Hands-on Activity 4.1	
Stacks	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** 08 - 26 - 25
Section: CPE21S4	**Date Submitted:** 08 - 26 - 25
Name(s): Santiago, David Owen A.	**Instructor:** Engr. Jimlord Quejado

6. Output

Table 4-1

```cpp
#include <iostream>
#include <stack> // Calling Stack from the STL
using namespace std;

int main() {

   stack<int> newStack;
   newStack.push(3); //Adds 3 to the stack
   newStack.push(8);
   newStack.push(15);
   // returns a boolean response depending on if the stack is empty or not
   cout << "Stack Empty? " << newStack.empty() << endl;
   // returns the size of the stack itself
   cout << "Stack Size: " << newStack.size() << endl;
   // returns the topmost element of the stack
   cout << "Top Element of the Stack: " << newStack.top() << endl;
   // removes the topmost element of the stack
   newStack.pop();
   cout << "Top Element of the Stack: " << newStack.top() << endl;
   cout << "Stack Size: " << newStack.size() << endl;
   return 0;
}
```

Output:

```
"D:\College\Year 2 CPE\Sem 1\CPE 010\stacks\cmake-build-debug\untitled.exe"
Stack Empty? 0
Stack Size: 3
Top Element of the Stack: 15
Top Element of the Stack: 8
Stack Size: 2

Process finished with exit code 0
```

Observation:

When using the STL stacks, we can directly access its functions like push, pop, and top. This is useful in comparison to implementing stacks through arrays or linked lists, which require function definition of each. Hence, using the stack STL can be useful if we want to keep our code clean, readable, and straightforward.

Table B.1 Stacks using arrays

Tasks:
Modify the code given above to include a function that will display all elements in the stack.
Provide a description of each operation provided.
Include your output in section 6.

New void function for printing:

Template:
I added a new template for the new display function

```
void display(); // new template for displaying
```

Void Function:
This new void function first checks if the stack is empty and returns a message if it's true. If it's not empty, then the function uses a for loop that iterates over each element in the array then prints them out.

```
void display() { // new function
   if(isEmpty()) {
       std::cout << "Stack is Empty." << std::endl; // returns empty if stack is empty
   }
   else {
       std::cout << "The elements of the stack are:" << std::endl;
       for(i = top; i >= 0; i--) { // for loop to display all elements
           std::cout << stack[i] <<", ";
       } std::cout << std::endl;
   }
}
```

Added option in switch case:
New option in the switch case statement

```
case 4: std::cout << isEmpty() << std::endl;
   break;
case 5: display(); // new option to display all elements
   break;
default: std::cout << "Invalid Choice." << std::endl;
```

Output:

```
"D:\College\Year 2 CPE\Sem 1\CPE 010\stacks\cmake-build-debug\untitled.exe"
Enter number of max elements for new stack:3

Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. Display
1
New Value:
9

Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. Display
1
New Value:
8

Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. Display
1
New Value:
7

Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. Display
4
0

Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. Display
5
The elements of the stack are:
7, 8, 9,

Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. Display
3
The element on the top of the stack is 7

Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. Display
```

Observation:

Since I included a void print function that iterates through each element of the array stack, I can then print each individual element individually, but not before checking if the stack is empty or not.

Table B.2. Stacks using Linked Lists

New Void Display function adjusted for linked list:

```
void display() {
   if (head == NULL) {
       std::cout << "Stack is Empty." << std::endl;
       return;
   }
   Node *temp;
   temp = head;
   while(temp != NULL) {
       std::cout << temp->data << " ";
       temp = temp->next;
   } std::cout << std::endl;
}
```

In main:

```
std::cout << "Elements of the stack: ";
display();
std::cout << std::endl;
```

Output:

```
"D:\College\Year 2 CPE\Sem 1\CPE 010\stacks\cmake-build-debug\untitled.exe"
After the first PUSH top of stack is: Top of Stack: 1
After the second PUSH top of stack is: Top of Stack: 5
After the third PUSH top of stack is: Top of Stack: 10
Elements of the stack: 10 5 1

After the first POP operation, top of stack is: Top of Stack: 5
After the second POP operation, top of stack: Top of Stack: 1
After the third POP operation, top of stack: Stack is Empty.

Process finished with exit code 0
```

Observation:

For a linked list, the implementation is different because of the use of pointers. It first begins by checking if the stack is empty or not. To access each element, a temporary node is created that serves as the head. Then, the function loops through each node's next data then prints it out and the temp node is updated to be the next node until the temp becomes null, which terminates the loop, successfully printing each element.

7. Supplementary Activity

Array Implementation:

```
#include <iostream>
#include <string>

const size_t maxCap = 100;
char stack[maxCap];
int top = -1;
int newData;

// stack operations (array)
bool isEmpty() { return top == -1; }
bool isFull() { return top == (int)maxCap - 1; }

void push(char c) {
   if (isFull()) {
       std::cout << "Stack Overflow." << std::endl;
       return;
   }
   stack[++top] = c;
}
char pop() {
   if (isEmpty()) {
       std::cout << "Stack Underflow." << std::endl;
       return '\0';
   }
   return stack[top--];
}
void Top() {
   if (isEmpty()) {
       std::cout << "Stack is Empty." << std::endl;
       return;
   }
   std::cout << "The element on the top of the stack is " << stack[top] << std::endl;
}
void display() {
   if (isEmpty()) {
       std::cout << "Stack is Empty." << std::endl;
   } else {
       std::cout << "The elements of the stack are:" << std::endl;
       for (int i = top; i >= 0; i--) {
           std::cout << stack[i] << ", ";
       }
       std::cout << std::endl;
   }
}

// symbol checking
bool isMatchingPair(char open, char close) {
   return (open == '(' && close == ')') ||
          (open == '{' && close == '}') ||
          (open == '[' && close == ']');
}

// check expression validity
bool checkValid(std::string expr) {
   top = -1;
   int pos = 0;
```

```
    for (char c : expr) {
        pos++;
        if (c == '(' || c == '{' || c == '[') {
            push(c);
        } else if (c == ')' || c == '}' || c == ']') {
            if (isEmpty()) {
                std::cout << "Error at position " << pos
                     << ": closing '" << c
                     << "' found but stack is empty." << std::endl;
                return false;
            }
            char topChar = pop();
            if (!isMatchingPair(topChar, c)) {
                std::cout << "Error at position " << pos
                     << ": expected match for '" << topChar
                     << "' but found '" << c << "'." << std::endl;
                return false;
            }
        }
    }
    if (!isEmpty()) {
        std::cout << "Error: stack not empty at end. Unmatched symbols remain." <<
std::endl;
        return false;
    }
    return true;
}

int main() {
    int choice;
    std::string input;

    while (true) {
        std::cout << "\nStack Operations:" << std::endl;
        std::cout << "1. PUSH (char)" << std::endl;
        std::cout << "2. POP" << std::endl;
        std::cout << "3. TOP" << std::endl;
        std::cout << "4. isEMPTY" << std::endl;
        std::cout << "5. Display" << std::endl;
        std::cout << "6. Check Expression Validity" << std::endl;
        std::cout << "Choice: ";
        std::cin >> choice;
        std::cin.ignore();

        switch (choice) {
            case 1: {
                char val;
                std::cout << "Enter character to push: ";
                std::cin >> val;
                push(val);
                break;
            }
            case 2: {
                char val = pop();
                if (val != '\0') std::cout << "Popped: " << val << std::endl;
                break;
            }
            case 3:
                Top();
                break;
            case 4:
                std::cout << (isEmpty() ? "Stack is Empty." : "Stack is Not Empty.") <<
```

```
std::endl;
                break;
            case 5:
                display();
                break;
            case 6:
                std::cout << "Enter expression: ";
                getline(std::cin, input);
                if (checkValid(input))
                    std::cout << "Expression is Valid." << std::endl;
                else
                    std::cout << "Expression is Not Valid." << std::endl;
                break;
            default:
                std::cout << "Invalid Choice." << std::endl;
        }
    }
    return 0;
}
```

Sample Output:

```
"D:\College\Year 2 CPE\Sem 1\CPE 010\stacks\cmake-build-debug\untitled.exe"

Stack Operations:
1. PUSH (char)
2. POP
3. TOP
4. isEMPTY
5. Display
6. Check Expression Validity
Choice:6
 Enter expression:(A + B) + (C - D)
 Expression is Valid.

Stack Operations:
1. PUSH (char)
2. POP
3. TOP
4. isEMPTY
5. Display
6. Check Expression Validity
Choice:
```

Linked List Implementation:

```
#include <iostream>
#include <string>

class Node {
public:
   char data;
   Node *next;
};
Node *head = NULL;

void push(char c);
char pop();
bool isEmpty();
bool isMatching(char open, char close);
void clearStack();
bool isValidExpression(const std::string &expr);
void display();
void enterElements();
void isValidPrint(std::string expr);

int main() {

   while (true) {
       int choice;
       std::string expr;
       std::cout << "\nLinked List Stack Operations: \n";
       std::cout << "Enter 1 to enter elements\n";
       std::cout << "Enter 2 to expression validity\n";
       std::cout << "Enter 3 to display elements\n";
       std::cout << "Enter 4 to stop\n";
       std::cout << "Enter choice: ";
       std::cin >> choice; std::cout <<"\n";
       std::cin.ignore();
       switch (choice) {

           case 1: // enter elements in stack
               enterElements();
               std::cout << "Current Stack is: "; display(); std::cout << std::endl;
               break;

           case 2: // check for expression validity
               std::cout << "Clearing stack for Expression Checking...\n";
               clearStack();

               std::cout << "Enter an expression: ";
               std::getline(std::cin, expr);

               isValidPrint(expr);
               break;

           case 3:
               std::cout << "Current Stack is: "; display(); std::cout << std::endl;
               break;

           case 4:
               std::cout << "Exiting...\n ";
               return 0;

           default:
```

```
                std::cout << "Invalid choice" << std::endl;
                break;

        }
    }

}

void push(char c) {
    Node *newNode = new Node;
    newNode->data = c;
    newNode->next = head;
    head = newNode;
}

char pop() {
    if (head == NULL) {
        return '\0';
    }
    char val = head->data;
    Node *temp = head;
    head = head->next;
    delete temp;
    return val;
}

bool isEmpty() {
    return head == NULL;
}

bool isMatching(char open, char close) {
    return (open == '(' && close == ')') ||
           (open == '{' && close == '}') ||
           (open == '[' && close == ']');
}

void clearStack() {
    while (!isEmpty()) {
        pop();
    }
}

bool isValidExpression(const std::string &expr) {
    clearStack();
    for (char c : expr) {
        if (c == '(' || c == '{' || c == '[') {
            push(c);
        } else if (c == ')' || c == '}' || c == ']') {
            if (isEmpty()) {
                clearStack();
                return false;
            }
            char topChar = pop();
            if (!isMatching(topChar, c)) {
                clearStack();
                return false;
            }
        }
    }
    return isEmpty();
}
```

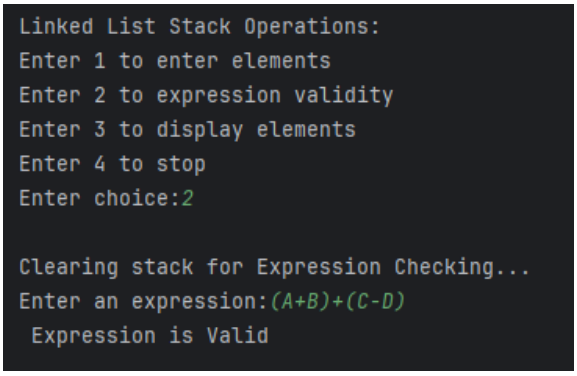
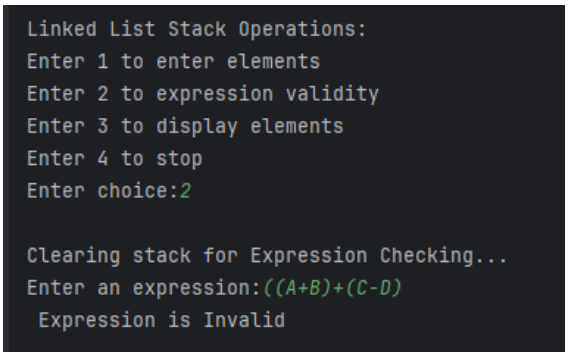
```
void display() {
   if (head == NULL) {
       std::cout << "Empty Stack" << std::endl;
       return;
   }
   Node *temp = head;
   while (temp != NULL) {
       std::cout << temp->data << " ";
       temp = temp->next;
   } std::cout << std::endl;
}

void enterElements() {
   char a; int b;
   std::cout<< "Enter number of elements: ";
   std::cin >> b;
   std::cin.ignore();

   for (int x = 0; x < b; x++) {
       std::cout << "Enter element " << x + 1 << ": ";
       std::cin >> a;
       push(a);
   }
   std::cout << std::endl;
}

void isValidPrint(std::string expr) {
   if (isValidExpression(expr)) {
       std::cout << "Expression is Valid" << std::endl;
   }
   else {
       std::cout << "Expression is Invalid" << std::endl;
   }
}
```

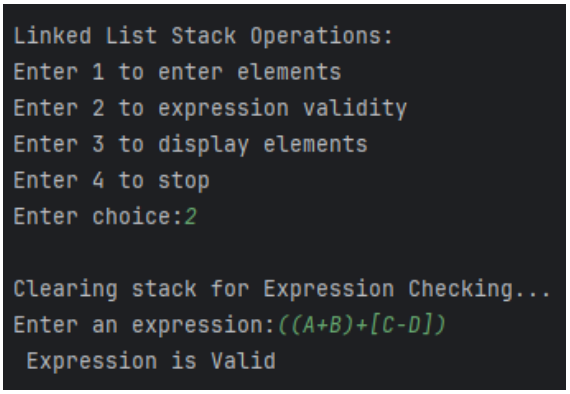
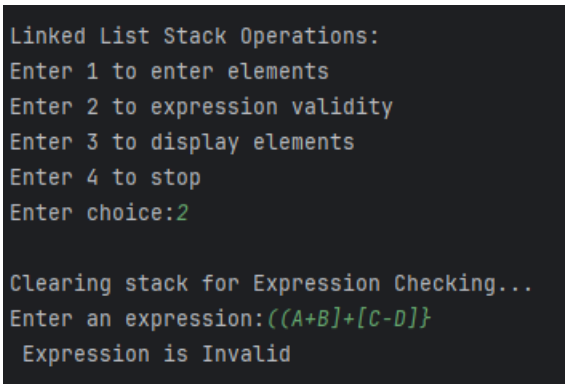
Sample Output:

```
"D:\College\Year 2 CPE\Sem 1\CPE 010\stacks\cmake-build-debug\untitled.exe"

Linked List Stack Operations:
Enter 1 to enter elements
Enter 2 to expression validity
Enter 3 to display elements
Enter 4 to stop
Enter choice:1

Enter number of elements:4
 Enter element 1:1
 Enter element 2:2
 Enter element 3:3
 Enter element 4:4

Current Stack is: 4 3 2 1


Linked List Stack Operations:
Enter 1 to enter elements
Enter 2 to expression validity
Enter 3 to display elements
Enter 4 to stop
Enter choice:3

Current Stack is: 4 3 2 1


Linked List Stack Operations:
Enter 1 to enter elements
Enter 2 to expression validity
Enter 3 to display elements
Enter 4 to stop
Enter choice:2

Clearing stack for Expression Checking...
Enter an expression:(A + B) + (C - D)
 Expression is Valid

Linked List Stack Operations:
Enter 1 to enter elements
Enter 2 to expression validity
Enter 3 to display elements
Enter 4 to stop
Enter choice:4

Exiting...

Process finished with exit code 0
```

For both implementations:

The programs are nested within a while loop where there are different choices given. Each choice is handled within a switch-case statement that does whatever function it is written. In the array implementation, there are options for direct stack operations like push, pop, etc. I added additional choices that display the elements of the stack and also initializes the valid expression checker. Meanwhile, on the linked list implementation utilizes more general choices—entering, displaying, then the valid expression checker along with an exit choice. For the valid expression checker, the user enters a full expression which is taken using the getline function to account for whitespaces. Once entered, it first checks each character (since the checker takes char arguments). If they aren't delimiters, they are ignored, else it writes opening delimiters into the stack, then once a closing delimiter is entered, the stack is then popped then compared to see if they match using the isValidExpression and isMatching functions. If they are matching, then they return true. Once all elements are popped out of the stack (which only happens when they have matching opening and ending delimiters), then the checker accepts its validity and prints a valid message. Otherwise, if there is still a remaining character in the stack, then the checker will print an invalid message.

Expression	Valid? (Y/N)	Output (Console Screenshot)	Analysis
(A+B)+(C-D)	Y		The checker begins by pushing the first opening delimiter (onto the stack. When it encounters the closing parenthesis), the top (is popped and correctly matched. Next, another opening parenthesis (is pushed, and later the closing parenthesis) appears, which also matches and is popped off. At the end of the expression, the stack is empty, meaning every opening delimiter has a proper closing pair. Therefore, the expression is valid.
((A+B)+(C-D)	N		The checker starts by pushing the first two opening delimiters (onto the stack. When it reaches the first closing parenthesis), the top (is popped and correctly matched. Then another opening parenthesis (is pushed, followed later by the closing parenthesis) that matches and pops it off. However, at the very end of the expression, there is still one unmatched (left in the stack. Since the stack is not empty after processing all characters, the expression is invalid.

((A+B)+[C-D])	Y		The checker begins by pushing the first two opening parentheses (into the stack. When it reaches the first closing parenthesis), the top (is popped and correctly matched. Next, it encounters another opening parenthesis (, which is pushed, followed by an opening square bracket [that is also pushed. When the closing square bracket] appears, it matches with the top [and is popped off. Finally, the last closing parenthesis) matches with the remaining (on top of the stack. At the end of the scan, the stack is empty, meaning all delimiters have been properly paired and closed, so the expression is valid.
((A+B]+[C-D]}	N		The checker goes through each character one by one. The first two characters are both opening parentheses (, so they are pushed into the stack in order. When the program reaches the closing bracket], it looks at the top of the stack, which contains (. Since (does not correctly match], the comparison fails. At this point, the checker concludes that the expression is invalid because the delimiters are not properly paired, so even without processing the remaining characters, the validity test is already rejected.

8. Conclusion

In conclusion, stacks, whether implemented using arrays or linked lists, are a highly versatile data structure with useful applications such as checking the validity of expressions. In the array implementation, stacks are straightforward. Since arrays do not dynamically allocate memory, they have fixed size limitations. Regardless of that limitation, they are efficient for direct access for push and pop operations. On the other hand, the linked list implementation offers dynamic memory allocation, allowing flexibility in the number of elements without predefined limits like its size. Both methods maintain the same principle of Last-In, First-Out (LIFO) for stacks, which is useful in problems like delimiter matching where the most recent opening symbol must be matched first with a closing symbol. This makes stacks an ideal choice for

expression validity checking, ensuring every opening delimiter is properly closed in the correct order. In conclusion, stacks can be utilized in versatile ways while interconnecting other data structures like linked lists and arrays in implementing them.

9. Assessment Rubric