1 the `string` class

1.1 Introduction

So far, the only way to manipulate character strings is by using direct pointers to arrays of chars. To copy, concatenate, or pass by value, this type is really inefficient.

The standard C++ distribution provides a very powerful type `string`. The underlying structure of this type is an array of char with a reference counter to avoid superfluous copies.

1.2 Example

```
#include <string>

int main(int argc, char **argv) {
  string s = "What a beautiful weather!!!";
  string t;
  t = s;
  cout << t << '\n';
}
```

1.3 Principal member functions and operators

`string()`	constructor
`string(const string &s)`	constructor
`string(const string &s, int pos, int n)`	constructor
`string(const char *s, int size)`	constructor
`string(const char *s)`	constructor
`string(int n, char c)`	constructor
`int length()`	number of characters in the string
`bool empty()`	is the string empty?
`char &operator [int k]`	access the n-th character
`string &operator =`	assignment (from other string, or char
`string &operator +`	concatenation
`void swap(string &s)`	permutes both strings
`int find(const string &sub, int from)`	find the substring
`string substr(int pos, int length)`	extract a substring
`bool operator == (const string &s1, const string &s2)`	compare two strings

1.4 example

```
#include <iostream>
#include <string>

void something(string s) {
  cout << "s = [" << s << "]\n";
  s[0] = 'X';
  cout << "s = [" << s << "]\n";
}

int main(int argc, char **argv) {
  string s1 = "University";
  string s2 = " of ";
  string s3(" Chicago");
  string s4;
  s4 = s1 + s2;
  s4 += s3;
  cout << s4 << "\n";
  s1 = s4.substr(11, 2);
  cout << s1 << "\n";
  something(s1);
  cout << s1 << "\n";
}
```

```
University of  Chicago
of
s = [of]
s = [Xf]
of
```