Lab Assignment 11
Pointers, Structures and Stack

COL 100

29 May 2021

1 Cyclic Swap

Write a function `void cyclicSwap(int *a, int *b, int *c)` that takes three integer pointers a, b and c, and swap them in right cyclic order.
Takes the 3 space separated integer input in the main() function. Then after calling `cyclicSwap` function by passing the address of each of 3 integers as argument, print the swapped value in the main() function.

Input format:

- First line contains an integer 3 space integers

Output format:

- Print space separated swapped values after calling `cyclicSwap` function

Example

Input:

```
1 2 3
```

Output:

```
3 1 2
```

2 Implement strcat

The `strcat()` function is used for string concatenation. It concatenates the specified string at the end of the another specified string. Write a function `char *strcat(char *dest, const char *src)` that appends the string pointed to by `src` to the end of the string pointed to by `dest`. This function returns a pointer to the resulting string `dest`.

Input format:

- First line contains first input string `dest`
- Second line contains second input string `src`

Output format:

- Print the concatenated string returned by `strcat()` function.

Example

Input:

```
abcde
fg
```

Output:

```
abcdefg
```

3 Longest common prefix

Longest common prefix for a pair of strings S1 and S2 is the longest string S which is the prefix of both S1 and S2. Implement the function `char* lcp(char **strs, int n, char *dest)` to find the longest common prefix string amongst an array of strings. Here `strs` points to the start of the array of input strings and `n` is the total no. of strings, `dest` is a pointer to store the result. Save the result of longest common prefix in string pointed by `dest`. This function returns a pointer to resulting string `dest`.
Assume length of input string to be in range [1, 10].

Input format:

- First line contains an integer N denoting the number of strings
- Each of next N lines contains one input strings

Output format:

- Print longest common prefix of all of N strings. If there is no common prefixes then print "No common prefix"

Example

Input:

```
3
str1
str2
str3
```

Output:

```
str
```

Input:

```
3
dog
dogecoin
cat
```

Output:

```
No common prefix
```

4 Pointer Arithmetic

Consider the following C statement declaring a 2 dimensional array.

```
int arr[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};
```

Assume the start address of the array `arr` in memory is 0x64(hex). Array is stored as row major order(elements of an array are being stored in row-wise fashion) in memory.
Assume integer takes 4 bytes of memory
What will be the output of the following printf?

1. `printf("%p", arr);`
2. `printf("%p", arr[0]);`
3. `printf("%d", arr[0][0]);`
4. `printf("%p", arr + 1);`
5. `printf("%p", arr[0] + 1);`
6. `printf("%ld", arr[1] - arr[0]);`
7. `printf("%d", *(*(arr + **arr + 1) + 1));`

5 Complex Numbers

We know that all complex numbers are of the form A + i B, where A is known as Real part of complex number and B is known as Imaginary part of complex number. Implement addition, subtraction, multiplication, and division (you may need conjugate operation for division) of complex numbers, using structures in C.
Use the following definition of structure

```
struct complex {
    float real;
    float imag;
}
```

You have to implement following four functions

```
complex add(complex n1, complex n2)
complex sub(complex n1, complex n2)
complex mul(complex n1, complex n2)
complex div(complex n1, complex n2)
```

Input format:

- First line contains two space separated float indicating real and imaginary part of first number
- Second line contains a single character denoting operation. Possible operations are '+', '-', '*' and '/'
- Third line contains two space separated float indicating real and imaginary part of second number

Output format:

- Print two space separated float indicating the real and imaginary part of the number obtained after applying operation on input numbers

Example

Input:

```
3 4
+
6 7
```

Output:

```
9 11
```

6 Implement Stack

A stack is a linear data structure that follows the Last in, First out principle (i.e. the last added elements are removed first).
Mainly, following are the four basic operations on the Stack

1. Push: Add an element to the top of a stack
2. Pop: Remove an element from the top of a stack
3. IsEmpty: Check if the stack is empty
4. Peek: Get the value of the top element without removing it

Implement the stack data structures. The element of stack is of type integer.
Use the following definition for stack

```
struct stack {
    int size;          // size of the stack, empty stack should have size = 0
    int items[100];     // holds the contents of the stack
}
```

You have to implement following four functions

```
void push(struct stack *pt, int x)         // add an element ‘x‘ to the stack
int pop(struct stack *pt)                 // pop the top element from the stack
int peek(struct stack *pt)                 // get the top element of the stack
int isEmpty(struct stack *pt)             // check if the stack is empty or not
```

Input format:

- First line contains number N denoting the no. of operations
- Next N line contains one operation each. Operations can be “push val”, “pop”, “peek” and “isEmpty”. If the operation is “push” then the value to be pushed is supplied after space.

Output format:

- For each “pop” operation, print the popped value in a new line.
- For each “peek” operation, print the peeked value in a new line.
- For each “isEmpty” operation, print “Empty” if the stack is empty. Otherwise print “Not Empty”
- For “push” operation, don’t print anything

Example

Input:

```
5
push 1
push 2
pop
peek
isEmpty
```

Output:

```
2
1
Not Empty
```

7 Balanced Parentheses I

Given a string s containing just the characters '(' and ')', determine if the input string is balanced. Print "Balanced" if the string is balanced otherwise print "Not Balanced"
You can use the stack data structure developed in the above question.
An input string is balanced if:

- Open brackets must be closed by the same type of brackets.
- Open brackets must be closed in the correct order.

Input format:

- First line contains an integer N denoting the number of inputs
- Next N line contains the input string

Output format:

- For each input string print "Balanced" or "Not Balanced" in a new line

Example

Input:

```
3
()()
()(
(((())())
```

Output:

```
Balanced
Not Balanced
Balanced
```

8 Balanced Parentheses II

Given a string s containing just the characters '(', ')', '[', ']' '{' and '}' , determine if the input string is balanced.
Print "Balanced" if the string is balanced otherwise print "Not Balanced"
You can use the stack data structure developed in the above question.
An input string is balanced if:

- Open brackets must be closed by the same type of brackets.
- Open brackets must be closed in the correct order.

Additional Constraints:

- Curved `( )` brackets can contain only `( )` brackets
- Square brackets `[ ]` can contain only `[ ]` and `( )` brackets
- Curly `{ }` brackets can contain `{ }`, `[ ]` and `( )` brackets

Input format:

- First line contains an integer N denoting the number of inputs
- Next N line contains the input string

Output format:

- For each input string print "Balanced" or "Not Balanced" in a new line

Example

Input:

```
3
{()()}
[(){}]
{[()]{}()}
```

Output:

```
Balanced
Not Balanced
Balanced
```

9 Change sign of float

Change the sign of float type number without using minus(-).
Hint: According to IEEE standard the most significant bit of float represents the sign of the number. How can you toggle that bit of float? Note that you cannot perform bitwise operation directly on float type. You may have to use type casting to be able to perform bitwise operations.

Input format:

- First line contains input float number

Output format:

- Print the output float number after changing sign

Example

Input:

```
23.54
```

Output:

```
-23.54
```

Input:

```
-86.9
```

Output:

```
86.9
```

Challenge Problems

10 Score of Parentheses

Given a balanced parentheses string s, compute the score of the string based on the following rule:

1. () has score 1
2. AB has score A + B, where A and B are balanced parentheses strings.
3. (A) has score 2 * A, where A is a balanced parentheses string.

Note: Input s is a balanced parentheses string, containing only (and)

Input format:

- First line contains an integer N denoting the number of inputs
- Next N line contains the input string

Output format:

- For each input string print its score in new line

Example

Input:

```
4
()
(())
()()
(()(()))
```

Output:

```
1
2
2
4
```

11 Stack Sequence

Given two sequences `pushed` and `popped` with distinct values, print "true" if and only if this could have been the result of a sequence of push and pop operations on an initially empty stack otherwise print "false".

Input format:

- First line contains an integer N denoting the number of elements in the `pushed/popped` sequence
- Second line contains N space separated integers denoting `pushed` sequence
- Third line contains N space separated integers denoting `popped` sequence

Output format:

- Print "true" if given sequence is a valid stack sequence otherwise print "false"

Note:

- `pushed` is a permutation of `popped`
- `pushed` and `popped` have distinct values

Example

Input:

```
5
1 2 3 4 5
4 5 3 2 1
```

Output:

```
true
```

Explanation: We might do the following sequence:
push(1), push(2), push(3), push(4), pop() → 4,
push(5), pop() → 5, pop() → 3, pop() → 2, pop() → 1
Input:

```
5
1 2 3 4 5
4 3 5 1 2
```

Output:

```
false
```

Explanation: 1 cannot be popped before 2

Submission and other logistics

This assignment will not be graded. You don't have to submit any questions on the Gradescope.
It is recommended to try and solve the problems in this assignment.

It is highly **recommended** that you name the code files and variables in those code files with proper names as per the question to easily identify them. Comments in your codes are also highly **recommended** and makes life easier for everyone.
You can check **2nd Chapter** in NASA's C style guide for styling recommendations.