BÁO CÁO THỰC HIỆN ĐỒ ÁN

Môn: Cấu trúc dữ liệu và giải thuật
Họ và tên SV: Đặng Đức Trường
MSSV: 20880108

Thiết lập lớp Node

Mục đích

Lớp Node được dùng để tạo thành các phần tử cấu thành nên cấu trúc dữ liệu dạng danh sách liên kết (DSLK).

Cấu trúc

Các thuộc tính:

- **T data**: là thuộc tính kiểu Generic dùng để lưu dữ liệu của 1 node
- **Node<T>? next**: là thuộc tính tham chiếu đến node kế tiếp của node đang được xét, thuộc tính này có thể null trong trường hợp đây là 1 node độc lập hoặc có thể hiểu đây là 1 DSLK có duy nhất 1 phần tử.

Các phương thức:

- **Node()** và **Node(T data)**: các hàm khởi tạo
- **T Data{}** và **Node<T> Next{}**: hàm getter và setter của các thuộc tính
- **Boolean hasNext()**: trả về true nếu node hiện tại có tham chiếu đến 1 node kế tiếp, ngược lại sẽ trả về false.

Thiết lập lớp cQueue

Mục đích

Đây là lớp dùng để tạo ra các data object dạng danh sách hàng chờ với nguyên tắc: đối tượng được nạp vào hàng chờ trước sẽ được lấy ra trước (FIFO).

Cấu trúc

Các thuộc tính

- **int count**: dùng để đếm số phần tử trong hàng chờ
- **Node<T>? head**: node ở đầu danh sách
- **Node<T>? tail**: node ở cuối danh sách

Các phương thức

- **int Count {}**: hàm getter của thuộc tính count
- **cQueue()**: hàm khởi tạo của queue
- **void Enqueue(T value)**: hàm thêm 1 giá trị vào queue. Giá trị được thêm vào sẽ được "đính" vào 1 node mới và nối tiếp node mới đó vào tail của queue, đồng thời node mới đó sẽ đóng vai trò làm tail mới của queue.
- **void Dequeue()**: hàm trả về giá trị của node ở đầu hàng chờ và xóa node đó khỏi hàng chờ.
- **T Peek()**: hàm trả về giá trị của node tại đầu hàng chờ nhưng không xóa node đó khỏi hàng chờ

- **Boolean isEmpty()**: trả về true nếu danh sách hàng chờ không có node nào, ngược lại sẽ trả về false.
- **cQueue<T> Copy()**: hàm sao chép queue thành 1 queue mới

Thiết lập lớp cStack

Mục đích

Đây là lớp dùng để tạo ra các data object dạng danh sách ngăn xếp với nguyên tắc: đối tượng được nạp vào hàng chờ trước nhất sẽ được lấy ra sau cùng (FILO).

Cấu trúc

Các thuộc tính

- **int count**: dùng để đếm số phần tử trong hàng chờ
- **Node<T>? top**: node ở đỉnh (node được thêm vào sau cùng) của ngăn xếp

Các phương thức

- **int Count {}**: hàm getter của thuộc tính count
- **cStack()**: hàm khởi tạo của stack
- **void Push(T value)**: hàm thêm 1 giá trị vào stack. Giá trị được thêm vào sẽ được "đính" vào 1 node mới và nối tiếp node mới đó vào top của stack, đồng thời node mới đó sẽ đóng vai trò làm top mới của stack.
- **void Pop()**: hàm trả về giá trị của node ở đầu ngăn xếp và xóa node đó khỏi ngăn xếp.
- **T Peek()**: hàm trả về giá trị của node tại đầu ngăn xếp nhưng không xóa node đó khỏi ngăn xếp
- **Boolean isEmpty()**: trả về true nếu ngăn xếp không có node nào, ngược lại sẽ trả về false.
- **cStack<T> Reverse()**: hàm sao chép và đảo ngược thứ tự 1 stack thành 1 stack mới.

Thiết lập lớp PolishNotationBuilder

Mục đích

Đây là lớp dùng để chuyển đổi 1 biểu thức ở dạng trung tố (infix) thành biểu thức hậu tố (postfix).

Các phương thức

- **static cQueue<string> postfixBuild(string original)**: phương thức này nhận vào chuỗi biểu thức gốc viết ở dạng trung tố, sau đó sẽ chuyển đổi và trả về biểu thức tương đương ở dạng hậu tố, ứng dụng cStack và cQueue.
- **static int opertRankCheck(char oprt)**: hàm trả về thứ tự ưu tiên của một toán tử, thứ tự này sẽ quyết định vị trí của toán tử đó trong chuỗi biểu thức hậu tố. Hàm này là hàm hỗ trợ của hàm postfixBuild.

Thiết lập lớp Calculator

Mục đích

Thực hiện tính giá trị của biểu thức ở dạng hậu tố

Các phương thức

- **static double execute(string expression)**: thực hiện tính toán giá trị của biểu thức sau khi biểu thức đó được chuyển từ dạng trung tố sang hậu tố nhờ hàm PolishNotationBuilder.postfixBuild. Sau đó dùng 2 hàm là calculate và doFactorial để tính giá trị của biểu thức.
- **static double calculate(char oprt, double first_operand, double second_operand)**: hàm tính giá trị của phép tính có 2 đối số.
- **static int doFactorial(int operand)**: hàm tính giá trị giai thừa.

Thiết lập các lớp phụ trợ

Mục đích

- **IOHelper**: lớp static chứa các hàm hỗ trợ đọc file và ghi kết quả ra file
- **Utils**: chứa các hàm tiện ích dùng chung

Các phương thức

IOHelper

- **static string[] readDataFromFile(string fileName)**: hàm hỗ trợ đọc các biểu thức được lưu trong file được chỉ định và lưu thành 1 mảng giá trị các biểu thức.
- **static void writeToFile(string fileName, double[] data)**: hàm ghi giá trị của mảng ra file với tên được chỉ định.

Utils

- **static bool isOperator(char check)**: hàm kiểm tra 1 ký tự truyền vào có phải là 1 toán tử hợp lệ hay không.
- **static bool isNumeric(char check)**: hàm kiểm tra 1 ký tự truyền vào có phải là 1 ký số hay không.
- **static bool isBracket(char check)**: hàm kiểm tra 1 ký tự truyền vào có phải là dấu ngoặc mở hoặc đóng hay không.

Hướng dẫn sử dụng

Bước 1: Đặt file “BIEUTHUC.inp” có chứa **dữ liệu hợp lệ** vào thư mục “~\20880108\Data”

Bước 2: Mở Visual Studio IDE (khuyến nghị phiên bản 2019 trở lên), sau đó tiến hành import project vào IDE.

Bước 3: Clean và chạy project

Bước 4: Mở file KETQUA.out tại thư mục “~\20880108\Data” để xem kết quả thực thi.

Ghi chú: trong phạm vi đồ án này, mọi trường hợp giá trị đầu vào không hợp lệ sẽ được bỏ qua và chỉ tập trung chủ yếu vào giải thuật tính toán giá trị biểu thức.