c d e

۱۰ دقیقه زمان تا ارسال

```
Stack s1=new …..
S1.push(read())
S1.push(read())
Print(s1.top())
Print(s1.pop))
S1.Push(read())
S1.Push(read())
S1.pop()
S1.Push(read())
S1.pop()
Print(s1.pop())
S1.pop()
S1.pop()
```

Stack

- کاربردهای پشته:
- ۱ – فراخوانی توابع
- ۲- تبدیل الگوریتهای بازگشی به غیر بازگشتی
- ۳- عبارات محاسباتی و منطقی(تبدیل و ارزیابی،...)

- ۱ – فراخوانی توابع:

چهار تابع A B C D:

```
A:  main(){
….
  call B()
L1:  …
 return}
B(){  …
   call    C();
L2: …
Return  }
C(){  …
   call    D();
L3: …
    Return  }
D(){
…..
return}
```

- واقعیتها در باره فراخوانی توابع:

1. آخرین تابع فراخوانی شده اول از همه به پایان رسید
2. در زمان برگشت از تابع، نیاز به آدرس برگشت داریم. آدرس؟ آدرس بعد از دستور call
3. آخرین آدرسی که به آن رسیده ایم اول از همه به آن نیاز داریم.

- نتیجه:

1. منطق فراخوانیها و برگشت مشابه منطق(ساختار) پشته است.
2. بلافاصله قبل اجرای دستور call، باید آدرس دستور بعد آن ذخیره شود.

- نحوه مدیریت(اجرای) دستورات Call و Return:

1. از پشته استفاده می کنیم
2. در زمان اجرای call آدرس برگشت و...را در پشته ذخیره می کنیم push(..)
3. موقع برگشت آدرس برگشت را از بالای پشته برمی داریم: pop() و goto

Stack

- کاربردهای پشته:
- ۱ – فراخوانی توابع
- ۲- تبدیل الگوریتهای بازگشی به غیر بازگشتی
- **۳- پردازش عبارات محاسباتی و منطقی**

تبدیل و ارزیابی عبارات

A+B یا +A B یا A B + add min >

عبارت(Expression): مجموعه ای از عملگرها و عملوندها

یک ترکیبی درست از عملگرها و عملوندها

عملوند(operand): مانند A,B 10, a+b, sqrt(16), sqrt(16)*20

عملگر(Operator): +,-, *, ++,(),=, >, >=, &&,and

نمایش عبارات:

میانوندی(infix): A+B

پیشوندی(prefix): +A B (operator op1 op2)

(A+B)*C → * + A B C ، A+(B*C)

پسوندی(postfix): A B+ C * A B +

عبارات:

- رابطه عبارات و توابع در زبانها:
  - A.store(I,j,x) max(a,b,c) add(a,b,c) +(a,b,c) +a b c
  - Push(S,10) پیشوندی + A B
  - Prolog , lisp (+ a b)
  - X=5, 5, x, -10
- تبدیل عبارات
- ارزیابی عبارات
- A+b
- A-b
- -a
- -(a+b)

Infix: 20 *5/2^2^3-12-6+50 # ورودی :

Postfix: خروجی

Tocken	S1(پشته)	output
20		20
*	*	
5	*	20 5
/	* pop→*	20 5 *
/		20 5 *
/	/	20 5 *
2	/	20 5 * 2
^	/ ^	20 5 * 2
2	/ ^	20 5 * 2 2
^	/ ^	
^	/ ^ ^	
3	/ ^ ^	20 5 * 2 2 3
-	/ ^ ^ pop →^	20 5 * 2 2 3 ^
-	/ ^ pop →^	20 5 * 2 2 3 ^ ^
-	/ pop → /	20 5 * 2 2 3 ^ ^ /

Infix: 20 *5/2^2^3-12-6+50

x			
-	/ pop → /		20 5 2 2 3 ^ ^ /
-	push	20 5 2 2 3 ^ ^ /	
12	-		20 5 2 2 3 ^ ^ / 12
-	- pop - →		20 5 2 2 3 ^ ^ / 12
-	empty	push	20 5 2 2 3 ^ ^ / 12 –
6	-		20 5 2 2 3 ^ ^ / 12 –
	-		20 5 2 2 3 ^ ^ / 12 – 6
+	- pop - →		20 5 2 2 3 ^ ^ / 12 - 6
+	empty	push	20 5 2 2 3 ^ ^ / 12 – 6 -
	+		20 5 2 2 3 ^ ^ / 12 – 6 –
50	+		20 5 2 2 3 ^ ^ / 12 – 6 – 50
#	+ pop		20 5 2 2 3 ^ ^ / 12 – 6 – 50
empty			20 5 2 2 3 ^ ^ / 12 – 6 – 50 +

ارزيابي عبارات ميانوندي ، پسوندي، پيشوند

X= 20 *5/2^2^3-12-6+50 ; x?

دستی یا با استفاده از پشته؟

- ارزیابی عبارت پسوندی با پشته
- ارزیابی عبارت میانوندی با پشته :

۱- روش غیر مستقیم: میانوندی به پسوندی و ارزیابی پسوندی با پشته

۲- ارزیابی مستقیم:

یک پشته برای عملگرها

یک پشته برای نگهداری عموندها

Infix→ prefix

عملگر : دودویی a+b

یکانی ~ ! a-

/

۶ ۳

۳/۶

؟۶/۳

```
Class  stack{
  int  size= ٢٥ ;
  int  top;
  Object elements[];
  stack(){
        elements=new Object [size];
  top=-1;     }
  stack(int len){
    size=len;
    elements=new Object [size];
   top=-1;    }
  push(object  data){
        if isFull()
              print(“Stack Overflow”);
        else{  // top++;
               elements[++top]=data;
        }
  }
```

```
        object pop(){
                if isEmpty()   print("Stack Underflow");
                else    return  elements[top--];
        }
        object gettop(){
                if isEmpty()   print("Stack Underflow");
                else return  elements[top]
         }
        isfull(){
          if( top==size-1)   //  retrun (top==size-1)
               return 1;
        else return  0;
        }
        isEmpty(){
            return (top==-1);
        }
}
```

Queue ADT

```
Class Queue{
        int front, rear;
        int size=?
        Object elements…
        Queue(){
        }
        Queue(int maxLen){
        }
        Add(Object data){
        }
        delete(){  …  return}
        isEmpty() {return   }
        isFull() (return  }
}
```

Queue implementation

روش ۱:

```
Class Queue{
    int front, rear;
    int size=maxqsize;  for example: 100
    Object elements[];
    Queue(){
        elements= new Objects[size];
        rear= 0;    front= size-1; (OR: ??, )
    }
    Queue(int maxLen){
        size=maxLen;
        elements= new Objects[size];
        rear=  0;    front= size-1; (OR: ??, )
    }
```

پیاده سازي صف (ادامه):

```
Add(Object data){   size  0..size-1
        If isFull(){ print(“   “);
        else{ elements[rear]=data;
                rear=(rear+1) % size;
        }
}
delete(){  if isEmpty()  print(“  “);
   else{  front= (front+1)%size;
            return  elements[front];;
        }
}
```

```
    isEpmty(){ return ( (front+1)%size ==rear);
    }
    isFull(){ return (rear==front);
    }
}// end of Queue
```

Queue implementation

روش۲ (توصیه می شود)

```
Class Queue{
    int front, rear;
    int size=maxqsize;  for example: 100
    Object elements[];
    Queue(){
        elements= new Objects[size];
        rear=0;    front=0; (OR: ??, )
    }
    Queue(int maxLen){
        size=maxLen;
        elements= new Objects[size];
        rear=0;    front=0; (OR: ??, )
    }
```

پیاده سازي صف (ادامه):

```
Add(Object data){   size  0..size-1
        If isFull(){ print(“   “);
        else{ rear=(rear+1) % size;
        elements[rear]=data;
}
delete(){  if isEmpty()  print(“  “);
              else{ front=(front+1)%size;
                    return  elements[front];
}
```

```
    isEpmty(){ return (front==rear);
    }
    isFull(){ return ((rear+1)%size ==front);
    }
}// of Queue
```

صف اولویت(priority queue)

تعریف

کاربرد

پیاده سازی صف اولویت:

۱- استفاده از چند صف (آرایه) با اولویتهای متفاوت

۲- استفاده از یک صف ، که همراه با خود عنصر، اولویت عنصر هم ذخیره می شود، یا ...