Introduction to Programming Discussion 2

Andrew Jianzhong Liu

ShanghaiTech University SIST

liujzh@shanghaitech.edu.cn

December 10, 2016

Overview

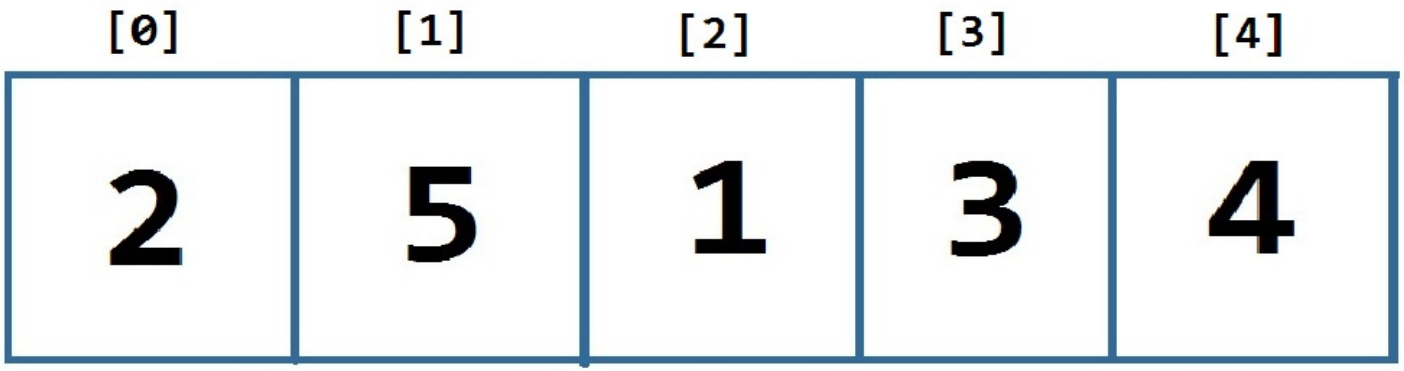
1. Simple Data Structures
   - Arrays
   - Strings
   - Stacks
   - Queues
   - Heaps
2. Basic Algorithms
   - Sorting
   - Searching
3. Exercises
4. Supplementary Information

Arrays

Concept

An array data structure is a data structure consisting of a collection of elements (values or variables), each identified by at least one array index or key. The simplest type of data structure is a linear array, also called one-dimensional array.

Graphical Representation



Arrays Cont.

Usage in C

```c
#include <stdio.h>

int main(){
  int a[5],i=0;
  while(i < 5){
    a[i] = i;
    i++;
  }
  for(i = 0; i < 5; i++)
    printf("%d ",a[i]);
  printf("\n");
  return 0;
}
```

array.c

Strings

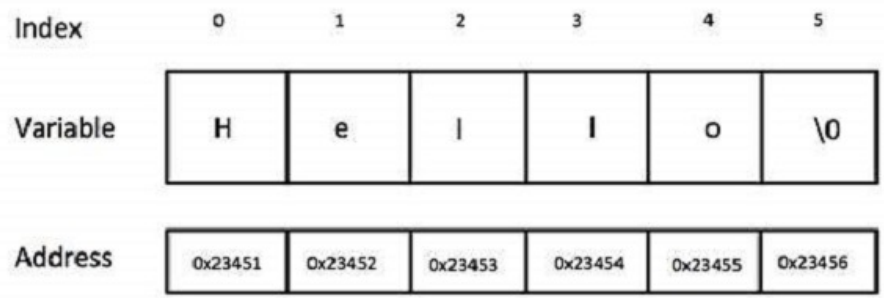
Concept

A string is a sequence of characters, usually implemented as an **array** of **bytes**.

ASCII

American Standard Code for Information Interchange, is a character encoding standard. ASCII codes represent text in computers, telecommunications equipment, and other devices.

Graphical Representation



Strings Cont.

Usage in C

```c
#include <stdio.h>

int main(){

  char str1[] = "Hello1!";
  char * str2 = "Hello2!";

  printf("%s\n",str1);
  printf("%s\n",str2);

  return 0;
}
```

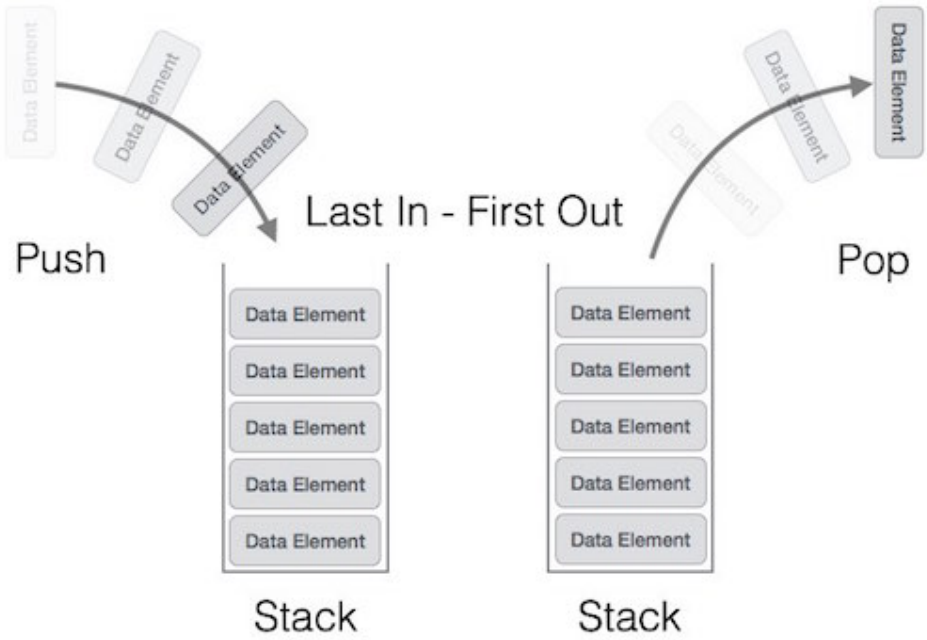
string.c

Stack

Concept

LIFO: Last In First Out.

Graphical Representation



Implementation in Python

Code

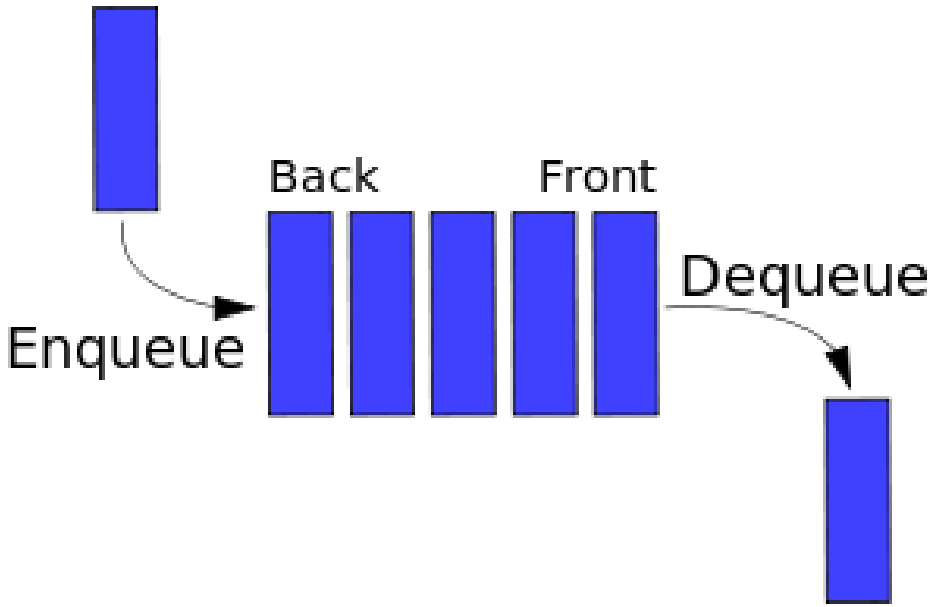
Refer to stack.py.

Queue

Concept

Stack is LIFO, Heap is FIFO (First in First Out)

Graphical Representation



Implementation in Python

Refer to queue.py

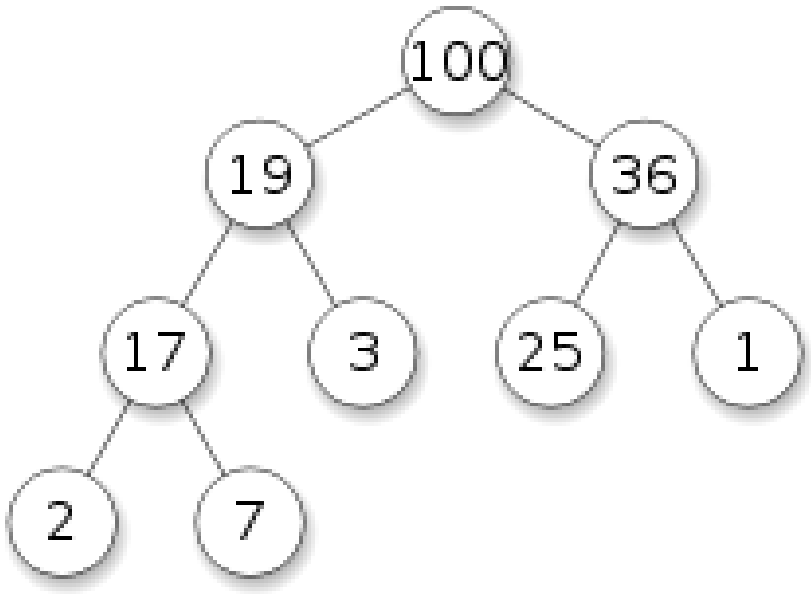
Binary Heaps

Properties

1. Shape Property: a binary heap is a complete binary tree; that is, all levels of the tree, except possibly the last one (deepest) are fully filled, and, if the last level of the tree is not complete, the nodes of that level are filled from left to right.
2. Heap property: the key stored in each node is either greater than or equal to or less than or equal to the keys in the node's children, according to some total order.

Binary Heaps Cont.

Graphical Representation



Sorting

Concept

A sorting algorithm is an algorithm that puts elements of a list in a certain order.

Common Sort Algorithms

1. Selection Sort
2. Bubble Sort
3. Merge Sort
4. Heap Sort
5. Quick Sort

Selection Sort

Concept

Find the smallest or largest element in the unsorted list and append it to the sorted list.

Example

Array: [13, 45, 23, 7, 1, 56]

Speed

Time complexity $O(n^2)$, space complexity constant.

Bubble Sort

Code implementation

```
n = int(input())
nums = []

for i in range(n):
  nums.append(int(input()))

for i in range(n):
  for j in range(1,n - i):
    if(nums[j-1] > nums[j]):
      tmp = nums[j-1]
      nums[j-1] = nums[j]
      nums[j] = tmp

for i in range(n):
  print(nums[i],end = ' ')
print()
```

bubblesort.py

Merge Sort

Procedure

Input array with length n:

1. If $n \leq 1$ then return the array
2. Do merge sort on the subarray from 0 to $\frac{n}{2}$
3. Do merge sort on the subarray from $\frac{n}{2} + 1$ to n
4. Initialize two pointers pointing to the start of the two arrays.
5. Make a new array with size n
6. While both pointers haven't reached the end of their respective arrays:
7. Dereference the pointers and compare the values
8. Copy the value to the new array and increment the pointer
9. When one pointer reaches the end, copy the remaining values to the new array

Linear Search

Concept

Linear search is a method for finding a target value within a list. It sequentially checks each element of the list for the target value until a match is found or until all the elements have been searched.

Python implementation

```
def linearsearch(arr,key):
    for element in arr:
        if element == key:
            return True
    return False

arr = [1,2,3,4,5,6]
print(linearsearch(arr,3))
print(linearsearch(arr,9))
```

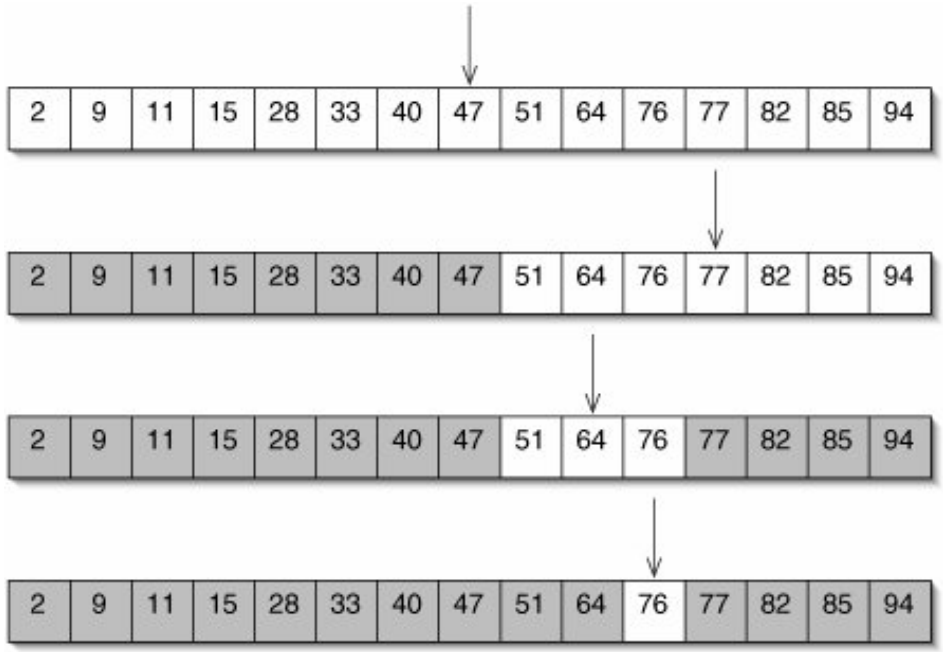
linearsearch.py

Binary Search

Concept

Binary search is a search algorithm that finds the position of a target value within a **sorted array**.

Visualization



Binary Search Cont.

Python Implementation

```
1  def binarysearch(arr, key):
       head = 0
3      tail = len(arr)-1
       while(head <= tail):
5          mid = (head + tail) // 2
           if arr[mid] == key:
7              return True
           else:
9              if key < arr[mid]:
                   tail = mid-1
11             else:
                   head = mid+1
13     return False

15 arr = [1,2,3,4,5,6,7,8]
   print(binarysearch(arr,4))
17 print(binarysearch(arr,10))
```

Exercise 1: WERTYU

Problem Description

A typesetter shifted his hand one key to the right. Given the sentences he has written, print out the original sentence.

Sample Input

O S, GOMR YPFSU

Sample Output

I AM FINE TODAY

Test input

EJP STR UPI

Exercise 2: Postfix Calculator

Problem Description

Reverse Polish Notation is a mathematical notation in which every operator follows all of its operands, for instance $2 + 1$ would be written as $2\ 1\ +$. Given a postfix expression, conduct an evaluation.

Sample Input

5 4 + 6 * 3 - 2 +

Sample Output

53

Program Stack and Heap

Why do we care?

1. Classes on Rust start tomorrow...
2. Learning Rust will require the knowledge of program stacks and heaps

Classes on Rust start tomorrow....

Contents of call stack

- Locally declared variables in functions
- Return address of previous functions.
- Other things managed by the program itself

Contents of program heap

- Dynamically allocated variables declared using malloc() or calloc() and released by free()

Program Stack and Heap Cont.

Code Examples in C

```c
#include <stdio.h>
#include <stdlib.h>

int main(){
  int a = 1, b = 2;
  int c[2];
  int * d = malloc(sizeof(int)*2);
  c[0] = a; c[1] = b;
  d[0] = a; d[1] = b;
  printf("c[] = [%d,%d]\nd[] = [%d,%d]\n",c[0],c[1],d[0],d
    [1]);
  free(d);
  return 0;
}
```

pstackheap.c

Program Stack and Heap Cont.

Comparison of Stack and Heap

Stack

- Auto allocation of variables
- Space managed by program and CPU
- Local variables only
- Size limited
- Variable sizes cannot be changed

Heap

- Manual allocation and frees, space defined by manual declaration
- Globally access guaranteed by pointer
- Bigger Sizes
- Memory fragmentation

Any Questions?

Thanks for coming!