CS2310 Computer Programming

Special Content: string VS cstring

Computer Science, City University of Hong Kong

Semester A 2023-24

Strings

```
#include <string>

...

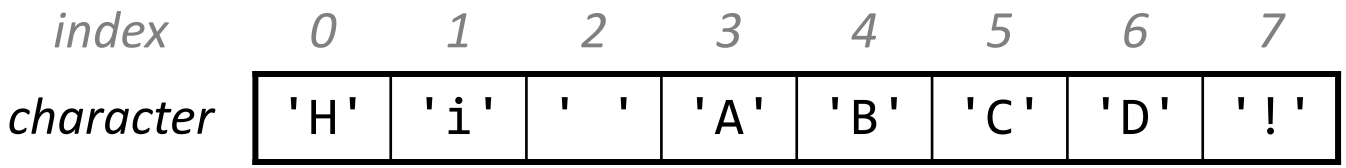
string s = "hello";
```

- A std::string can also represent (possibly empty) sequence of characters.
- std::string are *mutable* (can be changed) in C++.
  - As opposed to cstring
  - std::string is a class in C++

Accessing Characters

- Characters are still values of type `char`, with 0-based indexes:

```
string s = "Hi ABCD!";
```



- Individual characters can be accessed using `[`***`index`***`]` or `at`:

```
char c1 = s[3];                 // 'A'
char c2 = s.at(1);              // 'i'
```

Operator Overload

- **Concatenate** using + or += :

```
string s1 = "Dai";
s1 += "sy";                          // "Daisy"
```

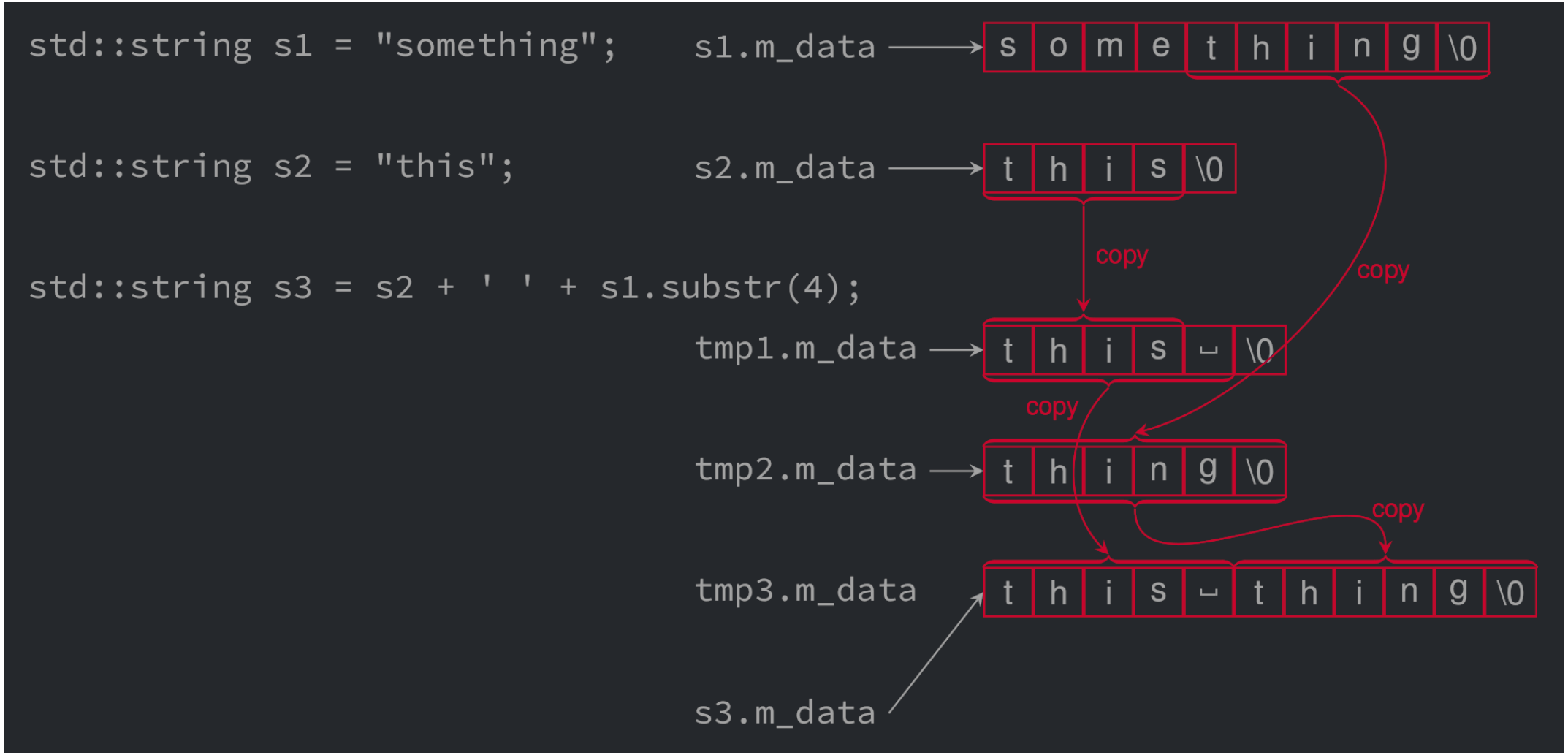
- **Compare** using relational operators (ASCII ordering):

```
string s2 = "Nick";            // == != < <= > >=
if (s1 < s2 && s2 != "Joe") {  // true
    ...
}
```

- Strings are **mutable** and can be changed (!):

```
s2.append(" Troccoli");              // "Nick Troccoli"
s2.erase(6, 7);                  // "Nick T"
s2[2] = '<';                     // "Ni<k T"
```

string Allocations and Copies



Operator Overload is Efficient

```
std::string_view s1 = "something";

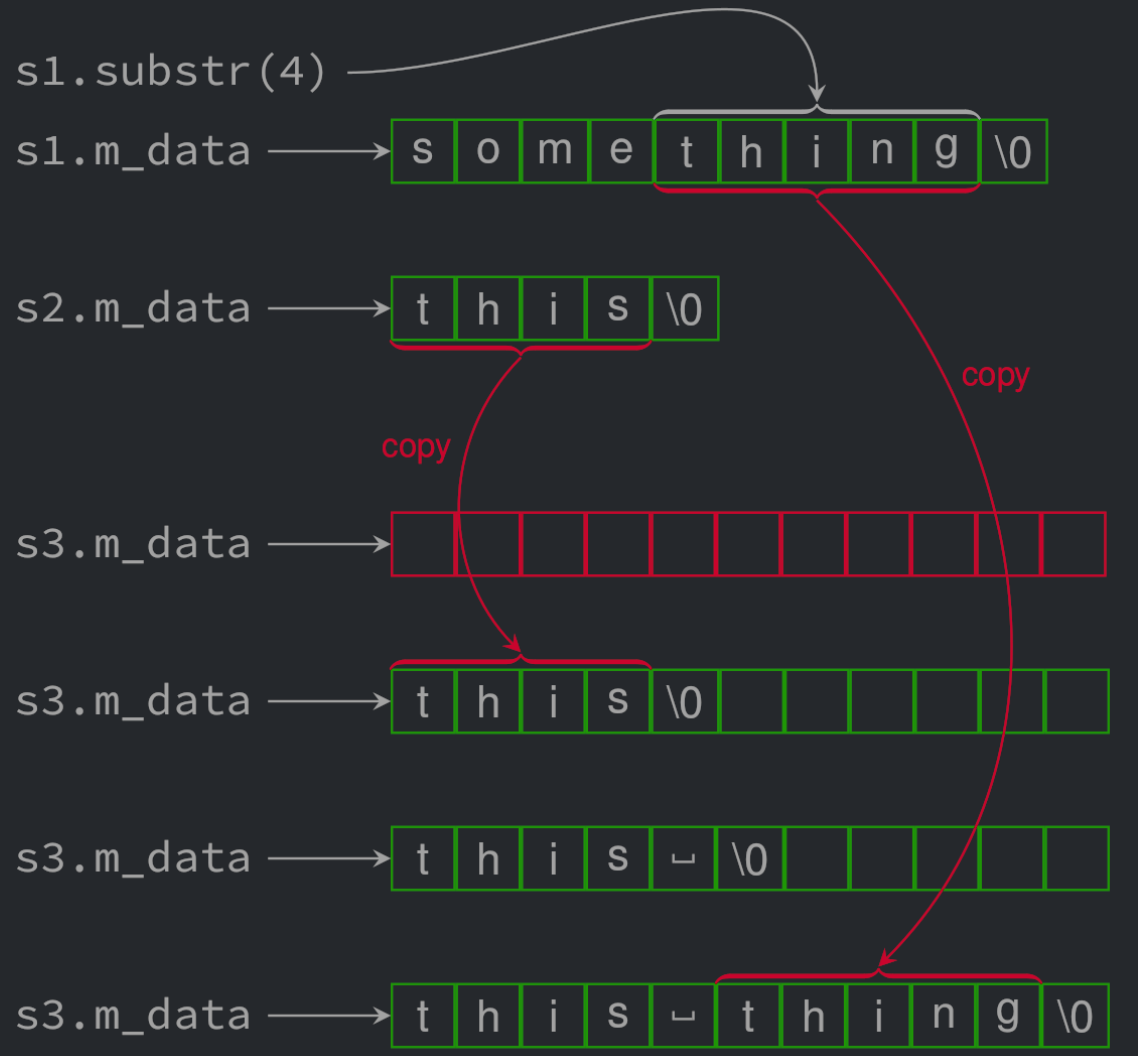
std::string_view s2 = "this";

std::string s3;
s3.reserve(11);

s3 += s2;

s3 += ' ';

s3 += s1.substr(4);
```



Member functions

Member function name	Description
`s.append(str)`	add text to the end of a string
`s.compare(str)`	return -1, 0, or 1 depending on relative ordering
`s.erase(index, length)`	delete text from a string starting at given index
`s.find(str)` `s.rfind(str)`	first or last index where the start of ***str*** appears in this string (returns `string::npos` if not found)
`s.insert(index, str)`	add text into a string at a given index
`s.length()` or `s.size()`	number of characters in this string
`s.replace(index, len, str)`	replaces *len* chars at given index with new text
`s.substr(start, length)` or `s.substr(start)`	the next *length* characters beginning at *start* (inclusive); if *length* omitted, grabs till end of string

```
string name = "Nick Troccoli";
if (name.find("Troccoli") != string::npos) {
    name.erase(6, 7);   // Nick T
}
```

String exercise

- Write a function **`nameDiamond`** that accepts a string parameter and prints its letters in a "diamond" format as shown below.
  - For example, `nameDiamond("DAISY")` should print:

Exercise solution

```
void nameDiamond(string& name) {
    // print top half of diamond
    for (int i = 1; i <= name.length(); i++) {
        cout << name.substr(0, i) << endl;
    }
    // print bottom half of diamond
    for (int i = 1; i < name.length(); i++) {
      for (int j = 0; j < i; j++) { // indent
          cout << " "; // with spaces
          }
          cout << name.substr(i, name.length() - i) << endl;
    }
}
```

```
D
DA
DAI
DAIS
DAISY
```

```
 AISY
  ISY
   SY
    Y
```

String user input

- `cin` reads string input, but only a word at a time:

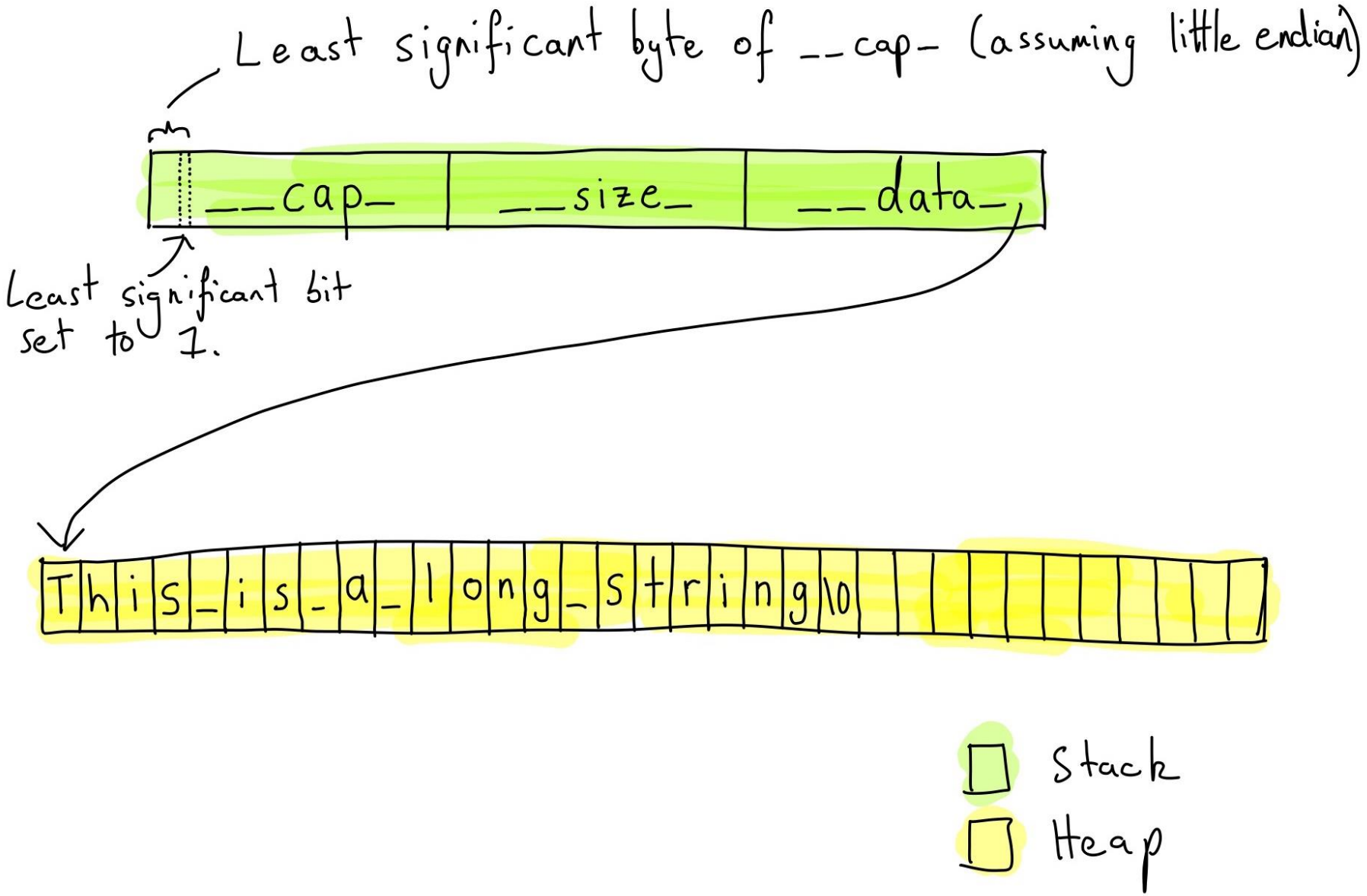
```
cout << "Type your name: ";
string name;
cin >> name;          // Type your name: John Doe
cout << "Hello, " << name << endl; // Hello, John
```

- C++ standard lib **`getline`** function is similar:

```
std::string name;
std::cout << "Type your name: ";
std::getline(std::cin, name);
std::cout << "Hello, " << name << std::endl;
```

Data Layout

- Short string mode
  - store up to 22 characters without heap allocation
- Long string mode
  - standard string implementation
  - capacity, size, and a pointer



C vs. C++ strings

- C++ has two kinds of strings:
  - **C strings** (`char` arrays) and **C++ strings** (`string` objects)

- A string literal such as `"hi there"` is a C string.
  - C strings don't include any methods/behavior shown previously.
    - No member functions like `length`, `find`, or operators.

- Converting between the two types:
  - `string("`***`text`***`")` C string to C++ string
  - ***`string`***`.c_str()` C++ string to C string

C string bugs

- `string s = "hi" + "there";   // C-string + C-string`
- `string s = "hi" + '?';       // C-string + char`
- `string s = "hi" + 41;        // C-string + int`
  - C strings can't be concatenated with +.
  - C-string + char/int produces garbage, not "hi?" or "hi41".
  - **This bug usually appears in print statements, and you'll see partial strings.**
- ```
  string s = "hi";
  s += 41;                       // "hi)"
  ```
  - Adds character with ASCII value 41, `')'`, doesn't produce `"hi41"`.
- `int n = (int) "42";          // n = 0x7ffdcb08`
  - Bug; sets n to the memory address of the C string `"42"` (ack!).
  ```

C string bugs fixed

- `string s = string("hi") + "there";`
- `string s = "hi";  // convert to C++ string`
  `s += "there";`
  - These both compile and work properly.
- `string s = "hi";  // C++ string + char`
  `s += '?';  // "hi?"`
  - Works, because of auto-conversion.
- `s += std::to_string(41);  // "hi?41"`
- `int n = std::stoi("42");  // 42`
  - Both to_string and stoi functions can be found in the <string> header