

# Házi feladat dokumentáció

---

Programozás alapjai 2.

Kis-Bogdán Kolos

## Választott feladat

---

[Feladatötletek listá](#)ról a dinamikus sztringet választottam.

### Dinamikus sztring

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!

## Specifikáció

---

### Működés

A program a leírtak szerint 20 karakter hosszú, dinamikusan foglalt tárolókban tárolja az adatokat, amiket láncolt listába fűz.

### Műveletek

#### Létrehozás

Sztringet lehet létrehozni:

- üresen
- `char`-ból
- `char *`-ból
- másik sztringből

#### Értékadás | =

Ugyanúgy működik, mint a létrehozás, csak felülírja a sztring tartalmát.

Visszatérési értéke: önmaga, a többszörös értékadás lehetővé tétele miatt.

#### Indexelés | []

Visszaadja az adott indexen lévő karakter referenciáját. Túlindexelés esetén `std::out_of_range` kivételt dob.

## Egyenlőség | `==` | `!=`

Összehasonlítja magát egy másik:

- sztringgel
- `char`-ral
- `char *`-gal

Visszatérési értéke `true` megegyeznek (`char`-ral csak egy hosszú sztring egyezhet meg), egyébként `false`.

A `!=` egyértelműen ennek az ellentéte.

## Összeadás | `+`

A sztringet összeadja egy másik:

- sztringgel
- `char`-ral
- `char *`-gal

Visszaad egy új sztringet.

## Hozzáfűzés | `+=`

A sztringhez hozzáfűz egy:

- sztringet
- `char`-t
- `char *`-t

Visszatérési értéke a jelenlegi sztring referenciája.

## Left-shift | krokodil | `<<`

A krokodil operátorral ki lehet írni a sztringet egy `std::ostream`-re. Visszaadja az `std::ostream` referenciát, lehetővé téve az egymás után fűzést.

## Right-shift | másfajta krokodil | `>>`

A másfajta krokodil operátorral `std::istream`-ről lehet sztringet beolvasni. Sorvége karakterig (`'\n'` | `'\r'`) vagy EOF-ig olvas. Visszaadja az `std::istream` referenciát.

## Megszüntetés | destruktor

Felszabadítja az összes dinamikusán foglalt memóriát.

## Tagfüggvények

## Részsstring készítés | `substr(const size_t from, const size_t length)`

A sztringnek egy megadott részét adja vissza. Kezdő indexnél zárta. Hibás indexelés esetén `std::out_of_range` típusú kivételt dob.

Visszatérési értéke: új sztring, ami csak az adott részt tartalmazza.

## C-sztring létrehozása | `c_str(const bool autoFree = true)`

`const char *`-ot csinál a sztringből, amit egy dinamikusan foglalt memóriaterületen tárol. Az `autoFree true` értékre állítása esetén a felhasználónak nem kell felszabadítani a memóriaterületet, hanem azt majd a sztring osztály megteszi. `false` értéknél a felszabadítást egyértelműen a felhasználónak kell megtennie.

Visszatérési értéke: `const char *`, ami a sztring tartalmából képzett karaktertömbre mutat, lezáró nulla van a végén.

## C-sztring felszabadítása | `free_c_str()`

Felszabadítja a `c_str()` által foglalt memóriát, ha van, a sztring destrukálása előtt.

## Hossz lekérdezése | `length()`

A sztring jelenlegi hosszának lekérdezése.

Visszatérési értéke: `size_t`

## Karakter vagy sztring keresése | `find(const char c) | find(const char *s) | find(String s)`

Megkeresi a sztringben az első előfordulását az adott keresési feltételnek.

Visszatérési értéke: `size_t`, első előfordulás indexe. Ha nem fordul elő, akkor `std::out_of_range` kivételt dob.

## Iterátor

Az osztálynak van iterátora, ami megszokott módon viselkedik.

A sztring elejét a `.begin()`-nel, a végét a `.end()`-el lehet lekérni.

## Tesztelés

A tesztelést a Jportán is használt `gtest_lite`-al fogom megvalósítani. A teszt az osztály minden részét, több különböző tesztesettel fogja tesztelni.

## Módosítások a specifikációban

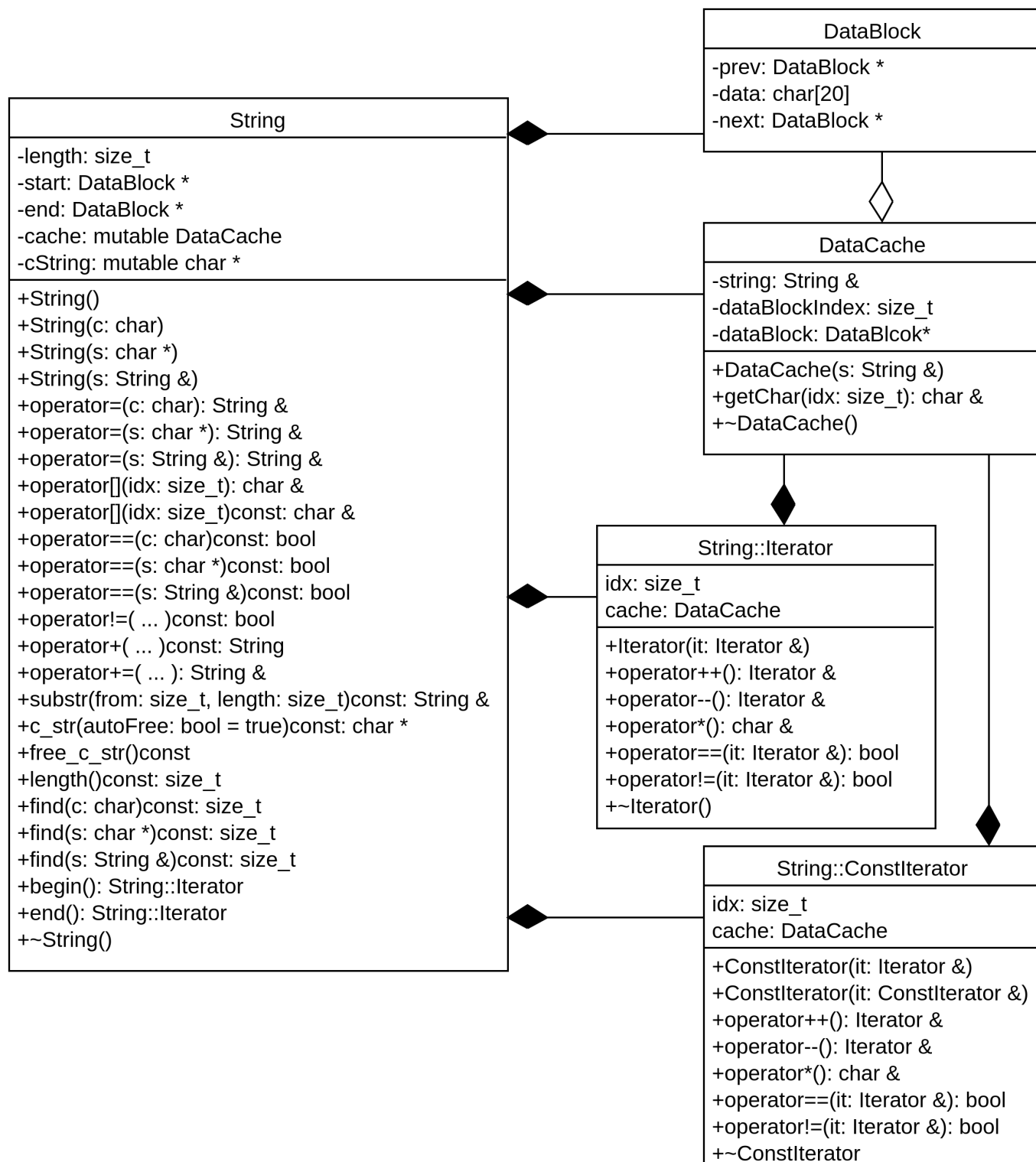
---

- `substring()` átnevezése `substr()`-re és paraméterek módosítása (`from` és `to` helyett `from` és `length`)
- `cString()` átnevezése `c_str()`-re
- `freeCString()` átnevezése `free_c_str()`-re

- + operátor hozzáadása
- += operátor hozzáadása

# Terv

## UML diagram



Megjegyzés: az `operator!=`, `operator+`, `operator+=` ugyanolyan paraméterekkel hívható, mint az `operator==`

## Említésre méltó algoritmusok

## Cache

A láncolt lista hozzáféréseinek gyorsítása a célja. Amikor egy sztringet indexel a felhasználó, az általában egymás utáni karakterekre történik.

Inicializáláskor a sztring első adatblokkjára mutató pointert tárolja el. (A sztringnek mindig van legalább egy adatblokkja)

Karakter indexeléskor két eset lehet:

1. a karakter beleesik a meglévő cache-be (sztring eleje, vége és a cache blokk): Ilyenkor egyszerűen ki lehet olvasni a megfelelő helyről.
2. a karakter nincs benne a cache-ben: Ilyenkor a kért adatblokkhoz a legközelebbi cache-ből el kell jutni a kívánt blokkig, majd kiolvasni a karaktert. Utána a cache-be ez a blokk kerül.

## Iterátor

Az iterátorban is van cache, így egy sztringhez egyszerre több iterátorral, több helyen lehet hozzáférni hatékonyan.

### `std::istream`-ről beolvasás

A sztring normál állapotában nincs mindig tele az utolsó adatblokk. A beolvasás menete:

1. utolsó adatblokk feltöltése adattal
2. új adatblokk fűzése a sztringhez és ennek feltöltése, amíg van adat

Megjegyzés: `std::istream`-ről mindig csak annyi karaktert olvas be, amennyit kell.

## Szubsztring képzés

1. teszteli, hogy a `from + length` túlíndexeli-e a sztringet, ha igen, kivételt dob
2. új üres sztringet hoz létre
3. karakterenként átmásolja az új sztringbe a kívánt részt

## C-sztring képzés

- `autoFree == true`
  1. ha nem `nullptr` a `cString`, akkor azt felszabadítja azt
  2. foglal memóriát `length + 1` karakternek a `cString`-be
  3. blokkonként átmásolja az adatokat a `cString`-be
  4. utolsó blokkból csak annyit másol, amennyi adat van benne
  5. `'\0'`-t tesz a végére
- `autoFree == false`
  1. foglal memóriát `length + 1` karakternek egy ideiglenes változóban
  2. blokkonként átmásolja az adatokat ebbe a változóba
  3. utolsó blokkból csak annyit másol, amennyi adat van benne
  4. `'\0'`-t tesz a végére
  5. `cString`-hez hozzá sem nyúl

Megjegyzés: `autoFree == false` esetén ezt a memóriát a felhasználónak kell felszabadítania.

## C-sztring felszabadítás

`free_c_str()` függvénnyel: függvény hívásakor felszabadítja a memóriát és `nullptr`-re állítja a `cString`-et, ha nem volt `nullptr`.

Destruktorral: automatikusan meghívja a `free_c_str()`-t, ha az `autoFree` igaz.

Megjegyzés: csak az `autoFree == true`-val végzett `c_str()` képzésnél lehet így felszabadítani a memóriát.

## Indexelés

Indexelésnél csak a felső határt kell nézni, mivel a `size_t` nem lehet negatív.

# Módosítások a tervben

---

- `NULL` módosítása `nullptr`-re
- UML diagram
  - `mutable` hozzáadása néhány adattaghoz
  - `operator+=()`-nél `const` törlése (mert ez így baromság volt)
- Indexelés megjegyzés törlése (`mutable`-vel már működik konstans objektumra)
- C-sztring képzés formázás módosítás
- Házi feladat specifikáció - specifikáció -> dokumentáció

# Tesztelés

---

## Memóriakezelés tesztelése

A memóriakezelést a laborokon is használt `memtrace` segítségével vizsgálom, mely minden fájlba utolsóként van importálva.

## Megfelelő működés tesztelése

A tesztelés `gtest_lite`-al van megvalósítva, a program működését 42 darab teszt ellenőrzi. Minden teszt eset megvizsgálja a `const` sztring működését is, amelyik esetben ennek persze van értelme. A tesztek a `main.cpp`-ben vannak megvalósítva.

Megjegyzés: a `TEST()` és `END` után csak azért van `;`, hogy az automatikus formázás megfelelően működjön.

### 1. Konstruktor, üres sztring

Konstruktor paraméter nélküli meghívásával `0` hosszú sztring jön létre.

### 2. Konstruktor, egy karakter

Egy karakterből létrehozott sztring hossza `1`.

### 3. Konstruktor, `nullptr`

`nullptr`-ből létrehozott sztring hossza 0.

#### 4. Konstruktor, rövid c-sztring

Rövid c-sztringből létrehozott sztring hossza megfelelő. Ez a teszt 5 hosszú sztringgel tesztel.

#### 5. Konstruktor, hosszú c-sztring

Hosszú c-sztringből létrehozott sztring hossza megfelelő. Ez a teszt 50 hosszú sztringgel tesztel.

#### 6. Másoló konstruktor

Sztring másolatának hossza megegyezik a másolt sztring hosszával.

#### 7. `c_str()`, automatikus felszabadítás

Sztringből képzett `char *` megfelelő adatot tartalmaz, és a sztring ezt maga után fel is takarítja. `c_str()` többszöri futtatása hatására nincs memóriaszivárgás.

#### 8. `c_str()`, manuális felszabadítás

Sztringből képzett `char *` megfelelő adatot tartalmaz a sztring destruktálása után is. Egy sztringből több `c_str`-t is lehet lekérni, és ezek mindegyike jó.

#### 9. Egyenlővé tétel, üres sztring

Üres sztring egyenlővé tétele `char`-ral, üres `char *`-gal, rövid `char *`-gal, hosszú `char *`-gal, sztringgel és `nullptr`-rel. Mindegyikre megfelelően viselkedik és nincs memóriaszivárgás.

#### 10. Egyenlővé tétel, rövid sztring

Rövid sztring egyenlővé tétele `char`-ral, üres `char *`-gal, rövid `char *`-gal, hosszú `char *`-gal, sztringgel és `nullptr`-rel. Mindegyikre megfelelően viselkedik és nincs memóriaszivárgás.

#### 11. Egyenlővé tétel, hosszú sztring

Hosszú sztring egyenlővé tétele `char`-ral, üres `char *`-gal, rövid `char *`-gal, hosszú `char *`-gal, sztringgel és `nullptr`-rel. Mindegyikre megfelelően viselkedik és nincs memóriaszivárgás.

#### 12. Egyenlővé tétel, önmaga

Sztring önmagával egyenlővé tétele nem okoz gondot.

#### 13. Indexelés, üres sztring

Minden pozitív, nulla és negatív index kivételt dob.

#### 14. Indexelés, 1 hosszú sztring

Csak a nulladik index nem dob kivételt, ezen kívül minden negatív és pozitív index kivételt dob. Indexelt karakter értéke módosítható.

## 15. Indexelés, 6 hosszú sztring

Indexelés megfelelően működik. Indexelt karakter értéke módosítható.

## 16. Indexelés, 65 hosszú sztring

Indexelés megfelelően működik. Indexelt karakter értéke módosítható.

## 17. Egyenlőség, egy karakter

Egy karakterrel csak egy hosszú sztring lehet egyenlő, az is csak akkor, ha a tartalom megegyezik.

## 18. Egyenlőség, `nullptr`

`nullptr`-rel semmilyen sztring nem egyenlő.

## 19. Egyenlőség, rövid c-sztring

## 20. Egyenlőség, hosszú c-sztring

## 21. Egyenlőség, sztring

## 22. Hozzáadás, egy karakter

## 23. Hozzáadás, több karakter

## 24. Hozzáadás, rövid c-sztring és `nullptr`

`nullptr` hozzáadása nem csinál semmit.

## 25. Hozzáadás, hosszú c-sztring

## 26. Hozzáadás, sztring

## 27. Hozzáadás, egy karakter

## 28. Hozzáadás, több karakter

## 29. Hozzáadás, rövid c-sztring

## 30. Hozzáadás, hosszú c-sztring

## 31. Hozzáadás, sztring

## 32. Részsstring, megfelelő indexelés

## 33. Részsstring, érvénytelen index

Kivételt dob.

## 34. Keresés, karakter

Ha nem találja meg, kivételt dob.



### 35. Keresés, c-sztring

`nullptr`-t nem lehet keresni

### 36. Keresés, sztring

### 37. Iterátor, alapvető használat

`*it`-nek lehet értéket adni.

### 38. Iterátor, érvénytelen index

Érvénytelen helyre mutató iterátor kivételt dob.

### 39. Iterátor, konstans sztringre

### 40. `std::istream`-ről beolvasás

Elválasztó karaktereknél megfelelően viselkedik. Daisy-chain működik.

### 41. `std::ostream`-re írás

Daisy-chain működik.

### 42. Konstans sztring `std::ostream`-re írás

Csak hogy 42 teszt legyen.

## class `String`

### Summary

| Members                                     | Descriptions  |
|---|---|
| <code>class ConstIterator</code>            | Iterator for constant string.   |
| <code>class DataBlock</code>                | Linked list element to store data.  |
| <code>class DataCache</code>                | Cache for indexing string.  |
| <code>class Iterator</code>                 | Iterator for string.  |
| <code>String()</code>                       | Construct a new empty String object.  |
| <code>explicit String(char c)</code>        | Construct a new String object from a single char.   |
| <code>explicit String(const char *s)</code> | Construct a new String object from a const char *.  |
| <code>String(const String &amp;s)</code>    | Copy constructor for string.  |
| <code>String &amp;operator=(char c)</code>  | Assign the value of a single char to the string. The previous contents of the string are discarded. |

| Members  | Descriptions   |
|--|--|
| <code>String &amp;operator=(const char *s)</code>            | Assign the value of a c-string to the string. The previous contents of the string are discarded.                 |
| <code>String &amp;operator=(const String &amp;s)</code>      | Assign the value of another string to the string. The previous contents of the string are discarded.             |
| <code>char &amp;operator[](size_t idx)</code>                | Indexing operator. Throws an <code>std::out_of_range</code> exception in case of over or underindexing.          |
| <code>char operator[](size_t idx) const</code>               | Constant indexing operator. Throws an <code>std::out_of_range</code> exception in case of over or underindexing. |
| <code>bool operator==(char c) const</code>                   | Compare with a single character. Only can be equal if the string has a length of 1.                              |
| <code>bool operator==(const char *s) const</code>            | Compare whti a c-string. Only can be equal if the length is the same.  |
| <code>bool operator==(const String &amp;s) const</code>      | Compare with another string. Only can be equal if the length is the same.  |
| <code>bool operator!=(char c) const</code>                   | Compare with a single character. Only can be equal if the string has a length of 1.                              |
| <code>bool operator!=(const char *s) const</code>            | Compare whti a c-string. Only can be equal if the length is the same.  |
| <code>bool operator!=(const String &amp;s) const</code>      | Compare with another string. Only can be equal if the length is the same.  |
| <code>String operator+(char c) const</code>                  | Add a character to the end of the string. Does not change original string.                                       |
| <code>String operator+(const char *s) const</code>           | Add a c-string to the end of the string. Does not change original string.  |
| <code>String operator+(constString &amp;s) const</code>      | Add a string to the end of the string. Does not change original string.  |
| <code>String &amp;operator+=(char c)</code>                  | Append a character to the current string. Changes the current string.  |
| <code>String &amp;operator+=(const char *s)</code>           | Append a c-string to the current string. Changes the current string.   |
| <code>String &amp;operator+=(constString &amp;s)</code>      | Append a string to the current string. Changes the current string.   |
| <code>String substr(size_t from, size_t length) const</code> | Create a substring from the string.  |
| <code>const char *c_str(bool autoFree) const</code>          | Create a c-string closed by a <code>\0</code> .  |

| Members   | Descriptions  |
|---|---|
| <code>void free_c_str() const</code>                | Free the memory allocated by <code>c_string()</code> before the string destructs.                                     |
| <code>size_t length() const</code>                  | Length of the string.   |
| <code>size_t find(char c) const</code>              | Find a character in the string. Returns on first match. Throws an <code>std::out_of_range</code> if it was not found. |
| <code>size_t find(const char *s) const</code>       | Find a c-string in the string. Returns on first match. Throws an <code>std::out_of_range</code> if it was not found.  |
| <code>size_t find(const String &amp;s) const</code> | Find a string in the string. Returns on first match. Throws an <code>std::out_of_range</code> if it was not found.    |
| <code>Iterator begin()</code>                       | Get an iterator pointing to the start of the string.  |
| <code>ConstIterator begin() const</code>            | Get an iterator pointing to the start of the const string.  |
| <code>Iterator end()</code>                         | Get an iterator pointing to the character after the end of the string.  |
| <code>ConstIterator end() const</code>              | Get an iterator pointing to the character after the end of the const string.  |
| <code>~String()</code>                              | Destroy the String object.  |

## Members

`class ConstIterator`

Iterator for constant string.

`class DataBlock`

Linked list element to store data.

`class DataCache`

Cache for indexing string.

`class Iterator`

Iterator for string.

`String()`

Construct a new empty String object.

`explicit String(char c)`

Construct a new String object from a single char.

## Parameters

- `c` character

```
explicit String(const char *s)
```

Construct a new String object from a const char \*.

## Parameters

- `s` pointer to a c-string, must include `\0` at the end

```
String(const String &s)
```

Copy constructor for string.

## Parameters

- `s` string to be copied

```
String &operator=(char c)
```

Assign the value of a single char to the string. The previous contents of the string are discarded.

## Parameters

- `c` character

## Returns

reference of the current string, can be daisy-chained

```
String &operator=(const char *s)
```

Assign the value of a c-string to the string. The previous contents of the string are discarded.

## Parameters

- `s` c-string

## Returns

reference of the current string, can be daisy-chained

```
String &operator=(const String &s)
```

Assign the value of another string to the string. The previous contents of the string are discarded.

## Parameters

- `s` string that the value should be copied from

## Returns

reference of the current string, can be daisy-chained

```
char &operator[](size_t idx)
```

Indexing operator. Throws an `std::out_of_range` exception in case of over or underindexing.

## Parameters

- `idx` index of the desired character

## Returns

reference to the indexed character

```
char operator[](size_t idx) const
```

Constant indexing operator. Throws an `std::out_of_range` exception in case of over or underindexing.

## Parameters

- `idx` index of the desired character

## Returns

indexed character

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

Compare with a single character. Only can be equal if the string has a length of 1.

## Parameters

- `c` character to be compared with

## Returns

true string and character are the same

## Returns

false otherwise

```
bool operator==(const char *s) const
```

Compare with a c-string. Only can be equal if the length is the same.

## Parameters

- `s` c-string to be compared with

## Returns

true string and c-string are equal

## Returns

false otherwise

```
bool operator==(const String &s) const
```

Compare with another string. Only can be equal if the length is the same.

## Parameters

- `s` string to be compared with

## Returns

true two strings are the same

## Returns

false otherwise

```
bool operator!=(char c) const
```

Compare with a single character. Only can be equal if the string has a length of 1.

## Parameters

- `c` character to be compared with

## Returns

false string and character are the same

## Returns

true otherwise

```
bool operator!=(const char *s) const
```

Compare with a c-string. Only can be equal if the length is the same.

## Parameters

- `s` c-string to be compared with

## Returns

false string and c-string are equal

## Returns

true otherwise

```
bool operator!=(const String &s) const
```

Compare with another string. Only can be equal if the length is the same.

## Parameters

- `s` string to be compared with

## Returns

false two strings are the same

## Returns

true otherwise

```
String operator+(char c) const
```

Add a character to the end of the string. Does not change original string.

## Parameters

- `c` character to be added

## Returns

a new string with the character on the end

```
String operator+(const char *s) const
```

Add a c-string to the end of the string. Does not change original string.

## Parameters

- `s` c-string to be added

## Returns

a new string with the c-string added onto the end

```
String operator+(constString &s) const
```

Add a string to the end of the string. Does not change original string.

## Parameters

- `s` string to be added

## Returns

a new string with the string added to the end

```
String &operator+=(char c)
```

Append a character to the current string. Changes the current string.

## Parameters

- `c` character to be added

## Returns

reference to the current string, which includes the character

```
String &operator+=(const char *s)
```

Append a c-string to the current string. Changes the current string.

## Parameters

- `s` c-string to be added

## Returns

reference to the current string, which includes the c-string

```
String &operator+=(const String &s)
```

Append a string to the current string. Changes the current string.

## Parameters

- `s` string to be added

## Returns

reference to the current string, which includes the string

```
String substr(size_t from, size_t length) const
```

Create a substring from the string.

## Parameters

- `from` index of the first character of the substring, inclusive
- `length` count of the characters of the substring

## Returns



a new string containing the desired part of the string

```
const char *c_str(bool autoFree = true) const
```

Create a c-string closed by a \0.

### Parameters

- `autoFree` true: the c-string will be automatically deleted when the string destructs false: the user has to delete the memory used for the c-string

### Returns

pointer to the c-string

```
void free_c_str() const
```

Free the memory allocated by `c_string()` before the string destructs.

```
size_t length() const
```

Length of the string.

### Returns

length

```
size_t find(char c) const
```

Find a character in the string. Returns on first match. Throws an `std::out_of_range` if it was not found.

### Parameters

- `c` character to be found

### Returns

index of the character

```
size_t find(const char *s) const
```

Find a c-string in the string. Returns on first match. Throws an `std::out_of_range` if it was not found.

### Parameters

- `s` c-string to be found

### Returns

the index of the first character of the c-string

```
size_t find(const String &s) const
```

Find a string in the string. Returns on first match. Throws an `std::out_of_range` if it was not found.

### Parameters

- `s` string to be found

### Returns

index of the first character of the string

```
Iterator begin()
```

Get an iterator pointing to the start of the string.

### Returns

iterator

```
ConstIterator begin() const
```

Get an iterator pointing to the start of the const string.

### Returns

const iterator

```
Iterator end()
```

Get an iterator pointing to the character after the end of the string.

### Returns

iterator

```
ConstIterator end() const
```

Get an iterator pointing to the character after the end of the const string.

### Returns

const iterator

```
~String()
```

Destroy the String object.

```
class DataBlock
```

---

Linked list element to store data.

## Summary

| Members                      | Descriptions                  |
|------------------------------|-------------------------------|
| <code>DataBlock *prev</code> | Pointer to prevoius datablock |
| <code>char data</code>       | Actual data...                |
| <code>DataBlock *next</code> | Pointer to next datablock     |

## Members

`DataBlock *prev`

`char data`

`DataBlock *next`

## class Iterator

Iterator for string.

## Summary

| Members   | Descriptions                                  |
|---|---|
| <code>Iterator(const Iterator &amp;it)</code>                   | Copy constructor for iterator.                |
| <code>Iterator &amp;operator++()</code>                         | Increment iterator.                           |
| <code>Iterator &amp;operator--()</code>                         | Decrement iterator.                           |
| <code>char &amp;operator*() const</code>                        | Get the character referenced by the iterator. |
| <code>bool operator==(const Iterator &amp;it) const</code>      | Compare with another iterator.                |
| <code>bool operator==(const ConstIterator &amp;it) const</code> | Compare with another const iterator.          |
| <code>bool operator!=(const Iterator &amp;it) const</code>      | Compare with another iterator.                |
| <code>bool operator!=(const ConstIterator &amp;it) const</code> | Compare with another const iterator.          |
| <code>~Iterator()</code>  | Destroy the Iterator object.                  |

## Members

`Iterator(const Iterator &it)`

Copy constructor for iterator.

### Parameters

- `it` iterator to be copied

```
Iterator &operator++()
```

Increment iterator.

### Returns

reference to the incremented iterator

```
Iterator &operator--()
```

Decrement iterator.

### Returns

Reference to the decremented iterator

```
char &operator*() const
```

Get the character referenced by the iterator.

### Returns

reference to the character

```
bool operator==(const Iterator &it) const
```

Compare with another iterator.

### Parameters

- `it` iterator to be compared with

### Returns

true iterators point to the same character

### Returns

false otherwise

```
bool operator==(const ConstIterator &it) const
```

Compare with another const iterator.

### Parameters

- `it` iterator to be compared with

### Returns

true iterators point to the same character

### Returns

false otherwise

```
bool operator!=(const Iterator &it) const
```

Compare with another iterator.

### Parameters

- `it` iterator to be compared with

### Returns

false iterators point to the same character

### Returns

true otherwise

```
bool operator!=(const ConstIterator &it) const
```

Compare with another const iterator.

### Parameters

- `it` iterator to be compared with

### Returns

false iterators point to the same character

### Returns

true otherwise

```
~Iterator()
```

Destroy the Iterator object.

## class ConstIterator

---

Iterator for constant string.

## Summary

| Members   | Descriptions   |
|---|--|
| <code>ConstIterator(const ConstIterator &amp;it)</code>         | Copy constructor for const iterator.                 |
| <code>explicit ConstIterator(const Iterator &amp;it)</code>     | Construct a new Const Iterator object from Iterator. |
| <code>ConstIterator &amp;operator++()</code>                    | Increment iterator.                                  |
| <code>ConstIterator &amp;operator--()</code>                    | Decrement iterator.                                  |
| <code>char operator*() const</code>                             | Get the character referenced by the iterator.        |
| <code>bool operator==(const Iterator &amp;it) const</code>      | Compare with another iterator.                       |
| <code>bool operator==(const ConstIterator &amp;it) const</code> | Compare with another const iterator.                 |
| <code>bool operator!=(const Iterator &amp;it) const</code>      | Compare with another iterator.                       |
| <code>bool operator!=(const ConstIterator &amp;it) const</code> | Compare with another const iterator.                 |
| <code>~ConstIterator()</code>                                   | Destroy the Const Iterator object.                   |

## Members

`ConstIterator(const ConstIterator &it)`

Copy constructor for const iterator.

### Parameters

- `it` iterator to be copied

`explicit ConstIterator(const Iterator &it)`

Construct a new Const Iterator object from Iterator.

### Parameters

- `it` iterator to be copied

`ConstIterator &operator++()`

Increment iterator.

### Returns

reference to the incremented iterator

`ConstIterator &operator--()`

Decrement iterator.

### Returns

Reference to the decremented iterator

```
char operator*() const
```

Get the character referenced by the iterator.

### Returns

character

```
bool operator==(const Iterator &it) const
```

Compare with another iterator.

### Parameters

- `it` iterator to be compared with

### Returns

true iterators point to the same character

### Returns

false otherwise

```
bool operator==(const ConstIterator &it) const
```

Compare with another const iterator.

### Parameters

- `it` iterator to be compared with

### Returns

true iterators point to the same character

### Returns

false otherwise

```
bool operator!=(const Iterator &it) const
```

Compare with another iterator.

### Parameters

- `it` iterator to be compared with

### Returns

false iterators point to the same character

### Returns

true otherwise

```
bool operator!=(const ConstIterator &it) const
```

Compare with another const iterator.

### Parameters

- `it` iterator to be compared with

### Returns

false iterators point to the same character

### Returns

true otherwise

```
~ConstIterator()
```

Destroy the Const Iterator object.

## class DataCache

---

Cache for indexing string.

## Summary

| Members  | Descriptions  |
|--|---|
| <pre>explicit<br/>DataCache(String &amp;s)</pre> | Construct a new DataCache object from a string.   |
| <pre>DataCache(const<br/>DataCache &amp;c)</pre> | Copy constructor for DataCache.   |
| <pre>char &amp;getChar(size_t<br/>idx)</pre>     | Get a character from the string. Does not check for overindexing.                       |
| <pre>void reset()</pre>                          | reset the cache to the original state. Recommended to run when the linked list changes. |
| <pre>~DataCache()</pre>                          | Destroy the DataCache object.   |



## Members

```
explicit DataCache(String &s)
```

Construct a new DataCache object from a string.

### Parameters

- `s` string for caching

```
DataCache(const DataCache &c)
```

Copy constructor for DataCache.

### Parameters

- `c` DataCache to be copied

```
char &getChar(size_t idx)
```

Get a character from the string. Does not check for overindexing.

### Parameters

- `idx` index of the desired character

### Returns

reference for the indexed character

```
void reset()
```

reset the cache to the original state. Recommended to run when the linked list changes.

```
~DataCache()
```

Destroy the DataCache object.