NGĂN XẾP - HÀNG ĐỢI
HÀNG ĐỢI ƯU TIÊN

1. Cấu trúc dữ liệu ngăn xếp:

a. Giới thiệu:

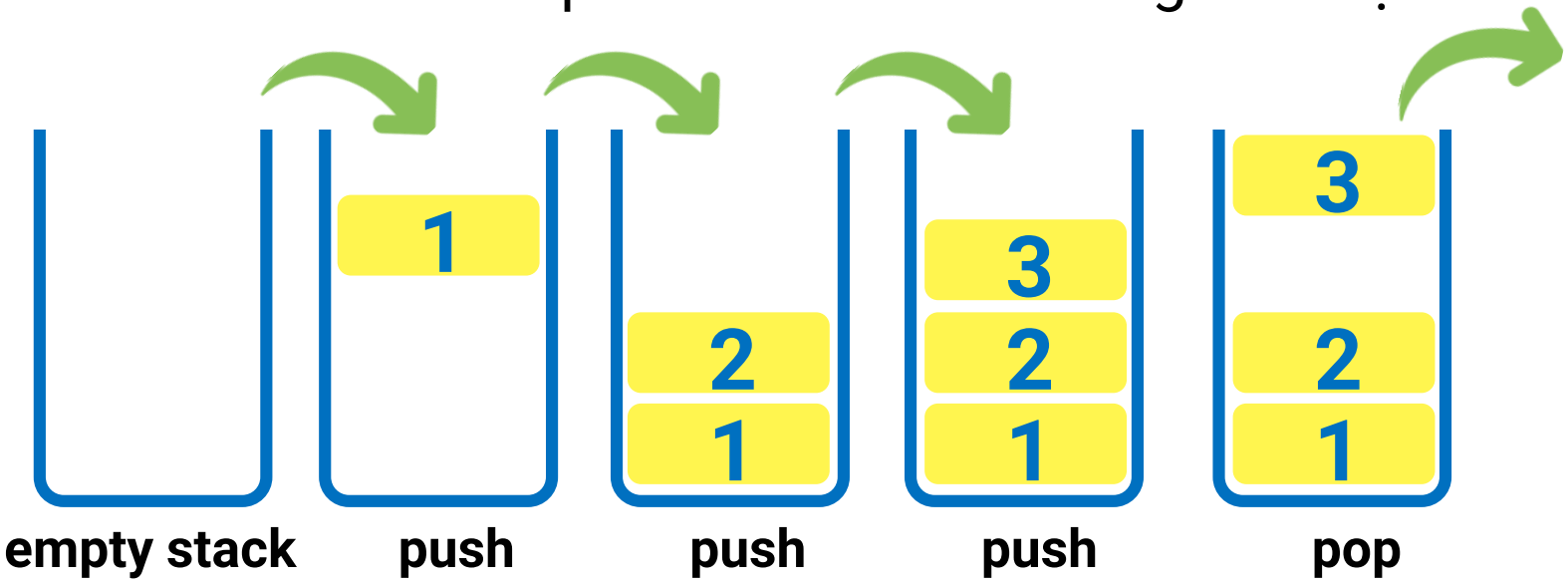
Ngăn xếp (stack) là một cấu trúc dữ liệu có quan hệ mật thiết với cơ chế hoạt động của đệ quy. Để hiểu được cách hàm đệ quy hoạt động, ta cần nắm được cách hoạt động của cấu trúc dữ liệu ngăn xếp,

Ngăn xếp là một cấu trúc dữ liệu hỗ trợ 2 thao tác push và pop. Trong đó push giúp thêm 1 phần tử vào đỉnh ngăn xếp, pop giúp xóa 1 phần tử khỏi đỉnh ngăn xếp. Cả 2 thao tác này đều được thực hiện ở đỉnh ngăn xếp.

1. Cấu trúc dữ liệu ngăn xếp:

a. Giới thiệu:

Ngăn xếp hoạt động theo nguyên tắc viết tắt là LIFO (Last In First Out) nghĩa là vào cuối thì ra đầu. Các phần tử vào cuối cùng sẽ được ra đầu tiên.



Trong chương trình tồn tại một bộ nhớ là bộ nhớ ngăn xếp, cách hoạt động của bộ nhớ này tương tự như cách hoạt động của cấu trúc dữ liệu ngăn xếp.

1. Cấu trúc dữ liệu ngăn xếp:

b. Các hàm phổ biến của stack:

```
Khai báo ngăn xếp: stack<data_type> stack_name;
Ví dụ: stack<int> st;
```

Hàm	Chức năng
push(x)	Thêm x vào đỉnh ngăn xếp
pop()	Xóa phần tử khỏi ngăn xếp (chú ý nếu ngăn xếp rỗng việc pop sẽ gây lỗi)
size()	Trả về số lượng phần tử trong ngăn xếp
empty()	Kiểm tra ngăn xếp rỗng
top()	Lấy về phần tử ở đỉnh ngăn xếp nhưng không xóa

c. Một vài ứng dụng của ngăn xếp:

Bài toán 1: Kiểm tra biểu thức dấu ngoặc đúng. Ví dụ về biểu thức dấu ngoặc đúng: (), ((())), (())()...

Ý tưởng:

Sử dụng ngăn xếp để lưu các dấu mở ngoặc, nếu gặp dấu đóng ngoặc thì kiểm tra trong stack có dấu mở ngoặc tương ứng hay không, nếu có thì xóa dấu mở ngoặc đó đi, ngược lại có thể kết luận biểu thức là không đúng.

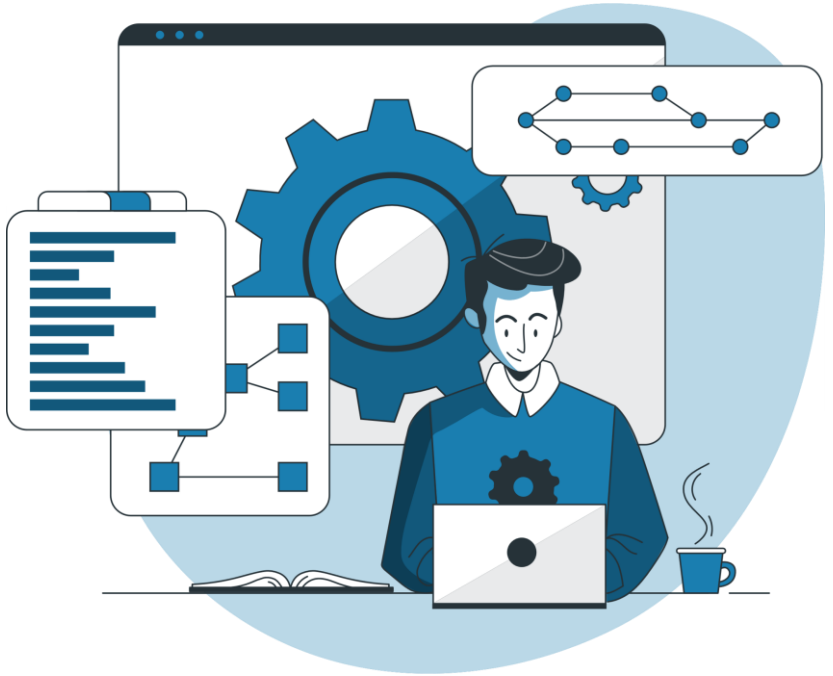
Code

```
bool check(string s){
    stack<char> st;
    for(char x : s){
        if(x == '('){
            st.push(x);
        }
        else{
            if(st.empty()) return false;
            st.pop();
        }
    }
    return st.empty();
}
```

c. Một vài ứng dụng của ngăn xếp:

Bài toán 2: Biến đổi biểu thức trung tố, tiền tố, hậu tố. Có 3 biểu thức thường gặp, biểu thức mà các bạn hay sử dụng là biểu thức trung tố, biểu thức ở dạng trung tố thân thiện với con người nhưng lại khó khăn với máy tính trong quá trình tính toán giá trị của biểu thức.

- Biểu thức trung tố (Infix) : (A + B) * (C - D)
- Biểu thức tiền tố (Prefix) : *+AB-CD
- Biểu thức hậu tố (Postfix) : AB+CD-*



c. Một vài ứng dụng của ngăn xếp:

Biến đổi tiền tố - trung tố:

Thuật toán:

Duyệt từ cuối về đầu của biểu thức tiền tố, nếu gặp toán hạng thì đẩy vào ngăn xếp, nếu gặp toán tử thì lấy và xóa 2 toán hạng ở đỉnh ngăn xếp và áp dụng toán tử này với 2 toán hạng đó (Chú ý thứ tự và bổ sung đóng mở ngoặc tròn), đẩy ngược biểu thức thu được lại vào ngăn xếp. Biểu thức ở đỉnh ngăn xếp cuối cùng là biểu thức trung tố thu được.

Biểu thức trung tố (Infix): (A + B) * (C - D)

Kí tự (Duyệt từ cuối về đầu)	Trạng thái ngăn xếp
D	D
C	C, D
-	(C - D)
B	B, (C - D)
A	A, B, (C - D)
+	(A + B), (C - D)
*	((A + B) * (C - D))

c. Một vài ứng dụng của ngăn xếp:

Biến đổi tiền tố - trung tố:

Code

```
void solve(string s){
    stack<string> st;
    for(int i=s.length()-1;i>=0;i--){
        if(isalpha(s[i])){
            st.push(string(1,s[i]));
        }
        else{
            string operand1 = st.top();st.pop();
            string operand2 = st.top();st.pop();
            string tmp = "("+operand1+s[i]+operand2+")";
            st.push(tmp);
        }
    }
    cout<<st.top()<<endl;
}
```

c. Một vài ứng dụng của ngăn xếp:

Biến đổi tiền tố - hậu tố:

Biểu thức tiền tố : *+AB-CD

Kí tự (Duyệt từ cuối)	Trạng thái ngăn xếp
D	D
C	C, D
-	CD-
B	B, CD-
A	A, B, CD-
+	AB+, CD-
*	AB+CD-*

Code

```
void solve(string s){
    stack<string> st;
    for(int i=s.length()-1;i>=0;i--){
        if(isalpha(s[i])){
            st.push(string(1,s[i]));
        }
        else{
            string op1 = st.top();st.pop();
            string op2 = st.top();st.pop();
            string tmp = op1 + op2 + s[i];
            st.push(tmp);
        }
    }
    cout<<st.top()<<endl;
}
```

c. Một vài ứng dụng của ngăn xếp:

Biến đổi trung tố - hậu tố:

Thuật toán (Áp dụng cho cả biểu thức trung tố không đầy đủ ngoặc)

Duyệt biểu thức trung tố từ đầu tới cuối

- Nếu là toán hạng thì thêm vào xâu kết quả
- Nếu là dấu '(' thì thêm vào ngăn xếp
- Nếu là dấu ')' thì lần lượt pop các kí tự trong ngăn xếp và đưa vào xâu kết quả cho tới khi gặp dấu ')' tương ứng.
- Nếu là toán hạng: Lần lượt pop các toán hạng ở đỉnh ngăn xếp nếu thứ tự ưu tiên của nó lớn hơn hoặc bằng thứ tự ưu tiên của kí tự đang xét. Đẩy toán hạng này vào ngăn xếp.

Lần lượt đưa mọi ký tự trong ngăn xếp và thêm vào cuối xâu kết quả.

c. Một vài ứng dụng của ngăn xếp:

Biến đổi trung tố - hậu tố:

Biểu thức trung tố : (A + B) – C * (D / E) + F

Thứ tự ưu tiên : ^ : 3, * / : 2, + - : 1, (: 0

Kí tự (Duyệt xuôi)	Trạng thái ngăn xếp	Xâu kết quả
(	(, (	
A	(	A
+	+, (	A
B	+, (	AB
)		AB+
-	-	AB+
C	-	AB+C
*	*, -	AB+C
(	(, *, -	AB+C
D	(, *, -	AB+CD
/	/, (, *, -	AB+CD
E	/, (, *, -	AB+CDE
)		AB+CDE/*-
+	+	AB+CDE/*-
F	+	AB+CDE/*-F
Duyệt xong	Rỗng	AB+CDE/*-F+

c. Một vài ứng dụng của ngăn xếp:

Tính toán giá trị biểu thức trung tố, các số xuất hiện trong biểu thức là số có 1 chữ số.

```
void solve(string s){
   stack<int> st;
   for(int i=0;i<s.length();i++){
      if(isdigit(s[i])){
         st.push(s[i]-'0');
      }
      else{
         int t2 = st.top();st.pop();
         int t1 = st.top();st.pop();
         if(s[i]=='+') st.push(t1+t2);
         else if(s[i]=='-') st.push(t1-t2);
         else if(s[i]=='*') st.push(t1*t2);
         else st.push(t1/t2);
      }
      //cout<<"loop\n";
   }
   cout<<st.top()<<endl;
}
```

```
int main(){
   solve("332*+5-");
}
```

c. Một vài ứng dụng của ngăn xếp:

Tính toán giá trị biểu thức hậu tố, các số xuất hiện trong biểu thức là số có 1 chữ số.

```
void solve(string s){
   stack<int> st;
   for(int i=s.length()-1;i>=0;i--){
      if(isdigit(s[i])){
         st.push(s[i]-'0');
      }
      else{
         int o1 =st.top();st.pop();
         int o2 =st.top();st.pop();
         if(s[i]=='+') st.push(o1+o2);
         else if(s[i]=='/') st.push(o1/o2);
         else if(s[i]=='-') st.push(o1-o2);
         else st.push(o1*o2);
      }
   }
   cout<<st.top()<<endl;
}
int main(){
   solve("-+9/532");
}
```

Tính toán giá trị biểu thức trung tố, các số trong biểu thức có thể có nhiều chữ số

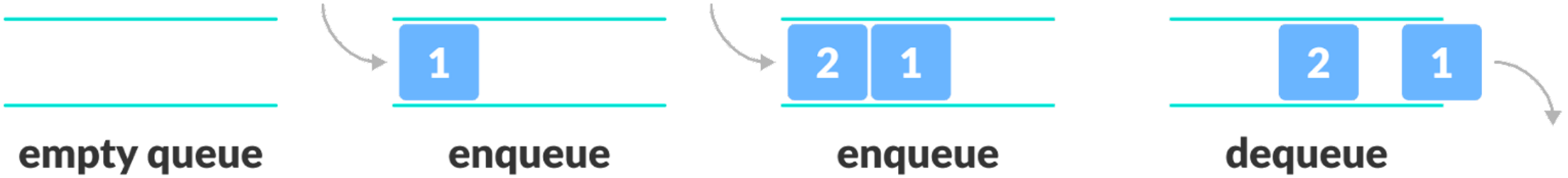
```
int pre(char c){
   if(c=='*' || c=='/') return 2;
   else if(c=='+' || c=='-') return 1;
   return 0;
}
ll compute(ll a,ll b, char c){
   if(c=='+') return a+b;
   else if(c=='-') return a-b;
   else if(c=='*') return a*b;
   else return a/b;
}
void solve(string s){
   stack<char> st1;
   stack<ll> st2;
   for(int i=0;i<sz(s);i++){
      if(s[i]=='('){
         st1.push(s[i]);
      }
      else if(isdigit(s[i])){
         ll tmp = 0;
         while(i<sz(s) && isdigit(s[i])){
            tmp = tmp*10+s[i]-'0'; ++i;
         }
         --i; st2.push(tmp);
      }
      else if(s[i]==')'){
         while(!st1.empty() && st1.top()!='('){
            ll o1 = st2.top();st2.pop();
            ll o2 = st2.top();st2.pop();
            st2.push(compute(o2,o1,st1.top())); st1.pop();
         }
         if(!st1.top())
            st1.pop();
      }
      else{
         while(!st1.empty() && pre(s[i])<=pre(st1.top())){
            ll o1=st2.top();st2.pop();
            ll o2=st2.top();st2.pop();
            st2.push(compute(o2,o1,st1.top())); st1.pop();
         }
         st1.push(s[i]);
      }
   }
   while(!st1.empty()){
      ll o1=st2.top();st2.pop();
      ll o2=st2.top();st2.pop();

   st2.push(compute(o2,o1,st1.top()));
      st1.pop();
   }
    cout<<st2.top()<<endl;
}
```

2. Cấu trúc dữ liệu hàng đợi:

a. Giới thiệu:

Hàng đợi (Queue) là một cấu trúc dữ liệu quan trọng với cách thức hoạt động là **FIFO : First In First Out**, tức là phần tử nào vào trước sẽ được ra trước.

- **Thao tác enqueue:** Thêm 1 phần tử vào cuối hàng đợi
- **Thao tác dequeue:** Xóa 1 phần tử khỏi đầu hàng đợi



2. Cấu trúc dữ liệu hàng đợi:

b. Các hàm phổ biến của hàng đợi:

```
Khai báo hàng đợi: queue<data_type> queue_name;
Ví dụ: queue<int> q;
```

Hàm	Chức năng
push(x)	Thêm x vào cuối hàng đợi
front()	Truy cập vào phần tử ở đầu hàng đợi
pop()	Xóa phần tử khỏi đầu hàng đợi
size()	Trả về số phần tử trong hàng đợi
empty()	Kiểm tra hàng đợi rỗng
back()	Truy cập vào phần tử ở cuối hàng đợi

2. Cấu trúc dữ liệu hàng đợi:

c. Một vài ứng dụng của hàng đợi:

Bài toán 1: Tìm số thao tác ít nhất để biến đổi từ số thứ 1 sang số thứ 2. Dạng bài tập này ta coi mỗi trạng thái của số như 1 đỉnh của đồ thị và thao tác biến đổi giữa 2 số như là cạnh của đồ thị, khi đó bài toán tương tự như tìm đường đi ngắn nhất giữa 2 đỉnh của đồ thị

Cho hai số nguyên dương a và b và hai thao tác dưới đây:

- **Thao tác 1:** Trừ a đi 1 đơn vị
- **Thao tác 2:** Nhân b lên 2

Tìm số thao tác tối thiểu để biến đổi a thành b ?

c. Một vài ứng dụng của hàng đợi:

Code

```
int bfs(int s,int t){
   queue<pi> q;
   unordered_set<int> se;
   q.push({s,0}); se.insert(s);
   while(!q.empty()){
      pi tmp = q.front();
      q.pop();
      if(tmp.fi == t) return tmp.se;
      if(tmp.fi*2==t || tmp.fi-1==t) return tmp.se+1;
      if(!present(se, tmp.fi*2) && tmp.fi<t){
         q.push({tmp.fi*2,tmp.se+1});
         se.insert(tmp.fi*2);
      }
      if(!present(se, tmp.fi-1) && tmp.fi>=1){
         q.push({tmp.fi-1,tmp.se+1});
         se.insert(tmp.fi-1);
      }
   }
}
```

c. Một vài ứng dụng của hàng đợi:

Bài toán 2: Sinh ra các số từ các chữ số cho trước như số lộc phát, số nhị phân...

In ra N số lộc phát (chỉ bao gồm 6 và 8) theo thứ tự tăng dần

```
void solve(int n){
    queue<int> q; vector<int> res;
    q.push(6); q.push(8);
    while(true){
        int u = q.front(); q.pop();
        res.push_back(u);
        if(res.size() == n) break;
        q.push(u * 10 + 6);
        q.push(u * 10 + 8);
    }
    for(int x : res)
        cout << x << ' ';
}
```

c. Một vài ứng dụng của hàng đợi:

Bài toán 3: Tìm số bước di chuyển tối thiểu của quân cờ trên bàn cờ, giữa 2 ô trên mảng 2 chiều...

Bài toán này các bạn đã học trong phần BFS

3. Hàng đợi ưu tiên:

Hàng đợi ưu tiên (priority_queue) khác với hàng đợi thông thường đó là nó không phụ thuộc vào thứ tự bạn thêm phần tử vào hàng đợi mà phần tử ở đỉnh hàng đợi luôn là lớn nhất hoặc nhỏ nhất. Priority_queue trong C++ STL mặc định thì phần tử ở đầu hàng đợi luôn là lớn nhất.

Ví dụ hàng đợi mặc định:

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
int main(){
   priority_queue<int> Q;
   Q.push(3); Q.push(1);
   Q.push(2); Q.push(4);
   cout << Q.top() << endl; // 4
   Q.pop(); // xoa 4
   Q.push(0); Q.push(5);
   cout << Q.top() << endl;
}
```

OUTPUT

```
4 5
```

3. Hàng đợi ưu tiên:

Hàng đợi mà phần tử ở đỉnh là nhỏ nhất

```
#include <bits/stdc++.h>
using namespace std;

using ll = long long;

int main(){
   priority_queue<int, vector<int>,greater<int>> Q;
   Q.push(3);
   Q.push(1);
   Q.push(2);
   Q.push(4);
   cout << Q.top() << endl; // 1
   Q.pop(); // xoa 1
   Q.push(0);
   Q.push(5);
   cout << Q.top() << endl; // 0
}
```

OUTPUT

1 0

3. Hàng đợi ưu tiên:

Ví dụ: Bài toán nối dây: Cho N sợi dây với độ dài khác nhau được lưu trong mảng. Nhiệm vụ của bạn là nối N sợi dây thành một sợi sao cho tổng chi phí nối dây là nhỏ nhất. Biết chi phi nối dây là tổng độ dài của 2 sợi dây được đem nối.

Code

```
using namespace std;
using ll = long long;

int main(){
   int n;cin>>n;
   priority_queue<ll, vector<ll>, greater<ll>> q;
   for(int i = 0; i < n; i++){
      int x; cin >> x;
      q.push(x);
   }
   ll ans = 0;
   while(q.size() != 1){
      ll top1 = q.top(); q.pop();
      ll top2 = q.top(); q.pop();
      ans += top1 + top2;
      q.push(top1 + top2);
   }
   cout<<ans<<"\n";
}
```

INPUT
4 4 3 2 6

OUTPUT
9