Week 4 Lab 1

**1. Explain the below**

**a. Data structures**

Data structures are used to store data in a computer in an organized form. In C Programming Language Different types of data structures are; Array, Stack, Queue, Linked List, Tree. In term of computer programming language, a data structure may be selected or designed to store data for the purpose of working on it with various algorithms.

There are two types of data structures.

* Primitive data structure
* Non-primitive data structure

Primitive Data structure:

The primitive data structures are primitive data types. The int, char, float, double, and pointer are the primitive data structures that can hold a single value.

Non Primitive Data structure:

Non Primitive Data structure emphasize on grouping same or different data items with relationship between each data item.

The non-primitive data structure is divided into two types:

* Linear data structure
* Non-linear data structure

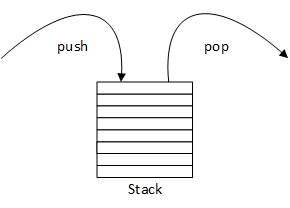
**b. Differences between linear data structures and non-linear data structures.**

|  |  |  |
| --- | --- | --- |
| **BASIS** | **LINEAR DATA STRUCTURE** | **NON-LINEAR DATA STRUCTURE** |
| Basic | The data items are arranged in an orderly manner where the elements are attached adjacently. | It arranges the data in a sorted order and there exists a relationship between the data elements. |
| Traversing of the data | The data elements can be accessed in one time (single run). | Traversing of data elements in one go is not possible. |
| Ease of implementation | Simpler | Complex |
| Levels involved | Single level | Multiple level |
| Examples | Array, queue, stack, linked list, etc. | Tree and graph. |
| Memory utilization | Ineffective | Effective |

**c. Stacks with diagrammatic representation (include a separate diagram for each operation of the stack).**

A stack is an Abstract Data Type (ADT), commonly used in most programming languages. It is named stack as it behaves like a real-world stack, for example – a deck of cards or a pile of plates, etc.

Diagrammatic Stack representation:



Stack basic operations:

1) push()

- Check if the stack is full.

- If the stack is full, then display "Stack overflow".

- If the stack is not full, increment top to the next location.

- Assign data to the top element.



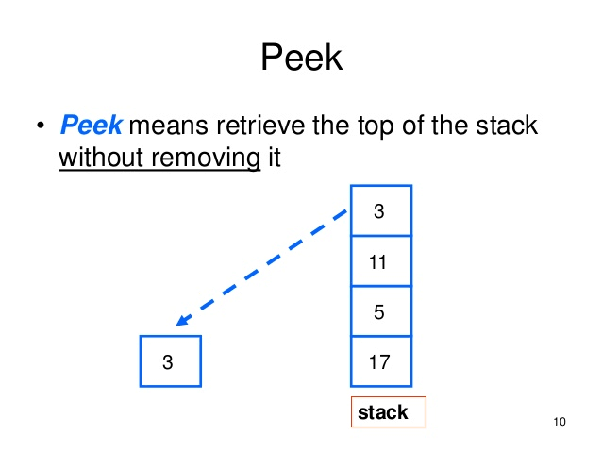
1. Pop()

* Check if the stack is empty.
* If the stack is empty, then display "Stack Underflow".
* If the stack is not empty, copy top in a temporary variable.
* Decrement top to the previous location.
* Delete the temporary variable.



1. Peek()

* Print the top most element from the stack



**d. Differences between Linked lists and Stacks**

|  |  |
| --- | --- |
| **Stack** | **Linked List** |
| An abstract data type that serves as a collection of elements with two principal operations which are push and pop | A linear collection of data elements whose order is not given by their locations in memory |
| Push, pop and peek are the main operations performed in the stack. | Insert, delete and traversing are the main operations performed in linked list. |
| Stack can read the topmost element | Linked list is required to traverse through each element to access a specific element |
| Works according to the FIFO mechanics | Elements connect to each other by references |
| Simpler than linked list | More complex than stack |

**e. Give two applications of stacks.**

- Stacks can be used for expression evaluation.

- Stacks can be used to check parenthesis matching in an expression.

**2. (Reversing the Words of a Sentence) Write a program that inputs a line of text and uses a stack to print the line reversed.**

**Code:**

#include <stdio.h>

#include <string.h>

#define MAX 100 /\*maximum no. of characters\*/

/\*stack variables\*/

int top=-1;

int item;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*string declaration\*/

char stack\_string[MAX];

/\*function to push character (item)\*/

void pushChar(char item);

/\*function to pop character (item)\*/

char popChar(void);

/\*function to check stack is empty or not\*/

int isEmpty(void);

/\*function to check stack is full or not\*/

int isFull(void);

int main()

{

char str[MAX];

int i;

printf("Input a string: ");

scanf("%[^\n]s",str); /\*read string with spaces\*/

/\*gets(str);-can be used to read string with spaces\*/

for(i=0;i<strlen(str);i++)

pushChar(str[i]);

for(i=0;i<strlen(str);i++)

str[i]=popChar();

printf("Reversed String is: %s\n",str);

return 0;

}

/\*function definition of pushChar\*/

void pushChar(char item)

{

/\*check for full\*/

if(isFull())

{

printf("\nStack is FULL !!!\n");

return;

}

/\*increase top and push item in stack\*/

top=top+1;

stack\_string[top]=item;

}

/\*function definition of popChar\*/

char popChar()

{

/\*check for empty\*/

if(isEmpty())

{

printf("\nStack is EMPTY!!!\n");

return 0;

}

/\*pop item and decrease top\*/

item = stack\_string[top];

top=top-1;

return item;

}

/\*function definition of isEmpty\*/

int isEmpty()

{

if(top==-1)

return 1;

else

return 0;

}

/\*function definition of isFull\*/

int isFull()

{

if(top==MAX-1)

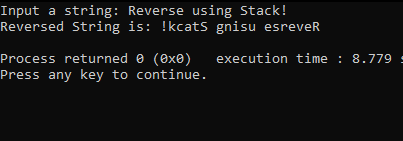
return 1;

else

return 0;

}

**Output:**



1. **(Palindrome Tester) Write a program that uses a stack to determine whether a string is a palindrome (i.e., the string is spelled identically backward and forward). The program should ignore spaces and punctuation.**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 50

int top = -1, front = 0;

int stack[MAX];

void push(char);

void pop();

void main()

{

int i, choice;

char s[MAX], b;

while (1)

{

printf("1-enter string\n2-exit\n");

printf("enter your choice\n");

scanf("%d", &choice);

switch (choice)

{

case 1:

printf("Enter the String\n");

//check for punctuation and spaces

while((s = getchar()) != '\n'){

if(isspace(s)==FALSE || ispunct(s)==FALSE);

string[i] = tolower(s);

i++;

}

scanf("%s", s);

for (i = 0;s[i] != '\0';i++)

{

b = s[i];

push(b);

}

for (i = 0;i < (strlen(s) / 2);i++)

{

if (stack[top] == stack[front])

{

pop();

front++;

}

else

{

printf("%s is not a palindrome\n", s);

break;

}

}

if ((strlen(s) / 2) == front)

printf("%s is palindrome\n", s);

front = 0;

top = -1;

break;

case 2:

exit(0);

default:

printf("enter correct choice\n");

}

}

}

/\* to push a character into stack \*/

void push(char a)

{

top++;

stack[top] = a;

}

/\* to delete an element in stack \*/

void pop()

{

top--;

}

**Output:**

