Лекция III

13 октября 2017

Работа с текстом и символами в C++

Символьный тип

- **char** - ещё один фундаментальный тип данных в языке C
- Используется для хранения одиночных символов. Сами символы указываются в **одинарных** кавычках, например: 'a', 'b', 'd', '4', '%', '!'
- Символы хранятся в виде **целых чисел**. Фактически, **char** является ещё одним целочисленным типом
- Это единственный тип, размер которого ограничен стандратом языка. Размер **char** **всегда** равен 1 байту
- Но всё просто не бывает: стандарт не обговаривает, должен ли тип **char** быть знаковым или беззнаковым
- Существуют специальные символы, которые в текстовом виде представлены более чем одним знаком: '\n', '\t', '\r'
- Также стандартом определены беззнаковый **unsigned char** и знаковый **signed char** типы

Символы и кодировка

Кодировка - специальная таблица, связывающая каждый символ с соответствующим ему целым числом
Базовая для ЭВМ - **ASCII**. Как правило, входит в любую другую расширенную кодировку. Основные характеристики

- 128 символов
- целочисленные коды: от нуля до 127
- каждый символ занимает один байт
- Включает в себя все цифры, буквы английского алфавита в нижнем/верхнем регистрах и некоторые другие символы (тильда, процент, '@', решётка и т.п.)
- Коды букв и цифр - идут последовательно

Символьный тип

```
1  char symbol = '%';
2  cout << symbol << "\n"; // печатаем знак ↩
       процента
3
4  symbol = '#'; // знак переноса строки
5  cout << "Выводим символ решётки "
6       << symbol << "\n";
7
8  // А поскольку char — целочисленный:
9  symbol = '5';
10 symbol += 2;
11
12 // Выводим на экран семёрку
13 cout << symbol << "\n";
```

Символьный тип

```
1  char sym = '9';
2
3  cout << "\n Значение sym: " << sym;
4  cout << "\n Код sym: " << int(sym)
5              << "\n";
6
7  bool is_less = '2' < sym;
8  // Переменная is_less здесь равна true
9
10 // Печать всех букв английского алфавита
11 sym = 'a'
12 while (sym <= 'z') {
13   cout << sym;
14   ++sym;
15 }
```

Символьный тип

```
1  char option;
2
3  cout << "Хотите продолжать (y/n)?\n";
4  // Значение в переменную типа char
5  // можно считать с консоли так
6  cin >> option;
7
8  while ( option != 'y' || option != 'Y' ) {
9    cout << "А если подумать?\n";
10   // Ещё один вариант получения
11   // одного символа с терминала
12   cin.get( option );
13 }
```

Работа с текстом. С-строки

Строка в С - это одномерный массив значений типа **char**, последним символом которой считается специальный нулевой символ (он же - символ окончания строки): \0 (целочисленный код - 0). Обратите внимание, символ конца строки **не обязан совпадать** с последним элементом массива.

```
1  char phrase1[] = "Первая строка!";
2  std::cout << phrase1;
3
4  char phrase2[] = { 'A', 'B', 'C', '\0' };
5  std::cout << "\nВторой символ в phrase2: "
6            << phrase2[1];
7
8  // А это не строка, а массив
9  // элементов типа char
10 char char_arr[15];
```

Символы и кодировка

Ещё одна кодировка, про которую полезно знать:
UTF8

- символ может состоять от 1 до 4 байт (следовательно, от 8 до 32 бит)
- вводится понятие "code point равное 1 байту (8 битам) символа
- включает в себя все возможные региональные алфавиты
- Коды от 0 до 127 совпадают с кодировкой ASCII

```
1 char word[20];
2
3 // Небезопасно! Следует всегда избегать
4 cin >> word;
```

В чём проблема?

- Оператор ввода >> никак не ограничивает длину последовательности символов: в массив **word** будут помещены все, до первого пробела или переноса строки.
- Символ окончания строки \0 не добавляется в **word**. Формально, он строкой и не является.

Неформатированный ввод

```
cin.getline(str, count)
cin.getline(str, count, delimiter = '\n')
```

1. **str** - массив типа **char**
2. **count** - положительное целое число, означающее максимальное число символов, записываемых в str **с учётом символа конца строки** \0.
3. **delimiter** - символ типа **char**, на котором прекращается считывание знаков в строку str.

Работа с текстом. С-строки. Ввод с консоли

```
1  const size_t TEXT_SZ = 20;
2  ...
3  char phrase[TEXT_SZ];
4
5  // Вот другое дело
6  cin.getline(phrase, TEXT_SZ);
7
8  char word[TEXT_SZ];
9  // Считываем все символы до первого пробела
10 cin.getline(word, TEXT_SZ, ' ');
```

Работа с текстом. Стандартная библиотека

Библиотека языка C для работы со строками:

```
1 #include <cstring>
```

Получение длины строки **в байтах**

```
size_t strlen(str)
```

```
1  char text[] = "A string";
2  // На экране покажется число 8
3  cout << "Длина строки: " << strlen(text) << "\n";
4
5  char text2[] = "Строка на русском";
6  /* А здесь — зависит от ОС и её локальных
7     настроек. В русскоязычных Windows 7, 8,
8     10 — скорее всего покажет длину равную 17.
9     В современных ОС, основанных на Linux,
10    наиболее вероятное значение — 32 */
11 cout << "Длина строки: " << strlen(text2)
12       << "\n";
```

Посимвольное сравнение строк

```
int strcmp(first, second)
```

Функция возращает:

1. значение $<$ **0** - первый несовпадающий символ в строке **first** меньше, чем в строке **second**;
2. значение $==$ **0** - все символы в обоих строках совпадают;
3. значение $>$ **0** - первый несовпадающий символ в строке **first** больше, чем в строке **second**;

Сравнение идёт с начала строк и до тех пор, пока в одной из них не встретится символ окончания строки. Фактически сравниваются целочисленные коды каждого символа (байта), а не буквы в лексикографическом смысле.

Сравнение строк

```
1 char key_color[] = "зелёный", answer[50];
2
3 do {
4   cout << "\nУгадайте цвет: ";
5   cin.getline(answer, 50, ' ');
6 } while ( strcmp(key_color, answer) != 0 );
7
8 cout << "Правильный ответ!\n";
```

Возможный вывод программы:

```
Угадайте слово: красный
Угадайте цвет: коричневый
Угадайте цвет: зелёный
Правильный ответ!
```

Копирование строки

`strcpy(destination, source)`

- Функция копирует строку **source** в строку **destination**
- Копируются **все** символы из строки, включая и завершающий *символ окончания строки*
- Следить за размером строк следует самостоятельно!

```
1 char source[] = "We make world better!",
2      dest[40];
3 strcpy(dest, source);
4
5 cout << dest << "\n";
```

Работа с текстом. C-строки

Что вносит сложность?

- Нет операторов сравнения
- Необходимо следить за размером при любом добавлении в существующую строку
- Поиск символов и/или текстовых фрагментов идёт реализован через специальный тип данных - **указатели**

Как упростить работу с текстом?

Воспользоваться стандартной библиотекой для строк языка C++. Но про функцию **strlen** не забываем

Работа с текстом. C++ строки

В стандартной библиотеке языка C++ реализован специальный тип данных для работы со строками. Подключается он так:

```
1 #include <string>
```

Какие преимущества даёт по сравнению с прямым использованием массивов типа **char**?

- Автоматическое выделение место под строку при её создании и изменении.
- Использование привычных операторов сравнения: $>$, $<$, $==$, $!=$ и другие
- Использование оператора $+$ для объединения строк

Работа с текстом. C++ строки

Определение переменной типа **string**.

```
1  // Переменной типа string
2  // может быть присвоена С–строкой
3  string s1 = "Строка 1", s2;
4  s2 = "Строка 2";
5  cout << s1 << "\n" << s2 << "\n";
6
7  string s3 = "English-based string";
8  cout << s3[0] << s3[2] << s3[4] << "\n";
9
10 cout << "Строка s3 посимвольно:\n"
11 for (char sym : s3) { cout << s3 << ' '; }
12 cout << "\n";
13
14 string s4 = s1 + ";" + s2;
15 cout << s4 << "\n";
```

Работа с текстом. C++ строки

Ввод строки с терминала

```
getline(cin, string& str_to_fill,
        char delimiter = '\n');
```

- Функция считывает все введённые символы в строку, которую указали в аргументе **str_to_fill**
- Первым параметром идёт переменная, которая отвечает за ввод информации из какого-нибудь источника. Для консольного ввода это всегда **cin**
- Возможно задать разделитель **delimiter** - символ, **после которого** текст считан не будет. Сам разделитель в строку **str_to_fill** не помещается и не учавствует в дальнейших операциях ввода.

Ввод строки с терминала

```
1  string s1, word, s2;
2
3  // Можно использовать и оператор ввода
4  // считываются все символы до первого ↩
       пробела
5  cin >> word;
6
7  getline(cin, s1);
8  getline(cin, s2, '*');
9
10 cout << "Введённое слово" << word;
11 cout << "\nПервая строка:\n" << s1;
12 cout << "\nВторая строка:\n" << s2;
```

Работа с текстом. C++ строки

Получение длины (числа байт) строки.

```
size_t str.size()
size_t str.length()
```

```
1 string s1 = "France";
2
3 // Выведет 6
4 cout << "Длина s1: " << s1.size() << "\n";
5
6 // Также как и с C-строками, конкретное
7 // значение зависит от ОС
8 string s2 = "Провение технических работ";
9 cout << s2.length() << "\n";
```

Работа с текстом. C++ строки

Сравнение строк: для полного сравнения строк используются привычные операторы сравнения: $>$, $>=$, $<$, $<=$, $==$, $!=$

```
1  string s1 = "France",
2         s2 = "Russia";
3
4  if ( s1 < s2 ) {
5    cout << "Кто бы сомневался\n";
6  }
7
8  // Проверка на равенство
9  bool is_equals = s1 == s2;
10 // is_equals равен false
```

Работа с текстом. C++ строки

Сравнение строк: как операторы сравнения $>$, $>=$, $<$, $<=$, $==$, $!=$ работают.

Строка s1 считается **меньше**, чем s2 в двух случаях (s1 $<$ s2)

1. первый несовпадающий символ в s1 имеет **целочисленный код** меньше, чем таковой в s2
2. все символы из s1 совпадают с начальными символами в s2, но вторая строка имеет **большую длину**

Строка s1 считается **больше**, чем s2 в двух случаях (s1 $>$ s2)

1. первый несовпадающий символ в s1 имеет **целочисленный код** больше, чем таковой в s2
2. все символы из s1 совпадают с начальными символами в s2, но вторая строка имеет **меньшую длину**

Иначе - строки считаются **равными**.

Работа с текстом. C++ строки

Частичное сравнение строк

```
(1) int str.compare(other_str)
(2) int str.compare(size_t pos, size_t len,
                    other_str)
(3) int str.compare(size_t pos, size_t len,
      other_str, size_t o_pos, size_t o_len)
```

, где **str** - переменная типа **string**, которую сравниваем. А **other_str** - с которой сравниваем.

1. полное сравнение строк **str** и **other_str**
2. сравнение фрагмента внутри **str**, начиная с символа на позиции **pos** и длиной **len**, со строкой **other_str**
3. сравнение фрагментов из **str** и **other_str**. Позиция и длина для второго задаются аргументами **o_pos** и **o_len**

Метод **compare** возвращает **нуль**, если s1 == s2; **число больше нуля**, если s1 > s2; **число меньше нуля**, если s1 < s2.

Работа с текстом. C++ строки

Частичное сравнение строк: пользуемся следующим методом

```
1  string s1 = "два отличия найдите",
2         s2 = "найдите кота";
3  size_t nlen = strlen("найдите");
4
5  if (s1.compare(s2) < 0) {
6    cout << "Вторая строка больше первой\n";
7  }
8
9  size_t end = string::npos,
10       pos1 = s1.size() - nlen,
11       pos2 = 0;
12
13 if (s1.compare(pos1, end, s2, pos2, nlen) == 0) {
14   cout << "слово 'найдите' есть в "
15           "обеих строках\n";
16 }
```

string::npos - специальное значение, обозначающая конец строки

Работа с текстом. C++ строки

Частичное сравнение со строками языка C

```
(4) int str.compare(c_str)
(5) int str.compare(size_t pos, size_t len,
                    c_str)
(6) int str.compare(size_t pos, size_t len,
                    c_str, size_t c_len)
```

, где **str** - переменная типа **string**, которую сравниваем. А **c_str** - C-строка (символы в двойных кавычках или соответствующий массив типа **char**).

4. полное сравнение строк **str** и **c_str**
5. сравнение фрагмента внутри **str**, начиная с символа на позиции **pos** и длиной **len**, со строкой **c_str**
6. сравнение фрагментов из **str** и **c_str**. Для C-строки можно указать только длину фрагмента для сравнения через аргумент **c_len**

Частичное сравнение со строками языка C

```
1  string s1 = "два отличия найдите",
2  size_t nlen = strlen("найдите");
3
4  size_t end = string::npos,
5         pos1 = s1.size() - nlen;
6
7  char c_str[] = "найдите что-нибудь";
8
9  if (s1.compare(pos1, end, c_str, nlen) == 0) {
10   cout << "Строки равны только по "
11           "слову 'найдите' на "
12           "соответствующих позициях";
13 }
```

Удаление всего содержимого строки из переменной

```
void str.clear()
```

```
1 std::string s1 = "France";
2 // Покажет длину в 6 байт
3 std::cout << "Длина s1: " << s1.size();
4
5 s1.clear();
6 // Покажет длину равную 0
7 std::cout << "Длина s1: " << s1.size();
```

Добавление текста к строке

```
1  string s1 = "Быть",
2         s2 = " или не быть",
3         s3;
4
5  s3 = s1 + s2;
6  s3 += '?';
7  s3 += " Вот в чём вопрос!";
8
9  cout << "Получилась строка:\n" << s3
10      << "\n";
```

Работа с текстом. C++ строки

Вставка на указанную позицию

```
(1) str.insert(size_t pos, other_str)
(2) str.insert(size_t pos, other_str,
                size_t o_pos, size_t o_len)
(3) str.insert(size_t pos, c_str)
(4) str.insert(size_t pos, c_str, size_t c_len)
(5) str.insert(size_t pos, size_t count, char sym)
```

1. Вставляет строку **other_str** в **str** сразу **перед** номером символа, заданного аргументом **pos**
2. Вставляет фрагмент из **other_str**, длиной **o_len** и начиная с символа **o_pos** в **str**
3. Вставляет C-строку **c_str** в **str** перед символом за номером **pos**.
4. Вставляет фрагмент C-строки **c_str**, длиною **c_len**, в **str** перед символом за номером **pos**.
5. Вставляет в строку **str** символ **sym** в количестве **count** штук перед символом за номером **pos**.

Вставка на указанную позицию

```
1 string s1 = "Что дела?";
2 size_t w_len = strlen("Что");
3
4 s1.insert(w_len + 1, "за ");
5
6 // Напечатает "Что за дела?"
7 cout << s1 << "\n";
```

Работа с текстом. C++ строки

Преобразование в C-строку

```
char* str.c_str()
```

Метод вернёт массив (точнее - указатель на массив), который является корректной C-строкой.

```
1 string s1 = "Странное сообщение";
2 char c_str[] = "и не говори";
3
4 if ( strcmp(c_str, s1.c_str()) == 0 ) {
5   cout << "Такого не может быть\n";
6 }
```

Работа с текстом. C++ строки

Выделение подстроки

```
string str.substr(size_t start,
                  size_t len = string::npos)
```

start - переменная типа **size_t**, указывающая позицию первого символа подстроки. **len** - количество символов для извлечения.

```
1  string s1 = "Phase transitions are"
2              " great part of physics";
3  size_t part1 = strlen("Phase transitions are "),
4         part2 = strlen("great");
5
6  string s2 = s1.substr(part1, part2);
7  // Печатаем: "great"
8  cout << s2 << "\n";
9
10 string s3 = s1.substr(part1 + part2 + 1);
11 // Печатаем: "part of physics"
12 cout << s3 << "\n";
```

Работа с текстом. C++ строки

Поиск в строке

```
size_t str.find(other, size_t pos = 0)
size_t str.rfind(other_str,
                 size_t rpos = npos)
```

- Возращает позицию первого символа аргумента **other** в строке **str**, если **other** присутствует в **str** в качестве текстового фрагмента.
- **other** может быть переменной типа **string**, C-строкой, и переменной типа **char**.
- Поиск начинается с позиции, определяемой вторым аргументом.
- **rfind** - поиск с конца строки (просмотр символов идёт справа налево).

Поиск в строке

```
1  string s1 = "Сопротивление обратно ↩
       пропорционально силе тока";
2  size_t found_pos = s1.find("обр");
3
4  if ( found_pos != std::string::npos ) {
5    cout << "Позиция \"обр\": " << found_pos;
6  }
7
8  found_pos = s1.find("ТЧК", 6);
9  if ( found_pos == string::npos ) {
10   cout << "\n\"ТЧК\" в исходной строке не ↩
         обнаружена";
11 }
```

Работа с текстом. C++ строки

Поиск всех вхождений символа в строку

```
1  string text = "Да, были люди в наше время,\n"
2                "Не то, что нынешнее племя:\n"
3                "Богатыри — не вы!\n"
4                "Плохая им досталась доля:\n"
5                "Немногие вернулись с поля...\n"
6                "Не будь на то господня воля,\n"
7                "Не отдали б Москвы!\n";
8  cout << "Ищем все запятые в тексте\n" << text;
9  cout << "\n----------------------------\n";
10
11 size_t comma_pos = text.find(',');
12 while ( comma_pos != string::npos ) {
13   size_t hplc = comma_pos + 1;
14   cout << "',' найдена на " << hplc <<" месте\n";
15   // Поиск продолжается с первого символа,
16   // идущего после ','
17   comma_pos = text.find(',', hplc);
18 }
```

Работа с текстом. C++ строки

Замена части текста в строке

```
(1) str.replace(size_t pos, size_t len, other_str)
(2) str.replace(size_t pos, size_t len, other_str,
                size_t o_pos, size_t o_len)
(3) str.replace(size_t pos, size_t len, c_str)
(4) str.replace(size_t pos, size_t len, c_str,
                size_t c_len)
(5) str.replace(size_t pos, size_t len,
                size_t count, char sym)
```

1. В строке **str** символы в количестве **len** штук (сколько символов из исходной строки удаляем), начиная с номера **pos**, заменяются на строку **other_str**
2. Аналогично, но замена происходит на фрагмент из **other_str**, длиной **o_len** и начиная с символа **o_pos**
3. Замена нужного количества символов на C-строку **c_str**
4. Замена нужного количества символов на **o_len** символов из C-строки **c_str**
5. Замена происходит на символ **sym** в количестве **count**

Замена части текста в строке

```
1 string str = "сегодня будет абракадабра!";
2 size_t w_len = strlen("абракадабра");
3
4 str.replace(str.size() - w_len - 1,
5               w_len, "всё хорошо");
6 cout << "И у нас " << str;
```

Работа с текстом. C++ строки

Преобразование строк в числа с помощью библиотеки <**cstdlib**>

```
(1) int       atoi( c_str )
(2) double    atof( c_str )
(3) long      atol( c_str )
(4) long long atoll( c_str )
```

Данные четыре функции пытаются преобразовать переданную им C-строку в соответствующее числовое значение. Если преобразование не удалось, то возвращаемый результат не определён (UB).

Работа с текстом. C++ строки

Преобразование строк в числа с помощью функций из **<stdlib.h>**

```
1  char sf[] = "456.7788",
2       si[] = "-7485",
3       sl[] = "313377317135";
4
5  double d_num = atof(  sf.c_str() );
6  int    i_num = atoi(  si.c_str() );
7  size_t l_num = atoll( sl.c_str() );
8
9  cout << d_num << " " << i_num
10      << " " << l_num << "\n";
```

Работа с текстом. C++ строки

Передача переменных типа **string** в функции

Передача таких переменных должны происходить по **ссылке**. В случае передачи по значению, каждый аргумент типа **string** будет вызывать копирование всего текста, который содержит передаваемая переменная.

```
1  void process_str1(string, string );
2  void process_str2(string&, string&);
3  void process_str3(const string&, const string& );
4
5  string s_one = "строка", s_two = "опять строка";
6  // Здесь всегда происходит копирование
7  process_str1(s_one, s_two);
8  // Передача по ссылке => нет копирования
9  // Но строки могут внутри функции изменяться
10 process_str2(s_one, s_two);
11 // Передача по неизменяемой ссылке
12 process_str3(s_one, s_two);
```