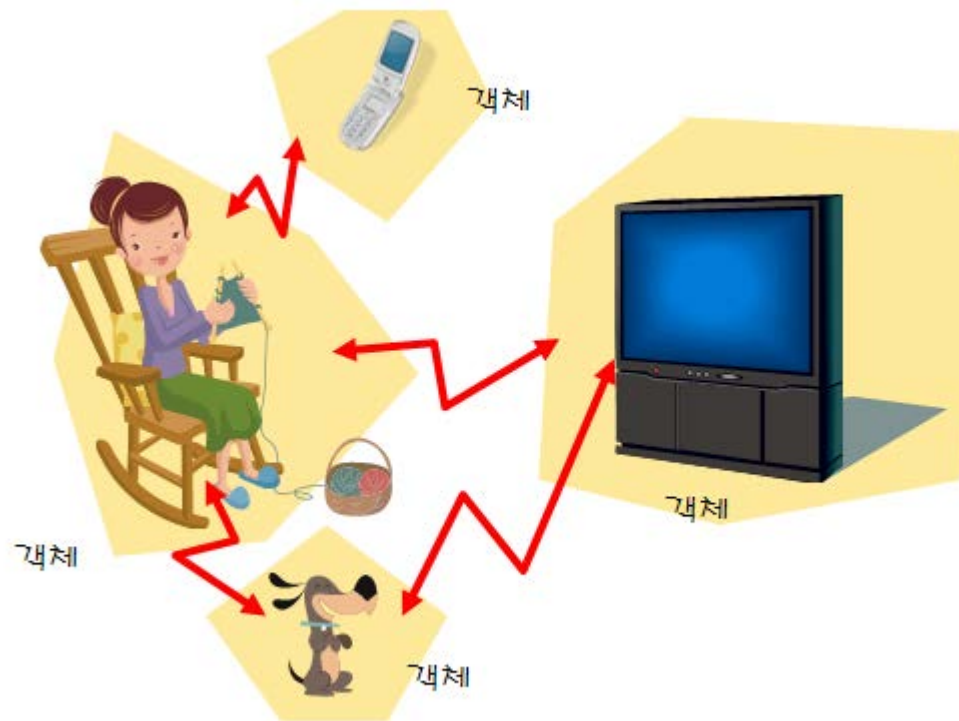




Power C++

제17장 템플릿





이번 장에서 학습할 내용



- 함수 템플릿
- 클래스 템플릿
- 스택 예제

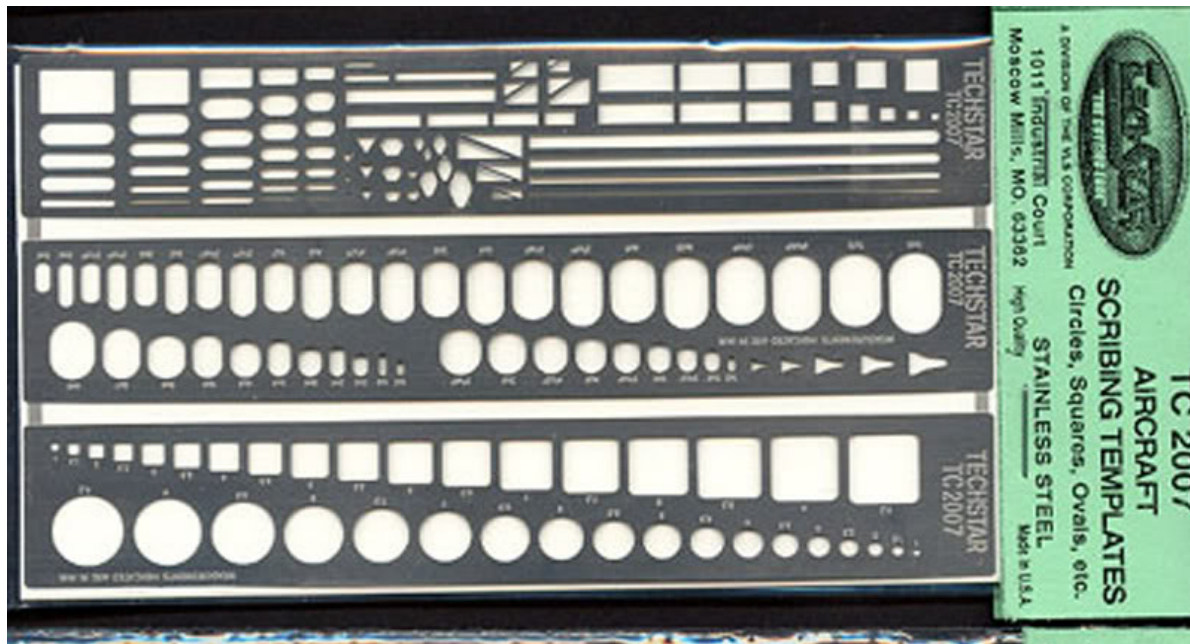
일반적인
하나의 코드로
다양한
자료형을
처리하는
기법을
살펴봅시다.





템플릿이란?

- 템플릿(template): 물건을 만들 때 사용되는 틀이나 모형을 의미
- 함수 템플릿(function template): 함수를 찍어내기 위한 형틀





함수 get_max()

```
int get_max(int x, int y)
{
    if( x > y ) return x;
    else return y;
}
```

만약 float
값중에서
최대값을
구하는 함수가
필요하다면?





함수 get_max()

```
float get_max(float x, float y)
{
    if( x > y ) return x;
    else return y;
}
```

핵심적인
내용은 같고
매개 변수의
타입만
달라진다.





일반화 프로그래밍

- 일반화 프로그래밍(generic programming): 일반적인 코드를 작성하고 이 코드를 정수나 문자열과 같은 다양한 타입의 객체에 대하여 재사용하는 프로그래밍 기법



int 버전으로 필요하시다구요.

템플릿 함수

```
____ get_max(____x , ____ y)
{
    if( x > y) return x;
    else return y;
}
```



```
int get_max(int x , int y)
{
    if( x > y) return x;
    else return y;
}
```



get_max()

```
template<typename T>  
T get_max(T x, T y)  
{  
    if( x > y ) return x;  
    else return y;  
}
```

자료형이
변수처럼
표기되어
있음을 알 수
있다





템플릿 함수의 사용

`get_max(1, 3)` 으로 호출

```
template <typename T>
T get_max(T x, T y)
{
    if(x > y) return x;
    else return y;
}
```

```
int get_max(int x, int y)
{
    if(x > y) return x;
    else return y;
}
```

`get_max(1.8, 3.7)` 으로 호출

```
double get_max(double x, double y)
{
    if(x > y) return x;
    else return y;
}
```




예제

get_max.cpp

```
#include <iostream>
using namespace std;

template <typename T>
T get_max(T x, T y)
{
    if(x > y)    return x;
    else return y;
}

int main()
{
    // 아래의 문장은 정수 버전 get_max()를 호출한다.
    cout << get_max(1, 3) << endl;

    // 아래의 문장은 실수 버전 get_max()를 호출한다.
    cout << get_max(1.2, 3.9) << endl;

    return 0;
}
```



실행 결과

실행 결과

3

3.9

계속하려면 아무 키나 누르십시오 ...



템플릿 함수의 특수화

```
template <typename T>                // 함수 템플릿으로 정의
void print_array(T[] a, int n)
{
    for(int i=0; i<n; i++)
        cout << a[i] << " ";
    cout << endl;
}

template <>                            // 템플릿 특수화
void print_array(char[] a, int n)      // 매개 변수가 char인 경우에는 이 함수가
    호출된다.
{
    cout << a << endl;
}
```



함수 템플릿과 함수 중복

swap_values.cpp

```
#include <iostream>
using namespace std;
```

템플릿 함수

```
template <typename T>
void swap_values(T& x, T& y)
{
    T temp;
    temp = x;
    x = y;
    y = temp;
}
```

함수 중복

```
void swap_values(char* s1, char* s2)
{
    int len;

    len = (strlen(s1) >= strlen(s2)) ? strlen(s1) : strlen(s2);
    char* tmp = new char[len + 1];

    strcpy(tmp, s1);
    strcpy(s1, s2);
    strcpy(s2, tmp);
    delete[] tmp;
}
```



함수 템플릿과 함수 중복

```
int main()
{
    int x=100, y=200;
    swap_values(x, y);           // x, y가 모두 int 타입- OK!
    cout << x << " " << y << endl;

    char s1[100]="This is a first string";
    char s2[100]="This is a second string";
    swap_values(s1, s2);        // s1, s2가 모두 배열 - 오버로딩 함수 호출
    cout << s1<< " " << s2<< endl;
    return 0;
}
```

실행 결과

200 100

This is a second string This is a first string

계속하려면 아무 키나 누르십시오 . . .



두개의 타입 매개 변수

```
template<typename T1, typename T2>  
void copy(T1 a1[], T2 a2[], int n)  
{  
    for (int i = 0; i < n; ++i)  
        a1[i] = a2[i];  
}
```



중간 점검 문제

1. 변수의 절대값을 구하는 `int abs(int x)`를 템플릿 함수로 정의하여 보자.
2. 두수의 합을 계산하는 `int add(int a, int b)`를 템플릿 함수로 구현하여 보자.
3. `displayArray()`라는 함수는 배열을 매개 변수로 받아서 반복 루프를 사용하여서 배열의 원소를 화면에 출력한다. 어떤 타입의 배열도 처리할 수 있도록 함수 템플릿으로 정의하여 보라.



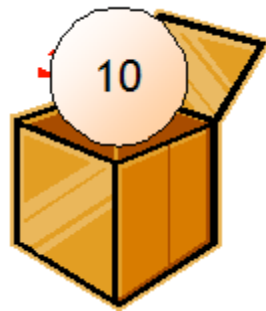


클래스 템플릿

- 클래스 템플릿(class template): 클래스를 찍어내는 틀(template)

```
template <typename 타입이름, ...> class 클래스이름  
{  
    ...  
}
```

- 예제: 하나의 값을 저장하고 있는 박스



정수를 저장하는 상자



예제

```
class Box {  
    int data;  
public:  
    Box() { }  
    void set(int value) {  
        data = value;  
    }  
    int get() {  
        return data;  
    }  
};  
  
int main()  
{  
    Box box;  
    box.set(100);  
    cout << box.get() << endl;  
    return 0;  
}
```



실행 결과

실행 결과

100

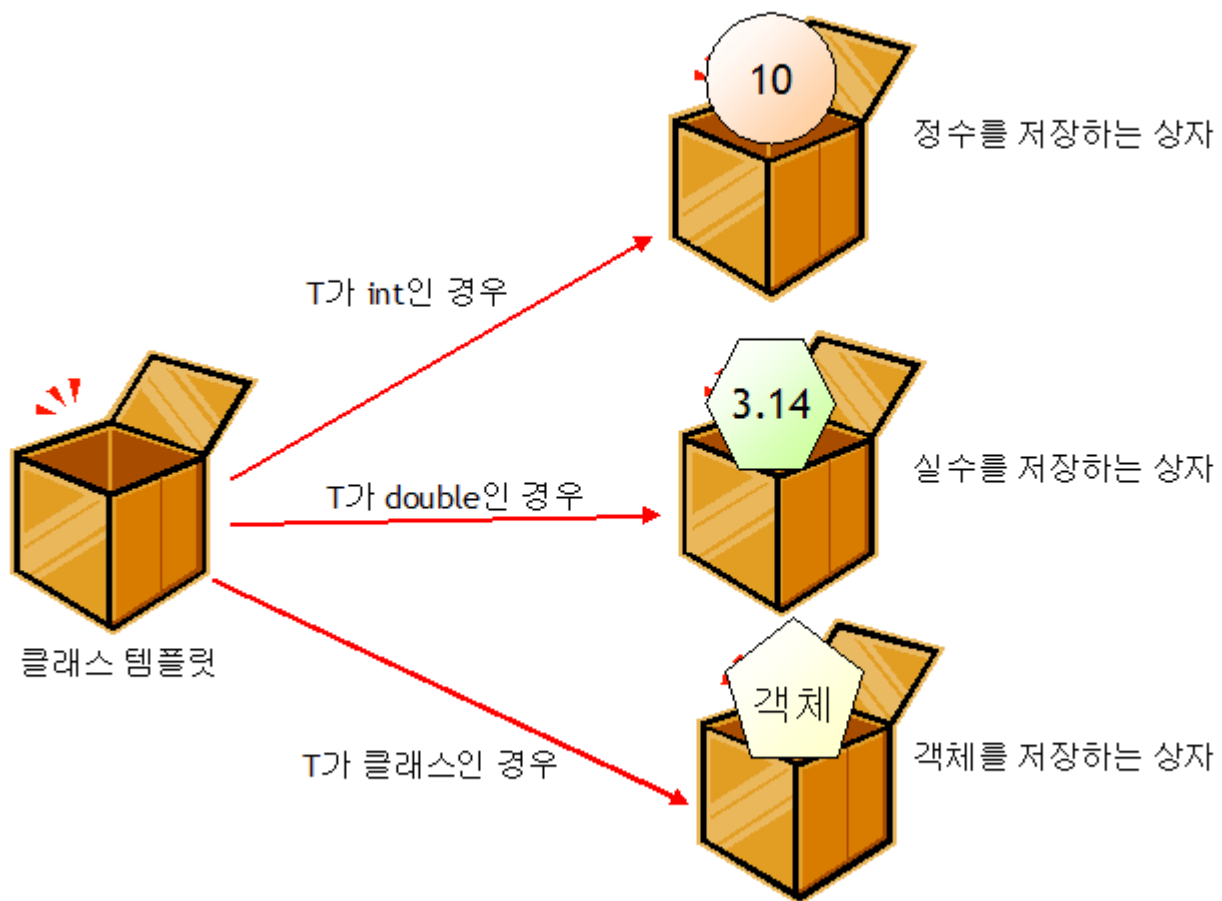
계속하려면 아무 키나 누르십시오 . . .

클래스
템플릿으로
만들어 보자.





클래스 템플릿 버전





클래스 템플릿 정의

```
template <typename T>  
class 클래스이름  
{  
...// T를 어디서든지 사용할 수 있다.  
}
```



예제

```
#include <iostream>
using namespace std;

template <typename T>
class Box {
    T data; // T는 타입(type)을 나타낸다.
public:
    Box() { }
    void set(T value) {
        data = value;
    }
    T get() {
        return data;
    }
};
```

클래스 템플릿



```
int main()
```

```
{
```

```
Box<int> box;
```

```
box.set(100);
```

```
cout << box.get() <<endl;
```

정수 버전 클래스

```
Box<double> box1;
```

```
box1.set(3.141592);
```

```
cout << box1.get() <<endl;
```

실수 버전 클래스

```
return 0;
```

```
}
```

실행 결과

100

3.14159

계속하려면 아무 키나 누르십시오 . . .



클래스 외부에 정의

```
template <typename T>
class Box {
    T data; // T는 타입(type)을 나타낸다.
public:
    Box();
    void set(T value);
    T get();
};
```

```
template <typename T>
Box<T>::Box() {
}
```

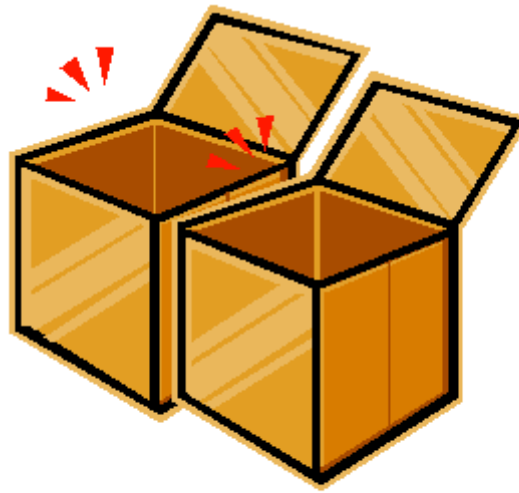
```
template <typename T>
void Box<T>::set(T value) {
    data = value;
}
```

```
template <typename T>
T Box<T>::get() {
    return data;
}
```



두 개의 타입 매개 변수

- 두 개의 데이터를 저장하는 클래스 Box2



Box2 클래스 템플릿



예제

```
#include <iostream>
using namespace std;

template <typename T1, typename T2>
class Box2 {

    T1 first_data; // T1은 타입(type)을 나타낸다.
    T2 second_data; // T2는 타입(type)을 나타낸다.

public:
    Box2() { }
    T1 get_first();
    T2 get_second();
    void set_first(T1 value) {
        first_data = value;
    }
    void set_second(T2 value) {
        second_data = value;
    }
};
```

두개의 타입 매개
변수를 가지는 클
래스 템플릿



예제

```
template <typename T1, typename T2>
T1 Box2<T1, T2>::get_first() {
    return first_data;
}
template <typename T1, typename T2>
T2 Box2<T1, T2>::get_second() {
    return second_data;
}
int main()
{
    Box2<int, double> b;
    b.set_first(10);
    b.set_second(3.14);
    cout << "(" << b.get_first() << ", " << b.get_second() << ")" << endl;
    return 0;
}
```

실행 결과

(10, 3.14)

계속하려면 아무 키나 누르십시오 . . .



중간 점검 문제

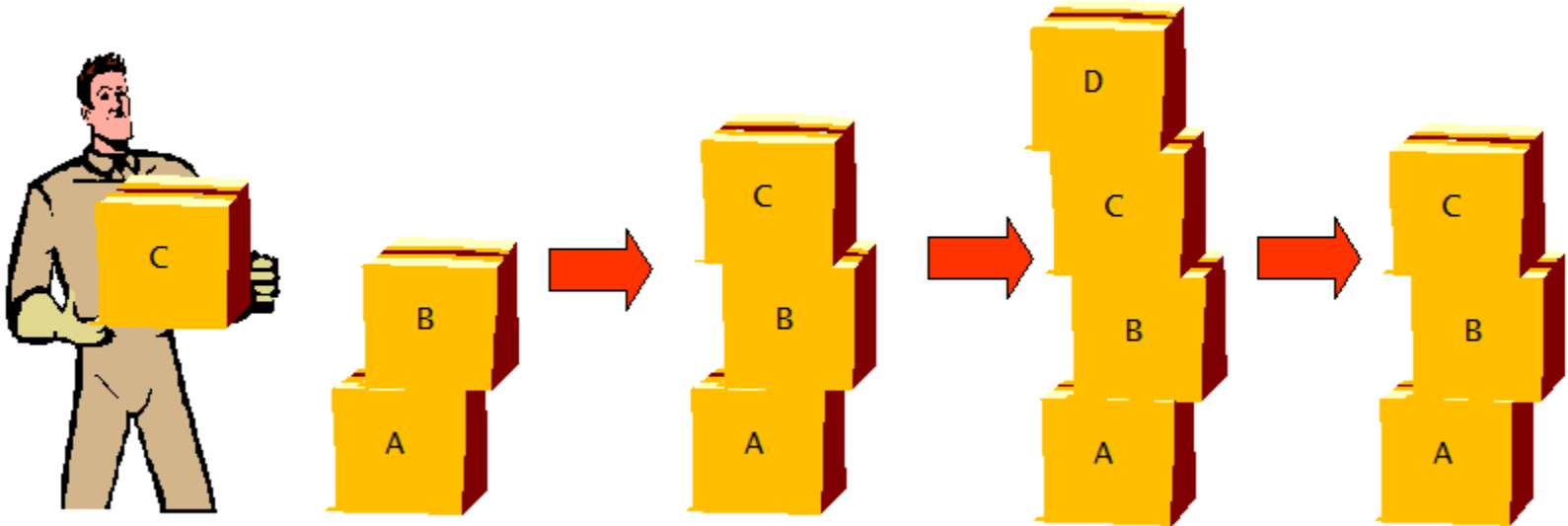
1. 클래스 템플릿 형태로 라이브러리를 제공하면 어떤 장점이 있는가?
2. 세개의 데이터를 가지고 있는 Triple라는 클래스를 클래스 템플릿으로 작성하여 보라.





예제: 스택

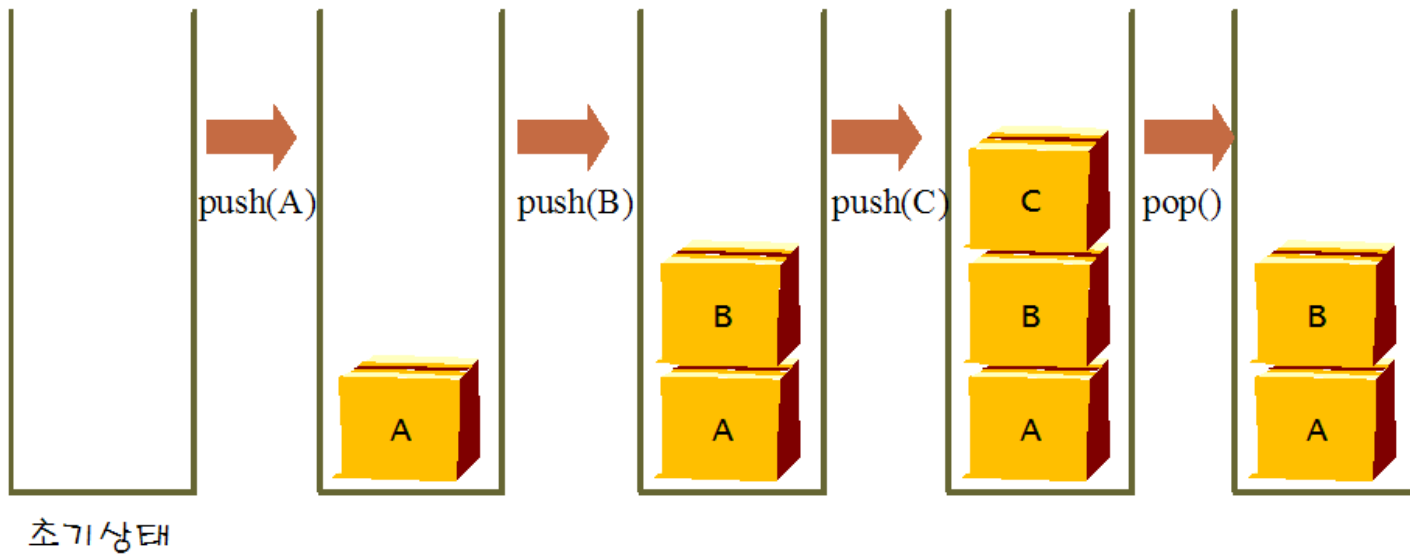
- 스택(stack): 후입 선출(LIFO: Last-In First-Out) 자료 구조





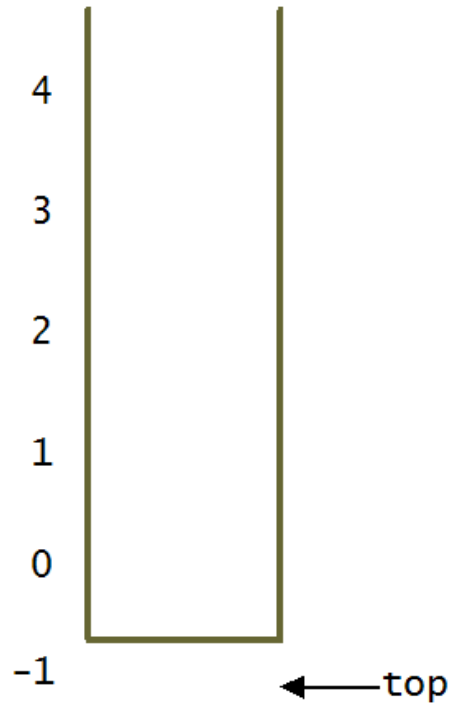
스택의 연산들

- `is_empty(s)` ::= 스택이 비어있는지를 검사한다.
- `is_full(s)` ::= 스택이 가득 찼는가를 검사한다.
- `push(s, e)` ::= 스택의 맨 위에 요소 `e`를 추가한다.
- `pop(s)` ::= 스택의 맨 위에 있는 요소를 삭제한다.

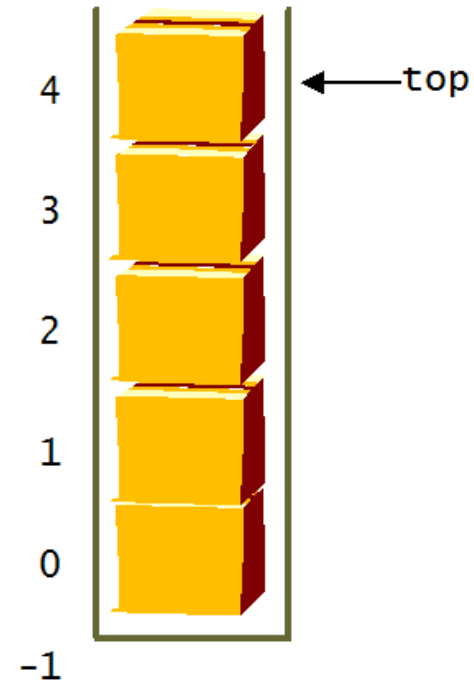




스택의 공백 상태와 포화 상태



(a) 공백상태



(b) 포화상태



isEmpty() , isFull()

isEmpty()

```
if top = -1  
    then return TRUE  
    else return FALSE
```

isFull()

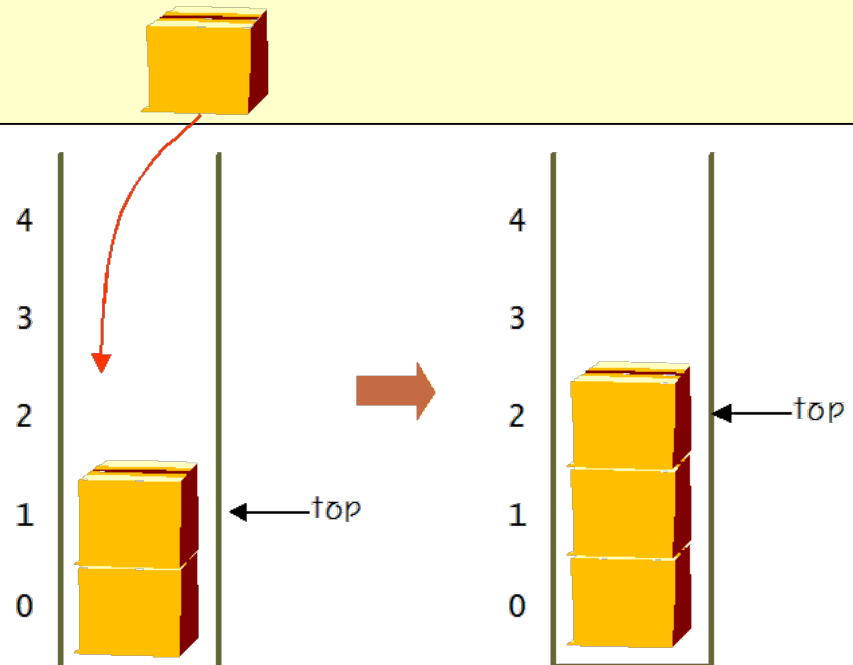
```
if top = (MAX_STACK_SIZE-1)  
    then return TRUE  
    else return FALSE
```



push()

push(x)

```
if isFull()  
  then error "overflow"  
else  $top \leftarrow top + 1$   
    $stack[top] \leftarrow x$ 
```

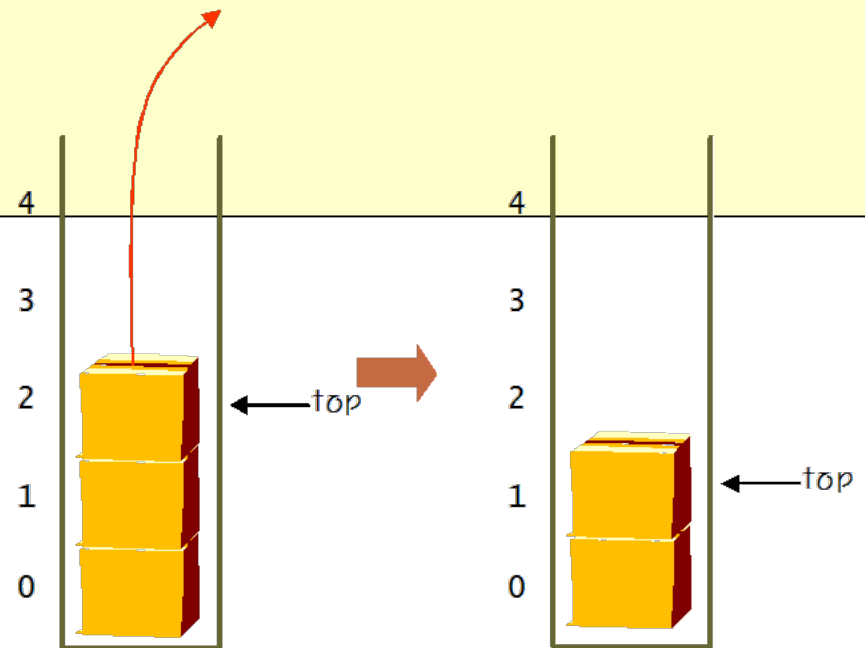




pop()

pop(x)

```
if isEmpty()  
  then error "underflow"  
  else  $e \leftarrow \text{stack}[\text{top}]$   
        $\text{top} \leftarrow \text{top} - 1$   
  return  $e$ 
```





스택의 구현



```
#include <iostream>
using namespace std;
// 예외 처리를 위한 클래스
class FullStack
{
};

// 예외 처리를 위한 클래스
class EmptyStack
{
};
```



스택의 구현



```
template <class T>  class는 typename을 대체하는 것.  
                    typename을 쓸 것을 권장함.  
class Stack {  
private:  
    T* s;  
    int  size;  
    int  top;  
public:  
    Stack(int n = 100) : size(n), top(-1)  
    {  
        s = new T[size];  
    }  
    ~Stack() { delete []s; }  
    void push(T v);  
    T pop();  
    bool isEmpty() const { return top == -1;}  
    bool isFull() const { return top == size - 1;}  
};
```



스택의 구현



```
template< typename T >
void Stack< T >::push( T v )
{
    if ( isFull() )
        throw FullStack();
    s[ ++top ] = v;
}

template< typename T >
T Stack< T >::pop( )
{
    if ( isEmpty() )
        throw EmptyStack();
    return s[ top-- ];
}
```



스택의 구현



```
int main()
{
    Stack<int> s; // 크기가 100인 정수형 스택
    s.push(100);
    s.push(200);
    s.push(300);
    s.push(400);
    cout << s.pop() << endl;
    cout << s.pop() << endl;
    cout << s.pop() << endl;
    cout << s.pop() << endl;
    return 0;
}
```



400
300
200
100
계속하려면 아무 키나 누르십시오 . . .



스택의 응용

stack2.cpp

```
...    // 앞의 스택 클래스 포함
int main()
{
    Stack<char> s; // 크기가 100인 문자형 스택
    string str = "madamimadam";

    for(int i=0;i<str.length(); i++)
        s.push(str[i]);

    for(int i=0;i<str.length(); i++) {
        if( s.pop() != str[i] ) {
            cout << "주어진 문자열은 회문이 아님" << endl;
            return 0;
        }
    }
    cout << "주어진 문자열은 회문임" << endl;
    return 0;
}
```



실행 결과

실행 결과

주어진 문자열은 회문임
계속하려면 아무 키나 누르십시오 . . .



Q & A

