Программирование на C++

Минцифры России

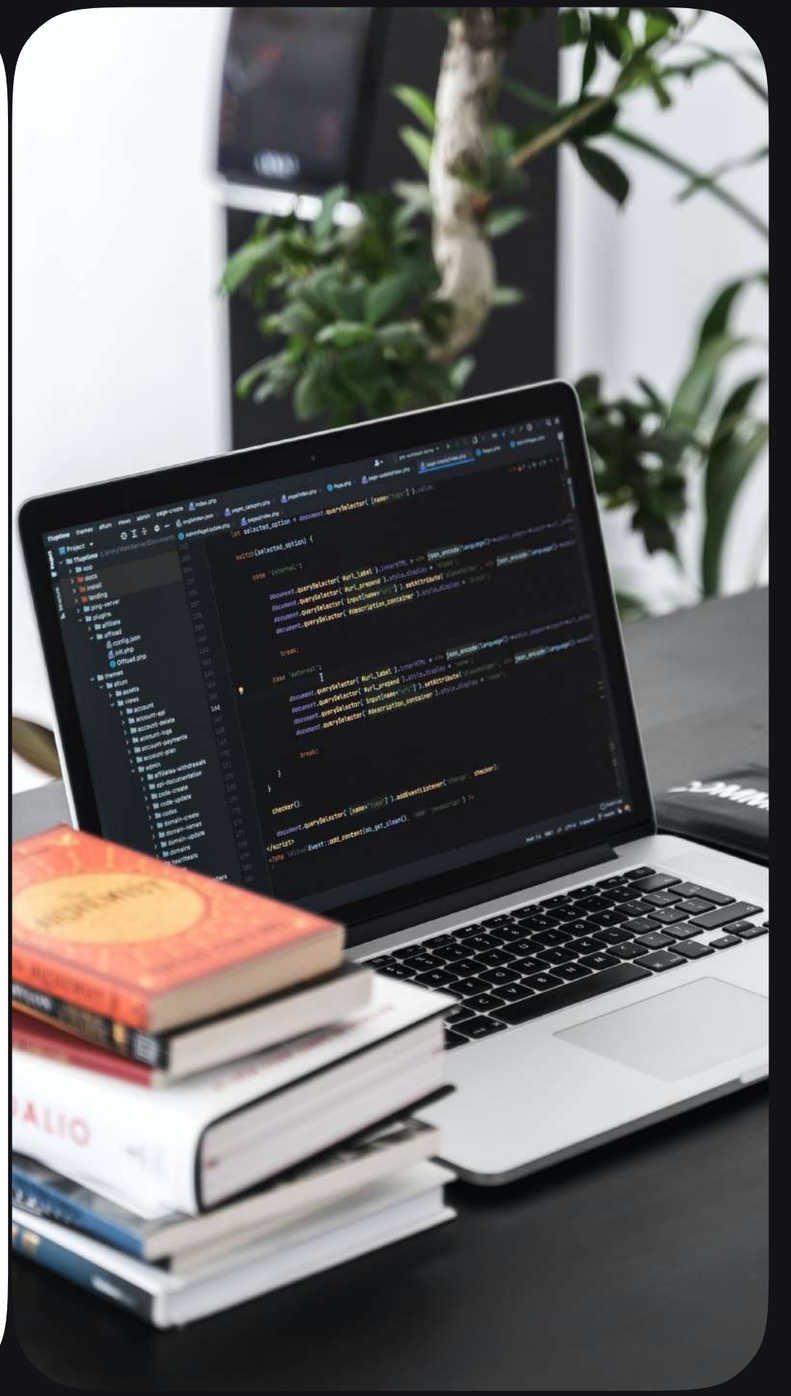
UCHi DOMA

20.35 УНИВЕРСИТЕТ

Урок 7 Модуль 3

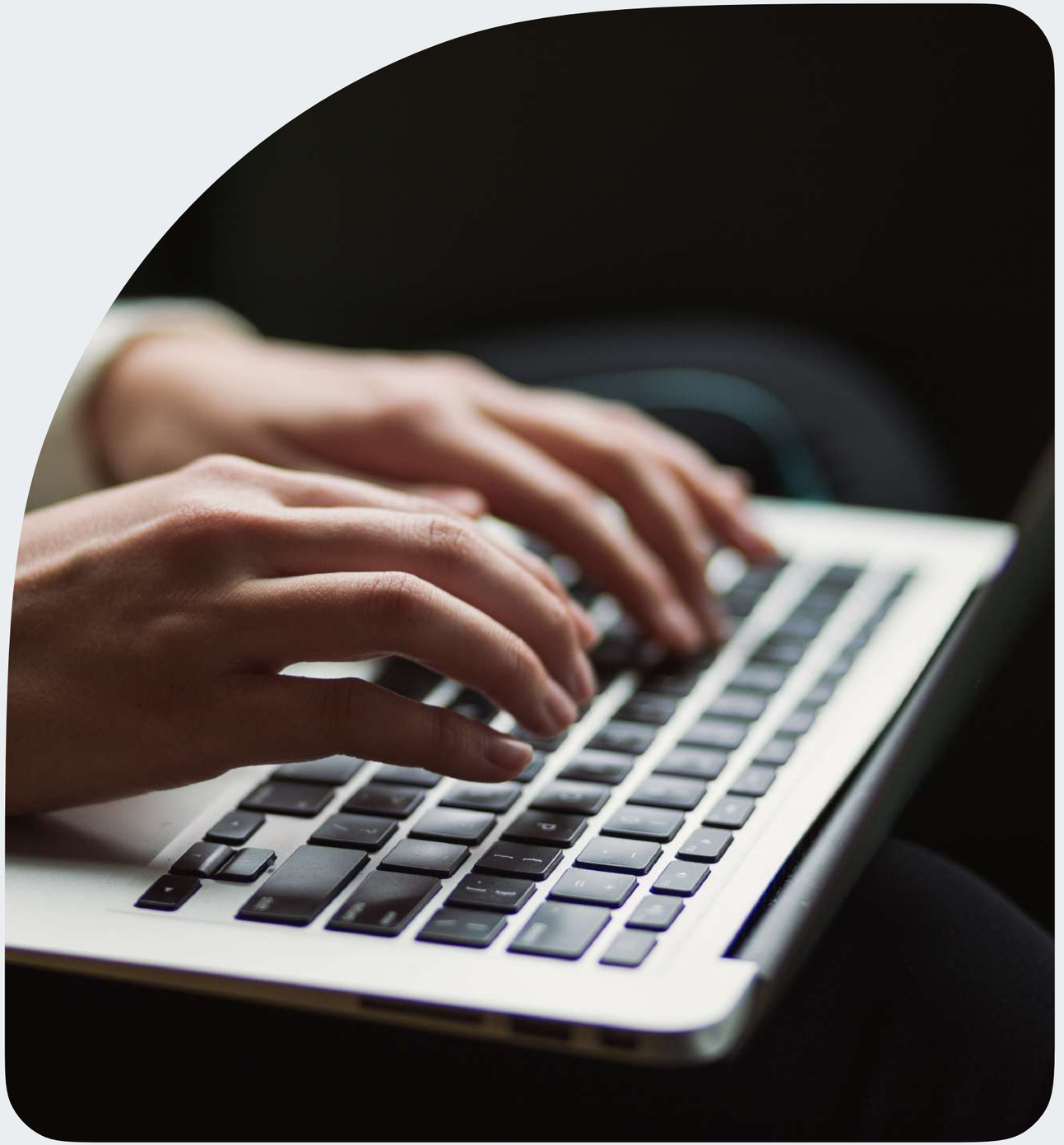
Класс string

Полезные материалы



Цели урока

- изучить класс string
- отработать на практике написание алгоритмов с хранением данных в переменных класса string на C++



Класс string

Класс **string** предназначен для работы со строками типа **char***, которые представляют собой строку с завершающим нулем.

Класс **string** был введенн как альтернативный вариант для работы со строками типа **char***.

Строки, которые завершаются символом ‘**\0**’ еще называются С-строками.

Подключение библиотеки

Чтобы использовать возможности класса **string** необходимо подключить библиотеку **<string>** и пространство имен **std**.

```
#include <string>

using namespace std;
```

Объявление переменной

```
// тип string
string s1; // переменная с именем s1 типа string
string s2 = "This is a string variable"; // объявление с инициализацией
// использование переменной типа string с оператором присваивания
s1 = s2; // s1 = "This is a string variable"
s2 = "New text";
```

Преимущества класса string в сравнении с типом char*?

- ✓ возможность обработки строк стандартными операторами C++ (=, +, = =, <> и т.п.).
- ✓ обеспечение лучшей надежности (безопасности) программного кода. Например, при копировании строк, если строка-источник имеет больший размер чем строка-приемник
- ✓ обеспечение строки, как самостоятельного типа данных, что обеспечивает непротиворечивость данных

Недостатки класса string в сравнении с типом char*?

Основным недостатком типа **string** в сравнении с типом **char***, есть замедленная скорость обработки данных. Это связано с тем, что тип **string** — это, фактически, контейнерный класс. А работа с классом требует дополнительной реализации программного кода, который, в свою очередь занимает лишнее время.

Операторы, применяемые к string

С объектами класса **string** можно использовать следующие операторы:

- **=** — присваивание
- **+** — конкатенация (объединение строк)
- **+=** — присваивание с конкатенацией
- **==** — равенство
- **!=** — неравенство
- **<** — меньше
- **<=** — меньше или равно
- **>** — больше
- **>=** — больше или равно
- **[ ]** — индексация

Примеры операций

```
// тип string, операции над строками
string s1 = "s-1";
string s2 = "s-2";
string s3;
bool b;

// операция '=' (присваивание строк)
s3 = s1; // s3 = "s-1"

// операция '+' - конкатенация строк
s3 = s3 + s2; // s3 = "s-1s-2"

// операция '+=' - присваивание с конкатенацией
s3 = "s-3";
s3 += "abc"; // s3 = "s-3abc"

// операция [] - индексация
char c;
s1 = "abcd";
c = s1[2]; // c = 'c'
c = s1[0]; // c = 'a'
```

Примеры операций

```
// операция '==' - сравнение строк
b = s2==s1; // b = false
b = s2=="s-2"; // b = true

// операция '!=' - сравнение строк (не равно)
s1 = "s1";
s2 = "s2";
b = s1 != s2; // b = true

// операции '<' и '>' - сравнение строк
s1 = "abcd";
s2 = "de";
b = s1 > s2; // b = false
b = s1 < s2; // b = true

// операции '<=' и '>=' - сравнение строк (меньше или равно, больше или равно)
s1 = "abcd";
s2 = "ab";
b = s1 >= s2; // b = true
b = s1 <= s2; // b = false
b = s2 >= "ab"; // b = true
```

Конструкторы класса string

Как и любой класс, класс **string** имеет ряд конструкторов. Основные из них следующие:

```
string();
string(const char * str);
  string(const string & str);
```

Примеры инициализации с помощью конструкторов

```
string s1("Hello!"); // инициализация - конструктор string(const char * str)
string s2 = "Hello!"; // инициализация - конструктор string(const char * str)
char * ps = "Hello";
string s3(ps); // инициализация
string s4(s3); // инициализация - конструктор string(const string & str)
  string s5; // инициализация - конструктор string()
```

Присваивание строк. Функция assign()

Чтобы присвоить одну строку другой, можно применить один из двух методов:

- ✓ использовать оператор присваивания ‘=’
- ✓ использовать функцию **assign()** из класса **string**

Функция assign() имеет несколько реализаций

Присваивание строк. Функция assign()

Первый вариант — это вызов функции без параметров:

```
string &assign(void);
```

В этом случае происходит простое присваивание одной строки другой.

Присваивание строк. Функция assign()

Второй вариант позволяет копировать заданное количество символов из строки:

```
string &assign(const string & s, size_type st, size_type num);
```

где

- **s** — объект, из которого берется исходная строка
- **st** — индекс (позиция) в строке, из которой начинается копирование num символов
- **num** — количество символов, которые нужно скопировать из позиции st
- **size_type** — порядковый тип данных

Присваивание строк. Функция assign()

Третий вариант функции **assign**() копирует
в вызывающий объект первые num символов строки **s**:

```
string & assign(const char * s, size_type num);
```

где

- **s** — строка, которая завершается символом ‘\0’
- **num** — количество символов, которые
копируются в вызывающий объект.
Копируются первые **num** символов из строки **s**.

Пример функции assign()

```
// присваивание строк, функция assign()
string s1 = "doma.uchi.ru";
string s2;
string s3;
char * ps = "doma.uchi.ru";

s3 = s1; // s3 = "doma.uchi.ru"
s2.assign(s1); // s2 = "doma.uchi.ru"
s2.assign(s1, 0, 4); // s2 = "doma"
s2.assign(ps, 9); // s2 = "doma.uchi"
```

Объединение строк. Функция append()

Для объединения строк используется функция **append**(). Для добавления строк также можно использовать операцию ‘+’, например:

```
string s1;
string s2;
s1 = "abc";
s2 = "def";
  s1 = s1 + s2; // s1 = "abcdef"
```

Объединение строк. Функция append()

Однако, функция **append**() хорошо подходит, если нужно добавлять часть строки.

Функция имеет следующие варианты реализации:

```
string &append(const string & s, size_type start);
string &append(const char * s, size_type num);
```

Пример

Объединение строк. Функция append()

```
string s1 = "abcdef";
s2 = "1234567890";
append(s2, 3, 4); // s1 = "abcdef4567"


char * ps = "1234567890";
s1 = "abcdef";
s1.append(ps, 3); // s1 = "abcdef123"
```

Пример

Вставка символов в строке. Функция insert()

Чтобы вставить одну строку в заданную позицию другой строки нужно использовать функцию **insert()**, которая имеет несколько вариантов реализации.

Первый вариант функции позволяет вставить полностью всю строку **s** в заданную позицию **start** вызывающей строки (вызывающего объекта):

```
string & insert(size_type start, const string &s);
```

Второй вариант функции позволяет вставить часть (параметры **insStart**, num) строки **s** в заданную позицию **start** вызывающей строки:

В вышеприведенных функциях:

- **s** — строка, которая вставляется в вызывающую строку
- **start** — позиция в вызывающей строке, из которой осуществляется вставка строки **s**
- **insStart** — позиция в строке **s**, из которой происходит вставка
- **num** — количество символов в строке **s**, которые вставляются с позиции **insStart**

```
string & insert(size_type start, const string &s, size_type insStart, size_type num);
```

Пример

Вставка символов в строке. Функция insert()

```
string s1 = "abcdef";

string s2 = "1234567890";

s1.insert(3, s2); // s1 = "abc"+"1234567890"+"def"="abc1234567890def"

s2.insert(2, s1, 1, 3); // s2 = "12bcd34567890"
```

Замена символов в строке. Функция replace()

Функция **replace()** выполняет замену символов в вызывающей строке. Функция имеет следующие варианты реализации:

```
string &replace(size_type start, size_type num, const string &s);

  string &replace(size_type start, size_type num, const string &s, size_type replStart,
  size_type replNum);
```

В первом варианте реализации вызывающая строка заменяется строкой **s**. Есть возможность задать позицию (**start**) и количество символов (**num**) в вызывающей строке, которые нужно заменить строкой **s**.

Второй вариант функции **replace()** отличается от первого тем, что позволяет заменять вызывающую строку только частью строки **s**. В этом случае задаются два дополнительных параметра: позиция **replStart** и количество символов в строке **s**, которые образуют подстроку, которая заменяет вызывающую строку.

Пример

Замена символов в строке. Функция replace()

```
string s1 = "abcdef";
string s2 = "1234567890";

s2.replace(2, 4, s1); // s2 = "12abcdef7890"
s2 = "1234567890";
s2.replace(3, 2, s1); // s2 = "123abcdef67890"
s2 = "1234567890";
s2.replace(5, 1, s1); // s2 = "12345abcdef7890"

// замена символов, функция replace()
string s1 = "abcdef";
string s2 = "1234567890";

s2.replace(2, 4, s1); // s2 = "12abcdef7890"
s2 = "1234567890";
s2.replace(3, 2, s1); // s2 = "123abcdef67890"

s2 = "1234567890";
s2.replace(5, 1, s1); // s2 = "12345abcdef7890"
s2 = "1234567890";
s2.replace(5, 1, s1, 2, 3); // s2 = "12345cde7890"
s2 = "1234567890";
s2.replace(4, 2, s1, 0, 4); // s2 = "1234abcd7890"
```

Удаление заданного количества символов из строки. Функция erase()

Для удаления символов из вызывающей строки используется функция **erase()**:

```
string & erase(size_type index=0, size_type num = npos);
```

где

- **index** — индекс (позиция), начиная из которой нужно удалить символы в вызывающей строке
- **num** — количество символов, которые удаляются

Пример

Удаление заданного количества символов из строки. Функция erase()

```
string s = "01234567890";

s.erase(3, 5); // s = "012890"

s = "01234567890";

s.erase(); // s = ""
```

Поиск символа в строке. Функция find()

В классе **string** поиск строки в подстроке можно делать двумя способами, которые отличаются направлением поиска:

- Путем просмотра строки от начала до конца с помощью функции **find()**
- Путем просмотра строки от конца к началу функцией **rfind()**

Прототип функции **find()** имеет вид:

```
string & erase(size_type index=0, size_type num = npos);
```

где

- **s** — подстрока, которая ищется в строке, что вызывает данную функцию. Функция осуществляет поиск первого вхождения строки **s**. Если подстрока s найдена в строке, что вызвала данную функцию, тогда возвращается позиция первого вхождения. В противном случае возвращается –1
- **start** — позиция, из которой осуществляется поиск

Поиск символа в строке. Функция rfind()

Прототип функции **rfind()** имеет вид:

```
size_type rfind(const string &s, size_type start = npos) const;
```

где

- **s** — подстрока, которая ищется в вызывающей строке. Поиск подстроки в строке осуществляется от конца к началу. Если подстрока **s** найдена в вызывающей строке, то функция возвращает позицию первого вхождения. В противном случае функция возвращает –1
- **npos** — позиция последнего символа вызывающей строки
- **start** — позиция, из которой осуществляется поиск

Пример

Поиск символа в строке. Функция find()

```
// тип string, функция find()
string s1 = "01234567890";
string s2 = "345";
string s3 = "abcd";
int pos;


pos = s1.find(s2); // pos = 3
pos = s1.find(s2, 1); // pos = 3
pos = s1.find("jklmn", 0); // pos = -1
pos = s1.find(s3); // pos = -1
pos = s2.find(s1); // pos = -1
```

Пример

Поиск символа в строке. Функция rfind()

```
// тип string, функции find() и rfind()
string s1 = "01234567890";
string s2 = "345";
string s3 = "abcd";
string s4 = "abcd---abcd";
int pos;

pos = s1.rfind(s2); // pos = 3
pos = s1.rfind(s2, 12); // pos = 3
pos = s1.rfind(s2, 3); // pos = 3
pos = s1.rfind(s2, 2); // pos = -1
pos = s2.rfind(s1); // pos = -1
pos = s1.rfind(s3, 0); // pos = -1

// разница между функциями find() и rfind()
pos = s4.rfind(s3); // pos = 7
pos = s4.find(s3); // pos = 0
```

Сравнение частей строк. Функция compare()

Поскольку тип **string** — это классом, то, чтобы сравнить две строки между собой можно использовать операцию ‘= =’. Если две строки одинаковы, то результат сравнения будет true. В противном случае, результат сравнения будет **false**.

Но если нужно сравнить часть одной строки с другой, то для этого предусмотрена функция **compare()**.

Прототип функции **compare()**:

```
int compare(size_type start, size_type num, const string &s) const;
```

где

- **s** — строка, которая сравнивается с вызывающей строкой
- **start** — позиция (индекс) в строке **s**, из которой начинается просмотр символов строки для сравнения
- **num** — количество символов в строке **s**, которые сравниваются с вызывающей строкой.

Если вызывающая строка меньше строки **s**, то функция возвращает –1 (отрицательное значение). Если вызывающая строка больше строки **s**, функция возвращает 1 (положительное значение). Если две строки равны, функция возвращает 0.

Пример

Поиск символа в строке. Функция compare.

```
// тип string, функция compare()
string s1 = "012345";
string s2 = "0123456789";
int res;

res = s1.compare(s2); // res = -1
res = s1.compare("33333"); // res = -1
res = s1.compare("012345"); // res = 0
res = s1.compare("345"); // res = -1
res = s1.compare(0, 5, s2); // res = -1
res = s2.compare(0, 5, s1); // res = -1
res = s1.compare(0, 5, "012345"); // res = -1
res = s2.compare(s1); // res = 1
res = s2.compare("456"); // res = -1
res = s2.compare("000000"); // res = 1
```

Определение длины строки типа string. Функция length()

Для определения количества символов в строке используется функция **length()** без параметров.

Пример:

```
string str = "Hello world!";
int len;

// определить длину строки str
len = str.length(); // len = 12
```