Chương 2_2:

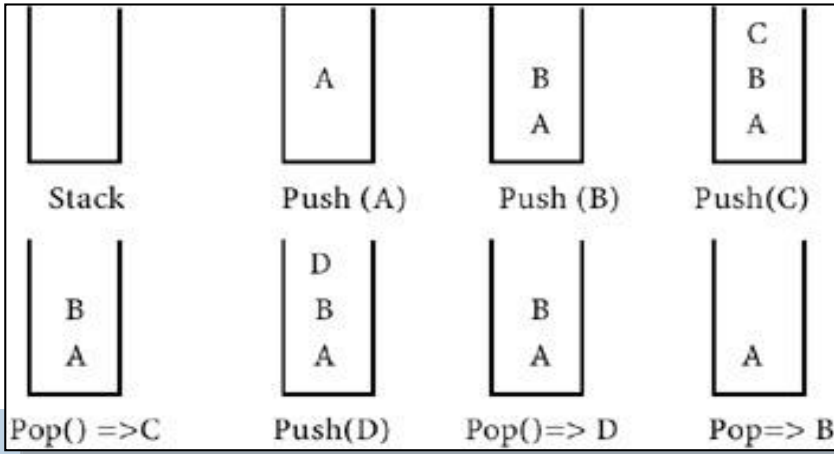
NGĂN XẾP – STACK

Nội dung

- Ngăn xếp (Stack)
  - Khái niệm Stack
  - Các thao tác trên Stack
  - Hiện thực Stack
  - Ứng dụng của Stack

Stack - Khái niệm

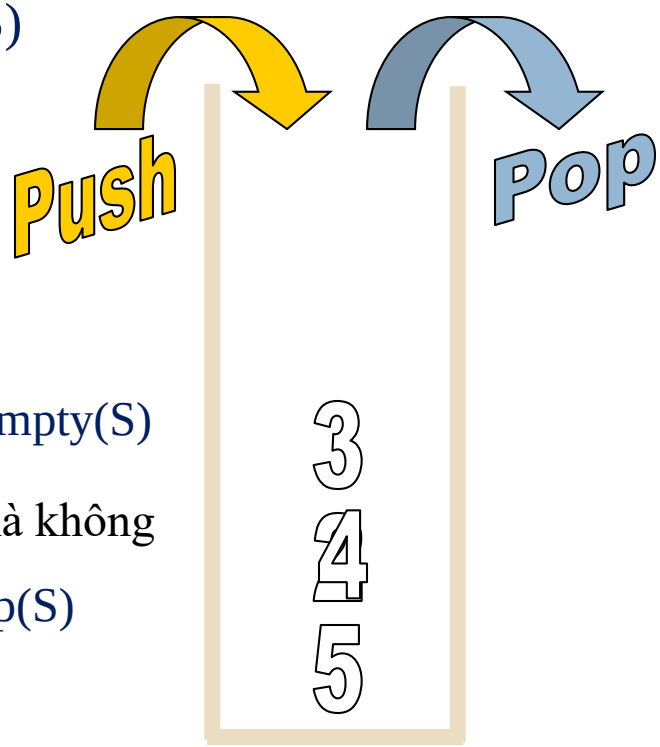
- Stack là một danh sách mà các đối tượng được **thêm vào** và **lấy ra** chỉ ở một đầu của danh sách (A stack is simply a list of elements with insertions and deletions permitted at one end)
- Các phần tử được đưa vào và lấy ra trong ngăn xếp theo nguyên tắc "vào sau ra trước" và cấu trúc dữ liệu này còn được gọi là cấu trúc LIFO (Last In – First Out - *Vào sau ra trước*)
- Two operations are defined on a stack:
  - Push - Inserting information into the stack
  - Pop - retrieving information from the stack



Stack – Các thao tác

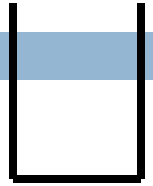
- Stack hỗ trợ 2 thao tác chính:
  - **S.Push(x)**:**Thêm** 1 đối tượng vào Stack Push(x,S)
  - **S.Pop()**: **Lấy** 1 đối tượng ra khỏi Stack Pop(S)
- Ví dụ:

  5 2 3 - - 4
- Stack cũng hỗ trợ một số thao tác khác:
  - **S.is_Empty**(S): Kiểm tra xem Stack có rỗng không Empty(S)
  - **S.Top**(): Trả về giá trị của phần tử nằm ở đầu Stack mà không hủy nó khỏi Stack. Nếu Stack rỗng thì lỗi sẽ xảy ra Top(S)
  - Len(S): Trả về số phần tử trong stack
  - S.Create_Stack(): tạo ngăn xếp rỗng CreateStack(S)

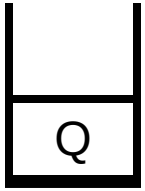


Ví dụ về stack

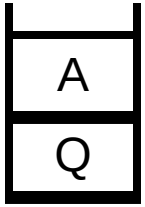
- Stack rỗng:



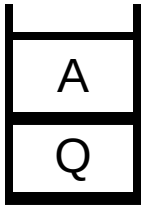
- Đẩy (push) Q vào:



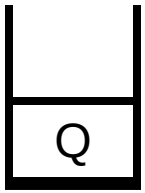
- Đẩy A vào:



- Lấy (pop) ra một => được A:



- Lấy ra một => được Q và stack rỗng:



Stack – Hiện thực Stack
(Implementation of a Stack)

Mảng 1 chiều

Kích thước stack khi quá thiếu, lúc quá thừa

Push / Pop khá phức tạp

Danh sách LK

Cấp phát động!

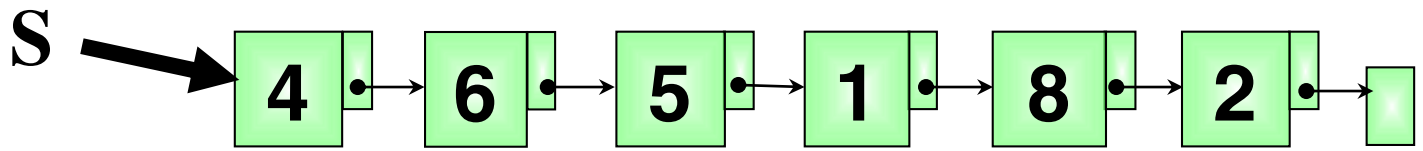
Push/Pop khá dễ dàng

Cài đặt Stack

- Dùng mảng 1 chiều


```
Data    S [N];
int     t;
```

- Dùng danh sách liên kết đơn



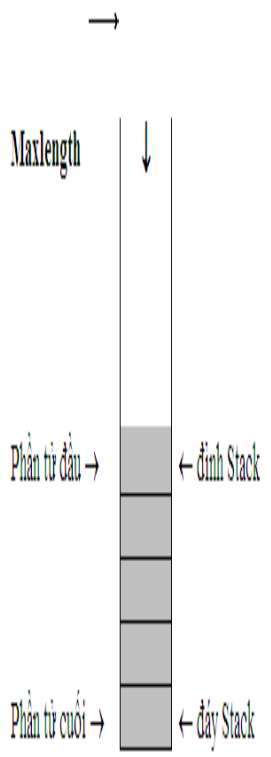
List S

- Thêm và hủy cùng phía

Hiện thực Stack dùng mảng
(Implementation of a Stack using Array)

- Có thể tạo một Stack bằng cách khai báo một mảng 1 chiều với kích thước tối đa là N (ví dụ: N =1000)
  - Stack có thể chứa tối đa N phần tử đánh số từ 0 đến N-1
- Phần tử nằm ở đỉnh Stack sẽ có chỉ số là top
- Để khai báo một Stack, ta cần một mảng 1 chiều,
  và 1 biến số nguyên top cho biết chỉ số của đỉnh Stack:

```
struct Stack {
   DataType  list[N];
   int   top;
};
```



Hiện thực Stack dùng mảng (tt.)

(Implementation of a Stack using Array)

- Các hàm cần cài đặt:
  - Init(Stack &s): Khởi tạo Stack
  - isEmpty(Stack s)
  - Push(Stack &s , DataType x)
  - Pop(Stack &s)
  - Top(Stack &s)
  - Khi cài đặt bằng mảng 1 chiều, Stack bị giới hạn kích thước nên cần xây dựng thêm một thao tác phụ cho Stack:
    - **isFull**(): Kiểm tra xem Stack có đầy chưa, vì khi Stack đầy, việc gọi đến hàm **Push()** sẽ bị lỗi

Hiện thực Stack dùng mảng (tt.)

(Implementation of a Stack using Array)

- **Khởi tạo Stack:**

```
5   class ArrayStack:
6       '''LIPO stack
7       '''
8       def __init__(self):
9           #Tạo stack rỗng
10          self.data =[]
```

```
void Init (Stack &s)
{
    s.top = 0;
}
```

Hiện thực Stack dùng mảng (tt.)

(Implementation of a Stack using Array)

- **Kiểm tra Stack có rỗng hay không:**

```
14      def is_empty(self):
15          #return True if the stack is empty
16          return len(self.data) == 0
```

```
int isEmpty(Stack s)
{
    if ( s.top==0 )
            return 1; // stack rỗng
    else
            return 0;
}
```

Hiện thực Stack dùng mảng (tt.)

(Implementation of a Stack using Array)

- **Thêm một phần tử x vào Stack**

```
17      def push(self,e):
18          #Add element a to the top of the stack
19          self.data.append(e)
```

```
void Push (Stack &s, DataType x)
{   if (!isFull(s)) // stack chưa đầy
    {    s.list[s.top] = x;
         s.top++;
    }
}
```

Hiện thực Stack dùng mảng (tt.)

(Implementation of a Stack using Array)

- **Lấy một phần tử ra khỏi Stack**

```
26      def pop(self):
27          #Remove and return the element from the top of the stack
28          #Raise Empty excepty if the stack is empty
29          if self.is_empty():
30              raise Empty('Stack is EMpty')
31          return self.data.pop()
```

```
DataType Pop(Stack &s)
{    DataType x;
     if (!isEmpty(s)) // stack không rỗng
     {    s.top--;
          x = s.list[s.top];
     }
     return x;
}
```

Hiện thực Stack dùng mảng (tt.)

(Implementation of a Stack using Array)

- **Xem phần tử ở đỉnh Stack**

```
20      def top(self):
21          #Return the element at the top of the stack
22          #Raise Empty Exception if the stack is empty
23          if self.is_empty():
24              raise Empty('Stack is Empty')
25          return self.data[-1] #the last in the list
```

```
DataType Top(Stack s)
{
    DataType x;
    if (!isEmpty(s)) // stack không rỗng
    {    x = s.list[s.top-1];}
    return x;
}
```

Hiện thực Stack dùng mảng (tt.)

(Implementation of a Stack using Array)

- **Đổi về kiểu chuỗi:**

```
32      def __str__(self):
33          #A string representation of the stack
34          # An arrow show the top of the stack
35          return ''.join(str(self.data))+'->'
36      ############################
```

```
11      def __len__(self):
12          #return the number of elements
13          return len(self.data)
14      def is_empty(self):
```

Hiện thực Stack dùng mảng (tt.)

(Implementation of a Stack using Array)

- **Thực thi chương trình**

```
    print('==============Demo Stack===============')
if __name__ =='__main__':
    S = ArrayStack()
    S.push(5)
    S.push(3)
    print('Stack Length: ',len(S))
    print('S: ',S)
    print('Pop ',S.pop())
    print('Is stack Empty? ',S.is_empty())
    print('Pop ',S.pop())
    print('Is stack Empty? ', S.is_empty())
    print('S:',S)
    S.push(7)
    S.push(9)
    print('Top Element in Stack : ', S.top())
    S.push(4)
    S.push(6)
    print('S: ',S)
```

Hiện thực Stack dùng mảng (tt.)

(Implementation of a Stack using Array)

- **Kiểm tra Stack có đầy hay không:**
  - **Đầy: hàm trả về 1**
  - **Ngược lại: hàm trả về 0**

```
int isFull(Stack s)
{
    if (s.top>=N)
        return 1;
    else
        return 0;
}
```

Hiện thực Stack dùng mảng (tt.)

(Implementation of a Stack using Array)

Nhận xét:

- Các thao tác trên đều làm việc với chi phí O(1)
- Việc cài đặt Stack thông qua mảng một chiều đơn giản và khá hiệu quả
- Tuy nhiên, hạn chế lớn nhất của phương án cài đặt này là giới hạn về kích thước của Stack (N). Dung lượng sử dụng cho 1 Stack là O(n)
  - Giá trị của N có thể quá nhỏ so với nhu cầu thực tế hoặc quá lớn sẽ làm lãng phí bộ nhớ

Operation	Running Time
S.push(e)	$O(1)^*$
S.pop( )	$O(1)^*$
S.top( )	$O(1)$
S.is_empty( )	$O(1)$
len(S)	$O(1)$

*amortized

Performance of our array-based stack implementation.

Hiện thực Stack dùng mảng (tt.)

(Implementation of a Stack using Array)

- Ứng dụng: đảo ngược dữ liệu

Ví dụ:

- In các dòng trong một tập tin theo thứ tự ngược lại.
- Đảo ngược các phần tử của danh sách bằng cách sử dụng stack.
- Đảo ngược thứ tự các phần tử được lưu trữ trong một ngăn xếp.

```
# reversing data using a stack
def reverse_file(filename):
  ''' Overwrite given file with its conent line-by-line reversed'''

  S = ArrayStack()
  original = open(filename)
  for line in original:
    S.push(line.rstrip('\n')) # we will re-insert newlines when writing
  original.close()

  # Now we overwrite with contents in LIFO order
  output = open(filename, 'w') # reopening file overwrites original
  while not S.is_empty():
    output.write(S.pop() + '\n') # re-insert newline characters
  output.close()

##########
file = open("initial.txt", 'w')
file.write("I am going home.\n")
file.write("Today is a holiday.")
file.close()

!cat initial.txt
print('\n\n')
reverse_file("initial.txt")
!cat initial.txt
```

```
I am going home.
Today is a holiday.


Today is a holiday.
I am going home.
```

Hiện thực Stack dùng mảng (tt.)

(Implementation of a Stack using Array)

We perform a left-to-right scan of the original sequence, using a stack S to facilitate the matching of grouping symbols.

```
51  def is_matched(expr):
52      '''Return True if all delimiters are properly matched; fakse otherwise'''
53      lefty = '({[' #opening delimiters
54      righty =')}]' #respective closing delimiters
55      S= ArrayStack()
56      for c in expr:
57          if c in lefty:#push left delimiter on stack
58              S.push(c)
59          elif c in righty:
60              if S.is_empty():
61                  return False # Nothing to match
62              if righty.index(c) != lefty.index(S.pop()):
63                  return False   #mismatch
64      return S.is_empty() #were all symbols matched
```

```
18 ##########
19
```

```
80      expr1 = '[(5+x)-(y+z)]}'
81      print(is_matched(expr1))
82      expr2 ='[(5+x)-(y+z)]'
83      print(is_matched(expr2))
```

```
False
True
```

Hiện thực Stack dùng mảng (tt.)

(Implementation of a Stack using Array)

- Adjoining code demonstrates the use of stacks in checking for matching tags in a HTML document.

```
def is_matched_html(raw):
  ''' return True if all HTML tags are properly match; False otherwise'''
  S = ArrayStack()
  j = raw.find('<')                    # find first '<' character (if any)
  while j != -1:
    k = raw.find('>', j+1)             # find next '>' character
    if k == -1:
      print('Invalid Tag')
      return False                     # invalid tag
    tag = raw[j+1:k]                   # strip away < >
    if not tag.startswith('/'):        # this is opening tag
      S.push(tag)
    else:                              # this is closing tag
      if S.is_empty():
        print('Stack is empty. Nothing to match with')
        return False                   # nothing to match with
      if tag[1:] != S.pop():
        print('Tag Mismatch:', tag)
        return False                   # mismatched delimiter
    j = raw.find('<', k+1)             # find next '<' character (if any)
  return S.is_empty()

#################

is_matched_html('''<body>
<center>
<h1> The Little Boat </h1>
</center>
<p> The storm tossed the little boat like a cheap sneaker in an
old washing machine. The three drunken fishermen were used to
such treatment, of course, but not the tree salesman, who even as
a stowaway now felt that he had overpaid for the voyage. </p>
<ol>
<li> Will the salesman die? </li>
<li> What color is the boat? </li>
<li> And what about Naomi? </li>
</ol>
</body>''')
```

```
True
```

Hiện thực Stack dùng DSLK
(Implementation of a Stack using Linked List)

- Các phương thức cần cài đặt:
  - __init__(self): Khởi tạo ngăn xếp (Stack) với 1 danh sách
  - Is_Empty(self): Kiểm tra ngăn xếp rỗng
  - __str__(self): Chuyển ngăn xếp về kiểu chuỗi
  - Push(self , Gia_tri): Thực hiện đưa Gia_tri vào ngăn xếp bằng cách thêm vào đầu danh sách.
  - Pop(self): Thực hiện lấy ra 1 giá trị từ ngăn xếp

Hiện thực Stack dùng DSLK
(Implementation of a Stack using Linked List)

- Các phương thức cần cài đặt:
  - __init__(self): Khởi tạo ngăn xếp (Stack) với 1 danh sách

```
1  #Tạo lớp Stack
2  class NganXep:
3      #Khởi tạo Ngăn xếp Stack
4      def __init__(self):
5          self.danh_sach = []
6      #def
7      #Kiểm tra ngăn xếp có rỗng không
```

Hiện thực Stack dùng DSLK
(Implementation of a Stack using Linked List)

- Is_Empty(self): Kiểm tra ngăn xếp rỗng

```
7       #Kiểm tra ngăn xếp có rỗng không
8       def is_Empty(self):
9           return len(self.danh_sach) == 0
10      #def
```

Hiện thực Stack dùng DSLK
(Implementation of a Stack using Linked List)

- __str__(self): Chuyển ngăn xếp về kiểu chuỗi

```
11      #Đổi về chuỗi
12      def __str__(self):
13          kq = 'Ngăn xếp ['
14          stt = 0
15          for x in self.danh_sach:
16              stt = stt +1
17              if stt ==1:
18                  kq = kq + str(x)
19              else:
20                  kq = kq +' ->' +str(x)
21              #if
22          #for
23          kq = kq +']'
24          return kq
25      #def
```

Hiện thực Stack dùng DSLK
(Implementation of a Stack using Linked List)

- Push(self , Gia_tri): Thực hiện đưa Gia_tri vào ngăn xếp

```
26      #Đẩy vào stack
27      def Push(self,gia_tri):
28          #đẩy vào
29          self.danh_sach.insert(0,gia_tri) #Thêm vào đầu
30      #def
31      #Lấy ra khỏi stack
```

Hiện thực Stack dùng DSLK
(Implementation of a Stack using Linked List)

- Pop(self): Thực hiện lấy ra 1 giá trị từ ngăn xếp

```
31      #Lấy ra khỏi stack
32      def Pop(self):
33          #Kiểm tra rỗng
34          if self.is_Empty():
35              return None
36          else:
37              return self.danh_sach.pop(0)
38      #def
39  #Class
```

Hiện thực Stack dùng DSLK
(Implementation of a Stack using Linked List)

- Tạo đối tượng ngăn xếp

```
40  #Tạo đối tượng ngăn xếp
41  if __name__ =='__main__':
42      ngan_xep =NganXep()
43      print(ngan_xep)
44      print('-----Đẩy vào-----')
45      for i in range (1,6):
46          print(f'* Đẩy vào {i}')
47          ngan_xep.Push(i)
48          print(ngan_xep)
49      #for
50      print('-----Lấy ra-----')
51      while not ngan_xep.is_Empty():
52          gt =ngan_xep.Pop()
53          print(f'* Lấy ra -> {gt}')
54          print(ngan_xep)
55      #while
```

Hiện thực Stack dùng DSLK
(Implementation of a Stack using Linked List)

- Có thể tạo một Stack bằng cách sử dụng một danh sách liên kết đơn (DSLK)
- Khai báo các cấu trúc:

```
struct Node
{
    DataType data;
    Node *pNext;
};
struct Stack
{
    Node *top;
};
```

Hiện thực Stack dùng DSLK
(Implementation of a Stack using Linked List)

- Các hàm cần cài đặt:
  - Init(Stack &s): Khởi tạo Stack
  - isEmpty(Stack s)
  - Push(Stack &s , DataType x)
  - Pop(Stack &s)
  - Top(Stack &s)

Hiện thực Stack dùng DSLK (tt.)
(Implementation of a Stack using Linked List)

- **Khởi tạo Stack:**

```
void Init( Stack &s )
{
      s.top = NULL;
}
```

Hiện thực Stack dùng DSLK (tt.)
(Implementation of a Stack using Linked List)

- **Kiểm tra xem Stack có rỗng không:**

```
int  isEmpty ( Stack s )
{
    return  s.top == NULL ? 1 : 0;
}
```

Hiện thực Stack dùng DSLK (tt.)

(Implementation of a Stack using Linked List)

- **Thêm một phần tử vào Stack:**

```
void  Push ( Stack &s,  DataType  x )
{
        Node *p = new Node;
        if ( p==NULL )  { cout<<"Khong du bo nho"; return; }
        p->data = x;
        p->pNext = NULL;
        if (s.top==NULL)        // if (isEmpty(s))
                s.top = p;
        else{
                p->pNext = s.top;
                s.top = p;
            }
}
```

Thêm phần tử vào đầu danh sách

Hiện thực Stack dùng DSLK (tt.)

(Implementation of a Stack Using Linked List)

- Lấy một phần tử ra khỏi Stack:

```
DataType Pop ( Stack &s )
{
        if ( s.top==NULL ){
                cout<<"Stack rỗng";  return 0;
        }
        DataType x;
        Node *p = s.top;
        s.top = s.top->pNext;
        x = p->data;
        delete p;
        return x;
}
```

Lấy và xóa phần tử ở đầu danh sách

Hiện thực Stack dùng DSLK (tt.)
(Implementation of a Stack Using Linked List)

- Xem phần tử ở đỉnh Stack:

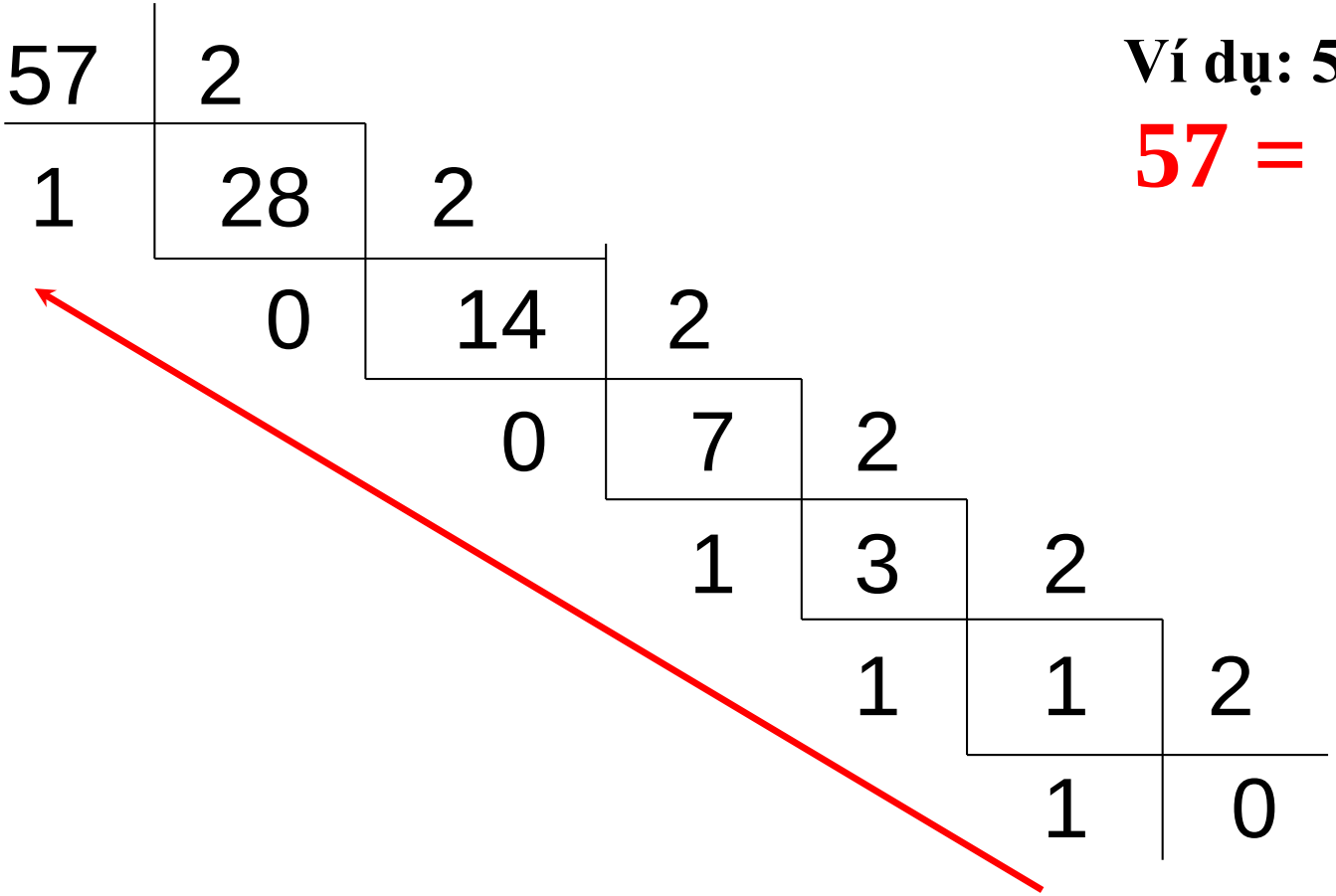
```
DataType Top ( Stack s )
{
        if ( s.top==NULL ){
                cout<<"Stack rỗng";  return 0;
        }
        DataType x;
        x = s.top->data;
        return x;
}
```

Stack - Ứng dụng

- Stack thích hợp lưu trữ các loại dữ liệu mà trình tự truy xuất ngược với trình tự lưu trữ
- Một số ứng dụng của Stack:
  - Trong trình biên dịch (thông dịch), khi thực hiện các thủ tục, Stack được sử dụng để lưu môi trường của các thủ tục
  - Lưu dữ liệu khi giải một số bài toán của lý thuyết đồ thị (như tìm đường đi)
  - Ứng dụng trong các bài toán tính toán biểu thức
  - Khử đệ qui
  - …

Stack - Ứng dụng

Bài tập: đổi số từ cơ số 10 sang cơ số x



Ví dụ: 57 = $???_2$

$57 = 111001_2$

Cài đặt thuật toánđổi số từ cơ số 10 sang cơ số x

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
#define max  50;
struct Stack
   {
      int a[max];
     int Top;
   }  S;
void Push(int n);
int Pop();
void  chuyen_doi(int N );
```

```
int main( )
{
   int n;
   printf(" Nhap so can chuyen doi");
   scanf("%d",&n);
   chuyen_doi(n);
   return 0;
}
void Push(int n)
{
   if(S.Top>=max) printf ("Stack day");
   else
   {
       S.Top++;
       S.a[S.Top]=n;
   }
}
```

```
int Pop( )
{
   if(S.Top == 0) return 0;
  else
     {int n = S.a[S.Top];
        S.Top--;
        return n;
     }
}
void  chuyen_doi( int N)
{
   int R;
   while (N!=0)
   {
       R = N % 2;
       Push(R);
       N = N/2;
   }
   while (S.Top!=0)  printf("%d",Pop());
}
```

```
void main()
{
    Stack s;
    int coso, so, sodu;
    Init(s);
    // Nhập số cần chuyển vào biến so …
    // Nhập cơ số cần chuyển vào biến coso…
    while (so != 0)
    {
        sodu = so % coso;
        Push (s, sodu);  // push so du vao stack
        so = so/coso;
    }
    cout<<"Kết quả: ";
    while ( !isEmpty(s) )
        cout<<Pop(s);  // pop so du ra khoi stack
}
```

Stack – Bài tập

1. Cho một STACK đang chứa các phần tử sau: A, B, C, E, F. Biểu diễn STACK và viết lệnh tương ứng bằng 2 cách cài đặt bằng mảng và cài đặt bằng danh sách liên dựa trên các mô tả sau:
   - Thêm A, B, C vào STACK.
   - Hủy 2 phần tử ra khỏi STACK.
   - In giá trị của phần tử trên đỉnh stack

Stack - Ứng dụng

Ví dụ: thủ tục Quick_Sort dùng Stack để khử đệ qui:

- Bước 1. l=1; r=n;
- Bước 2. Chọn phần tử giữa x=a[(l+r) / 2]
- Bước 3. Phân hoạch (l, r) thành (l1, r1) và (l2, r2) bằng cách xét:
  - y thuộc (l1, r1) nếu y≤x
  - y thuộc (l2, r2) ngược lại
- Bước 4. Nếu phân hoạch (l2, r2) có nhiều hơn 1 phần tử thì thực hiện:
  - Cất (l2, r2) vào Stack
  - Nếu (l1, r1) có nhiều hơn 1 phần tử thì thực hiện:
    - l = l1
    - r = r1
    - Quay lên bước 2
  - Ngược lại
    - Lấy (l, r) ra khỏi Stack, nếu Stack khác rỗng thì quay lên bước 2, ngược lại thì dừng

Stack - Ứng dụng

- Thuật toán Ba Lan ngược
  (Reverse Polish Notation – RPN)
  - Định nghĩa RPN:
    - Biểu thức toán học trong đó các toán tử được viết sau toán hạng và không dùng dấu ngoặc
  - Phát minh bởi Jan Lukasiewics một nhà khoa học Ba Lan vào những năm 1950

Thuật toán Ba Lan ngược - RPN

Postfix (RPN): toán tử viết **sau** toán hạng
Prefix : toán tử viết **trước** toán hạng
Infix : toán tử viết **giữa** toán hạng

Examples:

Infix	RPN (Postfix)	Prefix
A + B	A B +	+ A B
A * B + C	A B * C +	+ * A B C
A * (B + C)	A B C + *	* A + B C
A - (B - (C - D))	A B C D - - -	- A - B - C D
A - B - C - D	A B - C - D -	- - - A B C D

Lượng giá biểu thức RPN

***Kỹ thuật gạch dưới*:**

1. Duyệt từ trái sang phải của biểu thức cho đến khi gặp toán tử.

2. Gạch dưới 2 toán hạng ngay trước toán tử và kết hợp chúng bằng toán tử trên

3. Lặp đi lặp lại cho đến hết biểu thức.

Ví dụ 2*((3+4)-(5-6))

2 3 4 + 5 6 - - *

→ 2 3 4 + 5 6 - - *

→ 2 7 5 6 - - *

→ 2 7 5 6 - - *

→ 2 7 -1 - *

→ 2 7 -1 - * → 2 8 * → 2 8 * → 16

Dùng Stack để tính giá trị RPN

1. Khởi tạo Stack rỗng .
2. Lặp cho đến khi kết thúc biểu thức:

   Đọc 1 phần tử của biểu thức

   Nếu phần tử là *toán hạng* thì đưa vào Stack.

   Ngược lại (là *phép toán*):

   Lấy ra 2 phần tử trong Stack.

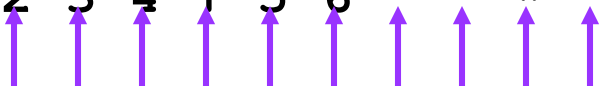
   Áp dụng phép toán cho 02 phần tử vừa lấy ra.

   Đưa kết quả vào Stack.
3. Giá trị của biểu thức chính là phần tử cuối cùng của Stack.

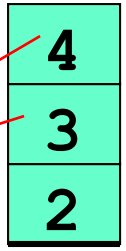
- + , - 1
- *, / 2
- ^ 3

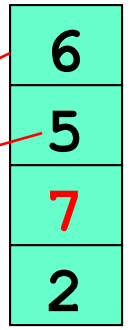
2*((3+4)-(5-6))

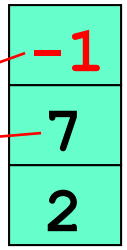
Example: `2 3 4 + 5 6 - - *`

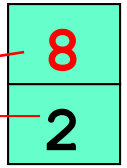


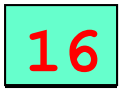
- Push 2
- Push 3
- Push 4
- Read +
  - Pop 4, Pop 3, 3 + 4 = 7
- Push 7
- Push 5
- Push 6
- Read -
  - Pop 6, Pop 5, 5 - 6 = -1
- Push -1
- Read -
  - Pop -1, Pop 7, 7 - -1 = 8
- Push 8
- Read *
  - Pop 8, Pop 2, 2 * 8 = 16
- Push 16











Chuyển Infix thành Postfix

1. Khởi tạo Stack rỗng (chứa các phép toán)
2. Lặp cho đến khi kết thúc biểu thức:

   Đọc 01 phần tử của biểu thức *(01 phần tử có thể là hằng, biến, phép toán, ")" hay "(")*

   Nếu phần tử là:

   2.1 **"("**: đưa vào Stack.

   2.2 **")"**: lấy các phần tử của Stack ra cho đến khi gặp "(" trong Stack.

Chuyển Infix thành Postfix (tt.)

2.3 Một phép toán: + - * /

Nếu Stack rỗng: đưa vào Stack.

Nếu Stack khác rỗng và **phép toán có độ ưu tiên cao hơn phần tử ở đầu Stack**: đưa vào Stack.

Nếu Stack khác rỗng và phép toán có **độ ưu tiên thấp hơn hoặc bằng phần tử ở đầu Stack**:

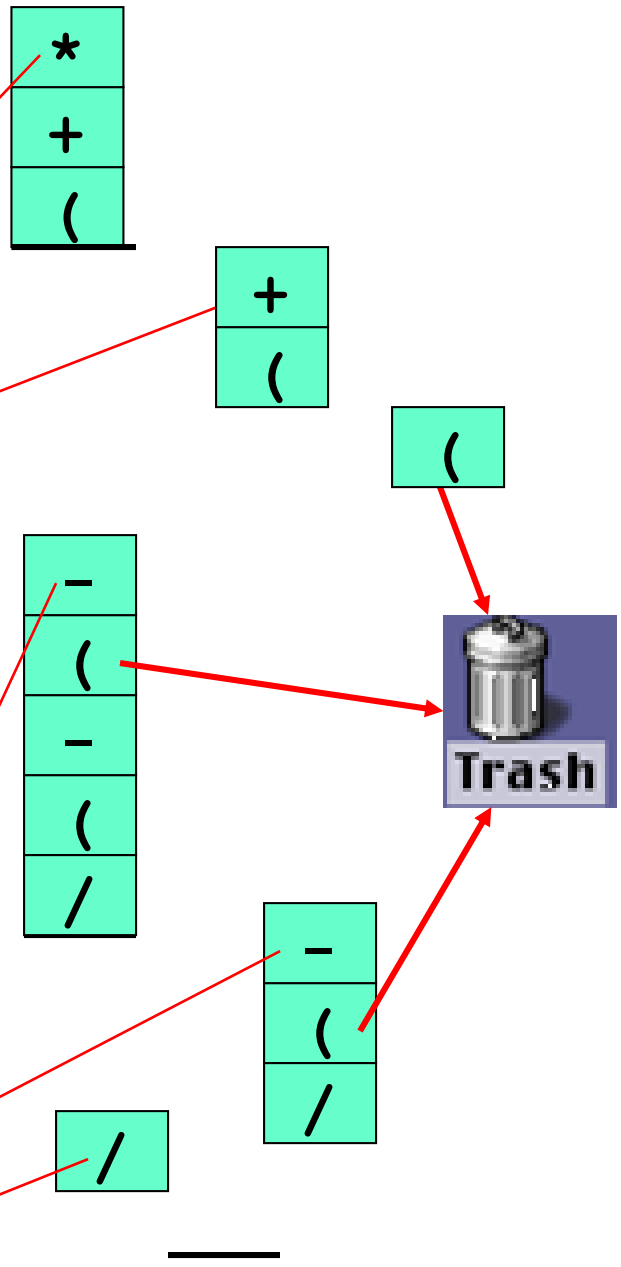
- lấy phần tử từ Stack ra;
- sau đó lặp lại việc so sánh với phần tử ở đầu Stack.

2.4 Hằng hoặc biến: đưa vào kết quả.

3. Lấy hết tất cả các phần tử của Stack ra.

Example: **(A+B*C)/(D-(E-F))**

	Output
Push (	
Display A	A
Push +	
Display B	AB
Push *	
Display C	ABC
Read)	
Pop *, Display *,	ABC*
Pop +, Display +, Pop (	ABC*+
Push /	
Push (	
Display D	ABC*+D
Push -	
Push (	
Display E	ABC*+DE
Push -	
Display F	ABC*+DEF
Read)	
Pop -, Display -, Pop (	ABC*+DEF-
Read)	
Pop -, Display -, Pop (	ABC*+DEF--
Pop /, Display /	ABC*+DEF--/



Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[];

KQ=“”

Ví dụ

A + (B*C - (D/E^F) * G) * H

A + (B*C - (D/E^F) * G) * H

S=[];

KQ="A"

Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[+(;*;];

KQ=ABC

Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[+(

KQ=ABC*

Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[+(-[;

KQ=ABC*

Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[+(-(];

KQ=ABC*D

Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[+(-(/;

KQ=ABC*D

Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[+(-(/];

KQ=ABC*DE

Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[+((-((/^];

KQ=ABC*DE

Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[+(-(/^];

KQ=ABC*DEF

Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[+(-(/^;];

KQ=ABC*DEF^/

Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[+(-*;

KQ=ABC*DEF^/

Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[+(-*];

KQ=ABC*DEF^/G

Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[+[[;]*;];

KQ=ABC*DEF^/G*-

Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[+];

KQ=ABC*DEF^/G*-

Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[+*];

KQ=ABC*DEF^/G*-H

Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[];

KQ=ABC*DEF^/G*-H*+

```
Procedure ConvertInfixToRPN (T: String)
Stack := ∅;
FOR<Phần tử T đọc được từ biểu thức infix> DO
//T có thể là hằng, biến, toán tử hoặc dấu ngoặc được đọc từ biểu thức infix theo thứ tự từ trái qua phải
CASE  T
'(': Push(T);
')':
    REPEAT
            x := Pop;
            if x ≠ '(' then Output(x);
    UNTILL  x = '(';
'+', '-', '*', '/':
    BEGIN
        WHILE(Stack ≠ ∅) and (Priority(T) ≤ Priority(Top) DO
            Output(Pop);
        Push(T);
   END
ELSE  Output(T);
ENDCASE;
WHILE (Stack ≠ ∅) DO Output(Pop);
```

Ví dụ: (2 * 3 + 7 / 8) * (5 - 1)

Đọc	Xử lý	Stack	Output
(	Đẩy vào Stack	(	
2	Hiển thị	(	2
*	Phép "*" được ưu tiên hơn "(" ở đỉnh Stack, đẩy "*" vào Stack	(*	
3	Hiển thị	(*	2 3
+	Phép "+" ưu tiên không cao hơn " * " ở đỉnh Stack. Lấy ra và hiển thị " * ". Phép "+" ưu tiên cao hơn "(" ở đỉnh Stack, đẩy "+" vào Stack	(+	2 3*
7	Hiển thị	(+	2 3*7
/	Phép "/" ưu tiên hơn "+" ở đỉnh Stack là, đẩy "/" vào Stack	(+/	**2 3 * 7**
8	Hiển thị	(+/	**2 3*7 8**

Ví dụ: (2 * 3 + 7 / 8) * (5 - 1)

Đọc	Xử lý	Stack	Output
)	Lấy ra và hiển thị các phần tử trong Stack tới khi lấy phải dấu "("	□	2 3*7 8/+
*	Stack đang là rỗng, đẩy * vào Stack	*	
(	Đẩy vào Stack	*(	
5	Hiển thị	*(	2 3*7 8/+5
-	Phép "-" ưu tiên hơn "(" ở đỉnh Stack, đẩy "-" vào Stack	*(-	
1	Hiển thị	*(-	2 3*7 8/+5 1
)	Lấy ra và hiển thị các phần tử trong Stack tới khi lấy phải dấu "("	*	2 3*7 8/+5 1-
Hết	Lấy ra và hiển thị hết các phần tử còn lại trong Stack		2 3*78/+51-*

```
#include<stdio.h>
#include<iostream.h>
#include<conio.h>
#include<string.h>
#include<ctype.h>
#include<stdlib.h>
typedef char xau[10];
xau z,e[20],e1[20],s[20];
int i,j,n,top;
int uutien(xau z)
{int k=0;
if(strcmp(z,"$")==0) k=1;
if(strcmp(z,"(")==0) k=2;
if(strcmp(z,"+")==0||strcmp(z,"-")==0) k=3;
if(strcmp(z,"*")==0||strcmp(z,"/")==0) k=4;
return k;
}
```

```
void taobtbalan()
{strcpy(s[1],"$"); top=1;j=0;strcpy(z,e[1]); i=1;
while (strcmp(z,"#") !=0)
{
  if (strcmp (z,"(")==0) { top++; strcpy(s[top],z);}
  else {if (strcmp(z,")")==0)
            {while (strcmp(s[top],"(") !=0)
                        {j++; strcpy(e1[j],s[top]);
                        strcpy(s[top],"");top--;
                        }
            strcpy(s[top],""); top--;
            }
       else {if (strcmp(z,"+")==0 || strcmp(z,"-")==0 ||
            strcmp(z,"*")==0 || strcmp(z,"/")==0)
                  {while (uutien(s[top])>= uutien(z))
                  {j++;strcpy(e1[j],s[top]);
                  strcpy(s[top],""); top--; }
                  top++;strcpy(s[top],z);
                  }
              else {j++;strcpy(e1[j],z); }
  }
  }
```

```
while (strcmp(s[top],"$") !=0)
{j++;strcpy(e1[j],s[top]);strcpy(s[top],"");
  top--;}
j++; strcpy(e1[j],"#");
}
float giatri()
{ xau u,v,w; float x,y,r;
j=1; strcpy(z,e1[j]); top=0;
while (strcmp(z,"#") !=0)
{printf("\nj= %d, z= %s",j,z);
```

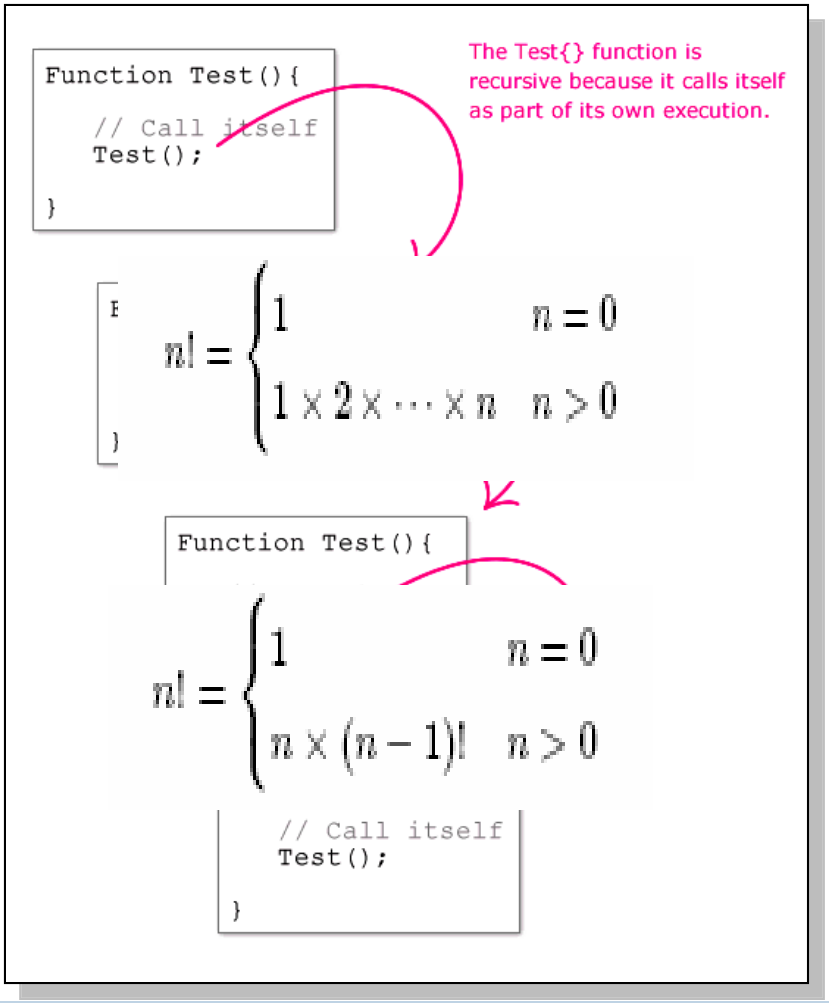
```
if (strcmp(z,"+")==0||strcmp(z,"-")==0 ||
   strcmp(z,"*")==0 ||
strcmp(z,"/")==0 || strcmp(z,"(")==0
   ||strcmp(z,")")==0)
{ strcpy(v,s[top]); top--;strcpy(u,s[top]); top--;
x=atof(u); y= atof(v);
if(strcmp(z,"+")==0) r = x + y;
if(strcmp(z,"-")==0) r = x - y;
if(strcmp(z,"*")==0) r = x * y;
if(strcmp(z,"/")==0) r = x / y;
sprintf(w,"%f",r); top++;
printf("\nx = %0.2f, y = %0.2f, w = %s",x,y,w);
   getch();
strcpy(s[top],w);
}
else { top++; strcpy(s[top],z); };
j++; strcpy(z,e1[j]);
}
return atof(s[top]);
}
```

```
void main()
{ float gt;
printf("\nnhay nhap so va dau + - * / () # : \n");
i=1;
do {gets(z); strcpy(e[i],z); i++; } while (strcmp(z,"#")
  !=0);
printf("\nbieu thuc so hoc goc :"); n = i-1;
for (i=1; i<=n;i++) printf("%s ",e[i]); printf("\n");
taobtbalan();
printf("\nbieu thuc toan hoc ba lan : "); j=1;
do { printf("%s ",e1[j]); strcpy(z,e1[j]); j++;
   } while (strcmp(z,"#") !=0) ;
printf("\ntinh gia tri bieu thuc : "); gt = giatri();
printf("\n\n  gia tri bieu thuc la : %0.2f",gt) ;
getch();
}
```

Ứng dụng ngăn xếp để loại bỏ đệ qui của chương trình

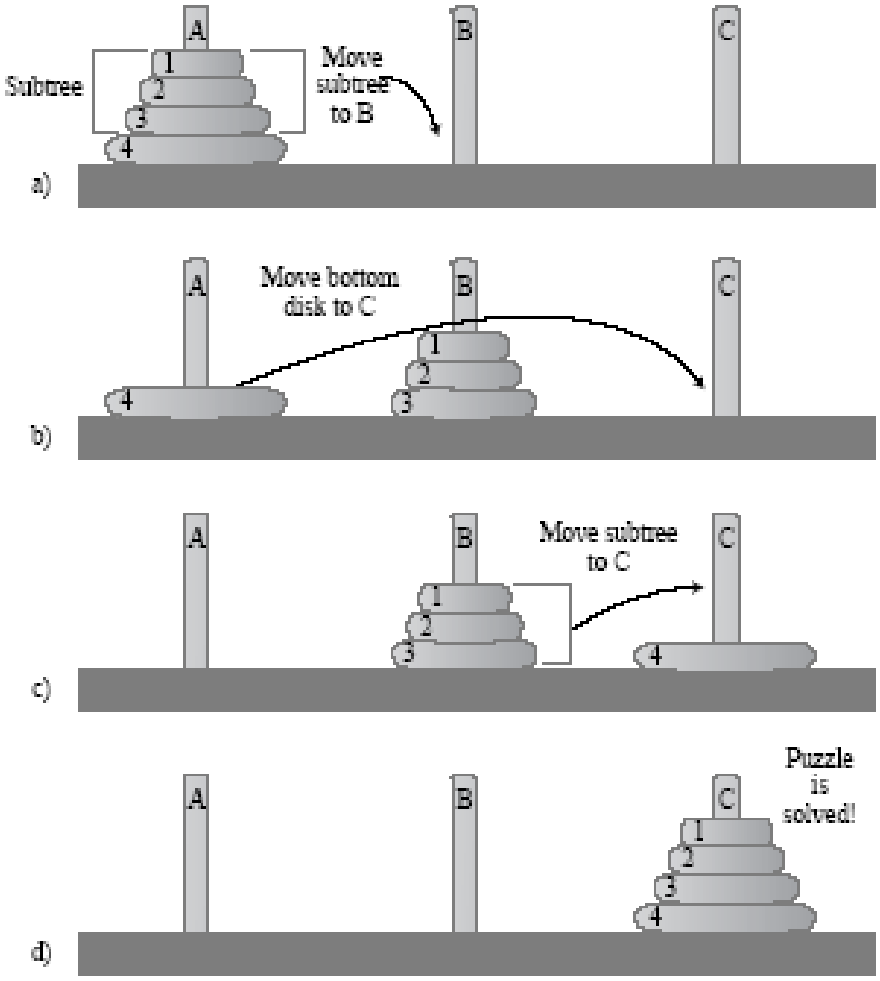
- Đệ qui là một lệnh trong thân hàm gọi đến chính nó.
- Vấn đề: đệ qui sẽ ngừng khi nào?

```
#include <iostream.h>
int giaiThua(int n);
void main()
{       int gt4, gt7;
        gt4 = giaiThua(4);
        cout << "4! =" << gt4 << endl;}
int giaiThua(int n)
{       int gt;
        if(n==1) return(1);
        gt = giaiThua(n-1)*n;
        return gt;}
```



Ứng dụng ngăn xếp để loại bỏ đệ qui của chương trình

- Dùng ngăn xếp để loại bỏ chương trình đệ qui của Bài toán Tháp Hà Nội
- Có ba cọc A,B,C. Khởi đầu cọc A có một số đĩa xếp theo thứ tự nhỏ dần lên trên đỉnh. Bài toán đặt ra là phải chuyển toàn bộ chồng đĩa từ A sang B. Mỗi lần thực hiện chuyển một đĩa từ một cọc sang một cọc khác và không được đặt đĩa lớn nằm trên đĩa nhỏ



Giải một số bài tập đệ quy

Giải thuật đệ quy bài toán Tháp Hà Nội:

- Trường hợp suy biến (điểm dừng):

  Nếu n = 1 thì chuyển đĩa từ A qua C

- Trường hợp chung (n ≥ 2):

  Thử với n=2: + Chuyển đĩa thứ nhất từ A sang B

  + Chuyển đĩa thứ hai từ A sang C

  + Chuyển đĩa thứ nhất từ B sang C

→ Tổng quát:

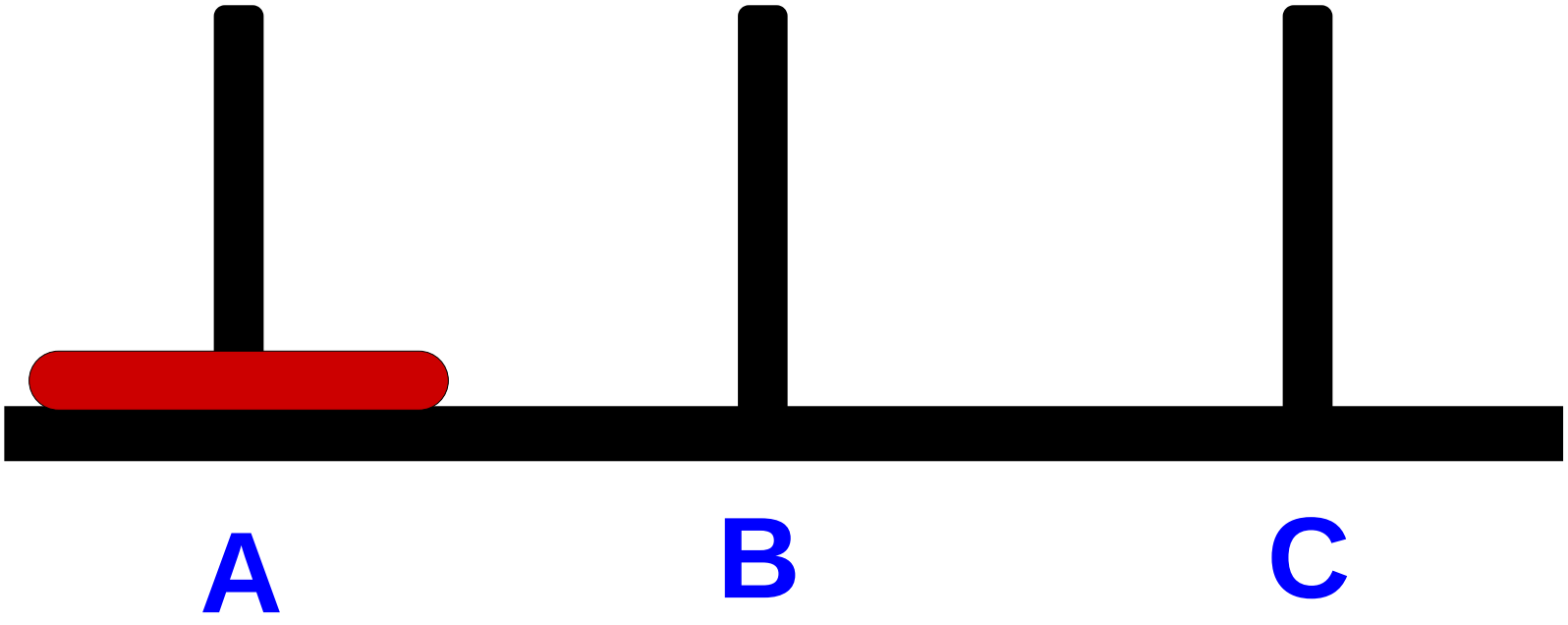
+ Chuyển (n -1) đĩa từ A sang B (C làm trung gian)

+ Chuyển 1 đĩa từ A sang C (B làm trung gian)

+ Chuyển (n -1) đĩa từ B sang C (A làm trung gian)

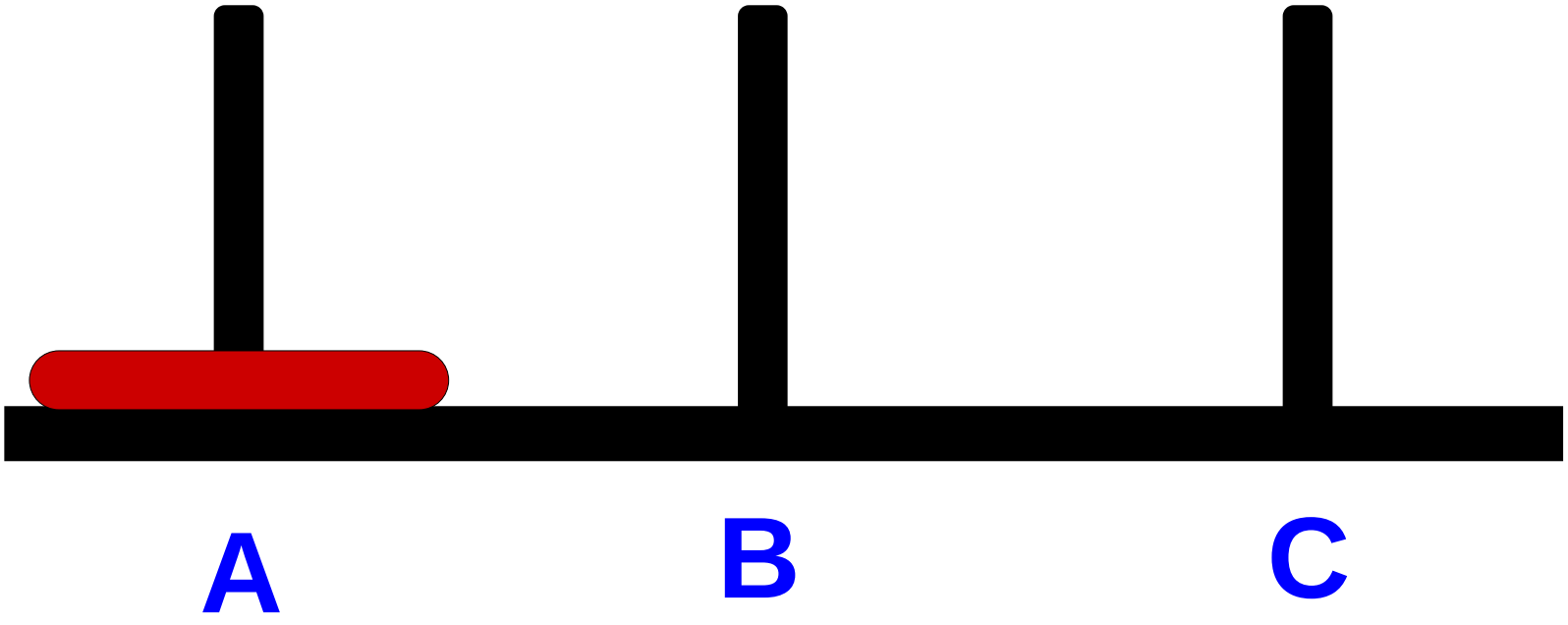
Giải một số bài tập đệ quy

1 đĩa



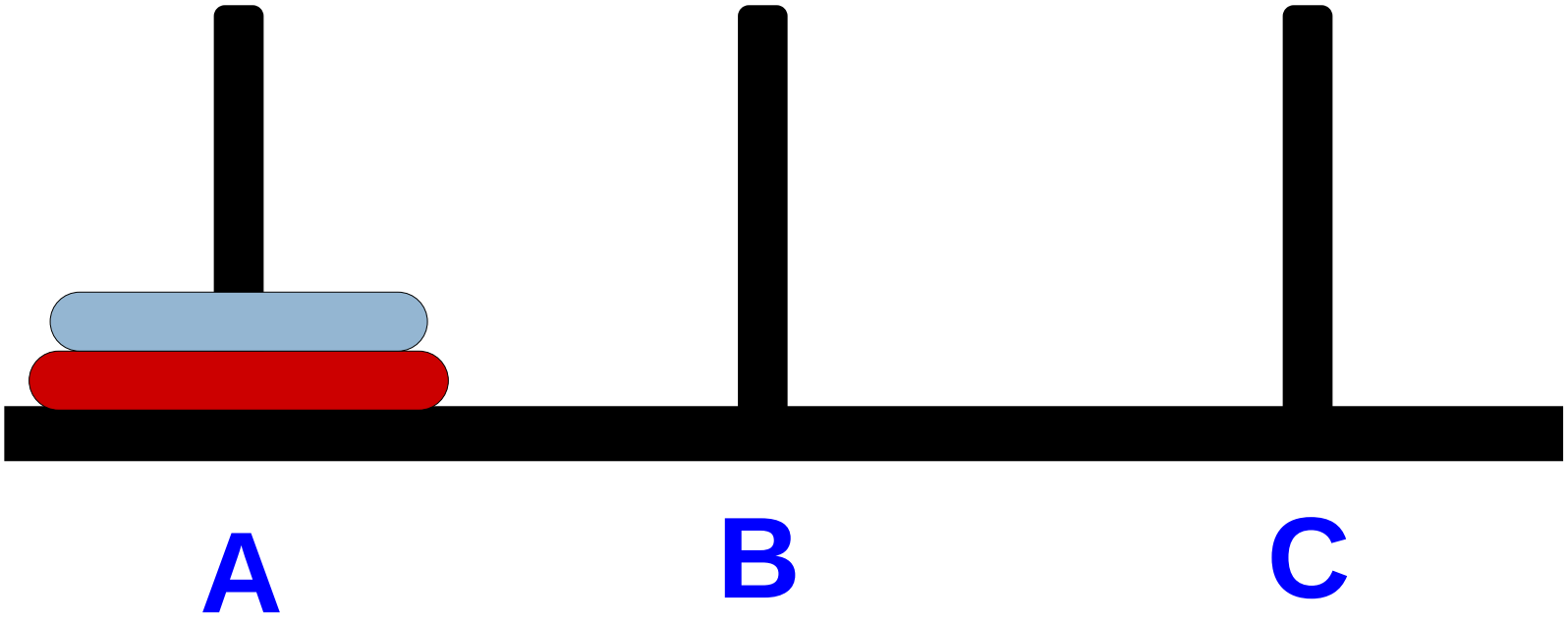
Giải một số bài tập đệ quy

1 đĩa



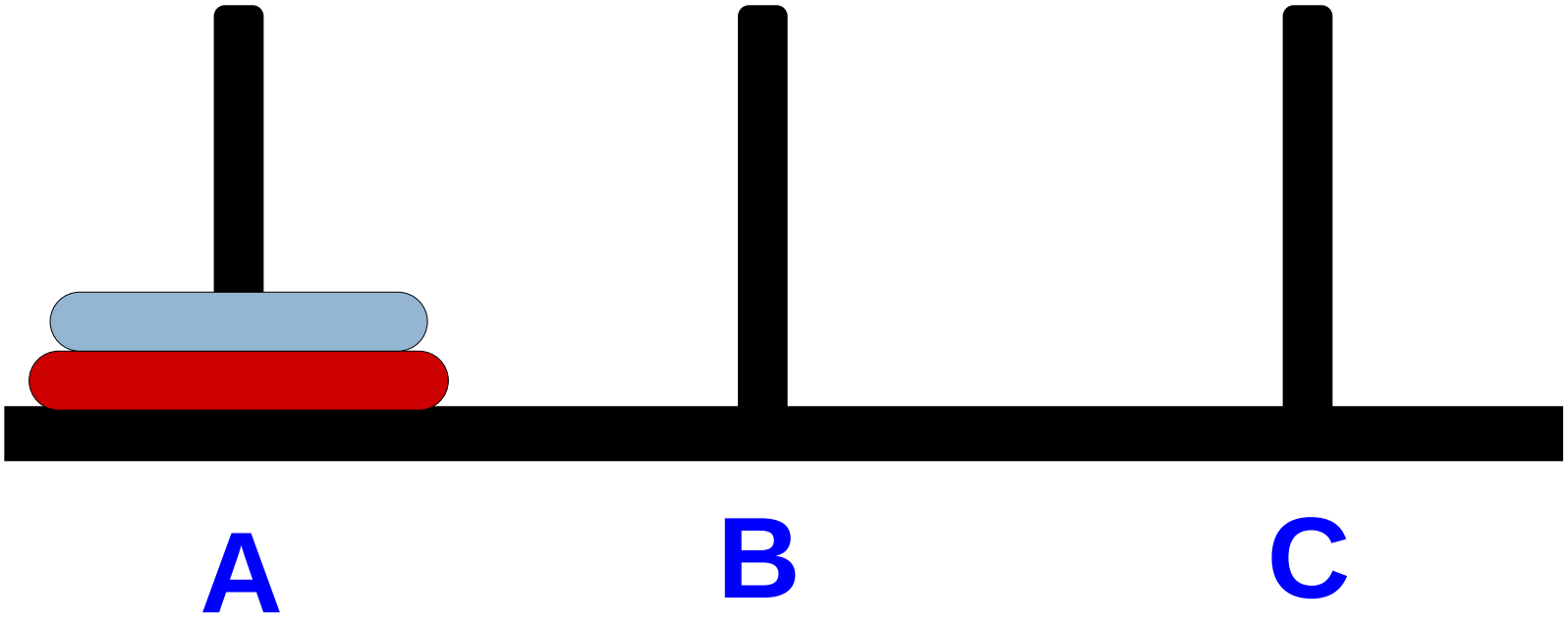
Giải một số bài tập đệ quy

2 đĩa



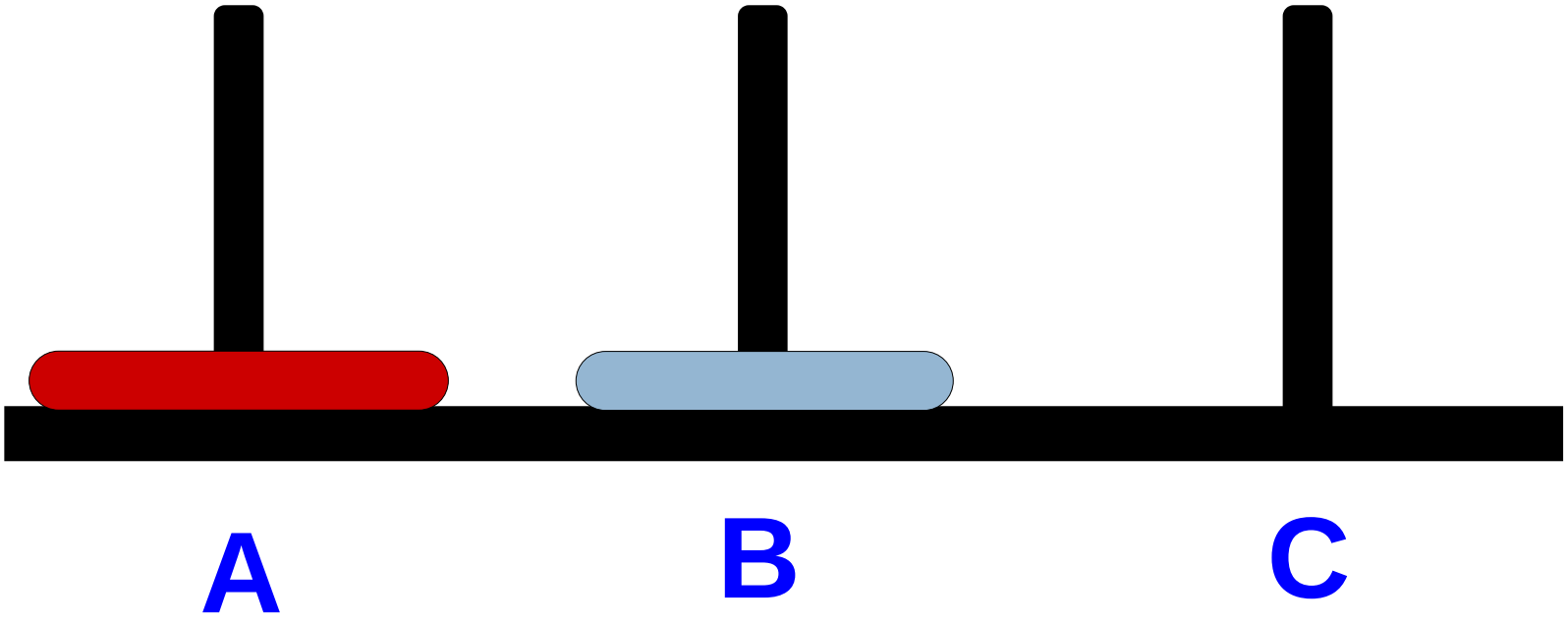
Giải một số bài tập đệ quy

2 đĩa



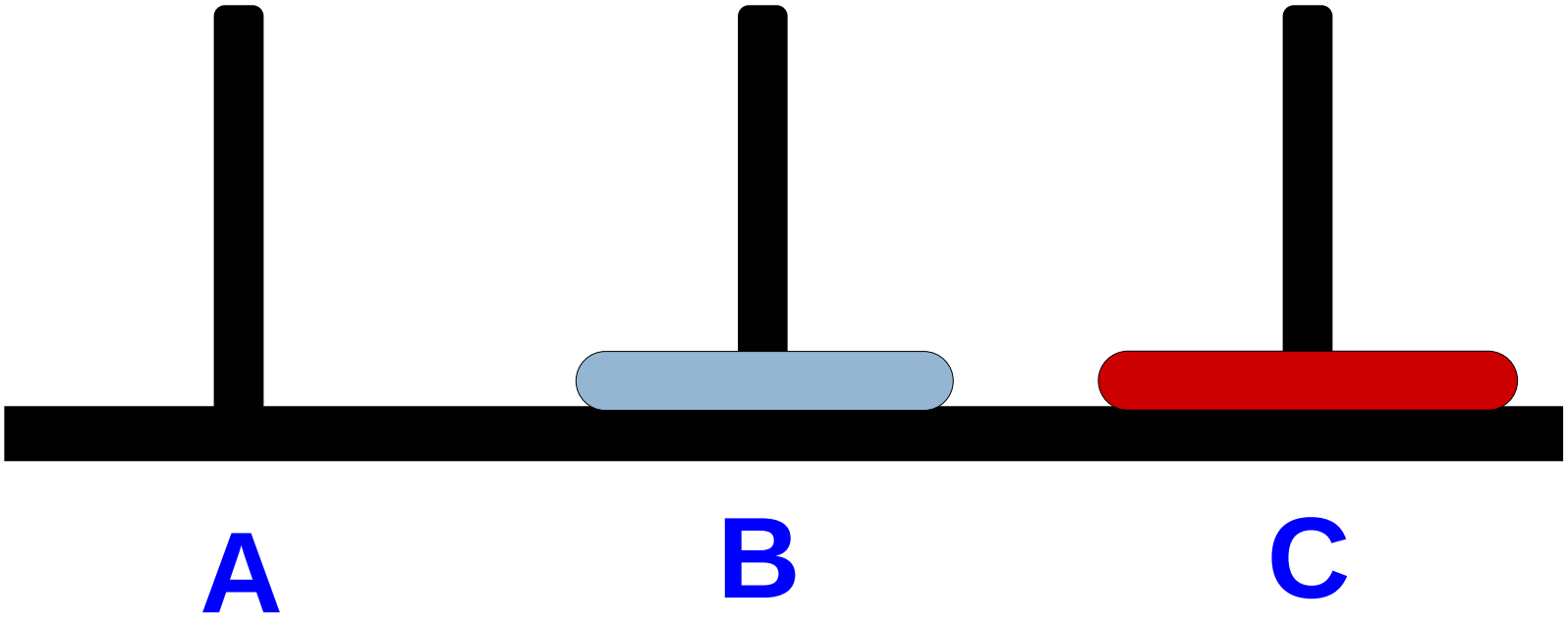
Giải một số bài tập đệ quy

2 đĩa

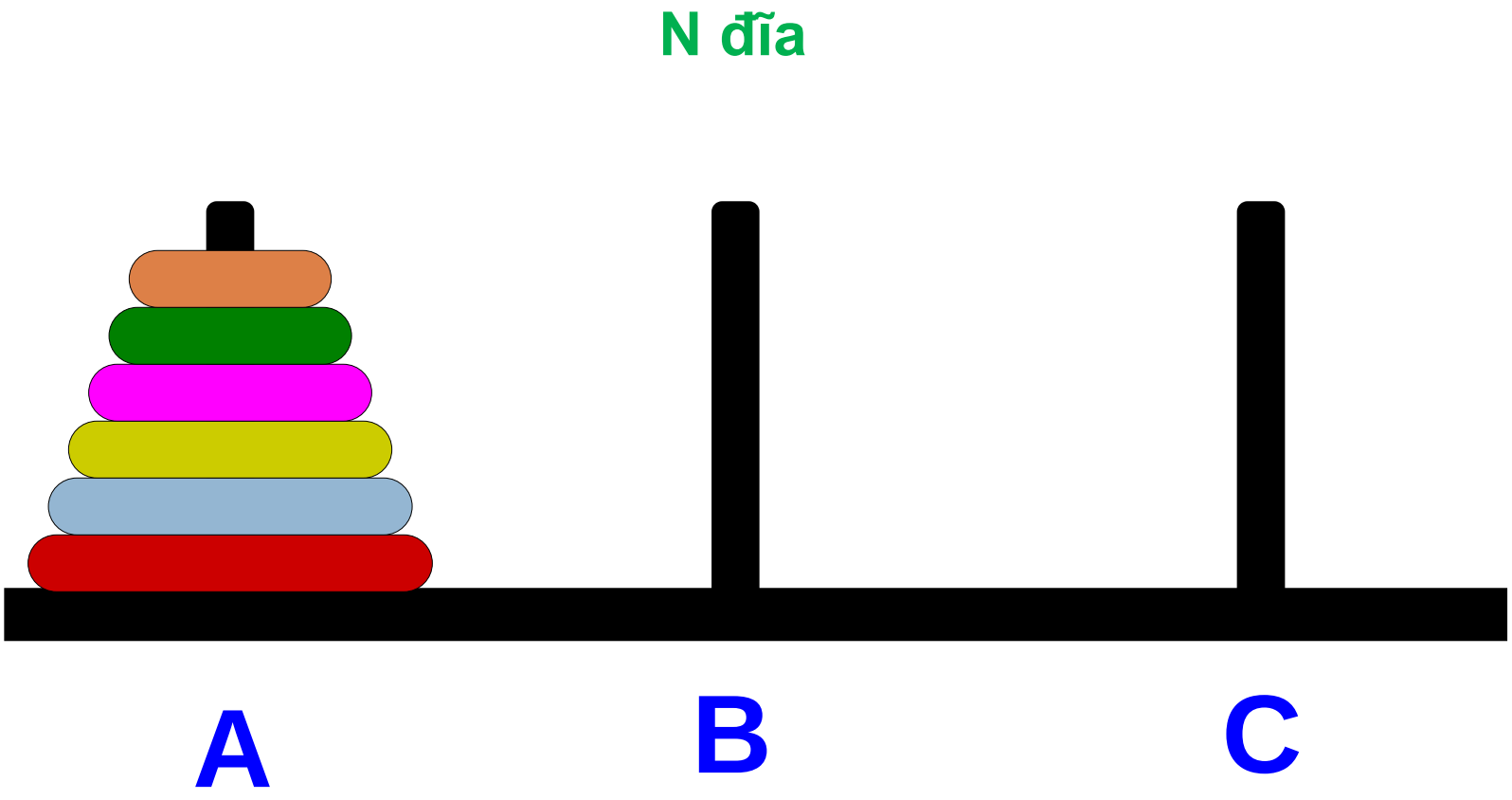


Giải một số bài tập đệ quy

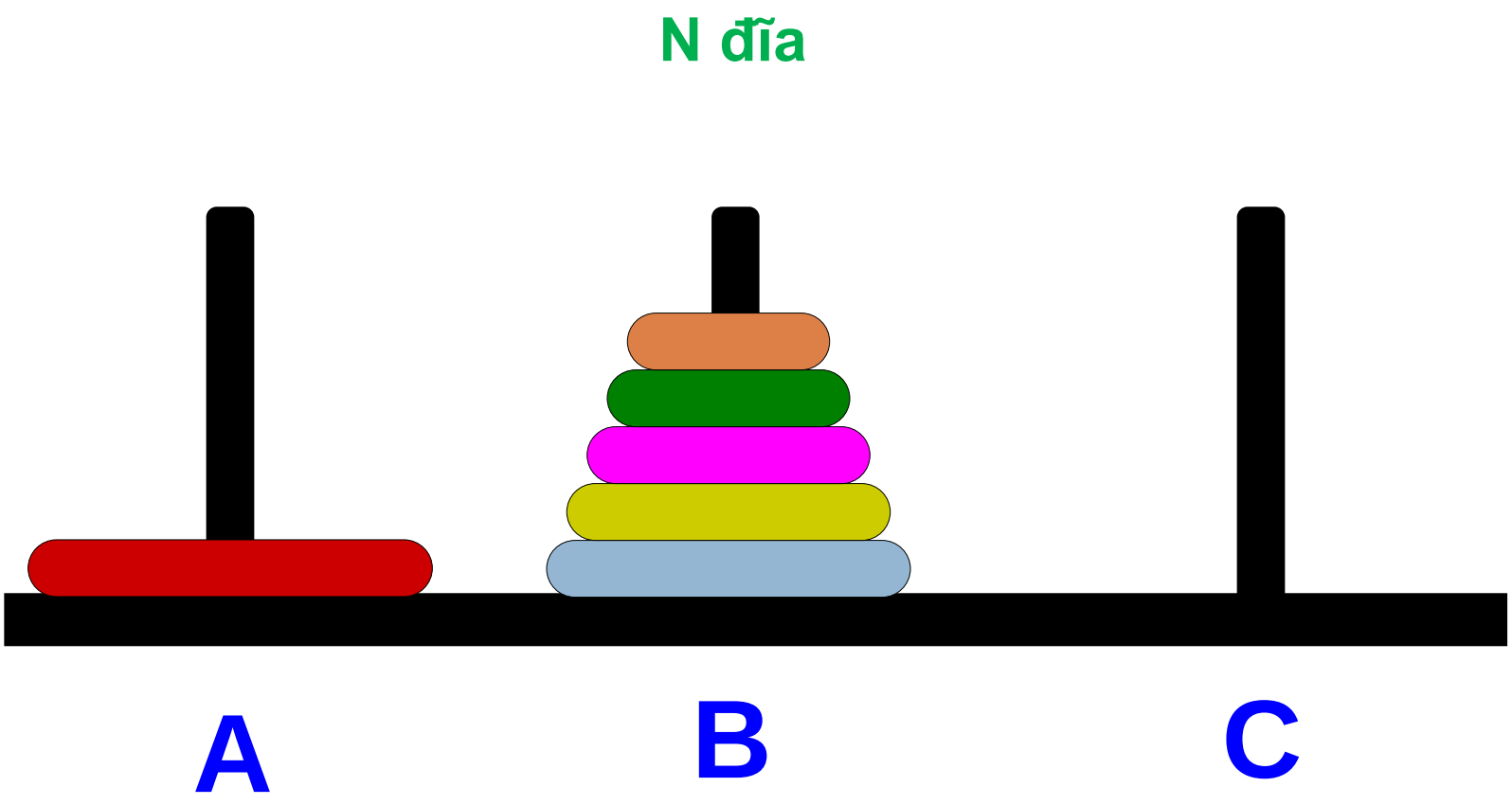
2 đĩa



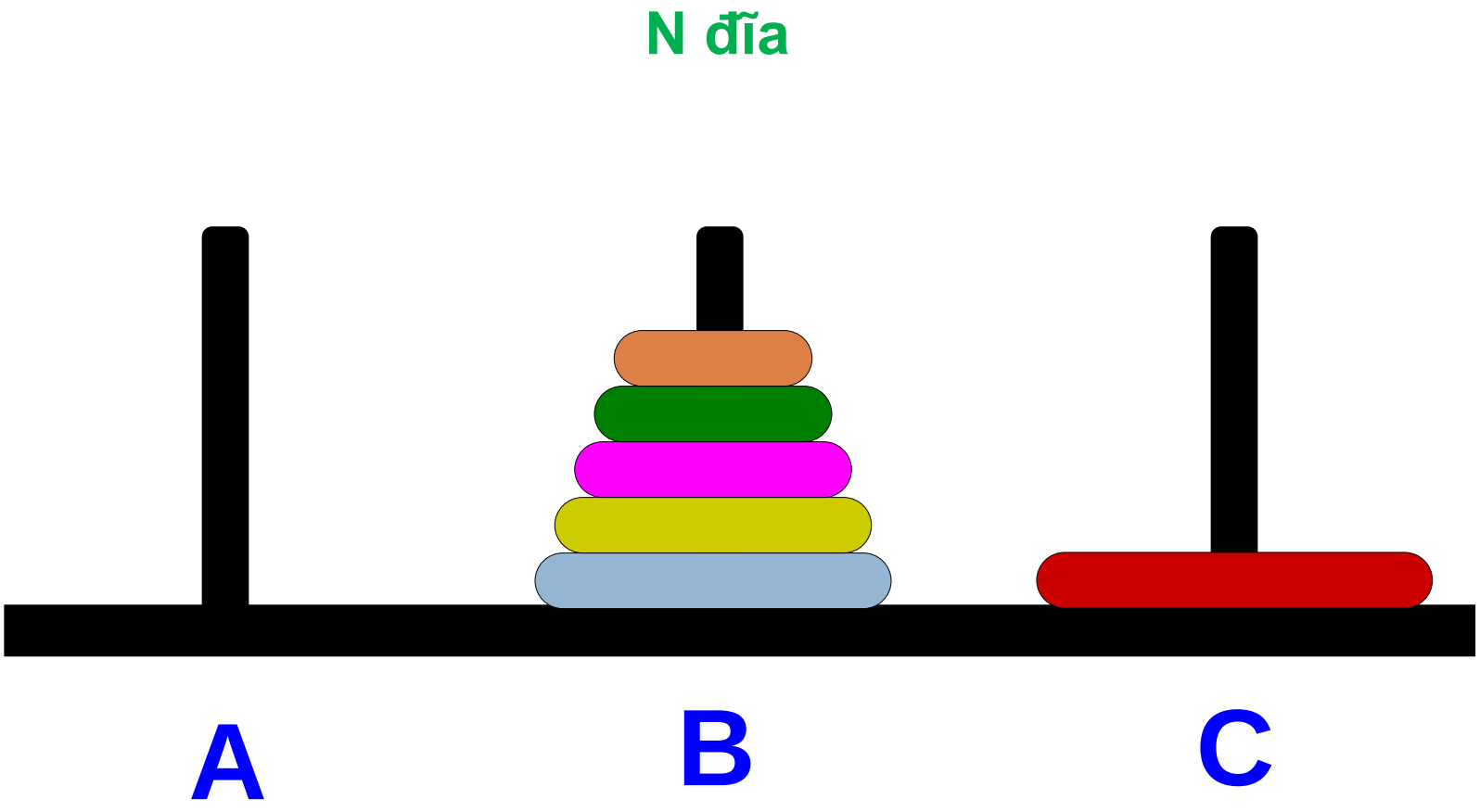
Giải một số bài tập đệ quy



Giải một số bài tập đệ quy



Giải một số bài tập đệ quy



Giải một số bài tập đệ quy

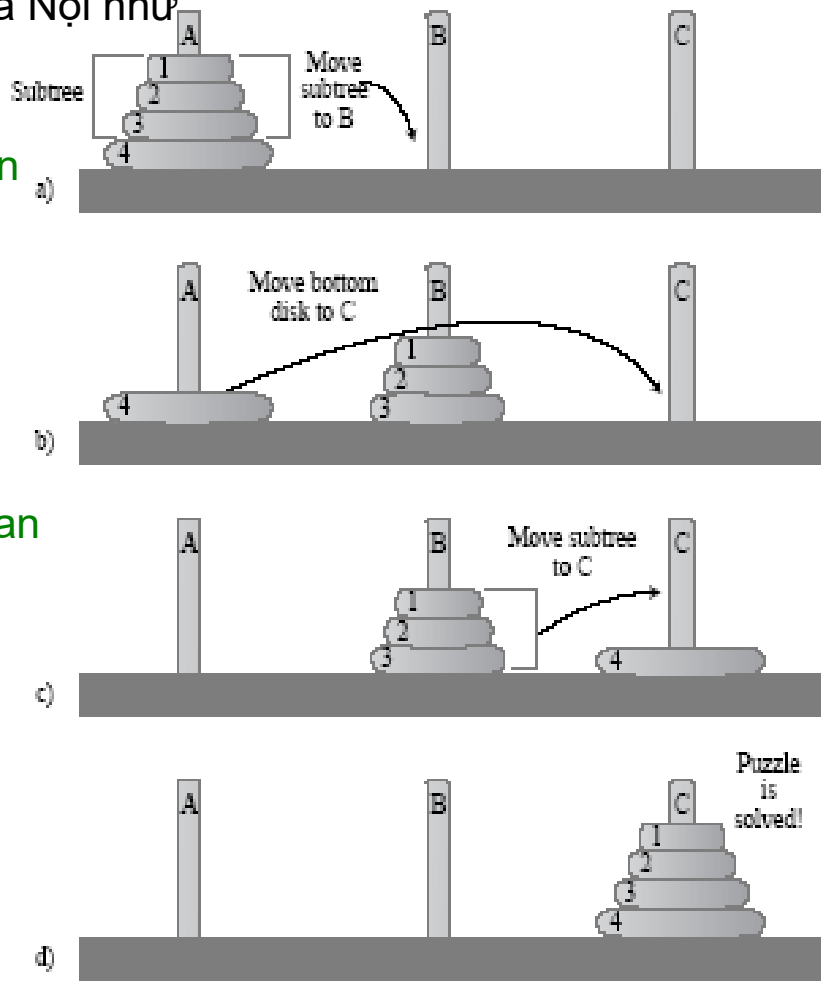
Giải thuật đệ quy bài toán Tháp Hà Nội:

```
void HaNoi( int n, char A, char B, char C ) {
        if (n==1)
                cout<<A<<"→"<< C;
        else{
                HaNoi(n -1, A, C, B);
                HaNoi(1, A, B, C);
                HaNoi(n -1, B, A, C);
        }
}
```

Ứng dụng ngăn xếp để loại bỏ đệ qui của chương trình

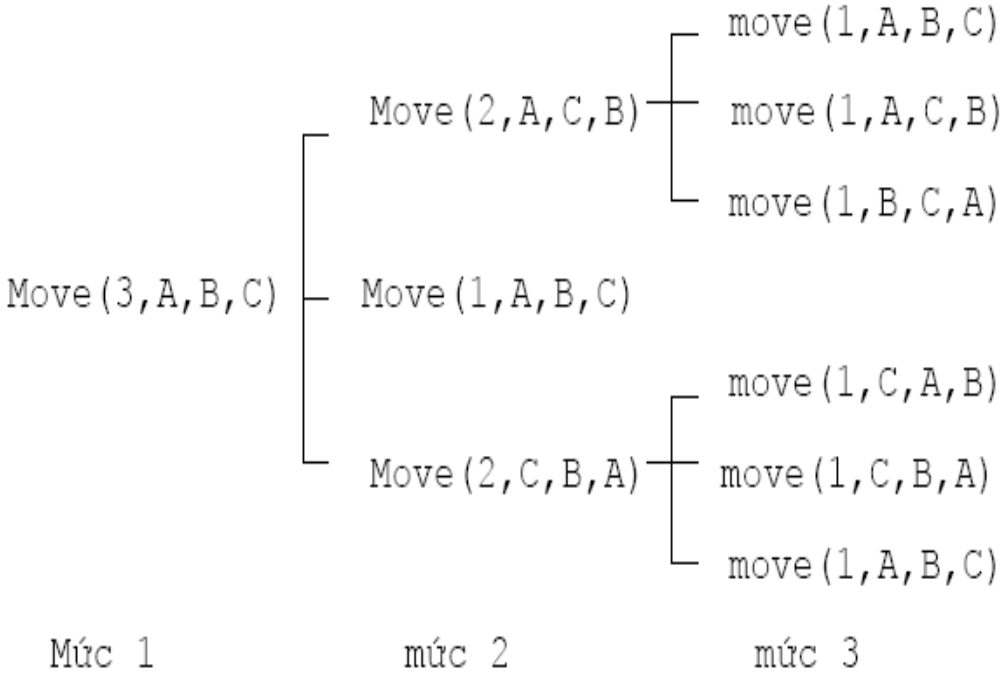
Chương trình con đệ qui để giải bài toán tháp Hà Nội như sau:

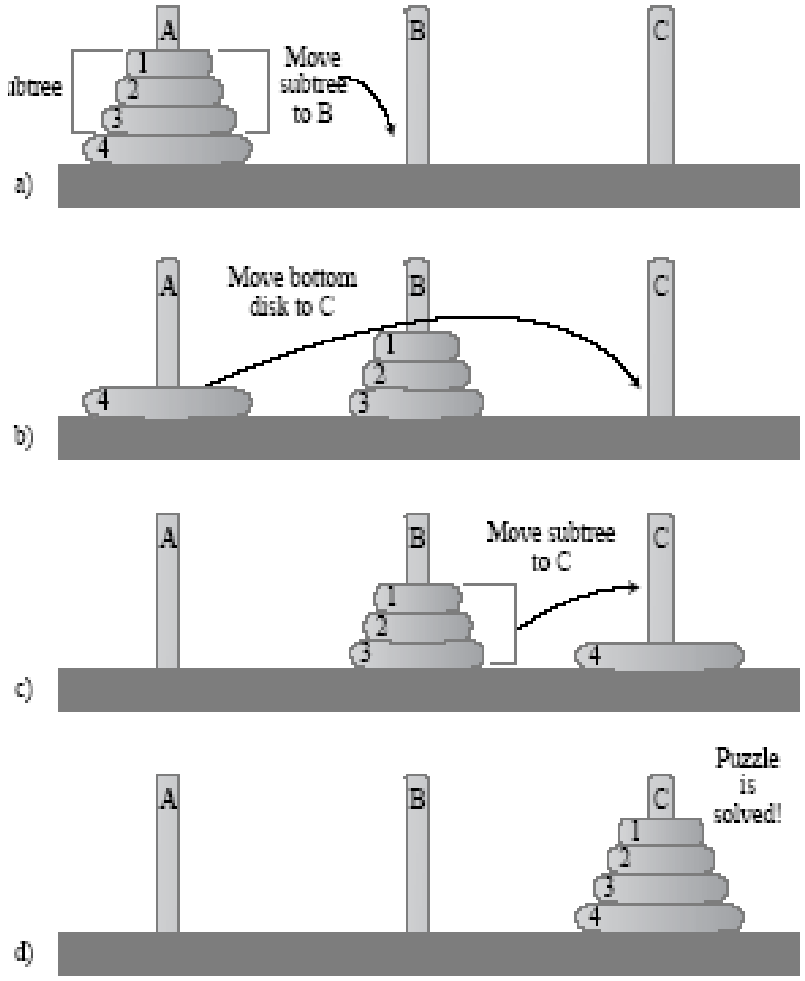
```
void Move(int N, int A, int B, int C)
{//n: số đĩa, A,C,B: cọc nguồn , đích và trung gian
    if (n==1)
            printf("Chuyen 1 dia tu %c
            sang %c\n",Temp.A,Temp.C);
    else {
            Move(n-1, A,B,C);
//chuyển n-1 đĩa từ cọc nguồn sang cọc trung gian
            Move(1,A,C,B);
//chuyển 1 đĩa từ cọc nguồn sang cọc đích
            Move(n-1,B,C,A);
//chuyển n-1 đĩa từ cọc trung gian sang cọc đích
    }
}
```



Ứng dụng ngăn xếp để loại bỏ đệ qui của chương trình

Đổi B thành C, C thành B





Ứng dụng ngăn xếp để loại bỏ đệ qui của chương trình

Qui tắc khử đệ qui:

- Mỗi khi chương trình con đệ qui được gọi, ứng với việc đi từ mức i vào mức i+1, ta phải lưu trữ các biến cục bộ của chương trình con ở bước i vào ngăn xếp. Ta cũng phải lưu "địa chỉ mã lệnh" chưa được thi hành của chương trình con ở mức i.
- Lập trình bằng ngôn ngữ cấp cao thì đây không phải là địa chỉ ô nhớ chứa mã lệnh của máy mà ta sẽ tổ chức sao cho khi mức i+1 hoàn thành thì lệnh tiếp theo sẽ được thực hiện là lệnh đầu tiên chưa được thi hành trong mức i.
- Tập hợp các biến cục bộ của mỗi lần gọi chương trình con xem như là một mẩu tin hoạt động (activation record).
- Mỗi lần thực hiện chương trình con tại mức i thì phải xoá mẩu tin lưu các biến cục bộ ở mức này trong ngăn xếp.

Ứng dụng ngăn xếp để loại bỏ đệ qui của chương trình

- Ý tưởng này thể hiện trong cài đặt khử đệ qui cho bài toán tháp Hà Nội là: mẩu tin lưu trữ các biến cục bộ của chương trình con thực hiện sau thì được đưa vào ngăn xếp trước để nó được lấy ra dùng sau.

```
//Kiểu cấu trúc lưu trữ biến cục bộ
Struct phantu{
        int N;   //số đĩa
        int A, B, C;   //nguồn, trung gian,  dich
} ;
```

Ứng dụng ngăn xếp để loại bỏ đệ qui của chương trình

```
// Chương trình con MOVE không đệ qui
    void Move(ElementType X){
        ElementType Temp, Temp1;
        Stack S;
        Create_Stack(&S);
        Push(X,&S);
        do
        {
            Temp=Top(S); //Lay phan tu dau
            Pop(&S); //Xoa phan tu dau
```

Ứng dụng ngăn xếp để loại bỏ đệ qui của chương trình

```
if (Temp.N==1)
    printf("Chuyen 1 dia tu %c sang %c\n",Temp.A,Temp.B);
else
{
    // Luu cho loi goi Move(n-1,C,B,A)
    Temp1.N=Temp.N-1;
    Temp1.A=Temp.C;
    Temp1.B=Temp.B;
    Temp1.C=Temp.A;
    Push(Temp1,&S);
```

Ứng dụng ngăn xếp để loại bỏ đệ qui của chương trình

```
//Luu cho loi goi Move(n-1,A,C,B)
    Temp1.N=Temp.N-1;
    Temp1.A=Temp.A;
    Temp1.B=Temp.C;
    Temp1.C=Temp.B;
    Push(Temp1,&S);
}
} while (!Empty_Stack(S));
}
```

Ứng dụng ngăn xếp để loại bỏ đệ qui của chương trình

Minh họa cho lời gọi Move(x) với 3 đĩa, tức là x.N=3.

Ngăn xếp khởi đầu:

3,A,B,C

Ngăn xếp sau lần lặp thứ nhất:

2,A,C,B
1,A,B,C
2,C,B,A

Ngăn xếp sau lần lặp thứ hai

1,A,B,C
1,A,C,B
1,B,C,A
1,A,B,C
2,C,B,A

Các lần lặp 3,4,5,6

2,C,B,A

Lần lặp 7

1,C,A,B
1,C,B,A
1,A,B,C

Chương trình con in ra các phép chuyển và dẫn đến ngăn xếp rỗng.