Лекция 7: Строки. Стандартные процедуры и функции для работы со строками в C++

Введение

В языке программирования C++ строки — это последовательности символов, используемые для хранения и обработки текстовой информации. В C++ для работы со строками применяются два подхода: классические C-строки (массивы символов) и строки из стандартной библиотеки `std::string`. В этой лекции мы рассмотрим как базовые методы работы с классическими C-строками, так и более современные функции для работы с `std::string`.

1. Классические C-строки

Классические строки в C++ представляют собой массивы символов, оканчивающиеся нулевым символом (`\0`). В этом случае необходимо использовать функции из библиотеки `cstring` для работы с такими строками.

Объявление и инициализация C-строк

Пример объявления и инициализации строки в стиле C:

```
char str[100] = "Привет, мир!";
```

Основные функции работы с C-строками из `cstring`

1. `strlen()` — вычисляет длину строки (без учета нулевого символа).

   ```
   #include <cstring>
   size_t length = strlen(str);  // Вернет 12
   ```

2. `strcpy()` — копирует одну строку в другую.

   ```
   char dest[100];
   strcpy(dest, str);  // Копирует содержимое str в dest
   ```

3. `strcat()` — объединяет две строки (добавляет одну строку к другой).

   ```
   char additional[] = " Как дела?";
   strcat(str, additional);  // str станет "Привет, мир! Как дела?"
   ```

4. `strcmp()` — сравнивает две строки.

```
int result = strcmp(str, "Привет, мир!");  // Возвращает 0, если строки
равны
```

5. `strchr()` — ищет первое вхождение символа в строке.

```
char* ptr = strchr(str, 'м');  // Возвращает указатель на 'м' в строке
```

6. `strstr()` — ищет подстроку в строке.

```
char* subptr = strstr(str, "мир");  // Возвращает указатель на начало "мир"
```

Пример работы с C-строками

```
#include <iostream>
#include <cstring>

int main() {
    char str[100] = "Привет";
    char add[50] = ", мир!";

    strcat(str, add);  // Объединяем строки
    std::cout << "Итоговая строка: " << str << std::endl;

    return 0;
}
```

2. Строки из стандартной библиотеки `std::string`

В языке C++ класс `std::string` предоставляет более удобные и мощные средства для работы со строками по сравнению с C-строками. Для использования этого класса необходимо подключить заголовочный файл `<string>`.

Объявление и инициализация

```
#include <string>

std::string str = "Привет, мир!";
```

Основные методы и функции для работы со строками

1. `length()` **или** `size()` — возвращает длину строки.

```
size_t length = str.length();  // Возвращает 12
```

2. `empty()` — проверяет, является ли строка пустой.

```
bool isEmpty = str.empty();  // Возвращает false
```

3. `append()` **или оператор** `+=` — добавляет одну строку к другой.

```
str += " Как дела?";  // str станет "Привет, мир! Как дела?"
```

4. `substr()` — возвращает подстроку строки.

```
std::string sub = str.substr(0, 6);  // Вернет "Привет"
```

5. `find()` — ищет подстроку в строке и возвращает индекс первого вхождения.

```
size_t pos = str.find("мир");  // Вернет 8
```

6. `replace()` — заменяет часть строки на другую строку.

```
str.replace(8, 3, "вселенная");  // str станет "Привет, вселенная!"
```

7. `compare()` — сравнивает две строки.

```
int result = str.compare("Привет, вселенная!");  // Вернет 0, если строки
равны
```

8. **Оператор доступа** `[]` — доступ к отдельным символам строки.

```
char ch = str[0];  // Вернет 'П'
```

Пример работы со строками `std::string`

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

```
int main() {
    std::string str = "Привет";
    std::string add = ", мир!";

    str += add;  // Объединяем строки
    std::cout << "Итоговая строка: " << str << std::endl;

    std::string sub = str.substr(8, 3);  // Извлекаем подстроку
    std::cout << "Подстрока: " << sub << std::endl;

    return 0;
}
```

3. Преимущества использования `std::string`

- **Безопасность**: в отличие от C-строк, строки `std::string` автоматически управляют памятью.
- **Удобство**: встроенные методы `std::string` позволяют легко выполнять сложные операции со строками, такие как поиск, сравнение и изменение.
- **Расширяемость**: объект `std::string` может динамически расширяться при добавлении новых символов.

4. Преобразование между C-строками и `std::string`

Иногда может понадобиться преобразование между C-строками и объектами `std::string`.

- Преобразование из C-строки в `std::string`:

  ```
  std::string str = "Привет";
  ```

- Преобразование из `std::string` в C-строку:

  ```
  const char* c_str = str.c_str();
  ```

Заключение

В C++ строки являются важным и мощным инструментом для работы с текстовыми данными. Мы можем использовать как классические C-строки, так и более безопасные и функциональные строки `std::string`. Выбор подхода зависит от конкретных задач, но в большинстве случаев рекомендуется использовать `std::string`, так как этот класс предлагает более удобные и безопасные механизмы для работы со строками.