Data Structures
Chapter 3

Problem set

1. **Stack**
   - Stack Concept
     - STL stack class
   - Stack Implementations
     - Using Fixed Array
     - Using Dynamic Array
     - Using Vector
     - Using STL Template

2. Queue

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

```cpp
int main () { // stack initialization using range-based for
  // int list[] = {1, 2, 3, 4, 5, 0, 6, 0, 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";
}
```

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

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept, Handong Global University

2

`stack1_stl.cpp`

```cpp
int main () { // stack initialization using range-based for
  // int list[] = {1, 2, 3, 4, 5, 0, 6, 0, 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";
}
```
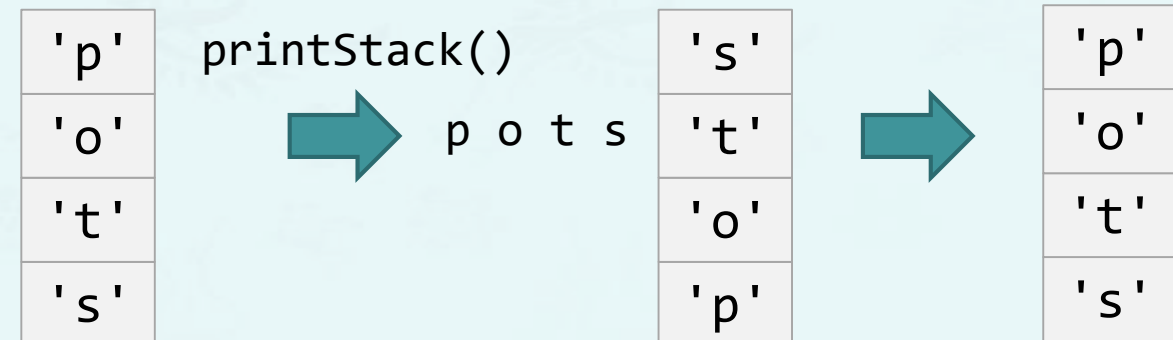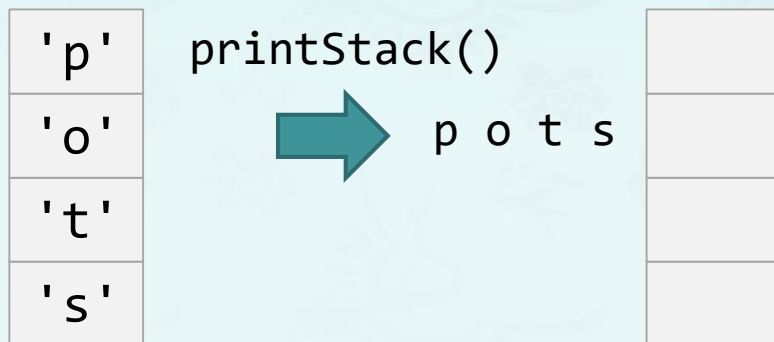
Step 1:
- Test this driver with "**int list[]**".
- Change a few places as needed.
- Fix `printStack()` not to empty the stack.
- Add **printStack_fromBottom()**.
- Be familiar with the concepts of stack.

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept, Handong Global University

3

- In-house programming principles: DRY, KISS, NMN, NSE
  - Remove the side effect of `printStack()`.

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


Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept, Handong Global University

4

`stack1_stl.cpp`

- In-house programming principles: DRY, KISS, NMN, NSE
  - Remove the side effect of `printStack()`.

```
void printStack(stack<string> s) {
  while (!s.empty()) {
    cout << s.top() << ' ';
    s.pop();
  }
}
```
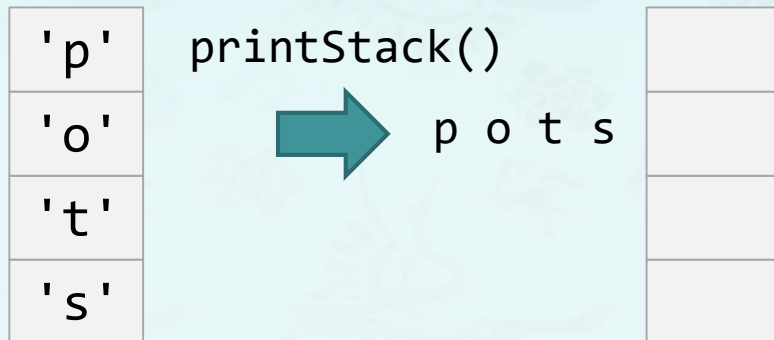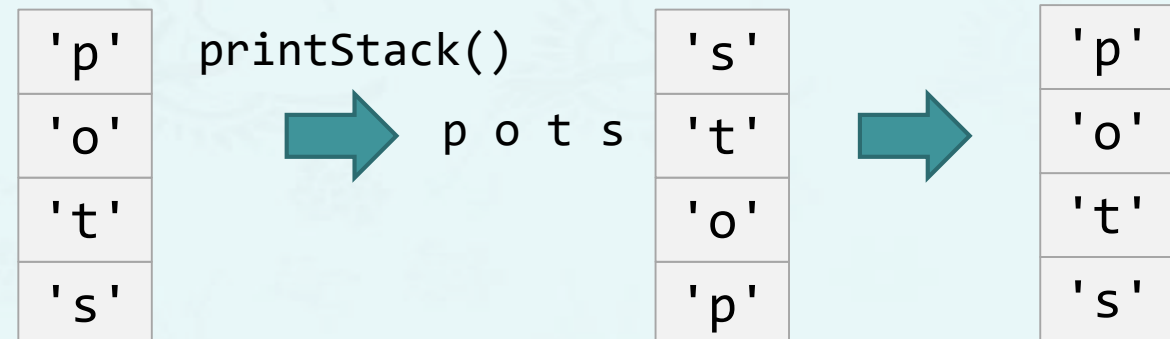
```
void printStack(stack<string> s) {
  stack<string> t;
  while (!s.empty()) {
    cout << s.top() << ' ';
    t.push(s.top());
    s.pop();
  }
  while (!t.empty()) {
    s.push(t.top());
    t.pop();
  }
} // brute-force version
```
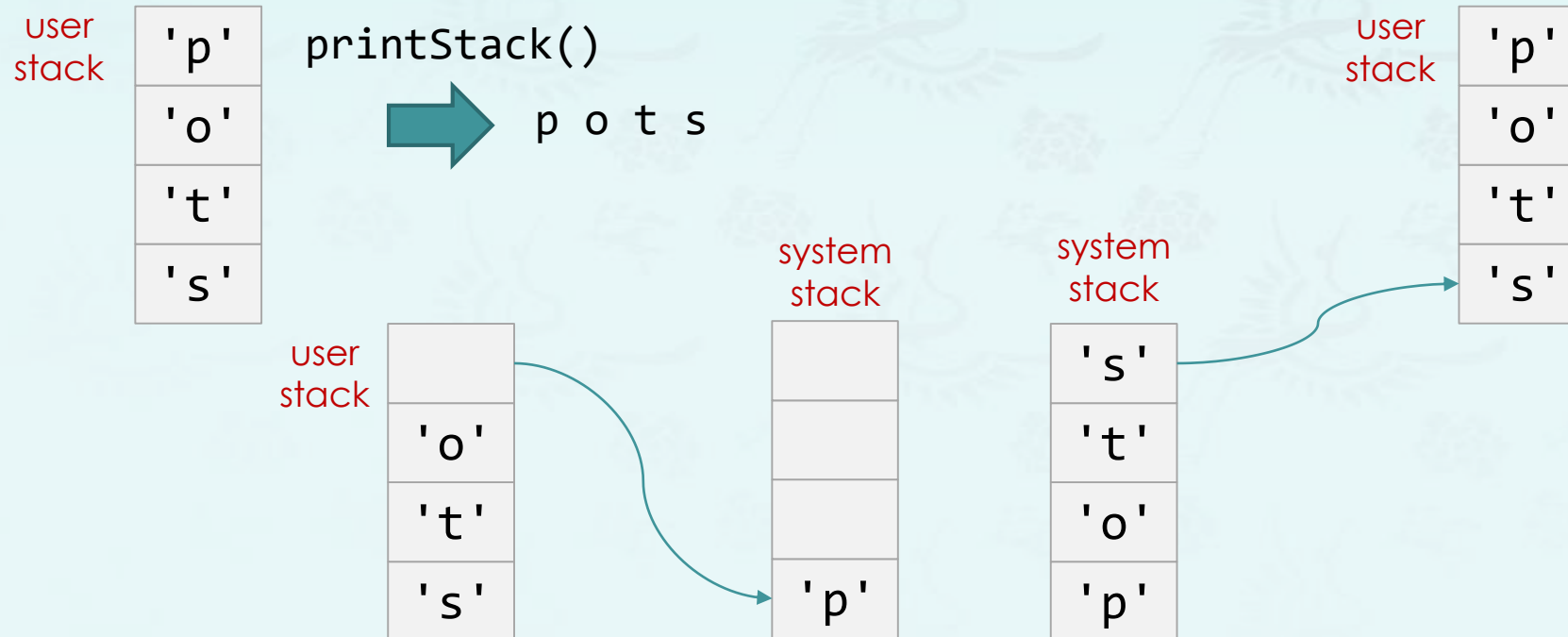

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept, Handong Global University

5

- Using recursion, print stack items from **top to bottom**.
  - Utilize the fact that the recursion uses the **system stack**.

```
void printStack(stack<string> s) {
  if (s.empty()) return;

  // your code here – print & pop & recursive call & push

} // recursion version
```

user
stack

'p'
'o'
't'
's'

printStack()

→ p o t s

user
stack

'o'
't'
's'

system
stack

'p'

system
stack

's'
't'
'o'
'p'

user
stack

'p'
'o'
't'
's'

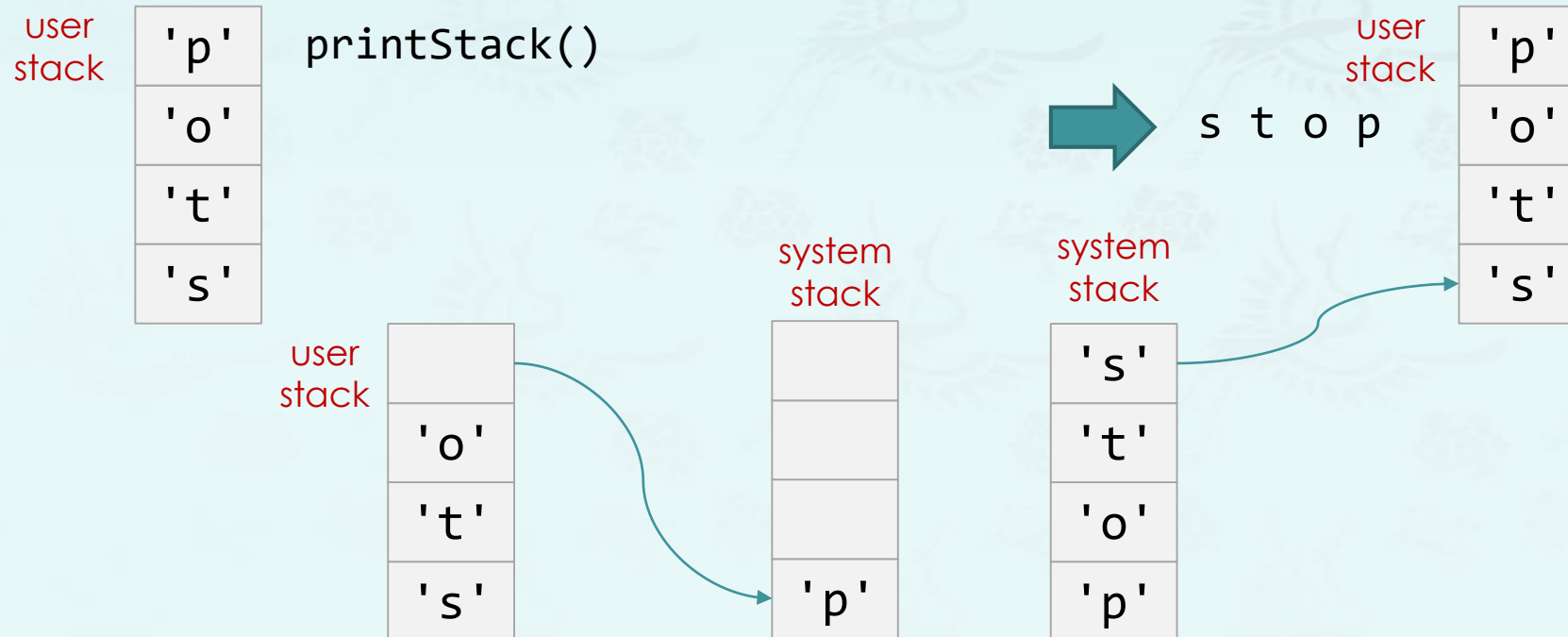*Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept, Handong Global University*

6

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

- Using recursion, print stack items from bottom to top.
  - Utilize the fact that the recursion uses the **system stack**.

```cpp
void printStack(stack<string> s) {
  if (s.empty()) return;

  // your code here – top & pop & recursive call & print/push

} // recursion version
```

printStack()

s t o p

`stack1_stl.cpp`

- Using recursion, print stack items from bottom to top.
  - Utilize the fact that the recursion uses the **system stack**.

```
void printStack(stack<string> s) {
  if (s.empty()) return;

  // your code

} // recursion
```

user
stack
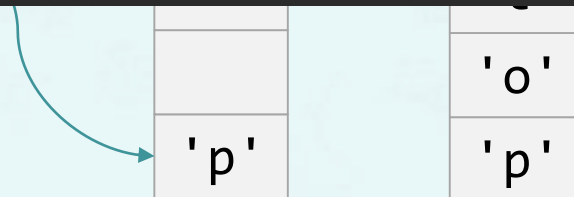
| 'p' |
| 'o' |
| 't' |
| 's' |

printS

user
stack

```
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>
```

| 'o' |
| 't' |
| 's' |

| 'p' |

| 'o' |
| 'p' |

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept, Handong Global University

8

`stack2_arr.cpp`

```cpp
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.

```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

9

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept, Handong Global University

`stack2_arr.cpp`

```cpp
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;
}
```

a shortcoming (stay tuned)

```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];
}
```
N is not decremented

```cpp
void push(stack s, string item) {
    s->item[s->N++] = item;
}
```
use N and incremented N points an empty slot

```cpp
void printStack(stack s) {
    // your code here
```

`ttom(stack s) {`

Step 2:
- Make this code snippet into a program, and name it **stack2_arr.app**.
- Create **main()** that works like **stack1_stl.cpp.**
- Test it with string data type.
- Create **stack2i_arr.app** such that it can handle **int** data type.
- Test it with **int** data type.

`stack3_arr.cpp`

```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; }
```

```cpp
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
}
```

`stack3_arr.cpp`

```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;
```

```cpp
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) {
```

Step 3:
- Make this code snippet into a program, and name it **stack3_arr.app**.
- Create or modify functions as needed that works like **stack1_stl.cpp.**
- Add **DPRINT** to show its **size** and **capacity** right after every **push()** call.
- Test it with **string** data type.
- Create **stack3i_arr.app** such that it can handle **int** data type.
- Test it with **int** data type.

(stack s) {

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept, Handong Global University

12

`stack4_vec.cpp`

```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) {
  s->item.pop_back();
}
```

Step 4:
- Make this code snippet into a program, and name it **stack4_vec.app**.
- Create functions as needed that works like **stack1_stl.cpp**.
- Add **DPRINT** to show its **size** and **capacity** right after every **push()** call. Capacity increases automatically, not decreases.
- Test it with **string** data type.
- Create **stack4i_arr.cpp** such that it can handle **int** data type.
- Test it with **int** data type.

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept, Handong Global University

13

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

```cpp
struct Stack {                    stack4_vec.cpp
  vector<string> item;
};
using stack = Stack *;

void free(stack s) {
  delete s;
}

string top(stack s) {
  return s->item.back();
}
```

Compare these two program segments and see how to use **Templates** in C++ for generic programming.

```cpp
template<typename T>
struct Stack {
  vector<T> item;
};

template<typename T>
using stack = Stack<T> *;

template<typename T>
void free(stack<T> s) {
  delete s;
}

template<typename T>
T top(stack<T> s) {
  return s->item.back();
}
```

`stack4_vecT.cpp`

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

void free(stack s) {
    delete s;
}

string top(stack s) {
    return s->item.back();
}
```

`stack4_vec.cpp`

```cpp
template<typename T>
struct Stack {
    vector<T> item;
};

template<typename T>
using stack = Stack<T> *;

template<typename T>
void free(stack<T> s) {
    delete s;
}

template<typename T>
```

Compare these two program segments and see how to use **Templates** in C++ programming.

Step 5:
- Based on **stack4_vec.cpp**, create **stack4_vecT.cpp** such that it uses Templates in C++.
- Based on **stack2_arr.cpp**, create **stack2_arrT.cpp** such that it uses Templates in C++.
- Test it with **int** data type.

15

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept, Handong Global University

Pset - Stack:

- Files provided: this pdf file
- Files to submit:
  - **step 3: `stack3_arr.cpp`**
  - **step 4: `stack4_vec.cpp`**
  - **step 5: `stack4_vecT.cpp, stack2_arrT.cpp`**
- Due:
  - 11:55 pm
- Grade:
  - step 3: 1.0 point
  - step 4: 1.0 point
  - step 5: 1.0 point

Data Structures
Chapter 3

1. **Stack**
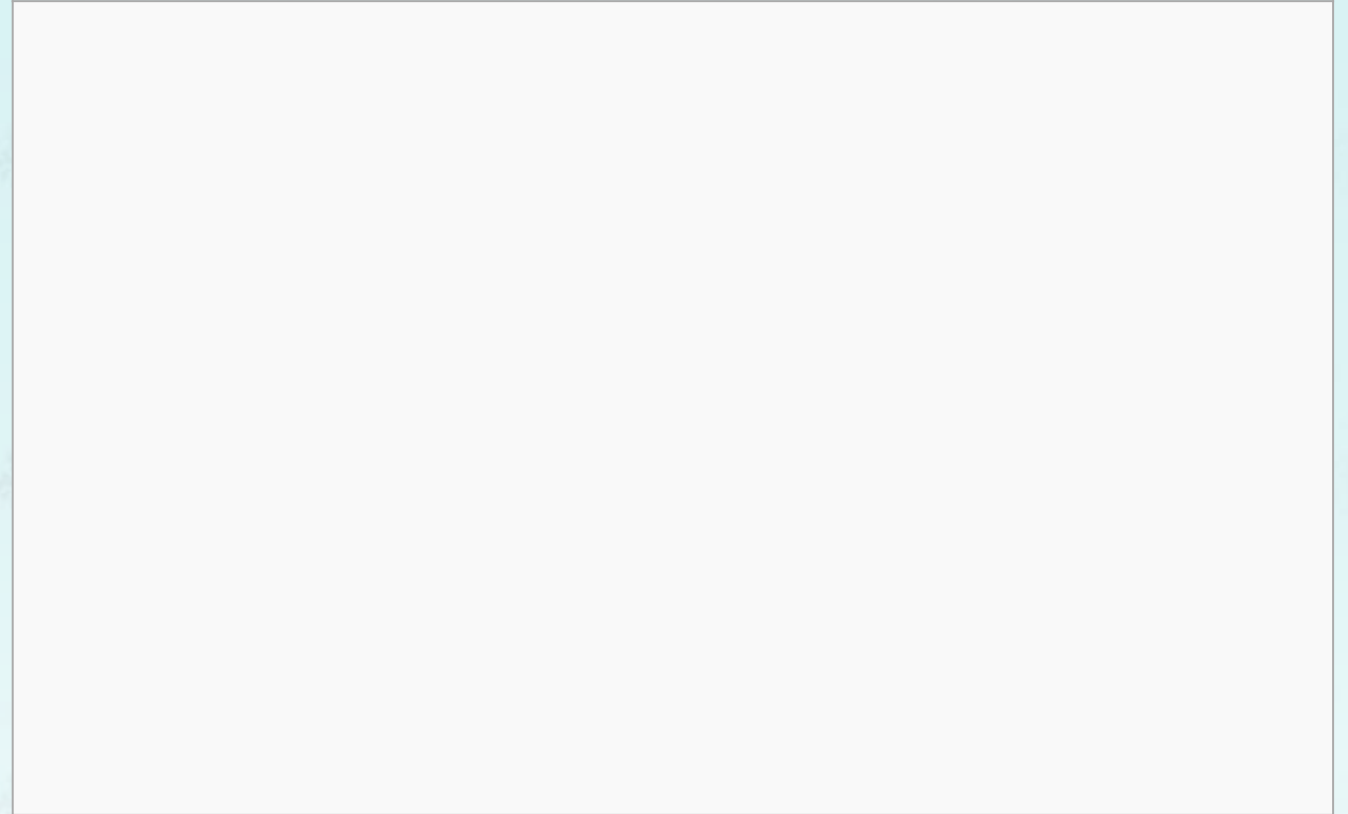   - Stack Concept
     - STL stack class
   - Stack Implementations
     - Using Fixed Array
     - Using Dynamic Array
     - Using Vector
     - Using STL Template
2. Queue

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

17

Stack: version.5T – using a vector<> in C++ STL

stack5.cpp

Compare these two program segments and see how to use `Templates` in C++ for generic programming.

Write a main() function to test STL stack and observe the capacity

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept, Handong Global University