HASAN KALYONCU
ÜNİVERSİTESİ

Session 5: Data Structures

Data Structures and Algorithm 1 - Lab

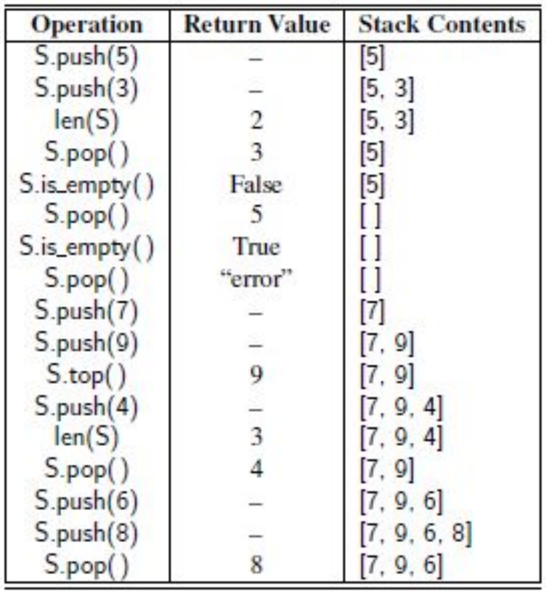
Yahya Tawil
22 Oct 2021

Last Session

- ❑ We wrote a linked list in C from scratch.
- ❑ Introducing concepts of Constructor/Destructor in C++.
- ❑ Introducing the concept of dynamic allocation (`free/malloc` and `new/delete`).

Review Stack and Queue

- Stack: First In Last Out (**FILO**). Methods: **Push** (add) and **Pop** (delete).

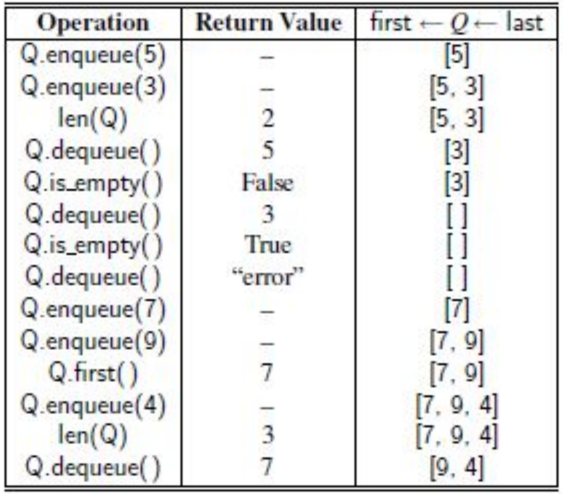


Operation	Return Value	Stack Contents
S.push(5)	–	[5]
S.push(3)	–	[5, 3]
len(S)	2	[5, 3]
S.pop()	3	[5]
S.is_empty()	False	[5]
S.pop()	5	[ ]
S.is_empty()	True	[ ]
S.pop()	“error”	[ ]
S.push(7)	–	[7]
S.push(9)	–	[7, 9]
S.top()	9	[7, 9]
S.push(4)	–	[7, 9, 4]
len(S)	3	[7, 9, 4]
S.pop()	4	[7, 9]
S.push(6)	–	[7, 9, 6]
S.push(8)	–	[7, 9, 6, 8]
S.pop()	8	[7, 9, 6]

Data Structures and Algorithms in Python-Wiley (2013)

Review Stack and Queue

- Queue: First In First Out (**FIFO**). Methods: **enqueue** (add) and **dequeue** (delete).



Operation	Return Value	first ← Q ← last
Q.enqueue(5)	–	[5]
Q.enqueue(3)	–	[5, 3]
len(Q)	2	[5, 3]
Q.dequeue( )	5	[3]
Q.is_empty( )	False	[3]
Q.dequeue( )	3	[ ]
Q.is_empty( )	True	[ ]
Q.dequeue( )	"error"	[ ]
Q.enqueue(7)	–	[7]
Q.enqueue(9)	–	[7, 9]
Q.first( )	7	[7, 9]
Q.enqueue(4)	–	[7, 9, 4]
len(Q)	3	[7, 9, 4]
Q.dequeue( )	7	[9, 4]

Data Structures and Algorithms in Python-Wiley (2013)

Basic Stack Implementation Using Arrays

```
int top=0; // empty
int stack_len ;

int* stack(int length =STACK_LEN,
int* init = NULL )
{
    int* stack_add = new
int[length];
    stack_len = length;

    if(init  != NULL)
    {
        for(int i =0;i<length;i++)
          stack_add[i] = init[i];
    }
    cout<<"stack created with
length:"<<length<<endl;
    return stack_add;}
```

```
void push(int * stack, int item)
{
    if(top >= stack_len)
    {
        cout<<"stack
overflow"<<endl;
        return;
    }
    stack[top++]=item;

}

int pop(int * stack)
{
    return stack[--top];
}
```

```
int main(){   int * S = stack(5);
    print(S);
    push(S,100);
    print(S);
    push(S,200);
    print(S);
    push(S,300);
    print(S);
    push(S,400);
    print(S);
    push(S,500);
    print(S);
    push(S,600);
    pop(S);
    print(S);
    pop(S);
    print(S);  return 0;}
```

Application of a stack

Implement a function that reverses a list of elements: [1,2,3,4,5,6,10]

```
int a[] = {1,2,3,4,5,6,10};
int * S = stack(sizeof(a)/sizeof(int));
for(int i=0;i<sizeof(a)/sizeof(int);i++)
    push(S,a[i]);

for(int i=0;i<sizeof(a)/sizeof(int);i++)
    a[i] = pop(S);

 for(int i=0;i<sizeof(a)/sizeof(int);i++)
    cout<<a[i]<<endl;
return 0;
```

Application of a stack (postfix notation)

Infix notation: **X + Y**

Postfix notation (also known as "Reverse Polish notation"): **X Y +**

Prefix notation (also known as "Polish notation"): **+ X Y**

https://www.cs.man.ac.uk/~pjj/cs212/fix.html

Application of a stack (postfix notation)

Convert this expression to postfix

(2 + 3) × 11 + 1

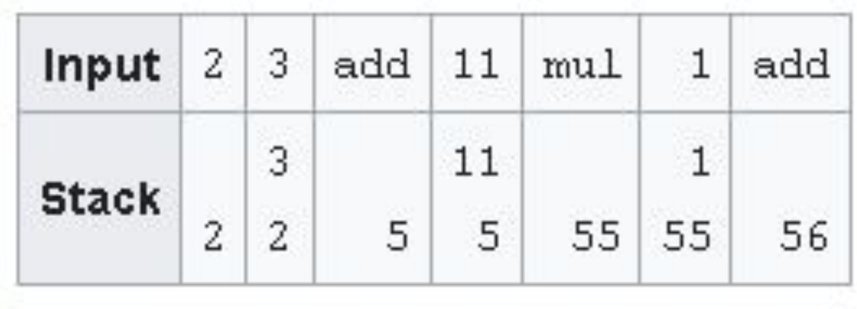
(2 3 +) × 11 + 1

T1 × 11 + 1

T1 11 × + 1

T2 + 1

T2 1 + = 2 3 + 11 * 1 +

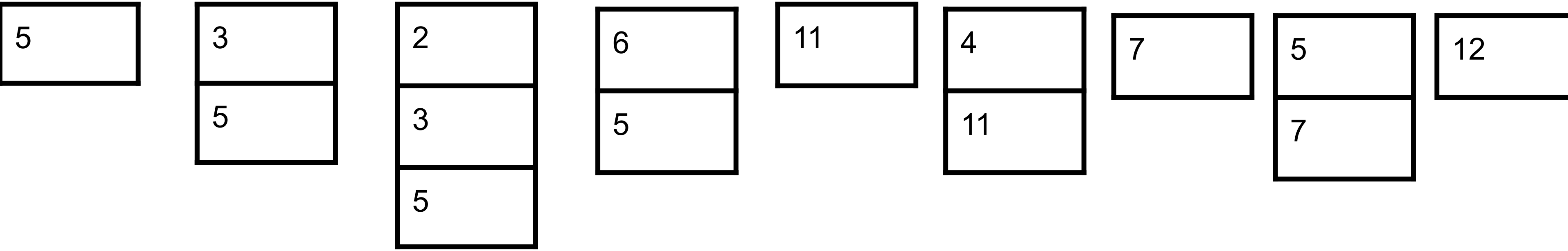


Input	2	3	add	11	mul	1	add
Stack	2	3 2	5	11 5	55	1 55	56

https://en.wikipedia.org/wiki/Stack-oriented_programming

Application of a stack (postfix notation)

Evaluate the postfix expression: 5 3 2 * + 4 - 5 +



5

3
5

2
3
5

6
5

11

4
11

7

5
7

12

Application of a stack (prefix notation)

Convert this expression to prefix

A*B-C/D+E

*A B - C/D+E

T1 - C/D + E

T1 - / C D + E

T1 - T2 + E

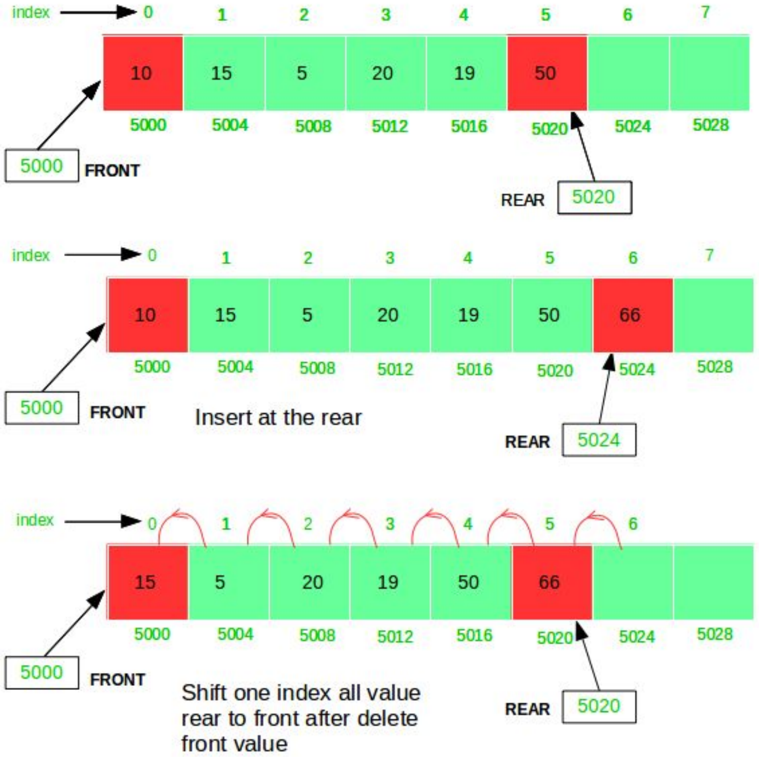
-T1 T2 + E

+T3 E = + - * A B / C D E

input	E	D	C	op/	B	A	op*	op-	op+
stack	E	D E	C D E	C/D E	B C/D E	A B C/D E	A*B C/D E	A*B - C/D E	A*B-C/D+E

https://en.wikipedia.org/wiki/Stack-oriented_programming

Basic Queue Implementation Using Arrays



index	0	1	2	3	4	5	6	7
	10	15	5	20	19	50		
	5000	5004	5008	5012	5016	5020	5024	5028

index	0	1	2	3	4	5	6	7
	10	15	5	20	19	50	66	
	5000	5004	5008	5012	5016	5020	5024	5028

Insert at the rear

index	0	1	2	3	4	5	6	
	15	5	20	19	50	66		
	5000	5004	5008	5012	5016	5020	5024	5028

Shift one index all value
rear to front after delete
front value

Basic Queue Implementation Using Arrays

```
Class Queue {
    int front, rear, capacity;
    int* queue;
public:
    Queue(int c)
    {
        front = rear = 0;
        capacity = c;            ]
        queue = new int[capacity ];
    }

    ~Queue() { delete[] queue; }
```

```
// function to insert an element
// at the rear of the queue
void queueEnqueue(int data)
{
    // check queue is full or not
    if (capacity == rear) {
        printf("\nQueue is full\n");
        return;
     }

    // insert element at the rear
    else {
        queue[rear] = data;
        rear++;
    }
    return;
}
```

```
// print queue elements
    void queueDisplay()
    {
        int i;
        if (front == rear) {
            printf("\nQueue is
Empty\n");
            return;
        }

        // traverse front to rear
and print elements
        for (i = front; i < rear;
i++) {
            printf(" %d <-- ",
queue[i]);
        }
        return;
    }
```

Finding a better implementation for Heap

❑ The last Array-based implementation did a shift for all items for each dequeue operation. **Wasting resources**.

❑ Any proposed solutions ?

Finding a better implementation for Heap

- The last Array-based implementation did a shift for all items for each dequeue operation. **Wasting resources**.

- An proposed solutions ?
  - move the front pointer instead of moving the the whole queue.
  - Use linked list and change header pointer along way.

Assignment 5: Exercise 1

- Complete the code to implement the queue using linked list.

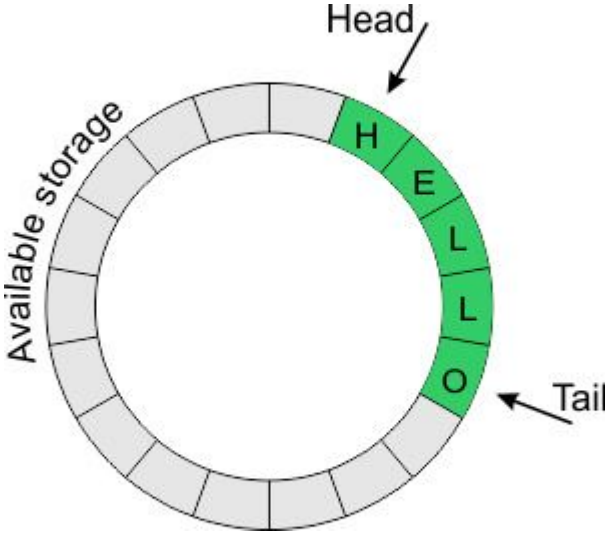
```
class Node{

    int data;
    Node* next;

    public:
    Node(int val)
    {
        data = val;
        next = NULL;
    }
};
```

```
class Linkedlist{
    Node* head;

    void insertNode(int
val)
    {
        //code here
    }
};
```

```
class queue{
    Linkedlist Lst;

    public:
    void queueEnqueue( int
data)
    {
        Lst.insertNode(data);
    }
    // rest of the code
};
```

Assignment 5: Exercise 2

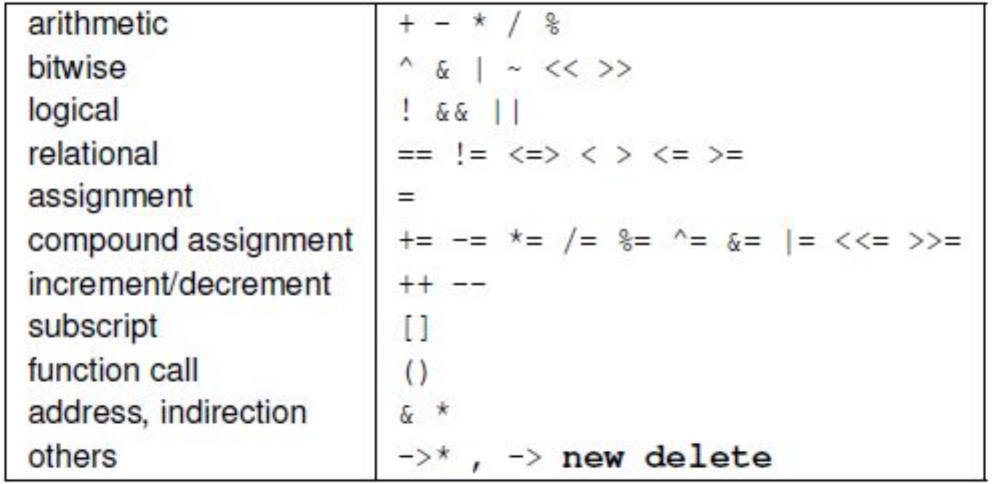
- Implement a circular buffer either in C or C++ (you’re going to be asked to stand and present next session: The concept, the operations and the implementation).



C++ Basics (Operator Overloading)

```
class Array10{
  public:
  Array10(float * init)
  {
      for(int i=0;i<10;i++) _array[i] = init[i];
  }
  float& operator[](int idx){return _array[idx];}
  private:
  float _array[10];

};
using namespace std;
int main(){
    float array[10]={0,1,2,3,4.2,5,6,7,8.5,9.1};
    Array10 arr(array);
    cout<<arr[9];
    return 0;
}
```

operators that can be overloaded:



<table>
<tr><td>arithmetic</td><td>+ - * / %</td></tr>
<tr><td>bitwise</td><td>^ & | ~ << >></td></tr>
<tr><td>logical</td><td>! && ||</td></tr>
<tr><td>relational</td><td>== != <=> < > <= >=</td></tr>
<tr><td>assignment</td><td>=</td></tr>
<tr><td>compound assignment</td><td>+= -= *= /= %= ^= &= |= <<= >>=</td></tr>
<tr><td>increment/decrement</td><td>++ --</td></tr>
<tr><td>subscript</td><td>[]</td></tr>
<tr><td>function call</td><td>()</td></tr>
<tr><td>address, indirection</td><td>& *</td></tr>
<tr><td>others</td><td>->* , -> <b>new delete</b></td></tr>
</table>

C++ Basics (Operator Overloading)

```cpp
#include <iostream>

class point{
  public:
    point(double x,double y){this->x = x ; this->y = y;}
    point(){this->x = 0 ; this->y = 0;}
    double x;
    double y;
};

point operator+(const point& p1,const point& p2)
{
  point temp;
  temp.x = p1.x+p2.x;
  temp.y = p1.y + p2.y;
  return temp;
}
```

```cpp
using namespace std;
int main(){
    point pnt1(1,8);
    point pnt2(7,2);
    point pnt3 = pnt1 + pnt2;

    cout<<pnt3.x<<","<<pnt3.y;
    return 0;
}
```

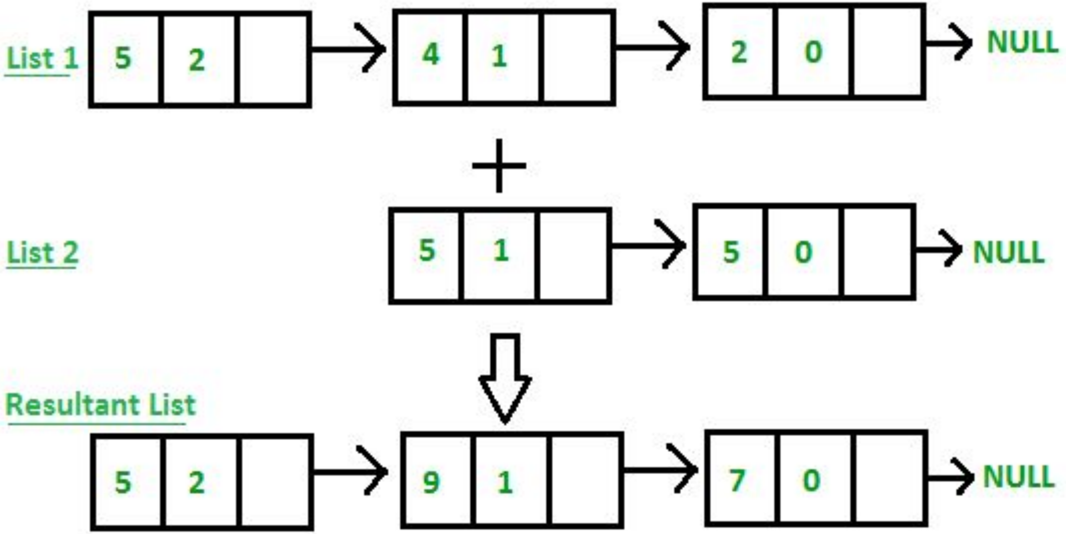
[]Operator Overloading for Linked List

```
int operator[](int idx)
    {
        Node* temp1 = head;
        int ListLen = 0;
        while (temp1 != NULL) {
            temp1 = temp1->next;
            ListLen++;
        }
        // Check if the position to be
        // deleted is less than the length
        // of the linked list.
        assert (ListLen > idx);
        temp1 = head;
        while(idx -- > 0)
        {
            temp1 = temp1->next;
        }
        return temp1->data;}
```

```
int main(){

    Linkedlist List;
    List.insertNode(10);
    List.insertNode(20);
    List.insertNode(30);
    int val = List[2];
    cout<<val<<endl;
}
```

Assignment 5: Exercise 3

- Extend your code in exercise 1 using operator overloading. Overload addition operator + to be able to add the data of 2 lists together like the following:



Single Linked List in C++ std (***optional*** material)

- Forward list in STL implements singly linked list. Introduced from C++11.
- https://www.cplusplus.com/reference/forward_list/forward_list/
- `for(int&d:list)` is called Range-based for loop introduced in C++11.

```
#include <iostream>
#include <forward_list>

using namespace std;

int main()
{
    forward_list <int> list;
    list.assign({1,2,3,4});
    list.remove(2);
    for(int&d:list)
    {cout<<d<<",";}
    cout<<endl;
    return 0;
}
```

Stack and Queue in C++ stl (***optional*** material)

```
#include <queue>
using namespace std;
// Driver Code
int main()
{
    queue<int> gquiz;
    gquiz.push(10);
    gquiz.push(20);
    gquiz.push(30);

    cout << "\ngquiz.size() : " << gquiz.size();
    cout << "\ngquiz.front() : " << gquiz.front();
    cout << "\ngquiz.back() : " << gquiz.back();

    cout << "\ngquiz.pop() : ";
    gquiz.pop();
      return 0;}
```

```
#include <stack>
using namespace std;
int main() {
    stack<int> stack;
    stack.push(21);
    stack.push(22);
    stack.push(24);
    stack.push(25);

         stack.pop();
    stack.pop();

    while (!stack.empty()) {
        cout << ' ' << stack.top();
        stack.pop();
    }
}
```