Programming Methodology
Practice Session #11

Class Template

Class Template (1)

- Class의 member들이 정해진 type이 아닌 **일반적인 (generic) type**으로 class의 작동이 가능하도록 구현하는 기능.
  - 서로 다른 type의 data를 처리하기 위해 여러 개의 class를 정의할 필요가 없다.

- Template class의 instance를 만들 때 type을 지정하면 지정된 type에 대해 **구체화(specialization)**된다.
  - `int`, `double` 등의 기본 type 외에 structure나 class에 대해서도 구체화가 가능하다.

Class Template (2)

```
template <class T>
class TemplateExample       // Template class
{
    private:
        T list[5];          // Member Variable
    public:
        void setData(int index, T data);       // List[index]에 data 저장
        T getData(int index);                  // List[index]의 값 return
};

// Member function 구현
template <class T>
void TemplateExample<T>::setData(int index, T data)     // 문법에 유의
{   list[index] = T;   }

template <class T>
T TemplateExample<T>::getData(int index)
{   return list[index];   }
```

Class Template (3)

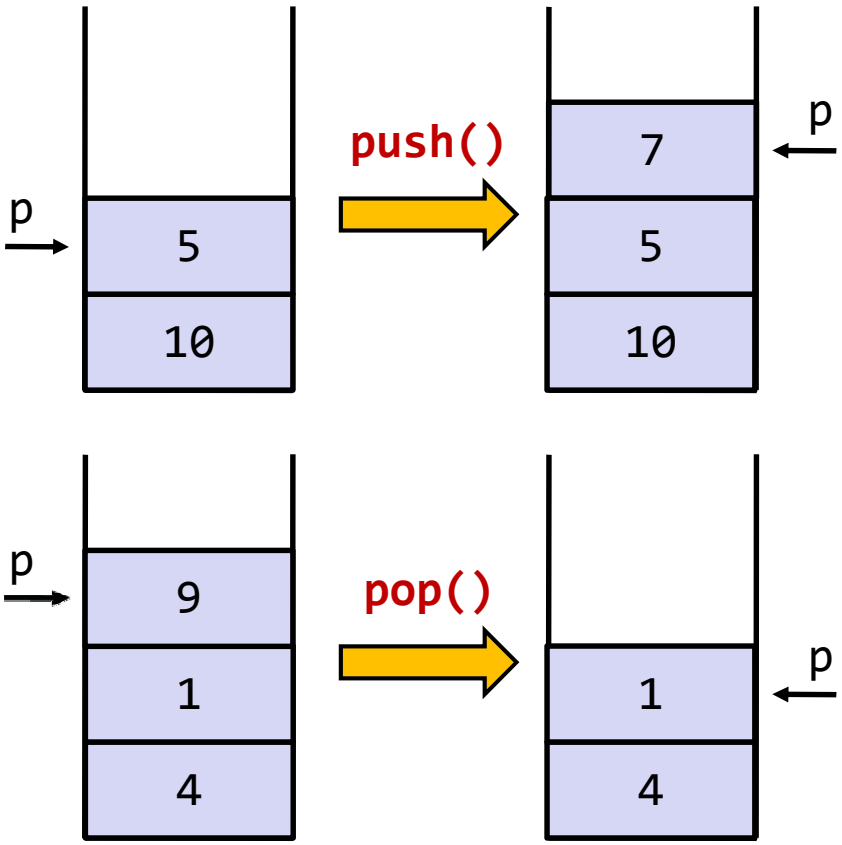
```
void main()
{
    int i;
    TemplateExample<int> obj1;          // int type에 대해 구체화
    TemplateExample<char> obj2;         // char type에 대해 구체화

    obj1.setData(0, 4);       // TemplateExample<int>의 member function 호출
    obj1.setData(1, 2);
    obj1.setData(2, 5);

    obj2.setData(0, 'R');     // TemplateExample<char>의 member function 호출
    obj2.setData(1, 'L');
    obj2.setData(2, 'C');

    for(i=0; i<3; i++)
          cout << obj1.getData(i) << ", " << obj2.getData(i) << endl;
}
```

Sample Practice (1)

- **Stack<class T>** class를 구현한다.
  - `T* data; int p;`
  - `Stack();`
  - `void push(T data);`
  - `T pop();`
  - `int empty();`
- **Queue<class T>** class를 구현한다.
  - `T* data; int head, tail;`
  - `Queue();`
  - `void enqueue(T data);`
  - `T dequeue();`
  - `int empty();`

Sample Practice (2)

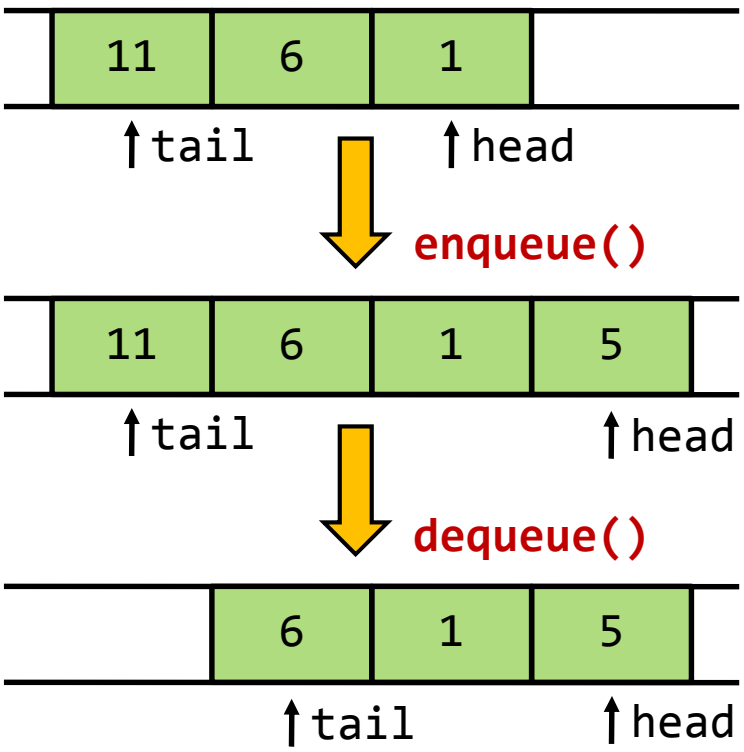
- **Stack** (LIFO: **Last In, First Out**)



- `void push(T data)`
  - data를 Stack의 가장 윗부분(top)에 저장하고, stack pointer p를 증가시킨다.
- `T pop()`
  - Stack의 가장 윗부분(top)에 저장되어 있는 data를 return하고, stack pointer p를 감소시킨다.
- `int empty()`
  - Stack이 비었는지를 체크

Sample Practice (3)

- **Queue** (FIFO: **First In, First Out**)



- **void** enqueue(T data)
  - data를 Queue의 head쪽에 저장하고, head pointer를 증가시킨다.
- T dequeue()
  - Queue의 tail쪽에 저장되어 있는 data를 return하고, tail pointer 를 증가시킨다.
- **int** empty()
  - Queue가 비었는지를 체크

Sample Practice (4)

```
void main()
{
    // TODO: Create a stack as "int" type

    // Push int numbers to the stack
    cout << "PUSH : 3" << endl;      st.push(3);
    cout << "PUSH : 10" << endl;     st.push(10);
    cout << "PUSH : 7" << endl;      st.push(7);
    cout << "PUSH : 15" << endl;     st.push(15);

    // Pop int numbers from the stack
    cout << "POP : " << st.pop() << endl;
    cout << "POP : " << st.pop() << endl;
    cout << "POP : " << st.pop() << endl;

    // Check the stack is empty
    if(st.empty())  cout << "The stack is empty!!" << endl;
    else            cout << "The stack is not empty!!" << endl;

    cout << endl;
```

Sample Practice (5)

```
    // TODO: Create a queue as "double" type

    // Enqueue double numbers to the queue
    cout << "ENQUEUE : 6.4" << endl;     qu.enQueue(6.4);
    cout << "ENQUEUE : 12" << endl;      qu.enQueue(12);
    cout << "ENQUEUE : 80" << endl;      qu.enQueue(80);

    // Dequeue double numbers from the queue
    cout << "DEQUEUE : " << qu.deQueue( ) << endl;
    cout << "DEQUEUE : " << qu.deQueue( ) << endl;
    cout << "DEQUEUE : " << qu.deQueue( ) << endl;

    // Check the queue is empty
    if(qu.empty( ))  cout << "The queue is empty!!" << endl;
    else             cout << "The queue is not empty!!" << endl;
}
```

Sample Practice (6)

```
PUSH : 3
PUSH : 10
PUSH : 7
PUSH : 15
POP : 15
POP : 7
POP : 10
The stack is not empty!!

ENQUEUE : 6.4
ENQUEUE : 12
ENQUEUE : 80
DEQUEUE : 6.4
DEQUEUE : 12
DEQUEUE : 80
The queue is empty!!
계속하려면 아무 키나 누르십시오 . . .
```