# Tugas Praktikum Stack

Praktikum Algoritma dan Struktur Data


**Oleh:**

Samudero Dirgantoro / 5223600016

**Program Studi D4 Teknologi Game**

**Departemen Teknologi Multimedia Kreatif**

**Politeknik Elektronika Negeri Surabaya**


**2023/2024**

## A. Percobaan

1. (8.3) Implement stack using a linked list.

- C# ke C++:

```
#include <iostream>
class ListStack {
   struct Node {
      int value;
      Node* next;
      Node(int v, Node* n) : value(v), next(n) {}};
   Node* head = NULL;
   int count = 0;
public:
   int size() const {
      return count;}
   bool Empty() const {
      return count == 0;}
   int peek() const {
      if (Empty()) {
         throw
std::runtime_error("ListStackEmptyException");}
      return head->value;}
   void Push(int value) {
      head = new Node(value, head);
      count++;}
   int Pop() {
      if (Empty()) {
         throw
std::runtime_error("ListStackEmptyException");}
      int value = head->value;
      Node* temp = head;
      head = head->next;
```

```
        delete temp;
        count--;
        return value;}
    void insertAtBottom(int value) {
        if (Empty()) {
            Push(value);
        } else {
            int temp = Pop();
            insertAtBottom(value);
            Push(temp);}}
    void print() const {
        Node* temp = head;
        while (temp != NULL) {
            std::cout << temp->value << " ";
            temp = temp->next;}}};

int main() {
    ListStack s;
    for (int i = 1; i <= 100; i++)
        s.Push(i);
    for (int i = 1; i <= 50; i++)
        s.Pop();
    s.print();}
```

- Hal yang Berubah: Struktur data yang digunakan untuk mengelola tumpukan tetap sama, dengan penggunaan class ListStack, dalam C# maupun C++. Tetapi dalam C++, struktur node dimasukkan ke dalam class ListStack, sedangkan dalam C# dideklarasikan secara terpisah. Kode Private dan Internal dihilangkan karena sudah tidak diperlukan lagi. Perubahan InvalidOperationException menjadi std::runtime_error.

Perubahan fungsi print Console.Write menjadi std::cout.

- Rangkuman: Kode tersebut merupakan implementasi struktur data tumpukan (stack) dalam bahasa C++. Kode menyediakan fungsi-fungsi dasar seperti Push, Pop, dan peek, serta fungsi lainnya seperti size, Empty, insertAtBottom, dan print untuk mengelola dan memanipulasi tumpukan. Fungsi main bertujuan untuk menguji fungsionalitas tumpukan yang telah dibuat. Dalam contoh pada modul stack C# . Sekarang pada C++ tumpukan diisi dengan angka dari 1 hingga 100 menggunakan loop for, kemudian 50 elemen teratas dihapus menggunakan loop for lainnya. Hasilnya kemudian dicetak menggunakan fungsi print.

2. (8.4) Problems in Stack. Balanced Parenthesis
    - C# ke C++:

```
#include <iostream>
#include <stack>
#include <string>
bool isBalancedParenthesis(const std::string& expn) {
   std::stack<char> stk;
   for (char ch : expn) {
      switch (ch) {
         case '{':
         case '[':
         case '(':
            stk.push(ch);
            break;
         case '}':
            if (stk.empty() || stk.top() != '{') {
               return false;}
            stk.pop();
```

```
            break;
        case ']':
            if (stk.empty() || stk.top() != '[') {
                return false;}
            stk.pop();
            break;
        case ')':
            if (stk.empty() || stk.top() != '(') {
                return false;}
            stk.pop();
            break;}}
    return stk.empty();}
int main() {
    std::string expn = "{()}"; //Hasil true, akan menjadi
false jika menjadi terbalik atau })({}
    bool value = isBalancedParenthesis(expn);
    std::cout << "Given Expn: " << expn << std::endl;
    std::cout << "Result after isParenthesisMatched: " <<
std::boolalpha << value << "\n";}
```

- Hal yang Berubah: Mengganti Stack<char> dengan std::stack<char>. Menggunakan std::cout dari daripada Console.WriteLine seperti dalam C#. Nilai True False pada C++ dicetak menggunakan std::boolalpha, sedangkan dalam C# tidak.
- Rangkuman: Tujuan dari kode tersebut adalah untuk memeriksa apakah urutan tanda kurung sudah seimbang dan sesuai. Kode tersebut menggunakan struktur stack untuk memeriksa setiap tanda kurung buka memiliki pasangan tanda kurung tutup yang sesuai dengannya. Jika tanda kurung sudah sesuai, fungsi isBalancedParenthesis akan mengembalikan nilai

true, dan sebaliknya jika tidak seimbang akan mengembalikan nilai false.

3. (8.5) Infix-to-Postfix Conversion
    - C# ke C++:

```
#include <iostream>
#include <stack>
#include <string>
int precedence(char op) {
   switch(op) {
      case '+':
      case '-':
         return 1;
      case '*':
      case '/':
      case '%':
         return 2;
      case '^':
         return 3;
      default:
         return -1;}}
std::string infixToPostfix(const std::string& expn) {
   std::stack<char> stk;
   std::string output = "";
   char temp;
   for (char ch : expn) {
      if (ch >= '0' && ch <= '9') {
         output += ch;
      } else {
         switch (ch) {
            case '+':
            case '-':
            case '*':
```

```
            case '/':
            case '%':
            case '^':
               while (!stk.empty() && precedence(ch) <=
precedence(stk.top())) {
                  temp = stk.top();
                  stk.pop();
                  output += " ";
                  output += temp;}
               stk.push(ch);
               output += " ";
               break;
            case '(':
               stk.push(ch);
               break;
            case ')':
               while (!stk.empty() && (temp = stk.top()) !=
'(') {
                  stk.pop();
                  output += " ";
                  output += temp;
                  output += " ";}
               stk.pop();
               break;}}}
   while (!stk.empty()) {
      temp = stk.top();
      stk.pop();
      output += " ";
      output += temp;}
   return output;}
int main() {
   std::string expn = "10+((3))*5/(16-4)";
```

```
    std::string value = infixToPostfix(expn);
    std::cout << "Infix Expn: " << expn << "\n" << "Postfix
Expn: " << value;}
```

- Hal yang Berubah: Penggunaan char[] dan Stack<char> diganti dengan std::string dan std::stack<char> dalam C++. Fungsi precedence() ditambahkan dalam kode C++. Penambahan fungsi InfixtoPostfix. Kemudian proses pencetakan hasil yang sebelumnya Console.WriteLine() di C#, diubah menjadi std::cout.
- Rangkuman: Kode tersebut menggunakan struktur data stack untuk menyimpan operator-operator yang ditemukan dalam infix. Fungsi precedence menentukan prioritas operator. Selanjutnya, fungsi infixToPostfix menerima ekspresi infix sebagai string dan mengembalikan ekspresi postfix yang sudah dikonversi. Fungsi main digunakan untuk menguji fungsi konversi dengan memberikan ekspresi infix "10+((3))*5/(16-4)" dan mencetak hasil konversinya.

4. (8.6) Infix-to-Prefix Conversion

- C# ke C++:

```
#include <iostream>
#include <string>
void ReverseString(std::string& expn) {
   int lower = 0;
   int upper = expn.length() - 1;
   char tempChar;
   while (lower < upper) {
      tempChar = expn[lower];
      expn[lower] = expn[upper];
      expn[upper] = tempChar;
      lower++;
      upper--;}}
```

```
void ReplaceParanthesis(std::string& a) {
    for (char& ch : a) {
        if (ch == '(') {
            ch = ')';
        } else if (ch == ')') {
            ch = '(';}}}
std::string InfixToPostfix(const std::string& expn) {
    std::string output = "";
    for (char ch : expn) {
        if (ch == '(') {
            output += '(';
        } else if (ch == ')') {
            output += ')';
        } else {
            output += ch;}}
    return output;}
std::string InfixToPrefix(const std::string& expn) {
    std::string expnCopy = expn;
    ReverseString(expnCopy);
    ReplaceParanthesis(expnCopy);
    std::string postfixExpn = InfixToPostfix(expnCopy);
    ReverseString(postfixExpn);
    return postfixExpn;}
int main() {
    std::string expn = "10+((3))*5/(16-4)";
    std::string value = InfixToPrefix(expn);
    std::cout << "Infix Expn: " << expn << "\n";
    std::cout << "Prefix Expn: " << value << "\n"}
```

- Hal yang Berubah: Mengganti loop foreachmenjadi loop for dengan indeks. Dalam C# mengubah tanda kurung dengan fungsi ReplaceParanthesis, sedangkan C++ menyalin tanda kurung tanpa perubahan dengan

fungsi InfixToPostfix. Dalam pemanggilan fungsi rekursif, C# menggunakan rekursi langsung pada fungsi InfixToPostfix, sedangkan C++ memanggil fungsi InfixToPostfix dalam satu level. Output Console.WriteLine diganti menjadi std::cout pada C++.

- Rangkuman: Kode tersebut adalah sebuah program untuk mengubah infix menjadi prefix. ReplaceParanthesis, mengganti tanda kurung buka dengan tutup dan sebaliknya. Memodifikasi dengan membalikkan urutan karakter menggunakan fungsi reverseString. Selanjutnya, ekspresi yang sudah dimodifikasi dikonversi menjadi postfix dengan fungsi infixToPostfix yang tersedia. Hasil akhirnya ditampilkan dalam fungsi Main dengan mencetak hasil infix dan prefix.

5. (8.7) Postfix Evaluate
    - C# ke C++:

```
#include <iostream>
#include <stack>
#include <sstream>
int postfixEvaluate(const std::string& expn) {
   std::stack<int> stk;
   std::istringstream iss(expn);
   std::string token;
   while (iss >> token) {
      if (token == "+" || token == "-" || token == "*" ||
token == "/") {
         int num2 = stk.top(); stk.pop();
         int num1 = stk.top(); stk.pop();

         if (token == "+") stk.push(num1 + num2);
         else if (token == "-") stk.push(num1 - num2);
```

```
            else if (token == "*") stk.push(num1 * num2);
            else if (token == "/") stk.push(num1 / num2);
        } else {
            stk.push(std::stoi(token));}}
    return stk.top();}
int main() {
    std::string expn = "6 5 2 3 + 8 * + 3 + *";
    int value = postfixEvaluate(expn);
    std::cout << "Given Postfix Expn: " << expn << "\n";
    std::cout << "Result after Evaluation: " << value <<
"\n";}
```

- Hal yang Berubah: Perubahan kode hampir mirip dengan program sebelumnya, yaitu seperti tipe data char[] dan Stack<char> diganti menjadi std::string dan std::stack<char>. Kemudian, fungsi precedence() ditambahkan untuk menentukan prioritas operator, sama seperti sebelumnya. Selanjutnya hasil dievaluasi dan dicetak menggunakan std::cout, berbeda dengan C# yang menggunakan Console.WriteLine().
- Rangkuman: Kode tersebut merupakan implementasi evaluasi ekspresi postfix. Postfix dipisahkan menjadi token-token menggunakan spasi. Kemudian, setiap token dievaluasi satu per satu. Jika token merupakan operator “+”, “-“, “*”, “/”, dua angka terakhir diambil dari tumpukan (stack), dievaluasi dengan operator tersebut, dan hasilnya dimasukkan kembali ke dalam stack. Dalam fungsi Main(), hasil postfix akan diberikan dan hasil evaluasi, kemudian dicetak ke konsol.

6. (8.8) Reverse Stack/Palindrome
    - Implementasi:

```
#include <iostream>
#include <stack>
```

```
#include <string>
template<typename T>
void reverseStack(std::stack<T>& stk) {
   if (stk.empty()) {
      return;
   } else {
      T value = stk.top();
      stk.pop();
      reverseStack(stk);
      stk.push(value);}}
bool isPalindrome(const std::string& str) {
   std::stack<char> charStack;
   for (char ch : str) {
      charStack.push(ch);}
   reverseStack(charStack);
   for (char ch : str) {
      if (ch != charStack.top()) {
         return false;}
      charStack.pop();}
   return true;}
int main() {
   std::string str;
   std::cout << "cek polindrom: ";
   std::cin >> str;
   std::cout <<palindrom(str) << "\n";}
```

- Rangkuman: Program ini adalah implementasi deteksi palindrom menggunakan struktur data stack dan menggunakan referensi fungsi reverseStack, untuk membalikkan isi dari sebuah stack. Fungsi palindrom menerima string sebagai input dan mengembalikan true(1) jika string tersebut adalah palindrom, dan false(0) jika tidak.