Московский государственный технический университет им. Н.Э. Баумана

Строки

Панилов Павел Алексеевич
2025

Работа со строками в C++

1. Введение в строки как массивы символов
2. Создание и инициализация строк
3. Операции со строками
4. Функции для работы со строками
5. Манипуляции с отдельными символами
6. Работа с подстроками
7. Форматирование строк
8. Дополнительные операции со строками
9. Примеры кода
10. Регулярные выражения в C++
11. Многомерные массивы символов

Зачем нужно работать со строками?

Работа со строками является неотъемлемой частью многих программ и приложений.

Вот некоторые типичные сценарии, когда работа со строками становится важной:

- **Обработка текстовых данных:** Строки используются для чтения, записи и обработки текстовых данных из файлов, баз данных и пользовательского ввода.
- **Манипуляции с текстом:** Строки позволяют выполнять различные операции над текстом, такие как поиск, замена, разделение, объединение и многое другое.
- **Интерфейсы пользователя:** Строки играют важную роль в создании пользовательских интерфейсов, включая отображение текстовых сообщений, меток и ввода текста.
- **Анализ и обработка данных:** В текстовых данных можно найти информацию, которую нужно извлечь и обработать, например, при анализе лог-файлов или веб-страниц.

Введение в строки как массивы символов

Что такое строка в C++?

В языке программирования C++, строка представляет собой последовательность символов.

Эти символы могут быть любыми буквами, цифрами или специальными знаками, которые представлены в таблице ASCII (или Unicode в случае многих современных языков программирования).

Строка в C++ фактически является массивом символов, где каждый символ имеет свой индекс (позицию) в массиве.

Введение в строки как массивы символов

Символы и ASCII-коды

В компьютерном мире каждый символ представлен числовым значением, называемым ASCII-кодом.

ASCII (American Standard Code for Information Interchange) - это стандартный набор символов (буквы, цифры, пунктуация и специальные символы), каждому из которых соответствует уникальный числовой код.

Например, буква "A" имеет ASCII-код 65, а буква "a" - 97. Этот стандарт обеспечивает согласованный способ представления текстовой информации в компьютерах.

Когда вы работаете с массивами символов (строками) в C++, каждый символ в строке на самом деле является числовым значением, которое можно интерпретировать как символ согласно **таблице ASCII.**

Введение в строки как массивы символов

Таблица ASCII

	0	1	2	3	4	5	6	7	8	9
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht
1	nl	vt	np	cr	so	si	dle	dc1	dc2	dc3
2	dc4	nak	syn	etb	can	em	sub	esc	fs	gs
3	rs	us	sp	!	“	#	$	%	&	‘
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	‘	a	b	C
10	d	e	f	g	h	i	j	k	l	M
11	n	o	p	q	r	s	t	u	v	W
12	x	y	z	{	|	}	~	del		

Введение в строки как массивы символов

Пример

```
char myChar = 'A'; // Символ 'A'
int asciiCode = static_cast<int>(myChar); // Преобразование символа в ASCII-код
(65)
std::cout << "ASCII-код символа 'A': " << asciiCode << std::endl;
```

В этом примере символ 'A' представлен числовым значением 65 в соответствии с таблицей ASCII.

Использование массивов символов в C++ позволяет программистам манипулировать текстовой информацией, выполнять операции с символами и создавать сложные текстовые структуры, которые широко используются в различных типах программ.

Создание и инициализация строк

Создание символьного массива (строки)

В C++, строки часто представлены как массивы символов типа char.
Для создания строки в виде символьного массива с заданной длиной используется следующий синтаксис:

```
char myString[50]; // Создание массива символов длиной 50
```

В этом примере myString - это массив символов (строка), который может содержать от 0 до 49 символов.
Первый символ в массиве имеет индекс 0, второй - индекс 1, и так далее.
Последний символ будет иметь индекс 49.

Создание и инициализация строк

Инициализация символьного массива

Созданный массив символов можно инициализировать значениями сразу после его определения. Инициализация может быть выполнена явно с использованием литералов или результатами других выражений.

Пример 1: Инициализация литералом

```
char greeting[] = "Hello"; // Инициализация массива символов литералом "Hello"
```

В этом примере greeting - это массив символов, который автоматически получает размер 6 (5 символов для "Hello" + символ нуля '\0' для обозначения конца строки).

В массиве хранится: **'H', 'e', 'l', 'l', 'o', '\0'.**

Создание и инициализация строк

Пример 2: Инициализация с помощью явных символов

```
char name[] = {'J', 'o', 'h', 'n', '\0'}; // Инициализация символами
```

Здесь name - это массив символов, представляющий строку "John".

Важно заметить, что мы вручную добавляем символ '\0' в конце массива.

Использование символа нуля ('\0') важно, так как многие функции и операции со строками в C++ полагаются на его наличие для определения конца строки.

Убедитесь, что каждая строка завершается символом '\0' для корректной работы с ней в C++.

Операции со строками

Чтение строк

Для чтения строк с клавиатуры в C++ вы можете использовать функцию **cin.getline()**.
Эта функция позволяет вам вводить строку с клавиатуры и сохранять ее в определенный массив символов.

```
char myString[50];
cin.getline(myString, 50);
```

В этом примере myString - это массив символов, который имеет длину 50.

Функция **cin.getline()** считывает символы с клавиатуры до тех пор, пока не встретит символ новой строки ('\n') или пока не будет считано максимальное количество символов (в данном случае, 50 символов).

Полученная строка будет храниться в массиве myString.

Операции со строками

Вывод строк

Для вывода строк на экран в C++ используется оператор <<.

Если myString - это массив символов (или строка), вы можете вывести его содержимое следующим образом:

```
cout << myString;
```

В этом случае cout - это стандартный поток вывода (обычно консоль), а myString - это массив символов (или строка), который вы хотите вывести.

Команда **cout << myString** отправляет содержимое массива символов в стандартный вывод, который обычно появляется в командной строке или консоли при запуске программы.

Операции со строками

Пример использования операций чтения и вывода строк

```
#include <iostream>

int main() {
    char myString[50];

    // Чтение строки с клавиатуры
    std::cout << "Введите строку: ";
    std::cin.getline(myString, 50);

    // Вывод строки на экран
    std::cout << "Вы ввели: " << myString << std::endl;

    return 0;
}
```

В этом примере программа запрашивает ввод строки с клавиатуры, сохраняет ее в массиве myString с использованием cin.getline(), а затем выводит введенную строку на экран с использованием cout << myString.

Операции со строками

Конкатенация строк

Конкатенация строк означает объединение двух или более строк в одну.

В C++, для конкатенации строк вы можете использовать функцию **strcat()**, которая находится в заголовочном файле **<cstring>**.

Эта функция принимает два аргумента:

- строку-назначение (destination)
- строку-источник (source).

Она добавляет содержимое строки-источника в конец строки-назначения.

Операции со строками

Пример использования strcat()

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

int main() {
    char destination[50] = "Hello, ";
    char source[] = "world!";

    strcat(destination, source); // Конкатенация строк

    std::cout << "Конкатенированная строка: " << destination << std::endl;

    return 0;
}
```

В этом примере строка "world!" добавляется в конец строки "Hello, ", и результат ("Hello, world!") сохраняется в destination.

Операции со строками

Доступ к символам строки

Когда вы создаете строку как массив символов в C++, каждый символ имеет свой индекс (позицию) в массиве.

Для доступа к символам строки используются индексы.

Индексация начинается с 0 для первого символа в строке.

Операции со строками

Пример использования индексов для доступа к символам

```
#include <iostream>

int main() {
    char myString[] = "Hello";

    // Доступ к символам строки с использованием индексов
    std::cout << "Первый символ: " << myString[0] << std::endl; // Выводит 'H'
    std::cout << "Второй символ: " << myString[1] << std::endl; // Выводит 'e'
    std::cout << "Третий символ: " << myString[2] << std::endl; // Выводит 'l'

    return 0;
}
```

Обратите внимание, что индексы должны быть в пределах длины строки минус один, чтобы избежать обращения к недопустимым памяти и ошибкам выполнения программы.

В этом примере myString[0] обращается к первому символу строки "Hello", который является буквой 'H'. Аналогично, myString[1] обращается ко второму символу 'e', и myString[2] обращается к третьему символу 'l'.

Функции для работы со строками

Длина строки

В C++, для определения длины строки (количества символов в строке) можно использовать функцию **strlen()**. Она находится в заголовочном файле <cstring>.

Пример использования strlen()

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

int main() {
    char myString[] = "Hello, World!";

    // Получение длины строки с использованием strlen()
    int length = strlen(myString);

    std::cout << "Длина строки: " << length << std::endl; // Выводит 13

    return 0;
}
```

Функция strlen() возвращает количество символов в строке до символа нуля ('\0'),

Обратите внимание, что функция strlen() не считает символ нуля в конце строки при подсчете длины.

Строка "Hello, World!" содержит 13 символов, и strlen(myString) возвращает это число.

Функции для работы со строками

Копирование строк

В C++, для копирования строк вы можете использовать функцию **strcpy()**, которая находится в заголовочном файле <cstring>. Эта функция копирует содержимое одной строки в другую.

Пример использования strcpy()

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

int main() {
    char source[] = "Hello, World!";
    char destination[50]; // Создаем пустую строку для копирования
        // Копирование строки с использованием strcpy()
    strcpy(destination, source);

    std::cout << "Скопированная строка: " << destination << std::endl; //
Выводит "Hello, World!"
        return 0;
}
```

В этом примере strcpy(destination, source) копирует содержимое строки source в строку destination.

Обратите внимание, что destination должен иметь достаточный размер для хранения содержимого source, включая символ нуля ('\0').
Если destination не имеет достаточного размера, это может привести к переполнению буфера, что может привести к непредсказуемому поведению программы.

strcpy() не выполняет проверку на переполнение.

Функции для работы со строками

Сравнение строк в C++

В C++, для сравнения строк используется функция strcmp(), которая находится в заголовочном файле <cstring>.

Эта функция сравнивает две строки и возвращает целое число, которое указывает на результат сравнения:

- 0, если строки равны
- Отрицательное число, если первая строка меньше второй (лексикографически)
- Положительное число, если первая строка больше второй (лексикографически)

Функции для работы со строками

Пример использования strcmp()

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

int main() {
    char str1[] = "apple";
    char str2[] = "banana";

    int result = strcmp(str1, str2);

    if (result == 0) {
        std::cout << "Строки равны." << std::endl;
    } else if (result < 0) {
        std::cout << "Первая строка меньше второй." << std::endl;
    } else {
        std::cout << "Первая строка больше второй." << std::endl;
    }

    return 0;
}
```

В этом примере strcmp(str1, str2) сравнивает строки "apple" и "banana".

Поскольку "apple" лексикографически меньше, результат будет отрицательным.

Функции для работы со строками

Важные замечания

- **strcmp()** сравнивает строки до символа нуля ('\0').

Это значит, что он сравнивает строки до их конечных символов.

- **strcmp()** сравнивает строки с учетом регистра

Это означает, что при сравнении двух символов ASCII заглавные буквы будут рассматриваться как меньшие, чем строчные буквы.
Например, буква 'A' (ASCII код 65) будет меньше по значению, чем 'a' (ASCII код 97), и поэтому 'A' считается меньшим символом, чем 'a' в результате сравнения строк.

Функции для работы со строками

Пример

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

int main() {
    char str1[] = "Apple";
    char str2[] = "apple";

    int result = strcmp(str1, str2);

    if (result == 0) {
        std::cout << "Строки равны." << std::endl;
    } else if (result < 0) {
        std::cout << "Строка 'Apple' меньше строки 'apple'." << std::endl;
    } else {
        std::cout << "Строка 'Apple' больше строки 'apple'." << std::endl;
    }

    return 0;
}
```

Поскольку функция учитывает регистр, 'A' (в "Apple") меньше по значению, чем 'a' (в "apple"), и поэтому строка "Apple" считается меньшей, чем строка "apple". Результат сравнения будет отрицательным.

Манипуляции с отдельными символами в C++

Преобразование регистра символов

В C++ вы можете использовать функции **toupper()** и **tolower()** для преобразования отдельных символов в верхний и нижний регистр соответственно.

Обе функции находятся в заголовочном файле **<cctype>**.

Эти функции особенно полезны, когда вам нужно сравнивать символы без учета регистра или преобразовывать символы в одинаковый регистр для обработки в программе.

Манипуляции с отдельными символами в C++

toupper()

Эта функция используется для преобразования символа в верхний регистр.
Если символ уже в верхнем регистре или не является буквой, он остается неизменным.

Пример использования toupper()

```
#include <iostream>
#include <cctype>

int main() {
    char lowercase = 'a';
    char uppercase = toupper(lowercase);

    std::cout << "Символ в верхнем регистре: " << uppercase
<< std::endl; // Выводит 'A'

    return 0;
}
```

функция toupper() преобразует символ 'a' в 'A'.

Манипуляции с отдельными символами в C++

tolower()

Эта функция используется для преобразования символа в нижний регистр.
Если символ уже в нижнем регистре или не является буквой, он остается неизменным.

Пример использования tolower():

```
#include <iostream>
#include <cctype>

int main() {
    char uppercase = 'B';
    char lowercase = tolower(uppercase);

    std::cout << "Символ в нижнем регистре: " << lowercase <<
std::endl; // Выводит 'b'

    return 0;
}
```

функция tolower() преобразует символ 'B' в 'b'.

Манипуляции с отдельными символами в C++

Поиск символа в строке в C++

В C++, для поиска первого вхождения символа в строке вы можете использовать функцию strchr(), которая находится в заголовочном файле <cstring>.

strchr() ищет указанный символ в строке и возвращает указатель на первое вхождение этого символа.

Важно отметить, что strchr() находит только первое вхождение символа в строке.

Манипуляции с отдельными символами в C++

Использование strchr()

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

int main() {
    char myString[] = "Hello, World!";

    char* result = strchr(myString, 'o'); // Поиск символа 'o' в строке

    if (result != nullptr) {
        std::cout << "Символ 'o' найден в позиции: " << (result - myString) << std::endl;
    } else {
        std::cout << "Символ 'o' не найден в строке." << std::endl;
    }

    return 0;
}
```

(result - myString) используется для вычисления позиции символа 'o' в строке myString. Таким образом, мы можем узнать, в какой позиции находится найденный символ.

strchr(myString, 'o') ищет символ 'o' в строке "Hello, World!". Если символ найден, функция вернет указатель на его первое вхождение. Если символ не найден, она вернет nullptr.

Манипуляции с отдельными символами в C++

Использование циклов для поиска всех вхождений

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

int main() {
    char myString[] = "Hello, World!";
    char target = 'o';

    std::cout << "Все позиции символа 'o' в строке: ";

    for (int i = 0; i < strlen(myString); ++i) {
        if (myString[i] == target) {
            std::cout << i << " ";
        }
    }

    std::cout << std::endl;

    return 0;
}
```

Цикл проходит через каждый символ в строке myString и сравнивает его с целевым символом 'o'. Если символы совпадают, программа выводит индекс символа в строке.

Манипуляции с отдельными символами в C++

Работа с подстроками в C++

В C++, для выделения подстроки определенной длины из строки вы можете использовать функцию **strncpy()**, которая находится в заголовочном файле <cstring>.

strncpy() копирует указанное количество символов из одной строки в другую.

Манипуляции с отдельными символами в C++

Использование strncpy()

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

int main() {
    char source[] = "Hello, World!";
    char destination[20]; // Создаем массив для подстроки

    int length = 5; // Длина подстроки, которую мы хотим выделить

    strncpy(destination, source, length); // Копируем подстроку

    // Добавляем завершающий символ нуля, чтобы строка была корректно завершена
    destination[length] = '\0';

    std::cout << "Выделенная подстрока: " << destination << std::endl; // Выводит "Hello"

    return 0;
}
```

strncpy(destination, source, length) копирует первые 5 символов из строки "Hello, World!" в массив destination.
Затем добавляется символ нуля ('\0'), чтобы корректно завершить строку в массиве destination.

Важные замечания

strncpy() всегда копирует фиксированное количество символов (length), даже если исходная строка короче. Поэтому в destination могут остаться неинициализированные символы, если length больше, чем длина исходной строки.

Если length больше или равно длине source, strncpy() не добавляет символ нуля в конец destination. Поэтому необходимо добавить его вручную.

Манипуляции с отдельными символами в C++

Использование std::string

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

int main() {
    std::string source = "Hello, World!";
    int length = 5;

    std::string substring = source.substr(0, length);

    std::cout << "Выделенная подстрока: " << substring << std::endl; // Выводит "Hello"

    return 0;
}
```

В современном C++ предпочтительнее использовать объекты класса std::string, которые предоставляют удобные методы для работы с подстроками и автоматически управляют памятью.

std::string::substr() используется для выделения подстроки. Этот метод предоставляет более безопасный и удобный способ работы с подстроками в C++.

Манипуляции с отдельными символами в C++

Поиск подстроки в строке с использованием strstr()

В C++, функция strstr() используется для поиска первого вхождения подстроки в строке.
Эта функция находится в заголовочном файле <cstring>.

Использование strstr():

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

int main() {
    const char* haystack = "Hello, World!";
    const char* needle = "World";

    const char* result = strstr(haystack, needle); // Поиск подстроки
        if (result != nullptr) {
        std::cout << "Подстрока найдена в позиции: " << result - haystack
<< std::endl;
    } else {
        std::cout << "Подстрока не найдена в строке." << std::endl;
    }
        return 0;
}
```

- haystack представляет строку, в которой вы ищете подстроку.
- needle представляет подстроку, которую вы ищете в haystack.
- strstr(haystack, needle) ищет первое вхождение подстроки needle в строке haystack и возвращает указатель на первое вхождение.

Если подстрока не найдена, функция возвращает nullptr.

Примечания

- strstr() сравнивает символы в строках до символа нуля ('\0')
- Функция чувствительна к регистру, поэтому "World" и "world" будут считаться разными подстроками.

Подстрока "World" найдена в позиции 7 (индексация начинается с 0) в строке "Hello, World!".

Манипуляции с отдельными символами в C++

Поиск подстроки в строке с использованием strstr():

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

int main() {
    const char* haystack = "Hello, World!";
    const char* needle = "World";

    const char* result = strstr(haystack, needle); // Поиск подстроки

    if (result != nullptr) {
        std::cout << "Подстрока найдена в позиции: " << result - haystack
<< std::endl;
    } else {
        std::cout << "Подстрока не найдена в строке." << std::endl;
    }

    return 0;
}
```

- const char* haystack = "Hello, World!"; - это создание указателя haystack, который указывает на первый символ строки "Hello, World!". В данном случае, строка "Hello, World!" представляет собой массив символов, заканчивающийся нулевым символом, и haystack указывает на первый символ этой строки.

- const char* needle = "World"; - это создание указателя needle, который указывает на первый символ строки "World".

- const char* result = strstr(haystack, needle); - функция strstr() ищет первое вхождение строки, указанной в needle, в строке, указанной в haystack. Результат (result) - это указатель на первый символ вхождения needle в haystack.

- result - haystack - это арифметика указателей. Вычитание указателей в C/C++ дает разницу в их адресах. В данном случае, result - haystack дает позицию первого символа вхождения needle в haystack.

Форматирование строк

Форматирование строк с использованием printf():

printf() - это функция в языке программирования C и его производных (включая C++), используемая для форматирования и вывода текстовых данных.
Она является мощным инструментом для создания отформатированных строк, что позволяет вам контролировать, как данные будут выглядеть в выходной строке.

<cstdio> - это заголовочный файл в языках программирования C и C++.
Он предоставляет функции для работы с файлами и ввода/вывода на стандартные потоки (например, консольный ввод и вывод).
В C++, он представлен как <cstdio>.

В <cstdio> определены функции для ввода и вывода данных, такие как printf(), scanf(), fopen(), fclose(), fread(), fwrite(), fprintf(), fscanf(), и многие другие.

Также этот заголовочный файл содержит определения для различных констант, таких как stdin, stdout, stderr, которые представляют стандартные потоки ввода, вывода и ошибок соответственно.

Форматирование строк

Основный синтаксис

```
#include <cstdio>

int main() {
    int number = 42;
    float pi = 3.14159;
    char character = 'A';
    const char* string = "Hello, World!";

    printf("Целое число: %d\n", number);
    printf("Число с плавающей точкой: %f\n", pi);
    printf("Символ: %c\n", character);
    printf("Строка: %s\n", string);

    return 0;
}
```

- %d используется для отображения целых чисел.
- %f используется для отображения чисел с плавающей точкой.
- %c используется для отображения символов.
- %s используется для отображения строк.

Форматирование строк

Дополнительные возможности

Ширина поля: Вы можете указать минимальную ширину поля, которая определяет минимальное количество символов для отображения.
Например, %5d отображает целое число, занимая не менее 5 символов.
Точность: Для чисел с плавающей точкой вы можете указать точность после запятой.
Например, %.2f отобразит число с двумя знаками после запятой.

```
#include <cstdio>

int main() {
    int number = 42;
    float pi = 3.14159;

    printf("Целое число: %10d\n", number);  // Ширина поля 10 символов
    printf("Число с плавающей точкой: %.2f\n", pi);  // 2 знака после запятой

    return 0;
}
```

Форматирование строк

\n - Символ новой строки (перевода строки):

Символ \n является управляющим символом, который обозначает конец текущей строки и переход на новую строку. При выводе текста на консоль или в файл, если встречается символ \n, следующий вывод начнется с новой строки. Это позволяет организовать текст в более удобочитаемом виде, разделяя его на строки.

Пример

```
#include <iostream>

int main() {
    std::cout << "Первая строка\nВторая строка\nТретья строка" << std::endl;
    return 0;
}
```

Вывод

```
Первая строка
Вторая строка
Третья строка
```

Форматирование строк

\t - Символ табуляции (горизонтальной табуляции)

Символ \t представляет собой символ табуляции. При выводе текста он вставляет определенное количество пробелов (называемых табуляцией) для создания горизонтального отступа. Табуляция обычно используется для выравнивания текста и создания столбцов, что делает вывод более организованным и легко читаемым.

Пример

```
#include <iostream>

int main() {
    std::cout << "Колонка 1\tКолонка 2\tКолонка 3" << std::endl;
    std::cout << "Данные 1\tДанные 2\tДанные 3" << std::endl;
    return 0;
}
```

Вывод

```
Колонка 1   Колонка 2   Колонка 3
Данные 1    Данные 2    Данные 3
```

Символы табуляции \t вставляют пробелы между словами, чтобы выровнять текст в столбцах.
В точности, количество пробелов, которые добавляет символ табуляции, может варьироваться в зависимости от настроек консоли или текстового редактора, но обычно это 4 или 8 пробелов.

Форматирование строк

Основные спецификаторы формата в C++

%d, %i - для целых чисел

```
int number = 42;
printf("Целое число: %d\n",
number);
// Вывод: Целое число: 42
```

%f - для чисел с плавающей точкой

```
float floatingNumber = 3.14;
printf("Число с плавающей точкой: %f\n", floatingNumber);
// Вывод: Число с плавающей точкой: 3.140000
```

%.2f - для чисел с плавающей точкой с ограничением знаков после запятой

```
float pi = 3.14159;
printf("Число с плавающей точкой: %.2f\n", pi);
// Вывод: Число с плавающей точкой: 3.14
```

Форматирование строк

Основные спецификаторы формата в C++

%c - для символов

```
char character = 'A';
printf("Символ: %c\n", character);
// Вывод: Символ: A
```

%s - для строк

```
const char* string = "Hello, World!";
printf("Строка: %s\n", string);
// Вывод: Строка: Hello, World!
```

Ширина поля

```
int number = 42;
printf("Целое число: %10d\n",
number);
// Вывод: Целое число:         42
```

Заполнение символами

```
int number = 42;
printf("Целое число: %010d\n",
number);
// Вывод: Целое число:
0000000042
```

Динамические строки и использование динамической памяти в C++

В языке C++, динамические строки позволяют вам создавать строки переменной длины, которые могут изменяться во время выполнения программы.
Вместо использования статического массива символов (как в обычных строках) вы выделяете память динамически, что означает, что размер строки может меняться по мере необходимости.

Выделение динамической памяти для строки

new char[14] выделяет память под 14 символов (включая завершающий нулевой символ) и возвращает указатель на первый символ в этой памяти.

Функция strcpy() копирует строку "Hello, World!" в выделенную память.

```
#include <iostream>
#include <cstring> // Для функции strcpy

int main() {
    // Выделение динамической памяти для строки "Hello, World!"
    char* dynamicString = new char[14]; // 14 - это количество символов в "Hello, World!"

    // Копирование строки в выделенную память
    strcpy(dynamicString, "Hello, World!");

    // Использование строки
    std::cout << dynamicString << std::endl;

    // Освобождение динамической памяти после использования
    delete[] dynamicString;

    return 0;
}
```

Динамические строки и использование динамической памяти в C++

Освобождение динамической памяти:

Когда динамически выделенная память больше не нужна, ее необходимо явно освободить, чтобы избежать утечек памяти. Для этого используется оператор delete[].

```
delete[] dynamicString;
```

Этот оператор сообщает операционной системе, что память, ранее выделенная с помощью new[], больше не используется и теперь может быть использована для других целей.

Динамические строки и использование динамической памяти в C++

Применение std::string вместо динамических массивов:

В C++ также есть стандартный класс std::string, который управляет динамически выделенной памятью за вас, что делает работу со строками более безопасной и удобной.

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

int main() {
    // Использование std::string
    std::string dynamicString = "Hello, World!";

    // Нет необходимости вручную выделять или
освобождать память
    // Использование строки
    std::cout << dynamicString << std::endl;

    return 0;
}
```

std::string автоматически управляет выделением и освобождением памяти, что делает его более предпочтительным выбором для работы со строками в современных программах на C++.

Разделение строки на подстроки

Применение strtok()

В C++ можно использовать функцию strtok() для разделения строки на подстроки на основе определенных разделителей.
strtok() является функцией из стандартной библиотеки C и продолжает поддерживаться в C++.

Пример использования strtok() в C++

```
Токен: Hello
Токен: World
Токен: This
Токен: Is
Токен: C++-Programming
```

Этот код разделяет строку str на подстроки, используя запятые в качестве разделителей.
Первый вызов strtok() возвращает указатель на первый токен. Последующие вызовы с аргументом nullptr возвращают следующие токены.

strtok() изменяет исходную строку, заменяя разделители нулевыми символами. Поэтому строка str будет изменена после использования strtok().

```
#include <iostream>
#include <cstring> // Для функции strtok

int main() {
    char str[] = "Hello,World,This,Is,C++-Programming";
    const char delimiters[] = ","; // Разделители - запятые

    // Первый вызов strtok() с исходной строкой
    char* token = strtok(str, delimiters);

    // Последующие вызовы strtok() возвращают следующие токены
    while (token != nullptr) {
        std::cout << "Токен: " << token << std::endl;
        token = strtok(nullptr, delimiters); // nullptr указывает, что мы используем ту
же строку
    }

    return 0;
}
```

Разделение строки на подстроки

Важные замечания

Изменение исходной строки: Функция strtok() изменяет входную строку, заменяя разделители нулевыми символами. Будьте осторожны при использовании этой функции с константными строками (например, строковыми литералами), так как попытка изменить их приведет к ошибке компиляции.

Потокобезопасность: Функция strtok() не является потокобезопасной (не безопасной для использования в многопоточных приложениях) из-за внутреннего статического указателя на текущее место в строке. В многопоточных приложениях рекомендуется использовать strtok_r() (для POSIX-совместимых систем) или strtok_s() (в стандарте C11), которые являются потокобезопасными версиями функции.

Внимание к разделителям: Убедитесь, что вы корректно выбираете разделители в соответствии с вашими потребностями. В примере выше мы использовали запятые и пробелы, но вы можете использовать любые символы или строки в качестве разделителей.

Внимание к пустым подстрокам: Если входная строка содержит подстроки, которые начинаются или заканчиваются разделителями, strtok() вернет пустые строки в качестве подстрок. Вы можете добавить дополнительную проверку, чтобы игнорировать эти пустые подстроки.

Удаление пробелов из начала и конца строки

Использование std::isspace()

Функция std::isspace() является частью библиотеки <cctype> в C++.

Она определяет, является ли переданный ей символ пробельным символом, таким как пробел, табуляция, новая строка и так далее.

Она возвращает ненулевое значение (true) в случае, если символ является пробельным, и 0 (false) в противном случае.

Удаление пробелов из начала и конца строки

Пример использования std::isspace() для определения пробельных символов

```
#include <iostream> // Подключаем заголовочный файл iostream для работы с вводом и выводом
#include <cctype>   // Подключаем заголовочный файл cctype для работы с функцией isspace
int main() { // Объявляем функцию main, которая является входной точкой программы
    char space = ' ';  // Создаем переменную char с именем space и присваиваем ей значение пробела
    char tab = '\t';   // Создаем переменную char с именем tab и присваиваем ей значение символа табуляции
    char newline = '\n'; // Создаем переменную char с именем newline и присваиваем ей значение символа новой строки

    if (std::isspace(space)) { // Проверяем, является ли символ space пробельным символом, используя функцию isspace
        std::cout << "Пробел - пробельный символ" << std::endl; // Если символ space - пробельный, выводим сообщение о том, что это пробельный символ
    }
    if (std::isspace(tab)) { // Проверяем, является ли символ tab символом табуляции, используя функцию isspace
        std::cout << "Табуляция - пробельный символ" << std::endl; // Если символ tab - пробельный, выводим сообщение о том, что это пробельный символ табуляции
    }
    if (std::isspace(newline)) { // Проверяем, является ли символ newline символом новой строки, используя функцию isspace
        std::cout << "Новая строка - пробельный символ" << std::endl; // Если символ newline - пробельный, выводим сообщение о том, что это пробельный символ новой строки
    }
    return 0; // Возвращаем 0, указывая на успешное завершение программы
}
```

Обрезка строки

```
#include <iostream>  // Подключаем заголовочный файл iostream для работы с вводом и выводом в стандартный поток
#include <cctype>    // Подключаем заголовочный файл cctype для работы с функцией isspace
#include <cstring>   // Подключаем заголовочный файл cstring для работы с функцией strlen

int main() { // Объявляем функцию main, которая является входной точкой программы
    char input[] = "  Текст с пробелами в начале и конце строки  "; // Объявляем символьный массив и инициализируем его строкой с пробелами в начале и конце
    int length = std::strlen(input); // Вычисляем длину строки, используя функцию strlen из библиотеки cstring
    // Удаляем пробелы из начала
    int start = 0; // Объявляем переменную start и инициализируем ее нулем, которая будет указывать на начало строки
    while (std::isspace(input[start]) && start < length) { // Пока символы в начале строки являются пробельными и start меньше длины строки
        ++start; // Увеличиваем start, чтобы перейти к следующему символу
    }
    // Удаляем пробелы из конца
    int end = length - 1; // Объявляем переменную end и инициализируем ее значением длины строки минус один, которая будет указывать на конец строки
    while (std::isspace(input[end]) && end >= 0) { // Пока символы в конце строки являются пробельными и end больше или равно нулю
        --end; // Уменьшаем end, чтобы перейти к предыдущему символу
    }
    // Обрезаем строку, устанавливая нулевой символ в правильной позиции
    input[end + 1] = '\0'; // Устанавливаем нулевой символ в позицию end + 1, обрезая строку после последнего непробельного символа
    std::cout << "Строка после удаления пробелов: \"" << input + start << "\"" << std::endl; // Выводим результат в стандартный поток вывода
    return 0; // Возвращаем 0, указывая на успешное завершение программы
}
```

мы используем std::isspace() для определения пробельных символов в начале и конце строки. После нахождения первого непробельного символа в начале и последнего непробельного символа в конце, мы устанавливаем нулевой символ ('\0') в правильной позиции, чтобы обрезать строку.

Обрезка строки

```
#include <iostream> // Подключаем заголовочный файл iostream для работы с вводом и выводом
#include <cctype>   // Подключаем заголовочный файл cctype для работы с функцией isspace
#include <cstring>  // Подключаем заголовочный файл cstring для работы с функцией strlen

void trimString(char* str) { // Объявляем функцию trimString, которая принимает указатель на символьный массив в качестве аргумента
    int length = std::strlen(str); // Вычисляем длину строки

    int start = 0; // Объявляем переменную start и инициализируем ее нулем, которая будет указывать на начало строки
    while (std::isspace(str[start]) && start < length) { // Пока символы в начале строки являются пробельными и start меньше длины строки
        ++start; // Увеличиваем start, чтобы перейти к следующему символу
    }
    int end = length - 1; // Объявляем переменную end и инициализируем ее значением длины строки минус один, которая будет указывать на конец строки
    while (std::isspace(str[end]) && end >= start) { // Пока символы в конце строки являются пробельными и end больше или равно start
        --end; // Уменьшаем end, чтобы перейти к предыдущему символу
    }
    str[end + 1] = '\0'; // Устанавливаем нулевой символ в позицию end + 1, обрезая строку после последнего непробельного символа
}
int main() { // Объявляем функцию main, которая является входной точкой программы
    char input[] = "  Текст с пробелами в начале и конце строки  "; // Объявляем символьный массив и инициализируем его строкой с пробелами в начале и
конце

    trimString(input); // Вызываем функцию trimString, чтобы обрезать строку

    std::cout << "Строка после удаления пробелов: \"" << input << "\"" << std::endl; // Выводим результат в стандартный поток вывода

    return 0; // Возвращаем 0, указывая на успешное завершение программы
}
```

Обрезка строки

```
#include <iostream> // Подключаем заголовочный файл iostream для работы с вводом и выводом данных
#include <cctype>   // Подключаем заголовочный файл cctype для работы с функцией isspace()

void trimString(char* str) { // Объявляем функцию trimString, которая принимает указатель на символьный массив в качестве аргумента
    char space = ' ';       // Создаем переменную char с именем space и присваиваем ей значение пробела (' ')
    char tab = '\t';        // Создаем переменную char с именем tab и присваиваем ей значение символа табуляции ('\t')
    char newline = '\n';    // Создаем переменную char с именем newline и присваиваем ей значение символа новой строки ('\n')

    int length = std::strlen(str); // Вычисляем длину входной строки с использованием функции std::strlen()

    int start = 0; // Объявляем переменную start и инициализируем ее нулем, которая будет указывать на начало строки
    // Пока символы в начале строки являются пробелами, символами табуляции или символами новой строки, и позиция start меньше длины строки
    while ((str[start] == space || str[start] == tab || str[start] == newline) && start < length) {
        ++start; // Увеличиваем позицию start, чтобы перейти к следующему символу
    }

    int end = length - 1; // Объявляем переменную end и инициализируем ее значением длины строки минус один, указывая на последний символ строки
    // Пока символы в конце строки являются пробелами, символами табуляции или символами новой строки, и позиция end больше или равна позиции start
    while ((str[end] == space || str[end] == tab || str[end] == newline) && end >= start) {
        --end; // Уменьшаем позицию end, чтобы перейти к предыдущему символу
    }

    str[end + 1] = '\0'; // Устанавливаем нулевой символ ('\0') в позицию после последнего непробельного символа, обрезая строку

    // В результате работы функции строка str теперь содержит обрезанную версию исходной строки, без пробелов в начале и конце
}

int main() { // Объявляем функцию main, которая является входной точкой программы
    char input[] = "  Текст с пробелами в начале и конце строки  "; // Инициализируем символьный массив input строкой с пробелами в начале и конце
    trimString(input); // Вызываем функцию trimString для обрезки строки

    std::cout << "Строка после удаления пробелов: \"" << input << "\"" << std::endl; // Выводим результат в стандартный вывод

    return 0; // Возвращаем 0, указывая на успешное завершение программы
}
```

Задача 1: Подсчет количества символов в строке

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

int main() {
    char str[100]; // Объявление символьного массива для хранения введенной строки, размерностью 100 символов.
    std::cout << "Введите строку: "; // Вывод на экран запроса на ввод строки.
    std::cin.getline(str, 100); // Считывание строки с клавиатуры и сохранение ее в массив str. Максимальная длина вводимой строки - 100 символов.

    int length = strlen(str); // Вычисление длины введенной строки с использованием функции strlen() из библиотеки cstring.
    std::cout << "Количество символов в строке: " << length << std::endl; // Вывод на экран количества символов в введенной строке.

    return 0; // Возврат нуля, что обозначает успешное завершение программы.
}
```

Задача 2: Переворот строки

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

int main() {
    char str[100]; // Объявление символьного массива для хранения введенной строки, размерностью 100 символов.
    std::cout << "Введите строку: "; // Вывод на экран запроса на ввод строки.
    std::cin.getline(str, 100); // Считывание строки с клавиатуры и сохранение ее в массив str. Максимальная длина вводимой строки - 100 символов.

    int length = strlen(str); // Вычисление длины введенной строки с использованием функции strlen() из библиотеки cstring.

    for (int i = 0; i < length / 2; ++i) { // Цикл, который идет до середины строки.
        std::swap(str[i], str[length - i - 1]); // Обмен символов: символ в начале строки меняется с символом в конце строки, затем второй слева с вторым справа и так далее, пока не достигнута середина строки.
    }

    std::cout << "Перевернутая строка: " << str << std::endl; // Вывод на экран перевернутой строки.

    return 0; // Возврат нуля, что обозначает успешное завершение программы.
}
```

Задача 3: Объединение строки

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

int main() {
    char str1[100], str2[100]; // Объявление двух символьных массивов для хранения введенных строк, каждый размерностью 100 символов.
    std::cout << "Введите первую строку: "; // Вывод на экран запроса на ввод первой строки.
    std::cin.getline(str1, 100); // Считывание первой строки с клавиатуры и сохранение ее в массив str1. Максимальная длина вводимой строки - 100 символов.
    std::cout << "Введите вторую строку: "; // Вывод на экран запроса на ввод второй строки.
    std::cin.getline(str2, 100); // Считывание второй строки с клавиатуры и сохранение ее в массив str2. Максимальная длина вводимой строки - 100 символов.

    strcat(str1, str2); // Конкатенация (объединение) строк str1 и str2 с использованием функции strcat() из библиотеки cstring. Результат сохраняется в str1.

    std::cout << "Результат конкатенации: " << str1 << std::endl; // Вывод на экран результата конкатенации строк.

    return 0; // Возврат нуля, что обозначает успешное завершение программы.
}
```

Задача 4: Проверка на палиндром (строка, которая читается одинаково слева направо и справа налево)

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

// Функция для определения, является ли строка палиндромом
bool isPalindrome(const char* str) {
    int length = strlen(str); // Вычисляем длину строки с использованием функции strlen() из библиотеки cstring.
    for (int i = 0; i < length / 2; ++i) { // Цикл, который идет до середины строки.
        // Проверяем, совпадают ли символы с обоих концов строки, игнорируя регистр с использованием tolower().
        if (tolower(str[i]) != tolower(str[length - i - 1])) {
            return false; // Если символы не совпадают, строка не является палиндромом, возвращаем false.
        }
    }
    return true; // Если все пары символов совпали, строка является палиндромом, возвращаем true.
}
int main() {
    char str[100]; // Объявление символьного массива для хранения введенной строки, размерностью 100 символов.
    std::cout << "Введите строку: "; // Вывод на экран запроса на ввод строки.
    std::cin.getline(str, 100); // Считывание строки с клавиатуры и сохранение ее в массив str. Максимальная длина вводимой строки - 100 символов.

    if (isPalindrome(str)) { // Вызов функции isPalindrome() для проверки, является ли введенная строка палиндромом.
        std::cout << "Это палиндром." << std::endl; // Если строка - палиндром, выводим соответствующее сообщение.
    } else {
        std::cout << "Это не палиндром." << std::endl; // Если строка не палиндром, выводим соответствующее сообщение.
    }
    return 0; // Возврат нуля, что обозначает успешное завершение программы.
}
```

Задачи

Задача 1: Поиск самого длинного слова
Напишите программу, которая считывает строку с клавиатуры и находит самое длинное слово в этой строке. Выведите это слово.

Задача 2: Удаление символа
Напишите программу, которая считывает строку и символ с клавиатуры. Затем удалите все вхождения этого символа из строки и выведите результат.

Задача 3: Замена подстроки
Напишите программу, которая считывает строку и подстроку с клавиатуры. Затем замените все вхождения подстроки в строке на другую заданную строку и выведите результат.

Задача 4: Проверка анаграмм
Напишите программу, которая считывает две строки с клавиатуры и определяет, являются ли они анаграммами (имеют одинаковые символы в разном порядке). Выведите "Да", если строки - анаграммы, и "Нет" в противном случае.

Задача 5: Выделение слов из строки
Напишите программу, которая считывает строку с клавиатуры и разбивает ее на слова. Выведите каждое слово отдельно на новой строке.

Спасибо!

Панилов Павел Алексеевич