

# Programming Study:

1. Introduction
2. Stack

---

## 2nd week

# 1. Introduction



# Agenda

---

- Introductions
- Algorithm
- Pseudo Code
- Abstract Data Type(ADT)

# Algorithm

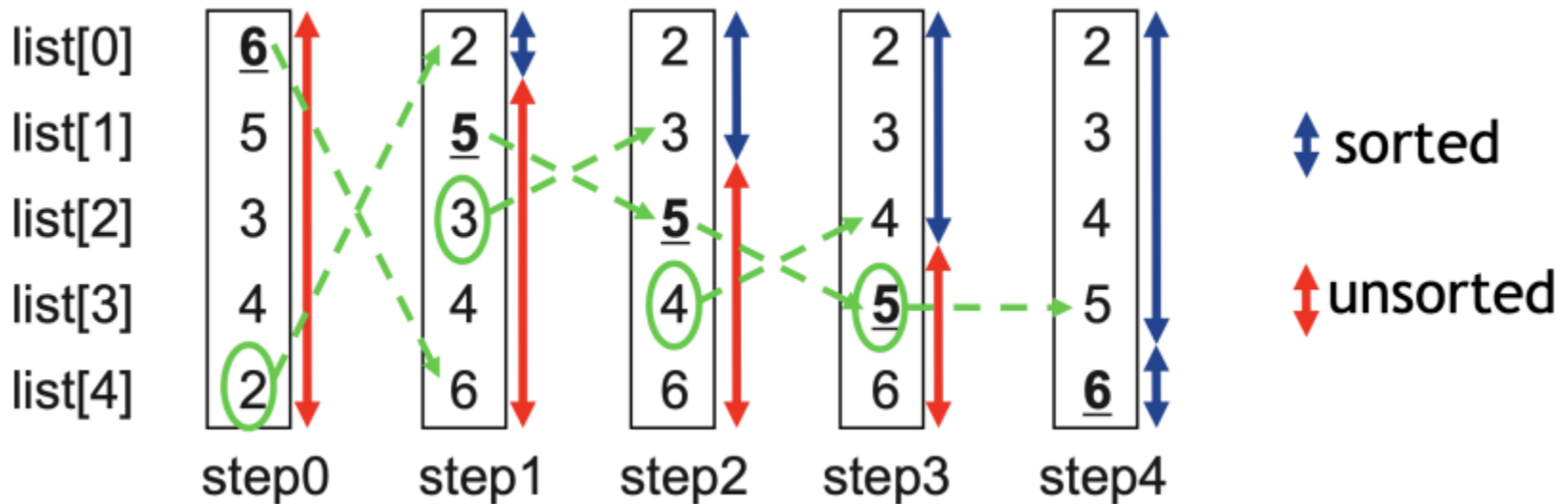
---

- 알고리즘: 특정 task를 수행하기 위한 instruction들의 유한 집합.
  - **Input:** 0 또는 그 이상의 입력을 지님
  - **Output:** 적어도 하나 이상의 값을 가짐
  - **Definiteness:** 명확하고, 모호하지 않음
  - **Finiteness:** 유한 시간 내에 종료되어야 함
  - **Effectiveness:** 기본적이고 실현 가능한 instruction

# Example: Selection Sort

- 문제 정의:  $n$ 개의 정수들을 정렬해야 한다.

해결 방법: 정렬되지 않은 수 중에서 가장 작은 수를 골라 정렬된 것 이후에 놓는다.



```
for (i = 0; i < n; i++) {  
    Examine list[i] to list[n-1] and suppose the smallest int is at list [min];  
    Interchange list[i] and list[min];  
}
```

# Pseudo Code

---

- Pseudo code(의사 코드): 알고리즘 로직을 일반적인 언어로 작성한 코드

```
for (i = 0; i < n; i++) {  
    Examine list[i] to list[n-1] and suppose the smallest int is at list [min];  
    Interchange list[i] and list[min];  
}
```

- Pseudo code(의사 코드)가 필요한 이유
  - 코드 작성 전 사고 정립에 도움을 줌
  - 의사 코드를 통해 프로그래밍을 하는데 더 수월함
  - 코드 검토를 하기 쉬움

# Abstract Data Type(ADT, 추상자료형)

---

- ADT: 구체적인 기능 구현에 대한 코드가 없이, 순수하게 기능이 무엇인지를 나열한 것
- Example: 지갑의 ADT

```
class Wallet{  
private:  
    int coin_500;  
    int bill_1000;  
  
public:  
    void TakeOutMoney(Wallet* pw, int coinNum, int billNum);  
    void PutMoney(Wallet* pw, int coinNum, int billNum);  
};
```

**void TakeOutMoney(Wallet\* pw, int coinNum, int billNum);**

- 첫 번째 인자로 전달된 주소의 지갑에서 돈을 꺼낸다.
- 두 번째 인자로 꺼낼 동전의 수, 세 번째 인자로 꺼낼 지폐의 수를 전달한다.
- 그 돈만큼의 금액을 지갑에서 차감한다.

**void PutMoney(Wallet\* pw, int coinNum, int billNum);**

- 첫 번째 인자로 전달된 주소의 지갑에서 돈을 넣는다.
- 두 번째 인자로 넣을 동전의 수, 세 번째 인자로 넣을 지폐의 수를 전달한다.
- 넣을 만큼의 동전과 지폐의 수가 증가한다.

# Abstract Data Type(ADT, 추상자료형)

- ADT: 구체적인 기능 구현에 대한 코드가 없이, 순수하게 기능이 무엇인지를 나열한 것
- Stack의 ADT

```
CMakeLists.txt x hw1.cpp x main.cpp x Stack.h x
1 #include<iostream>
2
3 inline void error(const char *str){
4     std::cout << str << std::endl;
5     exit( Code: 1);
6 }
7
8 const int MAX_STACK_SIZE = 10;
9
10 class Stack{
11 private:
12     int top;
13     int* data = new int[MAX_STACK_SIZE];
14
15 public:
16     Stack();
17     ~Stack();
18     void push(int x);
19     void pop();
20     int peek();
21     bool isFull();
22     bool isEmpty();
23     int size();
24     void display();
25 };
```

```
CMakeLists.txt x hw1.cpp x main.cpp x Stack.h x
This file does not belong to any project target; code insight features might not work prop
1 #include <iostream>
2 #include "Stack.h"
3
4 Stack::Stack() {
5     top = -1;
6 }
7
8 Stack::~Stack(){
9     delete [] data;
10 }
11
12 void Stack::push(int x){
13     //TODO
14 }
15
16 void Stack::pop(){
17     //TODO
18 }
19
20 int Stack::peek(){
21     //TODO
22 }
23
24 bool Stack::isFull() {
25     //TODO
26 }
27
28 bool Stack::isEmpty() {
29     //TODO
30 }
```



## 2. Stack



# Agenda

---

- Introductions
- Stack
- Stack ADT
- Example: Brace check program

# Introduction

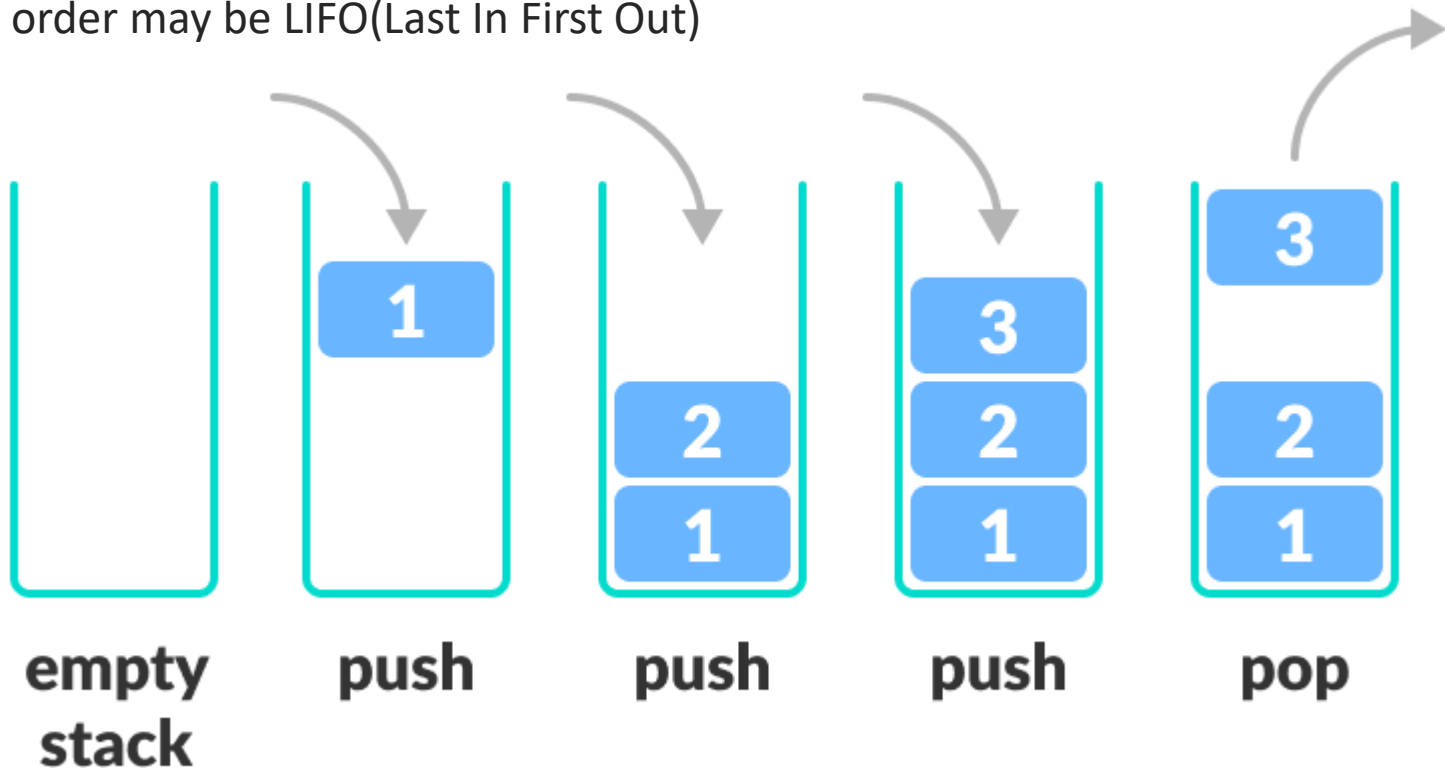
---

- 1) 모든 프로그램은 C++언어로 작성합니다.
- 2) 제출 시 main.cpp, stack.h를 하나의 파일에 담아 .zip으로 압축해주세요
- 3) //TODO에 해당하는 부분을 채워 제출하시면 됩니다.
- 4) 원한다면 필요에 따라 함수를 추가할 수 있습니다.
- 5) 단, 주어진 함수는 수정(ex. 함수명, 변수 변경 등)을 하지 않은 채 사용되어야 합니다.

# Stack

---

- Stack is a special case of ordered linear list
- Data insertion(push) and deletion(pop) are only happened at the top of the stack
- The order may be LIFO(Last In First Out)



# Stack ADT

```
CMakeLists.txt × hw1.cpp × main.cpp × Stack.h ×
1  #include<iostream>
2
3  inline void error(const char *str){
4      std::cout << str << std::endl;
5      exit( Code: 1);
6  }
7
8  const int MAX_STACK_SIZE = 10;
9
10 class Stack{
11     private:
12         int top;
13         int* data = new int[MAX_STACK_SIZE];
14
15     public:
16         Stack();
17         ~Stack();
18         void push(int x);
19         void pop();
20         int peek();
21         bool isFull();
22         bool isEmpty();
23         int size();
24         void display();
25 };
```

구현해야 할 함수

- push(int x)
- pop()
- peek()
- isFull()
- isEmpty()
- size()
- display()

# Example – Brace check program

소괄호들로 이루어진 문자열을 입력 받아,  
균형 잡힌 괄호(balanced parentheses) 문자열인지 판별하는 코드를 작성하시오.

Brackets	Matched
(( ))	O
( )	X
(( ( ) ) ( ) )	O
) ) ( (	X

예제 입력	예제 출력
(( )) ( )	NO
(( ( ) ( ) ) ( )	NO
(( ) ( ) ) (( ) )	YES
(( ( ) ( ) ) (( ( ) ) ) ) ( )	NO
( ) ( ) ( ) ( ) ( ) ( ) ( )	YES
(( ) (( ) ) ( ) (	NO
( ) ) ( (	NO

# Example – Brace check program

---

- 입력 받은 문자열의 길이만큼 반복문을 수행
- 여는 괄호를 만나면 stack에 '(' 를 push한다.
- 닫는 괄호를 만나면 stack에서 pop하며, pop을 할 수 없는 상태이면 NO를 출력한다.
- 마지막까지 반복했을 때 stack이 비어 있으면 YES, 아니면 NO를 출력한다.







# Example – Brace check program

```
CMakeLists.txt × main.cpp × stack.h ×
41 bool check_brace(char str[]){
42     Stack stack;
43
44     for (int j = 0; str[j]; j++){
45         if ( str[j] == '(' )
46             stack.push(x '(');
47         else if ( str[j] == ')' ){
48             if ( stack.isEmpty() )
49                 return false;
50             stack.pop();
51         }
52     }
53     if ( stack.isEmpty() )
54         return true;
55     return false;
56 }
57 int main(){
58     char str[50];
59     std::cin >> str;
60
61     if ( check_brace(str) )
62         std::cout << "YES" << std::endl;
63     else
64         std::cout << "NO" << std::endl;
65
66 }
```



# Example – ++Brace check program;

입력 문자열에 () 뿐만이 아니라, {}, []도 입력될 때  
균형 잡힌 괄호(balanced parentheses)를 판별하는 프로그램

Brackets	Matched	Balanced
{ ( ) } [ ]		
{ ( ) } ) [ ]		
{ ( ) [ ] }		
{ } ) ( [ ]	