Zaimplementuj strukturę danych „lista dwukierunkowa” w sposób następujący:

* Pojedynczy element struktury powinien przechowywać wartość typu int.
* Zdefiniuj metodę addTo(dlista l, int arg, unsigned pos), która pozwoli na dodanie nowego elementu na wskazaną pozycję listy;
* Zdefiniuj metode deleteFrom(dlista l, unsigned pos), która pozwoli na usunięcie elementu ze wskazanej pozycji listy;

Rozwiąż następujące zadanie.

Masz do dyspozycji dwa pliki:

VnumGrupy\_\*.tsk - zawiera dane wejściowe Twojego algorytmu;

solv\_VnumGrupy\_\*.tsk – zawiera docelowy wynik działania algorytmu.

Otwórz plik VnumGrupy\_\*.tsk w trybie binarnym, wczytuj jego zawartość bajt bo bajcie.

* Wczytaj bajt, to będzie kod polecenia.
* Jeżeli wczytany bajt jest kodem znaku ‘<’:

Wczytaj kolejny bajt, to będzie numer pozycji, **pos**,

Wczytaj kolejny bajt, to będzie wartość, **arg**,

Dodaj wartość **arg** na pozycję **pos**.

* Jeżeli wczytany bajt jest kodem znaku ‘>’:

Wczytaj kolejny bajt, to będzie numer pozycji, **pos**,

Usuń element listy o numerze **pos**.

* Jeżeli wczytany bajt jest kodem znaku ‘!’:

zaczynając od zerowego elementu(head) przejdź przez całą listę usuwając każdy element, który będzie przechowywać wartość większą od wartości przechowywanej w jego lewym sąsiedzie, w wyniku ma powstać ciąg nierosnący.

Przykład:

wejście: 4 3 5 7 -4 4 2

wyjście: 4 3 -4

Po wczytaniu całego pliku policz sumę cyfr wartości bezwzględnych wszystkich elementów przechowywanych w liście. Jeżeli lista jest pusta, zwróć zero.

Przykład: wejście: -58 -58 37 24

Wyjście: 42

Aby uzyskać dodatkowy punkt, zadanie ze znakiem ‘!’ oraz ostatnie wykonaj z zastosowaniem **iteratora, zapewniając złożoność liniową przejścia przez listę**.

W schematach 11.12 oraz 11.16

**i = 0;**

~~i = 1;~~

#include <iostream>

#include <string>

#include <fstream>

struct **element**

{

**element**(element\* prev, element\* next, int data): data(data), next(next), prev(prev){};

int data;

element\* next;

element\* prev;

};

class **lista**

{

element\* head = nullptr;

element\* tail = nullptr;

int counter = 0;

element\* **goTo**(int pos)

{

element\* temp = nullptr;

if (pos < counter / 2) // Idziemy od pocz�tku

{

temp = head;

for (int i = 0; i < pos; i++)

{

if (temp->next)

temp = temp->next;

else

break;

}

}

else // Idziemy od ko�ca

{

temp = tail;

for (int i = counter - 1; i > pos; i--)

{

if (temp->prev)

temp = temp->prev;

else

break;

}

}

return temp;

}

public:

~**lista**()

{

while (counter > 0)

{

deleteFrom(0);

}

}

void **addTo**(int arg, int pos)

{

if (pos == 0)

{

if (counter == 0)

{

head = new element(nullptr, nullptr, arg);

tail = head;

counter++;

}

else

{

element\* new\_head = new element(nullptr, head, arg);

head->prev = new\_head;

head = new\_head;

counter++;

}

}

else if (pos > 0 && pos < counter)

{

if (counter != 0)

{

element\* temp = goTo(pos - 1);

element\* new\_element = new element(temp, temp->next, arg);

new\_element->next->prev = new\_element;

temp->next = new\_element;

counter++;

}

}

else if (pos == counter)

{

element\* new\_tail = new element(tail, nullptr, arg);

tail->next = new\_tail;

tail = new\_tail;

counter++;

}

}

void **deleteFrom**(int pos)

{

if (counter != 0)

{

if (pos == 0)

{

element\* to\_delete = head;

head = head->next;

if (head)

head->prev = nullptr;

delete to\_delete;

counter--;

}

else if (pos > 0 && pos < counter - 1)

{

element\* temp = goTo(pos - 1);

element\* to\_delete = temp->next;

temp->next = temp->next->next;

temp->next->prev = temp;

delete to\_delete;

counter--;

}

else if (pos == counter - 1)

{

element\* to\_delete = tail;

tail = tail->prev;

tail->next = nullptr;

delete to\_delete;

counter--;

}

}

}

void **deleteEvery**()

{

if (counter > 1)

{

element\* left = head;

element\* right = head->next;

while (right)

{

while (right->data > left->data)

{

left->next = right->next;

if (right->next)

right->next->prev = left;

if (right == tail)

tail = left;

delete right;

counter--;

right = left->next;

if (!right)

break;

}

left = left->next;

if (left)

right = left->next;

}

}

}

int **getSum**()

{

int sum = 0;

element\* current = head;

for (int i = 0; i < counter; i++)

{

int number = current->data;

if (number < 0)

number = -number;

while (number > 0)

{

sum += number % 10;

number /= 10;

}

current = current->next;

}

return sum;

}

};

int **main**()

{

for (int i = 0; i < 10; i++)

{

lista l;

std::string path = "V3\_" + std::to\_string(i) + ".tsk";

std::cout << path << ":" << std::endl;

std::fstream inFile(path, std::ios::binary | std::ios::in);

char byte = 0;

while (!inFile.eof())

{

inFile.get(byte);

switch (byte)

{

case '<':

{

char pos = inFile.get();

char arg = inFile.get();

l.addTo(arg, pos);

}

break;

case '>':

{

char pos = inFile.get();

l.deleteFrom(pos);

}

break;

case '!':

{

l.deleteEvery();

}

break;

}

}

std::cout << l.getSum() << std::endl;

}

return 0;

}