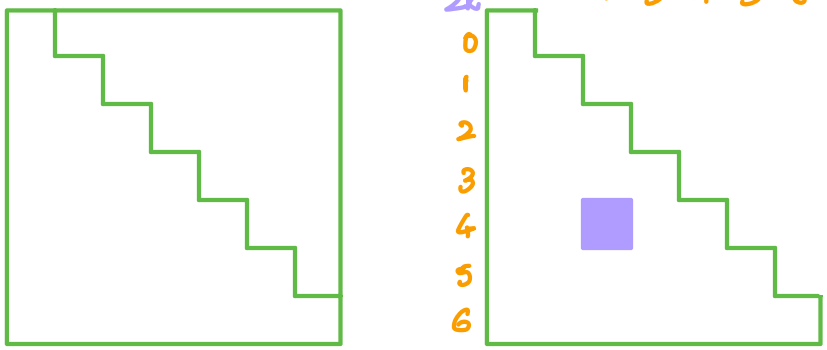
Ques 1:



address of A[i][j] = base address + $\left(\frac{i(i+1)}{2} + j\right) * 4$

$(i \leq j)$

$A[4][2] = 2000 + \left(\frac{4 \times 5}{2} + 2\right) * 4$

$= 2000 + (10+2) * 4$

$= 2000 + 48$

$= 2048$

Ques 2:

```
int length (Node *head)
{
    int count = 0;
    Node *temp = head;
    while (temp != NULL)
    {
        count++;
        temp = temp -> next;
    }
    return count;
}
```

```
void concatenate (Node *head1, Node *head2)
{
    int length1 = length(head1);
    int length2 = length(head2);

    Node *Shead, *lhead;

    if (length1 < length2)
    {
        Shead = head1;
        lhead = head2;
    }
    else
    {
        Shead = head2;
        lhead = head1;
    }

    Node *temp = Shead;
    while (temp->next != NULL)
        temp = temp->next;

    temp->next = lhead;

    temp = lhead;
    while (temp->next != NULL)
        temp = temp->next;

    temp->next = Shead;

    temp = Shead;
    while (temp->next != Shead)
    {
        cout << temp->data;
        temp = temp->next;
```

find length of both the linked list

Keep Shead at the head of small linked list and lhead points to head of large linked list.

Move temp to the last node of smaller linked list

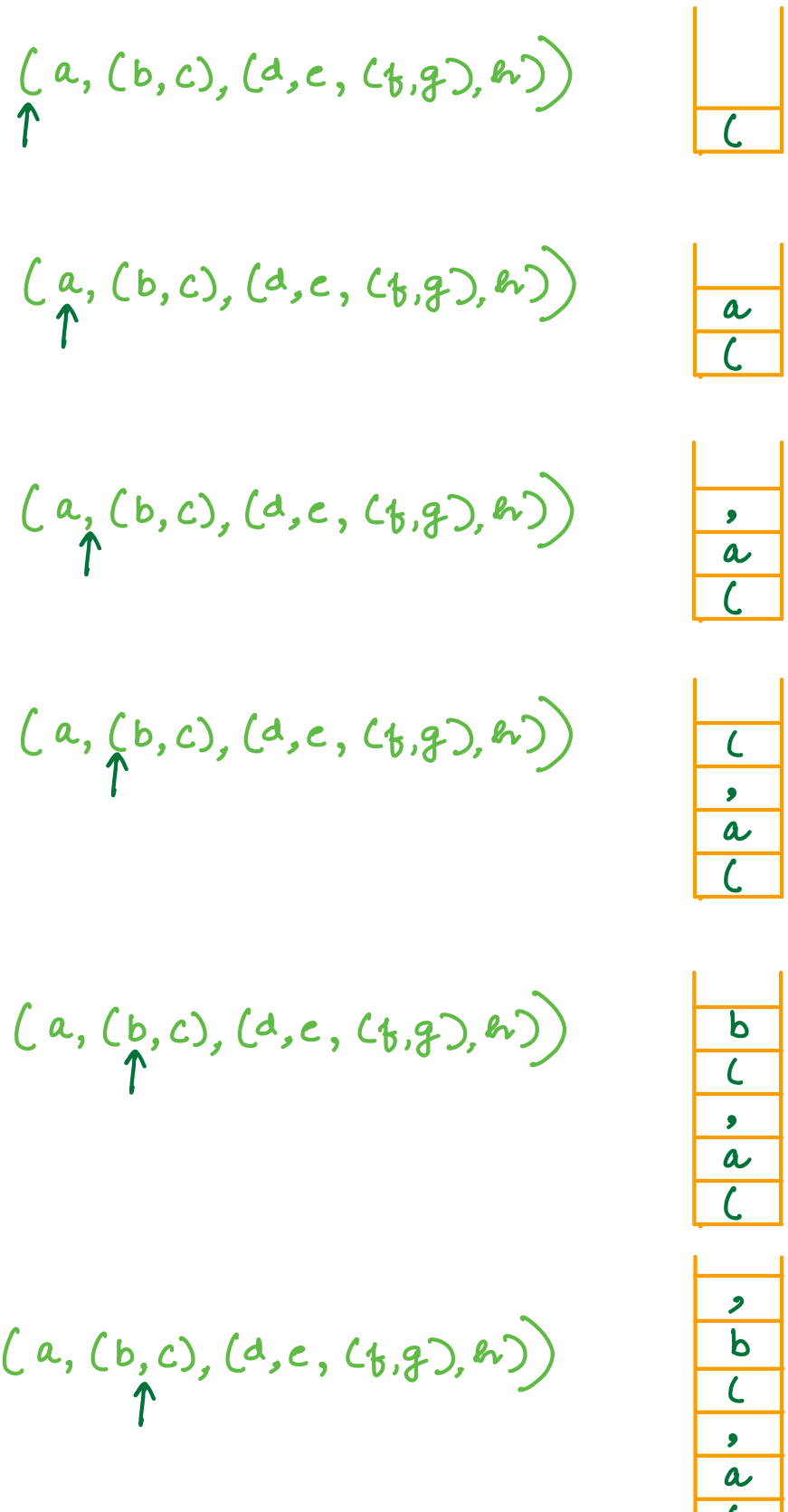
join tail of small linked list to head of large linked list.

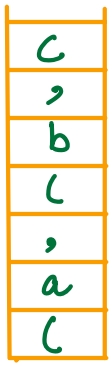
In last node of large linked list set next as head of small linked list. (To make it circular)

printing the circular linked list

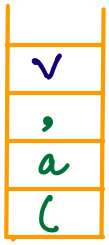
```
    }
    cout << temp -> data;
}
```

Ques 3:





(a, (b, c), (d, e, (f, g), h))

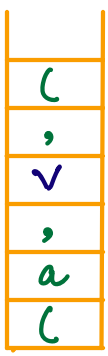


on closing bracket keep on poping the elements till you get opening bracket.

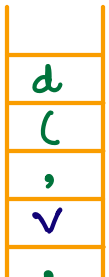
Check if c, b is valid or not; if yes push V in stack.

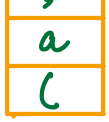
(a, (b, c), (d, e, (f, g), h))

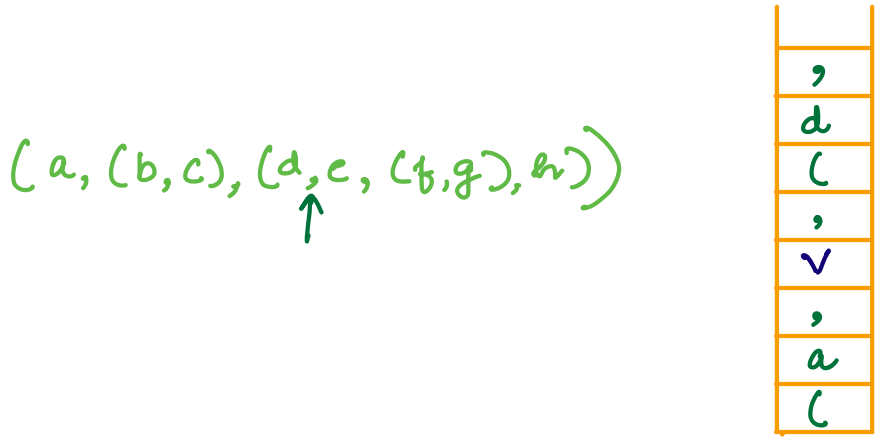
(a, (b, c), (d, e, (f, g), h))

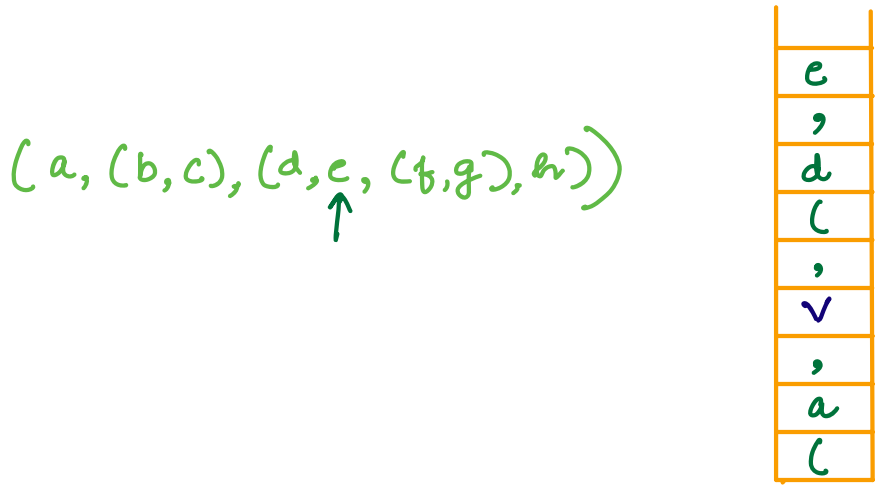


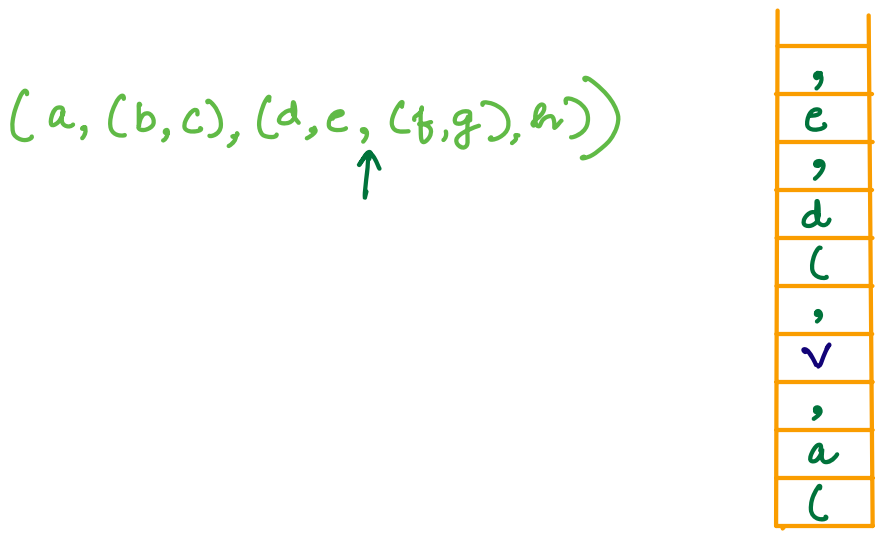
(a, (b, c), (d, e, (f, g), h))

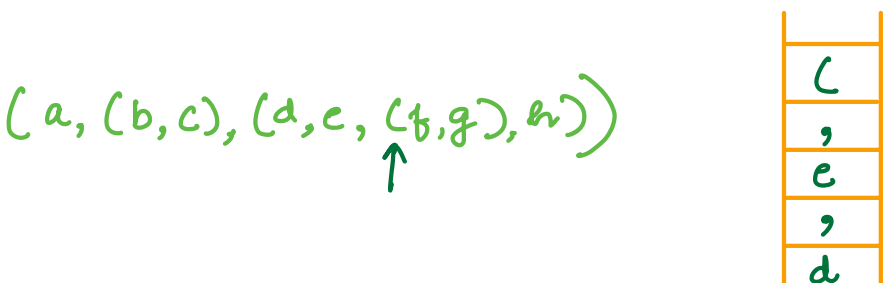


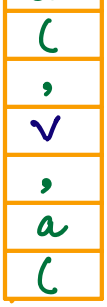

a
(


(a, (b, c), (d, e, (f, g), h))
,
d
(
,
v
,
a
(

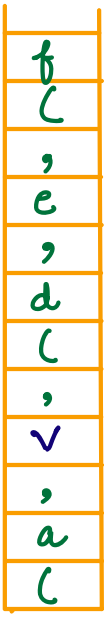

(a, (b, c), (d, e, (f, g), h))
e
,
d
(
,
v
,
a
(


(a, (b, c), (d, e, (f, g), h))
,
e
,
d
(
,
v
,
a
(

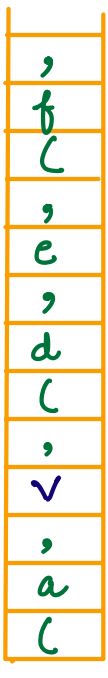

(a, (b, c), (d, e, (f, g), h))
(
,
e
,
d



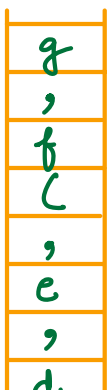
(a, (b, c), (d, e, (f, g), h))

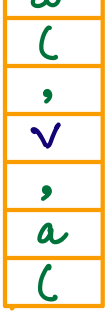


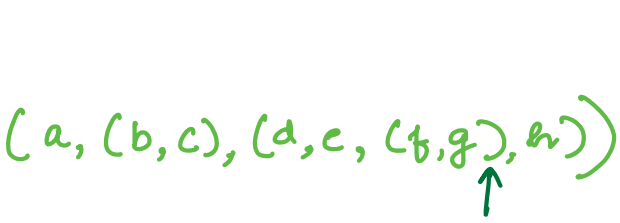
(a, (b, c), (d, e, (f, g), h))

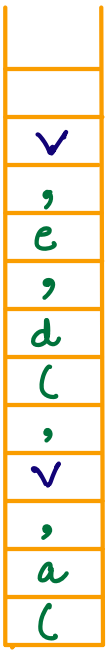


(a, (b, c), (d, e, (f, g), h))



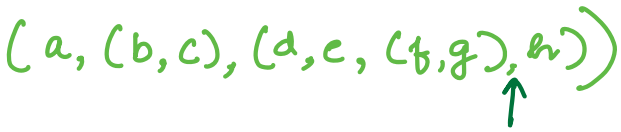


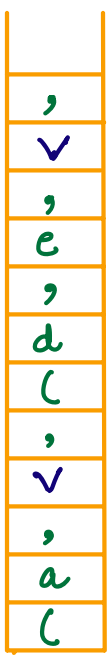


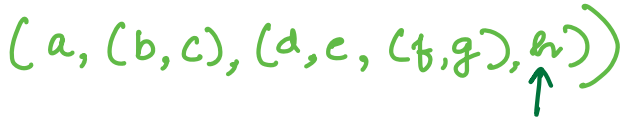


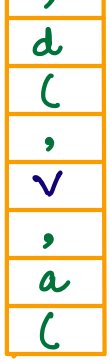
Pop the elements till you get opening bracket.

Check if g, f is valid. Push V

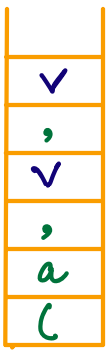








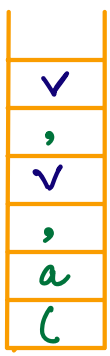
(a, (b, c), (d, e, (f, g), h))



Check if
b, V, c, d is valid?

Yes, push V

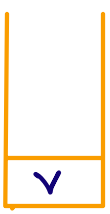
(a, (b, c), (d, e, (f, g), h))



Check if
V, V, a is valid?

Yes, push V

(a, (b, c), (d, e, (f, g), h))



At end we have
V in the stack
It denotes a
recursive list.

Logic:
- Push all the characters except ')' bracket.
- on ')' bracket, pop the elements till you get '(' bracket. Check whether the elements poped form a valid String or not. Eg: a, b is valid
  , b is invalid
  a, b, c, is invalid

If string is valid, push 'V'. If not valid then it is not a recursive list.

```cpp
#include<iostream>
#include<stack>

using namespace std ;

// "b,a" is valid
// ",a" is invalid
bool is_valid(string s)
{
    for(int i = 0 ; i < s.length() ; i++)
    {
        char ch = s[i] ;

        if(i%2 == 0) // at even index there should be alphabets
        {
            if(!isalpha(ch)) return false ;
        }
        else // at odd index there should be ,
        {
            if(ch != ',') return false ;
        }
    }
    return true ;
}

bool recursive_list(string s)
{
    stack<char> st ;

    for(int i = 0 ; i < s.length() ; i++)
    {
        char ch = s[i] ;

        if(ch == ')')
        {
            string temp = "" ;
            while(!st.empty() && st.top() != '(')
            {
                temp += st.top() ;
                st.pop() ;
            }

            // if stack becomes empty while searching for opening bracket it
            // means more no. of closing brackets then opening brackets
            if(st.empty())
                return false ;

            st.pop() ; // pop the opening bracket

            // if temp string is valid i.e. comma and alphabets are at
            // proper place then push V (V -> Valid)
            if(is_valid(temp))
                st.push('V') ;
            // if temp is not valid then it is not a recursive list
            else
                return false ;
        }
        else
            st.push(ch) ;
    }

    char last_char = st.top () ;
    st.pop() ;

    // if stack doesnot become empty then it shows there are
    // more opening brackets as compared to closing brackets
    return (last_char == 'V' && st.empty()) ;
}
```

```
int main()
{
    string s = "(a,(b,c),(d,e,(f,g),h))" ;
    cout << recursive_list(s) << endl ;

    return 0 ;
}
```

Ques 4:

```
#include<iostream>
#include<stack>

using namespace std ;

bool is_valid(string str)
{
    stack<int> st ;

    for(int i = 0 ; i < str.length() ; i++)
    {
        char ch = str[i] ;

        if(ch >= '0' && ch <= '9')
            st.push(ch - '0') ;
        else
        {
            if(st.empty())
                return false ;

            int e1 = st.top() ;
            st.pop() ;

            if(st.empty())
                return false ;

            int e2 = st.top() ;
            st.pop() ;

            if(ch == '+')
                st.push(e2 + e1) ;
            else if(ch == '-')
                st.push(e2 - e1) ;
            else if(ch == '*')
                st.push(e2 * e1) ;
            else if(ch == '/')
                st.push(e2 / e1) ;
        }
    }

    if(st.empty())
        return false ;

    st.pop() ;

    return st.empty() ;
}
```

```
int main()
{
    string str = "231*+9-" ;

    cout << is_valid(str) << endl ;

    return 0 ;
}
```