Pontosított feladatspecifikáció

Kurcz Valentin - IVKTE6

Feladat

Készítsen titkosító osztályt! Az osztály legyen képes tetszőleges hosszúságú szöveg kódolt formátumú tárolására. A kódoláshoz olyan egyszerű műveletet használjon, ami nem függ a kódolandó szöveg hosszától (pl. kizáró vagy). Valósítsa meg a szokásos string műveleteket! Az osztályt úgy tervezze meg, hogy az örökléssel könnyen felhasználható legyen, az algoritmus könnyen cserélhető legyen! Valósítsa meg az összes értelmes műveletet operátor átdefiniálással (overload), de nem kell ragaszkodni az összes operátor átdefiniálásához! Legyen az osztálynak iterátora is! 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!

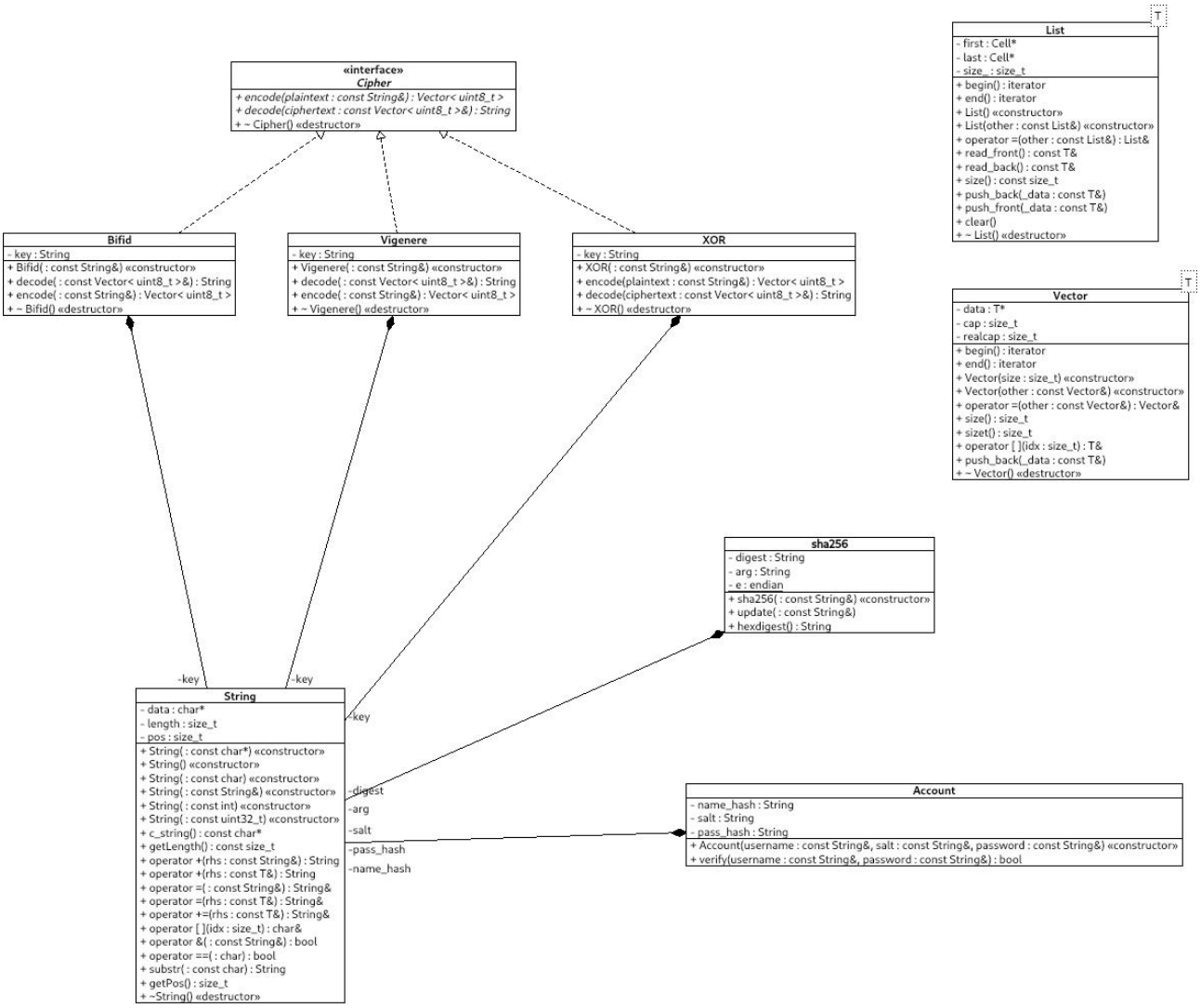
Feladatspecifikáció

A program a titkosító osztály használatával képes lesz szöveget kódolni és dekódolni is. A feladat kódolás esetében bemenetnek el fog várni a felhasználójától egy szöveget, kriptográfiában úgynevezett „plaintext”-et, és egy kulcsot vagy az adott titkosításnál elvárt plusz információt, amik felhasználásával fog létrejönni a kimenet, a „ciphertext”. Dekódolásnál a ciphertext és a kulcs lesz az elvárt bemenet, a kimenet pedig a plaintext. A bemenet és a kimenet formátuma is a titkosítási módszer függvényében szöveg vagy hexadecimális szám lesz. A felhasználónak lesz lehetősége válogatni legalább 3 db különböző titkosítási módszer közül, amik egyike a feladatban is említett XOR lesz. Továbbá a program képes lesz felhasználókat számontartani, regisztálni, illetve bejelentkeztetni felhasználónév és jelszó segítségével, amiket a program, a feladatban implementált SHA256 hash-függvény segítségével tárol. Így minden felhasználónak lesz egy saját tere, amiben használhatja a titkosító osztály funkcióit, aminek eredményei el lesznek tárolva úgy, hogy egy másik felhasználó ne láthasson bele.

Bemenetek:

A program egy felhasználói fiók bejelentkezéséhez vagy regisztrálásához elkér egy felhasználónevet és jelszót, ezek tetszőleges hosszú szövegek lehetnek.
A program fő funkciójánál, a dekódolásnál és enkódolásnál a már említett formátumban vár el egy kulcsot és ciphertextet / plaintextet.

UML Diagram:



Osztályok / Algoritmusok leírása:

sha256:

Az SHA256 algortimst impelemtálja. Tagváltozók szerepe:

- A digest eltárolja a végeredményként kapott 32db hexadecimális számból készített Stringet.
- Az arg az inputot tárolja, ami le van korlátozva Stringre, tekintve hogy a program szempontjából csak arra van szükség (felhasználónevek és jelszavak hashelése)
- Az e eltárolja, hogy az architektúra, amin a program fut, milyen endiannal operál, mivel ez az algortimus szempontjából fontos.

String:

Standard string osztály, ami ki van egészítve különféle konstruktorokkal, amik elősegítik a programon belül használt algoritmusokat, ilyen pl. az int vagy uint32_t paraméterű konstruktor. Operátorok is túl vannak terhelve, ahogyan azt egy felhasználója elvárná.

Vector:

Generikus template tároló. Optimalizálva van benne a push_back, mivel nem egyesével foglalja le mindig az új memória területet ha már megtelt, hanem előre foglal 10 elemnek helyet. Iterátor megvan valósítva benne.

List:

Generikus template tároló. Megvalósítása szabványt próbálja követni. Iterátor van benne.

Cipher:

Absztrakt osztály ami összeköti a titkosító algoritmusokat, virtuális függvényei a decode, encode, és a destruktora

Vigenere:

Egy szöveg→szöveg típusú titkosítás amely lényegében a Caesar ciphernek az egymás utáni kivitelezése.

XOR:

A plaintext minden byteját össze XOR-ozza a kulcs minden betűjével. A végeredményt Vector<uint8_t> ként adja vissza a program, mivel Stringként már nem lehet kezelni XOR után, hiszen gyakran elrontja a String formátumát az XOR.

Bifid:

Egy titkosítás amely szöveg → szöveg típusú, a titkosítás menetében egy Polybius négyzete kerül felírásra majd ez alapján minden katakter kap egy koordinátát amik felhasználásával készül a ciphertext. Részletesebb leírás: https://en.wikipedia.org/wiki/Bifid_cipher

Bacon:

Egy titkosítás amely szöveg → szöveg típusú, amely rendkívül egyszerűen minden karakterhez rendel egy egyedi 5 bites bit sorozatot, amiből úgy készíti a ciphertextet, hogy a bitsorozatokban helyettesíti az 1-eseket X karakterrel a 0-sokat Y karakterrel majd ezeket egymás utáni írása adja a ciphertextet.

Account:

Felhasználó nevének, jelszavának hashét és az annál a hashnél használt sót tárolja el.

NHF

Készítette Doxygen 1.10.0

1. Névtérmutató

1.1. Névtérlista

Az összes dokumentált névtér listája rövid leírásokkal:

2. Hierarchikus mutató

2.1. Osztályhierarchia

Majdnem (de nem teljesen) betűrendbe szedett leszármazási lista:

3. Osztálymutató

3.1. Osztálylista

Az összes osztály, struktúra, unió és interfész listája rövid leírásokkal:

4. Fájlmutató

4.1. Fájllista

Az összes dokumentált fájl listája rövid leírásokkal:

5. Névterek dokumentációja

5.1. gtest_lite névtér-referencia

gtest_lite: a keretrendszer függvényinek és objektumainak névtere

Osztályok

- struct Test

  Tesztek állapotát tároló osztály.

Függvények

- template<typename T1 , typename T2 >
  std::ostream & **EXPECT_** (T1 exp, T2 act, bool(∗pred)(T1, T2), const char ∗file, int line, const char ∗expr, const char ∗lhs="elvart", const char ∗rhs="aktual")

  általános sablon a várt értékhez.
- template<typename T1 , typename T2 >
  std::ostream & **EXPECT_** (T1 ∗exp, T2 ∗act, bool(∗pred)(T1 ∗, T2 ∗), const char ∗file, int line, const char ∗expr, const char ∗lhs="elvart", const char ∗rhs="aktual")

  pointerre specializált sablon a várt értékhez.
- std::ostream & EXPECTSTR (const char ∗exp, const char ∗act, bool(∗pred)(const char ∗, const char ∗), const char ∗file, int line, const char ∗expr, const char ∗lhs="elvart", const char ∗rhs="aktual")

  stringek összehasonlításához.
- template<typename T1 , typename T2 >
  bool eq (T1 a, T2 b)

  segéd sablonok a relációkhoz.
- bool **eqstr** (const char ∗a, const char ∗b)
- bool **eqstrcase** (const char ∗a, const char ∗b)

- template<typename T1 , typename T2 >
  bool **ne** (T1 a, T2 b)
- bool **nestr** (const char *a, const char *b)
- template<typename T1 , typename T2 >
  bool **le** (T1 a, T2 b)
- template<typename T1 , typename T2 >
  bool **lt** (T1 a, T2 b)
- template<typename T1 , typename T2 >
  bool **ge** (T1 a, T2 b)
- template<typename T1 , typename T2 >
  bool **gt** (T1 a, T2 b)
- template<typename T >
  bool **almostEQ** (T a, T b)

  Segédsablon valós számok összehasonlításához Nem bombabiztos, de nekünk most jó lesz Elméleti hátér: http://www.cygnus-software.com/papers/comparingfloats/comparingfloats.htm.

5.1.1. Részletes leírás

gtest_lite: a keretrendszer függvényinek és objektumainak névtere

5.1.2. Függvények dokumentációja

eq()

```
template<typename T1 , typename T2 >
bool gtest_lite::eq (
            T1 a,
            T2 b )
```

segéd sablonok a relációkhoz.

azért nem STL (algorithm), mert csak a függvény lehet, hogy menjen a deduckció

EXPECTSTR()

```
std::ostream & gtest_lite::EXPECTSTR (
            const char * exp,
            const char * act,
            bool(*)(const char *, const char *) pred,
            const char * file,
            int line,
            const char * expr,
            const char * lhs = "elvart",
            const char * rhs = "aktual" )  [inline]
```

stringek összehasonlításához.

azért nem spec. mert a sima EQ-ra másként kell működnie.

6. Osztályok dokumentációja

6.1. _Is_Types< F, T > struktúrasablon-referencia

Segédsablon típuskonverzió futás közbeni ellenőrzésere.

`#include <gtest_lite.h>`

Statikus publikus tagfüggvények

- template<typename D >
  static char(& **f** (D))[1]
- template<typename D >
  static char(& **f** (...))[2]

Statikus publikus attribútumok

- static bool const **convertable** = sizeof(f<T>(F())) == 1

6.1.1. Részletes leírás

template<typename F, typename T>
struct _Is_Types< F, T >

Segédsablon típuskonverzió futás közbeni ellenőrzésere.

Ez a dokumentáció a struktúráról a következő fájl alapján készült:

- gtest_lite.h

6.2. Account osztályreferencia

A fiók adataihoz tartozó műveletekért felelős osztály.

`#include <account.h>`

Publikus tagfüggvények

- **Account** (const String &username="", const String &salt="", const String &password="")
  Konstruktor.
- bool verify (const String &username, const String &password)
  Ellenőrzi, hogy a megadott felhasználónév + só, jelszó + só megegyezik-e a fiókban tároltakkal.

6.2.1. Részletes leírás

A fiók adataihoz tartozó műveletekért felelős osztály.

6.2.2. Tagfüggvények dokumentációja

verify()

```
bool Account::verify (
            const String & username,
            const String & password )
```

Ellenőrzi, hogy a megadott felhasználónév + só, jelszó + só megegyezik-e a fiókban tároltakkal.

Paraméterek

username	felhasználónév.
password	jelszó.

Visszatérési érték

bool.

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

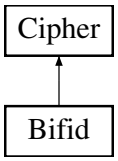
- account.h
- account.cpp

6.3. Bifid osztályreferencia

Bifid titkosítás.

`#include <cipher.h>`

A Bifid osztály származási diagramja:



Publikus tagfüggvények

- **Bifid** (const String &)
  Konstruktor.
- String decode (const Vector< uint8_t > &) const
  Dekódoló függvény.
- Vector< uint8_t > encode (const String &) const
  Enkódoló függvény.
- ~**Bifid** ()
  Destruktor.

Publikus tagfüggvények a(z) Cipher osztályból származnak

- virtual ~**Cipher** ()
  Virtuális destruktor.

6.3.1. Részletes leírás

Bifid titkosítás.

Leszármazott osztály, amely a Bifid féle titkosítást valosítja meg, melyben a titkosítás folyamata a következő:

1. egy 5x5 karakter mátrix kerül létrehozásra a kulcs alapján: B G W K Z Q P N D S I O A X E F C L U M T H Y V R ez az angol abc szerint történik tehát itt is csak az angol abc-ből alkotott szavakat lehet titkosítani.
2. a plaintextből koordinátákat alkotunk: F L E E A T O N C E 4 4 3 3 3 5 3 2 4 3 1 3 5 5 3 1 2 3 2 5
3. ez után egy sorba kiírjuk őket: 4 4 3 3 3 5 3 2 4 3 1 3 5 5 3 1 2 3 2 5
4. majd egymás mellett álló számpárokat koordinátáknak értelmezzük, ezket pedig karakterekké alakítva megkapjuk a titkosított szöveget: 44 33 35 32 43 13 55 31 23 25 U A E O L W R I N S

Szöveg -> Szöveg típusú tehát a ciphertext és a plaintext is String. Csak ASCII betűkkel működik, tehát az angol abc betűivel.

6.3.2. Tagfüggvények dokumentációja

decode()

```
String Bifid::decode (
            const Vector< uint8_t > & ciphertext ) const  [virtual]
```

Dekódoló függvény.

Ez végzi a titkosítást a taglalt módon.

Paraméterek

ciphertext.	

Visszatérési érték

String.

Megvalósítja a következőket: Cipher.

encode()

```
Vector< uint8_t > Bifid::encode (
            const String & plaintext ) const  [virtual]
```

Enkódoló függvény.

Ez végzi a tikosítás visszafejtését.

Paraméterek

plaintext.	

Visszatérési érték

Vector<uint8_t> annak ellenére, hogy itt biztosan szövegként térne vissza, egyszerűbb generikusan így kezelni a titkosításokat. Ez nem fog különösebb problémát okozni mivel ha tudjuk, hogy a Vector<uint8_t> elemei ASCII karakterek, akkor a Vector<uint8_t> -> String konverzió egyszerű.

Megvalósítja a következőket: Cipher.

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

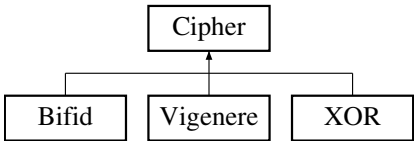
- cipher.h
- cipher.cpp

6.4. Cipher osztályreferencia

Absztakt osztály amely összeköti a különböző tikosítási osztályokat.

`#include <cipher.h>`

A Cipher osztály származási diagramja:



Publikus tagfüggvények

- virtual Vector< uint8_t > encode (const String &plaintext) const =0

  Tisztán virtuális függvény amely az enkódolásért felelős.
- virtual String decode (const Vector< uint8_t > &ciphertext) const =0

  Tisztán virtuális függvény amely a dekódolásért felelős.
- virtual ~**Cipher** ()

  Virtuális destruktor.

6.4.1. Részletes leírás

Absztakt osztály amely összeköti a különböző tikosítási osztályokat.

A titkosítási módszereknek közös metódusait köti össze örökléssel, de példányosítani nem lehet mert tisztán virtuális függvényeket tartalmaz.

6.4.2. Tagfüggvények dokumentációja

decode()

```
virtual String Cipher::decode (
            const Vector< uint8_t > & ciphertext ) const  [pure virtual]
```

Tisztán virtuális függvény amely a dekódolásért felelős.

Különböző titkosítási módoknál, különböző dekódolási algoritmusok érvényesülnek.

Paraméterek

ciphertext	egy Vector<uint8_t> típusú tároló amelynek elemeit dekódolni szeretnénk. Azért Vector<uint8_t> mert a titkosított szöveg legtöbb esetben nem tárolható Stringként, mert nem felel meg a formátuma (pl. NULL érték információt képvisel nem pedig lezáró karaktert).

Megvalósítják a következők: Vigenere, Bifid és XOR.

encode()

```
virtual Vector< uint8_t > Cipher::encode (
            const String & plaintext ) const  [pure virtual]
```

Tisztán virtuális függvény amely az enkódolásért felelős.

Különböző titkosítási módoknál, különböző enkódolási algoritmusok érvényesülnek.

Paraméterek

plaintext	egy String típusú titkosítandó szöveg.

Megvalósítják a következők: Vigenere, Bifid és XOR.

Ez a dokumentáció az osztályról a következő fájl alapján készült:

- cipher.h

6.5. List< T >::const_iterator osztályreferencia

Publikus tagfüggvények

- **const_iterator** (const Cell *cell=NULL)
- bool **operator==** (const iterator other) const
- iterator & **operator++** ()
- iterator **operator++** (int)
- bool **operator!=** (const iterator other) const
- const T & **operator*** () const
- const T * **operator->** () const

Ez a dokumentáció az osztályról a következő fájl alapján készült:

- list.hpp

6.6. Vector< T >::const_iterator osztályreferencia

Publikus tagfüggvények

- **const_iterator** (T *cell=NULL)
- bool **operator==** (const iterator other) const
- const_iterator & **operator++** ()
- const_iterator **operator++** (int)
- const_iterator & **operator-** (int rhs)
- const_iterator & **operator+** (int rhs)
- bool **operator!=** (const const_iterator other) const
- const T & **operator*** () const
- const T * **operator->** () const

Ez a dokumentáció az osztályról a következő fájl alapján készült:

- vector.hpp

6.7. List< T >::iterator osztályreferencia

Iterátor osztály a generikus használat jegyében.

`#include <list.hpp>`

Publikus tagfüggvények

- **iterator** (Cell *cell=NULL)
- bool operator== (const iterator other) const

  Összehasonlító operátor overload.
- iterator & operator++ ()

  Preinkremens operátor overload.
- iterator operator++ (int)

  Posztinkremens operátor overload.
- bool operator!= (const iterator other) const

  Összehasonlító-negált operátor overload.
- T & operator* () const

  Dereferáló operátor overload.
- T * **operator->** () const

  Nyíl operátor overload.

6.7.1. Részletes leírás

template<typename T>
class List< T >::iterator

Iterátor osztály a generikus használat jegyében.

6.7.2. Tagfüggvények dokumentációja

operator"!=()

```
template<typename T >
bool List< T >::iterator::operator!= (
            const iterator other ) const  [inline]
```

Összehasonlító-negált operátor overload.

Összehasoníltja az iterátor által mutatott elem adatát, egy másik iterátor által mutatott elem adatával.

Visszatérési érték

bool érték, igaz ha nem egyenlőek, hamis ha igen.

Paraméterek

other	másik iterátor

operator∗()

```
template<typename T >
T & List< T >::iterator::operator* ( ) const  [inline]
```

Dereferáló operátor overload.

Visszatérési érték

T& T referencia, visszadja az iterátor által mutatott elem adatának referenciáját.

operator++() [1/2]

```
template<typename T >
iterator & List< T >::iterator::operator++ ( )  [inline]
```

Preinkremens operátor overload.

Az iterátort átállítja a következő lista elemre preinkremens módon.

Visszatérési érték

iterator& iterátor refernciával tér vissza, így használható balértékként.

operator++() [2/2]

```
template<typename T >
iterator List< T >::iterator::operator++ (
            int  )  [inline]
```

Posztinkremens operátor overload.

Az iterátort átállítja a következő lista elemre posztinkremens módon.

Visszatérési érték

iterator iterátor tér vissza.

operator==()

```
template<typename T >
bool List< T >::iterator::operator== (
            const iterator other ) const  [inline]
```

Összehasonlító operátor overload.

Összehasoníltja az iterátor által mutatott elem adatát, egy másik iterátor által mutatott elem adatával.

Visszatérési érték

bool érték, igaz ha egyenlőek, hamis ha nem.

Paraméterek

other	másik iterátor

Ez a dokumentáció az osztályról a következő fájl alapján készült:

- list.hpp

6.8. Vector< T >::iterator osztályreferencia

Iterátor osztály a generikus használat jegyében.

```
#include <vector.hpp>
```

Publikus tagfüggvények

- **iterator** (T *cell=NULL)
- bool operator== (const iterator other) const

  Összehasonlító operátor overload.
- iterator & operator++ ()

Preinkremens operátor overload.

- iterator & operator- (int rhs)

  Kivonás operátor.
- iterator & operator+ (int rhs)

  Hozzáadás operátor.
- iterator operator++ (int)

  Posztinkremens operátor overload.
- bool operator!= (const iterator other) const

  Összehasonlító-negált operátor overload.
- T & operator∗ () const

  Dereferáló operátor overload.
- T ∗ **operator->** () const

  Nyíl operátor overload.

6.8.1. Részletes leírás

template<typename T>
class Vector< T >::iterator

Iterátor osztály a generikus használat jegyében.

6.8.2. Tagfüggvények dokumentációja

operator"!=()

```
template<typename T >
bool Vector< T >::iterator::operator!= (
            const iterator other ) const  [inline]
```

Összehasonlító-negált operátor overload.

Összehasoníltja az iterátor által mutatott elem adatát, egy másik iterátor által mutatott elem adatával.

Visszatérési érték

bool érték, igaz ha nem egyenlőek, hamis ha igen.

Paraméterek

other	másik iterátor

operator∗()

```
template<typename T >
T & Vector< T >::iterator::operator* ( ) const  [inline]
```

Dereferáló operátor overload.

Visszatérési érték

T& T referencia, visszadja az iterátor által mutatott elem adatának referenciáját.

operator+()

```
template<typename T >
iterator & Vector< T >::iterator::operator+ (
            int rhs )  [inline]
```

Hozzáadás operátor.

Az iterátort átállítja rhs-sel később lévő cellára. Nem kezel hibát.

Paraméterek

az	eltolás értéke.

operator++() [1/2]

```
template<typename T >
iterator & Vector< T >::iterator::operator++ ( )  [inline]
```

Preinkremens operátor overload.

Az iterátort átállítja a következő cellára preinkremens módon.

Visszatérési érték

iterator& iterátor refernciával tér vissza, így használható balértékként.

operator++() [2/2]

```
template<typename T >
iterator Vector< T >::iterator::operator++ (
            int  )  [inline]
```

Posztinkremens operátor overload.

Az iterátort átállítja a következő cellára posztinkremens módon.

Visszatérési érték

iterator& iterátor refernciával tér vissza, így használható balértékként.

operator-()

```
template<typename T >
iterator & Vector< T >::iterator::operator- (
            int rhs )  [inline]
```

Kivonás operátor.

Az iterátort átállítja rhs-sel előrébb lévő cellára. Nem kezel hibát.

Paraméterek

az	eltolás értéke.

operator==()

```
template<typename T >
bool Vector< T >::iterator::operator== (
            const iterator other ) const  [inline]
```

Összehasonlító operátor overload.

Összehasoníltja az iterátor által mutatott elem adatát, egy másik iterátor által mutatott elem adatával.

Visszatérési érték

bool érték, igaz ha egyenlőek, hamis ha nem.

Paraméterek

other	másik iterátor

Ez a dokumentáció az osztályról a következő fájl alapján készült:

- vector.hpp

6.9. List< T > osztálysablon-referencia

Template osztály amely az std::list szabványát követi.

```
#include <list.hpp>
```

Osztályok

- class const_iterator
- class iterator

  Iterátor osztály a generikus használat jegyében.

Publikus tagfüggvények

- iterator **begin** ()
- iterator **end** ()
- const_iterator **begin** () const
- const_iterator **end** () const
- List ()

  Konstruktor.
- List (const List &other)

Másoló konstruktor.

- List & operator= (const List &other)

  Értékadó operátor.
- const T & read_front () const

  Kiolvassa a lista legelső elemének adatát.
- const T & read_back () const

  Kiolvassa a lista utolsó elemének adatát.
- const size_t size ()

  Visszadja a lista aktuális méretét.
- void push_back (const T &_data)

  A lista legvégéhez fűz új egy elemet.
- void push_front (const T &_data)

  A lista elejére szúr egy új elemet.
- void clear ()

  Törli a lista adatát.
- ~**List** ()

  Destruktor.

6.9.1. Részletes leírás

template<**typename T**>
class List< **T** >

Template osztály amely az std::list szabványát követi.

Generikus tárolásra alkalmas osztály amely a láncolt lista adatstruktúrája szerint épül fel. Elemei duplán lácoltak, belső iterátor osztályt is használ.

6.9.2. Konstruktorok és destruktorok dokumentációja

List() [1/2]

```
template<typename T >
List< T >::List ( )  [inline]
```

Konstruktor.

Inicialiálja a lista objektumot üresen.

List() [2/2]

```
template<typename T >
List< T >::List (
            const List< T > & other )  [inline]
```

Másoló konstruktor.

Inicialiálja a listát egy másik lista adatával.

Paraméterek

other	a másik lista.

6.9.3. Tagfüggvények dokumentációja

clear()

```
template<typename T >
void List< T >::clear ( )  [inline]
```

Törli a lista adatát.

Kiüríti a listát, méretét 0-ra állítja.

Visszatérési érték

void.

operator=()

```
template<typename T >
List & List< T >::operator= (
            const List< T > & other )  [inline]
```

Értékadó operátor.

A lista eddigi memória területét fölszabadítja, majd inicializálja a masik lista elemeivel.

Paraméterek

other	a másik lista.

Visszatérési érték

List&, List referencia így használható balértékként.

push_back()

```
template<typename T >
void List< T >::push_back (
            const T & _data )  [inline]
```

A lista legvégéhez fűz új egy elemet.

Paraméterek

_data	az új elem tartalma.

Visszatérési érték

void.

push_front()

```
template<typename T >
void List< T >::push_front (
            const T & _data )  [inline]
```

A lista elejére szúr egy új elemet.

Paraméterek

_data	az új elem tartalma.

Visszatérési érték

void.

read_back()

```
template<typename T >
const T & List< T >::read_back ( ) const  [inline]
```

Kiolvassa a lista utolsó elemének adatát.

Ha nem üres a lista visszadja az utolsó elem adatát, máskülönben exceptiont dob.

Visszatérési érték

const T& konstans referneciával tér vissza, így optimális marad nagyobb T struktúrákon is.

read_front()

```
template<typename T >
const T & List< T >::read_front ( ) const  [inline]
```

Kiolvassa a lista legelső elemének adatát.

Ha nem üres a lista visszadja a legelső elem adatát, máskülönben exceptiont dob.

Visszatérési érték

const T& konstans referneciával tér vissza, így optimális marad nagyobb T struktúrákon is.

size()

```
template<typename T >
const size_t List< T >::size ( )  [inline]
```

Visszadja a lista aktuális méretét.

Visszatérési érték

const size_t a lista mérete.

Ez a dokumentáció az osztályról a következő fájl alapján készült:

- list.hpp

6.10. sha256 osztályreferencia

SHA256 hash függvény.

```
#include <sha256.h>
```

Publikus tagfüggvények

- sha256 (const String &)
  Konstruktor.
- void update (const String &)
  Lecseréli a digest-et egy új hash-el.
- String hexdigest () const
  Visszadja a digest Stringet.

6.10.1. Részletes leírás

SHA256 hash függvény.

Az SHA256-os hash függvényt megvalósító osztály, amellyel tetszőleges hosszú Stringeket tudunk hashelni.

6.10.2. Konstruktorok és destruktorok dokumentációja

sha256()

```
sha256::sha256 (
            const String & _arg )
```

Konstruktor.

Lényegében ez tartalmazza a hash-elő függvényt.

Paraméterek

input	String.

Inicializáljuk a hash értékeket: (első 32 bitje, az első 8 prím négyzetgyökének):

Pre-processzálás

64db 8 bites adat feldolgozása mint 16db 32 bites adat

Hashelés folyamata

Hash értékének kimentése Stringbe

6.10.3. Tagfüggvények dokumentációja

hexdigest()

```
String sha256::hexdigest ( ) const
```

Visszadja a digest Stringet.

Visszatérési érték

String.

update()

```
void sha256::update (
            const String & _arg )
```

Lecseréli a digest-et egy új hash-el.

Az eddigi arg-hoz hozzá fűz egy másik Stringet, majd ezt az új arg-on újra futtatja a hash függvényt, és frissíti a digestet ez által.

Paraméterek

a	hozzáfűzendő String.

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

- sha256.h
- sha256.cpp

6.11. String osztályreferencia

String osztály, ami követi az std::string mintáját.

```
#include <string.h>
```

Publikus tagfüggvények

- String (const char *)

  Konstruktor.
- String ()

  Paraméter nélüli konstruktor.
- String (const char)

  Konstruktor.
- String (const String &)

  Másoló konstruktor.
- String (const int)

  Konstruktor.
- String (const uint32_t)

  Konstruktor.
- const char * c_string () const

  Visszadja read-only-ként a String karaktertömbjének pointerjét.
- const size_t getLength () const

  Visszadja a String hosszát.
- bool isalpha () const

  Visszadja, hogy a tárol szöveg minden karaktere angol abc-beli e.
- String operator+ (const String &rhs) const

  Összefűző + operátor.
- template<typename T >
  String operator+ (const T &rhs) const

  Összefűző + operátor.
- String & operator= (const String &)

  Értékadó operátor.
- template<typename T >
  String & operator= (const T &rhs)

  Értékadó operátor.
- template<typename T >
  String & operator+= (const T &rhs)

  Hozzáfűző + operátor.
- char & operator[ ] (size_t idx)

  Indexelű operátor.
- const char & operator[ ] (size_t idx) const

  Konstans indexelű operátor.
- bool operator& (const String &) const

  Összehasonlító operátor.
- bool operator== (char) const

  Needle in the haystack.
- String substr (const char)

  Rész stringet készít.
- size_t **getPos** () const

  Visszadja pos értékét.
- void **toUpper** ()

  A szöveg összes abc-beli katarkterét nagy betűre cseréli.
- void **toLower** ()

  A szöveg összes abc-beli katarkterét kis betűre cseréli.
- ~String ()

  Destruktor.

6.11.1. Részletes leírás

String osztály, ami követi az std::string mintáját.

A szövegkezelésért felelős osztály, ami lehetővé teszi a szövegekkel való műveleteket.

6.11.2. Konstruktorok és destruktorok dokumentációja

String() [1/6]

```
String::String (
            const char * _data )
```

Konstruktor.

A Stringet az adott karaktertömb elemeivel inicializálja.

Paraméterek

karakter	tömb.

String() [2/6]

```
String::String ( )
```

Paraméter nélüli konstruktor.

A Stringet 0 mérettel, egy '\0' karakterrel inicializálja.

String() [3/6]

```
String::String (
            const char c )
```

Konstruktor.

A Stringet 1 mérettel, a paraméterként kapott char-ral inicializálja.

Paraméterek

karakter.	

String() [4/6]

```
String::String (
            const String & rhs )
```

Másoló konstruktor.

Paraméterek

egy	másik String objektum.

String() [5/6]

```
String::String (
            const int a )
```

Konstruktor.

A Stringet inicializálja az adott int szám alaki értékeinek számjegyeivel pl. a (int)100-ból "100"-at csinál.

Paraméterek

integer	szám.

String() [6/6]

```
String::String (
            const uint32_t a )
```

Konstruktor.

A Stringet inicializálja az adott uint32_t szám hexadecimális alaki értékeinek számjegyeivel.

Paraméterek

32	bites unsigned int.

~String()

```
String::~String ( )
```

Destruktor.

Fölszabadítja a dinamikusan foglalt karaktertömböt.

6.11.3. Tagfüggvények dokumentációja

c_string()

```
const char * String::c_string ( ) const  [inline]
```

Visszadja read-only-ként a String karaktertömbjének pointerjét.

Visszatérési érték

const char *.

getLength()

```
const size_t String::getLength ( ) const  [inline]
```

Visszadja a String hosszát.

Visszatérési érték

const size_t.

isalpha()

```
bool String::isalpha ( ) const
```

Visszadja, hogy a tárol szöveg minden karaktere angol abc-beli e.

Visszatérési érték

bool.

operator&()

```
bool String::operator& (
            const String & other ) const
```

Összehasonlító operátor.

Összehasonlítja a példányt egy másik Stringgel, és ez szerint visszatér egy bool-lal.

Paraméterek

rhs	amivel összehasonlítunk.

Visszatérési érték

bool az összehasonlítás logikai értéke.

operator+() [1/2]

```
String String::operator+ (
            const String & rhs ) const
```

Összefűző + operátor.

Készít egy új Stringet amelyben a példányt összefűzi a paraméterként kapott másik Stringgel. Operator overload.

Paraméterek

rhs	a hozzáfűzendő String.

Visszatérési érték

String.

operator+() [2/2]

```
template<typename T >
String String::operator+ (
            const T & rhs ) const  [inline]
```

Összefűző + operátor.

Készít egy új Stringet amelyben a példányt összefűzi a paraméterként kapott másik T típusú változóval. Csak azokra a T típusokra működik amikre implementálva van a T->String konverzió (const char*, char, int, uint32_t). Operator overload.

Paraméterek

rhs	a hozzáfűzendő változó.

Visszatérési érték

String.

operator+=()

```
template<typename T >
String & String::operator+= (
            const T & rhs )  [inline]
```

Hozzáfűző + operátor.

A példányhoz hozzáfűzi a paraméterként kapott másik T típusú változót. Csak azokra a T típusokra működik amikre implementálva van a T->String konverzió (const char*, char, int, uint32_t). Operator overload.

Paraméterek

rhs	a hozzáfűzendő változó.

Visszatérési érték

String.

operator=() [1/2]

```
String & String::operator= (
            const String & rhs )
```

Értékadó operátor.

A példány eddigi tartalmát tölri, majd újra inicializálja a paraméter String értékeivel. Operator overload.

Paraméterek

rhs	a példány új értéke.

Visszatérési érték

String& String referencia, tehát használható a balértékként is.

operator=() [2/2]

```
template<typename T >
String & String::operator= (
            const T & rhs )  [inline]
```

Értékadó operátor.

A példány eddigi tartalmát tölri, majd újra inicializálja a paraméter értékeivel. Operator overload. Csak azokra a T típusokra működik amikre implementálva van a T->String konverzió (const char*, char, int, uint32_t). Operator overload.

Paraméterek

rhs	a példány új értéke.

Visszatérési érték

String& String referencia, tehát használható a balértékként is.

operator==()

```
bool String::operator== (
            char c ) const
```

Needle in the haystack.

Megnézi hogy az példány tartalmazza-e az adott karaktert.

Paraméterek

karakter.	

Visszatérési érték

bool a keresés eredménye.

operator[]() [1/2]

```
char & String::operator[] (
            size_t idx )  [inline]
```

Indexelű operátor.

Visszadja az adott indexű karaktert, ha az index a karaktertömb méreténél kisebb. Ellenkező esetben exceptiont dob.

Paraméterek

idx	size_t index.

Visszatérési érték

char& referencia tehát használható balértékként.

operator[]() [2/2]

```
const char & String::operator[] (
            size_t idx ) const  [inline]
```

Konstans indexelű operátor.

Visszadja az adott indexű karaktert, ha az index a karaktertömb méreténél kisebb. Ellenkező esetben exceptiont dob.

Paraméterek

idx	size_t index.

Visszatérési érték

const char& referencia.

substr()

```
String String::substr (
            const char sep )
```

Rész stringet készít.

Az adott karakterig vagy az EOF-ig kimásol egy rész stringet. Változtatja a pos változót, hogy a következő substr hívásnál már a következő rész stringet adja vissza.

Paraméterek

sep	szeparátor karakter.

Visszatérési érték

String.

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

- string.h
- string.cpp

6.12. gtest_lite::Test struktúrareferencia

Tesztek állapotát tároló osztály.

`#include <gtest_lite.h>`

Publikus tagfüggvények

- void **begin** (const char ∗n)

  Teszt kezdete.
- std::ostream & **end** (bool memchk=false)

  Teszt vége.
- bool **fail** ()
- bool **astatus** ()
- std::ostream & **expect** (bool st, const char ∗file, int line, const char ∗expr, bool pr=false)

  Eredményt adminisztráló tagfüggvény True a jó eset.
- ∼**Test** ()

  Destruktor.

Statikus publikus tagfüggvények

- static Test & getTest ()

Publikus attribútumok

- int **sum**

  tesztek számlálója
- int **failed**

  hibás tesztek
- int **ablocks**

  allokált blokkok száma
- bool **status**

  éppen futó teszt státusza
- bool **tmp**

  temp a kivételkezeléshez;
- std::string **name**

  éppen futó teszt neve
- std::fstream **null**

  nyelő, ha nem kell kiírni semmit

6.12.1. Részletes leírás

Tesztek állapotát tároló osztály.

Egyetlen egy statikus példány keletkezik, aminek a destruktora a futás végén hívódik meg.

6.12.2. Tagfüggvények dokumentációja

getTest()

```
static Test & gtest_lite::Test::getTest ( )  [inline], [static]
```

< egyedüli (singleton) példány

Ez a dokumentáció a struktúráról a következő fájl alapján készült:

- gtest_lite.h

6.13. Vector< T > osztálysablon-referencia

Generikus tároló osztály.

```
#include <vector.hpp>
```

Osztályok

- class const_iterator
- class iterator

  Iterátor osztály a generikus használat jegyében.

Publikus tagfüggvények

- iterator begin ()

  Visszadja a tároló első elemére mutató iterátort.
- iterator end ()

  Visszadja a tároló utolsó eleme utánra mutató iterátort.
- const_iterator **begin** () const
- const_iterator **end** () const
- Vector (size_t size=0)

  Konstruktor.
- Vector (const Vector &other)

  Másoló konstruktor.
- Vector & operator= (const Vector &other)

  Értékadó operátor.
- size_t size () const

  Méret lekérdezése.
- size_t sizet () const

  Valódi méret lekérdezése.

- T & operator[ ] (size_t idx)

  Indexelő operátor.
- const T & operator[ ] (size_t idx) const

  Konstans indexelő operátor.
- void push_back (const T &_data)

  Dinamikus nyújtásos hozzáadás.
- void kiir (std::ostream &os) const

  Kiírja az elemeket, az adott streamre.
- bool operator== (const Vector< T > &other)

  Tartalom szerint összehasonlító operátor.
- ~Vector ()

  Destruktor.

6.13.1. Részletes leírás

template<typename T>
class Vector< T >

Generikus tároló osztály.

Az std::vector szabványát követi, ezzel megvalósítva egy generikus nyújtató tömb osztályt.

6.13.2. Konstruktorok és destruktorok dokumentációja

Vector() [1/2]

```
template<typename T >
Vector< T >::Vector (
            size_t size = 0 )  [inline]
```

Konstruktor.

Lefoglal size méretű T típusú tömböt dinamikusan.

Paraméterek

size	a tömb mérete.

Vector() [2/2]

```
template<typename T >
Vector< T >::Vector (
            const Vector< T > & other )  [inline]
```

Másoló konstruktor.

Inicializálja a tárolót a paraméterként kapott másik Vector tároló adataival és méretével.

Paraméterek

other	a másik Vector típusú tároló.

~Vector()

```
template<typename T >
Vector< T >::~Vector ( )  [inline]
```

Destruktor.

A dinamikusan foglalt memória területet felszabadítja.

6.13.3. Tagfüggvények dokumentációja

begin()

```
template<typename T >
iterator Vector< T >::begin ( )  [inline]
```

Visszadja a tároló első elemére mutató iterátort.

Visszatérési érték

iterator

end()

```
template<typename T >
iterator Vector< T >::end ( )  [inline]
```

Visszadja a tároló utolsó eleme utánra mutató iterátort.

Visszatérési érték

iterator

kiir()

```
template<typename T >
void Vector< T >::kiir (
            std::ostream & os ) const  [inline]
```

Kiírja az elemeket, az adott streamre.

Duck typing.

Paraméterek

os	az adott output stream,

operator=()

```
template<typename T >
Vector & Vector< T >::operator= (
            const Vector< T > & other )  [inline]
```

Értékadó operátor.

Felszabadítja a tároló által foglalt memóriát, majd inicializálja a tárolót a paraméterként kapott másik Vector tároló adataival és méretével.

Paraméterek

other	a másik Vector típusó tároló.

Visszatérési érték

Vector& referencia tehát használható balértékként.

operator==()

```
template<typename T >
bool Vector< T >::operator== (
            const Vector< T > & other )  [inline]
```

Tartalom szerint összehasonlító operátor.

Paraméterek

other	egy másik Vector<T> tároló.

Visszatérési érték

bool.

operator[]() [1/2]

```
template<typename T >
T & Vector< T >::operator[] (
            size_t idx )  [inline]
```

Indexelő operátor.

Visszadja a tároló adott indexű elemét, ha az index a tároló tartományába esik. Máskülönben hibát kezel és exceptiont dob.

Paraméterek

idx	az adott index.

Visszatérési érték

T& referencia tehát használható balértékként.

operator[]() [2/2]

```
template<typename T >
const T & Vector< T >::operator[] (
            size_t idx ) const  [inline]
```

Konstans indexelő operátor.

Visszadja a tároló adott indexű elemét, ha az index a tároló tartományába esik. Máskülönben hibát kezel és exceptiont dob.

Paraméterek

idx	az adott index.

Visszatérési érték

const T&.

push_back()

```
template<typename T >
void Vector< T >::push_back (
            const T & _data )  [inline]
```

Dinamikus nyújtásos hozzáadás.

A tároló méretét megnyújta (ha szükséges), majd a végére rakja az adott új elemet. Megnyújtás alatt azt értjük, hogy egy új nagyobb tömböt foglalunk majd oda másoljuk a régi adatot. Itt használódik ki a realcap előnye, mivel minden nyújtásnál 10-zel előre megnyújtja a tároló méretét. így nem minden hozzáadásnál kell egyesével nyújtani, tehát a foglalás burstben történik.

size()

```
template<typename T >
size_t Vector< T >::size ( ) const  [inline]
```

Méret lekérdezése.

Visszadja a tároló felhasználó által gondolt méretét.

Visszatérési érték

size_t

sizet()

```
template<typename T >
size_t Vector< T >::sizet ( ) const  [inline]
```

Valódi méret lekérdezése.

Visszadja a tároló valódi méretét.

Visszatérési érték

size_t

Ez a dokumentáció az osztályról a következő fájl alapján készült:

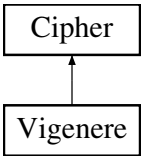
- vector.hpp

6.14. Vigenere osztályreferencia

Vigenere titkosítás.

```
#include <cipher.h>
```

A Vigenere osztály származási diagramja:



Publikus tagfüggvények

- **Vigenere** (const String &)
  Konstruktor.
- String decode (const Vector< uint8_t > &) const
  Dekódoló függvény.
- Vector< uint8_t > encode (const String &) const
  Enkódoló függvény.
- ~**Vigenere** ()
  Destruktor.

Publikus tagfüggvények a(z) Cipher osztályból származnak

- virtual ~**Cipher** ()
  Virtuális destruktor.

6.14.1. Részletes leírás

Vigenere titkosítás.

Leszármazott osztály, amely a Vigenére féle titkosítást valosítja meg, melyben több egymás utáni Caesar kódolás történik (ABC shiftelése). Szöveg -> Szöveg típusú tehát a ciphertext és a plaintext is String. Csak ASCII betűkkel működik, tehát az angol abc betűivel.

6.14.2. Tagfüggvények dokumentációja

decode()

```
String Vigenere::decode (
            const Vector< uint8_t > & ciphertext ) const  [virtual]
```

Dekódoló függvény.

Ez végzi az egymásutáni ABC shiftelést visszafele a kulcs alapján, így feloldva a titkosítást.

Paraméterek

ciphertext.	

Visszatérési érték

String.

Megvalósítja a következőket: Cipher.

encode()

```
Vector< uint8_t > Vigenere::encode (
            const String & plaintext ) const  [virtual]
```

Enkódoló függvény.

Ez végzi az egymásutáni ABC shiftelést a kulcs alapján, így titkosítva a szöveget.

Paraméterek

plaintext.	

Visszatérési érték

Vector<uint8_t> annak ellenére, hogy itt biztosan szövegként térne vissza, egyszerűbb generikusan így kezelni a titkosításokat. Ez nem fog különösebb problémát okozni mivel ha tudjuk, hogy a Vector<uint8_t> elemei ASCII karakterek, akkor a Vector<uint8_t> -> String konverzió egyszerű.

Megvalósítja a következőket: Cipher.

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

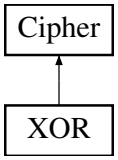
- cipher.h
- cipher.cpp

6.15. XOR osztályreferencia

XOR tiktosítás.

`#include <cipher.h>`

A XOR osztály származási diagramja:



Publikus tagfüggvények

- **XOR** (const String &)
  Konstruktor.
- Vector< uint8_t > encode (const String &plaintext) const
  Enkódoló függvény, amely plaintextet titkosítja XOR-ral.
- String decode (const Vector< uint8_t > &ciphertext) const
  Dekódoló függvény, amely ciphertext titkosítását oldja föl.
- ~**XOR** ()
  Destruktor.

Publikus tagfüggvények a(z) Cipher osztályból származnak

- virtual ~**Cipher** ()
  Virtuális destruktor.

6.15.1. Részletes leírás

XOR tiktosítás.

Leszármazott osztály, amely a XOR titkosítást valósítja meg, ahol a kulcs[i] elemét XOR-ozzuk a plaintext[i]/ciphertext[i] elemével. Ha a kulcs hossza < text hossza akkor a kulcsot ismételjük addig ameddig végig nem ér.

6.15.2. Tagfüggvények dokumentációja

decode()

```
String XOR::decode (
            const Vector< uint8_t > & ciphertext ) const  [virtual]
```

Dekódoló függvény, amely ciphertext titkosítását oldja föl.

Visszatérési érték

String

Paraméterek

ciphertext	feloldandó Vector<uint8_t> típusú tároló.

Megvalósítja a következőket: Cipher.

encode()

```
Vector< uint8_t > XOR::encode (
            const String & plaintext ) const  [virtual]
```

Enkódoló függvény, amely plaintextet titkosítja XOR-ral.

Visszatérési érték

Vector<uint8_t> mivel a XOR szinte mindig elrontja a String formátumát, ezért minden tiktosítandó karaktert uint8_t-ként kezelünk és adunk is tovább titkosítva.

Paraméterek

plaintext	titkosítandó String típusú szöveg.

Megvalósítja a következőket: Cipher.

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

- cipher.h
- cipher.cpp

7. Fájlok dokumentációja

7.1. account.h fájlreferencia

Az Account osztály header fájlja.

```
#include "string.h"
#include "sha256.h"
```

Osztályok

- class Account

  A fiók adataihoz tartozó műveletekért felelős osztály.

7.1.1. Részletes leírás

Az Account osztály header fájlja.

7.2. account.h

Ugrás a fájl dokumentációjához.

```
00001 #include "string.h"
00002 #include "sha256.h"
00010 class Account{
00011   String name_hash,salt, pass_hash;
00012  public:
00016   Account(const String& username = "",const String& salt="", const String& password=""):
      name_hash(username), salt(salt), pass_hash(password){};
00023   bool verify(const String& username, const String& password);
00024 };
00025
```

7.3. cipher.h fájlreferencia

A titkosító osztályok header fájlja.

```
#include "string.h"
#include "vector.hpp"
```

Osztályok

- class Cipher

  Absztakt osztály amely összeköti a különböző tikosítási osztályokat.
- class XOR

  XOR tiktosítás.
- class Vigenere

  Vigenere titkosítás.
- class Bifid

  Bifid titkosítás.

7.3.1. Részletes leírás

A titkosító osztályok header fájlja.

7.4. cipher.h

Ugrás a fájl dokumentációjához.

```
00001 #ifndef CIPHER
00002 #define CIPHER
00003 #include "string.h"
00004 #include "vector.hpp"
00005
00015 class Cipher{
00016   public:
00022  virtual Vector<uint8_t> encode(const String& plaintext) const = 0;
00023
00029  virtual String decode(const Vector<uint8_t>& ciphertext) const = 0;
00033  virtual ~Cipher(){};
00034 };
00035
00042 class XOR: public Cipher{
00043  String key;
00044   public:
00048  XOR(const String&);
00049
00055  Vector<uint8_t> encode(const String& plaintext) const;
00056
```

```
00062  String decode(const Vector<uint8_t>& ciphertext) const;
00063
00067  ~XOR(){};
00068 };
00076 class Vigenere: public Cipher{
00077  String key;
00078   public:
00082  Vigenere(const String&);
00089  String decode(const Vector<uint8_t>&)const;
00096  Vector<uint8_t> encode(const String&)const;
00100  ~Vigenere(){};
00101 };
00126 class Bifid: public Cipher{
00127  char key[5][5];
00131  struct Point{
00132   size_t x;
00133   size_t y;
00134   Point(size_t a=0, size_t b=0): x(a), y(b){}
00135  };
00141  Point find_it(char c) const;
00142   public:
00146  Bifid(const String&);
00153  String decode(const Vector<uint8_t>&)const;
00160  Vector<uint8_t> encode(const String&)const;
00164  ~Bifid(){};
00165 };
00166 #endif
```

7.5. gtest_lite.h fájlreferencia

(v3/2019)

```
#include <iostream>
#include <cassert>
#include <cmath>
#include <cstring>
#include <limits>
#include <cstdlib>
#include <string>
#include <fstream>
```

Osztályok

- struct _Is_Types< F, T >

  Segédsablon típuskonverzió futás közbeni ellenőrzésere.
- struct gtest_lite::Test

  Tesztek állapotát tároló osztály.

Névterek

- namespace gtest_lite

  gtest_lite: a keretrendszer függvényinek és objektumainak névtere

Makródefiníciók

- #define TEST(C, N) do { gtest_lite::test.begin(#C"."#N);

  Teszt kezdete.
- #define **END** gtest_lite::test.end(); } while (false);

  Teszteset vége.
- #define **ENDM** gtest_lite::test.end(true); } while (false);

  Teszteset vége allokált blokkok számának összehasonlításával Ez az ellenőrzés nem bomba biztos.
- #define ENDMsg(t) gtest_lite::test.end(true) << t << std::endl; } while (false);

  Teszteset vége allokált blokkok számának összehasonlításával Ez az ellenőrzés nem bomba biztos.
- #define **SUCCEED**() gtest_lite::test.expect(true, __FILE__, __LINE__, "SUCCEED()", true)

  Sikeres teszt makrója.
- #define **FAIL**() gtest_lite::test.expect(false, __FILE__, __LINE__, "FAIL()", true)

  Sikertelen teszt fatális hiba makrója.
- #define **ADD_FAILURE**() gtest_lite::test.expect(false, __FILE__, __LINE__, "ADD_FAILURE()", true)

  Sikertelen teszt makrója.
- #define **EXPECT_EQ**(expected, actual) gtest_lite::EXPECT_(expected, actual, gtest_lite::eq, __FILE__, _↩
  _LINE__, "EXPECT_EQ(" #expected ", " #actual ")")

  Azonosságot elváró makró
- #define **EXPECT_NE**(expected, actual) gtest_lite::EXPECT_(expected, actual, gtest_lite::ne, __FILE__, _↩
  _LINE__, "EXPECT_NE(" #expected ", " #actual ")", "etalon")

  Eltérést elváró makró
- #define **EXPECT_LE**(expected, actual) gtest_lite::EXPECT_(expected, actual, gtest_lite::le, __FILE__, __↩
  LINE__, "EXPECT_LE(" #expected ", " #actual ")", "etalon")

  Kisebb, vagy egyenlő relációt elváró makró
- #define **EXPECT_LT**(expected, actual) gtest_lite::EXPECT_(expected, actual, gtest_lite::lt, __FILE__, __↩
  LINE__, "EXPECT_LT(" #expected ", " #actual ")", "etalon")

  Kisebb, mint relációt elváró makró
- #define **EXPECT_GE**(expected, actual) gtest_lite::EXPECT_(expected, actual, gtest_lite::ge, __FILE__, _↩
  _LINE__, "EXPECT_GE(" #expected ", " #actual ")", "etalon")

  Nagyobb, vagy egyenlő relációt elváró makró
- #define **EXPECT_GT**(expected, actual) gtest_lite::EXPECT_(expected, actual, gtest_lite::gt, __FILE__, __↩
  LINE__, "EXPECT_GT(" #expected ", " #actual ")", "etalon")

  Nagyobb, mint relációt elváró makró
- #define **EXPECT_TRUE**(actual) gtest_lite::EXPECT_(true, actual, gtest_lite::eq, __FILE__, __LINE__↩
  , "EXPECT_TRUE(" #actual ")")

  Igaz értéket elváró makró
- #define **EXPECT_FALSE**(actual) gtest_lite::EXPECT_(false, actual, gtest_lite::eq, __FILE__, __LINE__↩
  , "EXPECT_FALSE(" #actual ")")

  Hamis értéket elváró makró
- #define **EXPECT_FLOAT_EQ**(expected, actual) gtest_lite::EXPECT_(expected, actual, gtest_lite::almostEQ, __FILE__, __LINE__, "EXPECT_FLOAT_EQ(" #expected ", " #actual ")")

  Valós számok azonosságát elváró makró
- #define **EXPECT_DOUBLE_EQ**(expected, actual) gtest_lite::EXPECT_(expected, actual, gtest_lite::almostEQ, __FILE__, __LINE__, "EXPECT_DOUBLE_EQ(" #expected ", " #actual ")")

  Valós számok azonosságát elváró makró
- #define **EXPECT_STREQ**(expected, actual) gtest_lite::EXPECTSTR(expected, actual, gtest_lite::eqstr, __↩
  FILE__, __LINE__, "EXPECT_STREQ(" #expected ", " #actual ")")

  C stringek (const char ∗) azonosságát tesztelő makró
- #define **EXPECT_STRNE**(expected, actual) gtest_lite::EXPECTSTR(expected, actual, gtest_lite::nestr, __↩
  FILE__, __LINE__, "EXPECT_STRNE(" #expected ", " #actual ")", "etalon")

  C stringek (const char ∗) eltéréset tesztelő makró

- #define **EXPECT_STRCASEEQ**(expected, actual) gtest_lite::EXPECTSTR(expected, actual, gtest_lite↩ ::eqstrcase, __FILE__, __LINE__, "EXPECT_STRCASEEQ(" #expected ", " #actual ")")

  C stringek (const char ∗) azonosságát tesztelő makró (kisbetű/nagybetű azonos)
- #define **EXPECT_STRCASENE**(expected, actual) gtest_lite::EXPECTSTR(expected, actual, gtest_lite↩ ::nestrcase, __FILE__, __LINE__, "EXPECT_STRCASENE(" #expected ", " #actual ")", "etalon")

  C stringek (const char ∗) eltéréset tesztelő makró (kisbetű/nagybetű azonos)
- #define EXPECT_THROW(statement, exception_type)

  Kivételt várunk.
- #define EXPECT_ANY_THROW(statement)

  Kivételt várunk.
- #define EXPECT_NO_THROW(statement)

  Nem várunk kivételt.
- #define ASSERT_NO_THROW(statement)

  Nem várunk kivételt.
- #define EXPECT_THROW_THROW(statement, exception_type)

  Kivételt várunk és továbbdobjuk – ilyen nincs a gtest-ben.
- #define **EXPECT_ENVEQ**(expected, actual) gtest_lite::EXPECTSTR(std::getenv(expected), actual, gtest_↩ lite::eqstr, __FILE__, __LINE__, "EXPECT_ENVEQ(" #expected ", " #actual ")")

  Környezeti változóhoz hasonlít – ilyen nincs a gtest-ben.
- #define **EXPECT_ENVCASEEQ**(expected, actual) gtest_lite::EXPECTSTR(std::getenv(expected), actual, gtest_lite::eqstrcase, __FILE__, __LINE__, "EXPECT_ENVCASEEQ(" #expected ", " #actual ")")

  Környezeti változóhoz hasonlít – ilyen nincs a gtest-ben (kisbetű/nagybetű azonos)
- #define ASSERT_EQ(expected, actual) gtest_lite::ASSERT_(expected, actual, gtest_lite::eq, "ASSER_EQ")

  Azonosságot elváró makró
- #define ASSERT_NO_THROW(statement)

  Nem várunk kivételt.
- #define CREATE_Has_(X)

  Segédmakró egy adattag, vagy tagfüggvény létezésének tesztelésére futási időben Ötlet: https↩ ://cpptalk.wordpress.com/2009/09/12/substitution-failure-is-not-an-error-2
  Használat: CREATE_Has_(size) ... if (Has_size<std::string>::member)...
- #define EXPECTTHROW(statement, exp, act)

  EXPECTTHROW: kivételkezelés.
- #define ASSERTTHROW(statement, exp, act)
- #define ASSERT_(expected, actual, fn, op)
- #define **GTINIT**(IS)
- #define **GTEND**(os)

Függvények

- void **hasMember** (...)

  Segédfüggvény egy publikus adattag, vagy tagfüggvény létezésének tesztelésére fordítási időben.
- template<typename T1 , typename T2 >
  std::ostream & **gtest_lite::EXPECT_** (T1 exp, T2 act, bool(∗pred)(T1, T2), const char ∗file, int line, const char ∗expr, const char ∗lhs="elvart", const char ∗rhs="aktual")

  általános sablon a várt értékhez.
- template<typename T1 , typename T2 >
  std::ostream & **gtest_lite::EXPECT_** (T1 ∗exp, T2 ∗act, bool(∗pred)(T1 ∗, T2 ∗), const char ∗file, int line, const char ∗expr, const char ∗lhs="elvart", const char ∗rhs="aktual")

  pointerre specializált sablon a várt értékhez.
- std::ostream & gtest_lite::EXPECTSTR (const char ∗exp, const char ∗act, bool(∗pred)(const char ∗, const char ∗), const char ∗file, int line, const char ∗expr, const char ∗lhs="elvart", const char ∗rhs="aktual")

  stringek összehasonlításához.

- template<typename T1 , typename T2 >
  bool gtest_lite::eq (T1 a, T2 b)

  segéd sablonok a relációkhoz.
- bool **gtest_lite::eqstr** (const char *a, const char *b)
- bool **gtest_lite::eqstrcase** (const char *a, const char *b)
- template<typename T1 , typename T2 >
  bool **gtest_lite::ne** (T1 a, T2 b)
- bool **gtest_lite::nestr** (const char *a, const char *b)
- template<typename T1 , typename T2 >
  bool **gtest_lite::le** (T1 a, T2 b)
- template<typename T1 , typename T2 >
  bool **gtest_lite::lt** (T1 a, T2 b)
- template<typename T1 , typename T2 >
  bool **gtest_lite::ge** (T1 a, T2 b)
- template<typename T1 , typename T2 >
  bool **gtest_lite::gt** (T1 a, T2 b)
- template<typename T >
  bool **gtest_lite::almostEQ** (T a, T b)

  Segédsablon valós számok összehasonlításához Nem bombabiztos, de nekünk most jó lesz Elméleti hátér: http://www.cygnus-software.com/papers/comparingfloats/comparingfloats.htm.

7.5.1. Részletes leírás

(v3/2019)

Google gtest keretrendszerhez hasonló rendszer. Sz.I. 2015., 2016., 2017. (_Has_X) Sz.I. 2018 (template), ENDM, ENDMsg, nullptr_t Sz.I. 2019 singleton Sz.I. 2021 ASSERT.., STRCASE... Sz.I. 2021 EXPEXT_REGEXP

A tesztelés legalapvetőbb funkcióit támogató függvények és makrók. Nem szálbiztos megvalósítás.

Szabadon felhasználható, bővíthető.

Használati példa: Teszteljük az f(x)=2*x függvényt: int f(int x) { return 2*x; }

int main() { TEST(TeszEsetNeve, TesztNeve) EXPECT_EQ(0, f(0)); EXPECT_EQ(4, f(2)) << "A függvény hibás eredményt adott" << std::endl; ... END ... // Fatális hiba esetén a teszteset nem fut tovább. Ezek az ASSERT... makrók. // Nem lehet a kiírásukhoz további üzenetet fűzni. PL: TEST(TeszEsetNeve, TesztNeve) ASSERT_NO_THROW(f(0)); // itt nem lehet << "duma" EXPECT_EQ(4, f(2)) << "A függvény hibás eredményt adott" << std::endl; ... END ...

A működés részleteinek megértése szorgalmi feladat.

7.5.2. Makródefiníciók dokumentációja

ASSERT_

```
#define ASSERT_(
                expected,
                actual,
                fn,
                op )
```

Érték:

```
    EXPECT_(expected, actual, fn, __FILE__, __LINE__, #op "(" #expected ", " #actual ")" ); \
    if (!gtest_lite::test.status) { gtest_lite::test.end(); break; }
```

ASSERT_EQ

```
#define ASSERT_EQ(
                expected,
                actual ) gtest_lite::ASSERT_(expected, actual, gtest_lite::eq, "ASSER_EQ")
```

Azonosságot elváró makró
ASSERT típusú ellenőrzések. CSak 1-2 van megvalósítva. Nem ostream& -val térnek vissza !!! Kivételt várunk

ASSERT_NO_THROW [1/2]

```
#define ASSERT_NO_THROW(
                statement )
```

Érték:

```
    try { gtest_lite::test.tmp = true; statement; } \
    catch (...) { gtest_lite::test.tmp = false; }\
    ASSERTTHROW(statement, "nem dob kivetelt.", "kivetelt dobott.")
```

Nem várunk kivételt.

ASSERT_NO_THROW [2/2]

```
#define ASSERT_NO_THROW(
                statement )
```

Érték:

```
    try { gtest_lite::test.tmp = true; statement; } \
    catch (...) { gtest_lite::test.tmp = false; }\
    ASSERTTHROW(statement, "nem dob kivetelt.", "kivetelt dobott.")
```

Nem várunk kivételt.

ASSERTTHROW

```
#define ASSERTTHROW(
                statement,
                exp,
                act )
```

Érték:

```
    gtest_lite::test.expect(gtest_lite::test.tmp, __FILE__, __LINE__, #statement) \
    « "** Az utasitas " « (act) \
    « "\n** Azt vartuk, hogy " « (exp) « std::endl; if (!gtest_lite::test.status) { gtest_lite::test.end();
      break; }
```

CREATE_Has_

```
#define CREATE_Has_(
                X )
```

Érték:

```
template<typename T> struct _Has_##X {  \
    struct Fallback { int X; };         \
    struct Derived : T, Fallback {};    \
    template<typename C, C> struct ChT; \
    template<typename D> static char (&f(ChT<int Fallback::*, &D::X>*))[1]; \
    template<typename D> static char (&f(...))[2]; \
    static bool const member = sizeof(f<Derived>(0)) == 2; \
};
```

Segédmakró egy adattag, vagy tagfüggvény létezésének tesztelésére futási időben Ötlet: https://cpptalk.wordpress.com/2009/09/12/substitution-failure-is-not-an-error-2
Használat: CREATE_Has_(size) ... if (Has_size<std::string>::member)...

ENDMsg

```
#define ENDMsg(
                t ) gtest_lite::test.end(true) << t << std::endl; } while (false);
```

Teszteset vége allokált blokkok számának összehasonlításával Ez az ellenőrzés nem bomba biztos.
Ha hiba van kiírja az üzenetet.

EXPECT_ANY_THROW

```
#define EXPECT_ANY_THROW(
                statement )
```

Érték:

```
    try { gtest_lite::test.tmp = false; statement; } \
    catch (...) { gtest_lite::test.tmp = true; } \
    EXPECTTHROW(statement, "kivetelt dob.", "nem dobott kivetelt.")
```

Kivételt várunk.

EXPECT_NO_THROW

```
#define EXPECT_NO_THROW(
                statement )
```

Érték:

```
    try { gtest_lite::test.tmp = true; statement; } \
    catch (...) { gtest_lite::test.tmp = false; }\
    EXPECTTHROW(statement, "nem dob kivetelt.", "kivetelt dobott.")
```

Nem várunk kivételt.

EXPECT_THROW

```
#define EXPECT_THROW(
                statement,
                exception_type )
```

Érték:

```
    try { gtest_lite::test.tmp = false; statement; } \
    catch (exception_type) { gtest_lite::test.tmp = true; } \
    catch (...) { } \
    EXPECTTHROW(statement, "kivetelt dob.", "nem dobott '"#exception_type"' kivetelt.")
```

Kivételt várunk.

EXPECT_THROW_THROW

```
#define EXPECT_THROW_THROW(
                statement,
                exception_type )
```

Érték:

```
    try { gtest_lite::test.tmp = false; statement; } \
    catch (exception_type) { gtest_lite::test.tmp = true; throw; } \
    EXPECTTHROW(statement, "kivetelt dob.", "nem dobott '"#exception_type"' kivetelt.")
```

Kivételt várunk és továbbdobjuk – ilyen nincs a gtest-ben.

EXPECTTHROW

```
#define EXPECTTHROW(
                statement,
                exp,
                act )
```

Érték:

```
    gtest_lite::test.expect(gtest_lite::test.tmp, __FILE__, __LINE__, #statement) \
    « "** Az utasitas " « (act) \
    « "\n** Azt vartuk, hogy " « (exp) « std::endl
```

EXPECTTHROW; kivételkezelés.

Belső megvalósításhoz tartozó makrók, és osztályok.

7.5.2.1. Nem célszerű közvetlenül használni, vagy módosítani

TEST

```
#define TEST(
                C,
                N ) do { gtest_lite::test.begin(#C"."#N);
```

Teszt kezdete.

A makró paraméterezése hasonlít a gtest paraméterezéséhez. Így az itt elkészített testek könnyen átemelhetők a gtest keretrendszerbe.

Paraméterek

C	- teszteset neve (csak a gtest kompatibilitás miatt van külön neve az eseteknek)
N	- teszt neve

7.6. gtest_lite.h

Ugrás a fájl dokumentációjához.

```
00001 #ifndef GTEST_LITE_H
00002 #define GTEST_LITE_H
00003
00043 #include <iostream>
00044 #include <cassert>
00045 #include <cmath>
00046 #include <cstring>
00047 #include <limits>
00048 #include <cstdlib>
00049 #include <string>
00050 #include <fstream>
00051 #if __cplusplus >= 201103L
00052 # include <iterator>
00053 # include <regex>
00054 #endif
00055 #ifdef MEMTRACE
00056 # include "memtrace.h"
00057 #endif
00058
00059 // Két makró az egyes tesztek elé és mögé:
00060 // A két makró a kapcsos zárójelekkel egy új blokkot hoz létre, amiben
00061 // a nevek lokálisak, így elkerülhető a névütközés.
00062
00068 #define TEST(C, N) do { gtest_lite::test.begin(#C"."#N);
00069
00071 #define END gtest_lite::test.end(); } while (false);
00072
00075 #define ENDM gtest_lite::test.end(true); } while (false);
00076
00080 #define ENDMsg(t) gtest_lite::test.end(true) « t « std::endl; } while (false);
00081
00082 // Eredmények vizsgálatát segítő makrók.
00083 // A paraméterek és a funkciók a gtest keretrendszerrel megegyeznek.
00084
00086 #define SUCCEED() gtest_lite::test.expect(true, __FILE__, __LINE__, "SUCCEED()", true)
00087
00089 #define FAIL() gtest_lite::test.expect(false, __FILE__, __LINE__, "FAIL()", true)
00090
00092 #define ADD_FAILURE() gtest_lite::test.expect(false, __FILE__, __LINE__, "ADD_FAILURE()", true)
00093
00095 #define EXPECT_EQ(expected, actual) gtest_lite::EXPECT_(expected, actual, gtest_lite::eq, __FILE__,
      __LINE__, "EXPECT_EQ(" #expected ", " #actual ")" )
00096
00098 #define EXPECT_NE(expected, actual) gtest_lite::EXPECT_(expected, actual, gtest_lite::ne, __FILE__,
      __LINE__, "EXPECT_NE(" #expected ", " #actual ")", "etalon" )
00099
00101 #define EXPECT_LE(expected, actual) gtest_lite::EXPECT_(expected, actual, gtest_lite::le, __FILE__,
      __LINE__, "EXPECT_LE(" #expected ", " #actual ")", "etalon" )
00102
00104 #define EXPECT_LT(expected, actual) gtest_lite::EXPECT_(expected, actual, gtest_lite::lt, __FILE__,
      __LINE__, "EXPECT_LT(" #expected ", " #actual ")", "etalon" )
00105
00107 #define EXPECT_GE(expected, actual) gtest_lite::EXPECT_(expected, actual, gtest_lite::ge, __FILE__,
      __LINE__, "EXPECT_GE(" #expected ", " #actual ")", "etalon" )
00108
00110 #define EXPECT_GT(expected, actual) gtest_lite::EXPECT_(expected, actual, gtest_lite::gt, __FILE__,
      __LINE__, "EXPECT_GT(" #expected ", " #actual ")", "etalon" )
00111
00113 #define EXPECT_TRUE(actual)  gtest_lite::EXPECT_(true, actual,  gtest_lite::eq, __FILE__, __LINE__,
      "EXPECT_TRUE(" #actual ")" )
00114
00116 #define EXPECT_FALSE(actual) gtest_lite::EXPECT_(false, actual, gtest_lite::eq, __FILE__, __LINE__,
      "EXPECT_FALSE(" #actual ")" )
00117
00119 #define EXPECT_FLOAT_EQ(expected, actual)  gtest_lite::EXPECT_(expected, actual, gtest_lite::almostEQ,
      __FILE__, __LINE__, "EXPECT_FLOAT_EQ(" #expected ", " #actual ")" )
00120
00122 #define EXPECT_DOUBLE_EQ(expected, actual) gtest_lite::EXPECT_(expected, actual, gtest_lite::almostEQ,
      __FILE__, __LINE__, "EXPECT_DOUBLE_EQ(" #expected ", " #actual ")" )
00123
00125 #define EXPECT_STREQ(expected, actual) gtest_lite::EXPECTSTR(expected, actual, gtest_lite::eqstr,
      __FILE__, __LINE__, "EXPECT_STREQ(" #expected ", " #actual ")" )
00126
```

```
00128 #define EXPECT_STRNE(expected, actual) gtest_lite::EXPECTSTR(expected, actual, gtest_lite::nestr,
      __FILE__, __LINE__, "EXPECT_STRNE(" #expected ", " #actual ")", "etalon" )
00129
00131 #define EXPECT_STRCASEEQ(expected, actual) gtest_lite::EXPECTSTR(expected, actual,
      gtest_lite::eqstrcase, __FILE__, __LINE__, "EXPECT_STRCASEEQ(" #expected ", " #actual ")" )
00132
00134 #define EXPECT_STRCASENE(expected, actual) gtest_lite::EXPECTSTR(expected, actual,
      gtest_lite::nestrcase, __FILE__, __LINE__, "EXPECT_STRCASENE(" #expected ", " #actual ")", "etalon" )
00135
00137 #define EXPECT_THROW(statement, exception_type) try { gtest_lite::test.tmp = false; statement; } \
00138     catch (exception_type) { gtest_lite::test.tmp = true; } \
00139     catch (...) { } \
00140     EXPECTTHROW(statement, "kivetelt dob.", "nem dobott '"#exception_type"' kivetelt.")
00141
00143 #define EXPECT_ANY_THROW(statement) try { gtest_lite::test.tmp = false; statement; } \
00144     catch (...) { gtest_lite::test.tmp = true; } \
00145     EXPECTTHROW(statement, "kivetelt dob.", "nem dobott kivetelt.")
00146
00148 #define EXPECT_NO_THROW(statement) try { gtest_lite::test.tmp = true; statement; } \
00149     catch (...) { gtest_lite::test.tmp = false; }\
00150     EXPECTTHROW(statement, "nem dob kivetelt.", "kivetelt dobott.")
00151
00153 #define ASSERT_NO_THROW(statement) try { gtest_lite::test.tmp = true; statement; } \
00154     catch (...) { gtest_lite::test.tmp = false; }\
00155     ASSERTTHROW(statement, "nem dob kivetelt.", "kivetelt dobott.")
00156
00158 #define EXPECT_THROW_THROW(statement, exception_type) try { gtest_lite::test.tmp = false; statement; }
      \
00159     catch (exception_type) { gtest_lite::test.tmp = true; throw; } \
00160     EXPECTTHROW(statement, "kivetelt dob.", "nem dobott '"#exception_type"' kivetelt.")
00161
00163 #define EXPECT_ENVEQ(expected, actual) gtest_lite::EXPECTSTR(std::getenv(expected), actual,
      gtest_lite::eqstr, __FILE__, __LINE__, "EXPECT_ENVEQ(" #expected ", " #actual ")" )
00164
00166 #define EXPECT_ENVCASEEQ(expected, actual) gtest_lite::EXPECTSTR(std::getenv(expected), actual,
      gtest_lite::eqstrcase, __FILE__, __LINE__, "EXPECT_ENVCASEEQ(" #expected ", " #actual ")" )
00167
00168 #if __cplusplus >= 201103L
00170 # define EXPECT_REGEXP(expected, actual, match, err) gtest_lite::EXPECTREGEXP(expected, actual, match,
      err, __FILE__, __LINE__, "EXPECT_REGEXP(" #expected ", " #actual ", " #match ")" )
00171 #endif
00175
00177 #define ASSERT_EQ(expected, actual) gtest_lite::ASSERT_(expected, actual, gtest_lite::eq, "ASSER_EQ")
00178
00180 #define ASSERT_NO_THROW(statement) try { gtest_lite::test.tmp = true; statement; } \
00181     catch (...) { gtest_lite::test.tmp = false; }\
00182     ASSERTTHROW(statement, "nem dob kivetelt.", "kivetelt dobott.")
00183
00184
00191 #define CREATE_Has_(X) \
00192 template<typename T> struct _Has_##X {  \
00193     struct Fallback { int X; };         \
00194     struct Derived : T, Fallback {};    \
00195     template<typename C, C> struct ChT; \
00196     template<typename D> static char (&f(ChT<int Fallback::*, &D::X>*))[1]; \
00197     template<typename D> static char (&f(...))[2]; \
00198     static bool const member = sizeof(f<Derived>(0)) == 2; \
00199 };
00200
00203 inline void hasMember(...) {}
00204
00206 template <typename F, typename T>
00207 struct _Is_Types {
00208     template<typename D> static char (&f(D))[1];
00209     template<typename D> static char (&f(...))[2];
00210     static bool const convertable = sizeof(f<T>(F())) == 1;
00211 };
00212
00217
00219 #define EXPECTTHROW(statement, exp, act) gtest_lite::test.expect(gtest_lite::test.tmp, __FILE__,
      __LINE__, #statement) \
00220     « "** Az utasitas " « (act) \
00221     « "\n** Azt vartuk, hogy " « (exp) « std::endl
00222
00223 #define ASSERTTHROW(statement, exp, act) gtest_lite::test.expect(gtest_lite::test.tmp, __FILE__,
      __LINE__, #statement) \
00224     « "** Az utasitas " « (act) \
00225     « "\n** Azt vartuk, hogy " « (exp) « std::endl; if (!gtest_lite::test.status) {
      gtest_lite::test.end(); break; }
00226
00227 #define ASSERT_(expected, actual, fn, op) EXPECT_(expected, actual, fn, __FILE__, __LINE__, #op "("
      #expected ", " #actual ")" ); \
00228     if (!gtest_lite::test.status) { gtest_lite::test.end(); break; }
00229
00230 #ifdef CPORTA
00231 #define GTINIT(is)  \
00232     int magic;      \
```

```
00233     is » magic;
00234 #else
00235 #define GTINIT(IS)
00236 #endif // CPORTA
00237
00238 #ifdef CPORTA
00239 #define GTEND(os)  \
00240     os « magic « (gtest_lite::test.fail() ? " NO" : " OK?") « std::endl;
00241 #else
00242 #define GTEND(os)
00243 #endif // CPORTA
00244
00246 namespace gtest_lite {
00247
00251 struct Test {
00252     int sum;
00253     int failed;
00254     int ablocks;
00255     bool status;
00256     bool tmp;
00257     std::string name;
00258     std::fstream null;
00259     static Test& getTest() {
00260         static Test instance;
00261         return instance;
00262     }
00263 private:
00264     Test() :sum(0), failed(0), status(false), null("/dev/null") {}
00265     Test(const Test&);
00266     void operator=(const Test&);
00267 public:
00269     void begin(const char *n) {
00270         name = n; status = true;
00271 #ifdef MEMTRACE
00272         ablocks = memtrace::allocated_blocks();
00273 #endif
00274 #ifndef CPORTA
00275         std::cerr « "\n---> " « name « std::endl;
00276 #endif // CPORTA
00277         ++sum;
00278     }
00280     std::ostream& end(bool memchk = false) {
00281 #ifdef MEMTRACE
00282         if (memchk && ablocks != memtrace::allocated_blocks()) {
00283             status = false;
00284             return std::cerr « "** Lehet, hogy nem szabaditott fel minden memoriat! **" « std::endl;
00285         }
00286 #endif
00287 #ifdef CPORTA
00288         if (!status)
00289 #endif // CPORTA
00290             std::cerr « (status ? "     SIKERES" : "** HIBAS ****") « "\t" « name « " <---" «
    std::endl;
00291         if (!status)
00292             return std::cerr;
00293         else
00294             return null;
00295     }
00296
00297     bool fail() { return failed; }
00298
00299     bool astatus() { return status; }
00300
00302     std::ostream& expect(bool st, const char *file, int line, const char *expr, bool pr = false) {
00303         if (!st) {
00304             ++failed;
00305             status = false;
00306         }
00307         if (!st || pr) {
00308             std::string str(file);
00309             size_t i = str.rfind("\\");
00310             if (i == std::string::npos) i = str.rfind("/");
00311             if (i == std::string::npos) i = 0; else i++;
00312             return std::cerr « "\n**** " « &file[i] « "(" « line « "): " « expr « " ****" « std::endl;
00313         }
00314         return null;
00315     }
00316
00318     ~Test() {
00319 #ifdef CPORTA
00320         if (failed)
00321 #endif // CPORTA
00322             std::cerr « "\n==== TESZT VEGE ==== HIBAS/OSSZES: " « failed « "/" « sum « std::endl;
00323     }
00324 };
00325
00328 static Test& test = Test::getTest();
```

```
00329
00331 template <typename T1, typename T2>
00332 std::ostream& EXPECT_(T1 exp, T2 act, bool (*pred)(T1, T2), const char *file, int line,
00333                       const char *expr, const char *lhs = "elvart", const char *rhs = "aktual") {
00334     return test.expect(pred(exp, act), file, line, expr)
00335         « "** " « lhs « ": " « std::boolalpha « exp
00336         « "\n** " « rhs « ": " « std::boolalpha « act « std::endl;
00337 }
00338
00340 template <typename T1, typename T2>
00341 std::ostream& EXPECT_(T1* exp, T2* act, bool (*pred)(T1*, T2*), const char *file, int line,
00342                       const char *expr, const char *lhs = "elvart", const char *rhs = "aktual") {
00343     return test.expect(pred(exp, act), file, line, expr)
00344         « "** " « lhs « ": " « (void*) exp
00345         « "\n** " « rhs « ": " « (void*) act « std::endl;
00346 }
00347
00348 #if __cplusplus >= 201103L
00350 template <typename T1>
00351 std::ostream& EXPECT_(T1* exp, std::nullptr_t act, bool (*pred)(T1*, std::nullptr_t), const char
      *file, int line,
00352                       const char *expr, const char *lhs = "elvart", const char *rhs = "aktual") {
00353     return test.expect(pred(exp, act), file, line, expr)
00354         « "** " « lhs « ": " « (void*) exp
00355         « "\n** " « rhs « ": " « (void*) act « std::endl;
00356 }
00357 #endif
00358
00361 inline
00362 std::ostream& EXPECTSTR(const char *exp, const char *act, bool (*pred)(const char*, const char*),
      const char *file, int line,
00363                       const char *expr, const char *lhs = "elvart", const char *rhs = "aktual") {
00364     return test.expect(pred(exp, act), file, line, expr)
00365         « "** " « lhs « ": " « (exp == NULL ? "NULL pointer" : std::string("\"") + exp +
      std::string("\""))
00366         « "\n** " « rhs « ": " « (act == NULL ? "NULL pointer" : std::string("\"") + act +
      std::string("\"")) « std::endl;
00367 }
00368
00369 #if __cplusplus >= 201103L
00371 template <typename E, typename S>
00372 int count_regexp(E exp, S str) {
00373     std::regex rexp(exp);
00374     auto w_beg = std::sregex_iterator(str.begin(), str.end(), rexp);
00375     auto w_end = std::sregex_iterator();
00376     return std::distance(w_beg, w_end);
00377 }
00378
00379 template <typename E, typename S>
00380 std::ostream& EXPECTREGEXP(E exp, S str, int match, const char *err,  const char *file, int line,
00381                     const char *expr, const char *lhs = "regexp", const char *rhs = "string",
00382                     const char *m = "elvart/illeszkedik") {
00383     int cnt = count_regexp(exp, str);
00384     if (match < 0) match = cnt;
00385     return test.expect(cnt == match, file, line, expr)
00386         « "** " « lhs « ": " « std::string("\"") + exp + std::string("\"")
00387         « "\n** " « rhs « ": " « (err == NULL ? std::string("\"") + str + std::string("\"") : err)
00388         « "\n** " « m « ": " « match « "/" « cnt « std::endl;
00389 }
00390 #endif
00391
00394 template <typename T1, typename T2>
00395 bool eq(T1 a, T2 b) { return a == b; }
00396
00397 inline
00398 bool eqstr(const char *a, const char *b) {
00399     if (a != NULL && b != NULL)
00400         return strcmp(a, b) == 0;
00401     return false;
00402 }
00403
00404 inline
00405 bool eqstrcase(const char *a, const char *b) {
00406     if (a != NULL && b != NULL) {
00407         while (toupper(*a) == toupper(*b) && *a != '\0') {
00408             a++;
00409             b++;
00410         }
00411         return *a == *b;
00412     }
00413     return false;
00414
00415 }
00416
00417 template <typename T1, typename T2>
00418 bool ne(T1 a, T2 b) { return a != b; }
00419
```

```
00420 inline
00421 bool nestr(const char *a, const char *b) {
00422     if (a != NULL && b != NULL)
00423         return strcmp(a, b) != 0;
00424     return false;
00425 }
00426
00427 template <typename T1, typename T2>
00428 bool le(T1 a, T2 b) { return a <= b; }
00429
00430 template <typename T1, typename T2>
00431 bool lt(T1 a, T2 b) { return a < b; }
00432
00433 template <typename T1, typename T2>
00434 bool ge(T1 a, T2 b) { return a >= b; }
00435
00436 template <typename T1, typename T2>
00437 bool gt(T1 a, T2 b) { return a > b; }
00438
00443 template <typename T>
00444 bool almostEQ(T a, T  b) {
00445     // eps: ha a relatív, vagy abszolút hiba ettől kisebb, akkor elfogadjuk
00446     T eps = 10 * std::numeric_limits<T>::epsilon(); // 10-szer a legkisebb érték
00447     if (a == b) return true;
00448     if (fabs(a - b) < eps)
00449        return true;
00450     double aa = fabs(a);
00451     double ba = fabs(b);
00452     if (aa < ba) {
00453         aa = ba;
00454         ba = fabs(a);
00455     }
00456     return (aa - ba) < aa * eps;
00457 }
00458
00459 } // namespace gtest_lite
00460
00461 #endif // GTEST_LITE_H
```

7.7. list.hpp fájlreferencia

Generikus List tároló header fájlja.

```
#include <cstddef>
#include <stdexcept>
#include <stdlib.h>
```

Osztályok

- class List< T >

  Template osztály amely az std::list szabványát követi.
- class List< T >::iterator

  Iterátor osztály a generikus használat jegyében.
- class List< T >::const_iterator

7.7.1. Részletes leírás

Generikus List tároló header fájlja.

7.8. list.hpp

Ugrás a fájl dokumentációjához.

```
00001 #ifndef LIST
00002 #define LIST
00003
00004 #include <cstddef>
00005 #include <stdexcept>
00006 #include <stdlib.h>
00007
00018 template <typename T>
00019 class List{
00023   struct Cell{
00024     Cell* next;
00025     Cell* prev;
00026     T data;
```

```
00027     Cell(Cell *_next, Cell *_prev, const T& _data): next(_next), prev(_prev), data(_data){}
00028   };
00029   Cell *first, *last;
00030   size_t size_;
00031 public:
00035   class iterator{
00036     Cell* cell;
00037   public:
00038     iterator(Cell* cell = NULL): cell(cell){}
00045     bool operator==(const iterator other) const{
00046      return cell == other.cell;
00047     }
00053     iterator& operator++(){
00054       if(cell != NULL) cell = cell->next;
00055       return *this;
00056     }
00062     iterator operator++(int){
00063       iterator tmp = *this;
00064       ++(*this);
00065       return tmp;
00066     }
00073     bool operator!=(const iterator other)const{
00074       return cell != other.cell;
00075     }
00080     T& operator*() const{
00081       return cell->data;
00082     }
00083
00087     T* operator->() const{
00088       return &(cell->data);
00089     }
00090   };
00091   class const_iterator{
00092     const Cell* cell;
00093   public:
00094     const_iterator(const Cell* cell = NULL): cell(cell){}
00095     bool operator==(const iterator other) const{
00096      return cell == other.cell;
00097     }
00098     iterator& operator++(){
00099       if(cell != NULL) cell = cell->next;
00100       return *this;
00101     }
00102     iterator operator++(int){
00103       iterator tmp = *this;
00104       ++(*this);
00105       return tmp;
00106     }
00107     bool operator!=(const iterator other)const{
00108       return cell != other.cell;
00109     }
00110     const T& operator*() const{
00111       return cell->data;
00112     }
00113     const T* operator->() const{
00114       return &(cell->data);
00115     }
00116   };
00117   iterator begin() {
00118     return iterator(first);
00119   }
00120   iterator end() {
00121     return iterator(NULL);
00122   }
00123   const_iterator begin() const{
00124     return const_iterator(first);
00125   }
00126   const_iterator end() const {
00127     return const_iterator(NULL);
00128   }
00133   List(): first(NULL), last(NULL), size_(0){}
00134
00140   List(const List& other): first(NULL), last(NULL), size_(0){
00141     Cell *tmp = other.first;
00142     while(tmp != NULL){
00143       (*this).push_back(tmp->data);
00144       tmp = tmp->next;
00145     }
00146   }
00147
00154   List& operator=(const List& other){
00155     (*this).clear();
00156     Cell *tmp = other.first;
00157     while(tmp != NULL){
00158       (*this).push_back(tmp->data);
00159       tmp = tmp->next;
00160     }
```

```
00161     return *this;
00162   }
00168   const T& read_front() const{
00169     if(first == NULL) throw std::runtime_error("Üres a lista, nincsen első elem!");
00170     else{
00171       return first->data;
00172     }
00173   }
00179   const T& read_back() const{
00180     if(first == NULL) throw std::runtime_error("Üres a lista, nincsen utolsó elem!");
00181     else{
00182       return last->data;
00183     }
00184   }
00189   const size_t size(){return size_;}
00195   void push_back(const T& _data){
00196     size_++;
00197     Cell *tmp = new Cell(NULL, last, _data);
00198     if(last == NULL){
00199       first = tmp;
00200     }
00201     else{
00202       last->next = tmp;
00203     }
00204     last = tmp;
00205   }
00211   void push_front(const T& _data){
00212     size_++;
00213     Cell *tmp = new Cell(first,NULL,_data);
00214     if(first == NULL){
00215       last = tmp;
00216     }
00217     else{
00218       first->prev = tmp;
00219     }
00220     first = tmp;
00221   }
00227   void clear(){
00228     while(first != NULL){
00229       Cell *mozgo = first->next;
00230       delete first;
00231       first = mozgo;
00232     }
00233     last = NULL;
00234     size_ = 0;
00235   }
00239   ~List(){
00240     while(first != NULL){
00241       Cell *mozgo = first->next;
00242       delete first;
00243       first = mozgo;
00244     }
00245   }
00246 };
00247 #endif // !LIST
```

7.9. sha256.h fájlreferencia

Az sha256 osztály header fájlja, ami tartalmaz az sha256 által használt egyéb globális függvények deklarációját is.

```
#include "string.h"
#include <cstdint>
#include "vector.hpp"
```

Osztályok

- class sha256

  SHA256 hash függvény.

Enumerációk

- enum endian { **little** , **big** }

  Ez egy enum ami olvashatóbbá teszi a programkódot.
- enum dir { **right** , **left** }

  Ez egy enum ami olvashatóbbá teszi a programkódot.

Függvények

- endian endian_check ()

  Az endiannes-t dönti el.

- template<typename T >
  void reverse_byte_order (T *buf, size_t buf_size)

  Megfordítja a byte sorrendet.

- template<typename T >
  T rotate (const T &buf, size_t buf_size, int rotN, dir d)

  Egy bitsorozaton végez forgatást.

7.9.1. Részletes leírás

Az sha256 osztály header fájlja, ami tartalmaz az sha256 által használt egyéb globális függvények deklarációját is.

7.9.2. Enumerációk dokumentációja

dir

```
enum dir
```

Ez egy enum ami olvashatóbbá teszi a programkódot.
A rotate függvényben hasznljuk, hogy jelezzük melyik irányba forgajtuk a bitsorozatot.

endian

```
enum endian
```

Ez egy enum ami olvashatóbbá teszi a programkódot.
Lehetséges értékei: little - ezzel jelezzük hogy little endian, big - ezzel pedig hogy big endian.

7.9.3. Függvények dokumentációja

endian_check()

```
endian endian_check ( )  [inline]
```

Az endiannes-t dönti el.
Végez egy ellenőrzést, hogy az architektúra, amin a program fut milyen endiannes-t használ.

Visszatérési érték

endian az endian enumot használja visszatérésnek.

reverse_byte_order()

```
template<typename T >
void reverse_byte_order (
            T * buf,
            size_t buf_size )
```

Megfordítja a byte sorrendet.

Paraméterek

buf	a buffer amiben a megfordítandó byteok vannak.
buf_size	a buffer mérete.

rotate()

```
template<typename T >
T rotate (
```

```
            const T & buf,
            size_t buf_size,
            int rotN,
            dir d )
```

Egy bitsorozaton végez forgatást.
Lényegében bit shift jobbra / balra, csak a kilépő bitek a legnagyobb / legkisebb helyiértékeken visszakerülnek.

Paraméterek

buf	a forgatandó bitsorozat.
buf_size	a bitsorozat mérete.
rotN	a forgatás hányszorosa.
d	a forgatás iránya.

7.10. sha256.h

Ugrás a fájl dokumentációjához.

```
00001 #ifndef SHA256
00002 #define SHA256
00003 #include "string.h"
00004 #include <cstdint>
00005 #include "vector.hpp"
00015 enum endian{
00016   little,
00017   big,
00018 };
00023 enum dir{
00024   right,
00025   left,
00026 };
00032 inline endian endian_check(){
00033   int n = 1;
00034   if(*(char*)&n == 1) return little;
00035   else return big;
00036 }
00042 template <typename T>
00043 void reverse_byte_order(T *buf, size_t buf_size){
00044   uint8_t *buf0 = buf;
00045   for (size_t i = 0; i<buf_size/2; ++i) {
00046     uint8_t temp = buf0[i];
00047     buf0[i] = buf0[buf_size - i -1];
00048     buf0[buf_size - i -1] = temp;
00049   }
00050 }
00059 template <typename T>
00060 T rotate(const T& buf, size_t buf_size, int rotN, dir d){
00061   switch(d){
00062     case right:
00063         return (buf » rotN) | (buf « (buf_size*8-rotN));
00064         break;
00065     case left:
00066         return (buf « rotN) | (buf » (buf_size*8-rotN));
00067         break;
00068     default:
00069         return buf;
00070   }
00071 }
00076 class sha256{
00077   String digest;
00078   String arg;
00079   static endian e;
00081   public:
00087   sha256(const String&);
00093   void update(const String&);
00098   String hexdigest() const;
00099 };
00100 #endif
```

7.11. string.h fájlreferencia

```
#include <cstddef>
#include <cstdint>
#include <cstring>
```

```
#include <iostream>
#include <stdexcept>
```

Osztályok

- class String

  String osztály, ami követi az std::string mintáját.

Függvények

- std::ostream & **operator**<< (std::ostream &os, const String &rhs)

  Globális inserter operátor.
- std::istream & operator>> (std::istream &is, String &rhs)

  Globális extractor operátor.

7.11.1. Részletes leírás

A String osztály header fájlja.

7.11.2. Függvények dokumentációja

operator>>()

```
std::istream & operator>> (
            std::istream & is,
            String & rhs )
```

Globális extractor operátor.
A whitespace-ket figyelmen kivűl hagyja.

7.12. string.h

Ugrás a fájl dokumentációjához.

```
00001 #ifndef STRING
00002 #define STRING
00003
00004 #include <cstddef>
00005 #include <cstdint>
00006 #include <cstring>
00007 #include <iostream>
00008 #include <stdexcept>
00009
00019 class String{
00020   char *data;
00021   size_t length;
00022   size_t pos;
00023   public:
00029   String(const char  *);
00034   String();
00040   String(const char);
00045   String(const String&);
00051   String(const int);
00057   String(const uint32_t);
00062   const char * c_string() const{
00063     return data;
00064   }
00069   const size_t getLength() const{
00070     return length;
00071   }
00076   bool isalpha() const;
00083   String operator+(const String& rhs) const;
00092   template<typename T>
00093   String operator+(const T& rhs) const{
00094     return *this + String(rhs);
00095   }
00102   String& operator=(const String&);
00111   template<typename T>
00112   String& operator=(const T& rhs){
00113     return *this = String(rhs);
00114   }
00123   template<typename T>
```

```
00124   String& operator+=(const T& rhs){
00125     return *this = *this + rhs;
00126   }
00133   char& operator[](size_t idx){
00134     if(idx < 0 || idx >= length) throw std::out_of_range("Az index a sztring határain kívül esik!");
00135     else{
00136       return data[idx];
00137     }
00138   }
00145   const char& operator[](size_t idx) const{
00146     if(idx < 0 || idx >= length) throw std::out_of_range("Az index a sztring határain kívül esik!");
00147     else{
00148       return data[idx];
00149     }
00150   }
00157   bool operator&(const String&) const;
00164   bool operator==(char) const;
00172   String substr(const char );
00176   size_t getPos() const;
00180   void toUpper();
00184   void toLower();
00189   ~String();
00190 };
00194 std::ostream& operator«(std::ostream& os, const String& rhs);
00199 std::istream& operator»(std::istream& is, String& rhs);
00200 #endif // !STRING
```

7.13. vector.hpp fájlreferencia

A Vector generikus tároló osztály header fájlja.

```
#include <cstddef>
#include <stdexcept>
```

Osztályok

- class Vector< T >

  Generikus tároló osztály.
- class Vector< T >::iterator

  Iterátor osztály a generikus használat jegyében.
- class Vector< T >::const_iterator

7.13.1. Részletes leírás

A Vector generikus tároló osztály header fájlja.

7.14. vector.hpp

Ugrás a fájl dokumentációjához.

```
00001 #ifndef VECTOR
00002 #define VECTOR
00003
00004 #include <cstddef>
00005 #include <stdexcept>
00015 template<typename T>
00016 class Vector{
00017   T* data;
00018   size_t cap;
00019   size_t realcap;
00020 public:
00024   class iterator{
00025     T* cell;
00026   public:
00027     iterator(T* cell = NULL): cell(cell){}
00034     bool operator==(const iterator other) const{
00035      return cell == other.cell;
00036     }
00042     iterator& operator++(){
00043       if(cell != NULL) ++cell;
00044       return *this;
00045     }
00052     iterator& operator-(int rhs){
00053       cell -= rhs;
00054       return *this;
00055     }
```

```
00062     iterator& operator+(int rhs){
00063       cell += rhs;
00064       return *this;
00065     }
00071     iterator operator++(int){
00072       iterator tmp = *this;
00073       ++(*this);
00074       return tmp;
00075     }
00082     bool operator!=(const iterator other)const{
00083       return cell != other.cell;
00084     }
00089     T& operator*() const{
00090       return *cell;
00091     }
00095     T* operator->() const{
00096       return cell;
00097     }
00098   };
00099   class const_iterator{
00100     T const * cell;
00101   public:
00102     const_iterator(T* cell = NULL): cell(cell){}
00103     bool operator==(const iterator other) const{
00104      return cell == other.cell;
00105     }
00106     const_iterator& operator++(){
00107       if(cell != NULL) ++cell;
00108       return *this;
00109     }
00110     const_iterator operator++(int){
00111       iterator tmp = *this;
00112       ++(*this);
00113       return tmp;
00114     }
00115     const_iterator& operator-(int rhs){
00116       cell -= rhs;
00117       return *this;
00118     }
00119     const_iterator& operator+(int rhs){
00120       cell += rhs;
00121       return *this;
00122     }
00123     bool operator!=(const const_iterator other)const{
00124       return cell != other.cell;
00125     }
00126     const T& operator*() const{
00127       return *cell;
00128     }
00129     const T* operator->() const{
00130       return cell;
00131     }
00132   };
00137   iterator begin() {
00138     return iterator(data);
00139   }
00144   iterator end() {
00145     return iterator(data+cap);
00146   }
00147   const_iterator begin() const{
00148     return const_iterator(data);
00149   }
00150   const_iterator end() const {
00151     return const_iterator(data+cap);
00152   }
00158   Vector(size_t size = 0): cap(size), realcap(size){
00159     data = new T[cap];
00160   }
00166   Vector(const Vector& other){
00167     cap = other.cap;
00168     realcap = other.realcap;
00169     data = new T[realcap];
00170     for (size_t i = 0; i < cap; i++) {
00171       data[i] = other.data[i];
00172     }
00173   }
00180   Vector& operator=(const Vector& other){
00181     if(&other != this){
00182       delete[] data;
00183       cap = other.cap;
00184       realcap = other.realcap;
00185       data = new T[realcap];
00186       for (size_t i = 0; i < cap; i++) {
00187         data[i] = other.data[i];
00188       }
00189     }
00190     return *this;
```

```
00191   }
00197   size_t size() const{
00198     return cap;
00199   }
00205   size_t sizet() const{
00206     return realcap;
00207   }
00215   T& operator[](size_t idx){
00216     if(idx < 0 || idx >= cap) throw std::out_of_range("Az index a vector határain kívül esik!");
00217     else{
00218       return data[idx];
00219     }
00220   }
00228   const T& operator[](size_t idx) const{
00229     if(idx < 0 || idx >= cap) throw std::out_of_range("Az index a vector határain kívül esik!");
00230     else{
00231       return data[idx];
00232     }
00233   }
00240   void push_back(const T& _data){
00241     if(cap == realcap){
00242       realcap += 10;
00243       T* new_data = new T[realcap];
00244       for (size_t i = 0; i < cap; i++) {
00245       new_data[i] = data[i];
00246       }
00247       new_data[cap] = _data;
00248       delete[] data;
00249       data = new_data;
00250     }
00251     else {
00252       data[cap] = _data;
00253     }
00254     cap++;
00255   }
00261   void kiir(std::ostream& os) const{
00262     for(const_iterator it = begin(); it != end(); ++it){
00263       os « *it « " ";
00264     }
00265     os « '\n';
00266   }
00272   bool operator==(const Vector<T>& other){
00273     if(other.size() == cap){
00274       for(size_t i = 0; i < cap; ++i){
00275         if(other[i] != (*this)[i]) return false;
00276       }
00277      return true;
00278     }
00279     return false;
00280   }
00285   ~Vector(){
00286     delete[] data;
00287   }
00288 };
00289 #endif // !VECTOR
```

Tárgymutató