



본 PSet 은 저의 강의 경험과 학생들의 의견 및 Stanford CS106 과 Harvard CS50 같은 강의에서 수집된 자료를 토대로 작성되었습니다.  
본 PSet 에 문제점이나 질문 혹은 의견이 있다면, 저의 이메일([idebtor@gmail.com](mailto:idebtor@gmail.com))로 알려 주시면 강의 개선에 많은 도움이 되겠습니다.

## PSet: Stack

### 목차

과제 수행 목적 .....	1
제공되는 파일 목록 .....	1
C++ Template .....	1
Step 1. Using a stack class in STL - stack version 1, stack_stl1.cpp .....	2
Step 2. Using a fixed sized array - stack version 2, stack2_arr.cpp .....	4
Step 3. Using a dynamic size array - version 3, stack3_arr.cpp .....	5
Step 4. Using a vector in C++ STL - stack4_vec.cpp .....	5
Step 5. Using Template - version 4T, stack4_vecT.cpp .....	6
과제 제출 .....	6
제출 파일 목록 .....	6
마감 기한 & 배점 .....	7

### 과제 수행 목적

이 프로젝트의 목적은 C++ STL(Standard Template Library)에서 제공하는 stack 과 유사한 스택을 우리가 스스로 만들어 보면서, 스택의 원리와 개념을 학습하고자 합니다. 더불어 C++ STL 를 stack 을 정의할 때 사용함으로 Template 을 활용하는 방법도 학습할 것입니다. 이러한 학습의 과정 중에 다양한 버전의 stack program 들을 만들어 나갈 것입니다.

- Stack 원리와 개념 이해하기
- Static Array 와 Dynamic Array 이용하여 Stack 만들기
- 성능 분석 이론 적용하기 및 Successive Doubling 알고리즘 이해하기
- STL vector 및 Template 으로 Stack 만들기
- Reference 와 lvalue 이해하고, Stack 에 적용하기

### 제공되는 파일 목록

- stack.pdf                      본 파일

### C++ Template

### Stack: Using template

- A **template** is a mechanism that allows a programmer to use types as parameters for a class or a function. The compiler then generates a specific class or function when we **later** provide specific types as arguments.
- A function/class defined using **template** is called a **generic function/class**. This is one of the key features of C++.
- Use **templates** when we need functions/classes that apply the same algorithm to a several types. So we can use the same function/class regardless of the types of the argument or result.
- The syntax is:
  - `template <class T> function_declaration;`
  - or
  - `template <typename T> function_declaration;`

### Pros and Cons of Templates

- **Pros:**
  - It provides us **type-safe, efficient** generic containers and generic algorithms
  - The main reason for using C++ and templates is the trade-offs in performance and maintainability outweigh the bigger size of the resulting code and longer compile times.
  - The drawbacks of not using them are likely to be much greater.
- **Cons:**
  - Templates can lead to **slower compile-times** and possibly larger executable.
  - Compilers often produce incomprehensible poor error diagnostics and **poor error messages**.

## Step 1. Using a stack class in STL - stack version 1, stack\_stl1.cpp

C++ STL 에서 제공하는 stack class 를 활용하는 법을 학습합니다. 다음 코드는 int 혹은 string 타입의 배열을 스택에 push 하거나 pop 을 하고, 출력합니다.

### Stack: version.1 – using a stack class in C++ STL

stack1\_stl1.cpp

```

int main () { // stack initialization using range-based for
// int list[] = {1, 2, 3, 4, 5, 0, 6, 0, 7, 0, 0, 0, 8};
string list[] = {"to", "be", "or", "not", "to", "-", "be", \
                "-", "-", "that", "-", "-", "-", "is"};

stack<string> s;
for (auto item : list) { // to be not that or be (5 6 4 7 3 2)
    if (item != "-") // type specific
        s.push(item);
    else {
        cout << s.top() << ' ';
        s.pop();
    }
}
cout << "\nsize: " << s.size(); // 2
cout << "\ntop : " << s.top(); // is (8)
cout << "\nstack T: "; printStack(s); // is to (8 1)
cout << "\nstack B: "; printStack_fromBottom(s); // to is (1 8)
cout << "\nHappy Coding";
}

void printStack(stack<string> s) {
    while (!s.empty()) {
        cout << s.top() << ' ';
        s.pop();
    }
    // cout << endl; // now, s is empty
}
  
```

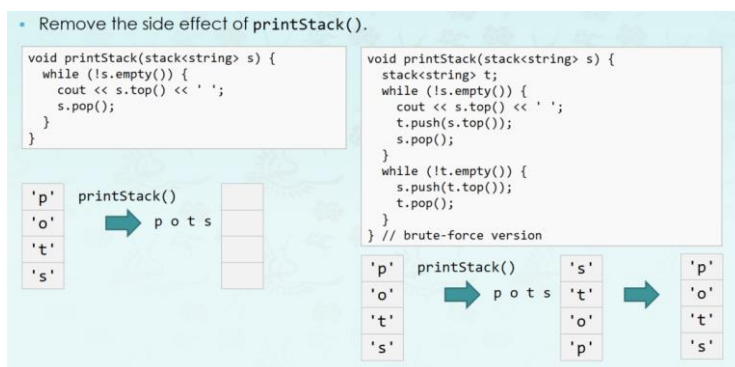
Prof. Youngsup Kim, [sdetlar@gmail.com](mailto:sdetlar@gmail.com), Data Structures, C&EE Dept., Hanyang Global University

- stack1\_stl1.cpp 로 STL stack 학습하기
  - 위 코드로 stack1\_stl1.cpp 파일을 작성하고 컴파일하여 실행해보며, 스택 함수의 호출과 그 결과에 대해 학습하시오. 일단, printStack\_fromBottom() 호출하는 줄은 comment 처리해야 할 것입니다.
  - int list[]를 활성화하고, string list[]는 comment 처리하고 다시 컴파일하고 실행해보십시오. 단, 앞에서 pop()의 기호로 사용한 "-" 대신 0 로 수정합니다.
- printStack\_fromBottom() 를 구현하기.

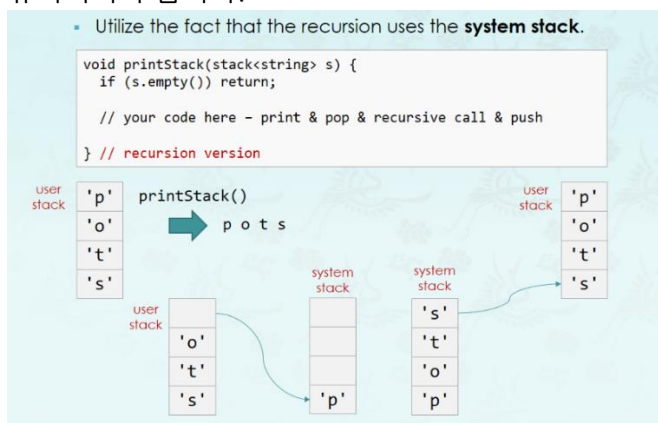
- printStack()의 문제는 stack의 내용을 top부터 출력하며, 동시에 stack의 내용을 다 비우게 됩니다. 이런 현상을 함수의 side effect라고 합니다. 자신의 함수의 이름과 다른 작업을 한 것입니다.

```
void printStack(stack<string> s) {
    while (!s.empty()) {
        cout << s.top() << ' ';
        s.pop();
    }
}
```

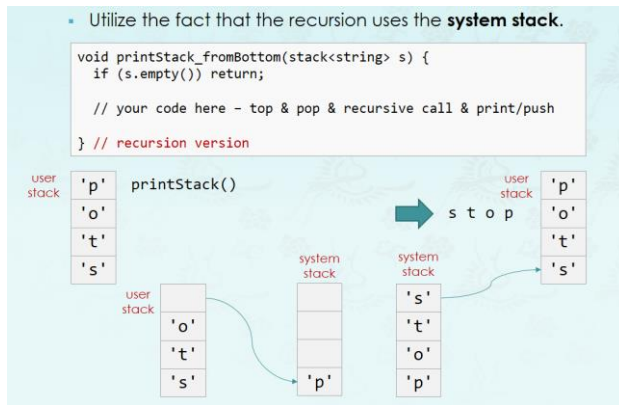
- 우리는 이제 NSE(No side effect)하도록 다음 그림의 왼쪽과 같이 printStack()코드를 수정할 수 있습니다. 이 코드를 이해하고 실행해보십시오.



- 아래 그림을 참고하고, 재귀용법(recursion)을 사용하여 stack의 내용을 위로부터 아래로 출력하도록 printStack()을 구현하십시오. 물론, 출력한 후에도 stack의 형태가 처음과 같이 유지되어야 합니다.



- 아래 그림을 참고하고, 재귀용법(recursion)을 사용하여 stack의 내용을 아래로부터 위로 출력하는 printStack\_fromBottom()함수를 구현하십시오. 물론, 출력한 후에도 stack의 형태가 처음과 같이 유지되어야 합니다.



- 위의 코드는 stack\_stl1.cpp 에 저장하고, 실행하면 다음과 같은 결과를 출력합니다.

```
PS C:\Github\nowicx\src> g++ stack1_stl.cpp; ./a
to be not that or be
size: 2
top : is
stack T: is to
stack B: to is
Happy Coding
PS C:\Github\nowicx\src>
```

- stack1\_stl.cpp 파일을 제출합니다.

## Step 2. Using a fixed sized array - stack version 2, stack2\_arr.cpp

- C++ STL stack 을 사용하지 않고, 이제는 사용자가 지정해주는 고정된 크기의 배열을 사용하여 스택 자체를 만들어 보고자 합니다.
- 다음 그림의 코드는 사용자가 정의한 struct Stack 으로 stack 을 만드는 기본적인 코드입니다. 이를 stack\_arr2.cpp 파일을 작성하십시오.
- 함수들 중에 하나는 Reference 로 매개변수를 정의하고, 또 다른 함수는 Reference 로 반환하십시오

**Stack: version.2 – using a fixed size array**

```
struct Stack {
    string *item;
    int N;
    int capacity;
};
using stack = Stack *;

stack newStack(int capacity) {
    stack s = new Stack;
    s->item = new string[capacity];
    s->N = 0;
    s->capacity = capacity;
    return s;
}

void free(stack s) {
    delete[] s->item;
    delete s;
}
```

*a shortcoming (stay tuned)*

*item[N] is next to be filled if any.*

**stack2\_arr.cpp**

```
int size(stack s) { return s->N; }

bool empty(stack s) { return s->N == 0; }

void pop(stack s) { s->N--; }

string top(stack s) {
    return s->item[s->N - 1];
}

void push(stack s, string item) {
    s->item[s->N++] = item;
}

void printStack(stack s) {
    // your code here
}

void printStack_fromBottom(stack s) {
    // your code here
}
```

*N is not decremented*

*use N and incremented*

*N points an empty slot*

- stack1\_stl.cpp 파일에서 사용한 main() 함수를 복사하여 붙여 넣기를 하여 실행해보십시오. 여기 Stack 요소가 string 타입이므로, string 배열로 테스트해 보십시오.
- string 배열로 테스트 한 후, stack2\_arr.cpp 를 복사하여 **stack2\_arr.cpp** 를 만드십시오. 그리고, stack2\_arr.cpp 파일을 적절히 수정하여 int 타입의 배열이 작동하게 하고, 테스트 하십시오.

### Step 3. Using a dynamic size array - version 3, stack3\_arr.cpp

사용자가 스택의 크기를 magic number 로 지정하는 것보다는 필요에 따라 자동적으로 스택의 크기가 조절되는 스택을 만들고자 합니다.

- 다음 그림을 참고하여 stack3\_arr.cpp 파일을 작성하십시오. 단, 함수들 중에 하나는 Reference 로 매개변수를 정의하고, 또 다른 함수는 Reference 로 반환하십시오

Stack: version.3 – using a dynamic size array

stack3\_arr.cpp

```

struct Stack {
    string *item;
    int N;
    int capacity;
};
using stack = Stack *;

stack newStack(int capacity = 1) {
    stack s = new Stack;
    s->item = new string[capacity];
    s->N = 0;
    s->capacity = capacity;
    return s;
}
void free(stack s) {
    delete[] s->item;
    delete s;
}
int size(stack s) { return s->N; }

bool empty(stack s) { return s->N == 0; }
void pop(stack s) {
    s->N--;
    // your code here
}
string top(stack s) {
    return s->item[s->N - 1];
}
void push(stack s, string item) {
    // your code here
    s->item[s->N++] = item;
}
void printStack(stack s) {
    // your code here
}
void printStack_fromBottom(stack s) {
    // your code here
}

```

- 여기서 정의한 Stack 을 사용하여 stack1\_stl.cpp main()에서 테스트한 것이 여기서도 작동할 수 있도록 필요한 코딩하십시오. 또한 push()함수를 호출한 직후, size 와 capacity 가 어떻게 변하는지 관찰할 수 있도록 출력하십시오.

### Step 4. Using a vector in C++ STL - stack4\_vec.cpp

C++ STL vector 를 사용하여 struct Stack 을 구현하여 vector 의 크기가 필요에 따라 자동적으로 변하는 것을 관찰하고자 합니다.

- 여기서 정의한 Stack 을 사용하여 stack1\_stl.cpp main()에서 테스트한 것이 여기서도 작동할 수 있도록 필요한 코딩하십시오.
- 또한 push()함수를 호출한 직후, size 와 capacity 가 어떻게 변하는지 관찰할 수 있도록 출력하십시오. STL vector 에도 capacity 와 같은 종류가 있습니다. 여기서 capacity 는 자동적으로 증가는 하지만, 감소하지는 않을 것입니다.
- 함수들 중에 하나는 Reference 로 매개변수를 정의하고, 또 다른 함수는 Reference 로 반환하십시오.

Stack: version.4 – using a vector in C++ STL

stack4\_vec.cpp

```

struct Stack {
    vector<string> item;
};
using stack = Stack *;

void free(stack s) {
    delete s;
}

int size(stack s) {
    return s->item.size();
}

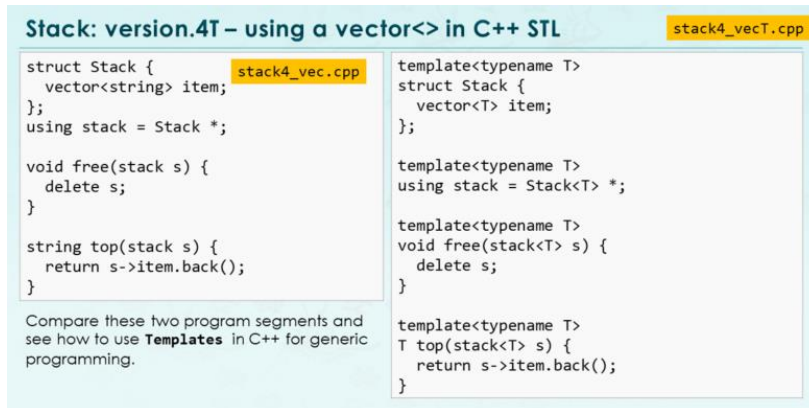
bool empty(stack s) {
    return s->item.empty();
}

void pop(stack s) {
    // your code here
}
string top(stack s) {
    // your code here
}
void push(stack s, string item) {
    // your code here
}
void printStack(stack s) {
    while (!empty(s)) {
        cout << top(s) << ' ';
        pop(s);
    }
    cout << endl; // stack is empty now
}

```

## Step 5. Using Template - version 4T, stack4\_vecT.cpp

다음 그림의 양쪽에 있는 코드를 비교해보십시오. 오른 쪽의 코드는 Template 을 사용하여 소위 Generic Programming 을 한 코드입니다.



- 앞에서 작성한 stack4\_vec.cpp 를 기본으로 하여, Template 를 사용하는 stack4\_vecT.cpp 파일을 작성하십시오.
- 앞에서 작성한 stack2\_arr.cpp 를 기본으로 하여, Template 를 사용하는 stack2\_arrT.cpp 파일을 작성하십시오.
- 함수들 중에 하나는 Reference 로 매개변수를 정의하고, 또 다른 함수는 Reference 로 반환하는 것을 확인하십시오.
- stack4\_vecT.cpp, stack2\_vecT.cpp 두 경우를 모두 테스트해보십시오  
또한 Reference 로 반환하는 경우도 테스트해보십시오.

## 과제 제출

- On my honour, I pledge that I have neither received nor provided improper assistance in the completion of this assignment.  
서명: \_\_\_\_\_ 학번: \_\_\_\_\_
- 제출하기 전에 코드가 제대로 컴파일이 되고 실행되는지 확인하세요. 제출 직전에 급하게 코드를 수정한 후 코드가 제대로 컴파일이 될 거라고 짐작하지 않는 게 좋습니다. “거의” 작동하는 코드도 틀린 것입니다.
- 과제가 컴파일 및 실행된다면, 마감 기한 전까지 과제의 일부만 완성했더라도 제출하기 바랍니다. 마감 시간 이후 24 시간 이내 제출하면, 만점에서 25% 감점하고 채점합니다. 그 이상 늦은 것은 채점하지 않으며, 0 점 처리합니다.
- 제출 후, **마감 기한 전까지** 수정 및 재제출이 가능합니다. 파일 하나만 수정하더라도 해당 파일과 관련된 파일들을 모두 재제출해야 합니다. 재제출 횟수는 제한 없습니다. 마감 기한 전에 **가장 마지막으로** 제출된 파일을 채점할 것입니다.

## 제출 파일 목록

다음 파일들을 Piazza 의 pset 폴더에 제출하세요.

- stack1\_stl.cpp
- stack3\_arr.cpp

- stack4\_vec.cpp
- stack4\_vecT.cpp
- stack2\_arrT.cpp

### 마감 기한 & 배점

---

- 마감 기한: 11:55 pm