HÁZI FELADAT

Programozás alapjai 2

Dokumentáció

Réfi Nándor

IDYW5S

2019.04.17.

1. Feladat

Készítsen olyan dinamikus sztringet, melyben a sztring karaktereit 20 karakter tárolására alkalmas tárolókból kialakított láncolt listában tárolja! Valósítsa meg az összes értelmes műveletet operátor átdefiniálással (overload), de nem kell ragaszkodni az összes operátor átdefiniálásához! Legyen az osztálynak iterátora is! Legyen képes az objektum perzisztens viselkedésre!

Specifikáljon egy egyszerű tesztfeladatot, amiben fel tudja használni az elkészített adatszerkezetet! A tesztprogramot külön modulként fordított programmal oldja meg! A megoldáshoz ne használjon STL tárolót!

2. Specifikáció

A sztring több 20 karakter tárolására alkalmas tömb láncolt listájaként lesz képes a következő műveletekre:

- másolás
- értékadás
- indexelés, indexhatár ellenőrzéssel
- iterátor (begin, end)
- hossz lekérdezése (size, length)
- egyenlőség vizsgálata
- sztringek, karakterek hozzáfűzése
- karakterek törlése

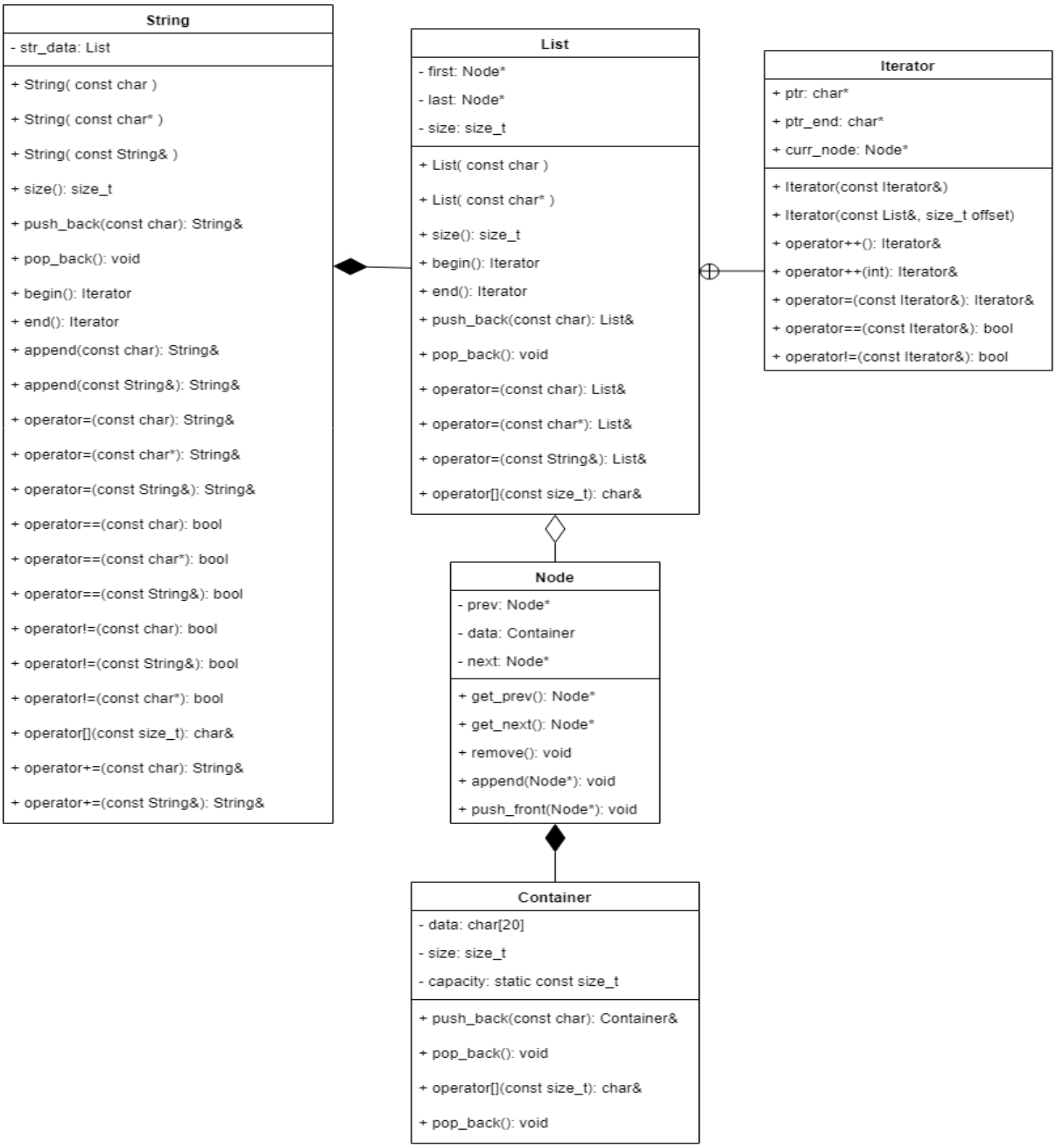
A sztring bármennyi karakter tárolására alkalmas lesz.

A tesztesetekben minden függvénynek megvizsgáljuk a láncolhatóságát, milyen visszatérési értéket várunk el valamint ellenőrizzük a hibakezelést is.

3. Terv

A feladat egy String osztály létrehozása, mely az std::string alapján működik. A String osztály a tartalmát karaktertömbök láncolt listájában fogja tárolni, amit a List osztály fog megvalósítani. A String osztály ezen a List objektumon fog műveleteket végezni melyek segítségével megvalósítható az std::string-hez hasonló működés, valamint ez a List tartalmazni fog egy Iterator osztályt, amely lehetővé teszi az adattagok közötti iterálást.

A List osztályt pedig a Node osztály objektumai fogják felépíteni, melyek pointerekkel hivatkoznak egymásra és ezzel kapcsolatos műveletek elvégézésre képesek. A Node osztály pedig leszármazik a Container osztályból, mely egy 20 karakter tárolására alkalmas tömböt tart nyilván.



4. Dokumentáció

string.h

String:

A String osztály, amely dinamikusan tárolja a benne lévő karaktereket. '\0' karakterrel van lezárva. Az osztálynak van iterátora. Többek között lehet belőle törölni, hozzáadni karaktereket, szövegeket hozzáfűzni, komparálni, indexelni.

Konstruktorok:

String()

String(const String&)

String(const char* c_str)

String(const char val)

Változók:

List str_data

Itt tároljuk dinamikusan a karaktereket.

Függvények:

iterator begin()
String elejére állított iterátort ad vissza.

iterator end()
String végére állított iterátort ad vissza.

size_t size() const
Visszaadja a String hosszát.

size_t length() const
Visszaadja a String hosszát.

void clear()
Törli a String tartalmát.

bool empty() const
True-val tér vissza,ha üres a String.

void swap()
Felcseréli a String tartalmát, pl.: „Alma” -> „amlA”

char& front()
const char& front() const
Visszaad egy referenciát az első elemre.

char& back()
const char& back() const
Visszaad egy refereniát az utolsó elemre.

String& push_back(const char ch)
A végéhez hozzáfűz egy karaktert.

void pop_back()
A végéről töröl egy karaktert.

String& append(const char*)
String& append(const String&)
String& operator+=(const char* ch)
String& operator+=(const String& str)
A String végéhez hozzáfűz egy c_stringet vagy karaktert.

list.h

List:

A List osztály a Node osztály tagjaiból álló láncolt lista kezeléséért felelős. Egy-egy Node osztályban 20-20 db karaktert lehet eltárolni. A Node osztályok pointerekkel hivatkoznak az előttük és utánuk lévő Node-okra.

Konstruktorok:

List()

List(const char val)

List(const char* val)

List(const List& val)

Változók:

Node *first
Első sentinel node.

Node *last
Utolsó sentinel node.

size_t size
Eltárolt karakterek száma.

Függvények:

void connect_nodes(Node *first_node, Node *next_node)
Két Node összekapcsolása.
Paraméterek:
first_node – első Node
next_node - következő Node, amit az első után kapcsolunk

void init_sentinels()
Sentinelek beállítása. Csak egyszer szabad meghívni a konstruktorokban.

void delete_node(Node *target)
Node törlése a listából. Ez rendezi az előtte meg utána lévő Nodeok pointereit is.

size_t get_size() const
Visszaadja a lista elemeinek számát

bool is_empty() const
True-val tér vissza, ha üres a lista

List& push_back(const char ch)
A lista végére fűz egy megadott karaktert.

void pop_back()
A lista végéről töröl egy karaktert.

void clear()
Törli a lista tartalmát.

iterator begin()
A lista elejére állít egy iterátort.

iterator end()
A lista végére állít egy iterátort.

iterator:

Az iterator osztály, mely a List osztályon belül foglal helyet, a listán belüli iterálást teszi lehetővé.

Konstruktorok:

iterator() : ptr(0), ptr_end(0)

iterator(const iterator& iter_r)

iterator(List& r_list, size_t offset = 0)
Egy megadott listára ad egy iterátort.
Paraméterek:
r_list – maga a lista
offset – hányadik elemére állítsuk az iterátort

Változók:

Node *curr_node
Jelenleg melyik Node-ban vagyunk.

char *ptr
Jelenleg kiválasztott elemre mutató pointer.

char *ptr_end
Utolsó elem után mutató pointer.

Függvények:

bool node_ends()
True-val tér vissza, ha a Node utolsó eleméhez értünk.

bool node_begins()
True-val tér vissza, ha a Node első eleméhez értünk.

Valamint rendelkezik *, =, ==, !=, ++, -- operátorokkal.

Node.hpp

Node:

A Node osztály tagja egy-egy listaelem, amelyek együttesen a List osztályt képzik. A Node-on belül találunk egy Container osztályt, amely nyilvántart egy 20-as karaktertömböt.

Konstruktorok:

Node()

Változók:

Node * prev
Az előtte lévő Node-ra mutató pointer.

Container data
20-as karaktertömb.

Node * next
Az utána lévő Node-ra mutató pointer.

Függvények:

Node* get_next() const
Visszaadja a következő Node pointerét.

Node* get_prev() const
Visszaadja az előtte lévő Node pointerét.

void set_next(Node *node)
A következő Node-ra mutató pointert állíthatjuk át.

void set_prev(Node *node)
Az előző Node-ra mutató pointert állíthatjuk át.

size_t get_size() const
Visszaadja a tároló méretét.

static size_t get_capacity()
Visszaadja, hogy maximum hány karaktert tárolhatunk.

bool is_empty()
True-val tér vissza, ha üres a tároló.

bool is_full()
True-val tér vissza, ha tele van a tároló.

void pop_back()
Töröl a tároló végéről.

Container& push_back(const char ch)
A tároló végéhez hozzáfűz egy karaktert.

char& get_last()
const char& get_last()
Visszaadja az utolsó elem referenciáját.

char& get_first()

const char& get_first() const
Visszaadja az első elem referenciáját.

char& operator[](const size_t n)
const char& operator[](const size_t n)
Indexelőoperátorok a tárolóra.

Container.h

Container:

A Container osztály egy 20-as karaktertömböt tart nyilván.

Konstruktorok:

Container()

Változók:

char data[20]
A 20 db karakter, melyet eltárolunk

size_t size
Eltárolt karakterek száma

static const size_t capacity
Maximálisan eltárolható karakterek száma

Függvények:

size_t get_size() const
Visszaadja a tároló méretét.

static size_t get_capacity()
Visszaadja, hogy maximum hány karaktert tárolhatunk.

bool is_empty()
True-val tér vissza, ha üres a tároló.

bool is_full()
True-val tér vissza, ha tele van a tároló.

void pop_back()
Töröl a tároló végéről.

Container& push_back(const char ch)
A tároló végéhez hozzáfűz egy karaktert.

char& get_last()
const char& get_last()
Visszaadja az utolsó elem referenciáját.

char& get_first()
const char& get_first() const
Visszaadja az első elem referenciáját.

char& operator[](const size_t n)
const char& operator[](const size_t n)
Indexelőoperátorok a tárolóra.

5. Tesztesetek

A tesztesetek a main.cpp fájlban találhatóak, a programot elindítva 1-től 8-ig választhatunk ki teszteseteket. 0-át beírva kiléphetünk a tesztelésből.

void test_1()
Konstruktorok működése.

void test_2()
Indexelő operátorok működése.

void test_3()
100 elemű dinamikus tömböt hoz létre.

void test_4()
Append, push_back, pop_back és += operátor.

void test_5()
Clear, empty és swap.

void test_6()
Túlindexelés és üres listából törlés.

void test_7()
Iterátorkezelés.

void test_8()
Logikai egyenlőség operátorok.