Sira Songpolrojjanakul

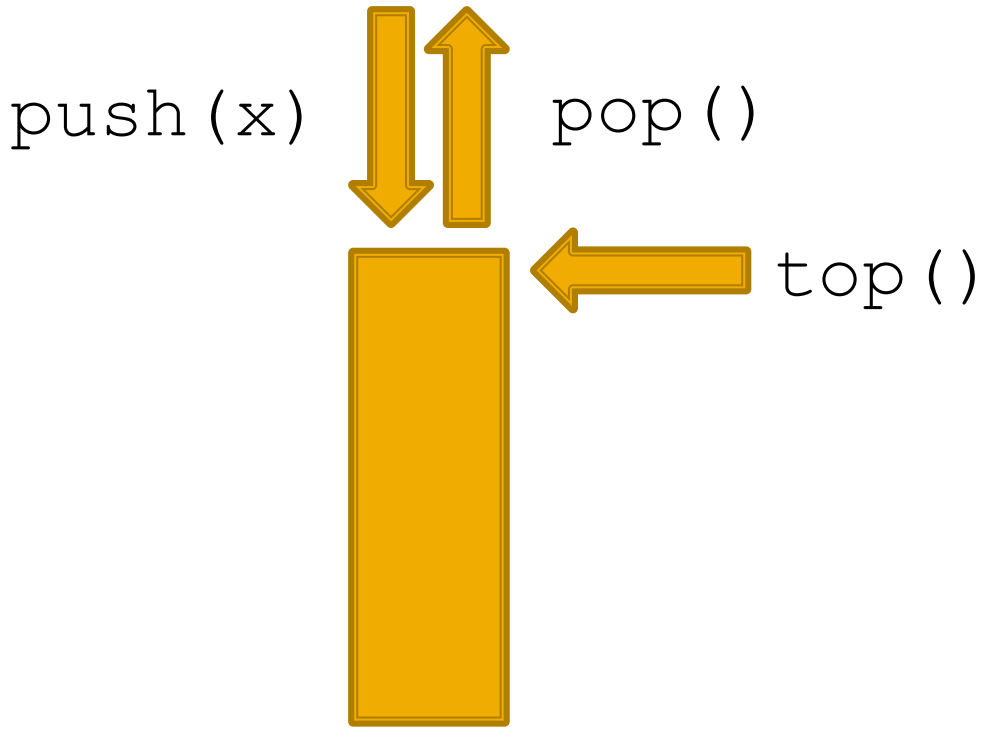
Standard Template Library (STL) Container I

ที่เก็บพื้นฐาน (Container adaptors)

- `stack (LIFO)`
- `queue (FIFO)`
- `priority_queue`

กองซ้อน (stack)

- ข้อมูลตัวสุดท้ายที่ใส่เข้าไป จะออกมาเป็นตัวแรก (Last In First Out; LIFO)



การประกาศ

- #include<stack>
- ประกาศกองซ้อน stack<**type**> **ชื่อ**;
- ตัวอย่างเช่น stack<int> stk; ก็จะได้กองซ้อนที่เก็บตัวเลขชื่อ stk
- ฟังก์ชันที่สำคัญ
- stk.push(x) // ใส่ x ลงไปในกองซ้อน
- stk.pop() // เอาข้อมูลออกจากกองซ้อน
- stk.top() // return ค่าบนสุดของกองซ้อน
- stk.size() // return ขนาดของกองซ้อน
- stk.empty() // return true ถ้ากองซ้อนว่าง

Need to know

- Library ที่ include นี้ เป็น libraly c++
- ต้องเขียนว่า using namespace std; หลังจาก include library ทั้งหมดแล้วด้วย
- เพื่อให้เราสามารถเขียนแค่ว่า stack<int> s1,s2; อะไรแบบนี้ได้
- หากไม่ใส่ไว้ ต้องเขียน `std::` นำหน้าในบางคำสั่งของ c++

Sample code w/o using namespace

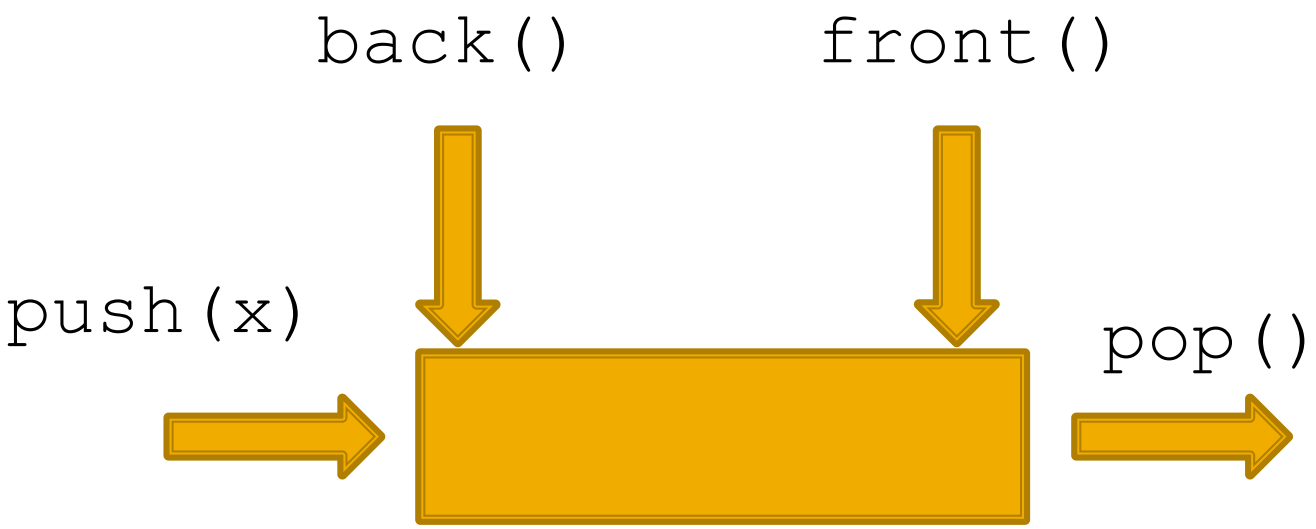
```
// stack::push/pop
#include <iostream>       // std::cout
#include <stack>          // std::stack

int main ()
{
  std::stack<int> mystack;

  for (int i=0; i<5; ++i) mystack.push(i);

  std::cout << "Popping out elements...";
  while (!mystack.empty())
  {
     std::cout << ' ' << mystack.top();
     mystack.pop();
  }
  std::cout << '\n';

  return 0;
}
```

Sample code w using namespace

```
// stack::push/pop
#include <iostream>       // std::cout
#include <stack>          // std::stack
using namespace std;
int main ()
{
  stack<int> mystack;

  for (int i=0; i<5; ++i) mystack.push(i);

  cout << "Popping out elements...";
  while (!mystack.empty())
  {
     cout << ' ' << mystack.top();
     mystack.pop();
  }
  cout << '\n';

  return 0;
}
```

แถวคอย (queue)

- ข้อมูลตัวแรกที่ใส่เข้าไป จะออกมาเป็นตัวแรก (First In First Out; FIFO)



การประกาศ

- #include<queue>
- ประกาศแถวคอย queue<**type**> **ชื่อ**;
- ตัวอย่างเช่น queue<int> Q; ก็จะได้แถวคอยที่เก็บตัวเลขชื่อ Q
- ฟังก์ชันที่สำคัญ
- Q.push(x) // ใส่ x ลงไปในแถวคอย
- Q.pop() // เอาข้อมูลออกจากแถวคอย
- Q.front() // return ค่าที่อยู่หน้าสุดของแถวคอย
- Q.back() // return ค่าที่อยู่ท้ายสุดของแถวคอย
- Q.size() // return ขนาดของแถวคอย
- Q.empty() // return true ถ้าแถวคอยว่าง

ผลลัพธ์คืออะไร

```
// queue::front
#include <iostream>          // std::cout
#include <queue>             // std::queue

int main ()
{
  std::queue<int> myqueue;

  myqueue.push(77);
  myqueue.push(16);

  myqueue.front() -= myqueue.back();     // 77-16=61

  std::cout << "myqueue.front() is now " <<
myqueue.front() << '\n';

  return 0;
}
```

myqueue.front() is now 61

แถวคอยตามลำดับความสำคัญ (priority_queue)

- ข้อมูลที่มีค่า key **มากที่สุด** จะถูกเอาออกมาจากแถวคอย
- ชื่ออื่นของ priority_queue คือ heap

การประกาศ

- #include<queue>
- ประกาศ priority_queue<**type**> **ชื่อ**;
- ตัวอย่างเช่น priority_queue<int> pq;
- ฟังก์ชันที่สำคัญ
- pq.push(x) // ใส่ x ลงไปใน pq
- pq.pop() // เอาข้อมูลออกจาก pq
- pq.top() // return ค่าที่มี key มากสุดของ pq
- pq.size() // return ขนาดของ pq
- pq.empty() // return true ถ้า pq ว่าง

ผลลัพธ์คืออะไร

```
// priority_queue::top
#include <iostream>       // std::cout
#include <queue>          // std::priority_queue

int main ()
{
  std::priority_queue<int> mypq;

  mypq.push(10);
  mypq.push(20);
  mypq.push(15);

  std::cout << "mypq.top() is now " << mypq.top() << '\n';

  return 0;
}
```

mypq.top() is now 20

หยุดตรวจ

- หากต้องการได้ค่า น้อยสุด จาก priority_queue แทน จะทำได้หรือไม่

- **เฉลย**
- เวลา push ให้ใส่ –x แทนที่จะเป็น x
- -pq.top() จะได้ค่าน้อยสุดใน pq ตามต้องการ

Complexity

- ทุกคำสั่ง ของ stack และ queue คือ O(1)
- O(1) หมายถึง ไม่ว่าข้อมูลจะใหญ่แค่ไหน ก็ยังใช้เวลาเท่าเดิม

- push กับ pop ใน priority_queue ใช้เวลาเป็น O(log N)
- O(log N) หมายถึง ใช้คำสั่งประมาณ log N คำสั่งย่อย จึงจะทำได้สำเร็จ เมื่อ N คือขนาดของข้อมูล

datatype ที่ใช้ได้กับ STL

- นอกจาก datatype พื้นฐาน พวก int/char/double แล้ว ยังสามารถใช้ struct กับ STL ได้

struct

ตัวอย่างการประกาศ struct

```
struct student {
   int id;
   char name[20];
}room609[55];   // ประกาศ struct student พร้อมประกาศตัวแปร
student room509[55]; // ประกาศตัวแปรอีก
```

เข้าถึงข้อมูล

```
room609[1].id=45984; // var.data
printf("%s\n",room509[22].name);
```

struct ใช้กับ STL

```
queue<student> Q;
struent temp,xx;
temp.id=55;      temp.name="xx";
Q.push(temp);    // push ด้วยข้อมูลตามที่ประกาศ
xx=Q.front();    // เอา student มารับ
int a=Q.front().id;  // รับเป็นตัวๆ อย่างนี้ก็ได้
```

แต่เราพบว่า ถเประกาศ `prioroty_queue<student> pq;` จะ `compile error!`
เพราะว่าใน `pq` มันต้องการรู้ด้วยว่า **อันไหนน้อยกว่าอันไหน** ไม่งั้นจะรู้ได้อย่างไรว่าตัวไหนมีค่ามากที่สุด
ต่อไปจะกล่าวถึง ชนิดข้อมูลที่เป็นประโยชน์ในการเขียนอีกซักเล็กน้อย

pair

คือ `struct` ที่ประกอบด้วยข้อมูลคู่หนึ่ง ตามชื่อของมัน

ประโยชน์

- ไม่ต้องประกาศ `struct`
- หากนำว่าเรียงจากน้อยไปมาก มันจะเรียงตามตัวหน้าก่อน ถ้าตัวหน้าเท่ากับ ก็จะเรียงตัวหลังจากน้อยไปมาก
- เช่น `{2,3}<{2,5}<{3,1}`

การใช้ pair

การประกาศ

```
pair<type1,type2> name;
```

เช่น

```
pair<int,int> x;
pair<int,string> person;

pair<int,int> p(5,10);
```

กำหนดค่าเริ่มต้นของ p เป็น(5,10)

การใช้ pair (ต่อ)

การเข้าถึงข้อมูล

name.first ข้อมูลตัวแรก

name.second ข้อมูลตัวที่สอง

เช่น x.first=5; string s=person.second;

p=x; // p x กับเป็น pair<int,int> มารับได้

การกำหนดค่า ด้วย make_pair

```
p=make_pair(127,609);
person=make_pair(1,”Mc”);
```

Skgrader: samarnmit

- ของออกมาขาย**ทีละชิ้น**
- มีเข้ามาซื้อของ
- ซื้อของชิ้นนั้นไม่ได้ ออกจากคิวทันที
- สำหรับของแต่ละชิ้น ใครซื้อได้ และได้เงินทอนเท่าไหร่

sol: samarnmit

```
#include<cstdio>
#include<queue>
using namespace std;
#define X first
#define Y second
int main()
{
	queue<pair<int,int> > Q;
	int id,money,price=-1;
	char s[5];
	while(scanf("%s",s),s[0]!='E')
	{
		if(s[0]=='A')
		{
			scanf("%d%d",&id,&money);
			Q.push(make_pair(id,money));
		}
		else	scanf("%d",&price);
```

sol: samarnmit (cont.)

```
if(price!=-1)
      while(!Q.empty())
      {
            if(Q.front.Y()>=price)
            {
                  printf("%d %d\n",Q.front().X
                                    ,Q.front().Y-price);
                  Q.pop();    price=-1;
                  break;
            }
            Q.pop();
      }

}
}
```

Did you know

Queue มันเข้าได้ทางเดียว (ท้ายคิว) ออกได้ทางเดียว (หน้าคิว)
มันมีคิวที่เข้าและออก ได้ทั้งทางหน้าและหลัง
เรียกว่า deque (เดค)
การประกาศใน STL คือ deque<type> name;
อยู่ใน library เดียวกับ queue
ฟังก์ชันในการใส่ข้อมูลกับเอาข้อมูลออกจะเป็น
push_front(x),push_back(x)
pop_front(),pop_back()

Application

Queue application

- จำลองการทำงาน (simulation)
- ไถลหน้าต่าง (sliding window)
- ค้นหาทางกว้าง (Breath First Search: BFS)

จำลองการทำงาน

- ก็คือ ทำตามคำบอกนั่นเอง
- ตัวอย่างเช่น ข้อซื้อของสมานมิตร เมื่อครู่นี้

UVa – 10935 Throwing cards away I

- มีไพ่อยู่ n ใบ หมายเลข 1 – n ไพ่หมายเลข 1 อยู่บนสุด เรียงลงมาเรื่อยๆ
- โยนไพ่ใบบนสุดทิ้งไป และเอาใบถัดไปไว้ใต้กอง
- ทำเรื่อยๆ จนเหลือใบเดียว
- ให้บอกมาว่า ไพ่ที่ถูกโยนทิ้ง คือไพ่อะไรบ้าง ตามลำดับ
- และ ระบุด้วยว่า ใบสุดท้ายที่เหลือ คืออะไร
- ! ระวัง ช่องว่างเกิน

ไถลหน้าต่าง

- เป็นกลวิธีในการแก้ปัญหาอย่างหนึ่ง
- มีตัวชี้ สองตัว ชี้หัวกับท้าย แล้วขยับท้ายไปเรื่อยๆ จนกว่าจะขัดเงื่อนไข จึงขยับตัวหน้า
- อาจใช้ คิว (หรือ เดค) มาใช้แทนตัวชี้ หัวท้ายก็ได้

UVa – 1121 Subsequence

- มีจำนวนเต็มบวกอยู่ N ตัว
- ช่วงติดกันที่สั้นที่สุด ที่บวกกันได้ >=S มีความยาวเท่าใด (ไม่มี ตอบ 0)

- เฉลย
- ก็ใส่เลขลงคิวไปเรื่อยๆ
- ถ้าบวกได้ S ก็เช็ค size ว่าได้ความยาวน้อยลงหรือไม่
- ถ้าเกิน S ก็ pop ตัวหน้าจนกว่า ผลรวมเลขในคิว <S
- ทุกขณะทำที่ผลรวมในคิว >=S ก็เช็คขนาดด้วย

ค้นหาแนวกว้าง

- เป็นหารค้นหาแบบหนึ่ง โดยใช้ คิว ประกอบการทำงาน
- จะกล่าวถึงในโอกาส ถัดๆ ไป

Stack application

- แปลงเลขฐาน
- ตรวจสอบวงเล็บ
- prefix /infix/postfix
- หอคอยแห่งฮานอย (Tower of Hanoi)
- ค้นหาทางลึก (Depth First Search: DFS)

แปลงเลขฐาน

- จำวิธีแปลงเลขฐาน ตอน ม.1 ได้หรือไม่
- จะแปลง N (ฐาน 10) เป็นเลขฐาน b อย่างไร
- คำตอบ
- หาร N ด้วย b ไปเรื่อยๆ (เขียนเศษที่ได้ด้วย) หารจนผลลัพธ์สุดท้ายมีค่าน้อยกว่า b แล้วอ่านเศษย้อนขึ้น จะได้เลขตามต้องการ

ตัวอย่าง แปลงเลขฐาน

- แปลง 127 เป็นเลข ฐาน 5
- 5) 127
- 5) 25 เศษ 2
- 5) 5 เศษ 0
- 1 เศษ 0
- นั่นคือ $127 = 1002_5$
- วิธีการย้อนเลขง่ายๆ คือ พอหารแล้วก็ push ลง stack
- เสร็จแล้วก็ pop ออกมาก็ได้เลขตามต้องการ

UVa – 11185 Ternary

- รับเลขฐาน 10 แปลงเป็นเลขฐาน 3

หยุดตรวจ

- รับเลขฐาน 8 แปลงเป็นเลขฐาน 16 ทำไงดี

- เฉลย
- `scanf("%o",&N);`
- `printf("%x",N);`

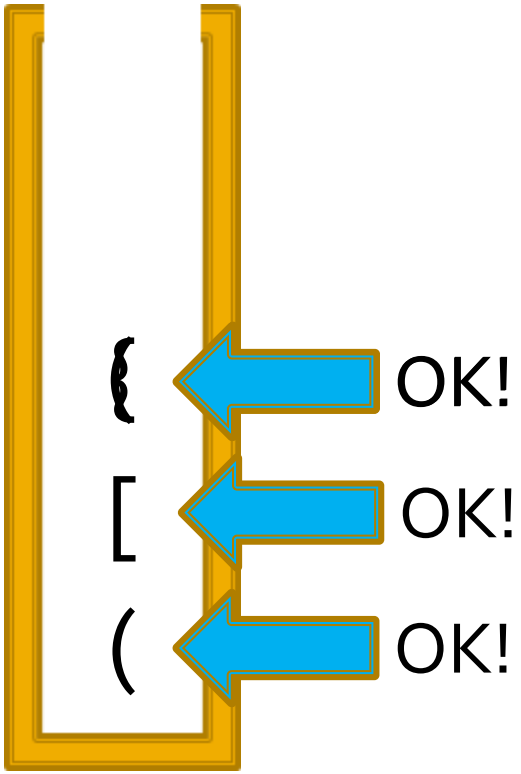
ตรวจสอบวงเล็บ

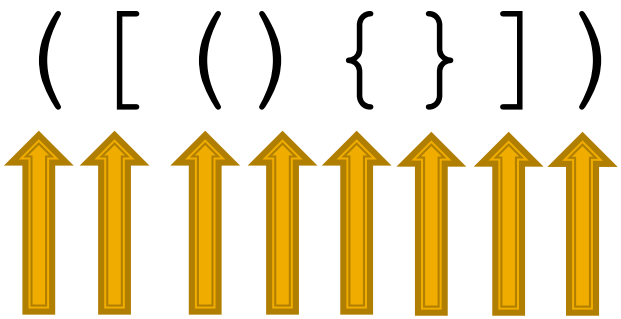
- ตรวจสอบว่า วงเล็บที่ให้มา เขียนถูกต้องตามรูปแบบหรือไม่
- เช่น
- (())() ถูก
- ()()() ถูก
- ()) ผิด
- [()] ถูก
- [(]) ผิด

ตรวจสอบวงเล็บ (วิธีทำ)

- ใช้ สแตค
- เมื่อเจอ**วงเล็บเปิด** push ตัวนั้นลง stack
- เมื่อเจอ**วงเล็บปิด** ให้ดู top of stack ว่าจับคู่ถูกต้องหรือไม่ ถ้าไม่แสดงว่า วงเล็บไม่ถูกต้อง

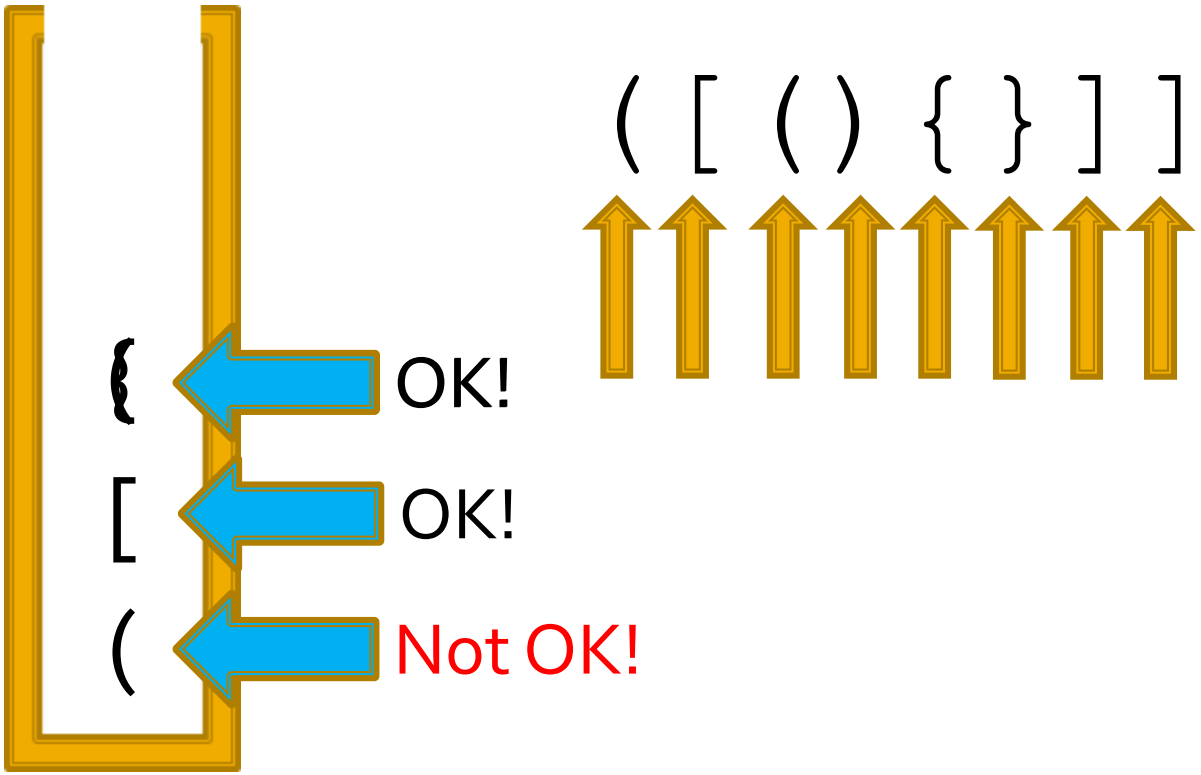
ตัวอย่าง ตรวจสอบวงเล็บ





OK!

ตัวอย่าง ตรวจสอบวงเล็บ 2



Not OK!

หยุดตรวจ

- คิดว่า วิธีการที่ให้ ถูกต้องแล้ว หรือไม่

- เฉลย: ผิด

ตรวจสอบวงเล็บ (วิธีทำ ฉบับเต็ม)

- ใช้ สแตค
- เมื่อเจอ**วงเล็บเปิด** push ตัวนั้นลง stack
- เมื่อเจอ**วงเล็บปิด** ให้ดู top of stack ว่าจับคู่ถูกต้องหรือไม่ ถ้าไม่แสดงว่า วงเล็บไม่ถูกต้อง
- ระหว่างทำ ถ้าเราต้องการ pop แต่ stack มันว่าง แสดงว่าวงเล็บไม่ถูกต้อง (วงเล็บปิดเกิน)
- เมื่อจบสตริง stack ต้องว่าพอดี ถ้าไม่ว่าง แสดงว่าวงเล็บไม่ถูกต้อง (วงเล็บเปิดเกิน)

Skgrader: toi_threeparen

- ให้สตริงวงเล็บมา ถามว่า ถูกต้องตามรูปแบบการเขียนวงเล็บหรือไม่

คำถาม

- ถ้าวงเล็บที่เช็คมีแค่แบบเดียว เช่น ()(()) จะมีวิธีอื่นที่ใช้ space เป็น O(1) หรือไม่
- คำตอบ **มี**
- ใช้จำนวนเต็มหนึ่งตัว ถ้าเป็น (ให้ **+1** / เป็น) ให้ **-1** ระหว่างทำ ห้ามน้อยกว่า **0** และจบเมื่อจบสตริงต้องเป็น **0**
- ก็คือ แทนที่จะเขียนสแตกเต็มๆ ก็เก็บแต่ขนาดมันก็พอ เพราะในสแตกมีแต่)

- ตรวจว่า จับกลุ่มวงเล็บในการคูณเลขถูกหรือไม่ เช่น (abc(af)g) ถูก / (abc(d) ผิด (วงเล็บไม่ครบ) / c()d ถูก จะทำอย่างไร
- ไม่ต้องสนใจตัวอักษร

Prefix Infix Postfix

- นิพจน์เติมใน (Infix) คือ รูปแบบการเขียนนิพจน์ L op R ซึ่งเป็นรูปแบบที่คนเขียนทั่วไป
- นิพจน์เติมหน้า (Prefix / Polish notation) คือ รูปแบบการเขียนนิพจน์ op L R
- นิพจน์เติมหลัง (Postfix / Reverse Polish notation) คือ รูปแบบการเขียนนิพจน์ L R op

ตัวอย่าง Prefix Infix Postfix

- Infix: 1 + 2 * 7

- Prefix: 1 + 2 * 7 → 1 + (2 * 7)
- → 1 + (* 2 7)
- → + 1 (* 2 7) → + 1 * 2 7

- Postfix: 1 + 2 * 7 → 1 + (2 * 7)
- → 1 + (2 7 *)
- → 1 (2 7 *) + → 1 2 7 * +

แล้วจำคำนวณค่าของนิพจน์อย่างไร

การคำนวณ Postfix

- ใช้สแตกช่วยในการคำนวณ
- สแตกเก็บแต่ ค่า เท่านั้น
- เมื่อเจอ ตัวดำเนินการ ให้ pop stack 2 ที โดย**ตัวที่สองเป็นตัวตั้ง** แล้วก็ดำเนินการตามเครื่องหมาย แล้ว push ผลลัพธ์ลง stack

ตัวอย่าง การคำนวณ Postfix

Postfix	Stack (ท้าย → หัว)	คำอธิบาย
5 4 * 6 3 / +	5 4	ใส่ลงสแตกไปเรื่อยๆ
5 4 * 6 3 / +	20	4 * 5 = 20
5 4 * 6 3 / +	20 6 3	
5 4 * 6 3 / +	20 2	6 / 3 = 2
5 4 * 6 3 / +	22	20 + 2 = 22

การคำนวณ Prefix

- ใช้สแตกช่วยในการคำนวณ
- สแตกเก็บแต่ ค่า เท่านั้น
- อ่านสตริง จาก ท้าย มา หน้า
- เมื่อเจอ ตัวดำเนินการ ให้ pop stack 2 ที โดย**ตัวแรกเป็นตัวตั้ง** แล้วก็ดำเนินการตามเครื่องหมาย แล้ว push ผลลัพธ์ลง stack (คล้ายกับการคำนวณ postfix)

ตัวอย่าง การคำนวณ Prefix

Prefix: + * 5 4 / 6 3

	Stack (ท้าย → หัว)	คำอธิบาย
3 6 / 4 5 * +	3 6	ใส่ลงสแตกไปเรื่อยๆ
3 6 / 4 5 * +	2	6 / 3 = 2
3 6 / 4 5 * +	2 4 5	
3 6 / 4 5 * +	2 20	5 * 4 = 20
3 6 / 4 5 * +	22	20 + 2 = 22

Postfix: 5 4 * 6 3 / +

สังเกตว่า reverse prefix ได้ postfix แต่เป็น R L op

การคำนวณ Infix

- คิดตรงๆ ไม่ได้
- เนื่องจากมันมีศักดิ์เครื่องหมาย ทำให้ไม่ได้คำนวณเรียงจากซ้ายไปขวา (หรือขวาไปซ้าย) แบบ นิพจน์เติมหลัง หรือ นิพจน์เติมหน้า
- ต้องแปลงเป็น postfix หรือ prefix ก่อน

แปลง Infix เป็น Postfix

- อ่านนิพจน์เติมใน เข้ามาเรื่อยๆ
- ถ้าเป็นตัวถูกดำเนินการ (operand) ให้ใส่ในผลลัพธ์ทันที
- ถ้าเป็น (ให้ push ลงสแตก
- ถ้าเป็นตัวดำเนินการ operator ให้พิจารณา ดังนี้
  - ถ้า ตัวดำเนินการนี้ มี**ศักดิ์ไม่ต่ำกว่า** top of stack ให้ push op นั้นลงไป
  - มิเช่นนั้น pop ออกมา จนกว่าจะเจอ ศักดิ์ที่ตัวนี้เอาไปวางต่อได้ (ศักดิ์พอดีกันหรือต่ำกว่า) (วงเล็บเปิด ถือว่าศักดิ์ต่ำมาก)
- ถ้าเป็น) ให้ pop เอา operator ทั้งหมดมาใส่ในผลลัพธ์ จนถึงวงเล็บเปิดตัวแรกที่เจอในสแตก (pop '(' ออกด้วย)
- เมื่อจบนิพจน์เติมในแล้ว สแตกยังไม่ว่าง ให้ pop operator ทั้งหมดมาใส่ในผลลัพธ์

เกี่ยวกับ ศักดิ์ ในการแปลง Infix เป็น Postfix

- (น้อยจัด เพื่อให้ใครๆ ก็มาต่อได้
- + - มีค่า 1
- * / มีค่า 2 (สูงกว่า + -)
- ^ มีค่า 4 (สูงกว่า * /)

- ต่อไปจะแสดงการแปลง $2^{3+1} \times (4 + 1 \times 3) \div 2$ หรือก็คือ 2^(3+1)*(4+1*3)/2 ให้ดู

ตัวอย่าง การแปลง Infix เป็น Postfix

- 2^(3+1)*(4+1*3)/2
- Stack:
- Postfix:

ตัวอย่าง การแปลง Infix เป็น Postfix

- 2^(3+1)*(4+1*3)/2
- Stack:
- Postfix: 2

ตัวอย่าง การแปลง Infix เป็น Postfix

- 2^(3+1)*(4+1*3)/2
- Stack: ^
- Postfix: 2

ตัวอย่าง การแปลง Infix เป็น Postfix

- 2^(3+1)*(4+1*3)/2
- Stack: ^ (
- Postfix: 2

ตัวอย่าง การแปลง Infix เป็น Postfix

- 2^(3+1)*(4+1*3)/2
- Stack: ^ (
- Postfix: 2 3

ตัวอย่าง การแปลง Infix เป็น Postfix

- 2^(3+1)*(4+1*3)/2
- Stack: ^ (+
- Postfix: 2 3

ตัวอย่าง การแปลง Infix เป็น Postfix

- 2^(3+1)*(4+1*3)/2
- Stack: ^ (+
- Postfix: 2 3 1

ตัวอย่าง การแปลง Infix เป็น Postfix

- 2^(3+1)*(4+1*3)/2
- Stack: ^
- Postfix: 2 3 1 +

ตัวอย่าง การแปลง Infix เป็น Postfix

- 2^(3+1)*(4+1*3)/2
- Stack: *
- Postfix: 2 3 1 + ^

ตัวอย่าง การแปลง Infix เป็น Postfix

- 2^(3+1)*(4+1*3)/2
- Stack: * (
- Postfix: 2 3 1 + ^

ตัวอย่าง การแปลง Infix เป็น Postfix

- 2^(3+1)*(4+1*3)/2
- Stack: * (
- Postfix: 2 3 1 + ^ 4

ตัวอย่าง การแปลง Infix เป็น Postfix

- 2^(3+1)*(4+1*3)/2
- Stack: * (+
- Postfix: 2 3 1 + ^ 4

ตัวอย่าง การแปลง Infix เป็น Postfix

- 2^(3+1)*(4+1*3)/2
- Stack: * (+
- Postfix: 2 3 1 + ^ 4 1

ตัวอย่าง การแปลง Infix เป็น Postfix

- 2^(3+1)*(4+1*3)/2
- Stack: * (+ *
- Postfix: 2 3 1 + ^ 4 1

ตัวอย่าง การแปลง Infix เป็น Postfix

- 2^(3+1)*(4+1*3)/2
- Stack: * (+ *
- Postfix: 2 3 1 + ^ 4 1 3

ตัวอย่าง การแปลง Infix เป็น Postfix

- 2^(3+1)*(4+1*3)/2
- Stack: *
- Postfix: 2 3 1 + ^ 4 1 3 * +

ตัวอย่าง การแปลง Infix เป็น Postfix

- 2^(3+1)*(4+1*3)/2
- Stack: * /
- Postfix: 2 3 1 + ^ 4 1 3 * +

ตัวอย่าง การแปลง Infix เป็น Postfix

- 2^(3+1)*(4+1*3)/2
- Stack: * /
- Postfix: 2 3 1 + ^ 4 1 3 * + 2

ตัวอย่าง การแปลง Infix เป็น Postfix

- 2^(3+1)*(4+1*3)/2
- Stack:
- Postfix: 2 3 1 + ^ 4 1 3 * + 2 / *

แปลง Postfix เป็น Infix

- ทำเหมือนคำนวณค่า postfix
- เพียงแต่แทนที่จะใส่ค่าเป็น ผลลัพธ์ที่ได้จากการดำเนินการ
- ก็ใส่ (A op B) แทน

ตัวอย่าง การแปลง Postfix เป็น Infix

- 2 5 * 3 +
- Stack: (bottom → top)
-

ตัวอย่าง การแปลง Postfix เป็น Infix

- 2 5 * 3 +
- Stack: (bottom → top)
- 2

ตัวอย่าง การแปลง Postfix เป็น Infix

- 2 5 * 3 +
- Stack: (bottom → top)
- 2 5

ตัวอย่าง การแปลง Postfix เป็น Infix

- 2 5 * 3 +
- Stack: (bottom → top)
- (2*5)

ตัวอย่าง การแปลง Postfix เป็น Infix

- 2 5 * 3 +
- Stack: (bottom → top)
- (2*5) 3

ตัวอย่าง การแปลง Postfix เป็น Infix

- 2 5 * 3 +
- Stack: (bottom → top)
- ((2*5)+3)

ตัวอย่าง การแปลง Postfix เป็น Infix

- 2 5 * 3 +
- Infix:
- ((2*5)+3)

แปลง Infix เป็น Prefix

- รับนิพจน์เติมใน มาก่อน `(A+B^C)*D+E^5`
- กลับสตริง `5^E+D*)C^B+A(`
- เปลี่ยนวงเล็บ) → (และ (→) `5^E+D*(C^B+A)`
- แปลงเป็น postfix `5E^DCB^A+*+`
- กลับ postfix ที่ได้ ก็จะเป็น prefix `+*+A^BCD^E5`

แปลง Prefix เป็น Infix

- ทำเหมือนคำนวณค่า prefix
- เพียงแต่แทนที่จะใส่ค่าเป็น ผลลัพธ์ที่ได้จากการดำเนินการ
- ก็ใส่ (A op B) แทน

ตัวอย่าง การแปลง Prefix เป็น Infix

- + * A B / C D
- Stack: (bottom → top)
-

ตัวอย่าง การแปลง Prefix เป็น Infix

- + * A B / C D [convert]
- Stack: (bottom → top)
-

ตัวอย่าง การแปลง Prefix เป็น Infix

- D C / B A * +
- Stack: (bottom → top)
-

ตัวอย่าง การแปลง Prefix เป็น Infix

- D C / B A * +
- Stack: (bottom → top)
- D

ตัวอย่าง การแปลง Prefix เป็น Infix

- D C / B A * +
- Stack: (bottom → top)
- D C

ตัวอย่าง การแปลง Prefix เป็น Infix

- D C / B A * +
- Stack: (bottom → top)
- (C/D)

ตัวอย่าง การแปลง Prefix เป็น Infix

- D C / B A * +
- Stack: (bottom → top)
- (C/D) B

ตัวอย่าง การแปลง Prefix เป็น Infix

- D C / B A * +
- Stack: (bottom → top)
- (C/D) B A

ตัวอย่าง การแปลง Prefix เป็น Infix

- D C / B A * +
- Stack: (bottom → top)
- (C/D) (A*B)

ตัวอย่าง การแปลง Prefix เป็น Infix

- D C / B A * +
- Stack: (bottom → top)
- (A*B)+(C/D)

ตัวอย่าง การแปลง Prefix เป็น Infix

- D C / B A * +
- Infix:
- (A*B)+(C/D)

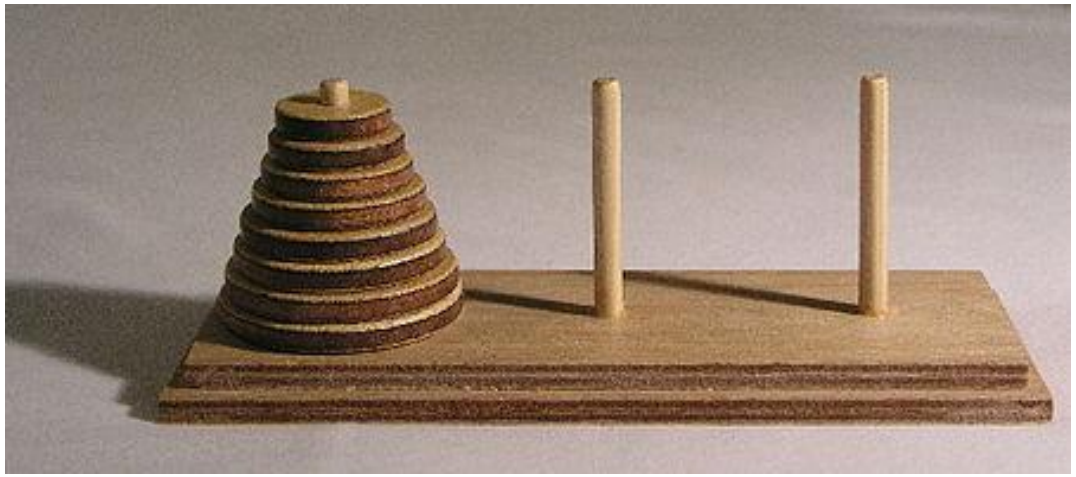
คำถาม

- แปลง postfix → prefix อย่างไร
- ทำเหมือนคำนวณค่า postfix แต่แทนที่จะใส่ผลลัพธ์ก็ใส่นิพจน์เติมหน้าแทน

- แปลง prefix → postfix อย่างไร
- ทำเหมือนคำนวณค่า prefix แต่แทนที่จะใส่ผลลัพธ์ก็ใส่นิพจน์เติมหลังแทน

หอคอยแห่งฮานอย

- มีเสาอยู่ 3 เสา A B C และมีแผ่นไม้ขนาดต่างๆ n แผ่น เริ่มต้นอยู่เสา A
- ต้องการย้ายแผ่นไม้ทั้ง n แผ่น จากเสา A ไปยังเสา C
- อย่างไรก็ดี ระหว่างย้าย ต้องย้ายทีละแผ่น และ แผ่นเล็กกว่าต้องอยู่เหนือแผ่นใหญ่กว่า
- ทำอย่างไร



วิธีการ

- ย้าย แผ่นไม้ $n-1$ แผ่น จากเสาเริ่มต้น (A) ไปยังเสาพักก่อน (B) โดยใช้เสาเป้าหมาย (C) ในการช่วยย้ายด้วย
- ย้ายแผ่น n จากเสาเริ่ม ไปยังเสาเป้าหมาย
- ย้ายแผ่นไม้ $n-1$ แผ่นที่อยู่ที่เสาพัก ไปยังเสาเป้าหมาย โดยใช้เสาเริ่มต้นในการช่วยย้ายด้วย

ข้อสังเกต

- จากการวิธีการเมื่อครู่ สังเกตว่าจะต้องทำลงไปเรื่อยๆ
- นั่นคือต้องใช้ฟังก์ชันเรียกตัวเอง เรื่อยๆ
- นั่นคือ ลดปัญหาจาก n สู่ 1
- เขียน Prototype ของ Tower of Hanoi ได้ว่า
- solve(plate ,source ,dest ,reserve)
- มี Base case: plate = 1 → move source to dest
- แล้วที่เหลือ เขียนอย่างไร

ฟังก์ชันแก้ปัญหา หอคอยแห่งฮานอย

- solve(plate – 1 ,source ,**reserve** ,dest)
  - พยายามย้าย n – 1 แผ่น ด้านบน ไปยังเสาพัก
- solve(1 ,source ,**dest** ,reserve)
  - แผ่นเดียวที่เหลือ ย้ายไปยังเป้าหมาย
- solve(plate – 1 , reserve ,**dest** ,source)
  - ย้าย n – 1 จากเสาพัก ไปยังเสาเป้าหมาย
- ต่อไปจะแสดง โค้ดตัวอย่าง ของฟังก์ชันนี้

Code ToH

```
void solve(int plate ,char source ,char dest ,char
reserve)
{
      if(plate==1)
            printf("move %c to %c\n",source,dest);
      else
      {
            solve(plate-1 ,source ,reserve ,dest);
            solve(1 ,source ,dest ,reserve);
            solve(plate-1 , reserve ,dest ,source);
      }
}
```

ค้นหาทางลึก

- ใช้สแตก ร่วมกับการค้นหาข้อมูล
- จะอธิบายในโอกาส ต่อๆ ไป

คำถาม

Programming Practice

- UVa – 12100 Printer Queue
- UVa – 540 Team Queue
- UVa – 127 Rails
- UVa – 1062 Containers

จบเนื้อหา

สวัสดี