STL: string

В C++ имеется несколько типов для «текстовых объектов»

1. C-стринг: `char*` и `const char*`
2. классы `string` и `wstring`, а начиная с C++11 `u16string`, `u32string`
3. `string_view`: строка «ссылка» (read-only) на внешнюю последовательность символов в C++17

Мотивация введения string

☞ Класс поведение которого подобно поведению «встроенных» типов:

- ✓ присваивание с помощью `=`
- ✓ сравнение с помощью `==, <, > ...`
- ✓ слияние с помощью `+, +=`
- ✓ выделение части текста, поиск, замена и другое

STL string: создание и копирование

```
#include <string>
using namespace std;
```

конструкторы, пример

```
string s0;                          // with no arguments: empty string
cout << "s0 is: " << s0 << endl; // s0 is:

string s1 ("Initial string");     // one argument: text in quotes
cout << "s1 is: " << s1 << endl; // s1 is: Initial string

string s2 = "Second string";      // string with assignment
cout << "s2 is: " << s2 << endl; // s2 is: Second string

// String by repeating one character: 1st arg: number characters,
string s3 (15,'*');                 // 2nd arg: character itself
cout << "s3 is: " << s3 << endl;   // s3 is: ***************
```

создание string из уже имеющегося текста

```
const char* line = "short line for testing";
string s4 (line);                 // full copy: ctor
cout << "s4 is: " << s4 << endl; // s4 is: short line for testing

// take only the first characters:  1st arg.- begin of c-string
string s5 (line,10);              // 2nd arg.- number of characters
cout << "s5 is: " << s5 << endl; // s5 is: short line

// copy substring by index:          1st arg.- stl-string
                                  // 2nd arg.- start index position
string s6 (s5,6,4);               // 3d arg.- number of characters
cout << "s6 is: " << s6 << endl; // s6 is: line

// copy substring using iterators:   1st arg: start iterator
string s7 (begin(s4),end(s4)-5);  // 2nd arg: end iterator
cout << "s7 is: " << s7 << endl;  // s7 is: short line for te
```

STL string: raw string literals C++11

☞ В C++11 появился удобный способ задания текста содержащего символы обратной косой черты '\' и новой строки '\n'

старый способ C++98

```
string t1="C:\\A\\B\\file.txt";
string t2="First\\nSecond\\nThird";
string t3="First\nSecond\nThird";
cout<<t1<<endl<<t2<<endl<<t3<<endl;
```

```
C:\A\B\file.txt
First\nSecond\nThird
First
Second
Third
```

новый в C++11

```
string rt1=R"(C:\A\B\file.txt)";
string rt2=R"(First\nSecond\nThird)";
string rt3=R"(First
Second
Third)";
cout<<rt1<<endl<<rt2<<endl<<rt3<<endl;
```

```
C:\A\B\file.txt
First\nSecond\nThird
First
Second
Third
```

STL string: элементарные операции

Размер и ёмкость

`size(), length()`	размер, длина стринга	
`empty()`	`true` для пустого стринга	
`capacity()`	размер зарезервированной памяти	
`reserve(num)`	запрос на резервирование памяти	
`resize(...)`	удаляет или добавляет символы в конец	
`clear()`	удаляет все элементы	
`shrink_to_fit()`	уменьшает `capacity` до `size()`	C++11

Доступ к отдельным символам текста

`[i], at(i)`	доступ к i-му символу	
`front(), back()`	первый и последний символы	C++11

Лексикографическое сравнение

`==, !=, <, <=, >, >=, compare()`

Модификация стринга

`+`	соединение, конкатенация
`+=,append(),push_back()`	добавление в конец
`insert()`	вставка символов в середину
`erase(),pop_back()`	удаление символов
`replace()`	замена части стринга

Полезные функции

`c_str(),data()`	возвращает C-string
`substr(pos,len)`	возвращает часть стринга
`find(str)`	поиск в стринге
`», «, getline()`	операции ввода/вывода в поток

функции find(), substr()

```
string tst("K+ pi- K- pi+");
cout << "find 'pi' in a string [" << tst << "]\n"; // [K+ pi- K- pi+]
auto p = tst.find(string("pi"));
if ( p != string::npos ) { // npos == not found
   string sub = tst.substr(p,3);
   cout << "found [" << sub << "]\n"; // found [pi-]
}
```

☞ `static const size_type npos = -1;` специальное значение для «неуспеха»: не найдена позиция символа, или ошибки в функции

функции sort() для string

```
string ts1("K+ K- eta");
cout << "sort string   [" << ts1 << "]\n"; // sort string   [K+ K- eta]
sort(begin(ts1),end(ts1));
cout << "after sorting [" << ts1 << "]\n"; // after sorting [  +-KKaet]
```

преобразование к C-string c_str() и функция data()

```
string sts("Hello wold!\n");
printf("%s",sts.c_str()); // Hello wold!
printf("%s",sts.data());  // Hello wold!
```

- текст хранящийся в string можно передавать в «C-функции»
- data() возвращает указатель на первый символ string C++11

STL: ввод-вывод в string

string stream: #include <sstream>

☞ позволяет связать string с потоком ввода-вывода и затем использовать имеющиеся функции для записи-чтения в стринг

- **«запись» Rational → string**

```
stringstream ss;                          // поток <-> пустой стринг
ss << Rational(13,17);                    // запись в поток
string mystr = ss.str();                  // извлечение стринга
cout << "mystr [" << mystr << "]\n"; // mystr [13/17]
```

- **«чтение» string → Rational**

```
stringstream tt(mystr);                 // поток <-> mystr
Rational myr;
tt >> myr;                              // чтение из потока
cout << "myr= " << myr << endl;      // myr= 13/17
```

Преобразования стринг $\leftrightarrow$ число

функции преобразования в C++11

`stoi(),stol(),stoll()`	к знаковому целому
`stoul(),stoull`	к без-знаковому целому
`stof(),stod(),stold()`	к числу с плавающей точкой
`to_string(),to_wstring()`	преобразует `Int` или `Float` к стрингу

- `int` $\rightarrow$ `string` **и преобразует как** `'%d'` **в** `printf`

```
int i = 12345;
string si = "'" + to_string(i) + "'";
cout << "si= " << si << endl; // si= '12345'
```

- `double` $\rightarrow$ `string` **и преобразует как** `'%f'` **в** `printf`

```
cout << "pi= '" << to_string(M_PI) << "'\n";  // pi= '3.141593'
cout << "eps= '" << to_string(1e-7) << "'\n"; // eps= '0.000000'
```

новые функции в C++17: from_chars() и to_chars()

```
#include <charconv> // conversion functions in C++17
```

from_chars() string → int — C++17

```
string Si = " 12345 mm";
int ival = 0;
auto [ptr, ec] = from_chars( Si.data()+1, Si.data()+Si.size(), ival );
if ( ec == errc() ) { // check error code
   cout << "ival= " << ival << endl;  // ival= 12345
   cout << "ptr-> '" << ptr << "'\n"; // ptr-> ' mm'
} else {
   cout << "Error: " << make_error_code(ec).message() << endl;
}
```

from_chars() string → double — C++17

```
string Sd = "1.e-7";
double dval = 0;
from_chars( Sd.data(), Sd.data()+Sd.size(), dval ); // no error check
cout << "dval= " << dval << endl;  // dval= 1e-07
```

to_chars() int → string — C++17

```
string St1(10,'_'); // ten underscores
auto [pt1, ec1] = to_chars( St1.data(), St1.data()+St1.size(), 12345 );
if ( ec1 == errc() ) { // check error code
   cout << "St1= '" << St1 << "'\n";          // St1= '12345_____'
   cout << "Nch= " << pt1-St1.data() << endl; // Nch= 5
} else {
   cout << "Error: " << make_error_code(ec1).message() << endl;
}
```

to_chars() double → string — C++17

```
string St2(20,' ');
to_chars( St2.data(), St2.data()+St2.size(), M_PI );
cout << "St2= '" << St2 << "'\n"; // St2= '3.141592653589793    '
St2 = string(10,' ');
to_chars( St2.data(), St2.data()+St2.size(), 1.2345e-7 );
cout << "St2= '" << St2 << "'\n"; // St2= '1.2345e-07'
```

☞ преобразование с double поддерживается не всеми компиляторами

Регулярные выражения (regex) C++11

Регулярное выражение: «шаблон» для поиска в тексте

- имеется формальный язык для написания таких шаблонов
- функции для манипуляции текстом: поиск, замена, удаление

удаление всех цифр из текста

```
#include <regex> // regular expression
auto RmDigits = [](const string& str)-> string {
   regex dig_re("[0-9]");                  // любая цифра
   return regex_replace(str, dig_re, ""); // замена на пустой стринг
};
string test="D5_Kp6_ct";
string out = RmDigits(test);
cout << "RmDigits: " << test << " -> " << out << endl;
// RmDigits: D5_Kp6_ct -> D_Kp_ct
```

взятие чисел в квадратные скобки

```
#include <regex> // regular expression
auto BrDig = [](const string& str)-> string {
   // шаблон для цифры взятой один или более раз
   static regex dig_re("[[:digit:]]+");
   // замена с подстановкой: $& - то что найденно по шаблону
   return regex_replace(str, dig_re, "[$&]");
};
string t = "a1 a2 a3 a44";
out = BrDig(t);
cout << "BrDig: " << t << " -> " << out << endl;
// BrDig: a1 a2 a3 a44 -> a[1] a[2] a[3] a[44]
```

C++ Библиотека ввода/вывода

В C++ имеются две библиотеки ввода/вывода

- ☞ **«C-стиль»**: `<cstdio>` с функциями `printf()`, `scanf()` ...
- ☞ **«C++стиль»**: `<iostream>` с объектно ориентированной структурой

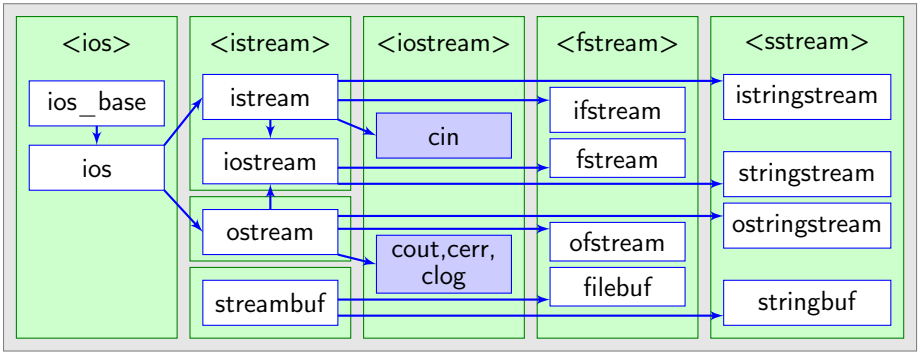
Преимущества <iostream>

- ✓ **Расширяемость:** легко добавить операторы $\ll$ и $\gg$ для пользовательских классов
- ✓ **Наследуемость:** стандартный набор классов для потоков можно наследовать и строить собственные классы

Преимущество <cstdio>

- ✓ **Эффективность:** ввод-вывод в `cstdio` работает быстрее

Цепочка наследования классов



- `<ios_base>` — базовые классы
- `<istream>, <ostream>` — основные классы ввода/вывода: `istream` поток для чтения, `ostream` поток для записи
- `<fstream>` — ввод/вывод в файл
- `<sstream>` — ввод/вывод в `string`

Манипуляторы

☞ Специальные потоковые объекты для управления вводом/выводом

основные манипуляторы определены в <iostream>

- `endl` — (end of line) выводит `'\n'` и очищает буфер
- `flush` — очистка буфера
- `ends` — (end of string) выводит `'\0'`
- `ws` — (white space) читает «пробелы» и пропускает их

некоторые манипуляторы из заголовочного файла <ios>

- `dec,hex,oct` – переключатели системы счисления для целых
- `fixed,scientific` – переключатели для чисел с плавающей точкой

- **некоторые манипуляторы из заголовочного файла `<iomanip>`**
  - `setw()` – ширина следующего вывода или ввода
  - `setfill()` – меняет символ заполнитель
  - `setprecision()` – количество знаков для чисел с плавающей точкой

☞ **пользоваться манипуляторами не слишком удобно:**

```
cout << hex << setfill('0') << setw(8) << i << dec << endl;
printf("0x%08x\n", i);

cout << scientific << setprecision(3) << setw(10) << x << fixed
     << setprecision(15) << setw(20) << y << setprecision(6) << endl;
printf("%10.3e %20.15f\n",x,y);
```

☞ Манипуляторы `hex`,`setprecision()`, `...` меняют поведение всего потока и возврат в значение по умолчанию требует повторного вызова

Файловые потоки

Чтение из файла и запись в файл

```
#include <fstream>                  // file I/O header
ifstream file_in("input.dat");      // открыть файл для чтения
int r;
file_in >> r;                       // extractor (оператор чтения)
cout << "r= " << r << endl;
ofstream file_out("output.out");    // открыть файл для записи
file_out << (r+1) << endl;          // inserter (оператор записи)
```

Возможные ошибки

1. файл `input.dat` может отсутствовать или быть недоступным
2. файл может содержать ошибочный формат данных
3. потеря данных из-за перезаписи уже существующего файла `output.out`

Работа с ошибками

1. **Проверка, что файл успешно открыт**

   ☞ `!file_in` — поток создан без ошибок, синоним `file_in.fail()`

   ☞ `file_in.is_open()` — проверка связан ли поток с файлом

```
// const char* input = "input.dat"; // C++98
string input = "input.dat";         // C++11
ifstream file_in(input);
if( !file_in ) { // check the stream is in good condition
     cerr << "can not open file: " << input << endl;
     exit(EXIT_FAILURE);
}
```

2 file_in.rdstate() возвращает состояние потока

- Состояние кодируется битами: `goodbit` – нет ошибок; `badbit` – поток не функционален; `failbit` – сбой чтения; `eofbit` – конец файла

```
file_in >> r;
ios::iostate state = file_in.rdstate(); // состояние потока
if( state ) {
  if( state & ios::eofbit ) {           // true если конец файла
      cerr << " end of file " << endl;
      exit(EXIT_SUCCESS);
   } else {                             // другие ошибки чтения
      cerr << "error reading file" << endl;
      exit(EXIT_FAILURE);
   }
}
```

Библиотека файловой системы C++17

The Filesystem Library, основное:

- Класс описывающий путь к файлу (path object)
  - манипуляции с путями, информация о путях
  - информация о типе объекта связанного с путем: файл, директория, ссылка ...
  - манипуляции с файлами: копирование, перемещение, создание
- Итераторы по директории, записи `directory_entry`
- Другое: права доступа, временные файлы, специальные файлы ...

Пространство имен `std::filesystem`

```
#include <filesystem>            // header
namespace fs = std::filesystem; // shortcut for std::filesystem
```

пример операций с путем: fs::path

```
fs::path p1("/Users");
cout << p1 << endl;   // "/Users"
p1.append("nefedov"); // adds a path with a directory separator
p1 /= "test";         // synonym for append()
cout << p1 << endl;   // "/Users/nefedov/test"
p1.concat("/new_");   // only adds new characters to the end
p1 += "cpp/a.out";    // synonym for concat()
cout << p1 << endl;   // "/Users/nefedov/test/new_cpp/a.out"

for (const auto& part : p1) { // path iteration
   cout << part << ", ";
}
cout << endl; // "/", "Users", "nefedov", "test", "new_cpp", "a.out"

PathInfo(p1); // следующий слайд
```

Некоторые «информационные» функции

```
void PathInfo(const fs::path& TestPath) {
   cout << "Path info for: " << TestPath << endl;
   cout << "canonical path " << fs::canonical(TestPath) << endl;
   cout << "exists=        " << fs::exists(TestPath) << endl;
   cout << "root_name=     " << TestPath.root_name() << endl;
   cout << "root_path=     " << TestPath.root_path() << endl;
   cout << "relative_path= " << TestPath.relative_path() << endl;
   cout << "parent_path=   " << TestPath.parent_path() << endl;
   if ( fs::is_regular_file(TestPath) ) {
      cout << "Regular file=  " << TestPath.filename() << endl;
      cout << "stem=          " << TestPath.stem() << endl;
      cout << "extension=     " << TestPath.extension() << endl;
      cout << "file size=     " << FileSize(TestPath) << endl; see next
      cout << "file time=     " << FileTime(TestPath) << endl; see next
   }
}
```

Output of FleInfo()

```
Path info for: "/Users/nefedov/test/new_cpp/a.out"
canonical path "/Volumes/CaseSensitive/test/cpp/MyLectures/new_cpp/a.out"
exists=        1
root_name=     ""
root_path=     "/"
relative_path= "Users/nefedov/test/new_cpp/a.out"
parent_path=   "/Users/nefedov/test/new_cpp"
Regular file=  "a.out"
stem=          "a"
extension=     ".out"
file size=     386952
file time=     Wed Apr 23 22:55:27 2024
```

Обратите внимание

- функции члены, такие как `TestPath.filename()`: быстрые, дешовые
- автономные (free-standing) функции, как `fs::exists(TestPath)`: затратные, обычно обращаются к реальной файловой системе

размер файла

```
uintmax_t FileSize( const fs::path& FileName ) {
   if ( fs::exists(FileName) && fs::is_regular_file(FileName) ) {
      auto err = std::error_code;
      auto filesize = fs::file_size(FileName, err);
      return filesize;
   }
   return -1;
}
```

время последней модификации файла

```
string FileTime( const fs::path& FileName ) {
   string ret; // empty string
   if ( fs::exists(FileName) && fs::is_regular_file(FileName) ) {
      auto ftime = fs::last_write_time(FileName);
      ftime += 3h; // UTC -> MSK
      ret = std::format(":%c", ftime); // C++20
   }
   return ret;
}
```

Пример итерации по директории, печать дерева файлов

```
void DirTree(const fs::path& pathToDir, int level=0) {
   if ( fs::exists(pathToDir) && fs::is_directory(pathToDir) ) {
      if( level == 0 ) { cout << "> " << pathToDir << "\n"; }
      auto shift = string(2+level*5,' ');
      for (const auto& entry : fs::directory_iterator(pathToDir)) {
         auto filename = entry.path().filename();
         if ( fs::is_directory(entry.status()) ) {
            cout << shift << "[+] " << filename << "\n";
            DirTree(entry, level + 1);
         } else if ( fs::is_regular_file(entry.status()) ) {
            cout << shift << filename << "\n";
         } else {
            cout << shift << " [?]" << filename << "\n";
         }
      }
   }
}
```

вызов `DirTree()` для рабочей директории

```
const fs::path p4 { fs::current_path() };
DirTree(p4);
```

Output: directory tree

```
> "/Volumes/CaseSensitive/test/cpp/MyLectures/new_cpp"
  "TestFileSystem.cpp"
  "a.out"
  [+] "dir1"
       [+] "dir2"
            "file2"
            [+] "dir3"
                 "file3"
       "file1"
  "has_include.cpp"
```