Attendance Check

Problem set

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

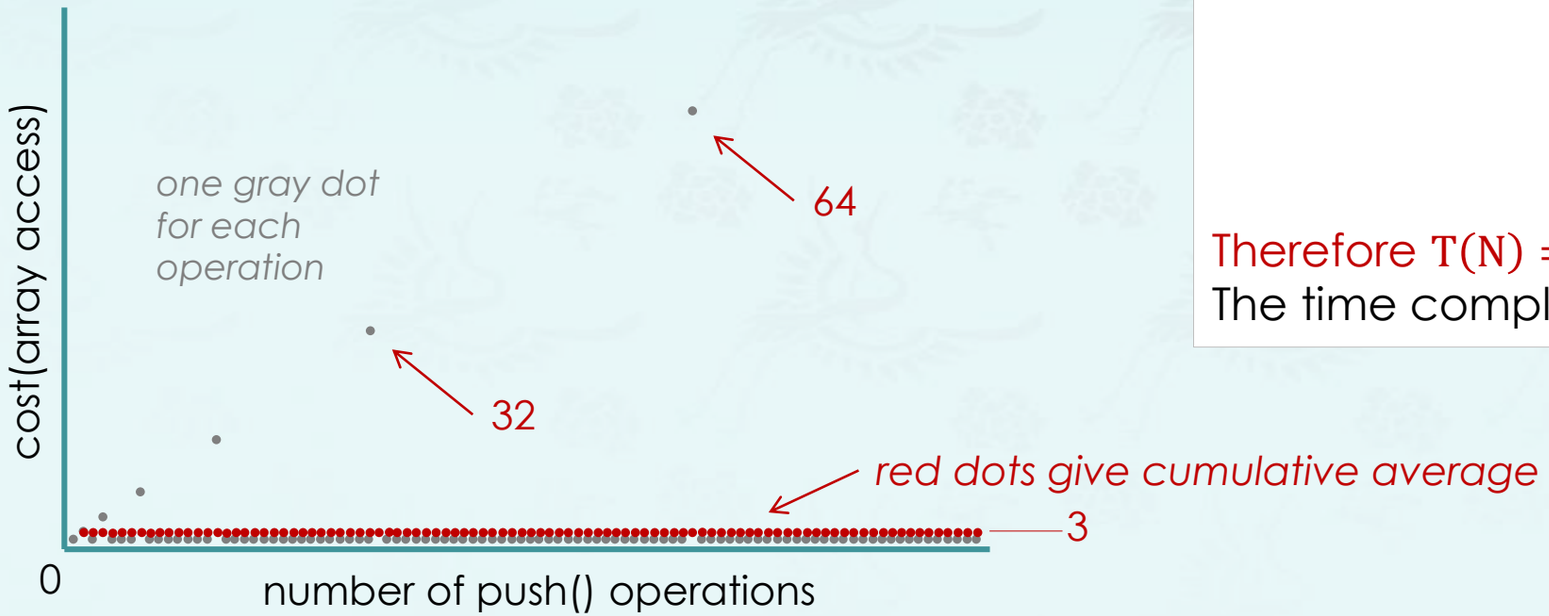
Stack: Using dynamic arrays

- **Q.** Cost of inserting first N items by resize(capacity * 2)?
- **A.** $\mathbf{T(N) = 1 + (1 + 2 + 4 + 8 + \ \ldots + N)}$

$$1 + a + a^2 + \ a^3 + \cdots + a^n = \frac{a^{n+1} - 1}{a - 1}$$

$$1 + 2 + 4 + \cdots + 2^n = \frac{2^{n+1} - 1}{2 - 1} = 2^{n+1} - 1$$

Therefore $T(N) = N + (1 + 2 + 4 + \cdots + N) = ?$
The time complexity of the algorithm is O(n).



Attendance Check

Problem set

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

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

stack1_stl.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";
}
```

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

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

stack1_stl.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:

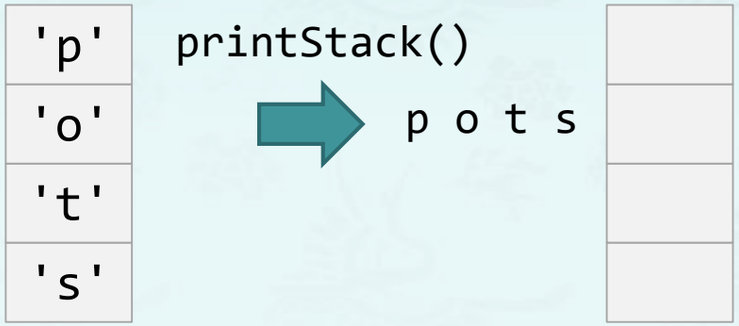
- Test this driver with "**int list[]**".
- Change a few places as needed.
- Fix `printStack()` not to empty the stack.
- Add **printStack_fromBottom()**.
- Be familiar with the concepts of stack.

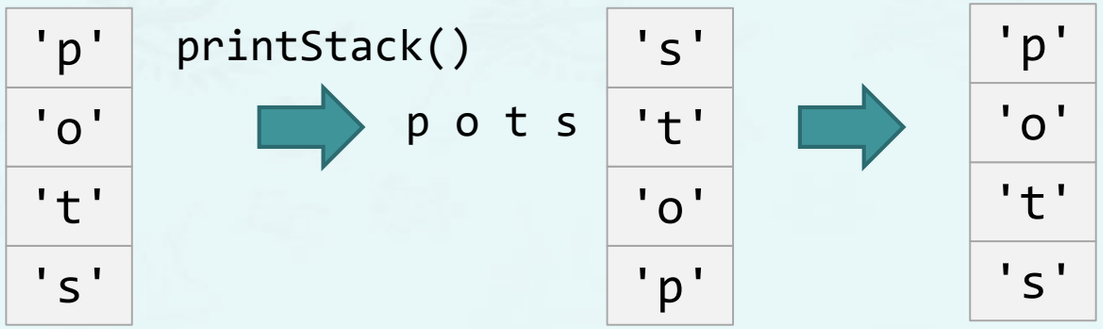
Stack: version.1 – using a stack class in C++ STL

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();
  }
}
```



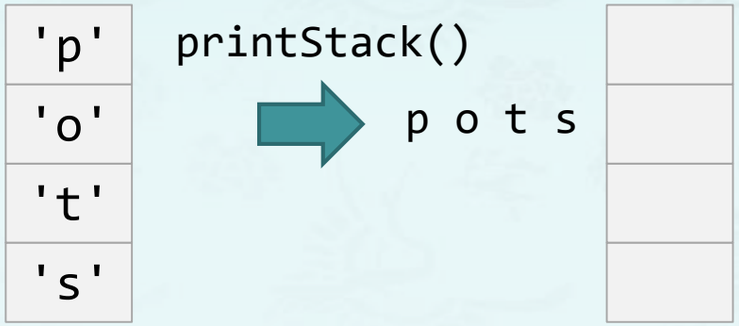


Stack: version.1 – using a stack class in C++ STL

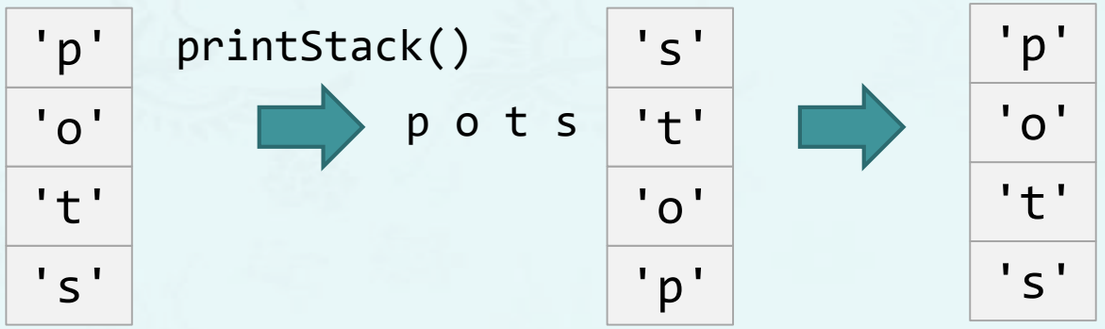
stack1_stl.cpp

- 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
```



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

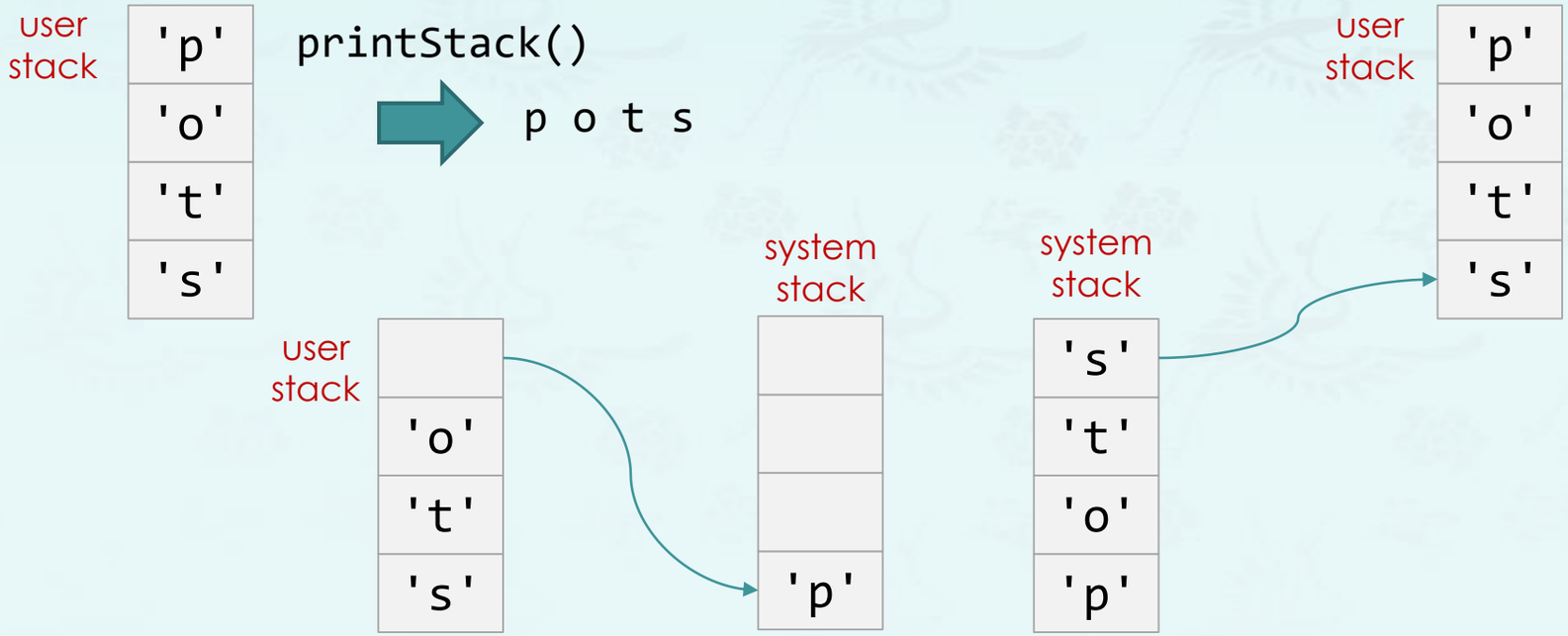
stack1_stl.cpp

- 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
```



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

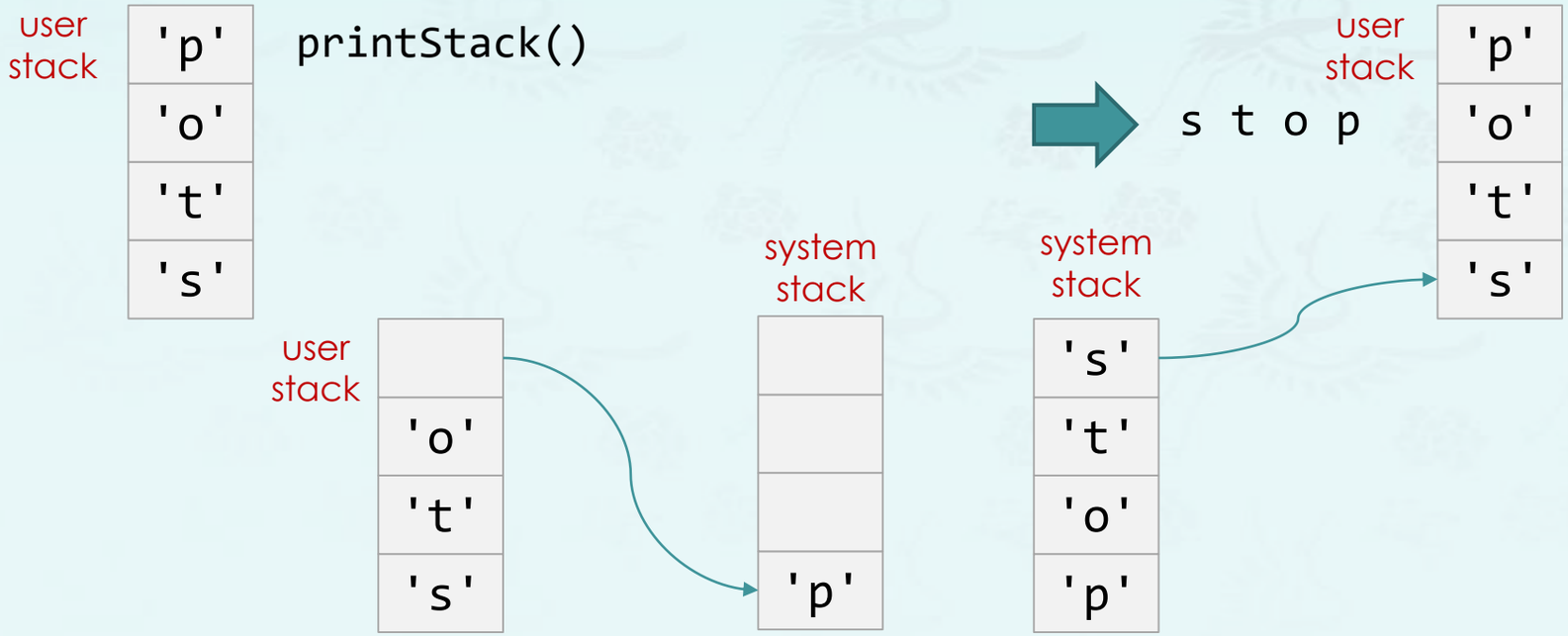
stack1_stl.cpp

- 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 here – top & pop & recursive call & print/push

} // recursion version
```



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

stack1_stl.cpp

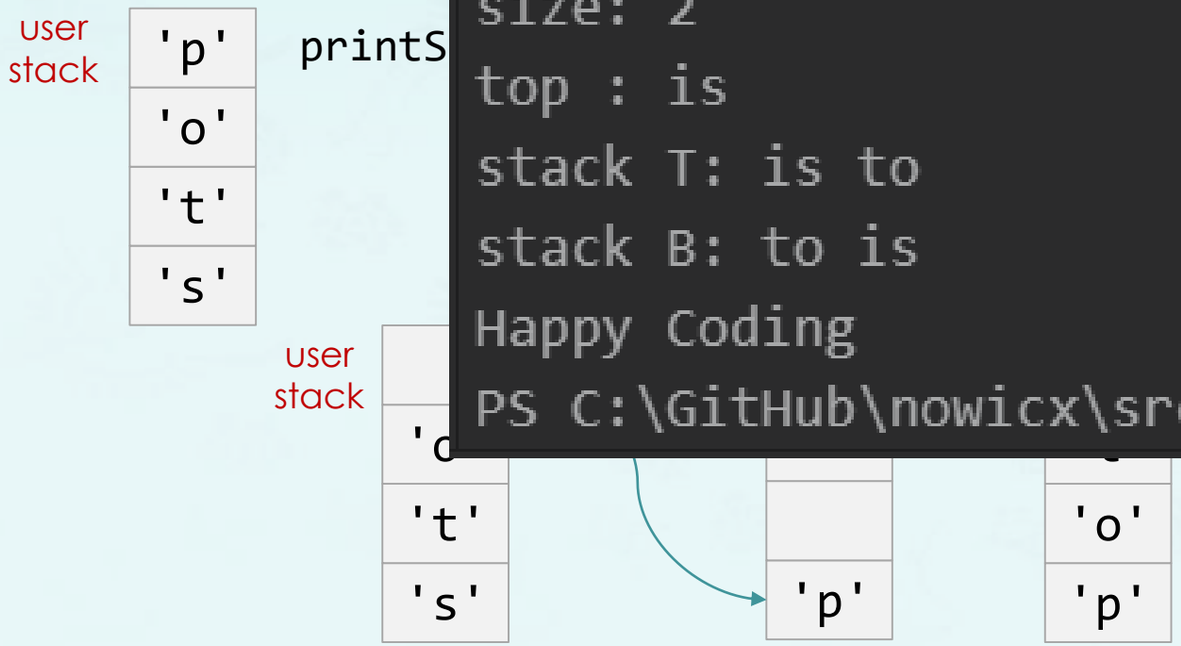
- Using recursion, print stack items from bottom to top.
  - Utilize the fact that the recursion uses the **system stack**.

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

  // your code

} // recursion
```

```
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>
```



Stack: version.2 – using a fixed size array

stack2_arr.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.

```
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

Stack: version.2 – using a fixed size array

stack2_arr.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)

```
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

ttom(stack s) {
```

N is not decremented

use N and incremented
N points an empty slot

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.

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
}
```

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;
```

```
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) {
```

```
(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: 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) {
  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.
- Test it with **string** data type.
- Create **stack4i_arr.app** such that it can handle **int** data type.
- Test it with **int** data type.

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

stack4_vecT.cpp

stack4_vec.cpp

```
struct Stack {
  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.

```
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();
}
```

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

stack4_vecT.cpp

stack4_vec.cpp

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

void free(stack s) {
  delete s;
}

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

```
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[illegible] 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.

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, The due date is yet to be determined(TBD).
- Grade:
  - step 3: 1.0 point
  - step 4: 1.0 point
  - step 5: 2.0 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