```
/******************************************************************************

Welcome to GDB Online.
  GDB online is an online compiler and debugger tool for C, C++, Python, PHP, Ruby,
  C#, OCaml, VB, Perl, Swift, Prolog, Javascript, Pascal, COBOL, HTML, CSS, JS
  Code, Compile, Run and Debug online from anywhere in world.

*******************************************************************************/
/*
Concatenate two input singly linked lists l1 and l2 into a new output linked list
that contains all the nodes of
l1 followed by all the nodes of l2.
*/

#include <iostream>
#include <string>

// Forward declaration
template <typename T> class SLinkedList;
template <typename T> void concat(SLinkedList<T>& l1, SLinkedList<T>& l2,
SLinkedList<T>& lout);

template <typename T>
class SNode {
private:
T elem;
SNode<T>* next;
friend class SLinkedList<T>; // Added "class" keyword for proper friend declaration
friend void concat<T>(SLinkedList<T>& l1, SLinkedList<T>& l2, SLinkedList<T>&
lout);
public:
SNode() : next(nullptr) {}
};

template <typename T>
class SLinkedList {
public:
SLinkedList();
~SLinkedList();
bool empty() const;
const T& front() const;
```

```
void addFront(const T& e);
void addBack(const T& e);
void removeFront();
friend void concat<T>(SLinkedList<T>& l1, SLinkedList<T>& l2, 
  SLinkedList<T>&
lout);
private:
SNode<T>* head;
};

template <typename T>
void concat(SLinkedList<T>& l1, SLinkedList<T>& l2, SLinkedList<T>& 
  lout); //Homework

template <typename T>
SLinkedList<T>::SLinkedList() : head(nullptr) { } // constructor

template <typename T>
bool SLinkedList<T>::empty() const {
return head == nullptr; // is list empty?
}

template <typename T>
const T& SLinkedList<T>::front() const {
return head->elem; // return front element
}

template <typename T>
SLinkedList<T>::~SLinkedList() {
while (!empty()) removeFront(); // destructor
}

template <typename T>
void SLinkedList<T>::addFront(const T& e) { // Add node to front
SNode<T>* v = new SNode<T>;
v->elem = e;
v->next = head;
head = v;
}

template <typename T>
void SLinkedList<T>::addBack(const T& e) { // Add node to the back
SNode<T>* v = new SNode<T>;
v->elem = e;
SNode<T>* n = head;
if (head == nullptr){ // Empty list
head = v;
return;
}
```

```cpp
while (n->next != nullptr) {
n = n->next;
}
n->next = v;
}

template <typename T>
void SLinkedList<T>::removeFront() { // Remove node from front
SNode<T>* old = head;
head = old->next;
delete old;
}

template <typename T>
void concat(SLinkedList<T>& l1, SLinkedList<T>& l2, SLinkedList<T>& lout) 
  {
// Your code here
 // Iterate through l1 and add its elements to lout
    SNode<T>* current = l1.head;
    while (current != nullptr) {
        lout.addBack(current->elem);
        current = current->next;
    }

    // Iterate through l2 and add its elements to lout
    current = l2.head;
    while (current != nullptr) {
        lout.addBack(current->elem);
        current = current->next;
    }
}

// Test
int main() {
SLinkedList<std::string> p1;
SLinkedList<std::string> p2;
SLinkedList<std::string> p3;
// Add elements
p1.addBack("C");
p1.addBack("C++");
p1.addBack("Java");
p1.addBack("Python");
p1.addBack("Javascript");

p2.addBack("Go");
p2.addBack("Rust");
p2.addBack("Julia");

// Concatenate the progLangsNew list to the end of progrLangs list
```

```
concat(p1,p2,p3);

// Print the concantenated list by repeatedly removing from list
while (!p3.empty()) { // Should print C C++ Java Python Javascript Go Rust Julia
std::cout << p3.front() << " ";
p3.removeFront();
}
std::cout << std::endl;
}
```